Interactive Evolutionary Computation in design applications for virtual worlds

Jan Kruse

A thesis submitted to

Auckland University of Technology

in fulfilment of the requirements for the degree

of

Master of Philosophy

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

[illegible]

____

Acknowledgements

First and foremost, I wish to thank my wife, Jeannine. You are an amazing person on all levels, and without your support and encouragement, this thesis would not exist.

I wish to thank my supervisors Dr. Andy Connor and Dr. Charles Walker for their invaluable advice, support and endless patience. Thank you for stopping me to try to do too much, and thank you for pushing me further, helping me to achieve more and get a deeper understanding of this fascinating topic. Thank you for helping to keep my work in balance.

I also wish to thank my family and friends for their kind words of motivation, for their support and help to make this a fantastic learning experience.

Further, I wish to thank Colab for providing such an open, flexible and modern teaching and learning environment. Students and Staff are a great pleasure to work with and I consider myself lucky to be part of it.

Finally, a big thank you to all my friends and colleagues at Weta Digital. You have always been a great source of inspiration and I am thankful for all the fantastic discussions we still have, and that I had the continuous opportunity to work with you.

Thank you all for your help and for making this great journey possible.

Table of Contents

List of Figures

Abstract

Modern films, games and virtual reality are highly dependent on convincing computer graphics. Models of high complexity are a requirement for the successful delivery of many animated scenes and environments. While workflows such as rendering, compositing and animation have been streamlined to accommodate increasing demands, modelling of complex models is still a laborious and costly task.

This research introduces the computational benefits of Interactive Genetic Algorithms to computer graphics modelling while compensating the negative effects of user fatigue, a commonly found issue with Interactive Evolutionary Computation. A multi-agent system is used to integrate Genetic Algorithms with computational agents and human designers. This workflow accelerates the layout and distribution of basic elements to form highly complex models. It captures the designer's intent through interaction, and encourages playful discovery.

A modelling pipeline integrating commercially available tools with Human-based Genetic Algorithms is implemented, and a Renderman Interface Bytestream (RIB) archive output is realized to provide easy adaptability for research and industry applications.

Comparisons between Interactive Genetic Algorithms and Human-based Genetic Algorithms applied to procedural modelling of computer graphics cities indicate that an agent-based evolutionary approach outperforms a purely human-centric solution: More iterations are possible in less time, which ultimately leads to better results and a superior user experience. Based on initial testing, a range of suggestions for future investigation are given.

1 Introduction

Convincing Computer Graphics models are a necessity for the creation of successful games, movies and virtual reality environments. Designing high-quality content is a laborious and costly task that requires substantial skill, time and resources (Okun & Zwerman, 2010; Shiffman, 2012). Often, many iterations are necessary to achieve the desired results. Some natural and architectural objects of higher complexity intensify this problem as they necessitate fine detail and a large number of smaller elements, which are the building blocks of the whole. This research addresses these issues by use of human-centric evolutionary computation in conjunction with autonomous agents in order to determine whether this process can be facilitated by semi-autonomous approaches. Interactive Genetic Algorithms, driven by user input, plus computational software agents that support the user in the decision making process, are the core of the solution presented in this thesis. By shifting the workload from the human user into computational areas, the laborious tasks of modelling are simplified and the process is partially automated. This study is concerned with two main ideas: First, it applies a hybrid intelligent system to interactive design and second, it seeks to explore the procedural generation of cities. Procedural city design is used as an example to demonstrate the process, identify potential benefits, and find possible issues of hybrid intelligent systems in design contexts. This research aims to identify whether the addition of two combined artificial intelligence methods, Interactive Evolutionary Computation and Autonomous Agents are beneficial to the human designer in design practice. In order to be able to find a solution to this question, the problem definition is being broken down into the following smaller pieces. First, Evolutionary Algorithms in general are discussed, and how to integrate the human user into the process to make Evolutionary Computation interactive. Further, a definition of Autonomous Agents in this specific context is provided. The agent architecture is being discussed and a software system to study the hybrid intelligent system is introduced. Finally, Generative Design in context of procedural city design as an example is being demonstrated, which serves as one of the two case study for this thesis.

1.1 Definitions

CG	Computer Graphics, the field of media creation by use of computer software including images, sound or animation
CGI	Computer Generated Images, specifically referring to images created by use of computer software
DCC	Digital Content Creation, field of producing digital media content such as images, sound or animation by use of computer software
EC	Evolutionary Computation, the field of computer science using biologically inspired search and optimization techniques
IEC	Interactive Evolutionary Computation, biologically inspired search and optimization techniques with a human user typically replacing a mathematical fitness function
GA	Genetic Algorithm, probably the most common Evolutionary Computation algorithm
IGA	Interactive Genetic Algorithm, Genetic Algorithm with a human user typically replacing a mathematical fitness function
HBGA	Human-based Genetic Algorithm, a multi-agent system replacing recombination and selection with human and/or computational agents
RIB	Renderman Interface Bytestream, file format used to describe scene data for Renderman compliant render engines
UI	User Interface, sometimes GUI (Graphical User Interface)

1.2 Background

The motivation for this research originated from personal experience of the author. Having spent a significant number of years in the visual effects industry, amongst all existing workflows, modelling always stood out as a very slow, laborious task. Especially large and highly complex models, for example the beautiful, highly detailed, digital version of New York in the 1920s for Peter Jackson’s “King Kong” took months to finish and require large teams to do the mostly manual work. As a consequence, complex models are also a substantial figure in every visual effects budget (Okun & Zwerman, 2010).

In order to reduce the amount of time and labour necessary, this study seeks to use computational solutions such as Evolutionary Computation and Agents for some of the tasks involved in modelling large structures.

1.3 Research Questions

This study is driven by the main research question ***“To what extent can Interactive Evolutionary Algorithms combined with autonomous agents support designers in complex form finding tasks?”*** In order to address the various parts of this multifaceted problem, it is broken down into several sub-questions:

- What are the issues that designers face in complex form finding tasks?
- How does the complexity of the task influence the selection of design tools?
- Which is a suitable choice of algorithm, selected from variety of evolutionary algorithms, to answer the main research question?
- How does the addition of an autonomous agent benefit the form finding process?
- How can the proposed combination of Interactive Evolutionary Algorithm and agents reduce the skills requirements for the designer, yet enable the designer to produce very complex structures?

1.4 Significance of Research

Design and creation of Computer Generated models is an important foundation for several different fields such as Games, Film, Visualization and Virtual Reality, to name a few. Convincing and seemingly complex models are necessary to create an immersive experience for the targeted audience across all fields (Okun & Zwerman, 2010). With computer hardware, digital cinemas and UHD television set, the mark for detailed models has been set even higher and content creators are seeking to improve the quality of their products without having to invest more money in today’s high risk environment. Artists and designers who are supplying the models have to adapt to very complex Digital Content Creation software (DCC), which often utilizes hundreds or thousands of parameters to drive the designs of the models. Naturally, modern DCC software has a very steep learning curve (Kostic, Radakovic, Cvetkovic, Trajkovic, & Jevremovic, 2012). The increasing complexity of the DCC software combined with the inversely proportional decrease in available budgets for the created content, leads to a predicament, which this research seeks to address by using computational intelligence rather than manual labour. The conflict between rising demands and shrinking budgets underlines the importance of this body of work and sets the foundation for this study.

Some research into design systems that reduce manual work and utilize computing power instead has been done prior to this study (Renner & Ekárt). Those systems are mostly driven by the computer and not the designer. Resulting models are evidence of the capability of the software programmer *creating* the software, rather than the designer *using* the software. This study seeks to *include* the designer rather than running an exclusive automated process lacking the designer's intent. Capturing the vision and using the designers aesthetic preference to drive the final design is an important element (Anderson, Buchsbaum, Potter, & Bonabeau, 2008) which based on the outcomes of a thorough literature review appears to have not yet been applied to complex three dimensional modelling based on computational methods. This too emphasizes the importance of this study.

Finally, this research adds another important element to existing knowledge by accounting for the main implication found in interactive design systems – user fatigue (Kamalian, Yeh, Zhang, Agogino, & Takagi, 2006). Numerous iterations, slow render times that create potentially unproductive periods and the before-mentioned complexity of DCC software contribute largely to the user's incrementing lack of concentration. Todd and Werner (Todd & Werner, 1999) formed the term "fitness bottleneck" in this context. This study addresses the issue by adding an autonomous agent to the software system, which supports the user and reduces the number of required interactive iterations.

1.5 Summary of Main Contributions

This thesis presents two main contributions to existing knowledge. First, an original method to integrate asset creation using Evolutionary Computation and RenderMan compliant outputs is discussed and demonstrated. This enables future researchers as well as pipeline programmers in the creative industries to utilize advanced Artificial Intelligence algorithms in conjunction with standard commercial render engines.

The second main contribution is the integration of an Interactive Genetic Algorithm with an Autonomous Agent system to support the human user in the process of creating models and lessen the effects of fatigue while increasing the number of possible iterations to refine the final product. Although the development of the workflow and the software system consumed most of the time of this research, initial testing of the academic prototype showed promising results and the foundation for further research in this field has been laid.

1.6 Structure

This Thesis is structured into four main parts, namely the Literature Review, Methodology and Research Design, Results and Observations, and finally Discussion and Conclusions.

The Literature Review unpacks the three main research components of this study: Generative Design, Evolutionary Computation and Agents. The section about Generative Design gives a short insight and definition of what this term in context of the vast field of Design encapsulates and how Generative Design is situated within the wider area of Computer Aided Design. The second component discusses the current knowledge about Intelligence, Machine Intelligence and more specifically Evolutionary Computation. The struggle to grasp what Intelligence in general means, as well as the ongoing discussion about machines and software and their capacity to exert seemingly intelligent behaviour is discussed. Finally, the field of Agents and Agent Systems as part of the third and probably most innovative component of this research is looked at.

Methodology and Research Design discusses practice-led, design science, action and empirical research as they are part of the overall Methodology being used in this study. The experimental process as well as the software systems used to measure performance of Interactive Evolutionary Computation with Agents is being introduced.

In the third main section of this thesis, results and observations are presented and discussed. The differences between Interactive Evolutionary Computation with and without Agents are highlighted in order to address the research questions. Further, the successful integration of Evolutionary Computation and standard render engines used by the visual effects industry is demonstrated.

Finally, the fourth section of this study, discusses the outcomes and presents the conclusions that can be drawn from this research. This section also discusses limitations of the current work and provides an outlook into possible future research.

2 Literature Review

2.1 Introduction

The problem discussed in this study is a multi-layered issue, and to answer the main research question, an investigation into two main fields is necessary: Design and Artificial Intelligence. Both are very broad fields of research and to narrow these further, and to address only the core matters of this enquiry, Generative Design (as part of Design), as well as Genetic Algorithms and Agents (both sub-fields of Artificial Intelligence) are discussed in more detail in the following literature review. These three main components address and inform the research questions, and specify the field of study in this thesis. This aids to identify the gap in existing knowledge, which this thesis attempts to fill.

It is important to note, that a deeper look into Genetic Algorithms and Agents will be taken, and only briefly at Interactive Evolutionary Computation in general, in order to understand the prerequisites of Human-based Genetic Algorithms, which are the centre of the Artificial Intelligence section of this research.

2.2 Generative Design

Generative Design, often also referred to as Procedural Design, is the area of form and shape finding by use of algorithmic help (Bohnacker, 2012). It is the overarching field that form finding is located in. There are some significant differences between manual design aided by computer software and automated design provided by computer software. This section of the literature review will help to understand where this research is placed.

While Computer Aided Design (CAD) systems and other tools to create three dimensional objects with help of computers are technically based on algorithms and program code in general, they require the designer to manually operate the software and provide the inputs necessary to create any shape and form. Some of the processes may be partially automated, but the designer still needs to draw the objects on screen. These objects are often primitives such as circles, rectangles, cubes, spheres and other two or three dimensional shapes, which when combined, form the desired complex object (Greenberg, 2007). While this manual process lays the foundation for Generative Design, it is not part of this study, because the goal is to investigate automated or semi-automated design processes, driven by algorithms.

Generative Design is the process of writing or applying often simple and small fragments of code that show objects on screen automatically, without the necessity to have the designer create underlying shapes in the first place (Greenberg, Xu, & Kumar, 2013). The creation of shapes is done by software, driven by an algorithm. The combination of many simple shapes, creates larger compound structures. These two or three dimensional structures (or objects) tend to be rather complex, given the simplicity of a few lines of code (Shiffman, 2012). For example, to create a complex procedural structure made of hundreds or thousands geometric primitives such as cubes and spheres, only 8 lines of code are required. These structures also often use recursive elements, i.e. functions or procedures that call themselves over and over again and therefore assemble a complex object from small, identical building blocks, much like some plants such as ferns or trees are complex structures that are made from very few, very simple individual elements (Reas & McWilliams, 2010). In case of the fern for example, the fractal or recursive nature is visible from a large scale to a microscopic level: The same shape is found over and over again, from plant to branch, and from branch to leaf, and so forth. This is usually referred to as self-similarity (Mandelbrot, 1983; Shiffman, 2012). In case of a procedurally modelled city, it is possible to apply some very similar techniques, while not fully self-similar, they are still based on repetition of simple building blocks, which when used in large numbers, resemble rather complex structures. An example would be windows on an office building, which are simple elements, but make up most of a large structure. Another example are streets. While most streets have similar, simple building blocks, as a whole they present a very complex and large system of gaps between buildings.

Further, there is little evidence in the literature that a computer generated city has been made using Generative Design driven by the user in conjunction with algorithmic help. While there have been a few attempts to create procedural three dimensional cities as laid out by Parish and Müller (2001), these were entirely computer driven and provided no user interaction. This leads to a computer generated city as such, but does not reflect the designer’s intent at all. The city is a result of the programmer’s imagination and it can therefore be argued that it is similar to manual Computer Aided Design, with the difference that the user (or programmer) does not draw objects on screen, but writes code to create these objects. This study follows a different approach in that the user is enabled to influence the design and layout by choosing preferred layouts. Parish and Müller allow the user simply to run the software, which then produces a random result, which may or may not resemble the designer’s vision. There have also been other studies into components of generatively designed cities, for instance street structures or building structures (Lechner et al., 2004), but these researchers did not consider interactive user input at all. Therefore, this study aims to fill a significant gap in the existing literature, by using Generative Design driven by user interaction to create complex structures while seeking to reach the designer’s original vision.

Using computers to explore the space of possible images, sculptures or other complex artistic forms such as musical compositions, has enabled researchers and artists to evolve pieces of art and led to the exploration of new ideas, from simple arbitrary color blobs to working functional forms such as boat designs, architectural forms or electronic circuits. Designers are being enabled to study more solutions in less time and to find forms that are outside the convention and expand their conceptual understanding. Evolutionary approaches have also led to new methods and principles, which can be exploited in future designs (Bentley, 2002).

Bohnacker (2012) demonstrates a variety of typographic and abstract graphics in his book about Generative Design. Other examples include Generative Art using L-Systems (Pearson, 2011), Generative Design using very simple autonomous agents (Shiffman, 2012) and studies in Architecture (Reas & McWilliams, 2010), to name a few.

Generative Design has become more common for a variety of reasons, the most significant probably being the abundance of computing power and also the arrival of suitable frameworks such as Processing (Greenberg et al., 2013) and Openframeworks (Perevalov, 2013). This study employs Generative Design for Infographics and Procedural Cities, by use of Processing (Greenberg et al., 2013) for the UI and 3Delight (“Welcome | 3delight,” 2014), a RenderMan compliant engine to create the actual CG images.

2.3 Artificial Intelligence

In this section of the literature review a brief overview of the current understanding and discussion about intelligence and in particular artificial intelligence is given. Further, knowledge representation and finally problem solving is discussed. This will set the context for the Artificial Intelligence algorithms and paradigms explored in this study. While Artificial intelligence is understood to be located in the area of Intelligence in general, this research is concerned with Evolutionary Computation and Agents in particular as outlined in Figure 1.

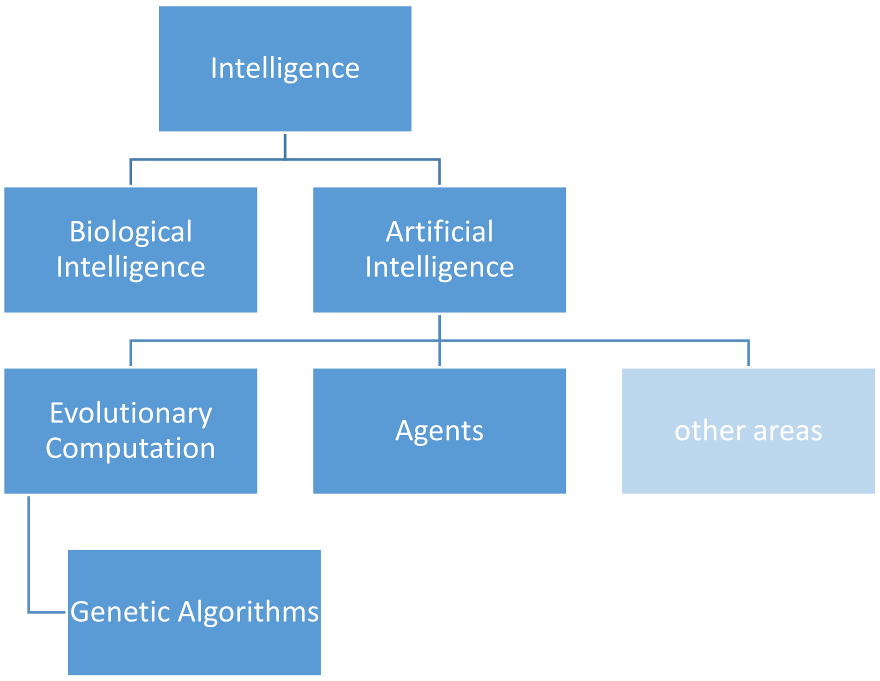


Figure 1 - Positioning Artificial Intelligence in the overall context (only relevant areas shown)

2.3.1 Definition of Artificial Intelligence

In his book about Evolutionary Computation, David Fogel (2006) suggests that the term artificial intelligence itself, might be misleading. In Fogel's view, intelligence just needs to be represented properly in form of a defined process to lose its artificial connotation. He claims that methods for generating intelligence need to be understood and applied in the same way, whether living beings, systems or machines are concerned, as these processes then become fundamentally similar and follow the same physics (Fogel, 2006). In this, he goes contrary to the majority of research in artificial intelligence, which has simply attempted to simulate intelligent behaviour as being shown by humans. Fogel proposes a definition independent of human behaviour, stating that intelligence is "the capability of a system to adapt its behaviour to meet its goals in a range of environments" (Fogel, 2006). This seems to be a very narrow definition compared to what other researchers such as Russell and Norvig (1994) break down into two main issues. Systems that imitate the thinking process of humans, and systems that attempt to model human behaviour, and respectively act like humans. The first could be represented by the *research* concerned with thinking or behaviour of humans. And the second, is reflected by the goal to imitate human behaviour, but idealize the behaviour first, to apply this ideal standard subsequently. The approach to imitate thinking is anchored in the interdisciplinary field of cognitive science, which combines AI computer models and experimental methods from psychology as well as neuroscience in an attempt to theorize how the human mind works, how humans think (Russell & Norvig, 1994).

A popular example for the modelling of human behaviour is the Turing Test (Turing, 1950). The British mathematician Alan Turing proposed to use a simple test instead of lists of definitions that reflect intelligence. Using separate rooms, a man and a woman communicate to an interrogator only through a computer. The goal is to deceive the interrogator into believing that the man is a woman and vice versa. Then, in a second phase, the man is replaced by a computer. The goal stays the same, the computer is tasked to convince the interrogator that it is a woman, whereas the female participant tries to imitate the behaviour of a man. The computer would be able to make mistakes and provide unclear responses, just as a human might do. Turing assumed, that if the computer is able to deceive the human interrogator the same way the man did, it would have passed the intelligent behaviour test (Turing, 1950). Turing predicted in 1950, that a computer fifty years later would be able to stand the test successfully. While his prediction was not entirely accurate, contemporary research into cognitive science has achieved remarkable results.

One of the most prominent examples for an approach based on cognitive science is IBM Watson, a supercomputer that imitates several different methods in the human thinking process in order to compete in Jeopardy games against human opponents (Ferrucci, Levas, Bagchi, Gondek, & Mueller, 2013). For instance, the main contribution to Watson's successful strategies to understand the questions (or in case of Jeopardy, the answers) is the imitation of linguistic features such as narratives analysis derived from natural languages research (Lally et al., 2012). Using an expert system that derives the questions to the answers given in Jeopardy from a large database, would take too long and require too much storage by todays hardware and software standards. Instead, Watson imitates different processes of human behaviour to learn, understand and combine knowledge to be successful in the game (Prager, Brown, & Chu-Carroll, 2012).

In this study, artificial intelligence is understood to be an imitation of human behaviour, as this research is concerned with systems that expose seemingly intelligent properties, and not with the cognitive process itself. Using methods that imitate human behavior leaves room to include systems that are classified as artificial intelligent approaches like optimization, search and knowledge engineering, contrary to Fogel's rather narrow definition (Fogel, 2006). These additional systems are not necessarily adaptive, but rather statistical, and expose a behaviour similar to what is considered intelligence and therefore could be included in the broad field of intelligence and particularly artificial intelligence (Negnevitsky, 2004). This study makes use of both, adaptive as well as statistical optimization and search methods, as discussed in sections 2.4 and 2.5.

Also noteworthy is the divide throughout the literature between hard AI and soft AI, basically referring to the difference of an AI system *acting* like a human, therefore mimicking human behaviour (hard AI), and using any technique that works to solve a particular problem in an seemingly intelligent way (soft AI).This topic is discussed in depth in context of autonomous agents in section 2.5.

2.3.2 Artificial Intelligence Systems

Buckland (2004) suggests that there a distinction has to be made between "Academic AI vs. Game AI" (Buckland, 2004) as Academic AI is concerned with an optimal solution to a problem, whereas Game AI is more concerned with the limitation of resources in order to keep the program using the Artificial Intelligence component interactive. While this is true in some cases, it is arguable that there are two implications with this viewpoint.

First, there are many researchers in the field of Computer Science engaged in the topic of optimization of algorithms, which require the program to run as efficiently as possible, not just optimally: Algorithmic efficiency seeks to achieve results not just by optimization at any cost, but achieving optimization with the least resources possible. For example, many problems in applied computation and engineering need to be solved in a time critical manner and the systems incorporate a balanced approach between speed and optimization. An example is the Genetic Algorithm, which will be discussed in a later section (2.4.1) of this thesis. The entire purpose of Genetic Algorithms is to solve an otherwise impossible (due to time constraints) problem in a reasonable timeframe. If that was not the intention, a brute force algorithm would certainly guarantee an optimal solution, given enough time. Second, in computer games and other applications, the issue of balance could lead to less engaging user experiences, if the resulting solution seems to be too ‘stupid’ to provide an exciting engagement. If the user loses interest or fatigues due to lack of quality, this would be similarly problematic as the lack of responsiveness. Academic research is concerned with finding different solutions in such cases, for example hybrid intelligent systems, which could offer a balance between speed and optimization (Negnevitsky, 2004). This is particularly relevant for this study, as it employs a hybrid intelligent system made of Genetic Algorithms, which are an Evolutionary Computation approach, in combination with human and computational agents. The details are shown throughout the following sections.

2.4 Evolutionary Computation

Evolutionary Computation is the field of research that is concerned with computation based on the concepts of natural evolution. Figure 2 gives an overview of Evolutionary

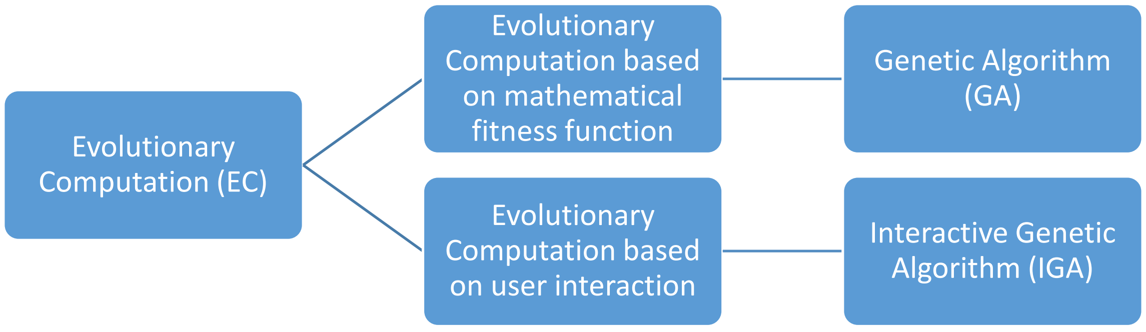

Evolutionary Computation (EC)
Evolutionary Computation based on mathematical fitness function
Evolutionary Computation based on user interaction
Genetic Algorithm (GA)
Interactive Genetic Algorithm (IGA)

Figure 2 - Field of Evolutionary Computation

Computation and its subfields. Specifically, it borrows ideas from Darwin's theory of evolution, which states that individuals as part of a population increase their chances of survival and reproduction by way of natural selection. This selection process allows for small variations of each individuals' properties, which are then passed on to the next generation through inheritance. Darwinism in conjunction with Mendel's concept of genetics formed what is known in biology as modern evolutionary synthesis (Keeton, 1996).

Biological evolution encapsulates the following concepts (Pimpale & Bhande, 2007):

- *DNA* (Deoxyribonucleic acid) is the molecular structure that encodes the genetic information of each cell of all living organisms. It is represented as a double helix.
- *Chromosomes* are strings of DNA.
- *Genotype* is the hereditary information encoded in the DNA.
- *Phenotype* are the observable properties as a result of the DNA.
- *Reproduction* is the creation of offspring by (usually) two parents, inheriting parts from both parents' DNA.
- *Crossover* is the process of synthesizing an offspring DNA, creating a new chromosome.

- *Mutation* is a small accidental change in the offspring DNA, potentially resulting in slight variations to a straight, non-mutated crossover. Mutation happens with a very low probability.
- *Survival of the fittest* is the concept of only the strongest properties of a DNA being sustained over many reproduction cycles. Weaker DNA properties could result in weaker offspring, which in turn has a lower chance of survival. Over many generations, this leads to the elimination of weak DNA. This is also used synonymously with the term *Evolution* in the literature (Pimpale & Bhande, 2007).

