FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Collaborative Procedural Modeling driven by User Feedback

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Andreas Winkler
Matrikelnummer 1129264

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr. Michael Wimmer
Mitwirkung: Mag. Martin Ilčík

Wien, 19. April 2017

Andreas Winkler	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Collaborative Procedural Modeling driven by User Feedback

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Andreas Winkler
Registration Number 1129264

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr. Michael Wimmer
Assistance: Mag. Martin Ilčík

Vienna, 19th April, 2017

Andreas Winkler	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Andreas Winkler
Neunkirchnerstraße 17
2732 Willendorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. April 2017

Andreas Winkler

Abstract

In this work a user centered procedural modeling framework is proposed which combines rule based content generation with the concepts of recommendation systems. Using the ACGAX modeling language, artists are able to write grammar scripts for the creation of diverse and complex 3D scenes, controlled with a simple goal notation. These scripts are evaluated and executed by the system to generate 3D-shapes using stored production rules from the cloud. The rule selection process is based on content based information filtering systems to create results matching the user's preferences. User feedback is collected in a way that integrates explicit feedback into the modeling work flow via manual locking operations. These actions allow users to directly control the derivation process of grammars by fixing certain parts of the derivation tree. The goal of this research is not only to create a modeling tool, but a growing database of grammars, rules and feedback records. By observing how users interact with the grammars, the system learns which rules are most suitable for certain goals. The proposed system is designed to to learn from a user's actions to improve the cloud based rule selection process for future modeling tasks.

Contents

CHAPTER 1

Introduction

Procedural modeling is a popular approach to create detailed 3D environments for the use in entertainment media or urban planning. Most procedural applications generate architectural content via symbolic matching of design grammars, which are able to generate a vast variety of visual content from a set of rules and parameters. While artists provide rules and grammars, the modeling framework generates visual scenes via automatic rule selection. With large scale projects and multiple artists involved, the complexity of the rule selection process increases greatly. In this work, the possibilities of a collaborative procedural modeling framework are explored, with the focus on user interaction.

1.1 Motivation

The core motivation behind this work is **content creation**. Any application of computer graphics requires visual content to be displayed. While the scope of computer generated (i.e. algorithmically generated) content is nearly limitless, it often misses certain aspects of user orientation. In most applications, manual content generation is not only far more specific to a user's needs but also incorporates an artists' skill into the generation process, which is something that modeling algorithms cannot really achieve. The motivation for this work is to combine the huge possibilities of procedural modeling with the user centered design of a manual modeling tool. The goal is to create a versatile modeling framework for procedural creation of 3D shapes while incorporating manual tools which allow artists to incorporate detailed design decisions into the procedural generation process.

1.2 Problem statement

With a steady increase in computation power comes a higher demand for digital assets, both in quality and quantity. This demand also driven by increasing standard screen

resolutions for most devices. Moving to HD and 4K resolutions, the differences in the required level of detail of the displayed content is significant. **Procedural content generation** or procedural modeling describes the task of creating digital assets such as 3D models, textures, sounds, etc. from the ground up by using code statements and scripts rather than recording or directly modeling objects and assets from the real world [STN16].
One important approach - in the scope of this thesis - at procedural modeling is the use of formal languages to describe the construction of an object. In the context of 3D-modeling, symbols are used to describe shapes which are then processed and transformed into other shapes by a set of production rules. By applying a combination of rules, weighted by certain criteria, a potentially endless variety of objects can be created from very little amount of code. While many languages for the creation of shapes have been developed, a significantly lower amount of research has been done for a meaningful selection and weighting of rules.

1.3 Scalability

With the use of procedural modeling frameworks in large scale projects, modeling tasks might greatly exceed the capacity of a single artist. Therefore, collaborative work on a shared rule-base is necessary, which results in a very large amount of data that needs to be managed. With more rules to choose from, the complexity of the rule selection process increases. Ideally, a modeling framework would select rules in a way that is restrictive enough to solve a specific goal, but allows for meaningful variation to utilize the benefits of collaborative work. The difficulty of this task increases with the number of involved artists and their supplied rules. In such a scenario, a purely automatic rule selection process can no longer fulfill these requirements. Therefore, some level of control has to be shifted to the user.

1.3.1 User feedback

Complex grammars are able to create highly detailed and versatile scenes, which makes it very important that the user can rely on the system to respect his or her preferences. Ideally, rules should be automatically selected matching not only the context but also the user's preferences and feedback. In a large collection of grammars and rules, specific results could be generated from few lines of code. The difficult part is to choose the most suitable rules to create results matching the users' demands. The selection of rules therefore depends on the collection and evaluation of large amounts of user feedback. As a part of this problem it is necessary to find suitable representations for a user's feedback as well as a way to retrieve it. For the selection of rules, the user's preferences need to be extracted from this information in order to create suitable end results for the user. This challenge can be defined as **user-centered rule selection** and is the main problem I am addressing in this thesis.

1.3.2 Recommendation systems

The general concept of user-centered rule selection is closely related to the idea of **recommendation systems** [FJN+14]. A recommendation system is a type of information filtering system, which aims to select information or data for a user by predicting his interests and preferences based on previous user feedback. Unrelated to procedural content generation, these systems are utilized in a wide area of fields, mostly for search queries and ranking systems to provide the most meaningful and relevant items for the user. Advanced systems not only consider the preferences of a single user, but derive them from other users with common interests. A growing userbase results in a constant learning process for the system.
While the utilization of user feedback is tightly coupled in the software of online shops, media databases or social networks, these features are insufficient in the context of procedural modeling. The concepts described in this thesis aim to close this gap.

1.4 Michelangelo modeling system

The procedural modeling language used in this project is based on the Michelangelo collaborative modeling framework by Martin Ilčík and Michael Wimmer [IW16]. Michelangelo is the first cloud based collaborative design framework. Its main contributions are a cloud repository of grammars containing rules which are shared between multiple artists over different grammars. The authors focused on advancing the possibilities of current procedural modeling frameworks to a cloud environment suitable for multiple users, greatly reducing the required management effort.

1.4.1 Overview

The framework generates 3D-shapes by parsing grammar scripts based on C# syntax. The files serve as a runnable script containing coding elements such as variables or loops which help to generate containers for grammar symbols and rules. In the most basic case, a grammar script defines Axioms and Rules, whereas Rules are applied based on an Axiom's goals.

1.4.2 Grammars

A grammar serves as a 'recipe' for a 3D model. It is a script file containing combination of code statements and definitions for axioms and rules. Axioms are initial shapes, rules are the operations transforming them into different, more refined shapes.

Axioms of a grammar are evaluated step by step: The system selects a single non-terminal axiom and requests a rule. If a rule is found, the shape is transformed accordingly and the system continues with the next axiom. If no more rules can be applied to an active axiom, it becomes a terminal shape and its derivation is finished.

1.4.3 Axioms

An axiom is a basic non-terminal symbol representing a three-dimensional box, which can be initialized with specific position, size and goals. Rules are applied to axioms, changing its appearance and semantics. When no more rule can be applied to an axiom, its derivation is finished and its shape is final. Axioms are always limited to the grammar they were defined in. They are not pushed into the database and are never accessed by other grammars.

1.4.4 Rules

Rules are operations iteratively applied to axioms in order to form more complex shapes. There are different types of rules applying both geometric and semantic changes to axioms. These include:

- **Transformation rules** include most basic geometric transformations, such as translation, scaling and rotating. Advanced parameters are available to specify rotation origins or selecting between local and global scaling.
- **Shape rules** are more complex transformations, changing the dimension of a shape, or creating multiple new shapes by splitting, copying or repeating an existing shape.
- **Material rules** change the color, hue or saturation of a shape.
- **Deformation rules** taper, shear or bend a shape.
- **Semantic rules** include rules to add goals and attributes to an object, or to set a shape's final geometry to a specific geometric primitive.
- **Concat** is a special rule which sequentially applies a collection of other rules to a shape in the same step.

Rules are not limited to the grammar they are defined in. When parsing a grammar, the framework selects any matching rule from all available grammars to apply to an axiom. Local rules (i.e. rules from the current grammar) are preferred over global rules. Grammars therefore do not necessarily have to contain both Axioms and Rules. When parsing a grammar with axioms, but no rules, the framework will attempt to apply global rules. A grammar without axioms, will provide rules for other grammars' axioms.

1.4.5 Goals

Goals are fundamental for the derivation of an axiom, as they steer the selection of rules. Rules are selected based on matching goals with the axiom. When an axiom is defined, a list of goals is provided. The goals are strings, serving as keywords to describe the purpose of the object.

As counterpart to the goals provided by axioms, rules fulfill goals. Each rule is defined with a list of goals that it is supposed to fulfill. This means a rule can only be applied to an axiom containing the goal(s) it fulfills.

1.4.6 Attributes

Attributes can be seen as additional parameters for axioms which can be added via a specific rule. An attribute consists of a key and a primitive value. When applying a rule, attribute values can be used as parameters. Alternatively, additional constraints based on attributes can be added to rules.

1.5 Existing framework

The modeling framework used in this project is an offline version of Michelangelo which provides all the necessary tools to create and parse grammars and render the resulting objects.