Modern evolutionary synthesis serves as the foundation for the many different types of evolutionary computation. While evolutionary computation borrows ideas and the notion of biological evolution from the natural process, evolutionary computation is merely an abstraction of evolutionary synthesis to emulate soft intelligent behaviour in computer software. The concepts were applied in different ways and evolved over time, so that there are now a multitude of different algorithms, which all borrow from the underlying idea of natural evolution. Examples are Genetic Algorithms (Holland, 1975), Evolution Strategies (Beyer & Schwefel, 2002) and Genetic Programming (Koza, 1992). While these all simulate natural evolution to an extent, they differ significantly in how they apply the evolutionary principles. Genetic Algorithms are heuristic search algorithms, used to find a solution in the space of all possible solutions. Evolution Strategies are designed to find solutions to technical optimization problems (Negnevitsky, 2004), and Genetic Programming generates computer programs that in turn attempt to solve the actual problem (Koza et al., 2003). Genetic Programming therefore programs computers by finding an optimal set of rules or section of code.

This study uses Genetic Algorithms, which are detailed in the following section.

2.4.1 Genetic Algorithm

The focus of this research is on Genetic Algorithms for two reasons. First, they are simple, yet powerful search algorithms. This study seeks to find at least one solution that fulfills the user’s intent within the virtually infinite number of possible city structures, shapes, layouts and forms. A search algorithm that is able to get to a result while other solutions such as random generation of cities might never create the vision of the designer, seems to be a reasonable choice, which is also strongly supported by existing research. Renner and Ekart (2003) conclude that "Genetic Algorithms are capable of solving surprisingly complex design problems". Second, they have been successfully applied to much simpler, but somewhat similar design problems as presented in this thesis, for instance finding coloured blobs and stripes that reflect the intent of the designer in the solution space of all possible combinations of colour blobs and stripes (Anderson et al., 2008). This research seeks to extrapolate the positive results to the more complex design issue of Infographics and Procedural City models. Using the existing research into Genetic Algorithms for design problems, seems to be a solid starting point.

Genetic Algorithms (GA), being part of the heuristic optimization or search algorithms, are very popular due to their relative simplicity, and are also well researched and understood (Fogel, 2006). They were introduced in the 1970s by John Holland and mimic natural evolution by abstracting the chromosomes into binary digits (Holland, 1975). These chromosomes are passed on from one population to a new population after genetics-inspired processes of crossover and mutation. An evaluation function called fitness function is then applied to establish each chromosome's performance towards the final goal. If the chromosome performs poorly, it is likely to be dropped from the pool of future 'parents'. Otherwise, if the chromosome's fitness is high, it is more likely to be selected for reproduction. The actual reproduction process is performed by using a crossover operator, which mixes parts of two parent's chromosomes to form the new child's chromosome. Finally, a mutation operator is applied to some of the new found chromosome in order to ensure a certain variation of the child's properties. This mutation operator randomly changes the value of individual digits of the chromosome binary string. Mutation operator and crossover operators effectively represent the probability of each operation (mutation and crossover) to happen. The process of simulating natural evolution is repeatedly applied for many generations and as a result, the fittest members of a population dominate, while the less fit become extinct. The underlying mechanisms of Genetic Algorithms are very simple, yet capable of showing seemingly complex behaviour and the ability to solve difficult problem sets (Negnevitsky, 2004).

While simple search and optimization algorithms such as hill climbing or gradient descent might have a tendency to get stuck in local maxima or minima, Genetic Algorithms avoid this issue due to their inherent creation of diversity by mutation. Genetic Algorithms are highly effective in many cases, and given that a robust fitness function and solid parameters for crossover and mutation have been selected, tend to avoid local optima in favour of a global solution (Fogel, 2002).

To verify the performance of Genetic Algorithms, including the parameters for selection and mutation probabilities, but also the correct implementation of the fitness function, mathematical test functions, which provide consistent conditions for testing, are usually employed. Figure 3 shows a three dimensional plot of the sphere function which is a standard test function for optimization problems in computer science. The sphere function is defined by the equation:

$$f(x) = \sum_{i=1}^{n} x_i^2$$

From the plot it is easy to see that there is just one minimum right in the centre of a

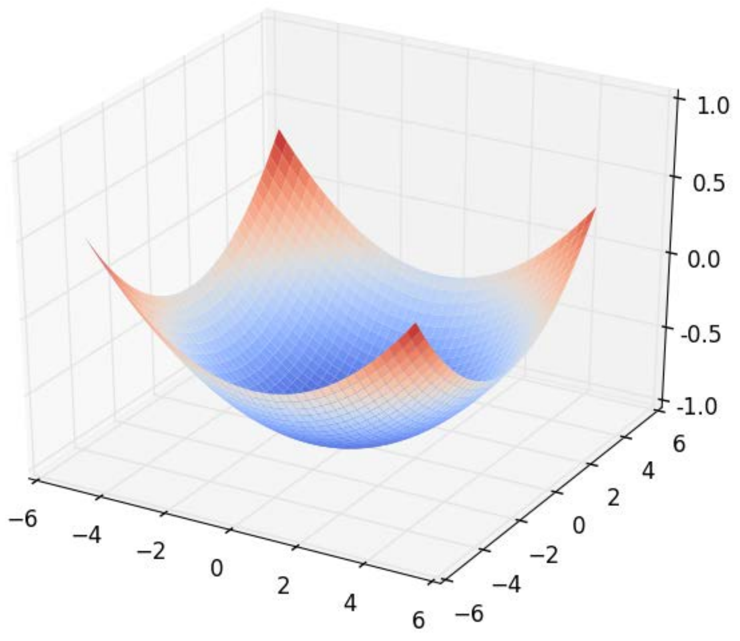


Figure 3 - Sphere function

large ‘valley’. A computer algorithm should be able to find the global minimum quickly and with a large chance of success.

In contrast, Figure 4 plots the Levi function N.13, again a standard test function defined by the equation:

$$f(x,y) = \sin(3\pi x)^2 + (x-1)^2(1+\sin(3\pi y)^2) + (y-1)^2(1+\sin(2\pi y)^2$$

This second function has a multitude of local minima and maxima. While humans can quickly point out the global minimum, a heuristic search algorithm might get stuck in any of the local minima and never find the global optimum.

Many different techniques to avoid this behaviour have been invented, for instance by randomizing part of their search instead of relying entirely on local optimization. The drawback of these potential improvements are longer runtimes and less efficiency (Bentley, 1999). Accordingly, careful selection of operators driving the GA is important and can be a tedious and time consuming task.

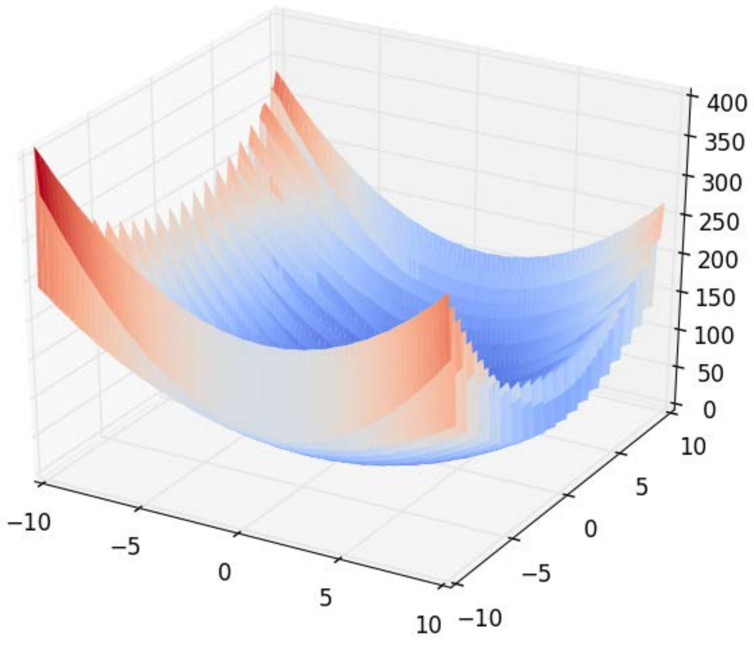


Figure 4 - Levi function N.13

Given that the Genetic Algorithm is the ‘weapon of choice’ in this study, a more detailed discussion of the principles will be given in the following paragraphs, breaking the GA down into smaller steps. This breakdown is loosely based on Melanie Mitchell’s textbook on Genetic Algorithms (M. Mitchell, 1998). For this study, slight variants of this textbook approach have been adopted, as discussed in the relevant chapters 3.2 (Software System) and 3.3 (Experimental Process).

1. Assuming that the problem to be solved has been clearly defined, the problem can be represented as a string of n elements. The length of the string is fix. Each element is a 1-bit digit and denotes the chromosome x of one possible candidate solution.

0	0	1	0	1	1	0	1	1	0

2. Define a population N with a size S as a fixed number of chromosomes x. Also define a crossover operator p_c as well as a mutation probability p_m.
3. Start by randomly generating a population N_1.
4. Then, calculate the fitness $f(x)$ of all elements x in N_1.
5. Selection takes place. Pick a pair of chromosomes x_1 and x_2 based on their fitness $f(x)$: The higher the fitness $f(x)$, the higher the probability of being selected. This ensures that the fittest x have the best chance to reproduce, while the chromosomes with the lowest fitness are unlikely to pass their DNA on to the next

0	0	1	0	1	1	0	1	1	0

x_1

1	0	0	1	1	0	0	1	0	1

x_2

generation (population N_2). It is important to note that selection is done without removing the chromosome from the mating pool, so that parents can be selected multiple times.

6. Create a pair of offspring chromosomes using the crossover probability p_c as follows: Both parents chromosomes are crossed over at a point randomly chosen

0	0	1	0	1	0	0	1	0	1

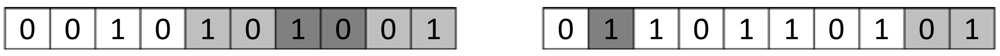
$offspring_1$

0	0	1	0	1	1	0	1	0	1

$offspring_2$

based on a uniform probability. In the special case that no crossover occurs, the parent chromosomes are each copied to the two offspring.

7. Each of the two offspring are mutated with probability p_m i.e. individual bits are



$offspring_1$ $offspring_2$

   switched from 0 to 1 or 1 to 0 accordingly.
8. Both offspring are placed in a new population N_2.
9. Repeat from step 5 until N_2 is the size of S.
10. Start over from step 4 using N_2 to select parent chromosomes for breeding the next population N_3 and so forth, until a termination condition, for instance a certain fitness $f(x)$across the population is reached.

A single iteration is known as a *generation* and the whole process of all generations is termed *run* (M. Mitchell, 1998). By the time of writing about one and a half decades ago, Mitchell states, probably due to the limitations in computing power and memory availability, that a typical run involves "50 to 500 or more generations". With today's desktop computers, let alone workstations or supercomputers, 10,000 to 100,000 or more generations are the norm. That is based on the observation that contemporary problems are often magnitudes more complex and require larger bit encoding. Therefore, the desire to keep runs small is still the same, but the amount of data required is much larger and necessitates more generations to satisfy the termination criterion.

The above mentioned process describes the foundation for most applications of GA. But over time, some variations and even significant changes to GA have been introduced. Typical examples are multi-point crossover, tournament selection, truncation selection or elitism the latter often referred to as replace worst (Lanzi, 2011). Further, many different models for the application of GA have been developed, such as Island Genetic Algorithms (iGA), which utilize several parallel running GA to reduce the tendency of GA of getting stuck in local maxima of complex functions (Skolicki & Jong, 2004).

Genetic Algorithms, which are being employed by this study in an extended form (see section 2.6), have successfully been applied to a range of different areas such as Engineering, Arts and Computer Science. Some examples include the optimization of machinery (Connor, 1996), evolved particle systems (Shiffman, 2012), generative jazz music (Biles, 1994) or optimizing the weights of neural networks (Fogel, 2002). According to Li & Kou (2001), Genetic Algorithms characterize them as being implicitly parallel, robust and scalable, as well as powerful in global search and optimization.

The question, why GA are valued as very powerful tools for search and optimization, finds a simple answer in an example of a small optimization exercise, the knapsack problem. The goal is to pack a number of items of different weights (sometimes the problem is also formulated with sizes) into a backpack. Not all items are going to fit, i.e. there is a limiting overall weight or size, but the intent is to identify the maximum number of items possible. If attempted manually, packing a random number n of items, weighing the backpack until the best possible outcome is reached, would take up to a maximum of 2^n solutions. While this might be practical for a number of 4 items, due to the exponential growth of the problem, a count of 64 items becomes 2^{64} potential solutions, which is impractical to test: Running this problem implemented as brute force (that is by random selection of items) on a fast contemporary workstation computer, testing all 18,446,744,073,709,551,616 candidates is going to take about 27,000 years based on about 200,000 individual fitness calculations per second. The GA in turn solved this problem in just under 10 seconds. While this seems to be an impressive result, it is important to note that the GA might not have found the single best possible solution to the problem, but a very high level of optimization. In general, GAs are not necessarily finding the *best solution* to certain search and optimization problems, but tend to find at least *a solution* relatively quickly, especially with rising complexity and otherwise impossibly large candidate spaces.

It is important to note that a huge range of different techniques with regards to encoding the phenotype (the actual representation of the DNA string) into a suitable genotype (the underlying representation, for example as a bit string) are available. This is also true for the process of crossover and mutation. Amongst other methods, encoding can be done by binary encoding, value encoding or tree encoding (Pimpale & Bhande, 2007). Binary encoding is the simplest form, using a string of binary information to encapsulate the high level phenotype, for example representing a RGB color value as three 8-bit numbers concatenated into a 24-bit string. Value encoding is a high level representation, for example using three float values to represent a RGB color. Here the DNA string is not a 24-bit string that is used for recombination and mutation, but three numbers that can be recombined from two parents into a child's DNA of three float numbers. In value encoding the DNA could be any type of value, including numbers, characters or whole words.

Crossover methods have evolved from simple single point crossover into many different ideas. To name a few, there is two-point crossover, where the DNA is split twice and the resulting three parts are recombined from two parents or uniform crossover, where a probability takes individual genes from both parents with a given probability (M. Mitchell, 1998). Instead of mixing bit strings, randomly taken from the DNA, the gene structure is preserved. For example, if three 8-bit color values were encoded into a 24-bit string, the split would only happen so that the 24-bits are not split randomly, but only every 8-bits. This maintains the consistency of the color values Red, Green and Blue instead of potentially splitting in the middle of the blue value. Therefore, consistency of parameters that are to be preserved through the evoluationary process is easier to manage than in single point or multi-point crossover (Pimpale & Bhande, 2007).

2.4.2 Interactive Evolutionary Computation

Genetic Algorithms (GA) belong to the simpler of Evolutionary Computation algorithms. They are relatively easy to implement and can be very effective if the solution space is very large. While GA have some disadvantages such as the tendency to get stuck in local maxima if not properly tuned (Goldberg, 1989; Skolicki & Jong, 2004) and the requirement of a fitness function, which can be a challenge if soft factors such as aesthetics come into play (Takagi, 2001), GAs have been well studied and well documented.

The GA was selected for this study based on its simplicity, but the specific implementation does not use a mathematical fitness function. Instead it integrates the user (designer/artist) into the process: The user provides the necessary fitness evaluation in the selection stage of the GA. This is known as Interactive Evolutionary Computation (IEC). The underlying idea of IEC was first introduced by Dawkins in the third chapter of his book "The Blind Watchmaker". Dawkins demonstrates the process of evolution based on Darwinian theories in a software program called "biomorphs", which uses human interaction to evaluate factors such as aesthetics, appeal or attractiveness (Dawkins, 1986). Karl Sims (1992) has taken this idea of user-computer interaction for Genetic Algorithms further and suggests that the human user does not have to understand the underlying process of creating the candidates, while still being able to produce results of high complexity. He argues that Interactive Evolution enables the computer as well as the human to achieve results that neither could have produced on their own. The term 'Interactive Evolutionary Computation' was finally formed by Takagi (2001), who also evaluated IEC in context of several different fields of research as a method to integrate computational optimization and human evaluation. Beside the ability to combine both optimization and evaluation for design subjects, IEC can also offer a significant benefit over other computational design methods such as Genetic Algorithms or manual computer aided design: The user can change their evaluation during the evolutionary process and drive the resulting populations into a different direction. This could potentially lead to the discovery of previously unknown outcomes and expose features that were not expected initially. Changing the objectives in regular Genetic Algorithms would require re-coding and is not practical nor efficient. IEC allows for alterations on the fly and as a result has been recognized as a 'novelty generator' (Gu, Xi Tang, & Frazer, 2006).

It is important to note, that the user does not necessarily evaluate the genotype or phenotype of the candidates of the Genetic Algorithm directly, but a different representation which is easier to grasp. For example, instead of a numerical bit string (genotype) or the associated colour values (phenotype) of some elements of an image, a user might select whole images that are based on both elementary parts (genotype/phenotype) (Takagi, 1998). This is an approach chosen for this research as detailed in section 3.2.3.2. The user is able to select rendered images, which are each based on an underlying set of parameters. These parameters are encoded into a collection of numerical strings, the DNA. This DNA is then modified (mutated) and crossed over by the Genetic Algorithm. The resulting parameter set is rendered back as a new image, which is then presented to the user for selection, and so forth.

While this is an elegant way to both capture the user's preference and also avoid forming a mathematical function for aesthetics or user preference, it poses a significant problem, which is related to the nature of humans. After a number of iterations, human users tend to fatigue and get slower to select, get distracted more easily and lose concentration due to the high number of visual triggers (Takagi & Iba, 2005). The effects of time saving while defining a computational solution for the design problem could potentially result in a less effective and less successful overall outcome of the interactive evolutionary process due to this fatigue effect. In the following section (2.5), Agents and Agent Systems are being discussed, as they could serve as a possible solution to this inherent problem of Interactive Evolutionary Computation.

Interactive Evolutionary Computation offers significant benefits over non-interactive approaches, as it removes the necessity to find a mathematical solution for the fitness function. The intractable nature of writing an equation for aesthetics, taste and preference is elegantly avoided. Instead, the user is employed to directly provide the selection of potential candidates, which makes this approach possibly a solution for this research.

2.5 Agents

Research into *agents* or *autonomous agents* is a relatively young field, which has been studied for about the past two decades. Most publications from the early 1990s presented agent definitions that are still valid and that build the foundation of our current understanding of the field. One of the main questions that arise when confronted with agents for the first time is what makes such an agent and how it differs from other software in general. Interestingly, it seems that the recent popularity of agents has led to a wide application and the explosive growth of agent applications. This caused a lot of confusion of what can be regarded to be 'an agent' and what is 'just software'. The lack of clarity has led to a variety of definitions. As D'Inverno and Luck (2004) point out, evidence of such difficulties can be found in the large number of different agent definitions. Franklin and Graesser (1996) provide ten of those definitions of renown researchers of the field.

Russell and Norvig (1994) state in their standard textbook about Artificial Intelligence, that an agent perceives its environment and acts on that environment through sensors and effectors accordingly. Smith, Cypher and Spohrer (1994) add the important attribute of agents being persistent, meaning they act based on their own ideas, following their own agendas when completing a task. This could also be interpreted as independence or autonomy, according to Wooldridge and Jennings (1995), who claim that agents need to act without human intervention and have to be in control of their actions and their internal state. While all of the former include or focus entirely on software, Brustoloni (1991) and Franklin (Franklin, 1997) require autonomous agents to be situated in the real world.

In essence for this research it is assumed that an agent is acting in some environment or is part thereof. It is capable of deriving inputs from its environment and act accordingly in an independent, autonomous manner. Further, an agent runs over a period of time, until it finishes its task and not necessarily when the human user decides to stop it. Some agents, e.g. on the internet might even act beyond the control of any human user (Franklin & Graesser, 1996).

D'Inverno and Luck (2004) offer classifications of agents based on the type, applications and purposes of agents. They consider software agents, interface agents, personal assistant agents, believable agents, electronic mail agents, information agents and teaching agents. Others, such as Franklin and Graesser (1996) base their classification on agent's inherent properties, namely reactive, autonomous, goal-oriented, temporally continuous, communicative, learning, mobile, flexible and character-based agents. Further there are Russell and Norvig (1994), who originally suggested to classify agents based on their degree of perceived intelligence and capability. In the following chapter, each of these potential classifications will be examined to determine, which ones are more suitable for the research conducted in this thesis.

Agents have been employed in a variety of different applications, ranging from financial trading agents (Barbosa & Belo, 2008), that autonomously trade foreign currencies in real-time on the financial markets, agents in art, which create new patterns (Greenberg et al., 2013) or structured three dimensional objects (Bohnacker, 2012), furthermore, agents are widely used in computer games (Buckland, 2004) and as AI characters (Bourg & Seemann, 2004). Agents have also been used to create stories (Riedl, Thue, & Bulitko, 2011) and play online chess games (Fogel, 2002). In general, Agents can be found in so many fields, that it is probably easier to distinguish them by their underlying architecture and the type of application they represent (Franklin & Graesser, 1996). This will be discussed in the following section of the Literature Review.

2.5.1 Agent Architectures

Russell and Norvig (2003) classify agents into five groups based on the agent's level of intelligence and capability.

- *Simple reflex agents* act based on their current perception and their function follows the condition-action rule, which is usually implemented as a simple if-then decision. They require a fully observable environment in order to succeed. See Figure 5 (as per Russell & Norvig, 2003).
- *Model-based reflex agents* differ from the above mainly through their ability to handle partially observable environments. They store descriptions of the un-observable environment and act similar to the reflex agent following a condition-action rule.

- *Goal-based agents* are model-based and use a database of desirable situations for their decision making process. The agents simply choose one of the multiple possibilities that lead to the desired goal.
- *Utility-based agents* store goal-states and non-goal states, from which they choose the most desirable state. This decision is made based on a utility function, which maps the state to a measure of the utility of a particular state.
- *Learning agents* are able to operate in unknown (or non-observable) environments through learning. These agents become more knowledgeable over time, compared to their initial state. See also Figure 6 (as per Russell & Norvig, 2003).

This research specifically involves Learning Agents, in order to capture the user's intent, which is unknown at the time of programming and therefore presents a set of non-observable parameters of the agent's environment when the process of form finding is

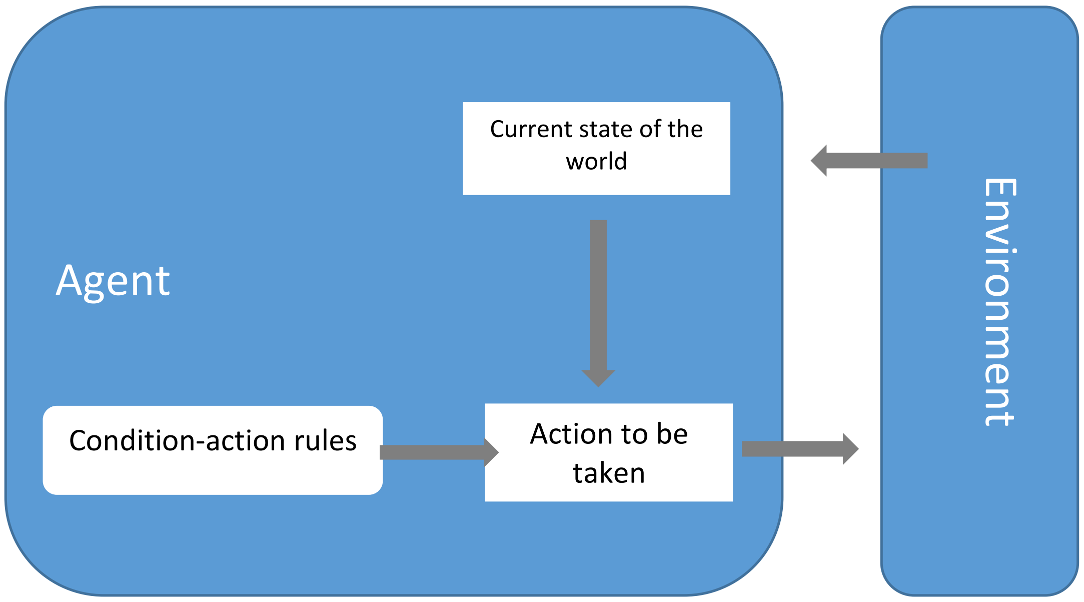


Figure 5 - Simple reflex agent (Russell & Norvig, 2003)

started. But through observation of the user's action and accordingly through evaluation of success and failure by comparison of the agent's prediction and the user's input, the agent will become more knowledgeable.

To be able to develop an agent of the Learning class, it is important to select a suitable architecture. The agent architecture is the core of an agent system, similar to the brain in humans. It resembles the underlying algorithm of the agent behaviour so that the agent is capable to solve the problem effectively. In case of this study, the aim is to choose the right candidate solutions for the following generation of the Genetic Algorithm. It is the computational methodology that creates the behaviour agents show in their respective environments. There are a multitude of different methodologies mentioned in the existing literature, which are driven by the viewpoint namely external or internal (Gómez-Sanz & Pavón, 2004). The external viewpoint reflects the classical description of an agent as autonomously interacting with its environment, while the internal viewpoint is concerned with the particular architecture and whether it is based on *beliefs*, *goals* or *plans*. For multi-agent systems, these two basic viewpoints have often been extended by different methodologies, which take for instance *user* and *organization* into account (Ricordel & Demazeau, 2002).