1.5.1 Rule selection

The rule selection process is the core component of this system. When a grammar is executed and evaluated, the system needs to choose a rule to be applied. Before probabilities for the actual rule selection process are calculated, the system needs to find out which rules can be applied in the given context. A rule is only applicable if it meets certain requirements, which are composed of goals, tags and attributes. Goals are the key elements of a grammar and control the order of rule application. During derivation, the system first selects one of the shape's goals. Then, the system searches for applicable rules which fulfill this goal (Rules which fulfill multiple goals are currently not considered). Finally, one of these rules is chosen at random. In the proposed extension to the framework, the probabilities for the rule selection are calculated with the aid of retrieved user feedback.

Rule sources

A major aspect of this project is that rules are not limited to a single grammar. While building a specific 3D object with a specialized grammar script is an important task, this system focuses on the combination of rules from different grammars. For the task of rule selection, the system either retrieves rules from the local grammar, or from the cloud. Whenever a rule to fulfill a specific goal is sought, rules defined in the grammar that is currently parsed are preferred. If no local rule is applicable, the system looks for stored rules in the cloud which originated from a different grammar.

This also means that any local rule within a grammar has to be made globally available for other grammars. For this task the system utilizes a database were the a grammar's

rules are stored when it is parsed. Global rules are retrieved from this database during the derivation of a grammar, when no matching local rule was found to fulfill a specific goal.
Rules are retrieved in this order to give users more control over their grammars. Defining rules locally will ensure some level of deterministic results within the grammar and allows the user to control for which parts of his grammar he wants to retrieve rules from the cloud.

1.5.2 Goals

When defining a grammar, goals are assigned to both axioms and rules, in the form of strings. These goals provide the semantic information required to steer the derivation process. An axiom is always defined with a set of goals. While this may differ depending on how rules are set up by the user, an axiom's goals should describe the desired end result of the shape. For instance, creating an axiom with the goal "chair", provides the basis for the derivation and creation of a chair model, assuming appropriate rules are provided.
To achieve this, goals are also assigned to rules, to specify which goals this rule is supposed to fulfill. In our example, in addition to an axiom with the goal "chair", the user could define a rule which fulfills the goal "chair". When this rule is applied, the axiom's goal is fulfilled (i.e. it becomes inactive). In order to create detailed results, rules can also assign new goals to a shape. While fulfilling the goal "chair", a rule could split the shape into new parts and assign different goals to them. E.g: a rule fulfills "chair" and creates three new shapes with the goals "lean", "seat" and "legs". The user could then define rules to fulfill these goals to further refine the model.
When an axiom is defined with multiple goals, only one of them is fulfilled at a time in the most common case. This leaves the resulting shape with an unfulfilled goal from the initial definition, in addition to possibly multiple goals that were added by the applied rule. When a shape has multiple unfulfilled (=active) goals, the order of assignment is important. Within an axiom and its derived shapes, the system keeps track over the timestamps (i.e. the index of the derivation step) of assigned goals. For further derivation of a non-terminal shape, the system always attempts to fulfill the oldest goal first. If the system cannot find an applicable rule to fulfill the oldest goal, the second oldest goal is next in line. However, the oldest unfulfilled goal is kept in the shape and will be evaluated first again in the next derivation step. As an addition to the framework, the special case when multiple goals were assigned in the same derivation step is addressed. A special algorithm is applied to find the optimal order of application (see Section 4.3).

The assignment and selection of goals basically follows a First-in-First-out strategy. This chosen application order is important for the quality of created results for more complex grammars. Generally, goals that were specified earlier correspond to the general appearance of a scene (e.g. "chair", "house", etc.), while later goals describe more detailed features.

1.5.3 Attributes

While goals control the derivation of axioms in a grammar, attributes are used for detailed rule selection and application. Attributes are identified with a name and contain numerical values, boolean values or strings. Attributes can be assigned to a shape by applying specific rules. Attribute values can then be used by other rules as parameters. For instance, an attribute "size" with the numeric value 1.5 is assigned to a shape. Later, a scaling rule reads the "size" attribute from the shape and scales the object by a factor of 1.5. Attributes can also be used for rule selection by adding constraints to a rule. In the above example, a constraint could be assigned to the scaling rule, so that it can only be applied to shapes with the attribute "size".

1.6 Approach

This project's goal is to extend the versatility of the existing procedural modeling framework while focusing on user feedback. The main idea is to provide a tool which allows a user to define grammars for the creation of 3D shapes and to provide feedback on the resulting objects. The presented project aims to combine the field of procedural content generation with the possibilities of recommendation systems. The main benefits I attempt to achieve with the proposed concept are the following:

- A **scalable system**, able to contain large amount of assets and user input.
- A **user centered design**, allowing to create matching results, with very little input required.
- An **adaptive environment**, capable of learning suitable rules.

1.6.1 Scalable system

The system is designed to work with an arbitrary amount of available data. The goal is to provide suitable results for a user-defined grammar. Depending on the context of the grammar, the selected rules could origin from the user's grammar itself, or it could be selected from one grammar out of thousands of available ones in the database, based on context and user feedback.

1.6.2 User centered design

Procedural modeling languages describe the construction of 3D Objects with symbols and code statements. Mostly depending on the code length, this ranges from a simple box to a large cathedral with detailed ornaments. While processing power and rendering time certainly impose a limit on the level of detail of a 3D Object, the modeling process itself is mostly limited by the amount of work an artist is willing to do.
The proposed system attempts to greatly simplify this process, by meaningful and context specific selection of rules from a database, while allowing the user to improve the

engine itself by providing positive or negative feedback for a created object. With a properly filled database, the required amount of rules for a specific task should be greatly lowered.

1.6.3 Adaptive environment

Procedural content generation is used to create a vast amount of unique assets. The dynamic nature of generated content is mostly due to variable parameters. While this does allow for a large variety, the objects created from a single script or grammar often only differ in scale, size, or color of shapes.
Each created grammar will provide new rules for the database. My approach is expanding this aspect through an adaptive learning process: Provided user feedback will improve the selection process for the rules for a particular shape. Even in an environment with unsuitable rules (i.e. rules which do not meet the user's demands in a specific application), user feedback will continue a steady learning process which moves the focus of the selection algorithm to more suitable rules. The goal is to provide a self-sustaining dynamic procedural modeling engine, which constantly improves its average results.

1.7 Structure of the work

The following chapters of this work describe the theoretical background for the methods used for this project. An overview of related techniques and interesting projects in the fields of procedural modeling and information retrieval is provided (Chapter 2).
The proposed concept, its challenges and goals, as well as alternative approaches are presented. I give an insight in the decisions made during development of the proposed system (Chapter 3).
In the next chapters, I will describe the technical details and the implementation of the project in detail (Chapter 4). This includes a description of the architecture, algorithms and user interface, as well as an overview of the recommended usage of the tool (Chapter 5). Last but not least, various results from different applications of this systems are presented (Chapter 6). I go into detail about the benefits and limitations of this system as well as possible future extensions and improvements (Chapter 7).

CHAPTER 2

Related Work

The use of design grammars is very common in procedural modeling applications. The languages used in recent frameworks such as Michelangelo are mostly based on a range of notable developments from the last few decades. In this chapter an overview of the most groundbreaking research in procedural modeling is presented. As user feedback is an important part of this work, an introduction to recommendation systems and its origins is shown. Recommendation systems as well as the retrieval of user feedback exist in various forms and variants, all with different challenges and benefits which are explored in this chapter.

2.1 Procedural content generation

The field of procedural content generation is evolving rapidly. Most recently developed algorithms and projects are however largely dependent on a few basic techniques.

L-systems

L-systems, or Lindenmayer-systems are a type of a formal grammar. They were introduced by the Hungarian theoretical biologist Aristid Lindenmayer in 1968 [Lin68] to describe the behavior and growth process of plants. In procedural modeling, this technique is used to generate realistic plant models as well as self-similar fractals.
An L-system is a parallel rewriting system: Starting with an initial string, symbols are iteratively replaced by applying production rules. An L-system consists of:

- An alphabet of symbols including terminal and non-terminal elements
- An axiom, or initiator string, containing symbols from the alphabet defining the initial state.

- A set of production rules, transforming a string of symbols from the alphabet into a successor string.

Starting with the initial string, in each iteration all symbols are derived in a parallel manner. When a production rule matching a symbol exists, it is replaced by that rule's output symbol. The system terminates when no more symbols can be changed. For the creation of 3D objects, additional procedures are required to translate terminal symbols into 3D shapes. Parametric L-systems additionally allow attributes to be assigned to symbols. This allows to add additional constraints to production rules based on attribute values. A vast amount of procedural modeling projects either directly utilize L-Systems or a language that is based on them.

Shape grammars

Shape grammars are a type of production systems for the generation of geometric shapes. They were first presented by George Stiny and James Gips in 1971 [SG71] to describe paintings and sculptures. The formalism has since then been adapted and utilized in various fields such as architecture, industrial design and engineering.
Shape grammars are closely related to L-Systems, but instead of working with strings, production rules are directly applied onto geometric shapes. A shape grammar consists of a collection of shape rules as well as a generation engine which steers the rule selection and application processes. Shape rules describe how shapes are transformed. Each rule consists of an input and output side, both sides containing a number of shapes as well as a marker. The marker is a non-terminal symbol which helps to orient and position the new shape. A minimal shape grammar contains a start rule, at least one production rule and a termination rule, which is required to stop the process - this is usually done by removing the marker.