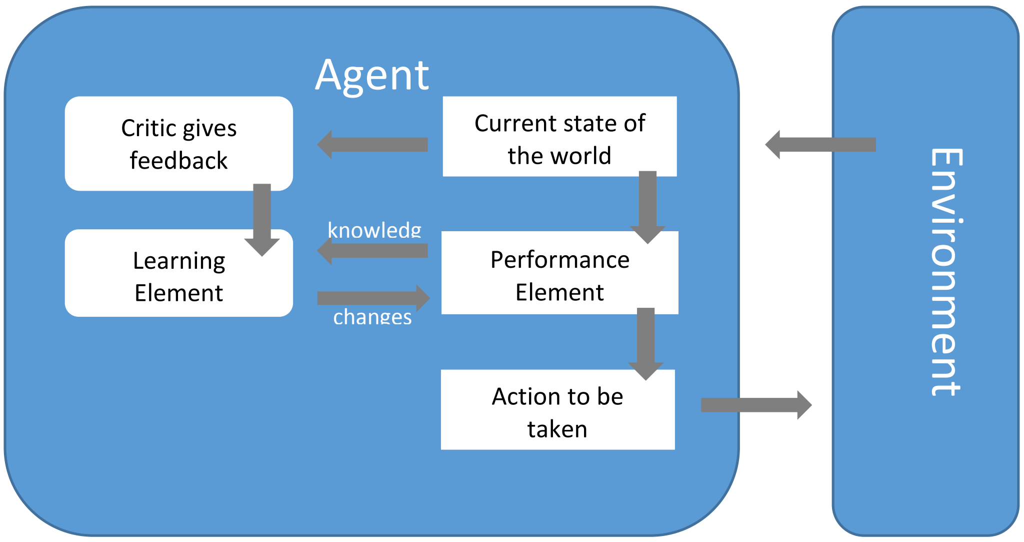


Figure 6 - Learning Agent (Russell & Norvig, 2003)

Sarker and Ray (2010) divide agent architectures into three categories: deliberative, reactive and hybrid. Deliberative architectures utilize an explicit world model to make decisions based on reasoning, pattern detection and via symbolic manipulation. Deliberative agents might take a long time to make decisions as they select the best possible action based on extensively collected information. Reactive architectures use current observations instead of symbolic reasoning to perform actions and have no inherent symbolic world model. While these architectures may be faster than deliberative agents, their outcome might not be optimal, as the decision process is instant, and does not incorporate any formal search for better solutions or incremental creation of an optimized world model. The third category, hybrid architectures are a combination of these two previous concepts and might provide a more effective and efficient solution in a reasonable time (Sarker & Ray, 2010). This categorization of agent architectures is relevant to this research, because a hybrid system seems to provide a way to account for a small, yet evolving data set (or subsequently world model), while achieving improved results in a reasonable time frame.

There are several development tools for agents and multi-agent systems available including GUI based frameworks and collections of libraries (Gómez-Sanz & Pavón, 2004). For this study, the WEKA (Hall et al., 2009) library has been employed to develop the learning agent for the software system. WEKA offers both a GUI based approach and a collection of Java libraries for modelling, learning, classification and ensemble learning techniques (Witten, Frank, & Hall, 2011).

2.5.2 Learning Agents

The idea of learning as opposed to simple behaviour is that perception is not only used to trigger certain actions depending on the observed changes in the environment, but that it is used to improve future decisions by the agent system. It is not a reaction to the environmental change, but rather a reaction to the agent's own experience (Russell & Norvig, 2003).

Many different forms of learning are being used in agent research. Some examples include Decision Trees, which is learning from observations to generate a decision flow chart, expert systems, which extract rules from examples or Reinforcement Learning, which is learning the value of actions by getting rewards or punishment depending on previously made decisions and applying these updated learned values to future actions (Sutton & Barto, 1998). In general, learning can be classified into three main categories: supervised, unsupervised and reinforcement learning (Russell & Norvig, 2003).

Supervised learning is based on examples, which are used as a training dataset to teach the system. This training set includes the right and wrong answers to a problem, which the system then learns as a function of inputs and outputs, or in other words as a relationship between actions and outcomes. This can be as simple as detecting whether an image contains a certain element (Shiffman, 2012) or which action to take, when a certain event occurs in the observed environment (Bourg & Seemann, 2004). It is important to note, that supervised learning does not require a teacher to provide the actual value for the correct solutions to the agent. The solution can also be derived by the agent from looking at all possible candidates through its own perception and getting the correct solution pointed out by the teacher. The difference is that the former requires some sort of table or key-value pairs for all right and wrong solutions, whereas the latter just requires someone to point to the right ones. This means that an agent in a fully observable environment might be able to perceive the consequences of its decisions, and learn from them to make future decisions. In a partially observable environment, this is more difficult and the agent needs more comprehensive feedback from the teacher in order to make future predictions (Russell & Norvig, 2003).

Unsupervised learning differs in that it requires detection of patterns in the observations, because the right and wrong solutions are not provided prior to the decision making process. Examples of unsupervised learning methods are statistical learning methods or some neural network implementations. Neural networks imitate the processes in the brain by using multiple simple units with inputs and outputs called *neurons* or in their simplest form *perceptrons*, which are connected in a network like structure. Inputs provide sensory information, which gets evaluated in one or more layers of neurons. The resulting sum of outputs by the neuron layers generate a behavioural pattern: If certain values reach the input side, a consistent response is created as an output (Negnevitsky, 2004).

Reinforcement learning is probably the most complex, but also most general learning method (Russell & Norvig, 2003). Actions taken by the agent inevitably lead to consequences, good or bad. The evaluation of successful and unsuccessful actions is used to maximize a reward function. The agent is not being led through the learning process as in supervised learning, but instead derives the most successful actions from the rewards it gains by trying them out (Sutton & Barto, 1998). Therefore, reinforcement learning does not require an expert, who provides the right or wrong solutions, which is an important feature of this type of learning. But more significantly, the agent is able to engage with uncertain, unknown new territory, because it learns entirely from its own experiences and not from a knowledgeable teacher. But reinforcement learning also imposes an important issue on the design of the agent architecture: Depending on the problem set and the intended use of the agent, a balance between exploration and exploitation must be maintained. If the agent has to prefer actions that lead to maximum rewards in order to arrive at a certain goal. At the same time, this implies that some actions which have not been tried before, might never be explored, although they could lead to even higher rewards and ultimately to the best outcome overall (Wiering & Otterlo, 2012).

Agent research, and study of learning agents specifically, a vast field of research and many concepts have been developed to improve aspects of agent architecture, communication and performance. One performance enhancement of learning algorithms, among many others, is Boosting. Boosting is a generic and often effective method of creation reliable predictions in machine learning (Freund, Schapire, & Abe, 1999). Learning algorithms often suffer from noise in the data or small numbers of training examples (T. Mitchell, 1997). Analysing and tracking the training error by use of a test set, and combining multiple resulting classifiers based on their training error score into a meta-classifier, enables the Boosting algorithm to classify instances better than individual classifiers based on noisy or small training data as shown by Schapire (1999). Human-centric evolutionary computation works with relatively small data sets, where tens or hundreds of iterations are typical, compared to non-interactive Genetic Algorithms, where thousands or millions of generations are possible depending on the computational resources. Therefore, the ability to train learning algorithms using advanced methods such as Boosting gain more importance in Interactive Evolutionary Computation as suggested by Kamalian et al. (2006) in context of electronics engineering design.

2.5.3 Multi Agent Systems

Multi Agent Systems (MAS) are computational systems that integrate more than one type of agent. These agents might interact and communicate with each other and perform different or similar tasks (Gómez-Sanz & Pavón, 2004). MAS are employed when a single agent might fail to solve problems by itself, because the problem is either too difficult to encapsulate into a single agent or it is impossible to do so. For example, if a human (agent) and a computational agent interact while working on a task, or if multiple different agent architectures have to be employed to solve an issue because the problem is beyond the scope of an individual agent (Balaji & Srinivasan, 2010).

MAS can be classified into homogeneous and heterogeneous architectures. In case of a homogeneous structure, all agents have the same underlying architecture. They only differ with regards to the environment they are in. Every agent contributes to the overall system by observing parts of the environment that other agents can not perceive. Sometimes there is an overlap between the observations made, in case that agents partially share the same part of the environment. In contrast, heterogeneous systems are made of agents of different architectures. The agents perform different tasks in different ways and complement each other. The most extreme example for a heterogeneous structure might be a MAS of human and computational agents. Homogeneous systems are relatively fast to create as they only require a single agent architecture. The advantage of heterogeneous systems is their ability to account for a wide range of different tasks, while keeping the individual agent relatively simple (Balaji & Srinivasan, 2010).

This study aims to combine interactive computation with autonomous computational agents. Therefore, the concept of MAS is important to understand. But as this study uses MAS in a very specific way, communication between agents and their hierarchical structure differs from common MAS and is detailed in the following section 2.6, in context of Human-based Genetic Algorithms.

2.6 Human-based Genetic Algorithm

Human-based Genetic Algorithms (HBGA) are an extension of Interactive Genetic Algorithms, which in turn are a sub-set of Interactive Evolutionary Computation (Figure

7). They employ the concept of multiple agents including the human user in a multi-agent system (MAS), which allows for a flexible, modular approach to combine the processes of selection, mutation and crossover. Accordingly, they are also referred to as multi-agent Genetic Algorithms (Kosorukoff, 2001). This section of the literature review, considers their layout and design, and will discuss how the previous sections are linked to Human-based Genetic Algorithms, and how they contribute to the software system as part of this research.

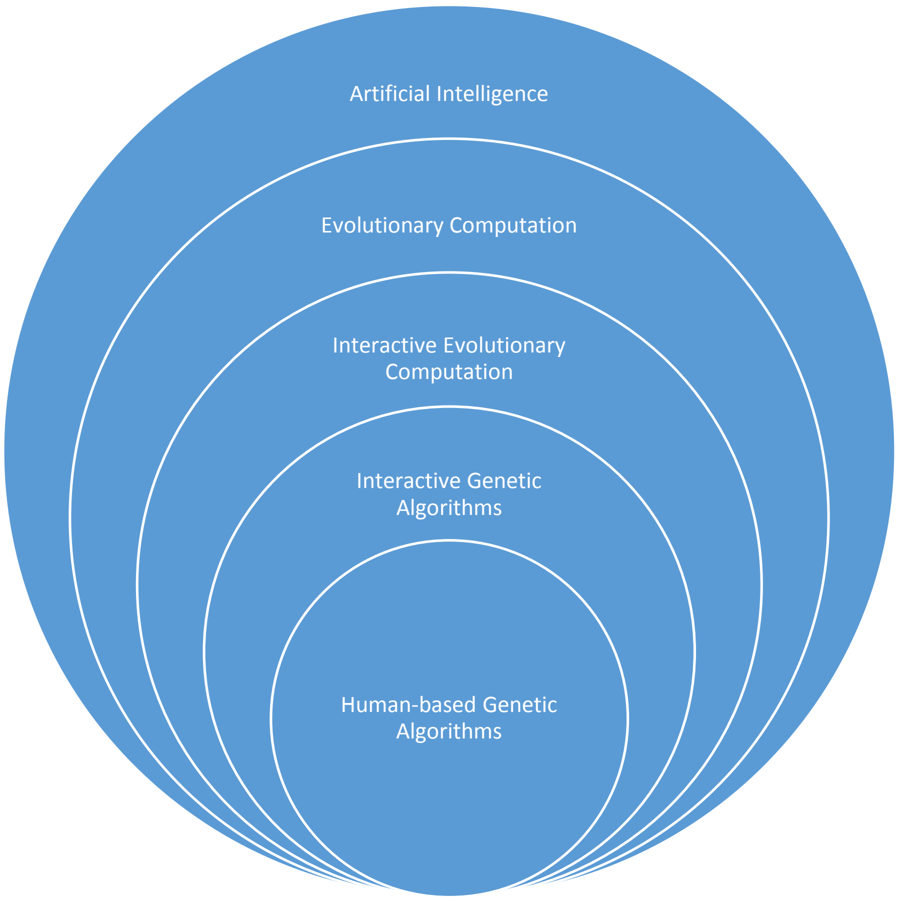

Artificial Intelligence
Evolutionary Computation
Interactive Evolutionary Computation
Interactive Genetic Algorithms
Human-based Genetic Algorithms

Figure 7 - Human-based Genetic Algorithms in the field of Artificial Intelligence

2.6.1 Multi-agent Genetic Algorithms

First introduced by Alex Kosorukoff as part of his research into knowledge management, Human-based Genetic Algorithms (Kosorukoff, 2001) are an additional class of genetic algorithms. Kosorukoff describes them as a form of outsourced primary genetic operators, which are the processes of selection, crossover and mutation. Drawing the parallel to a business organization, he exemplifies outsourcing as the transfer of "ownership of a business process to an external agent" (Kosorukoff, 2001). Kosorukoff further points out, that outsourcing effectively means the transfer of a function from the organization to an external agent. This function will be performed independently and unsupervised by the agent, sometimes even without any knowledge of how the agent works, which methods the agent employs and most importantly, partially or fully beyond the control of the organization. The organization only controls the choice of agents, but not their functionality. Similarly, an organizational function is introduced to coordinate the system of multiple agents.

In Human-based Genetic Algorithms, the three primary are simplified into *selection* and *recombination* (merging crossover and mutation). These two main functions can be taken over by either human or computational agents – not just exclusively, but even in combination. For example, there may be a computational recombination agent, a human selection agent like in Interactive Genetic Algorithms plus an additional computational selection agent (Figure 8). This is the defining feature of Human-based Genetic Algorithms. The term Human-based Genetic Algorithms may be considered to be slightly misleading for a number of reasons. Firstly, this class of Genetic Algorithms does not exclusively incorporate human agents. Secondly, the distinguishing feature is the use of a multi-agent system not necessarily a human-based system. Kosorukoff's own publication (Kosorukoff, 2001) also used the term multi-agent Genetic Algorithms, which would be a more distinguished term and probably less prone to be misinterpreted. Also, please note, in order to simplify reading this text, any human agent will be referred to as 'human' and any computational agent as 'agent'.

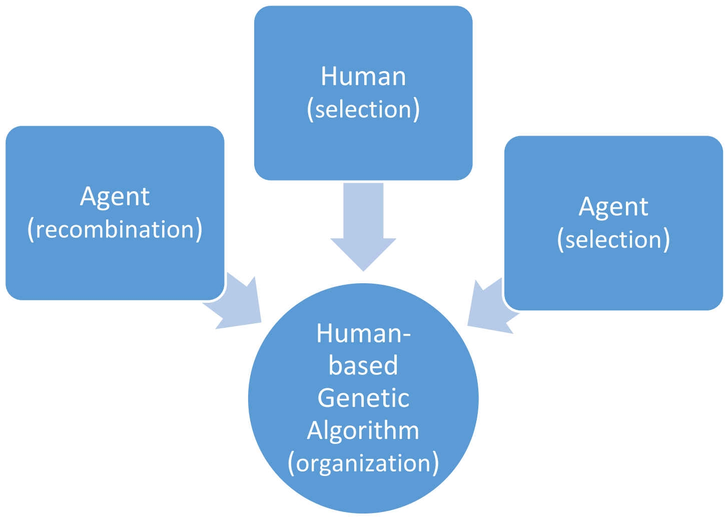


Figure 8 – Example of Genetic Algorithm as multi-agent system

Kosorukoff also identified a significant implication of Human-based Genetic Algorithms: The organizational function needs to be efficiently and carefully designed in order to allow for effective agent-agent or agent-human interaction (Kosorukoff, 2001). While this indication seems to be correct, it is equally true for simple Genetic Algorithms and especially Interactive Genetic Algorithms. The program structure defines how the entities (computational or human) interact, how effectively they perform and whether it is possible to achieve any sensible, desired outcomes at all. Therefore it seems as if this seemingly generic problem is outweighed by the robustness and flexibility of a multi-agent system. In relation to this study, it seems to be a huge advantage to be able to utilize agents in addition to human selection in order to augment the before mentioned issue of user fatigue. If a computational agent performs one or many iterations of selection instead of the human user, the capability of the human is probably utilized in a better, more effective way, and therefore allows for an overall larger number of iterations, which in turn leads to a better convergence of desired and achieved results. This is one of the core ideas, which this research seeks to explore.

In context of Human-based Genetic Algorithms, the idea to add several agents to the multi-agent system in order to perform the same task in a similar fashion, but based on different learning algorithms is called ensemble learning (Witten et al., 2011). Specifically, multiple classification models could be combined in order to increase performance of the overall system. This is known as *Boosting* in machine learning and has been established and thoroughly investigated by Freund and Schapire (1999). While this could be an interesting extension of this study, as it might lead to improved learning ability of the selection agent, it has to be reserved for future as it is beyond the scope of this thesis research (see section 5.4.3 - Boosting Human-based Genetic Algorithms).

2.7 Summary

In the literature review, three main areas relevant to this research have been highlighted. Procedural Cities in context of Generative Design, Genetic Algorithms as part of Evolutionary Computation, and the concept of Autonomous Agents and Multi Agent Systems. A critical discussion of some of the aspects of Genetic Algorithms, and in particular Interactive Genetic Algorithms has been provided. Extensions of the concept using an Agent-based approach have been shown and gaps in existing knowledge have been revealed. While the literature provides an understanding of what Generative Design and Evolutionary Computation contribute to current knowledge, there are only few examples of applying Interactive Genetic Algorithms for Design applications. In particular, there is no existing research using Human-based Genetic Algorithms, which employ human and computational agents to counteract the effects of fatigue in a computer graphics design context. The following section discusses the mixed method Methodology of this study, and introduces software systems and experimental process as part of the research design.

3 Methodology and Research Design

3.1 Methodology

This study uses a practice-led methodology (Candy, 2006) in conjunction with Design Science Research as suggested by Hevner et al (2004). Elements from Action Research (Davison, Martinsons, & Kock, 2004) and Empirical Research (Cohen, 1995) are also employed. As a result, the research is characterized by a pragmatic research philosophy using a mixed methods approach.

Practice-led research is an offspring of practice-based methods and both share a similar foundation and have many overlaps (Dean & Smith, 2009). But there are some significant differences, most importantly, practice-led research is not so much concerned with an actual product or artefact as its main outcome. Practice-led research aims to enhance the knowledge about processes and practice instead. The objective is to advance techniques and workflows, rather than merely creating an artefact. Practice-led research also seeks to create a better understanding of practice (Candy, 2006). The selection of procedural city generation as an application seeks to do this, namely through using different processes to create the building blocks of such cities. For example, this study employs an approach based on Delaunay triangulation (Okabe, Boots, Sugihara, & Chiu, 2000) for the street system (section 3.2.3.2) instead of a self-connecting L-system as proposed by Parish and Müller (2001). This allows the integration of global goals for the street system as pointed out by Parish and Müller. With a larger, coarse Delaunay mesh in addition to the smaller, fine grained local street level, the impression of main arteries in the street system is achieved as discussed in section 3.3.

Design Science Research (Hevner et al., 2004) on the other hand, adds the intent to widen the knowledge about human behaviour and predict its outcome. It aims to improve human (and organizational) capabilities, while creating new artefacts, a property shared with practice-led research. Further, Design Science Research is concerned with the improvement the performance of an artefact, which leads to the core of this study: Combining two existing, well researched and proven areas of artificial intelligence, to enable the designer to create very complex structures with comparatively little effort. The outcome is a process that allows users with limited knowledge about the huge parameter set of a design system to achieve significant outcomes.

The process involved has been detailed as a sequence of five phases (Figure 9) including an outcome expected for each individual step (Vaishnavi, 2008). The first phase is called Awareness of the problem and basically involves successfully identifying an issue in the field of research or sometimes an adaption of a problem from an allied discipline. The outcome of this phase is a formal or informal proposal. The second phase is closely related and somewhat intertwined with the Awareness, and is named Suggestion. Suggestion is essentially the first step towards a possible solution of the problem identified, and is usually a combination of existing elements, or new and existing elements, which offer new or improved functionality. The outcome is a provisional or tentative design.

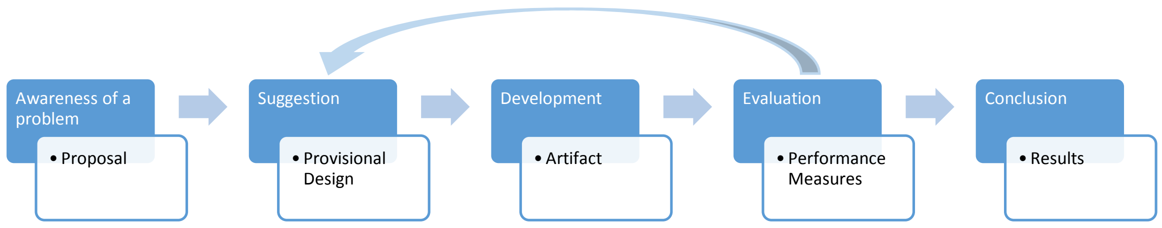


Figure 9 - Design Science Process

In phase three, called Development, the provisional design is further refined and implemented, resulting in an artefact. According to Vaishnavi (2008) this elaboration of the provisional design might require a significant creative effort and potentially the use of a high-level tool or package for its software development. The fourth phase of Evaluation is often the start of an iterative process: The artefact is evaluated and new ideas, insights, additional knowledge gained from producing the artefact or running it, are fed back into Suggestion to start a new iteration. Performance measures guide the Evaluation and provide a tool to gauge the improvements in this iterative loop. Finally, the fifth phase concludes the process of Design Science Research and produces results, which may be consolidated and *written up*. Results can often be categorized into *firm*, that is verifiable and repeatable outcomes, and remaining *loose ends* which defy explanation or expose anomalies in their behaviour. These are usually subject to future research. While the method is assumed to be linear from start to end, in practice it is often cyclic, as pointed out by Vaishnavi (2008).

This study combines both practice-led research and Design Science Research. The rationale for this springs from the idea to create a new workflow, specifically using RenderMan file format to synthesize a new pipeline for procedural city modelling. As later discussed in section 3.2.2, the use of RenderMan specific file formats bears quite a few technical advantages, and is a novelty in context of computationally generated city models. The development of the new workflow is done through practice-led research. Contrary to this first main contribution of this study, the implementation of an interactive evolutionary software tool and adding autonomous agents to the process required an iterative approach. Therefore, Design Science Research has been employed as a second methodology in order to enable a more traditional software development process. Especially the introduction of a graphical user interface (GUI) required quite a few iterations. Mainly, to create a simple interface, as any complex user interaction could have had an additional impact on the measurement of user fatigue. A steep learning curve might accelerate user fatigue and could potentially lead to misinterpreted results of this study. Simplifying the user interface as much as possible seemed to be a good choice, as a high number of UI elements requires more responses from the user (at least to understand what is presented to them) and therefore might impact on the overall fatigue. A reduced number of UI elements in turn, would focus this research on the fatigue caused by a large number of iterations, and therefore focus on the difference that an agent might make to an interactive genetic algorithm.

Action Research has also been considered as an appropriate methodology for this research. Action Research in computer science aims to understand real-world problems and design potential solutions, while simultaneously expanding scholarly knowledge. It therefore differs from most empirical methodologies as it does not just observe the world in its current state, but actively improves the studied situation (Easterbrook, Singer, Storey, & Damian, 2008). It could be argued that this study presents a new software system, based on a newly developed pipeline, which seeks to solve the issue of complex model creation in computer graphics. At the same time an evaluation of the process used to solve the real world problem, namely employing Interactive

Evolutionary Computation and adding agents to compensate for inherent problems resulting from the interactive process, is given. But comparing Design Science Research and Action Research, it seems that the focus of this study is on solving a technical problem using a development and evaluation process that introduces a new technological solution, namely using RIB archives, RenderMan compliant output and interactive design software in conjunction with Evolutionary Computation and Agents. This leads to the decision to employ Design Science Research, rather than Action Research, which focusses on understanding the problem more than providing a technical solution for it (Iivari & Venable, 2009).

The final element of the methodology used in this study is based on concepts from Cohen (1995). In his book "Empirical Methods for Artificial Intelligence" (Cohen, 1995), the author draws comparisons between research with moderately intelligent animals and agents. The latter behave fairly similar in the sense that the required methods can be borrowed from biological or behavioural research. For example, the scientific task is identical, whether the subject of study is a lab animal like a rat or an agent program, and is based on the following three elemental research questions (here for the case of an agent program):

1. *How will a change in the agent's structure affect its behaviour given a task and an environment?*
2. *How will a change in an agent's task affect its behaviour in a particular environment?*
3. *How will a change in an agent's environment affect its behaviour on a particular task?*

(Cohen, 1995)

To be able to answer these questions, an empirical generalization strategy is used, which is centred around the idea of building a program that shows a certain behaviour, while running in a particular environment. Next, features that lead to this behaviour have to be identified and built into a causal model. This model, once its predictions are accurate, is generalized and used to make predictions in other environments or for other tasks (Cohen, 1995). This essentially resembles the development of the agent architecture as discussed in section 3.2.4 in greater detail.

In context of the research design for this study it is also noteworthy that an interactive design process versus a fully automated approach has been preferred not just for the reason of practicality of finding a function for aesthetic as mentioned before. It is also about allowing a user with very limited knowledge about computer graphics modelling to create complex models. Changing from a purely (or predominantly) computational to an interactive approach, shifts the design process from the programmer creating the software system to the user of such software system. If a computational process was chosen, the programmer would implement the selection function and therefore predetermine all or most of the final layout of the city. While the software could still create variations, it would still converge towards the optimal solution of the mathematical aesthetic function as implemented during programming. Different preferences and taste of designers could not be captured by such system. The interactive evolutionary approach provides a reflection of what the user intends at run time of the software. Should the user decide to change the intent, the system adapts to that. Some evidence of this is shown in Results and Observations (section 4). Further, some of the implications of this are discussed in section 5.

3.2 Software System

The problem discussed in this research is rather complex: It is multidisciplinary and is comprised of three major sub-problems. Accordingly, the presented solution has multiple software components and utilizes two different AI techniques: Genetic Algorithms and Agents. To account for the complexity of the issue and to reduce the chance of creating hidden issues which negatively influence the outcome of the research, a simple approach known as *Dynamic Programming* (Sniedovich & Lew, 2006) has been chosen: The topic is broken down into smaller problems to be solved individually before merging the solutions to resolve the overarching issue.

Firstly, a rather simple Interactive Genetic Algorithm for solving a design problem has been implemented. The design issue resolved with this approach was the creation of Infographics, specifically Geotagged maps of New Zealand. Geotagged maps incorporate location markers into the design. For this specific application, the map colours, the marker shape and colours as well as the angle of the observing camera haven been chosen as parameters of the design problem.

Secondly, the Interactive Genetic Algorithm was applied to the considerably more complex task of creating a three dimensional digital city model. The process was also expanded through the concepts of Human Based Genetic Algorithms, by dividing the software system into agents, computational and human. Both human and computational agents are supervised by an organizational entity, which also runs the actual evolutionary process as explained in section 2.6. From a design stand point, the goal was to assemble a CG scene with full three dimensional geometry, the option to apply textures and adjust the virtual camera, and visualize by rendering images using a commercially available render engine. These requirements were intentionally chosen in order to guarantee that the study reflects a practical situation and therefore provides outcomes that are measureable and verifiable. The city model also introduces a significantly higher complexity and much larger set of adjustable parameters. In a manual modelling situation, the user would have to understand and work with this parameter set. In case of this study, this is being avoided through the use of evolutionary computation. The user interface has been kept as simple as possible to achieve results that can be easily reproduced. Further, the simplicity reduces the potential of misinterpretation of the result observed.