2.1.1 Recent developments

Based on basic techniques such as L-systems or shape grammars, numerous procedural modeling applications based on formal grammars have been developed. In this section a few of the most influential contributions are presented.

CityEngine

Parish and Müller [PM01] introduce CityEngine, a procedural modeling framework for the creation of 3D scenes of cities (not to be confused with ESRI's CityEngine, its indirect successor based on a different approach). The system is user-controllable and is capable of creating entire urban environments from geographical and statistical input data. This includes the generation of road networks, distribution of structure space as well as the creation of building models.
In the scope of this thesis, we are mostly interested in the building modeling process. CityEngine uses a procedural modeling system based on parametric L-Systems. For the

modeling of a building, an arbitrary ground plan is used, generated from a previous step in CityEngine. The bounding box of the structure, created from the ground plan, is used as axiom. In successive steps the system selects and applies suitable rules to the shape, refining, transforming it, or splitting it into multiple shapes. In each step, the building model gets more detailed. As a result, the level of detail is easily scalable.
Although CityEngine produces a variety of visually complex building types, the researchers note that these simple rules are not sufficient to represent actual functionality of buildings.

Split grammars

Wonka et al. [WWSR03] present a tool for automatic modeling of architecture. To derive building models, the researchers introduce split grammars, a new type of parametric set grammar. The idea of split grammars is to make them both powerful and simple, to produce a large variety of results but also allow to a controlled and automatic derivation. The split grammar is used to derive a 3D model of a building, starting from a simple initial shape (e.g. a cuboid, cylinder or prism). The grammar then decomposes basic shapes into individual parts such as windows, doors or ornaments to model detailed building facades. Attributes are assigned to the shapes to provide semantic information used to assign geometry and material in the post processing step. Interleaved with the split grammar, the system also uses a control grammar which steers the derivation process.
Production rules are stored in a large database of grammar rules and described with attributes. In a derivation step, the split grammar searches the database for rules with attributes matching those of the current shape. A rule is selected and applied to generate new shapes. Attributes for new shapes are inherited from the previous shape. The control grammar is used for calculates and assigns attributes to the shapes based on additional design constraints.

CGA shape

Müeller et al. [MWH$^+$06] introduced CGA shape, a type of a symbolic shape grammar developed for the generation of computer graphics architecture.
CGA shape is an extension of split grammars introduced by Wonka et al. The system refines basic shapes by sequentially applying production rules. In practice, CGA shape is used to model detailed buildings including facades, windows, doors and ornaments.

The production process with CGA shape starts with a configuration, a finite combination of shapes, which is used as axiom. Sequentially, shapes from the configuration are selected and matching production rules are applied. The resulting shape is then added to the configuration. This is repeated until no more non-terminal shapes exist.
Depending on the detail, a priority is assigned to each rule. This ensures a controlled derivation from low detail to high detail. In the modeling process, semantic information

such as the hierarchical structure and annotation are specified and stored, allowing the system to reuse design rules for different buildings.
The researchers achieved an unprecedented level of detail for large scale building models. CGA shape was developed with the goal in mind to improve the achievable complexity and detail of generated Buildings in urban modeling systems such as CityEngine.

2.2 Recommendation systems

Recommendation systems or recommender systems emerged as a subclass of information retrieval or information filtering systems [FBY92]. The basic goal of these systems is to manage information overload. When a user is confronted with a large amount of either unwanted or redundant data, information filtering systems aim to retrieve the most important pieces of information while discarding anything deemed unnecessary. This is basically as a binary classification task: Any unseen piece of information is classified as either relevant or irrelevant to the user. The classifier is based on a training set of items that the user found interesting.

2.2.1 Classes of recommendation systems

Recommendation systems predict the rating that a user would give to an item, or any piece of information. In order to determine the predicted level of importance of information, the system evaluates the user's characteristics. Depending on the type of data model used for the rating prediction, we distinguish collaborative filtering and content based filtering [FJN+14].

- **Collaborative filtering** focuses on the characteristics of multiple users. The system collects preferences and statistics from a large set of users to find matching interests. This approach is based on the assumption that if *user X* share's *user Y*'s opinion on *item A*, it is more likely that *X* will share *Y*'s opinion on *item B*, than that of a random person.
  The basic process to predict the rating of an active user for a specific item works in two simple steps:

  1. Find users with a similar rating pattern as the active user (i.e. Users with common interests).
  2. Calculate a predicted rating for the active user from the ratings of the found users.

- **Content based filtering** focuses on item profiles. The system describes items with keywords and attributes and constructs user profiles based on keywords and attributes of items the user previously rated. This means that if a user assigns a rating to a certain item, the system assumes that he would assign a similar rating

to an item with a similar description.
To describe and item, relevant features have to be found and abstracted. Depending on the type of data, there are various algorithms which can be used to identify the most important features and present the item in a vector space.
The user model is constructed based the user's history of rated items and interactions with the system. The user's profile is represented as a weighted vector of item attributes. The system uses these weights to calculate a prediction for an item.

In many applications, both collaborative and content based models are combined to a hybrid approach. Recommendation systems are used in a wide area of fields, most commonly on web applications to list specific items to a user either automatically or as a result of a search query. These applications include movies, music, advertisement, products, news articles, social contacts and many more.

2.2.2 Relevance feedback

Relevance feedback is a technique in information retrieval (IR) to improve search queries and was first described in 1971 [Roc71]. The basic concept is to utilize a user's feedback on existing search results to refine the search query and therefore provide more relevant results in consecutive searches. An information retrieval process utilizing a basic relevance feedback algorithm consists of the following steps:

1. The system received an initial search query Q and returns a set of results
2. The user provides positive or negative feedback by flagging results as relevant or irrelevant
3. The IR system calculates a modified search query Q' where (depending on the used model) the weights are shifted towards the relevant items and away from the irrelevant ones.
4. A new set of results is retrieved based on Q' and returned to the user.
5. The user is again asked for feedback on the new set of results (return to step 2)

2.2.3 Implicit and explicit user feedback

A recommendation system generally distinguishes between implicit and explicit user feedback [RRS11].

Explicit feedback is retrieved from direct user input. Commonly the user is asked to rate a specific item on a scale or to simply 'like' or 'dislike' an item. The collected data from explicit user feedback is easy to process, as a record only consists of references to a user and an item as well as a numerical value representing the given rating. An obvious

disadvantage is the additional required effort for the user to provide feedback. Therefore, feedback can only be sparsely collected.

Implicit feedback can be retrieved by analyzing the user's behavior. The recommendation system makes assumptions of the user's interests depending on what items he or she viewed. Implicit feedback is more convenient for the user but it is more difficult to implement, as the data is much more complex than in the explicit case [OK+98].

2.2.4 Related work

Very few applications of information filtering techniques in the area of procedural modeling have been researched in the past. In the field of computer graphics, content based filtering has mostly been used for image and shape retrieval.

Content based image retrieval

In 1997 Rui et al. [RHM97] introduced MARS, a content based image retrieval system. The researchers aimed to combine developments from the fields of image processing systems and information retrieval. While the extraction of information from an image is inherently different from the processing of 3D shapes, content based image retrieval serves as an example for the filtering of visual content within a recommendation system.

The information filtering in this work is based on the vector model, using term weighting and relevance feedback. This means that different weights are assigned to different keywords of the document, and additionally items are weighted by similarity. The researchers developed methods to translate image data into documents suited for their information retrieval algorithms. Related to the system proposed in this thesis, MARS was a first step for the integration of information retrieval into a computer graphics application.

3D shape retrieval

With an increasing number of 3D models available, various techniques for the retrieval of 3D shapes have been developed [TV08]. These methods solve the goal of retrieving shapes from a large database, via browsing or via queries, which are directly specified or constructed from a reference 3D model. The main tasks for shape retrieval systems are to extract and manage suitable descriptors from 3D models to measure similarities.

CHAPTER 3

Concept

The goal of this project is to include the concepts of recommendation systems and user centered design into the existing modeling framework, to benefit both from automatic generation and user feedback. In order to achieve this, the rule selection process has to take the user's preferences into account. This requires a method to collect and store user feedback and to incorporate it into the rule selection process. The proposed concept as well as certain limitations of the current framework introduce a number of challenges which need to be addressed.

3.1 Scenario

The system was developed with a few scenarios in mind, which greatly influenced the chosen architecture and methods.
In the chosen modeling language, the rule selection is mainly controlled with a simple goal notation. Generally, a user will define various axioms with a small number of goals each. The goals describe the type of object this axiom is supposed to be transformed in, i.e. the desired end result. In order for proper models to be created, appropriate rules are required to process these axioms. These rules are defined within different grammars and stored in a database, from where they can be retrieved in order to apply them to a shape.
The system is designed to allow multiple users to work on the database, adding and retrieving rules to the same global grammar. When users attempt to complete the same practical tasks with their grammars (e.g. two users independently want to model a chair) it is assumed they will use the same goals for the axioms. As a result the same set of rules will be applicable in their gramars. This set of rules could also include newly created rules from one of these users' grammars. Multiple users would therefore affect each others' grammars, wheter this side effect was intended or not.