Both case studies were designed with slightly different goals in mind. The former, Interactive Genetic Algorithm was targeting an understanding of computer graphics applications in context of evolutionary computation. As mentioned in the literature review, there are very few and very simple graphics design applications using interactive evolutionary computation such as the colour or pattern finding projects. Other examples include engineering design, which aims to achieve mechanical, physical or mathematical design goals, but not any aesthetic intent. This gap is filled by the Interactive Genetic Algorithm creating Infographics.

The latter case study, using Human Based Genetic Algorithms to create three dimensional computer graphics models, specifically models of urban environments, had the goal to build on the first case and add computational agents to support the user. The agent runs some of the selections and therefore reduces the workload on the user, potentially counteracting fatigue and perhaps increasing the number of possible iterations to refine the final model. Another aspect is the possibility to discover new designs, which have not been anticipated during conceptualization of a particular city design. The idea of using agents as an entity to create diversity is known as Novelty Search (Velez & Clune, 2014). While novelty search is not part of this thesis, Human-based Genetic Algorithms lay a foundation for this type of research, due to their inherent flexibility in agent configuration and organization.

Both case studies are detailed in sections 3.2.3.1 and 3.2.3.2.

The software prototypes have been developed using the Java programming language. The render engine used to visualize the results is RenderMan/3Delight ("Welcome | 3delight," 2014) and renders RIB archives written in ASCII mode. The GUI is based on the Processing framework (Fry & Reas, 2004).

The final sections 3.2.4 discusses the specifics of the Agent architecture as it has been implemented for the software prototype of the Human Based Genetic Algorithm.

3.2.1 Pipeline

The workflow, commonly labelled as "pipeline" in design and computer graphics contexts (Okun & Zwerman, 2010), simply describes a serial set of individual processes, much like the instructions in a recipe. Unlike an algorithm, which has a similar underlying structure, but features decisions, loops and other control elements, it usually encapsulates a linear process.

A resemblance of a state of the art visual effects pipeline including commonly used commercial software packages has intentionally been selected for this study for the following reasons. Using some of the tools of the industry such as Autodesk Maya, RenderMan compliant 3Delight renderer and common file formats found in most visual effects and design pipelines might make the core contribution very useable with little or no adaption work to be done by game and digital effects facilities. Therefore, the outcomes are widely applicable. Also, time constraints of this study were an additional factor: Developing every element from scratch would have been beyond the scope of this thesis. But the main reason behind the choice is consistency. When looking for testable and repeatable results, and draw defendable conclusions based on measured data, any hidden error in the software system, would prove to be problematic. Commercial software produced for a range of very large industries like game and film is tested and revised through many iterations and quality assurance processes as pointed out by Okun and Zwerman (2010). Within the scope of a research project like this one, a thorough and deep testing of components that are not directly related to the core concern of the study would be impossible, at least at the depth and scale of what commercial suppliers such as Pixar or Autodesk provide. Therefore, every component and every step of the presented pipeline has been carefully selected with the idea of consistency in mind.

The number of possible options and parameters of some of the steps in the process has also been reduced quite intentionally, following the same notion of stability and repeatability. For example, while it would have been possible to evolve each individual building that is used as part of the evolutionary computation for the final city, this additional layer has been removed. Another important reason was the limited hardware accessible for this study compared to what companies such as Weta Digital or Pixar provide. Without the use of a commercial size renderfarm, additional detailing is simply prohibitive both in terms of render times, but also regarding memory size and storage space. Instead, instances of one object are being used, which are simply copies of the same object at different scale in height and width. This work uses a combinations of multiple instances to form new variations of buildings, but overall this decision has reduced the number of possible variants, which leads to simpler appearance, but significantly more consistency in measurable results. The aim is not to create the most believable and pretty city, but to evaluate the performance of the proposed workflow in conjunction with a human user. Reducing the options focusses the user's attention on fewer details and allows for more accurate statements about the process. Probably the largest variable in measured data is the human designer expressing taste, preference and ideas through selection. It therefore seems necessary to reduce the variance were it is possible and least problematic. And while the simplification potentially offers less visual appeal than a fully photo-realistic render of a city, this decision probably did not impact on the final renderings as a suitable vehicle for this study, as render quality and realism are not part of this investigation.

A simplified overview of the city modelling pipeline used in this study is shown in Figure 10. Possible inputs for the core of this research are 3D building models (either evolved from primitives or manually created) or simple geometric primitives such as cylinders, cubes or combinations of both. These basic elements are passed on to the Human-based

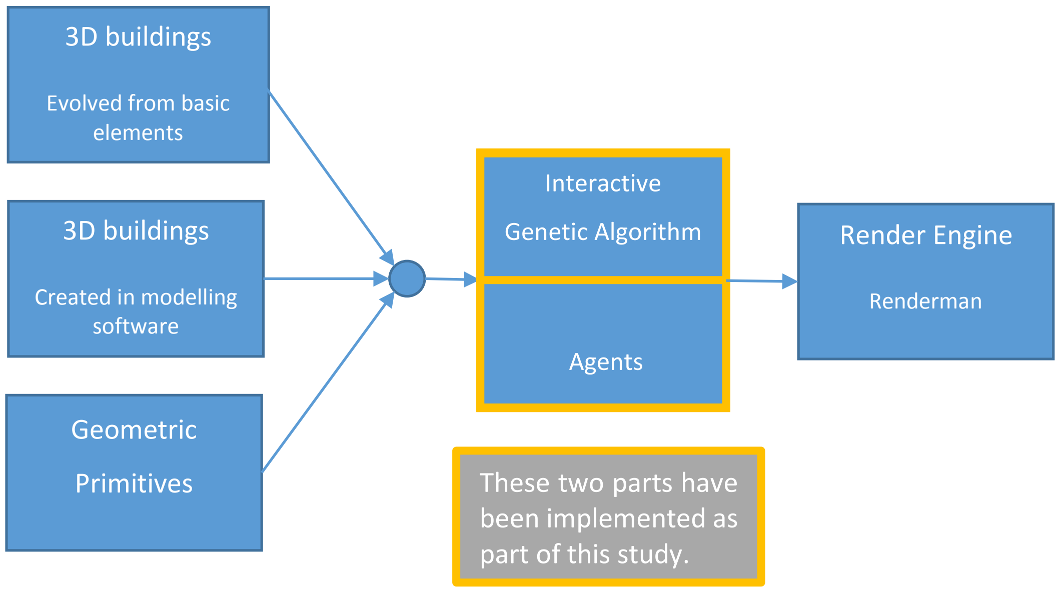

3D buildings
Evolved from basic elements
3D buildings
Created in modelling software
Geometric Primitives
Interactive Genetic Algorithm
Agents
Render Engine
Renderman
These two parts have been implemented as part of this study.

Figure 10 - simplified pipeline overview

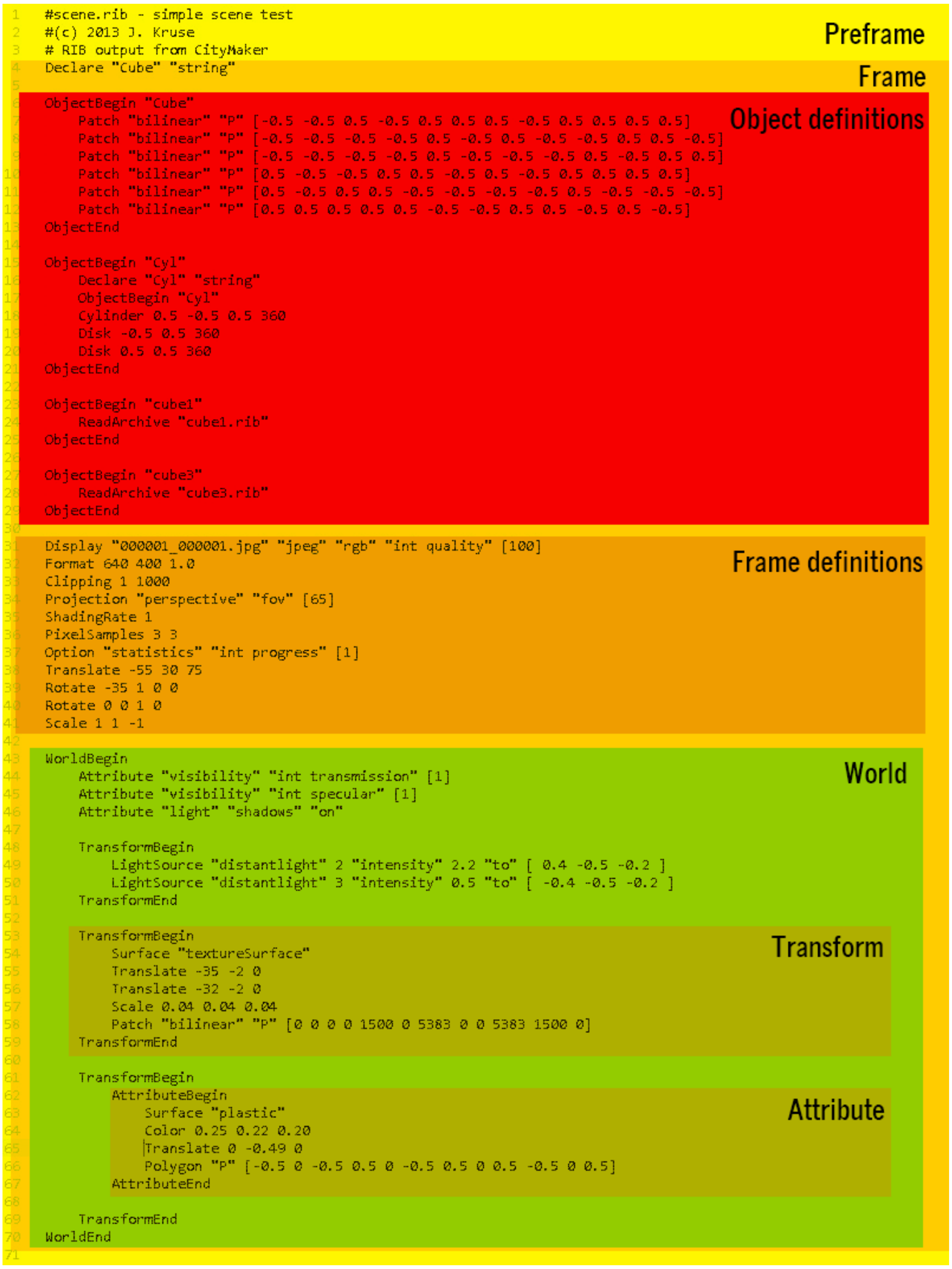
Genetic Algorithm, which is a combination of Interactive Genetic Algorithm and agent. This is the core of this research and represents, what has been implemented as a main part of this study. The output of this stage of the pipeline is a RIB archive as shown in the following section of this thesis. The RIB archive is passed on to the render engine, which generates the new images for the next generation of candidates. In case of this research, 3Delight (“Welcome | 3delight,” 2014) has been used for rendering. The open architecture of this pipeline allows for easy adaptation to existing work environments and makes the core of this research a very practical component for future study as well as industry application. Possible options have been suggested, in order to extend the presented pipeline in Section 5.4.

3.2.2 RIB archives

The models that are being created during the evolutionary process are saved in RIB files (RenderMan Interface Bytestream), which contain the scene and its elements including all models, instances, shader calls and texture elements in form of RenderMan Scene Description Language (“RenderMan Options and Attributes,” 2014). This file format is available in ASCII and binary, with the former being used in this study for clarity and simplicity. RIB files, also called RIB archives, contain the full render description for each candidate of a population and therefore each intermediary city model description is preserved for the whole process of evolving the final city. This allows for later analysis of the process at large, as well as each individual generation of cities. In general, RIB files commonly represent one image or one single frame of a sequence to be rendered. Each RIB archive gets processed by the RenderMan (or RenderMan compliant) render engine separately. The result is a single image, for example Jpeg or Tiff.

Figure 11 shows an example structure of a RIB archive as used in this study. The RenderMan Scene Description Language uses bracketed blocks for some of the items in the file such as Attributes, Transforms or the overall frame. Additionally, some common items such as frame description and preframe are single lines led in by a predefined keyword. Neither the order of the elements, nor the indentation is required and serves mere aesthetic and practical purposes to make reading the files easier.

First, there is the overall frame, which is only preceded by the preframe – normally comments. The frame contains global information such as global transforms and attributes. The following section in case of this example is a range of object definitions. These object definitions can be instanced in the world block. Using instances reduces the memory usage and storage requirements, and keeps the file simple and again more readable. Object definitions can either be primitives like cylinder, torus and sphere, or files that are read by the render engine during render time, for example additional RIB archives containing objects made of multiple primitives. Following are additional frame definitions that instruct the render engine to perform operations such as camera setup, image plane clipping and image file format and size. The next section is the main section of the file called world. It is bracketed by a WorldBegin and WorldEnd tag, and contains all attributes and objects that are actually to be rendered. In the World block, instances of previously defined objects or read-in RIB archives, are stored. Attributes can be single line items or blocks and apply to the following object call or instance call. If attributes and objects are bracketed by a TransformBegin and TransformEnd, any global transform (translate or rotate in the frame section) has no influence on this particular bracketed item. Instead, the local translate and rotate information within the transform block is applied. This is similar to pushMatrix() or popMatrix() in other programming languages and frameworks like Processing or Openframeworks.



```
#scene.rib - simple scene test
#(c) 2013 J. Kruse
# RIB output from CityMaker
Declare "Cube" "string"

ObjectBegin "Cube"
    Patch "bilinear" "P" [-0.5 -0.5 0.5 -0.5 0.5 0.5 0.5 -0.5 0.5 0.5 0.5 0.5]
    Patch "bilinear" "P" [-0.5 -0.5 -0.5 -0.5 0.5 -0.5 0.5 -0.5 -0.5 0.5 0.5 -0.5]
    Patch "bilinear" "P" [-0.5 -0.5 -0.5 -0.5 0.5 -0.5 -0.5 -0.5 0.5 -0.5 0.5 0.5]
    Patch "bilinear" "P" [0.5 -0.5 -0.5 0.5 0.5 -0.5 0.5 -0.5 0.5 0.5 0.5 0.5]
    Patch "bilinear" "P" [0.5 -0.5 0.5 0.5 -0.5 -0.5 -0.5 -0.5 0.5 -0.5 -0.5 -0.5]
    Patch "bilinear" "P" [0.5 0.5 0.5 0.5 0.5 -0.5 -0.5 0.5 0.5 -0.5 0.5 -0.5]
ObjectEnd

ObjectBegin "Cyl"
    Declare "Cyl" "string"
    ObjectBegin "Cyl"
    Cylinder 0.5 -0.5 0.5 360
    Disk -0.5 0.5 360
    Disk 0.5 0.5 360
ObjectEnd

ObjectBegin "cube1"
    ReadArchive "cube1.rib"
ObjectEnd

ObjectBegin "cube3"
    ReadArchive "cube3.rib"
ObjectEnd

Display "000001_000001.jpg" "jpeg" "rgb" "int quality" [100]
Format 640 400 1.0
Clipping 1 1000
Projection "perspective" "fov" [65]
ShadingRate 1
PixelSamples 3 3
Option "statistics" "int progress" [1]
Translate -55 30 75
Rotate -35 1 0 0
Rotate 0 0 1 0
Scale 1 1 -1

WorldBegin
    Attribute "visibility" "int transmission" [1]
    Attribute "visibility" "int specular" [1]
    Attribute "light" "shadows" "on"

    TransformBegin
        LightSource "distantlight" 2 "intensity" 2.2 "to" [ 0.4 -0.5 -0.2 ]
        LightSource "distantlight" 3 "intensity" 0.5 "to" [ -0.4 -0.5 -0.2 ]
    TransformEnd

    TransformBegin
        Surface "textureSurface"
        Translate -35 -2 0
        Translate -32 -2 0
        Scale 0.04 0.04 0.04
        Patch "bilinear" "P" [0 0 0 0 1500 0 5383 0 0 5383 1500 0]
    TransformEnd

    TransformBegin
        AttributeBegin
            Surface "plastic"
            Color 0.25 0.22 0.20
            Translate 0 -0.49 0
            Polygon "P" [-0.5 0 -0.5 0.5 0 -0.5 0.5 0 0.5 -0.5 0 0.5]
        AttributeEnd

    TransformEnd
WorldEnd
```

Figure 11 - Structure of a RIB archive

The example file shows only two objects (a patch and a bilinear), while the RIB archives used in this study have an additional number of instances, one for each building. This means, for the examples shown in this study, more than 10,000 instances have been created in the files and rendered into images by the render engine per candidate solution. To make the process more accessible, the instances have been simplified and are mostly textured cubes and cylinders. Suggestions, on how to expand on this foundation and how to increase the level of detail as part of future research, can be found in section 5.4 of this thesis. Instancing every object in the file will also enable any industry adaption quite easily, as the file name for ReadArchive can simply be replaced with any desired model, for example a detailed building. The presented pipeline would automatically incorporate such building at render time, and depending on availability of render power and memory capacity, significantly more complex city models could be generated by the same method.

Creating and writing RIB archives for this study has been implemented as a Java class, which adds all elements line by line and block by block (for the instanced buildings of the city). The result is saved as a *.rib file. Rendering the rib into an image file is done by a second Java class. This render class calls the RenderMan compliant renderer as a sub process and the previously written RIB file is passed as an argument. The rationale behind this is to keep the process of creating and rendering RIB files as flexible as possible. This allows for easy adaption to other RenderMan compliant render engines such as Pixar RenderMan, 3Delight or Pixie. While these all add a lot of features to the core RenderMan definition, they are all usable as the RIB file created by the Java class is strictly compliant with the original RenderMan format. The advantages of this approach are not only the flexibility mentioned above, but also possible adaption for future RenderMan compliant render engines and possible expansion using proprietary features of the chosen renderer. For example, 3Delight allows to add true Ray tracing features to a render. Shaders written to employ Ray tracing for reflections of glass and metal could easily be called from the created RIB file and applied to the instanced buildings as required. In the spirit of a practical visual effects or game development pipeline, the open RIB archive architecture provides expandability and adaptability for a variety of requirements as well as future proof extensibility.

The actual process of assembling the RIB archive file varies depending on the type of objects to be rendered. This is therefore discussed within each of the following sections 3.2.3.1 for Infographics and 3.2.3.2 for procedural cities.

3.2.3 Applications

This study employs two different applications as detailed in the following two subsections. Reasons for this split into multiple applications are given, the underlying algorithms described, and the user interface – which is particularly relevant due to the interactive nature of the project – is explained.

3.2.3.1 Infographics

During the interactive design process, the user is presented with 12 maps generated by the algorithm as shown in Figure 12. The user selects one map with the desired water colour, landmass colour, location marker shape and marker colour as well as camera angle. These attributes of a map are named abstraction parameters in the following, as they do not represent a direct parameter of the map such as RGB color values, but an abstract description, loosely modelled on human description. These abstraction parameters were derived through a phenomenological (empirically relating the underlying parameters to abstract description) approach. The idea here is to abstract parameters even if they could be directly entered by a savvy user. The reasoning behind this is to account for more complex items, which could not as easily be controlled by the user and therefore justify computational help. For example, most computer users are familiar with colour selection tools from photo editing software or even word processors. Such user would have little difficulty entering colours for water and land. But when it comes to three dimensional virtual camera control (the perspective that the map is shown from), even moderately educated animation artists often struggle, based on the researchers experience. A simplification or abstraction for camera control would be beneficial. Therefore, in order to keep a consistent control interface, all input parameters have been abstracted.

In case that the user is not satisfied with any of the 12 maps, he is asked to select the closest possible candidate map. Aside from this selection, there is only one dial for the user to give a 'confidence vote' of his selection and a confirm button. The confidence is

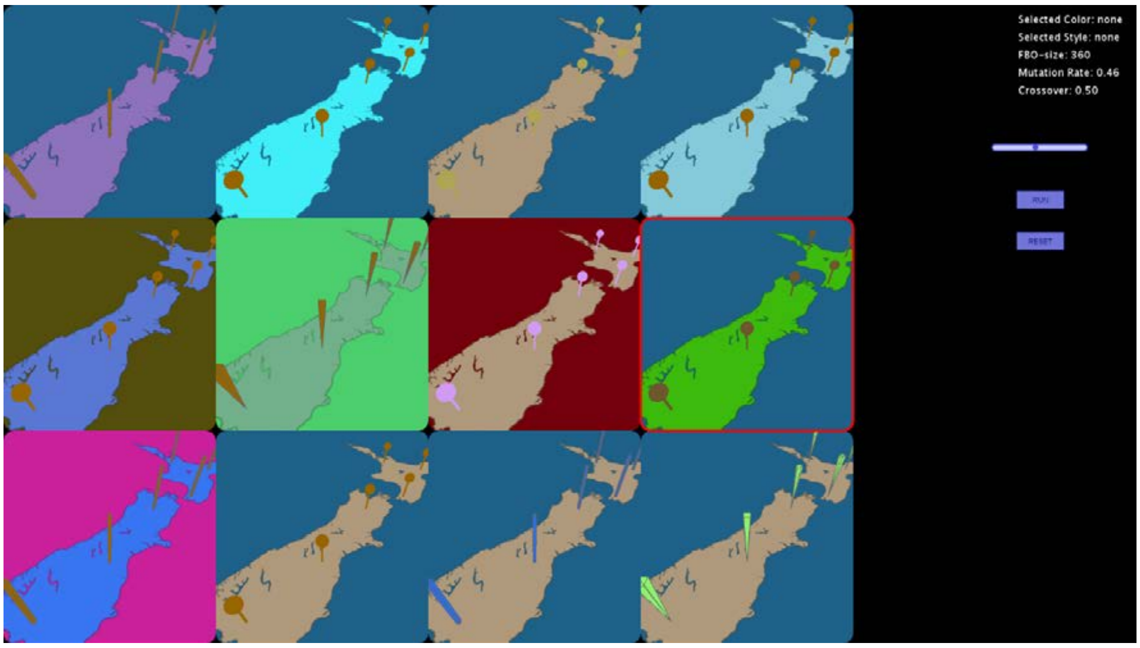

Selected Color: none
Selected Style: none
FBO-size: 360
Mutation Rate: 0.46
Crossover: 0.50

Figure 12 - Infographics using Genetic Algorithms

an abstraction that reflects the level of satisfaction of the selections made in relation to the desired design goal. For example, if the map colour and other parameters are all fairly close to the design goal, but simply not spot on, the user might select a high confidence level. If the maps are all not a match for the desired design, the confidence vote would be very low. While this seems to be a very fuzzy, subjective parameter, in practice it works really well and had a high influence on the speed of the design process. The parameter is part of the fitness evaluation of the Genetic Algorithm and is used to control the mutation rate with a fixed crossover probability at 0.5. The high crossover rate is based on the small number of candidates in each population. Would the crossover probability be significantly lower, a solution would still be reached, but take a high number of runs, which might go beyond the attention span of any human user. The chosen crossover rate is based on several tests conducted to establish a balance between keeping the average number of runs low and finding an acceptable solution. The user controlled mutation probability is inversely proportional to the slider setting, which means the higher the confidence, the lower the mutation rate.

3.2.3.2 Procedural Cities

The first important decision that led to the underlying idea of combining Interactive Evolutionary Computation and Agents for this study, was to computationally generate a model of a city. The reasoning behind this shift from manual laborious modelling to significant computational support, is the complexity and number of parameters that are necessary to create such model. While Parish and Müller (2001) point out, that the creation of systems of high visual complexity is an established process in computer graphics, it still requires a very high level of skill, knowledge and consumes a lot of time. Even breaking the overall model down into smaller units such as buildings, streets and layout, does not lead to a significant simplification or is less demanding towards the computer graphics expertise of the designer. Using computational approaches exclusively, pose the same problem on the software developer. Many different approaches have been taken to combine smaller, simple elements into a large system. These include L-systems for plant generation (Prusinkiewicz & Lindenmayer, 1996), dress design (Cho, 2002), level generation for jump and run games (Sorenson & Pasquier, 2010) and the before mentioned L-system based *city engine* by Parish and Müller (2001). But none of these processes close the gap between highly complex systems and the inclusion of the designer in the creation process. This is where the novelty of this solution lies. It enables a moderately skilled designer to create a large model of high complexity without compromising on the aesthetic demands to achieve said complexity.

While it seems that it is principally possible to use the proposed system of IGA and Agents for nearly any computer generated asset for games, film or virtual reality, a city has intentionally been selected as an example of high complexity and composed of many individual parts. Another approach would be the creation of a game or virtual reality character, which has a similar high number of possible parameters, but poses a different complexity, in that characters are not made of many small individual building blocks, but have to be organic and seamless in order to be believable. For example, a randomly shaped head can only be attached to a differently styled upper body if the goal is to make cartoon characters. For realistic humanoid characters, certain proportions have to be considered in order to produce believable results. Adding the rigging (skeleton for animation) into this consideration, a computer generated character is a very complex object per se. While it was possible to test the proposed system of IGA and agents in two different scenarios, Infographics and Procedural Cities, adding another very complex test case such as fully rigged characters would be beyond the scope of this thesis. Such an attempt would be a suitable subject for further research into the matter of IGA supported by agents, as outlined in section 5.4.

The design and creation of CG cities is a complex undertaking as pointed out in previous sections. By utilizing the experience from the Infographics generation by use of Genetic Algorithms and extrapolating the concept to drive considerably larger sets of parameters for city models, it is possible for a relatively novice DCC software user to produce an involved design.