Additionally the results of an existing grammar might differ from its results at a later point in time because new rules could have been added that fulfill goals that are used within the grammar. Newly added rules could completely change the rule selection process for an axiom of an existing grammar.

This creates a scenario where the results of a grammar might differ from the user's expections, depending on other users' contributions. It also cannot be guaranteed that a grammar that created pleasing results at one point will still satisfy the user's need at a later point in time. In this scenario a number of challenges arise.

3.2 Challenges

To keep the results of a certain grammar deterministic over time, it would be necessary to maintain older grammars, which could prove to be unsuitable in a lot of cases. In order to guarantee deterministic results, every version of a grammar that was successfully parsed at one point in time would have to be preserved. The system would essentially have to manage vast amounts of versions for most grammars in the database, since every newly added rule could affect the results of existing grammars. As a result, the biggest challenge for such a large scale system is to allow existing rules to evolve with the increasing knowledge of cloud. Previously created rules need to adapt to future changes without changing their source code.

In the most basic case, the rule selection process is controlled by the user via simple goal declarations. In order to improve this process beyond the context of a simple goal notation our goal is to fine-tune the rule selection by including all the available semantic and statistical information.

In cases were a user requires specific results for the creation of a generic object within this global system, it is necessary to ensure that the system provides results in a way that considers the user's specific needs. When a user models a scene of an airplane interior, he will not be pleased with a row of classic dining chairs. In the same context, this user's custom rule for the creation of a plane seat must not disturb the creation process of a dining room. It is necessary for the system to know the suitability of rules in specific situation in addition to the general context of goals.

In a large scale system, these challenges cannot be seen as fixed constraints but rather dynamic requirements, that grow with the scale of the database of rules. Ideally, the cloud would learn the optimal behavior for the selection of rules along with the addition of new rules, goals and grammars. Our biggest challenge is to find a suitable learning process for this task. We need to take numerous factors into account in order to find the best suited solution: It needs to be determined how the learning process would look like and what data it is based on.

In the scope of this system, a verification for the learned knowledge is required. The quality of a created shape in this system is mostly based on the user's preferences. While

the system must provide contextually relevant information, it is entirely possible that the user himself or herself does not have an optimal result in mind and wants to try creating different results until he or she is satisfied. In such a system there is no suitable way of verifying a result without manual user input. Therefore, unsupervised learning is not a suitable choice. The goal is to find a level of supervision that minimizes the required user input but provides enough information to create fitting results.

3.3 Approach

In order to master these challenges, numerous concepts are introduced in the proposed approach. The core concept of this work is to provide a dynamic interface between the user and the cloud that observes the user's natural behavior, while providing suitable results. The system extracts feedback from the observed user interaction in order to learn from his or her behavior. This idea is the central part of this work.

3.3.1 Learning approach

In the development process of this system numerous approaches for a learning process which fulfills the proposed requirements have been considered, such as reinforcement learning, or artificial neural networks [Bis06]. Generally, the decision comes down to the type of data that is available, as well as the level of supervision that is required. The data used in our model mainly consists of semantic descriptions of shapes. It is a mostly unstructured mix of goals, tags and attributes with variable length. This makes it difficult to extract fixed-sized feature vectors which are required in various approaches, such as neural networks. In addition to the complexity of the data, the knowledge of the system constantly evolves. Therefore, a train-once-use-forever approach is not applicable for this scenario. The definition of training data would be an additional challenge for such an approach. The correctness of a solution can only be determined manually by the user after creation. Filling a set of training data would take a large number of iterations until a proper decision model can be built.

3.3.2 Recommendation system

The chosen approach for this work is based on a recommendation system (as described in Section 2.2). This decision was made based on several factors: When a user creates a 3D-scene, he or she is not limited to any scope by the system. There are no predefined constraints for the content of a scene. Our goal is to allow the user to model whatever he or she wants. The item filtering process is based on the user's preferences, not on any general classification. Therefore, the system should help the user to find his or her preferred solution rather than limit the possible solutions. As the only reliable way of determining the quality of a resulting shape is to find out the user's opinion, there has to be some level of supervision. The dynamic nature of the cloud requires the information about the user's preferences to be constantly updated and expanded. A recommendation

system fits both these requirements. The classification of objects in our system directly corresponds to user feedback.

A recommendation system either relies on implicit user feedback based on observation of the user's behavior, or explicit user feedback, where some level of direct input is required.

Implicit approach

In the context of this system, the implicit collection of user feedback would revolve around the usage statistics of grammars, goals and rules. At an earlier stage, the probability model for rule selection was based on an implicit user feedback approach. This means that any grammar that was parsed at one point by the user was considered for the calculation of weights during rule selection. The probability of a rule was based on the rate of application of similar rules in a similar context (i.e. a shape with similar goals, attributes, etc.). So whenever a grammar was parsed, the weights for rule selection shifted. An obvious advantage of this approach was that the user would not have to be bothered with providing feedback. In theory this should speed up the workflow and overall make the program easier to use. However, this approach proved to cause one major problem:
Rules were weighted depending on their usage, which could lead to rules dominating the system when their grammar was parsed repeatedly. This proved to be very counterproductive when a grammar was still work in progress and the created objects were not actually desired results. Modeling 3D shapes is after all a complicated task, we cannot assume that a created object does not contain any unwanted design flaws. As a result, this purely implicit approach was rejected.

Explicit approach

In order to improve the shortcomings of the implicit approach, it was decided to rely on user input for the rule probability calculation. This is necessary because there is no reliable way to determine the quality of a grammar's result based on the user's behavior. We cannot extract any meaningful information only from the usage statistics of rules. A rule that was applied a large number of times, is not necessarily perfectly suited for the context. We need more (reliable) information in order to rate and classify the created result as well as the applied rules.

For the collection of explicit user feedback we need to clearly define both the type of feedback and the corresponding items, which the feedback is based on. The most basic choice for an item is the resulting shape produced by a grammar. A user could rate this shape and through that influence the probabilities for the creation of the next shape. This solves the classification problem from the implicit approach, but it does not yet meet our requirements. As the creation process for a shape consists of multiple iterations where different rules are applied, we decided on a more detailed level of feedback: Instead of rating the result, a user is asked to rate the application of a single rule. This additional

information makes it much easier to extract the probabilities for the rule selection process from the user feedback. However, this approach comes with a number of challenges: Rating the applied rules instead of the results is a lot less intuitive for the user and requires more detailed visualization of data and a more advanced user interface. The proposed approach provides a basic visualization of a shape's derivation tree with various possibilities for user interaction. In the future, this could be expanded to improve the usability for providing feedback.

Combining implicit and explicit feedback

Although the downsides of the proposed approach based on explicit feedback outweigh the benefits of implicit feedback, we want to improve the usability by combining it with implicit feedback. By integrating the manually applied feedback into the modeling workflow, the explicit feedback is no longer an additional task, but instead becomes a part of the actual modeling process. Our approach could be seen as a hybrid solution between explicit and implicit user feedback, with the goal of providing a high level of usability while retrieving meaningful and clear feedback data.

Content-based filtering

The system in its current state does neither distinguish between different users, nor does it store any user-specific data. The probability calculation for the selection of rules (which represent the items in our recommendation system) is based on their context, which is the shape they are applied to and the previously applied user feedback (i.e. to the particular rule in a similar context), disregarding any information about the user himself. This strategy is based on content-based filtering described in Section 2.2. The decision for content-based filtering rather than collaborative filtering was very easy to make, considering our focus on items (i.e. Rules) rather than different users.

3.3.3 Parse tree

In order to collect user feedback in the proposed hybrid manner, a visual representation of the scene's **parse trees** is introduced. A parse tree is a structure representing the exact order of applied rules responsible for the generation of a 3D shape in the scene. For each axiom, the root node of a tree is generated. Inner nodes of the tree represent intermediate derivation steps. Each child node represents the result of an applied rule to its parent node. The leaves of the trees represent the terminal shapes (i.e. the generated models which are rendered on screen).

Locking operations

The feedback is collected from a user's interaction with the generated parse tree. Following the hybrid feedback approach, a set of novel methods are introduced to integrate the feedback collection process into the modeling work flow rather than having to rely on explicit feedback.

- **Lock to root**. By locking a node of the parse tree, the user tells the system to keep this rule in place for the next derivation of the scene. As a result its preceding nodes up to the root are locked as well. This corresponds to positive feedback on the applied rule and its predecessors in the parse tree.
- **Lock descendents**. For additional positive feedback, locks can be extended to include all children of a node down to its leaves. This locks an entire sub-branch of the tree.
- **Reject**. By rejecting a node in the parse tree the scene will be regenerated. For this derivation, the rule corresponding to the rejected node will not be considered. This operation provides negative feedback for the affected rule.

CHAPTER 4

Technical Details

The existing system consists of a few tightly coupled components:
The **procedural modeling framework** parses user input (i.e. a grammar of rules, axioms or other code), handles rule application and visualizes the resulting 3D shapes.
The **rule selection** component is used in every derivation step of the grammar to choose from matching rules based on their compatibility to the current shape. At this point the feedback data is consulted to fine-tune the rule selection based on the current context. A probability is calculated and assigned to each matching rule, before one of them is randomly chosen.
An additional component is introduced to handle the provision of **user feedback**. After generating a 3D model, a visual representation of the derivation steps which resulted in the model is shown. This is where the user is able to provide explicit feedback on the created model and the rules that have been applied. This information is used to improve the results of future iterations.
In this chapter, these contributions and the used algorithms are described in detail.

4.1 Feedback

In this project, the selection of rules from the database, as well as the determination of goal sequences is dependent on stored user feedback. This section gives an overview how this feedback is collected.

4.1.1 Parse tree

When generating a scene from a grammar, the system constructs a parse tree for each axiom. This tree represents an axiom's derivation steps. Starting with the initial shape of an axiom, every time a rule is applied to a shape, a corresponding node is added to

the tree. Rules which result in multiple shapes (e.g. split, repeat) have multiple children at their respective node in the parse tree. After the final scene is created, the parse tree shows all applied rules from the initial shape as root, down to every rule which resulted in a terminal shape as leaf. The parse tree is visualized in the UI and allows the user to provide direct input on his perceived quality of the rule selection for the current grammar. The user can interact with the nodes within the parse tree to modify the conditions for rule selection in the next derivation of the grammar.

4.1.2 Locking

Locking allows a user to select a node in the parse tree, and lock it's state for the next derivation. This provides feedback for future derivation steps on three separate levels.

1. When a node in the tree is locked, the system is forced to re-use the corresponding rule when the grammar is parsed again. In order to ensure that a rule within the parse tree is used again, all it's preceding rules in the tree have to be locked as well, up to the initial shape of the axiom.
   In the next derivation process, the system will skip rule selection for any locked node in the previous parse tree, and will only select rules for those shapes which are unaffected by any locks. This means that the locked parts of the parse tree and its corresponding shapes will stay the same in the next derivation. This way a user can directly tell the system which parts (or rather steps within the derivation) of the object to keep and which to change.

2. When a user locks a node in the tree, the system stores information about this rule and the shape it was applied to in the database, along with a positive feedback value depending on the node's position within the tree. This information is later used in the rule selection process, to calculate the relevance of a rule within a certain context.

3. Locking a node will also store the order in which goals were fulfilled for its corresponding shape in the database. This information provides the base for the probability calculations for the determination of sequences for simultaneous goals.

In the proposed system, the user has multiple options how to lock a node. In the most basic case, the user locks a node in the parse tree up to the root of its axiom. This provides positive feedback for all nodes, while the actual weighted value decreases up towards the root.
Additionally, is is possible to extend an existing lock to the node's children. This operation will lock all descendent branches in the parse tree down to the leaves starting from the specified node. A constant positive feedback value is provided for all affected nodes.

4.1.3 Rejecting

In a similar way to locking, the user can also provide negative feedback on a created 3D shape, by rejecting a node within the parse tree. The reject operation is also referred to as 'regenerate' as it causes a branch of the derivation tree to be newly generated. This operation will lock every node except the rejected one and its decedents and force the system to start another derivation process on the grammar. In the next derivation steps, the system will select the same rules as before for the locked nodes and will be forced to use a different rule instead of the rejected one. As a result, all rules which were applied on descendants of the rejected node will be regenerated as well.
This operation allows users to change certain parts of a generated 3D object, while keeping everything else in place. A basic example would be a grammar for the creation of a house. The user is pleased with the resulting 3D object, except for one window. Therefore, he or she will look for the node or branch in the parse tree responsible for the creation of the window, and reject it. The system will keep the house as it is, but with a different rule selected for the window. This allows for more detailed modeling and allows the user to refine the results while keeping the grammar itself unchanged.
In addition to these constraints on the rule selection, rejecting a node will also store negative feedback on the corresponding rule in the database. Therefore, rejecting a node will not only affect recurring derivation processes of its local grammar, but also modify the weights for future rule selections for other grammars. When a user rejects an applied rule within a certain context, this will reduce the chance of this rule being selected on shapes in other grammars with a similar context.

4.1.4 Numerical feedback value

Performing one of the described operations on the parse tree will store a feedback record in the database for each affected node. Depending on the type of operation, different values will be applied. For each node, a feedback record consists of the applied rule, the shape it was applied to as well as a feedback value V in the range of [-1, 1]. Each node in the parse tree corresponds to a shape and an applied rule. For the different operations, the values are specified as:

- Reject: $V = -1$ for the rejected node.
- Lock to Root: $V = level(n)/level(m)$, where n is the node corresponding to this feedback entry and m is the node which the lock was originally applied to.
- Lock descendants: $V = +1$ for every descendant node.

Rejecting a node will only store negative feedback (-1) for the exact node it was applied to. Any descendants nodes are not directly affected, but the negative feedback on their parent node will of course indirectly affect their total probability. The lock to root operation will always apply a feedback of +1 to the node it was applied to, while for any preceding nodes up to the root the feedback values decrease down until a value of

$1/level(n)$ for the root node, where n is the node the lock was applied to, and the root node corresponds to the axiom and the rule that was first applied to it. Additional feedback via the lock descendants operation is always +1 for any affected node.

4.2 Rule probability

To determine the probabilities for the rule selection process, in addition to the feedback values that were applied to these rules, the relevance of the feedback has to be considered. When evaluating feedback entries for a rule, the current semantic context (i.e. the shape) has to be compared to the context where the feedback was applied. For this, semantic similarities between shapes have to be determined. The following calculations for the shape similarity as well as the total rule probability are adopted from the work of Ilčík et al. [IW16].

4.2.1 Shape similarity

The similarity between a pair of shapes is determined by comparing its semantics, which are separate sets of goals, tags and attributes. In this system, goals and tags are represented as string values, while an attribute is a pair of a key (string) and a value, which is either a boolean or numeric value or a string. Before a probability can be calculated, the similarity between the semantics have to be defined. For this, the Jaccard index [Jac12] is used, which describes the similarity between two sets A and B as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

Representing the semantics of a shape s, the sets of goals and tags are denoted as G_s and T_s. The set of string keys for the shape attributes with Boolean, numeric and string values are denoted as B_s, N_s, S_s respectively.

The similarity between goals and tags of a pair of shapes is defined as the weighted average of the Jaccard similarities between the respective sets.

$$sim_{GT}(a,b) = \frac{|T_a \cap T_b| + |G_a \cap G_b|}{|T_a \cup T_b| + |G_a \cup G_b|} \tag{4.2}$$

The similarities between attributes are limited to attributes which exist in the sets of both shapes. For Boolean values the similarity is defined as the ratio of equal elements to the count of all elements.

$$sim_B(x,y) = \sum_{k \in keys(x) \cap keys(y)} \frac{|x_k = y_k|}{|keys(x) \cap keys(y)|} \tag{4.3}$$

Numeric values additionally use a precomputed factor r_k which is the difference of the minimum and maximum values of the respective attribute within the feedback dataset.

$$sim_N(x, y) = \sum_{k \in keys(x) \cap keys(y)} \frac{|x_k - y_k|}{r_k |keys(x) \cap keys(y)|} \tag{4.4}$$

The similarity between string values is calculated from the average of the Jaccard similarities between the strings.

$$sim_S(x, y) = \sum_{k \in keys(x) \cap keys(y)} \frac{|J_{string}(x_k, y_k)|}{|keys(x) \cap keys(y)|} \tag{4.5}$$

The combined function for the semantic similarity between two shapes is defined as:

$$sim(a, b) = \frac{1}{Z} \left(\sum_{X \in \{T,G\}} |X_a \cap X_b| + \sum_{Y \in \{B,N,S\}} \sqrt{|Y_a \cup Y_b||Y_a \cap Y_b|} sim_y(Y_a, Y_b) \right) \tag{4.6}$$

with

$$Z = |T_a \cup T_b| + |G_a \cup G_b| + |B_a \cup B_b| + |N_a \cup N_b| + |S_a \cup S_b| \tag{4.7}$$

4.2.2 Probability calculation

The similarity function is used to determine the relevancy of a specific feedback record in comparison to the context of the current shape where a rule should be applied. Feedback entries on shapes with a greater similarity to the current shape are more relevant. The probability prediction for a rule r on a shape i where F_r is the set of feedback records regarding r is defined as:

$$P(i, r) = \frac{1}{2} \left(1 + \frac{\sum_{f \in F_r} sim(i, S_f)^2 * V_f)}{\sum_{f \in F_r} |V_f|} \right) \tag{4.8}$$

where S_f and V_f are shape and the value of the respective feedback record. The result is shifted to the interval of [0, 1]. For rules where no feedback records exist the result is 0.5. In the rule selection process for a specific shape, the probabilities for all candidate rules are calculated and normalized. The system chooses one rule at random based on their probabilities.