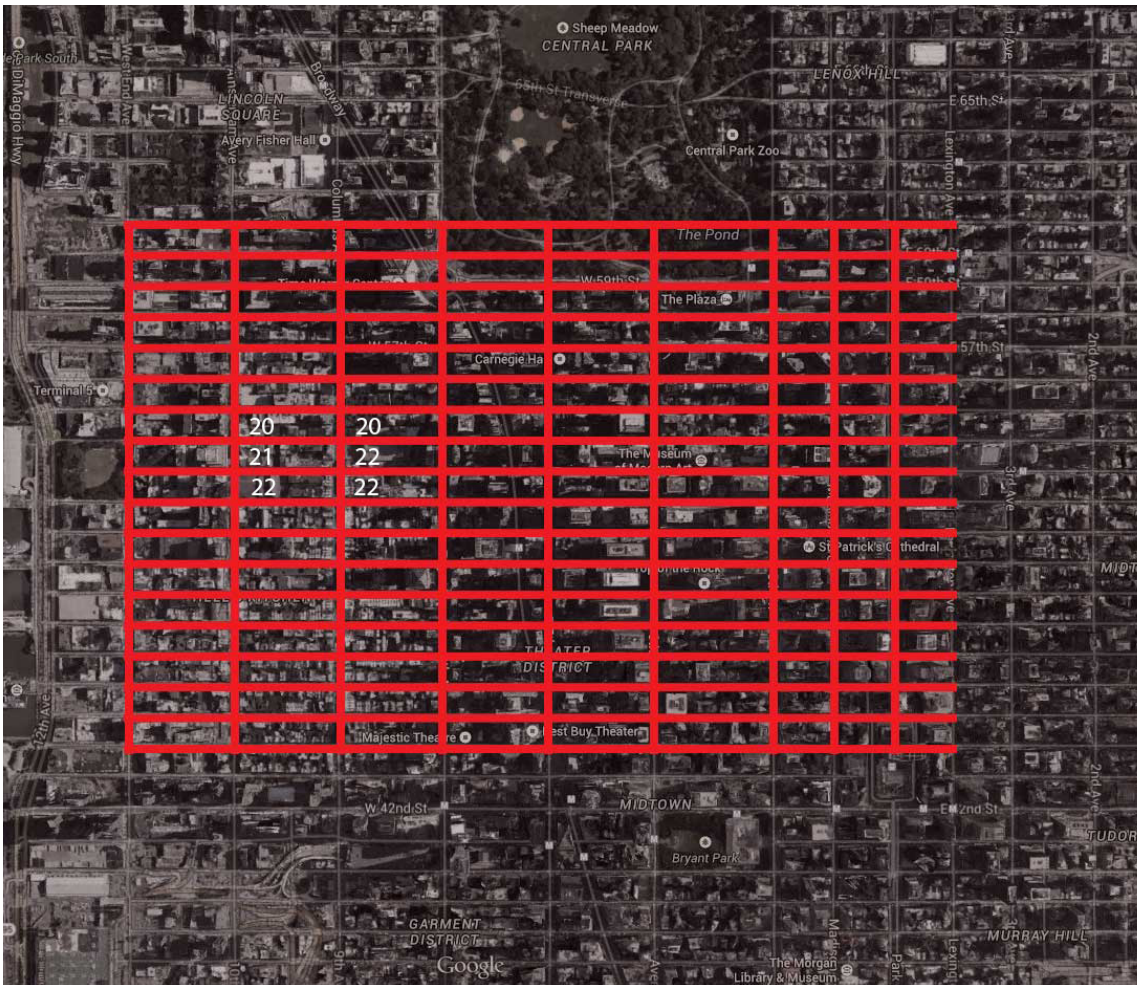


Figure 13 - Manhattan grid structure example including height value examples

Some of the decisions that could be considered limiting to the achievable design, such as using a square matrix to place buildings, separate land and water and drive the height and density of the city centre with its higher buildings, are indeed not limiting at all. For example, considering a map of the Manhattan peninsula of New York City and overlaying a simple square grid (see Figure 13), it is possible to break the complexity of Manhattan down into few small units, for example streets, buildings, water and land. Taking the idea further, the same grid overlay could be used to build a height-map of the buildings on the peninsula by assigning different height values to each grid cell (Figure 14). This is effectively reverse engineering the main spatial and aesthetic features of the city. Further, this approach does not limit the street layout to be square and grid like. Figure 15 is based on an underlying square grid, but shows typical non-square, European like cities, which evolved from a city centre and spread outwards like a web. And this reverse approach is the foundation of the city model as discussed in this study. A simple grid structure is introduced to represent buildings, land and water. A street-map is used to build the network of streets between the buildings on land, and finally an occupancy grid is used to indicate the presence of water.

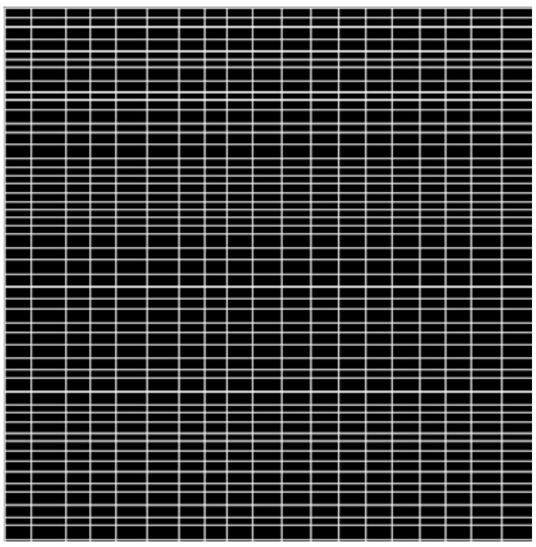


Figure 14 - New York style streets

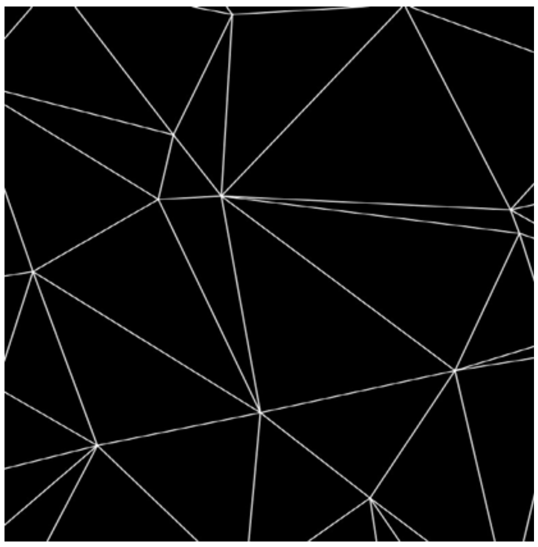


Figure 15 - European style street layout

A height-map is applied to the buildings in the grid, which drives the height of the buildings plus a pre-defined variance so that the elevation does not appear too uniform, but believable and aesthetically similar to what is expected based on looking at actual cities. Further, only simple instances of buildings are used during the design process.

This is for two reasons, one being the very practical realization that creating and rendering times have to be as short as possible to reduce additional user fatigue. Secondly, the instances can easily be replaced with prebuild or purchased models of very different architectural styles. This also allows us to vary between different cities such as North American 'New York' type, European or even Asian building types. The applicability of this approach is therefore much broader than the presented prototype might suggest. Further investigation is of course needed to verify this, but again, the limiting factor is predominantly the scope of this study and not the presented process itself.

While the GUI, which was again written in Java using the Processing framework to provide some graphics functions, is seemingly similar to the Infographic creation, it has the following important differences: There are only two input parameters, which reflect the two main parents of each population of the Genetic Algorithm. And there is no input for the confidence of the user anymore, as the underlying software system is more capable through the addition of the agent plus additional logging of previous selections and rejections, and makes decisions without the need of additional information. The motivation behind this simplification is the desire to reduce user fatigue. Additional inputs would require additional user attention. And as the models, which the user designs, grow in complexity, the risk of extending the required attention and interaction beyond the point of human capacity is rising as well. Minimizing the number of required interactions per iteration seemed the most appropriate way to focus the results of this study on said fatigue and avoiding it by adding agents.

The resulting city model is being kept fairly simple in order to make rendering of 9 candidates for a number of iterations feasible. This is just a limitation of the available hardware and could easily be changed in a commercial environment by using a small render farm. But to manage the rendering times (which are purely a result of the commercial render engine and not the software system presented here), only simple geometric primitives such as cylinders and cubes have been used for the object instances. Also, the texture is very small and simple, due to the limited memory available in the test system. Given that the software system has been designed to use instances of small building blocks instead of copies of existing models, the process of changing from this simple type of geometry to a fully textured, shaded, high polygon model for each house is just the mere adjustment of one configuration parameter. The system to produce feature film quality renders has been established and poses no limitation to the validity of the results. It is just a matter of using high end workstations, which could also be done in future research, assuming that the resources would be accessible.

A typical model, as it is shown to the user for selection, is shown in Figure 16. Some of the defining features include a New York style street system which divides the city into square blocks, some water visible in the far background as well as a distinct city centre on the far left with a number of larger than average buildings surrounding it.

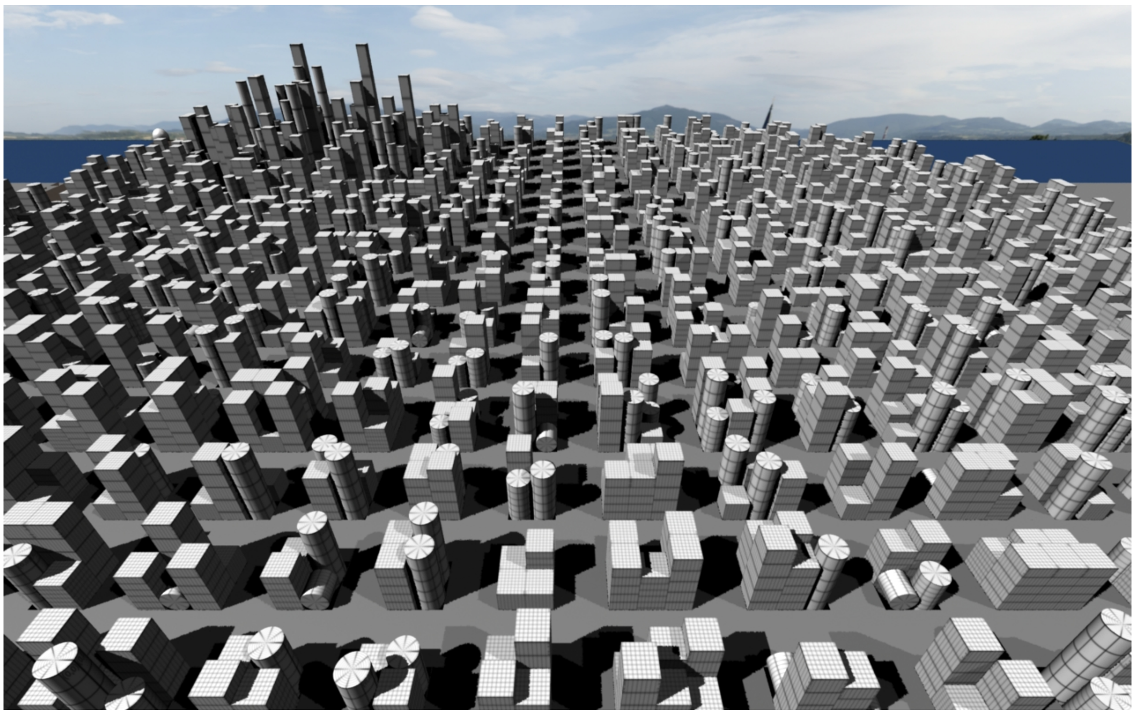


Figure 16 - Candidate showing New York style streets, water in the background, distinct city center

One aspect which resulted from the previous learnings using Interactive Genetic Algorithms to create Infographics, was the necessity to avoid local maxima. In order to keep a balance between fast search, which could potentially lead to the trap of local maxima and having to run a very high number of generations in order to maintain diversity, the agent architecture had to incorporate an indication for the level of confidence to be able to identify user preference. This confidence parameter impacts on the mutation rate of the Genetic Algorithm to keep the number of required iterations as low as possible, while being able to find a valid solution. This is detailed in the following section 3.2.4.

3.2.4 Implemented Agent Architecture

In this study, the agent architecture is built in multiple layers. Its foundation is a form of long term memory, which stores previous choices made by the agent itself. Plus, the memory keeps any user selections as a list of historic events. The second layer is a learning unit, which handles current and historic inputs and outputs by the agent and the user. The learner classifies new generations of candidates based on several available classification algorithms and makes new decisions following these classifications. The decisions made lead to actions, in this case selections of new candidates for future breeding. These selections are fed back into the Genetic Algorithm. While this architecture is relatively straight forward, it is still following the common view of what an agents is: As seen in the literature review, Fogel (2006) as well as Russel and Norvig (1994), all differentiate an agent from any other software by assuming that the agent observes (at least part of) its environment, makes decisions based on these observations and takes actions accordingly. All three of those assumptions are found in the agent architecture used in this research.

The main classifiers used in this study are based on decision trees and naïve Bayes. The reasoning behind this is to evaluate different learning approaches from different classes of learning algorithms. Employing the WEKA machine learning framework allows for a fast switch between different approaches, which provides an insight into the performance of learning algorithms based on very small training sets as used in this study. Given that the aim was to counteract the effects of fatigue by running as few iterations as possible, only a small number of user selections per run were available for training the agent, before the trained classifier had to evaluate a generation of candidates itself.

This study utilizes the C4.5 algorithm (Quinlan, 1986), which was used to induce the decision trees. Specifically, WEKA’s J48 classifier, which is an open source implementation of C4.5 revision 8, created the decision tree at runtime and refined it after each selection made by the user. This is how the initially untrained agent is able observe the actions taken by the user and utilize this information to build the decision tree independent of the goal desired by the user at the start of each run. It adapts every time the whole interactive process is started and does not rely on any assumptions made by the software developer. Every time the user gives new input, the resulting generation of cities is considered a new supervised input. Subsequently, the increasing size of the training set, growing with the iterative selection process by the user, improves the classifier of the learning algorithm due to the increasing number of valid test samples, and therefore the ability of the agent to predict user preference more accurately is improved with every interactive step. This also implies that the user only gives feedback to the agent's actions by making new selections, not through a direct rewards/punishment system. Therefore, the frequency of interactive and computational runs, has to be relatively high at the start, and only when the agent has received a certain number of valid test samples to build the classifier, its involvement is being increased and more computational selections are being conducted by the agent. This ratio between interactive user selection and computational selection by the agent can be set before the start of the interactive modelling process. Nearly all experiments of this study, are based on a run of 10 interactive selections, followed by another 10 selections, where the agent and the user made selections every other time. From the 20th selection run, the agent was responsible for 9 generations, while the user only interacted every 10th time. While this required the user to provide an initially high number of selections, the workload was relatively quickly reduced to a very low number. This pattern was only changed during initial testing, in order to find suitable parameters for this study.

Following the empirical research methods as discussed by Cohen (1995), the agent was first developed to run within a very simple environment, based on a fixed training set, which was previously generated by interactively creating populations and saving them to disk. This training set was used to induce the decision tree, effectively allowing to create the agent's decision unit in a controlled environment. The intent used for the creation of the training set was also predetermined in order to achieve verifiable results:

- A harbor city – implying the presence of water, not just dry land.
- A large population living in the city – implying high risers rather than suburbian one or two level buildings
- A visible business district – requiring a height map with a clearly focused center that leads to very high buildings in a confined area of the city

- A modern city layout – calling for a 'New York style' street layout, rather than an organic web like structure as found in typical European cities.

The agent was evaluated by use of a predefined testing set, again created by controlled user interaction in the same fashion the training set was made. Verifying the agent's behaviour in a number of such scenarios, using varying goals, which had previously been determined, allowed to run the computational agent under controlled conditions. Finally, once the behaviour was explored and confirmed, the agent was put into unknown environments, in this case, run concurrently with user interaction, while creating the decision tree at runtime. The interactive tool provided by the WEKA framework (Hall et al., 2009) was used to evaluate the classifier.

To address the issue of balance between number of required iterations and finding a solution without getting stuck in local maxima of the search space, the agent employs an evaluation that measures the level of confidence in the classifier. WEKA provides functions for this, using the training data set and a test data set. In order to be able to evaluate the classifier at run time without any previous knowledge of the user's preference, a small fraction of user selected populations were added to the test set instead of the training set. The classifier was applied to this test set, as it contains the valid user evaluations, and a score for correctly and incorrectly classifying instances (populations) is given. The higher this score, the higher the confidence of the agent to be able to correctly predict the user's preference. This confidence in turn, drives the mutation probability. A small probability for random mutation is always maintained, independent of the agent's confidence. This allows the user to change direction by selecting a sample that does not fit the previously chosen samples. For instance, if the user had a tendency to prefer a large amount of water surrounding the city centre and suddenly selects a candidate with little or no water areas at all, the sample would deviate from the mean of the parameter space quite significantly. Especially towards the end of the search process, when the set goal is nearly achieved and the agent's confidence is already very high, this diversion would have an impact on the level of confidence of the agent and subsequently on the mutation rate. This in turn, opens the search space up again and allows more random samples to be generated, therefore creating some diversity.

The DNA string used in the Genetic Algorithm is made of the following parameters:

• Ground	2D integer array, reflecting land or water
• Heightmap	2D integer array, height of buildings
• Streets	2D integer array, street map layout
• Buildings	2D integer array, type of building
• City	2D integer array, location of city center(s), a few distinct buildings

One of the necessities for inducing decision trees is to keep the number of attributes as low as possible, while still arriving at a solution in form of a usable classifier (Hall et al., 2009). To accommodate this requirement, the parameters for the agent have been abstracted from the candidates DNA string.

Instead of using each attribute of the DNA, some have been consolidated. For example, the height map is a grid-like structure represented as a 2D array in the DNA, which has been pre-processed into three parameters: Average building height, number of buildings and height of buildings in the city centre (or scale of the buildings compared to the average height in the remainder of the city). This reduces the number of parameters from originally 10,000 cells in the 2D grid, holding the height value for each individual cell, to just 3 attributes, which make induction of a decision possible for the agent.

In summary, the attributes for the agent are:

- Land/Water ratio
- Street type (European or New York style)
- Average building height
- Number of buildings
- Height of city centre

Additionally, the agent receives the class value, which indicates whether this instance of the training set has been selected or rejected. Finding the correct class value in new instances is ultimately the task of the agent.

3.3 Experimental Process

Using small, simple elements to create a larger, more complex structure is part of this research. Our design process will be driven by an algorithm and enable the designer to come up with new forms and shapes. The user will be in a position where a set goal of designing a complex three dimensional city, can be achieved without manually constructing the elements or overall structure. It is therefore important to understand the differences between manual Computer Aided Design and Generative Design as pointed out above.

This section of the study discusses the experimental practice and the empirical evidence gathered throughout the process. This includes testing the software prototypes and their performance. The performance has been measured using different stages of the software prototypes, for instance using simple random selection without the Genetic Algorithm, as well as performance using the Genetic Algorithm driven by the user.

3.3.1 Infographics

The performance of the Interactive Genetic Algorithm was measured using predefined goals to ensure consistency for each test. The selected goals were very different, some resembling familiar map designs like blue water, green land, others intentionally diverting from the convention. Some examples were also worded relatively precisely using specific colour or shape instructions, allowing for only narrow interpretation, while others were more abstract using attributes such as dark, bright, contrast. Examples of such predefined goals include:

- Blue water, green land mass, red markers, pin needle marker shape
- Blue water, green land mass, red markers, pyramid marker shape
- Black water, yellow land, blue marker, pin needle marker shape
- Dark water, bright land, high contrast marker, sharp marker shape

An exemplary run can be broken down as follows (Figure 17):

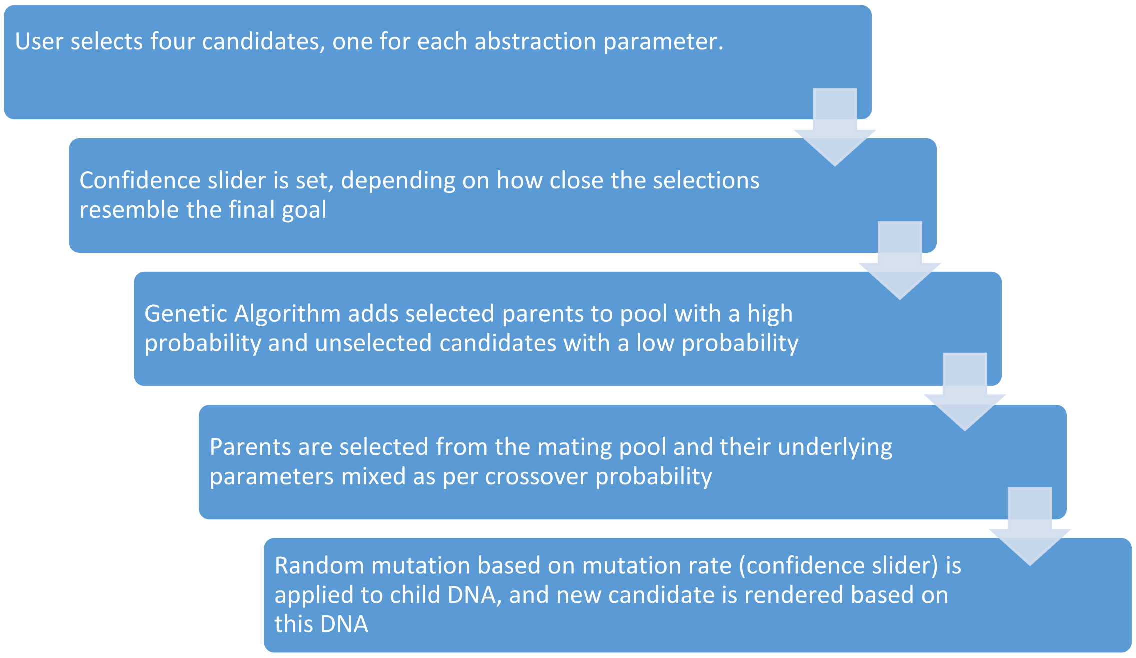

User selects four candidates, one for each abstraction parameter.
Confidence slider is set, depending on how close the selections resemble the final goal
Genetic Algorithm adds selected parents to pool with a high probability and unselected candidates with a low probability
Parents are selected from the mating pool and their underlying parameters mixed as per crossover probability
Random mutation based on mutation rate (confidence slider) is applied to child DNA, and new candidate is rendered based on this DNA

Figure 17 - Infographics process

All runs were conducted several times to increase the number of samples collected. And in order to generate a control for the Interactive Genetic Algorithm, a second run based on the same prerequisites was conducted, using just a random map generation instead of the support of the evolutionary process. The times for each run were noted as well, as they provide an indication for the attention span required and therefore an indication of possible fatigue.

3.3.2 Procedural City

Following the experiences from the Infographics experiments, some of the learnings have been incorporated into the experimental process of evolving procedural city models. The overall experimental process for procedural cities involves three different runs: Random, IGA and HBGA, but not necessarily in this order so that the user does not attempt the design with a preoccupied idea of how the software might perform. For all three variants, the user interaction is exactly the same, so that the underlying software system is fully transparent to the user. The differences are detailed in the following subsections.

In any case, each iteration of the user interaction starts with a set of 9 rendered city models. These are created by the software system depending on the underlying model, for example random selection, interactive genetic algorithm without agent and interactive genetic algorithm with agent. The user selects two parents with the mouse (both show a coloured border around the selection to give visual feedback) and starts the algorithm with a press of the *space* button. Everything else such as log information, RIB file creation and sub-processes for the render engine are hidden from the user. This keeps the user interface as simple as possible, with the idea to utilize maximum user focus for the actual task of selection.

After computation, including creating the new population of 9 cities, writing the RIB archives and rendering the resulting images, the user is presented with the next generation ready for selection.

Each individual approach was conducted based on a predefined set of goals, similar to the experimental process of the Infographics. Some of the requests that where made include relatively specific elements such as:

- A city with a lot of blue water, a city centre with very high buildings on the left and some smaller buildings on the right hand side of frame
- A city without water, multiple city centres with few or no small buildings visible in frame
- A city without water, no distinct city centre and only very small buildings

Other requests were kept abstract using high level description of the desired features. Again, this was designed so that the difference between a tight goal driven, perhaps client based approach and in difference a free creative design, independent of any strong prerequisites and maybe just loosely based on a design idea could be simulated. The latter examples include:

- A harbour city with a large population
- A city in a valley with a suburban feel
- A city by the sea with a lot of tourism

Similar to the experimental process with Infographics and Genetic Algorithms, a number of results were recorded for each different approach. These findings include the run times for the overall process from start to finding a result that the designer deemed final, the number of iterations required to get to the final result, and also the subjective feeling after performing a full run. While the latter is not necessarily representative for the quality of the algorithm, with regards to user fatigue, it might provide an idea whether the software system is successfully reducing the workload on the user.

3.3.2.1 Random Selection

While the testing of random candidate generation as a control was still feasible for the above mentioned Infographics, it turned out to be impractical for more complex issues, in this case city models. Preliminary testing lead to the conclusion that the large number of parameters and the long render times may not find any solution within hundreds or thousands of generations. But in order to provide at least some control measure for this study, the following approach has been taken, mainly to make the random selection a bit more practical.

Instead of asking the user to select parents, which are not used for random creation anyway, or watch every generation being rendered, which takes a considerable amount of time, with little chance of an actual outcome, a different workflow was used. The computer generated 100 random generations, which were batch processed overnight. The resulting images were numbered and sorted, so that any potential candidate could be identified. The user had to go through the 900 images to see if there was any solution available, and if so, the population that contained the accepted candidate could be noted. To put this in perspective, the longer Interactive runs finished after 56 generations (or 504 images) that the user selected from.

3.3.2.2 Interactive Genetic Algorithm

The Interactive Genetic Algorithm without the support of a computational agent was the first type of experiment, which employed the canonical Genetic Algorithm along with a human designer for aesthetic selection. Two parents are selected out of 9 candidates, which are presented as rendered images. These two parents enter the gene pool with the highest probability for breeding. Additionally, the rejected 7 candidates are entered with a very low probability of being used. Recombination is achieved by single point crossover at a random point of the DNA string. Mutation of parameters takes place based on a low probability. The resulting candidates are saved as RIB archives and added to a log system for future examination. Finally, the RIB archives are rendered using 3Delight (“Welcome | 3delight,” 2014) and presented to the user for the next evaluation. While this process is an interactive variant of a purely computational Genetic Algorithm, it resembles the idea of simple evolutionary computation very closely, in order to provide a control for Human-based Genetic Algorithm as discussed in the following section.