4.3 Goal sequences

When an axiom has multiple active goals, the system chooses the oldest one first. If multiple goals were added in the same derivation step the system will reach a state were there is more than one oldest goal. In this case, we have to determine the order in which these goals are to be processed. The order of the goals directly affects the order of applied rules and is therefore a crucial step in the creation of a specific 3D object.

As an extension of the previous example, we define an axiom with the goals "chair" and "wood". Now depending on which goal is fulfilled first, this could create completely different results. If we assume that the matching rules for "chair" model the shape of a chair, and "wood" results in a brown colored shape, then the order of goals decides whether the initial shape is first transformed into multiple shapes that form a chair, or into a piece of wood. In the first case, each created shape of the chair still has the "wood" goal. In the following steps a rule fulfilling "wood" will be chosen for each shape of the axiom. This might result in a chair made from different types of wood. In the second case, when "wood" is chosen first, a wooden shape with the goal "chair" is created. In successive steps, this wooden shape will be split and transformed into a chair. In general, we would assume this case to be more desirable. There might be a case however, were the first option is preferred. It is therefore important to find a probability model which allows all orders of goals, but assigns weights according to relevance and importance.

The basic approach to solve this problem is to determine a probability for each goal to come before or after another goal. This information is retrieved from user feedback on the order of goals in previous derivation steps. For each terminal shape (i.e. the end result of a parsed grammar) its sequence of fulfilled goals (i.e. the order in which the goals were fulfilled) is tracked. The stored sequences are weighted based on explicit user feedback on the resulting shape (see Section 4.1). When an axiom has to fulfill multiple goals simultaneously, a probability is calculated for each possible order for this particular set of goals. In further derivation steps the system will then attempt to fulfill goals in this order.
To determine a probability for each possible order of goals, the system first calculates the probability for pairs of goals based on their order and distance in tracked sequences (where the user has given feedback). The optimal solution for this problem would be to find the global maximum from all permutations. This means in a graph, with the goals as nodes and their pairwise probabilities as weights, we need to find the spanning path with maximal weight. While this computationally expensive operation could be approximated, it is not feasible for this problem. This is because we do not only need to find the globally optimal sequence, but the probability for all possible sequences, so we must not skip any nodes or paths within the graph. Therefore the system would need to aggregate all permutations of the set of goals, which has factorial complexity.
A far less complex approach that was considered, is to find a sequence by randomly (weighted with the pairwise probability) selecting the next unused goal until the se-

quence is completed. While this local approach is far less computationally expensive, it provides less desirable results (i.e. there is a notable difference between the local solution and the optimal global results).

For this project a hybrid approach was chosen: If the number of goals exceeds a set threshold, the sequences are calculated locally, up until the threshold. For all goals below the threshold, all possible combinations are evaluated. Even for complex grammars which create detailed objects, the number of simultaneous goals is way below the threshold and therefore it should be possible to calculate global results in most cases. For higher numbers the hybrid approach still provides decent results, while keeping the computational complexity at a reasonable level.

For the calculation of a goal sequence for a shape S and a set of goals G we define Q as the set of relevant sequence records (i.e. sequences containing at least two goals from G). The weight of a sequence record is defined as W_q. V_q is the feedback value assigned to this record.

$$W_q = sim(q, s) * \frac{|G_q \cap G|}{|G_q \cup G|} * V_q \tag{4.9}$$

The connectivity between two goals $a, b \in G$ is defined as:

$$con(a, b) = \sum_{q \in Q_{ab}} W_q \tag{4.10}$$

Q_{ab} is a subset of Q containing all sequence records where the goal a was assigned one derivation step earlier than b (i.e. b is a direct successor of a).

In global mode, the probability is assigned to each permutation of the set of goals G. The sum of connectivity values for a permutation p is defined as:

$$acc(p) = \sum_{i=1}^{n-1} con(p_i, p_{i+1}) \tag{4.11}$$

The final global weight for p is defined as:

$$W_g(p) = \begin{cases} \frac{-1}{acc(p)} & if\ acc(p) < 0 \\ 1 + acc(p) & else \end{cases} \tag{4.12}$$

In local mode, weights are iteratively calculated for pairs of goals. G' refers to the subset of goals $\in G$ which have not been completed yet (i.e. goals that are left to choose from) and g' is the goal that was previously chosen. We define the initial value of g' as a virtual starting goal that is always chosen first. For following iterations the local weight for each goal $g \in G'$ is defined as:

$$W_l(g) = con(g', g) + (1 - min) \tag{4.13}$$

min is the minimum weight of the elements of G'.

CHAPTER 5

Implementation

The client software is implemented in C# and is in its core an offline version of the Michelangelo web system. A modeling system consisting of a text-based grammar editor, a 3D window and basic controls is provided. A local database system stores all the cloud based information. In this chapter, the user interface, the architecture of the system as well as various use cases are described.

5.1 User interface

This implementation features a basic graphical user interface (5.1) to interact with the program logic. The GUI itself was developed mainly for research purposes and is not intended for large scale use. Instead it provides a single user a quick and easy access to the grammar-database and provides visual elements to display created results and allow the necessary user input.

The system's main window consists of several components: A 3D window to render the created shapes, a text area for the grammar's code, various control elements for database access, as well as a visualization of the parse-tree, to perform locking operations.

5.1.1 3D window

The 3D window provides basic mouse controls to modify the (orbiting) camera and allows the user to view his or her creations from different angles and positions.

5.1.2 Code area

On the left hand side of the window, a text area shows the current grammar's code and allows user to view, modify and create grammars for the use in this system. Any errors that occur when the grammar is parsed will be shown at the top of the text area.

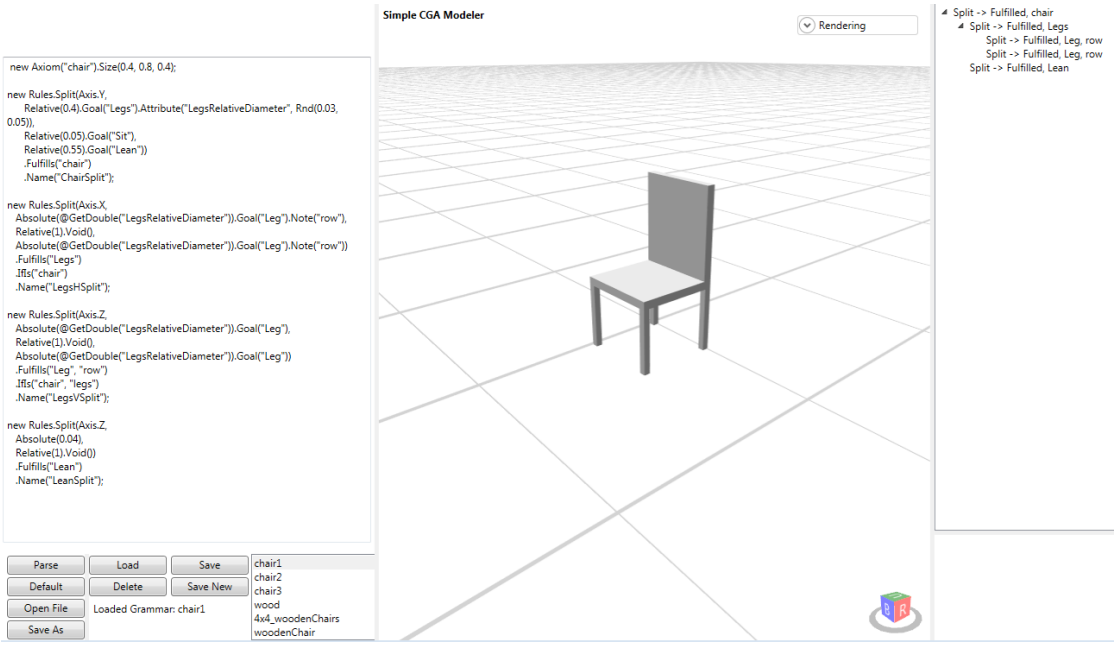


Figure 5.1: The graphical user interface of the client program

5.1.3 Control area

The control area below that grammar code features a list of the currently available grammars in the database as well as a set of buttons to perform basic operations to allow the user to access the database.

- **Parse**. This executes the current grammar script.
- **Default**. Restore the default grammar text.
- **Open File**. Load a grammar script from a file.
- **Save As**. Save the current grammar script to a file.
- **Load**. Load the selected grammar from the DB.
- **Delete**. Remove the selected grammar from the DB.
- **Save**. Save changes made to the selected grammar to the DB.
- **Save New**. Save the current grammar to a new DB entry.

5.1.4 Parse tree

Once a grammar has been parsed, a tree view will be visible on the right side of the screen. In this area, the derivation process of the parsed grammar is visualized in the form of a parse tree. Every axiom defined in the grammar is represented in the tree by a root element. Any child elements represent a derivation step result, i.e. a shape as well as the rule that was applied to it. The leafs of the tree are the terminal shapes which

are shown as visible objects in the 3D-window. Right clicking any of the tree nodes, will give access to the different locking operations described in Section 4.1.2. Performing said operations will modify the tree accordingly. Active locks in the tree are visualized with differently colored text. Locks can be manually removed the same way they were added. Rejecting a node will cause the grammar to be re-evaluated immediately.

5.2 Use cases

The client program containing the graphical user interface serves as an entry point for the system's global grammar stored in the cloud. The user interface is designed for advanced users with a programming or scripting background, rather than artists. In-depth knowledge about the ACGAX language is not required, but it is recommended to read the provided tutorials.
Apart from the complexity of the language itself, the basic work flow for the creation of 3D-Objects is very simple:

1. Create a grammar, by defining axioms, rules and corresponding goals.
2. Parse/execute the script to view the result and the parse tree.
3. Provide feedback to refine the derivation process in future iterations.
4. Repeat the process at step 1 or 2 to either edit the grammar script or re-create results based on the modified probabilities from the retrieved feedback data.

5.2.1 Common use cases

For the general usage of the client software a number of actions are required. In this section the common use-cases to create and execute grammars and provide feedback on created results are described in detail. All use cases refer to a single-user scenario, since in the system no users will directly interact with each other. While the decisions (e.g. feedback, goals, etc.) of a user will indirectly affect other users, they do not interact with each other. With each parsed and executed grammar, the cloud data is retrieved (and updated, whenever user feedback was added).

Creating a grammar

This scenario occurs, when the program is started and connected to the database. By default, the code window is filled with a predefined grammar script. The user can start editing this script, or load an existing grammar script. These can be loaded from a text file, or directly from the database. After the user has finished writing his grammar script, he or she can either save the script to a local file for later use, or parse the script to use it in the system. Storing a grammar in the database, and thereby inserting its rules into global pool of rules in the cloud will automatically start a parse attempt first. Parsing a grammar will check for correct syntax and notify the user of any detected

errors. When the grammar script was successfully evaluated, its rules will be loaded into the database. A grammar that has not been saved to the database will exist as a temporary grammar. In this case its rules can be used locally, but will not persist for different grammars in the future. The common approach for a user to create a grammar is to work with temporary grammars and local files until a (to some degree) satisfying solution is reached. Working grammars can be saved to the database after naming them.

After successful evaluation of the grammar, the derivation of its defined axioms is started. If the parsed grammar does not contain any axioms, no derivation takes place and no visual results are created. In that case a grammar can still provide rules for the databases. This process results in a 3D-model shown in the respective window as well as the corresponding parse tree. The user can observe the created shapes and perform edits to the grammar if needed. When the user is satisfied with the general appearance of the created result he or she can continue with locking.

Locking

After a grammar was successfully parsed and a 3D object was created, the user can perform advanced edits via the locking operations described in Section 4.1.2. The controls to perform these actions are accessible in the context menu of the parse tree. The user can perform either *lock to root*, or *regenerate* on any of the nodes. *Lock descendants* requires a *lock to root* to be applied first.
Locks are visualized with colored text in the parse tree, and can be removed via the context menu. Placed locks will persist over multiple parsing operations until they are removed or the grammar is edited.
The regenerate operation automatically starts another derivation process and does not persists for further iterations. It is currently not possible to reject multiple nodes simultaneously. Regenerating an inner node of the tree, will also cause its descendants to change, since the original rules might not be applicable anymore in the changed context.

5.3 Program architecture

The client software is implemented in C# and is composed of three, tightly coupled major components. The GUI and the grammar parser handle the user interaction and the parsing process of the grammar. Construction of the parse tree as well as user feedback retrieval are also part of this component. The rendering of the created shapes relies on various external libraries which will not be explained in detail. The database client and cloud interfaces are responsible for storage and retrieval of information to or from the cloud. The modeling language's functionality is enclosed within the ACGAX component. This is where the derivation of a grammar, as well as the application of rules and manipulation of shapes their semantics occurs. The database system itself can be seen as an exchangeable independent component.

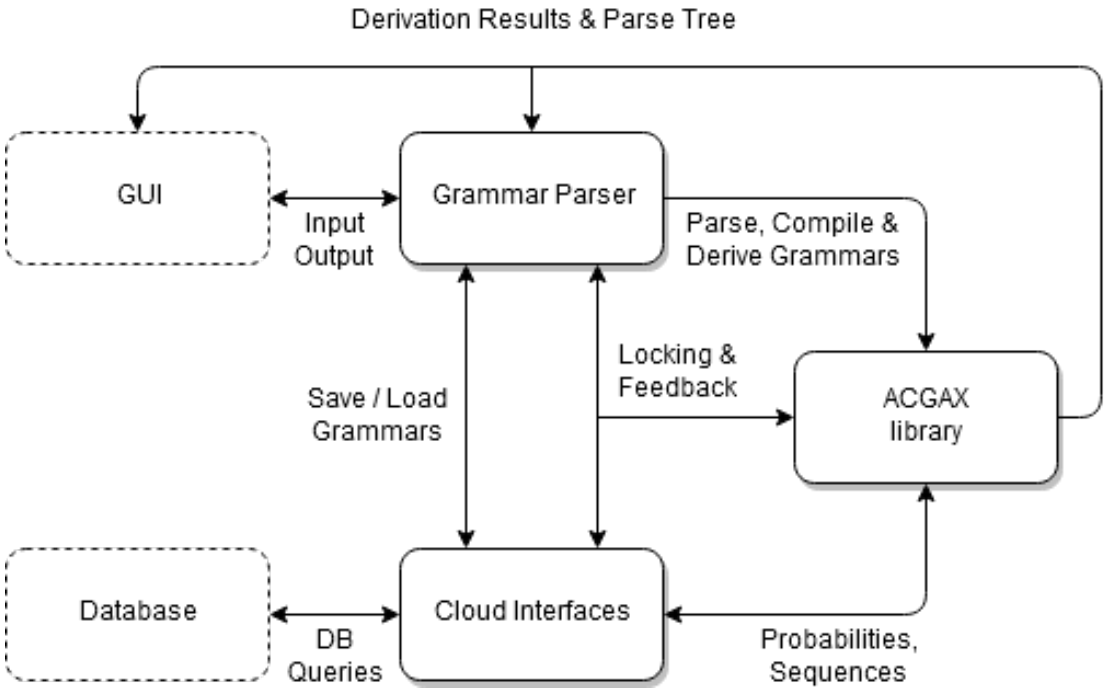


Figure 5.2: Overview of the system's components and their interactions

5.3.1 Grammar parser

The grammar parser can be seen as the main component of the program. It handles all the communication with the other components of the system and is directly coupled with the user interface. It's tasks are both the management of grammars as well forwarding requests between the database and the user.

Any user input in the GUI is evaluated in this component. User performed requests including database access or a parse or feedback operation are evaluated and forwarded to the other components accordingly.

Grammar management

The grammar parser component initiates and steers the derivation process of a grammar. When a user sends a parse request via the GUI, the grammar parser accesses the externally stored ACGAX language definitions to parse the code and build the grammar. This is also where the system checks for errors in the grammar. During these processes, the system creates a grammar object from the script. This means that scripting commands such as loops are transformed into actual rules or axioms. When the parsed grammar is a previously existing grammar that was modified, the system checks for differences to its previous version stored in the system. This step is performed to evaluate if the corresponding existing feedback records are still relevant to the rules of the modified

grammar. At the end of the building process, the grammar is compiled to a DLL file and inserted into the database. Granted the previous steps were successful, the created grammar's derivation process is started.

After each successful derivation, a parse tree is returned from the ACGAX component. For this tree a corresponding tree view GUI element is constructed and rendered. Locking operations performed by the user are stored with the previous parse tree. In the next parse step, the component provides the parse tree including any locking information to the ACGAX component for derivation of the grammar.

5.3.2 ACGAX

The ACGAX component contains libraries with all the information and definitions of the modeling language required to parse and execute a grammar script. This information is used in other components to perform the necessary operations. Additionally, this component contains classes representing the elements of the ACGAX language within the system, most notably grammars, shapes and rules.

Grammar evaluation

Grammars are the central items of the modeling language and contain definitions for axioms and rules. The derivation of a grammar is the main task of this component.

The axioms of a grammar are the starting points for the derivation of a grammar. Any grammar can contain an arbitrary amount of axioms. The derivation processes of different axioms are completely independent of each other. At least one axiom must exist to evaluate a grammar.

During the evaluation process (Figure 5.3) the system maintains a list of active axioms. Each axiom starts as active, containing it's initial shape. In each iteration, an active axiom is selected and it's next derivation step is executed. When this step finishes the derivation of the axiom (i.e. no rule is applicable to the shape or the number of maximum derivation steps has been reached), its shape is added to the set of terminal shapes (i.e. the evaluation results of the grammar) and the axiom becomes inactive. The evaluation of the grammar is finished when there are no more active axioms. The resulting shapes, alongside the created parse tree, are returned to be visualized in the GUI.