3.3.2.3 Human-based Genetic Algorithm

The Human-based Genetic Algorithm works similar to the approach detailed in the literature review (Kosorukoff, 2001). An Interactive Genetic Algorithm is the centre of this approach, with either a human agent or a computational agent being responsible for the selection, while the recombination and organizational components are always fully computational in this case. The user is presented with a generation of 9 different cities. The first generation is always randomly generated, as there are no previous selections, which the recombination could be based on. Next, the human-based selection is performed as detailed in section 3.3.2. The selected pair of cities is passed to the recombination algorithm, which takes both parents, creates the breading pool based on the fixed crossover probability and performs a single point crossover to create 9 new city DNAs. Mutation of the DNA strings is driven by the mutation rate, which is initially set to 0.02 based on previous experiments and based on the relatively large number of parameters encoded in the DNA string. This rate changes in later generations as soon as the agents returns a higher confidence evaluation based on a larger number of training sets (user rated city generations). The RIB archives are then created using the new 9 DNA strings. Finally, after rendering each RIB archive into a fully shaded, fully textured image, the new generation is shown to the user and the next iteration cycle begins.

This process, exclusively based on the selection feedback of the human user is performed for 10 iterations. The number of iterations required from the user before the agent starts to help is supervised by the organizational component of the Human-based Genetic Algorithm and can be changed based on experimental experience. This set of 10 human user iterations is the initial training set for the agent’s classifier. From generation 11 onwards, the user and the agent take turns running the selection. The reasoning behind this is to keep providing suitable training sets (by getting user input), which refines the initial training set of 10. Further, this allows the agent to start running iterations and therefore reduce the effort required by the human user. This approach also allows the agent to evaluate the quality of its classifier and provide a confidence vote, which ultimately leads to a reduction of the mutation rate. Finally, reaching iteration number 20, the agent runs 9 iterations, while the human user just interacts every 10 generations. As mentioned before, these thresholds for agent selection can be changed, if required. For example, the human user could continue to run every second generation in order to provide a larger training set for the agent. Or, the process could increase incrementally, with the agent running 2 out of 3 generations and so on. The thresholds chosen for this study were simply based on empirical observation and found to be effective. A discussion of this can be found in section 4 of this study.

3.3.2.4 Testing and Performance Measurements

In order to facilitate testing and gather information about the process of Interactive Evolutionary Computation with and without agent support, a few systems have been established at different points of the evaluation prototypes. First, every generation is logged as a RIB archive in human readable ASCII format to enable the researcher to examine the underlying DNA of each candidate in every generation. Also, the rendered images of each generation are saved, which allows for a reasonably fast examination compared to the genotype (the raw DNA data), as the images – the phenotype of the candidates - are visual representations of the DNA and naturally serve as a visual aid for comparisons. Further, different environmental parameters of each run are documented, such as run time, number of iterations, time per iteration, recombination parameters including crossover and mutation probabilities, as well as goal parameters and final candidates. The intent was to capture as much information as possible to identify possible signs of fatigue while understanding the conditions that might have led to it. With regards to the agents, debugging is slightly more difficult due to the autonomous nature of the agents architecture and its evolving decision making process, which is not predetermined, but adjusted with every human input. According to Wooldridge (2009) and others such as Russell and Norvig (2003), autonomous agents are notoriously difficult to debug. In an attempt to resolve this issue for this research, the classifiers as they were evolving were written out to disk as model files for WEKA (Hall et al., 2009). This allowed for offline examination of the decision trees in the interactive tools provided by the WEKA framework. The results are discussed in the following two sections of this thesis.

4 Results and Observations

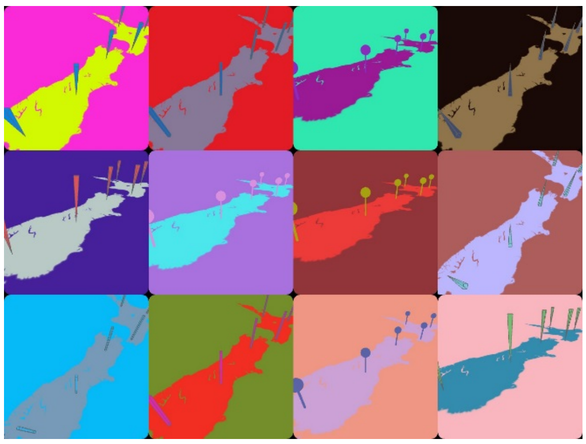
4.1 Introduction

This section presents the findings of the study. Empirical evidence as well as qualitative results are described and analysed in preparation for the discussion of the outcomes and final conclusions in section 5.

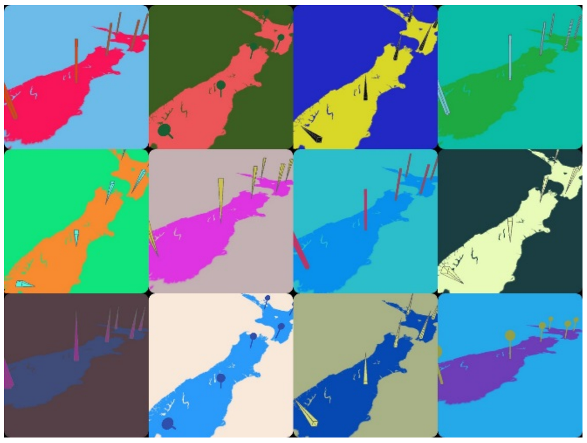
4.2 Infographics

A number of different tests were conducted using conditions defined prior to executing each individual run, as detailed above in section 3.3.1. Every run was done twice, executing two different algorithms. First using just a random map generation algorithm without any evolutionary computation. This provided a control for this experiment. The second run was done running the interactive genetic algorithm. Both situations were then repeated multiple times in order to gather a larger, more representative number of samples.

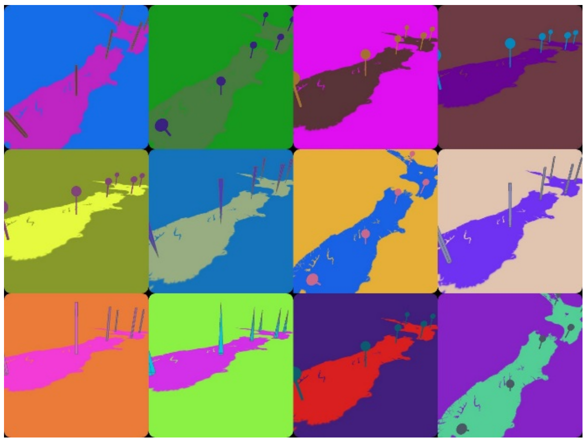
A fragment from an exemplary random run, with the goal of creating a blue water, green land, red pin needle marker infographic, looks as follows:



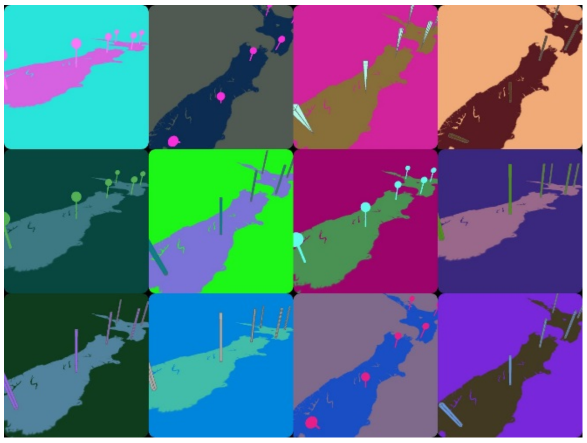
The first random population shows no valid candidates. While there are arguably two candidates with blue water visible, none of them has green land, nor red pin needle markers.



The second random population shows no valid candidates. There are a few candidates with blue water and one with green land, but none in combination, nor red pin needle markers at all.



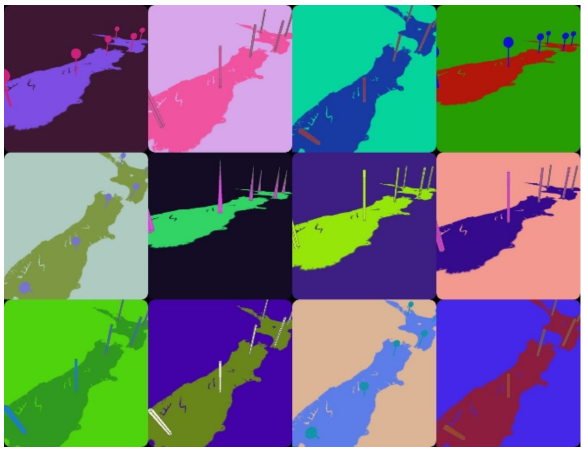
The third random population example shows no valid candidates. There are a few candidates with blue water and green land, but none in combination, nor red pin needle markers at all. The only candidate that is somewhat close to the brief is the second in the middle row, with blue water, pale green land, but spikey, triangular markers in a different colour.



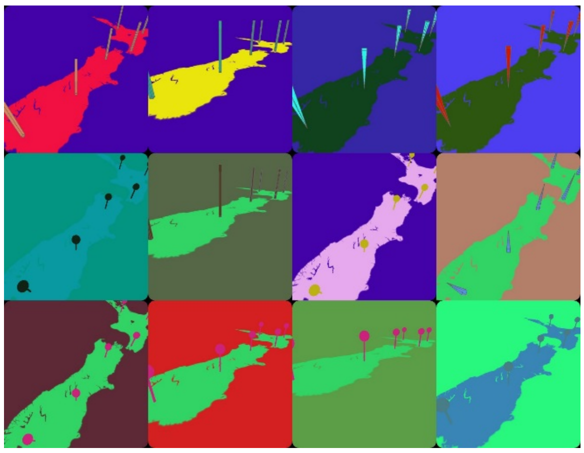
The fourth random population example shows individual parameters again, but no combination as requested. The second candidate in the last row has blue water and green land, but the wrong marker style and colour. The following, third candidate in the last row shows the right marker style and colour, but not the right land nor water.

These few examples are representative of the majority of candidates with randomly generated parameters. Although the number of parameters is relatively small compared to the case study of procedural cities, a vast number of possible random populations can be created without reaching a viable solution. It is important to consider that colour still leaves room for interpretation and therefore any candidate that comes close to the brief might be considered valid. For example, while the brief for the above random run was not 'pale green land', the level of saturation (colour intensity) is subject to individual interpretation. Any given problem that has a more specific aesthetic parameter than colour, for instance 'pin needle', removes this room for subjectivity and therefore requires a very specific solution to be presented: If the final result does not contain pin needle markers, but triangular markers, it would have to be dismissed.

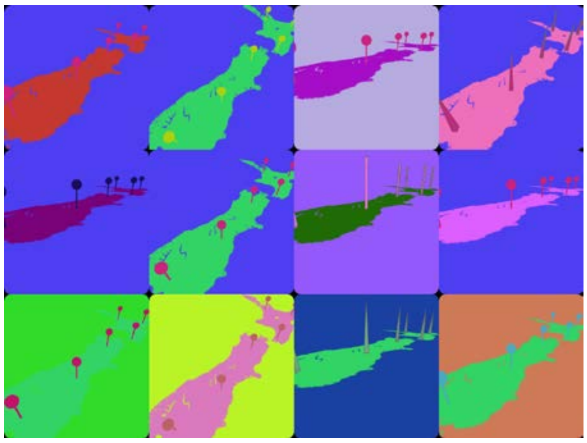
The following table shows a full run using Interactive Genetic Algorithms. The run finished after 5 iterations, following the initial random generation with a range of valid candidates. The brief was the same as for the above random run.



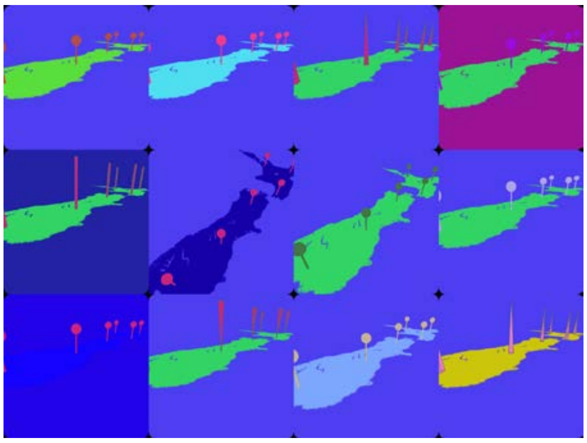
The first random population shows no valid candidates. A selection of four candidates for water colour, land colour, marker style and marker colour has been made, including the candidate top left, second middle, bottom right. All of the candidates show some of the desired features such as blue water, green land, red pin needle marker.



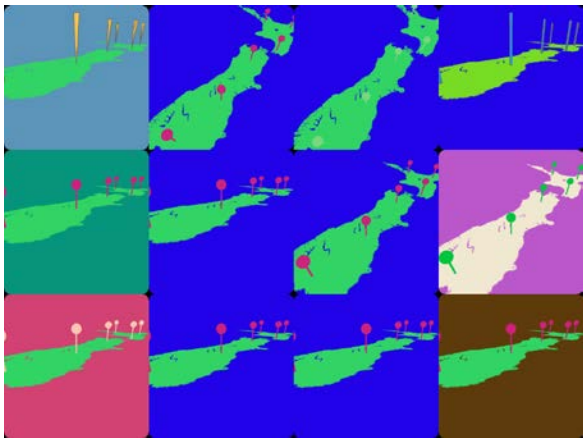
The second population shows no valid candidates. There are more candidates with blue water and green land, and two with red markers. The two top right candidates show promising colour combinations.



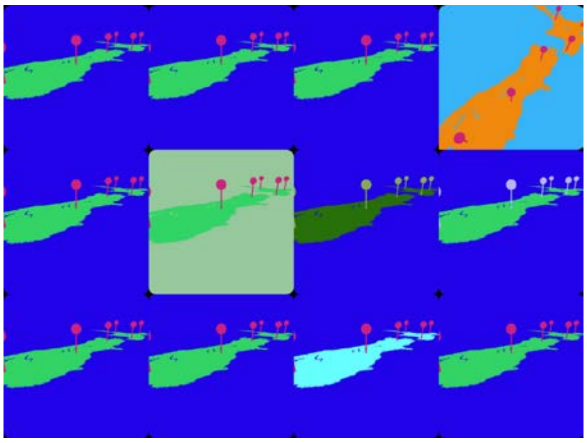
This population starts to show more of the desired colours, while the water is still purple and the land very bright green. Several examples have pin needle markers that are of red colour, while the red is still fairly magenta.



In this example, the water colour converges towards blue in many candidates. The left bottom shows a more desirable saturation and hue (less red), while the land colour in this particular candidate is off target (blue).Also the number of red pin needle markers has not significantly improved. At this stage the mutation rate was set lower in order to produce less variety and instead keep more of the features such as blue water.



This population shows a lot more blue water of the right colour, several examples of green land with red pin needle markers. While the run could have concluded at this stage, it was decided to run again in order to produce a more distinct red marker colour. The mutation rate was very low at this stage to avoid any major diversion from the very close to finished results.



This is the final generation of this run. Several nearly identical images indicate, that the IGA has converged towards these solutions. The final candidate as selected by the user is at the bottom right.

The following table (Figure 18) shows the summary of 30 experiments conducted with the average number of runs and times for random and interactive evolutionary runs:

	Random (# of generations)	Time (s)		IGA (# of generations)	Time (s)
Average	53	154		7	43
Random/IGA	100 %	100 %		13.2 %	27.9 %

Figure 18 - Summary of experiments

The comparison shows significant improvements using an Interactive Genetic Algorithm with variable mutation probability over simple random runs. The number of runs required is just more than 1/8th and while the IGA runs take much longer than the random ones, with the latter merely requiring the click of a button, looking at all 12 candidates and deciding whether the goal has been (coincidentally) achieved, the overall time savings are still quite significant (less than 1/3rd of time required for IGA runs). But the statistics, and in particular the times needed, only reflect part of the advantages of IGA over random generation. The user has to make a decision every time new candidates are presented. This requires concentration. And running an average of 53 generations as opposed to an average of 7 could be more tiring based on the render times and the user interactions necessary, even though the overall time needed is 27.9 % and not just 13.2%. While this is a subjective impression, it seems to play a role in user fatigue. The willingness to attempt another IGA run was higher than compared to random runs. The more engaging interactive evolutionary process seemed to be a factor in this as well. The soft factors mentioned can neither be confirmed nor dismissed in this study, and it could be interesting to look deeper into these psychological issues in future research.

Whenever the variety of Infographics did not include any of the desired properties for a number of runs, the user set the confidence slider lower, therefore increasing the mutation probability and subsequently the diversity of candidates in each population as discussed in section 5.2.

One interesting property of the system is the ability to change course fairly quickly, due to the variable mutation rate. For example, one run started with the goal to produce blue water, green land, red pin needle markers and a flat camera angle. After 5 runs, when the results were starting to converge towards the originally intended design, a new goal was set, without resetting the algorithm. The changed intent aimed to deliver a dark water, yellow land, red pin needle markers and a flat camera angle. The idea was to simulate a change of mind of the user or perhaps a client of the designer in the middle of the design process. The generation at that point had no candidate that came close to the new brief. Without starting over, the user simply selected candidates that remotely resembled the idea of the new goal such as slightly darker water, slightly more green-yellow land, red markers and a flat camera angle. The first few generations produced no useful outcome and the confidence slider was set much lower again. After about 6 additional generations the algorithm started to converge towards the new design goal again. The whole process took 17 generations to produce the desired result, including the originally ‘misleading’ generations towards the first, rejected design goal. This is a remarkable reduction compared to the random process, and while starting over with a fresh set of candidates could have been more effective based on the average number of 7 generations, it demonstrates the adaptability of the Interactive Genetic Algorithm and its potential to break out of maxima in the fitness function and find another solution relatively quickly. This is an important feature in creative and interactive processes. While following an originally conceptualized outcome, ideas or requirements might change and the designer has to be able to adapt his practice and the software tools used should not be a limiting factor. A similar case, where the software system supports the designer through added intelligence, involving the support of the computational agent is shown in section 4.3.3 in context of the Human-based Genetic Algorithm.

Using this rather simple design example with an intentionally reduced parameter set and a simplified user interface, the validity of Interactive Evolutionary Computation in conjunction with design problems in virtual worlds has been successfully confirmed. The principles of GUI simplicity and abstraction of parameters has been translated to the core of the software system of this research. Its application to a more complex issue with a significantly larger set of parameters has been successfully demonstrated as discussed in section 4.3 of this thesis.

4.3 Procedural City

As detailed in previous sections, the experimental process involved three different runs: Random selection, Interactive Genetic Algorithm without agent and Human-based Genetic Algorithm including a computational agent. These were conducted by the researcher and supervisor, and more extensive testing with a larger group of participants has been suggested as part of future work (see section 5.3).

The average computational process including rendering takes just under 9 seconds on average, when running on a high-end windows workstation (Intel Core i7 – 4770k, 16 GB RAM, NVidia GTX780, 2x SSD). The main portion of this time is devoted to rendering the images using 3Delight (running 4 cores, 8 threads), while less than a second is consumed by the generation of new populations, writing RIB archives and running the agent. It can therefore be assumed that the computation time is very similar for all three different scenarios as discussed in the following subsections. As a result, the influence of computation time of any of the three different approaches is negligible, and any fatigue of the user can most likely be linked to the number of required iterations and therefore the overall process, rather than differences in the computation times of each individual iteration.

4.3.1 Random Selection

An exception to the experimental process was the random selection: Initial testing did not show any promising candidates, even after a larger number of runs. In a few rare cases though, a random sample early on in the process could have been accepted under the assumption that the brief was not taken too rigorously, but none of the candidates resembled the previously stated goal for that run exactly. But the waiting times for each generation are relatively high, with only a slim chance of randomly striking an acceptable solution. Therefore, as outlined in section 3.3.2.1, involving the human into the full process, just to create a random control was deemed to be impractical. Instead, batch processing was realized and the user went through the results only. While this approach did not measure the fatigue generated by the actual (redundant) selection plus render times for every generation, having to go through 900 pictures proved to be very tiring too.

Two runs were conducted, and the experiment was not continued for the initially envisioned 10 runs. The time it took to evaluate the images seemed to justify the conclusion that random selection does not necessarily lead to a result within a practical time frame.

While it could be argued that manual creation takes much longer, this study's focus is on computational solutions, and the run times in the following sections demonstrate the difference between a manual / random and a computational approach.

One interesting finding is the relevance of the design goal though. When looking for a very specific outcome, for example a large city by the water with many high rise buildings on the left of screen and some flat buildings on the right, it was clear after 100 generations (or 900 images) that no solution had been found. But when the goal was set in a more abstract way, without any specific requirements, for instance large harbour city, some of the candidates seemed to fulfil that brief at least very loosely. But in saying this, it seems necessary that the set goal has to be specific enough, so that any broad interpretation, which might include many very different solutions, is avoided. A loose goal setting would not provide any contestable results with regards to the specific research question of this study. This research is concerned with the difference between pure Interactive Evolutionary Computation and Human-based Genetic Algorithms to address fatigue.

Further, it can be said that the task of looking through a large number of random samples seems daunting, despite the chance that there may be a viable solution in the random set. This finding leads to another question: Does the presence of a computational component such as evolutionary computation or an autonomous agent engage the user more, just by providing the impression of support and the identification of the user as part of a 'team' of human and computational agents? This area of research is located outside of the scope of this study as it investigates psychological aspects of user experience rather than mental or physical exhaustion in form of fatigue. But looking at the subjective perception of the presence of artificial intelligence during the capture of data might provide some interesting insights on a different level, and give some clues whether an agent needs to be successfully making decisions at all or if its presence alone improves the user experience, which in turn might compensate some of the effects of fatigue.

4.3.2 Interactive Genetic Algorithm

The following images shows a few examples of a typical run using Interactive Genetic Algorithms without the support of a computational agent. The predetermined goal was very specific, stating that a city with a lot of blue water, a distinct city center on the left and much smaller buildings on the right was to be created.

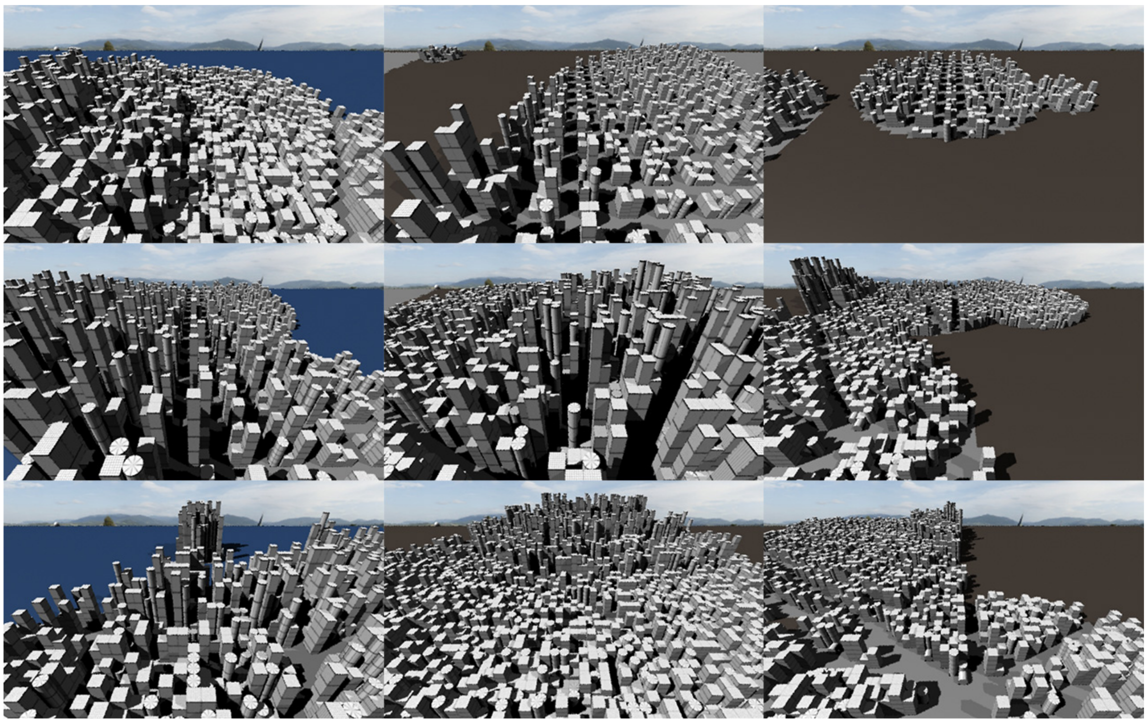


Figure 19 - Generation 1 of an IGA run

Figure 19 is the first generation of the example. Typically, the first generation contains a number of mixed results, here with 6 of 9 candidates containing just land and no water. Consequently, the user selected candidates top/left and bottom/left.

As a result of mainly choosing candidates that had water present, the user improved the number of available options in subsequent generations. The next image (Figure 20) shows generation 10, which has significantly more cities with water visible.

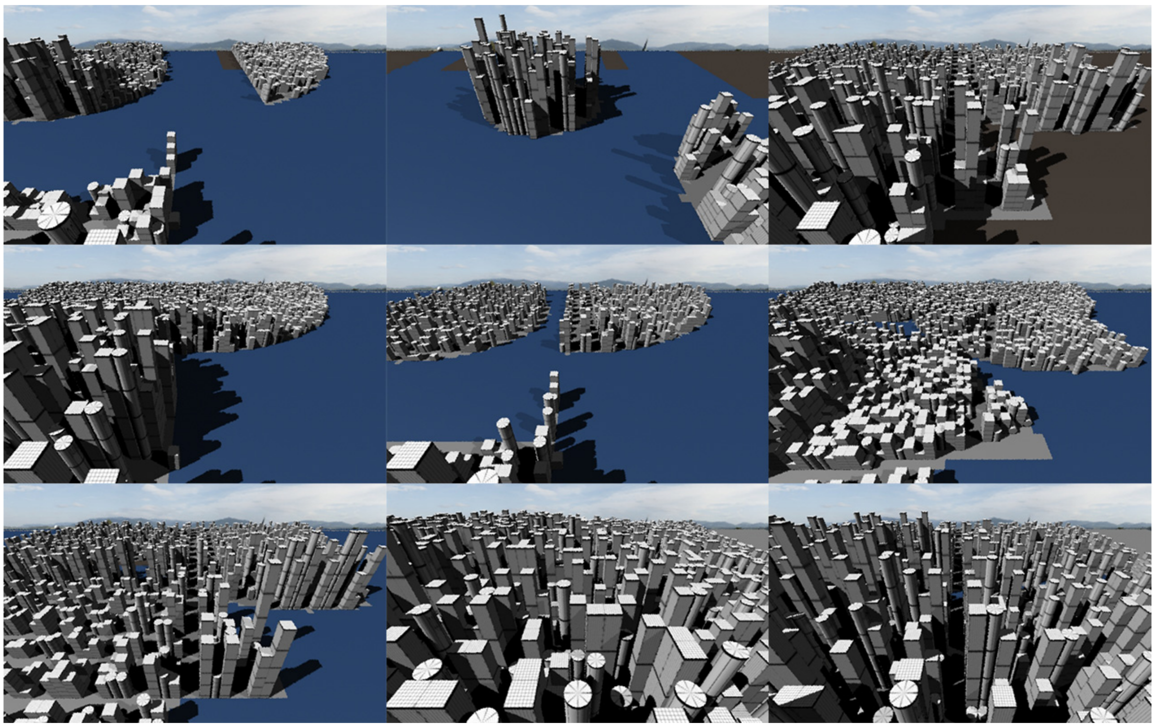


Figure 20 - Generation 10 of a typical IGA run

Another 10 generations later (Figure 21), a couple of candidates with water, also start developing high buildings on the left and somewhat smaller buildings on the right hand side.

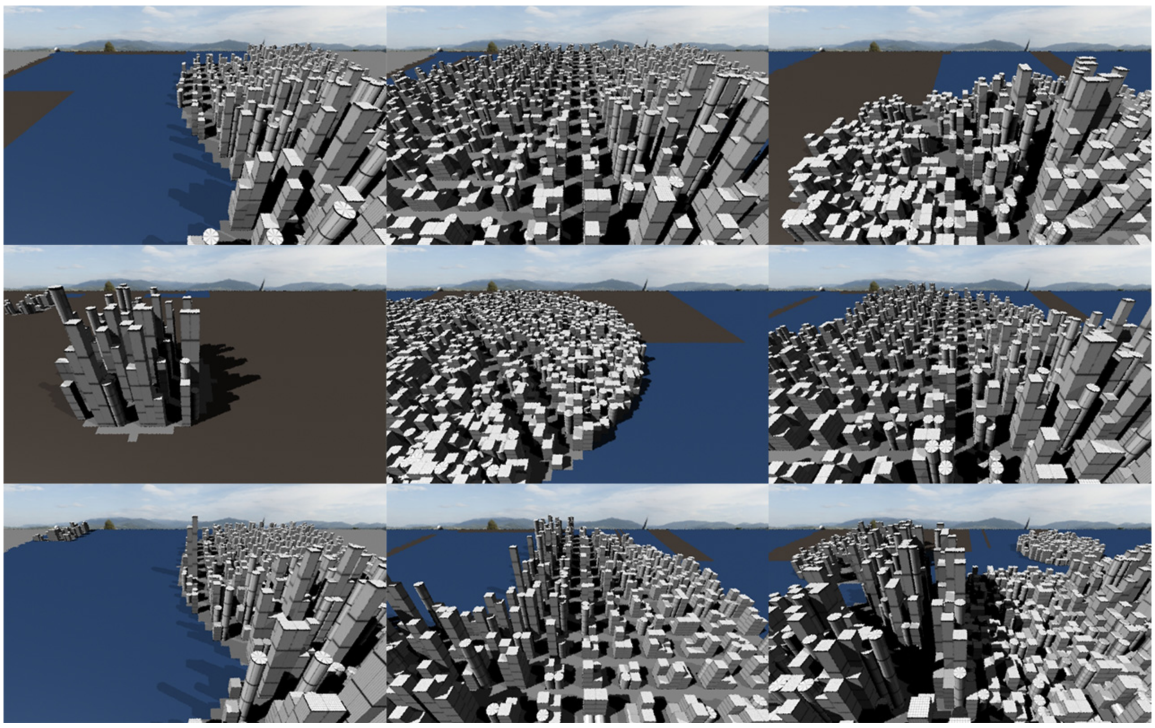


Figure 21 - Generation 20 of IGA

In generation 30 of this exemplary IGA run (Figure 22), a few issues can be observed, which are based on the slow convergence of the Genetic Algorithm.

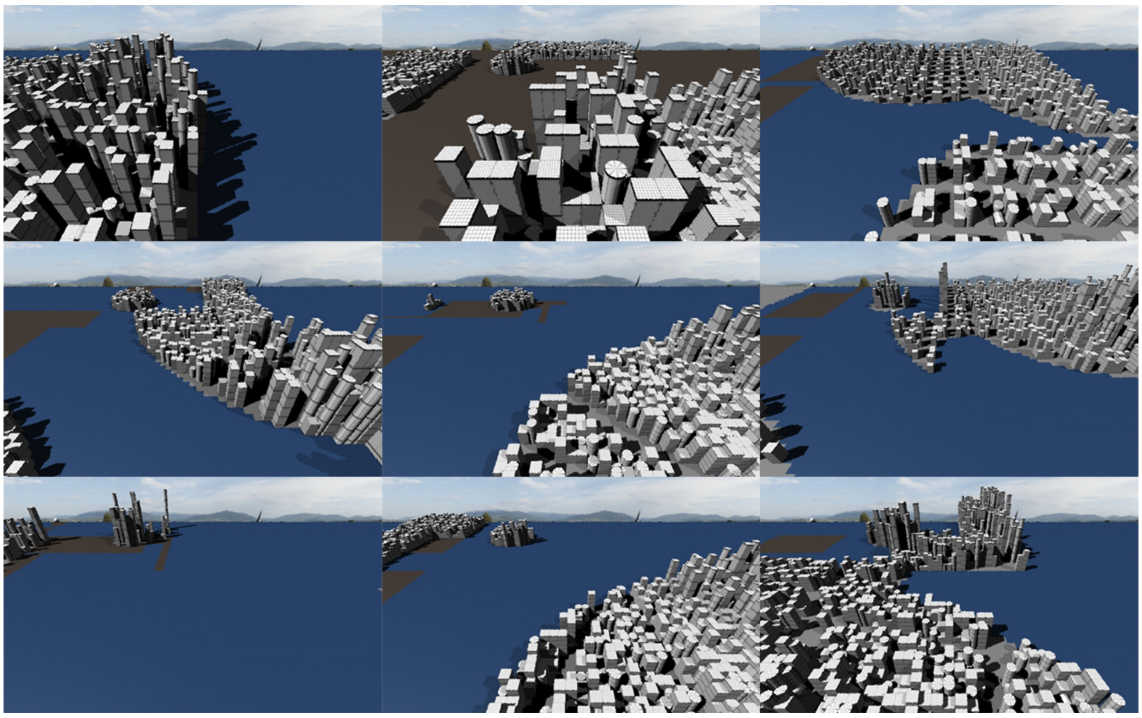


Figure 22 - Generation 30 of exemplary IGA run

While all candidates, except number 2, contain water, and candidate 1 has high buildings on the left side, the majority of candidates had the buildings on the left. But none of the candidates presented the required properties as outlined before the run started. Accordingly, another 10 generations were required in order to achieve the first promising results, as depicted in Figure 23.

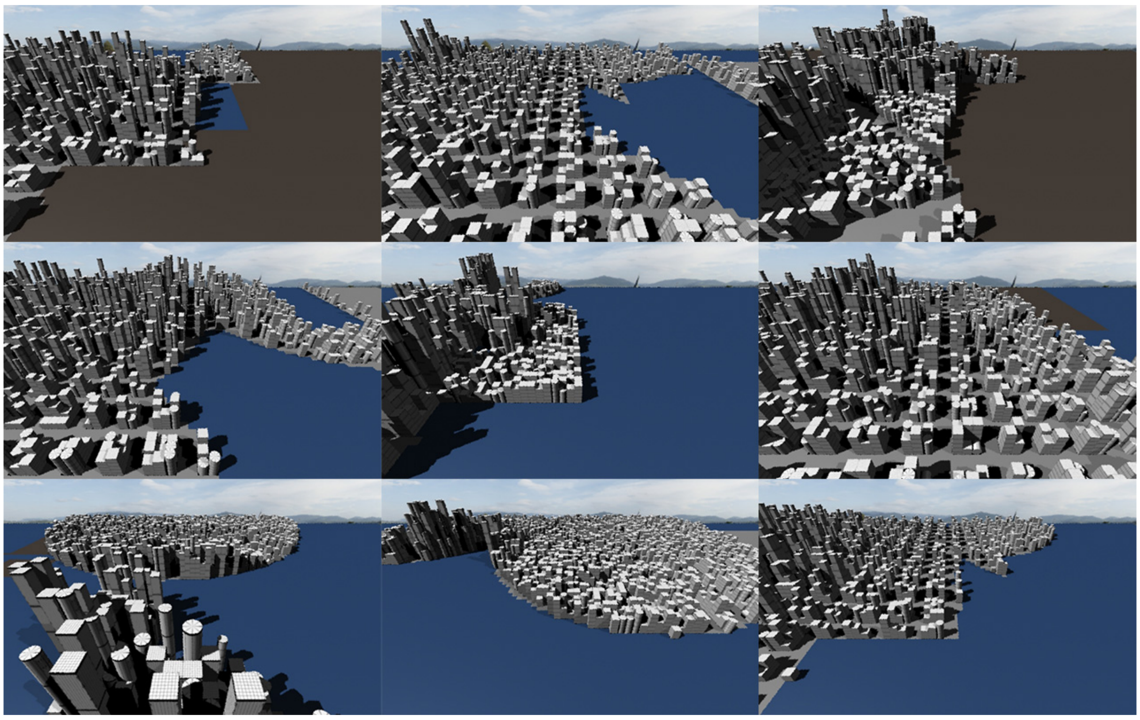


Figure 23 - Generation 40 of IGA run

After 7 additional iterations, the final candidate was found. Figure 24 shows the originally requested water, a distinct centre with high buildings on the left and low-rise buildings on the right side of frame. While the result satisfied all criteria of the brief, it required 47 generations to achieve it.

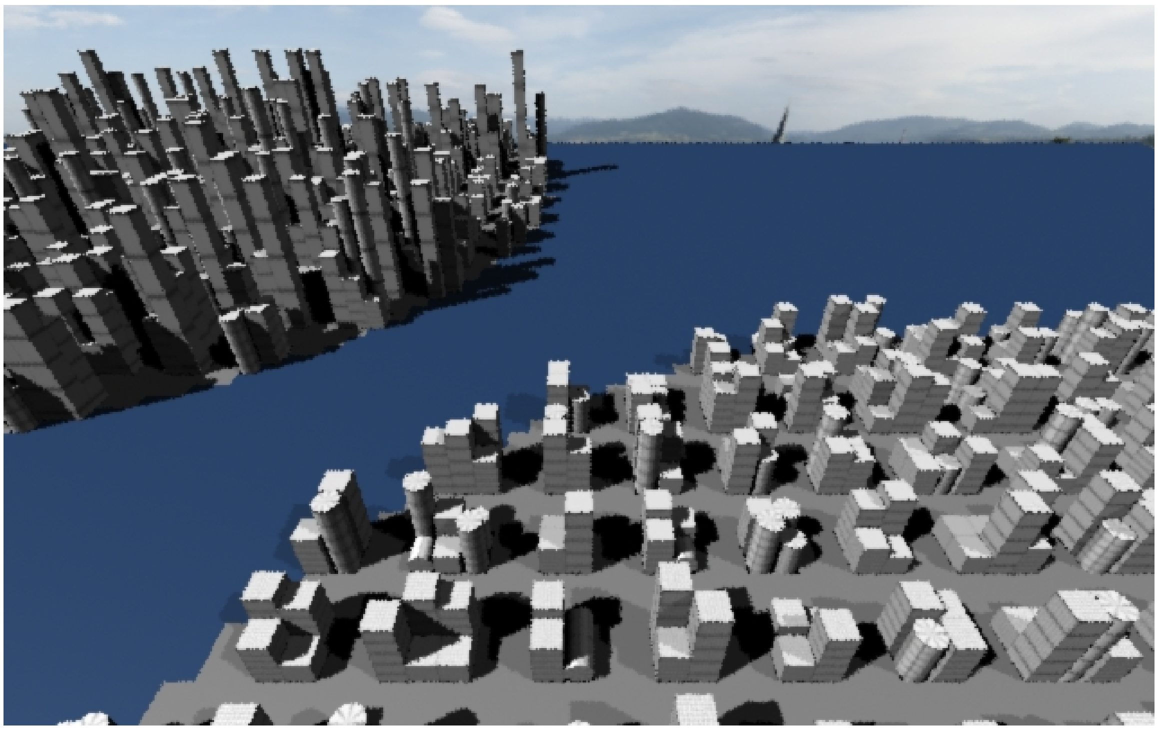


Figure 24 - Final candidate of IGA run

Overall, the Interactive Genetic Algorithm without an agent was tested in 36 runs with an average of 37 generations per run. The standard deviation was 11 generations, which shows a fairly wide spread. The smallest run was only 1 generation, although this was deemed a special case and likely due to the subjective nature of aesthetics combined with an element of 'luck' arising from the stochastic nature of the algorithms. After finding a suitable candidate right in the first generation, another few iterations were conducted, which showed even more promising results. The fact that the initial conclusion in generation 1, was later reviewed and seemingly better solutions were found in subsequent generations, underlines the implications of judging aesthetics based on a high level project brief (which was given for this particular run). A more detailed brief would probably have led to more iterations in the first place.

The average time per run was 18 minutes. Figure 25 shows an overview of the statistics of all runs conducted using Interactive Genetic Algorithms compared to Human-based Genetic Algorithms. For each run, the same brief was given based on examples shown in section 3.3.2. Results of both IGA and HBGA are being discussed side by side in the following section.

4.3.3 Human-based Genetic Algorithm

The Human-based Genetic Algorithm was tested in 36 runs, similar to the Interactive Genetic Algorithm and with the same predetermined goals, as discussed in the previous section. The average number of generations was 52 with a standard deviation was 14. The maximum was 91 generations and the minimum 21. The average time per run was 12 minutes. It is noteworthy that the total number of iterations is higher compared to IGA, but this higher number includes both interactive and computational generations conducted by the human designer and the autonomous agent respectively. The mean number of generations using HBGA that required user interaction is 18 and therefore much lower compared 37 using IGA.

	# runs	# generations (interactive)	# generations (total)	Time (average)
IGA	36	37	37	18
HBGA	36	18	52	12

Figure 25 - Comparison IGA versus HBGA

The higher number of generations in total indicates that a greater number of possible solutions is being explored, though a smaller number are being evaluated by the user in person.

Different initial mutation rates for the Genetic Algorithm were verified, with the majority of runs conducted at 0.2 probability. This showed a good performance in terms of relatively quick convergence, without the issue of getting stuck in local maxima. The latter was experienced at initial mutation rates of 0.02. At this low rate, the system seemed to produce little diversity even after only a few runs and the user could not achieve the predetermined goal as most candidates looked very similar and left no room for additional evolutionary breeding.

It is interesting to see that the average number of generations using the Human-based Genetic Algorithm is significantly higher compared to the Interactive Genetic Algorithm. This is not unexpected though. In case of the Interactive Genetic Algorithm, the user has to run every generation interactively. The time consumed per iteration is about 10 seconds render time plus user decision time, which was typically about 15 seconds. This means, an average run took just over 15 minutes. Comparing this to the Human-based Genetic Algorithm, the time for the first 10 runs is identical. But after that, the non-interactive generations, driven by the computational agent take virtually no time (under 1 second) for the decision making process and only the last generation that is to be presented to the user for interactive selection again, needs to be rendered, which takes the aforementioned 15 seconds. Therefore, many additional generations can be run in the same time, which the user seems to take advantage of in case of the Human-based Genetic Algorithm.

For the overall process it can be said, that if the user is less pleased with the results returned by the agent, the user will select candidates that are different, rather than similar. This triggers the agent to change course as well, running a lower level of confidence due to the inherent inability to predict the sudden random selection by the user, which in turn creates more diversity through increased mutation probability. Consequently, this allows the designer to choose a more intuitive, even unstructured approach to the modelling process, and a carefully, clearly planned execution is not a requirement anymore. The designer is essentially able to use playful discovery without endangering the end product. In a random or manual approach, this would cost either a lot of time, as many hundreds or thousands of parameters would have to be adjusted, or it would be impossible, given that a certain appearance of the buildings in the skyline can only be altered by changing the layout of the city blocks and the street pattern.

A few interesting cases could be observed, where an originally weak and seemingly unstructured response from the Algorithm in the first 10 iterations, is altered by the support of the software agent within a few iterations. The software agent suddenly drives the designs into a different direction from what the Genetic Algorithm did, and closely follows what it identified based on the human selections. Therefore, the overall system performed better than its individual components and provided a better user experience. For example, the brief was to create a city without water and flat buildings with no distinct city center. The Genetic Algorithm showed a high number of candidates with water, an average of 7 out of 9 per generation. The user inevitably selected candidates with land and no water. While the Genetic Algorithm continued to present candidates with water in higher proportion (Figure 26), the agent's classifier was trained by the user selection.

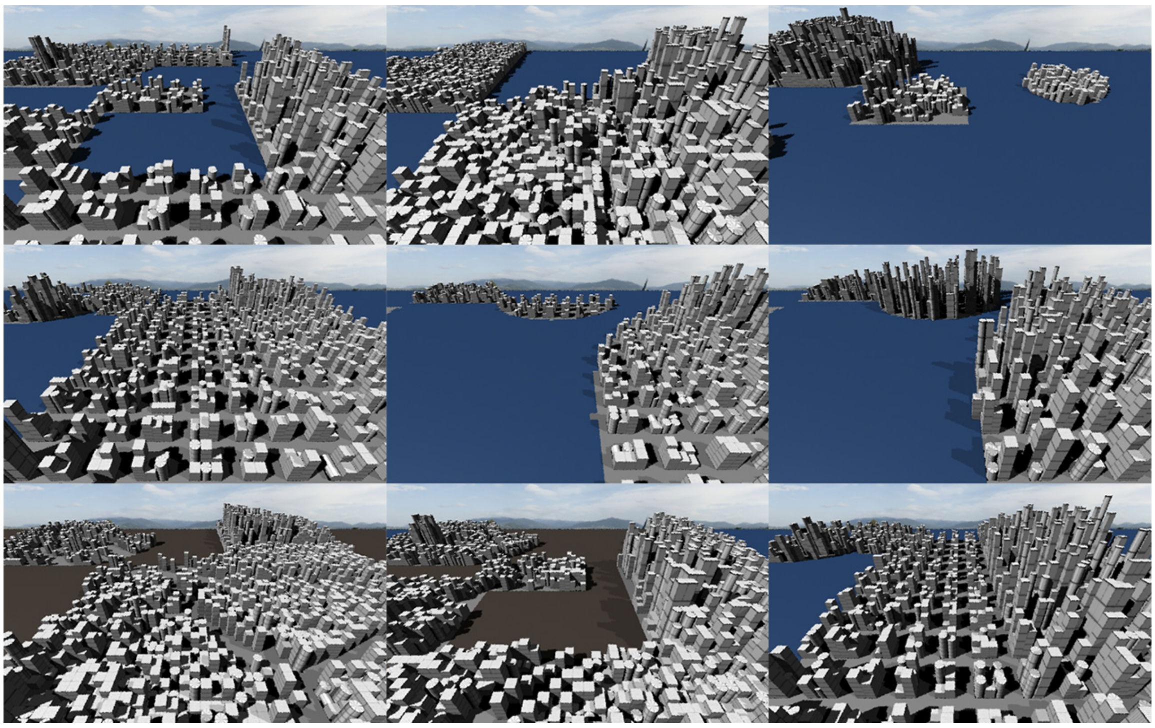


Figure 26 – Generation 10 of HBGA run presenting mostly water, contrary to the brief given

Once the agent came into effect, after only 4 additional generations, there were predominantly candidates containing land available (Figure 27). And even the candidates still containing water, had proportionally more land and buildings visible.

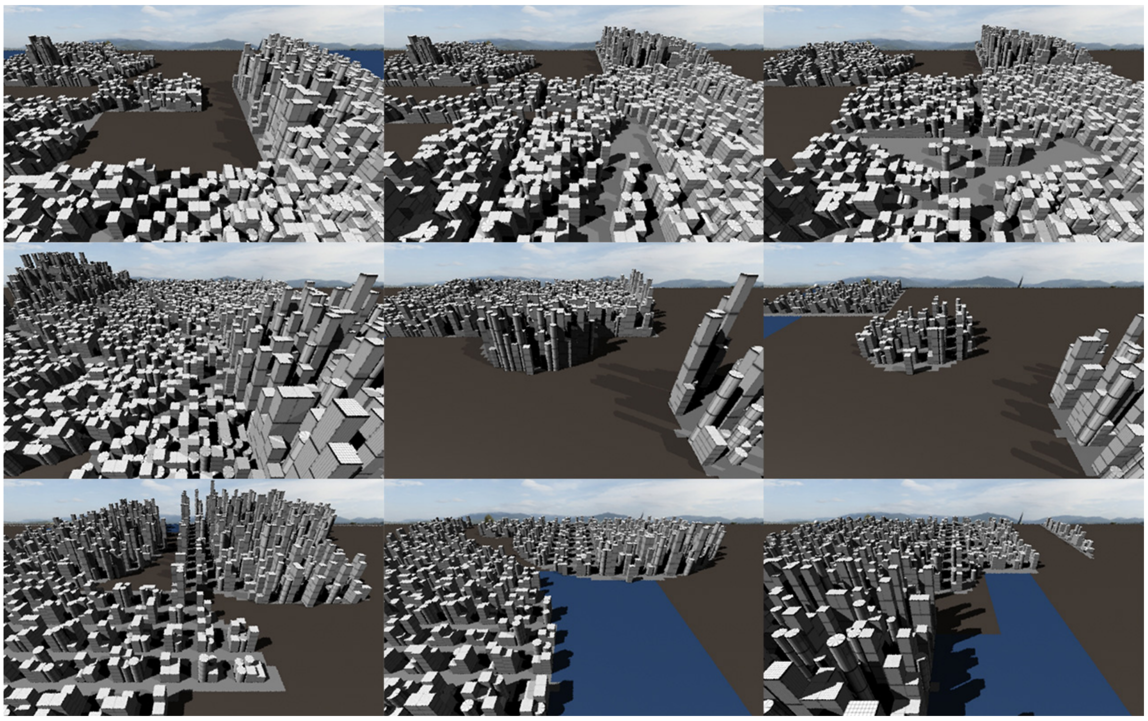


Figure 27 - Generation 14 of the same HBGA run as above, showing mostly land-based candidates

It was a bit surprising that the user did not always follow a straight approach towards the goal. For example, if a harbour city model was requested, quite a few selections involved no water at all. There are a few possible reasons for that. First, perhaps most of the other parameters did not fit the brief from the perspective of the user. Or, the fact that relatively little effort to create a new city layout was required, compared to manual modelling, led to a more playful attitude. Overall, it seems that the user is more adventurous using the Human-based Genetic Algorithm, changing direction a few times, for example from water on the left to water on the right when asked to make a harbour city. One would probably not attempt a drastic change after many man hours of modelling manually, as a larger diversion from the original layout might require a re-start of the whole manual modelling process. Due to the support of the agent and the relatively fast 'modelling' approach, there seems to be a lower boundary for otherwise significant changes. It seems that interactivity on one hand, but also the agent reducing fatigue on the other hand, allow for more user iterations and therefore exploration of different solutions.

The core of the agent architecture used in this study is the decision tree, which is induced at run-time and refined in subsequent iterations by use of the additional selections made by the user. Figure 28 shows part of a decision tree after 14 user-driven iterations.

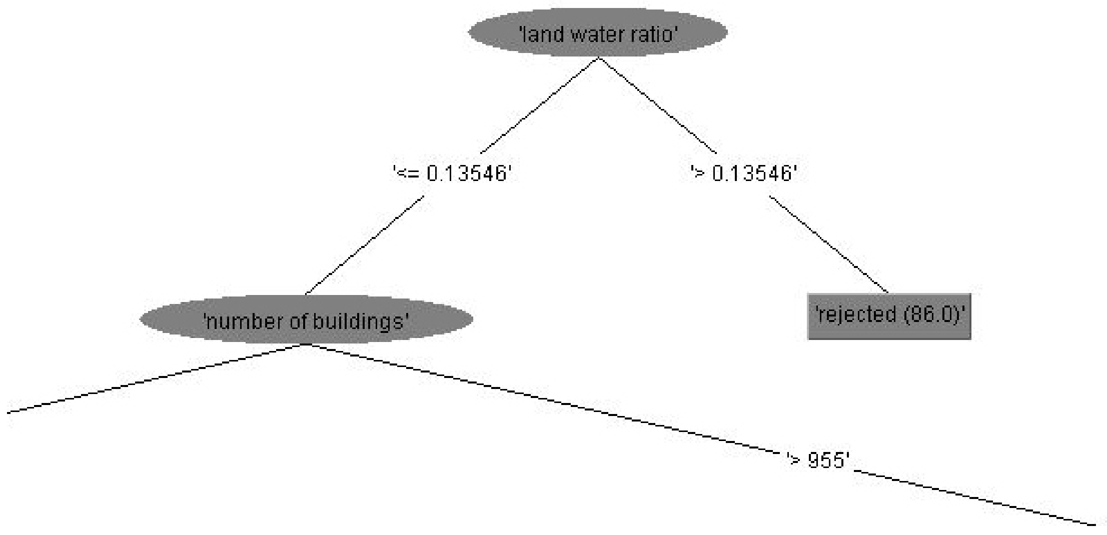


Figure 28 - Decision tree visualization with WEKA after 15 generations

The root attribute is land/water ratio, which gives a clear indication that any candidate with more water than 14% is to be rejected, which is true for 86 of all instances of the training set. The goal of this run was in fact to find a city with no water, and based on the training set, the classifier preferred any candidate with less than 14% of water. The full tree has 7 branches and shows a high rate of confidence (88% correctly classified instances), although most of the candidates were rejected right at the root, based on the amount of water compared to land. This is one indication of a useable classification tree, however, some of the other decision trees that have been examined throughout this study, were not as clear and had a lower level of confidence. There has been no clear indication that the J48 classifier produces viable results in every case. Sometimes the tree is not able to reliably identify candidates with a high confidence and the score was only around 60%. However, this is still a marginal improvement over a 50/50 ‘coin toss’, so there is some value in the use of the agent. It seems that perhaps the noise of the training data due to the low number of instance compared to other data mining tasks, might be a contributing factor. Hall et al. (2009) discuss this as a possible issue, and other work using J48 has identified that when the number of instances is low compared to the number of attributes, the J48 classifier becomes of limited use (Finlay, Connor, & Pears, 2011). In the instance of the procedural city generation, the classification is conducted on 5 attributes and typically the training set would include 10 initial instances. The selection of J48 was initially based on its popularity as a classifier, however a potential solution to improve the effectiveness of the classification might be the introduction of an alternative classifier, which could be part of future research and the matter requires further investigation. The WEKA framework allows for relatively easy adaption of different algorithms and the software system presented here, should be able to accommodate further testing, as suggested in section 5.4.

5 Discussion and Conclusions

5.1 Contributions

This thesis discusses two main contributions to knowledge. First, an original workflow using RIB archives for easy instancing, and rendering tools based on industry standards has been developed, which allows Multi-agent Genetic Algorithms to be employed in Computer Graphics design tasks. The open standard using Pixar's RIB format is widely used in the visual effects industry in order to accommodate rendering pipelines for film and game cinematics. Incorporating the same existing tools into a new workflow, opens new possibilities for research, but allows industry to adapt the offered processes with minimal adjustments.

Second, an enquiry into the viability of Human-based Genetic Algorithms to offset the known negative effects of fatigue in conjunction with common Interactive Genetic Algorithms has been conducted. Initial findings show some promising prospects in form of time savings and high number of possible iterations. The results have been documented and first conclusions have been drawn, which provide the cornerstone for deeper investigation in future.

5.2 Findings

The Infographics, while being only a preparation and test case for the main component of the research, the procedural city generation, still provided a number of important learnings.

The control over the mutation rate effectively translates into the ability to maintain a high level of novelty (through high mutation) for the presented candidates. In an abstractive sense, this helps to avoid the genetic algorithm getting stuck in local maxima in search of a global solution. In order to avoid the necessity for such control, which adds to the input required and therefore potentially compromises the goal to reduce user fatigue and increase the ability to execute a high number of runs if needed, the agent implementation of the Human Based Genetic Algorithm for the procedural city modelling had to account for the possibility of finding only local maxima in the search space. This is done by maintaining a balance between aggressive optimization towards the search goal and rewarding novelty and diversity by allowing a degree of mutation in each run.

An interesting observation is the influence of soft factors such as positive emotions. Fatigue seems not just to be based on attention span and focus, but also to be compensated by subjective positive emotions. Evaluating the candidate solutions presented by the computational agent seems to positively engage the designer more than when evaluating those presented by the IGA. The process of using interactive evolutionary concepts, which allow the user to observe convergence towards selected goals with each iteration, could be almost described as playful. Looking deeper into these finding is beyond the scope of this study and also centred in a different field of enquiry, though will be discussed briefly in section 5.4.

Looking at the number of iterations run by the Human-based Genetic Algorithm compared to pure Interactive Genetic Algorithm, it seems that the user might be happy to allow more iterations, if they are not interactive, but run by an agent. It seems not to be about keeping the maximum number of iterations low, but more about optimizing the final result within a certain time frame. The average time of the runs between IGA and HBGA were very similar, which could indicate that the user is more driven by time consumed, rather than the number of selections that have to be made by either the human or the agent. This might hint that the driving factor is indeed fatigue or attention span, and that a computational agent helps to optimize the result by running additional iterations. Based on this prototype, it looks as if the system of Interactive Evolutionary Computation and agents shows some promising benefits towards goal optimization, and a wider study could probably confirm this indication and provide additional insights.

When assessing the results it is important to note, that this study has been designed with a particular purpose in mind and a quick look at the bigger picture seems appropriate at this point. This research fills an important gap, as it provides an academic prototype for evolution-based interactive model design. The type of model as well as the combination of different agents, human and computational can be adjusted with a high flexibility. The long term idea is to take this framework and apply it to a large scale study using a significantly more complex modelling topic, namely computer generated character design, as part of a PhD. The idea to be able to model entire armies or crowds of characters is not new, but the approach is. Currently, character modelling and rigging is one of the most challenging tasks in computer graphics. Modelling an entire crowd, assuming that there are very few twins in any crowd, is exponentially more difficult and laborious (Wade, 2010). Applying the concepts and methods presented in this study could potentially reduce the amount of work not just during the design phase of individual characters, no matter if they are cartoon style or photo realistic, but also when creating large numbers of variants such as soccer fans in a stadium or soldiers in an army. The results of this study might give an indication of the possibilities, and prove that the software system works, but a larger collection of measurements would be desirable to test the application of agents with interactive genetic algorithms for different models and in particular computer graphics characters.

Summarizing it can be said, that the results of this study do not show conclusive evidence that agents lead to consistent improvements of Interactive Genetic Algorithms. But there are some indications that this approach has advantages. First, there are some promising signs when adding agents to the interactive process, for example the cases where the Genetic Algorithm seemed to suffer from a high mutation probability, which lead to a high diversity and no clear convergence. Once the agent ran some of the generations, a clear shift in direction towards the previous user selection was observable. This needs further proof, which a quantitative experiment could provide. Second, the observation that the user seemed to enjoy the interactive process more, once the agent was engaged, could prove to be a valuable insight. While this needs further investigation as well, looking at the psychological aspects of perceived intelligence by a computational system could provide additional value, seems like the next logical step in understanding the user experience of Interactive Evolutionary Computation better. This thesis provides a framework that both a quantitative study and an investigation into positive contributions to user experience could build on. Another aspect that the researcher intents to deepen in further study, is the application of the framework of Human-based Genetic Algorithms to computer generated character design. This thesis extends existing knowledge about Interactive Evolutionary Computation by providing a generic workflow based on RIB archives and industry standard render engines. The findings will make the increased complexity of character modelling and rigging manageable, so that a quantitative study is feasible, which might provide a better understanding of the mixed mode approach using human and computational agents in evolutionary computation for design. While the matter requires a lot more research in different areas, in can be said that this thesis has laid the foundation for this direction to be pursued.

5.3 Limitations

First of all, the models produced throughout this research are somewhat simplified. While justification for this approach has been provided and the software system has been designed to easily be expanded to accommodate much more complexity, the study has been conducted based on this limitation. As indicated in section 5.4 models with higher complexity could be part of future research. Aside from the reasons such as focus on the underlying agent research and reduction of additional user fatigue, the available hardware resources like memory size and processor speed are limiting factors as well. The outcomes are merely academic and would need justification in a commercial production environment. It would be desirable to run the model generation on a commercial render farm used by visual effects or games companies (game companies produce their cinematic sequences in a similar way as visual effects for film and television are created, by using non-real time rendering to increase the quality and complexity of the resulting movie clips). The purpose of this research is to present a new approach to interactive modelling, and not to implement a fully featured commercial product. Another hardware limitation is storage space, as this study has been conducted using a regular high spec PC, not a production facility sized network server. To get an indication of what is required, Weta Digital Ltd used 1 Petabyte of data for the production of the movie Avatar, with some individual shots larger than 4 Terabytes (Gruzas, 2012). Therefore it can be assumed, that it would be beneficial to run tests on a high capacity workstation or small render farm using highly detailed instanced models (see also section 5.4.1) in order to demonstrate the outcomes in a real world environment.

A further limitation is the amount of user testing with regards to measureable outcomes. While general conclusions can be drawn from the research conducted, it would be beneficial to run tests with a larger group of participants from at least two different levels of knowledge and skill. The software system has been developed with commercial applications, run by specialized users with moderate expertise in modelling, in mind. But based on the results of this study, even more convincing outcomes might be possible, if an approach with a broader participation would be investigated. For example, if two groups of experts and entirely novice users of modelling software would conduct user validation tests, it might be possible to demonstrate that even the full absence of previous expertise still leads to acceptable results due to the abstraction of parameters which the interactive, agent supported approach provides. This is subject to future research and beyond the scope of this master thesis.

5.4 Future Work

5.4.1 Procedurally generated buildings

There have been some successful attempts to procedurally generate buildings for city models in the past. One example uses shape grammars (Aschwanden et al., 2009) to model skyscrapers. Another approach is the use of L-systems as presented by Parish and Müller (2001). The generation of the overall city is different from the approach taken in this study. The latter uses L-systems for the city layout including available spaces for buildings and streets (Parish & Müller, 2001). The former is simply an extension of Parish's and Müller's work (Aschwanden et al., 2009). Future research into city models could make use of the existing L-system based approach. But there is also the possibility to evolve buildings using an evolutionary method such as genetic algorithms. Further, it would be possible to procedurally generate the textures for the buildings by use of rule based systems for example. While Parish and Müller (2001) had to generate all building textures by hand, and the same is true for this study, future work could enable more variety and complexity by eliminating manual texture painting.

5.4.2 Evaluation Agents

Aschwanden et al. (2009) investigate the use of agents to evaluate city model. They use a multi agent system to explore the usefulness of a procedural city. Their agent model is limited in that it is focused on the path finding and has a set of predefined goals. These limitations are purely based on the complexity of the urban environments which require long calculation times for each agent. According to Aschwanden et al. (2009) it is also virtually impossible to model the decision process of humans. One idea for future research into the usability of city models would be the addition of functional agents such emergency workers (firemen or policemen). There is also the possibility of adding vehicles for personal and public transport and the simulation of traffic flow. These options are subject to future research and might enable a refinement of the city model, once the basic structural elements such as streets and buildings have been evolved.

5.4.3 Boosting Human-based Genetic Algorithms

Another idea that could not be explored as part of this thesis is the addition of multiple learning agents for selection. Instead of running one learning selection agent to the Human-based Genetic Algorithm, it could be advantageous to use perhaps two or three, in order to improve overall system performance. Every time the user makes a selection, the whole population of candidate solutions is added to the training set for the learning agent. Both parent candidates selected by the human are added with a positive attribute, and all seven remaining unselected candidates are added with a negative attribute to the training data. Accordingly, the learning agent uses the learning algorithm to classify the population of candidates over and over again, with a growing training set and therefore improved quality of classification, and subsequently improved level of confidence for the learning agent to select two parents that the human user would have chosen as well. But this approach could be applied to the outcomes of the computational agent as well. Running the agent concurrently to the user and using every instance where the agent failed to classify the population correctly could be used to improve future classification modelling by the agent. As pointed out in the literature review of this study, this technique is being referred to as B*oosting* in machine learning. Boosting could improve the overall multi-agent system performance. But Boosting has also been strongly criticized as not being as effective in some cases as previously assumed (Long & Servedio, 2010). It would be interesting to identify whether the criticism of Boosting applies in this case as well, or if indeed a performance gain might be possible by using these methods. This investigation is beyond the scope of this research though and could be part of future work.

5.4.4 Psychological Aspects

As mentioned in section 4.2, it could be interesting to evaluate soft factors, which were perceived during IGA experiments. Observed engagement through the almost playful experience of selecting candidates for future populations, could be described as a gamification of modelling. It could be of value to investigate the effects of this playful experience on the overall modelling process in terms of efficiency improvements and long term engagement of the designer with the artefact.

Similarly, the presence of an intelligent system such as an agent seems to make the daunting process of waiting for rendered candidates of procedural cities a bit easier as it implies that an artificial intelligent system has taken some of the workload of the user. It would be interesting to test this objectively by running different tests, with an agent and with a placebo, and see if the *idea* of intelligence softens the effects of fatigue as effectively as an *actual* intelligent system does. This type of research is located in the field of psychology though and has to be left for future study.

5.5 Conclusion

Comparing Human-based Genetic Algorithms with Interactive Genetic Algorithms shows that HBGA is faster in the experiments conducted throughout this study, and allows for more iterations in the same time, which illustrates the potential of this approach. Adding a computational agent to human-centric Evolutionary Computation for complex modelling workflows seems have a positive impact on the designer's curiosity and perhaps also improves the user's engagement with the design task, which could be a viable measure to counteract fatigue. A successful first step in the direction of improving user experience in Interactive Evolutionary Computation for Computer Graphics has been documented in this thesis, and an adaptable, robust workflow that accesses existing pipeline tools has been established.

6 References

Anderson, C., Buchsbaum, D., Potter, J., & Bonabeau, E. (2008). Making Interactive Evolutionary Graphic Design Practical. In A. P. T. Yu, P. L. Davis, D. C. Baydar, & P. R. Roy (Eds.), *Evolutionary Computation in Practice* (pp. 125–141). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-75771-9_6

Aschwanden, G., Haegler, S., Halatsch, J., Jeker, R., Schmitt, G., & van Gool, L. (2009). Evaluation of 3D City Models Using Automatic Placed Urban Agents. Presented at the International Conference on Construction Applications of Virtual Reality, Sydney. Retrieved from http://www.academia.edu/283855/Evaluation_of_3D_City_Models_Using_Automatic_Placed_Urban_Agents

Balaji, P. G., & Srinivasan, D. (2010). An Introduction to Multi-Agent Systems. In D. Srinivasan & L. C. Jain (Eds.), *Innovations in Multi-Agent Systems and Applications - 1* (pp. 1–27). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-14435-6_1

Barbosa, R. P., & Belo, O. (2008). Autonomous Forex Trading Agents. In P. Perner (Ed.), *Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects* (pp. 389–403). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-70720-2_30

Bentley, P. (1999). *Evolutionary Design by Computers*. Morgan Kaufmann.

Bentley, P. (2002). *Creative evolutionary systems*. San Francisco, CA.; San Diego, CA: Morgan Kaufmann ; Academic Press.

Beyer, H.-G., & Schwefel, H.-P. (2002). Evolution Strategies - A Comprehensive Introduction, *1*(1), 3–52. http://doi.org/10.1023/A:1015059928466

Biles, J. A. (1994). GenJam: A Genetic Algorithm for Generating Jazz Solos. *International Computer Music Conference*. Retrieved from http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/divulgativos/heuristicos-busqueda/JazzSolosbyGAs.pdf

Bohnacker, H. (2012). *Generative design: visualize, program, and create with processing*. New York: Princeton Architectural Press.

Bourg, D. M., & Seemann, G. (2004). *AI for Game Developers* (1 edition). Sebastopol, CA: O'Reilly Media.

Brustoloni, J. C. (1991). *Autonomous agents: characterization and requirements*. School of Computer Science, Carnegie Mellon University.

Buckland, M. (2004). *Programming Game AI By Example* (1 edition). Plano, Texas: Jones & Bartlett Learning.

Candy, L. (2006, November). Practice Based Research Guide. Retrieved May 26, 2012, from http://www.scribd.com/doc/72480138/Practice-Based-Research-Guide

Cho, S.-B. (2002). Towards Creative Evolutionary Systems with Interactive Genetic Algorithm. *Applied Intelligence*, *16*(2), 129–138. http://doi.org/10.1023/A:1013614519179

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. Cambridge, Mass: MIT Press.

Connor, A. M. (1996). *The synthesis of hybrid mechanisms using genetic algorithms (BL)* (Ph.D.). Liverpool John Moores University (United Kingdom), England. Retrieved from http://search.proquest.com.ezproxy.aut.ac.nz/docview/301467506?pq-origsite=summon

Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, *14*(1), 65–86. http://doi.org/10.1111/j.1365-2575.2004.00162.x

Dawkins, R. (1986). *The blind watchmaker: why the evidence of evolution reveals a universe without design*. New York: Norton.

Dean, R. T., & Smith, H. (2009). *Practice-led Research, Research-led Practice in the Creative Arts* (1st ed.). Edinburgh: Edinburgh University Press.

D'Inverno, M., & Luck, M. (2004). *Understanding Agent Systems*. SpringerVerlag.

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to Advanced Empirical Software Engineering* (pp. 285–311). Springer London. Retrieved from http://link.springer.com/chapter/10.1007/978-1-84800-044-5_11

Ferrucci, D., Levas, A., Bagchi, S., Gondek, D., & Mueller, E. T. (2013). Watson: Beyond Jeopardy! *Artificial Intelligence*, *199–200*, 93–105. http://doi.org/10.1016/j.artint.2012.06.009

Finlay, J., Connor, A. M., & Pears, R. (2011). Mining Software Metrics from Jazz. In *2011 9th International Conference on Software Engineering Research, Management and Applications (SERA)* (pp. 39–45). http://doi.org/10.1109/SERA.2011.40

Fogel, D. B. (2002). *Blondie24: Playing at the Edge of Ai*. Morgan Kaufmann.

Fogel, D. B. (2006). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Wiley.

Franklin, S. (1997). *Artificial Minds*. MIT Press.

Franklin, S., & Graesser, A. (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents (pp. 21–35). Springer-Verlag.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, *14*(771-780), 1612.

Fry, B., & Reas, C. (2004). Processing.org. Retrieved October 3, 2014, from http://processing.org/

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning* (1 edition). Reading, Mass: Addison-Wesley Professional.

Gómez-Sanz, J. J., & Pavón, J. (2004). Methodologies for Developing Multi-Agent Systems. *J. UCS*, *10*(4), 359–374.

Greenberg, I. (2007). *Processing: Creative Coding and Computational Art* (1 edition). Berkeley, CA : New York: friendsofED.

Greenberg, I., Xu, D., & Kumar, D. (2013). *Processing: Creative Coding and Generative Art in Processing 2* (2 edition). Berkeley, Calif.; London: friendsofED.

Gruzas, K. (2012). *Case Study Weta Digital*. Wellington, New Zealand: Fujitsu Corporation.

Gu, Z., Xi Tang, M., & Frazer, J. H. (2006). Capturing aesthetic intention during interactive evolution. *Computer-Aided Design*, *38*(3), 224–237. http://doi.org/10.1016/j.cad.2005.10.008

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, *11*(1), 10–18. http://doi.org/10.1145/1656274.1656278

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. http://doi.org/10.2307/25148625

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor: University of Michigan Press.

Iivari, J., & Venable, J. (2009). Action research and design science research - Seemingly similar but decisively dissimilar. *ECIS 2009 Proceedings*. Retrieved from http://aisel.aisnet.org/ecis2009/73

Kamalian, R., Yeh, E., Zhang, Y., Agogino, A. M., & Takagi, H. (2006). Reducing Human Fatigue in Interactive Evolutionary Computation Through Fuzzy Systems and Machine Learning Systems. In *2006 IEEE International Conference on Fuzzy Systems* (pp. 678–684). http://doi.org/10.1109/FUZZY.2006.1681784

Keeton, W. T. (1996). *Biological Science* (6 edition). New York: W W Norton & Co Inc.

Kosorukoff, A. (2001). Human based genetic algorithm. In *2001 IEEE International Conference on Systems, Man, and Cybernetics* (Vol. 5, pp. 3464 –3469 vol.5). http://doi.org/10.1109/ICSMC.2001.972056

Kostic, Z., Radakovic, D., Cvetkovic, D., Trajkovic, S., & Jevremovic, A. (2012). Comparative Study of CAD Software, Web3D Technologies and Existing Solutions to Support Distance-Learning Students of Engineering Profile. *IJCSI International Journal of Computer Science Issues*, *9*(4), 7.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (1 edition). Cambridge, Mass: A Bradford Book.

Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., & Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (1st ed. 2003. Corr. 2nd printing edition). Norwell, Mass: Springer.

Lally, A., Prager, J. M., McCord, M. C., Boguraev, B. K., Patwardhan, S., Fan, J., … Chu-Carroll, J. (2012). Question analysis: How Watson reads a clue. *IBM Journal of Research and Development*, *56*(3.4), 2:1–2:14. http://doi.org/10.1147/JRD.2012.2184637

Lanzi, P. L. (2011). Genetic Algorithms and Other Evolutionary Techniques. Politecnico di Milano.

Lechner, T., Watson, B., Ren, P., Wilensky, U., Tisue, S., Felsen, M., … Felsen, M. (2004). *Procedural Modeling of Land Use in Cities*. Evanston, Illinois: Northwestern University.

Li, M., & Kou, J. (2001). The schema deceptiveness and deceptive problems of genetic algorithms. *Science in China Series : Information Sciences*, *44*(5), 342–350. http://doi.org/10.1007/BF02714737

Long, P. M., & Servedio, R. A. (2010). Random classification noise defeats all convex potential boosters. *Machine Learning*, *78*(3), 287–304. http://doi.org/10.1007/s10994-009-5165-z

Mandelbrot, B. B. (1983). *The fractal geometry of nature*. Macmillan.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.

Mitchell, T. (1997). *Machine Learning* (Auflage: International edition). New York: Mcgraw-Hill Publ.Comp.

Negnevitsky, M. (2004). *Artificial Intelligence: A Guide to Intelligent Systems* (2nd ed.). Addison-Wesley.

Okabe, A., Boots, B., Sugihara, K., & Chiu, S. N. (2000). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams* (2 edition). Chichester ; New York: Wiley.

Okun, J. A., & Zwerman, S. (Eds.). (2010). *The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures* (1st ed.). Burlington, MA: Focal Press.

Parish, Y. I. H., & Müller, P. (2001). Procedural Modeling of Cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 301–308). New York, NY, USA: ACM. http://doi.org/10.1145/383259.383292

Pearson, M. (2011). *Generative Art* (1 edition). Shelter Island, NY : London: Manning Publications.

Perevalov, D. (2013). *Mastering openFrameworks: Creative Coding Demystified*. Birmingham, UK: Packt Publishing.

Pimpale, P., & Bhande, N. (2007, December). *Genetic Algorithms Made Easy*. Retrieved from http://www.slideshare.net/pbpimpale/genetic-algorithms-200688

Prager, J. M., Brown, E. W., & Chu-Carroll, J. (2012). Special Questions and techniques. *IBM Journal of Research and Development*, *56*(3.4), 11:1–11:13. http://doi.org/10.1147/JRD.2012.2187392

Prusinkiewicz, P., & Lindenmayer, A. (1996). *The Algorithmic Beauty of Plants*. Springer.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. http://doi.org/10.1007/BF00116251

Reas, C., & McWilliams, C. (2010). *Form+Code in Design, Art, and Architecture* (1st edition). New York: Princeton Architectural Press.

RenderMan Options and Attributes. (2014). Retrieved November 18, 2014, from http://rendermansite.pixar.com/view/rib-scene-description

Renner, G., & Ekárt, A. (2003). Genetic algorithms in computer aided design. *Computer-Aided Design*, *35*(8), 709–726. http://doi.org/10.1016/S0010-4485(03)00003-4

Ricordel, P.-M., & Demazeau, Y. (2002). Volcano, a Vowels-Oriented Multi-agent Platform. In *Revised Papers from the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems: From Theory to Practice in Multi-Agent Systems* (pp. 253–262). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=646697.703762

Riedl, M., Thue, D., & Bulitko, V. (2011). Game AI as Storytelling. In P. A. González-Calero & M. A. Gómez-Martín (Eds.), *Artificial Intelligence for Computer Games* (pp. 125–150). Springer New York. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4419-8188-2_6

Russell, S. J., & Norvig, P. (1994). *Artificial intelligence: a modern approach* (1st ed). Upper Saddle River, N.J: Prentice Hall.

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: a modern approach* (2nd ed). Upper Saddle River, N.J: Prentice Hall.

Sarker, R. A., & Ray, T. (2010). *Agent-Based Evolutionary Search*. Springer Science & Business Media.

Schapire, R. E. (1999). A brief introduction to boosting. In *Ijcai* (Vol. 99, pp. 1401–1406). Retrieved from http://math.biu.ac.il/~louzouy/courses/seminar/boost1.pdf

Shiffman, D. (2012). *The Nature of Code: Simulating Natural Systems with Processing* (1 edition). New York, NY, USA: Kickstarter.

Sims, K. (1992). Interactive evolution of dynamical systems. In *Toward a practice of autonomous systems* (pp. 171–178).

Skolicki, Z., & Jong, K. D. (2004). Improving Evolutionary Algorithms with Multi-representation Island Models. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, … H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature - PPSN VIII* (pp. 420–429). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-30217-9_43

Smith, D. C., Cypher, A., & Spohrer, J. (1994). KidSim: Programming Agents Without a Programming Language. *Commun. ACM*, *37*(7), 54–67. http://doi.org/10.1145/176789.176795

Sniedovich, M., & Lew, A. (2006). Dynamic Programming: an overview. *Control and Cybernetics*, *35*(3), 513.

Sorenson, N., & Pasquier, P. (2010). Towards a Generic Framework for Automated Video Game Level Creation. In C. D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcazar, … G. N. Yannakakis (Eds.), *Applications of Evolutionary Computation* (pp. 131–140). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-12239-2_14

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, Mass: A Bradford Book.

Takagi, H. (1998). Interactive evolutionary computation: System optimization based on human subjective evaluation. In *IEEE Int. Conf. on Intelligent Engineering Systems (INES'98)* (pp. 17–19). Retrieved from http://pdf.aminer.org/000/305/069/discrete_fitness_values_for_improving_human_interface_in_an_interactive.pdf

Takagi, H. (2001). Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, *89*(9), 1275 –1296. http://doi.org/10.1109/5.949485

Takagi, H., & Iba, H. (2005). Interactive Evolutionary Computation. *New Generation Computing*, *23*(2), 113–114.

Todd, P. M., & Werner, G. M. (1999). Musical Networks. In N. Griffith & P. M. Todd (Eds.), (pp. 313–339). Cambridge, MA, USA: MIT Press. Retrieved from http://dl.acm.org/citation.cfm?id=346573.346629

Turing, A. M. (1950). Computing Machinery and Intelligence. In R. Epstein, G. Roberts, & G. Beber (Eds.), *Parsing the Turing Test* (pp. 23–65). Springer Netherlands. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4020-6710-5_3

Vaishnavi, V. (2008). *Design science research methods and patterns: innovating information and communication technology*. Boca Raton: Auerbach Publications.

Velez, R., & Clune, J. (2014). Novelty Search Creates Robots with General Skills for Exploration. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation* (pp. 737–744). New York, NY, USA: ACM. http://doi.org/10.1145/2576768.2598225

Wade, D. P. (Ed.). (2010). *d'artiste Character Modeling 2: Digital Artists Master Class* (Slp edition). Mylor, S. Aust.: Ballistic Publishing.

Welcome | 3delight. (2014). Retrieved October 3, 2014, from http://www.3delight.com/en/index.php

Wiering, M., & Otterlo, M. van (Eds.). (2012). *Reinforcement Learning: State-of-the-Art* (2012 edition). Heidelberg ; New York: Springer.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining : Practical Machine Learning Tools and Techniques* (3rd ed.). Burlington: Morgan Kaufmann.

Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems* (2nd Edition edition). Chichester, U.K: John Wiley & Sons.

Wooldridge, M., & Jennings, N. R. (1995). Agent theories, architectures, and languages: A survey. In M. J. Wooldridge & N. R. Jennings (Eds.), *Intelligent Agents* (pp. 1–39). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/3-540-58855-8_1