Derivation step

During the evaluation of a grammar, an axiom is stepwise derived. In other words, rules are applied to the shapes of the axiom.
As the number of shapes in an active axiom can be changed by some rules, an axiom might have multiple non-terminal shapes. In each derivation step, one of these shapes is chosen at random. The application of a rule is based on matching a single goal. As a shape might contain multiple goals, the system attempts to match the goals in order of

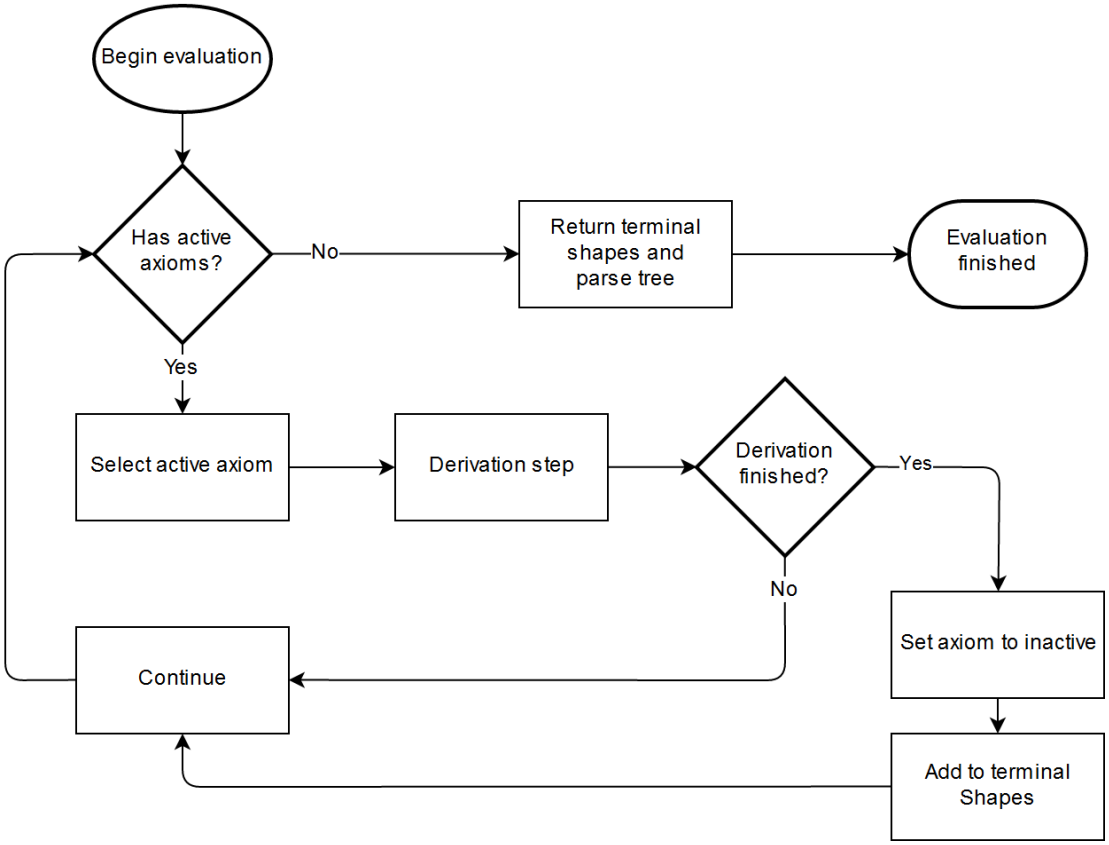


Figure 5.3: Overview of the evaluation process of a grammar.

their priority. As described in Section 1.5.2, oldest goals are matched first. When the matching was successful (i.e. at least one rule was found), the system continues with the rule selection, otherwise the next goal in order is matched. When no more goals are left to match, the shape is finalized and the derivation is over. When no more non-terminal shapes of an axiom are left, it becomes inactive.
A special case occurs, when a shape has multiple highest priority goals that originated from the same derivation step. In this scenario, the system retrieves an optimized ordering for goals, based on the calculations described in Section 4.3 (i.e. Based on user feedback, certain orderings are preferred). The system stores the sequence for this axiom and attempts to match the first goal of the sequence. When no match was found, its successor is next. If none of the sequences' goals could be matched, newer goals are considered. The calculated sequence persists and will be evaluated again in the next derivation step.

Rule selection

When the system attempts to find matching rules for the highest priority goal, it will first search for locally stored rules from the current grammar. When no rule was matched, the system will attempt to retrieve rules from the cloud. In either case, the system has to evaluate whether a rule is applicable in the context. For this task, the system compares the respective shape's semantics with the available rules. Generally, only rules which fulfill the required goal are applicable. In addition, the constraints and conditions of those rules must match the shape.

After a set of matching rule was retrieved for a selected goal, the system will determine the probabilities of these rules and pick one, based on a weighted random selection. The probabilities are calculated according to the functions presented in Section 4.2.2, depending on the existing feedback values for the particular rule. For the application of the selected rule, the system performs the following steps:

- The shape which the rule is applied on will have its semantics modified accordingly. The goal that was being fulfilled is added to the list of tags and removed from the set of active goals of the shape. If the particular goal was part of a goal sequence it is removed from the corresponding sequence.
- A parse tree node is created for the shape-rule pair and appended at the proper position in the parse tree. The node's parent is the node corresponding to the rule that this shape was created from, or the tree root, if its the first rule, directly applied to an axiom's shape.
- The actual rule application is performed. This task is performed by the ACGAX library and returns a set of shapes. These shapes are the results of the of rule that was applied, with their geometry and semantics set accordingly. Depending on the type of rule the returned set may consist of multiple shapes, which are all added to this axiom's set of non-terminal shapes.

5.3.3 Cloud interfaces

The cloud interfaces component contains all the logic necessary for communication with the database. This module also contains the implementation of the described algorithms for the calculations of rule probability, shape similarity and goal sequences. The database system used for this project is MongoDB, an open source document-oriented database program. Item records stored in the database are represented as JSON-like documents. Our system includes a MongoDB database client which provides all queries to store and retrieve items to or from the database. The database consists of various collections that are managed in this component:

- **Grammars**: The identification and code of all grammars are stored here, for quick and easy access

- **Rules**: Likewise, rules are stored in the DB. This collection is used to identify rules within the code of a grammar script.
- **CloudSemantics**: This collection includes all the semantics of rules in a grammar and is used for the matching of rules.
- **CompiledGrammars**: The compiled grammar DLL files are stored here.
- **Feedback**: This collection contains all records of user feedback which are created by locking operations. The probability calculations for rule selection depends on this data.
- **Sequences**: The sequence records store the order in which goals were fulfilled within a shape. This information is required to determine the most fitting goal sequences for the derivation of an axiom.

The cloud interfaces component contains the necessary operations to retrieve these records from the database, or to provide new information for the cloud. The JSON schemata of selected documents are shown in listings below (5.1, 5.2, 5.3 and 5.4).

Listing 5.1: Feedback

```
"Rule": {
  "type": "string"
},
"value": {
  "type": "integer"
},
"Goals": {
  "items": { "type": "string" },
  "type": "array"
},
"Tags": {
  "items": { "type": "string" },
  "type": "array"
},
"Attributes": {
  "items": { "type": "string" },
  "type": "array"
},
"AttributeValues": {
  "items": { "type": "Object" },
  "type": "array"
}
```

Listing 5.2: Sequences

```
"Goals": {
```

```
  "items": { "type": "string" },
  "type": "array"
},
"Time": {
  "items": { "type": "integer" },
  "type": "array"
},
"Value": {
  "type": "integer"
},
"sGoals": {
  "items": { "type": "string" },
  "type": "array"
},
"sTags": {
  "items": { "type": "string" },
  "type": "array"
},
"sAttributes": {
  "items": { "type": "string" },
  "type": "array"
},
"sAttributeValues": {
  "items": { "type": "Object" },
  "type": "array"
}
```

Listing 5.3: CloudSemantics

```
"Grammar": {
  "type": "string"
},
"Goals": {
  "items": { "type": "string"},
  "type": "array"
},
"Tags": {
  "items": { "type": "string"},
  "type": "array"
},
"Attributes": {
  "items": { "type": "string" },
  "type": "array"
},
"NotGoals": {
  "items": { "type": "string"},
  "type": "array"
},
"NotTags": {
```

```
  "items": { "type": "string" },
  "type": "array"
},
"NotAttributes": {
  "items": { "type": "string"},
  "type": "array"
}
```

Listing 5.4: Rules

```
"Rule": {
  "type": "string"
},
"Grammar": {
  "type": "string"
},
"Index": {
  "type": "integer"
},
"Code": {
  "type": "string"
}
```

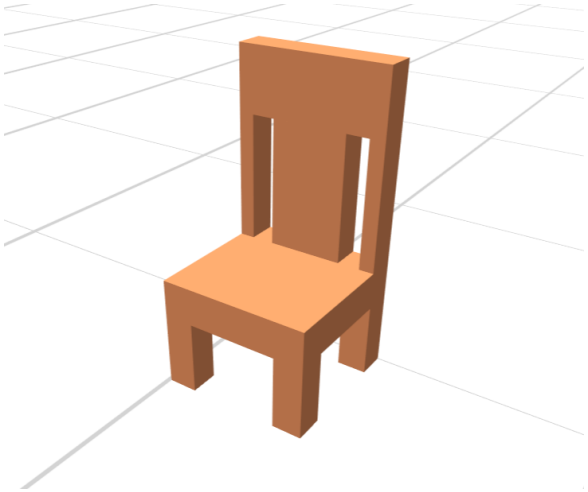
CHAPTER 6

Results

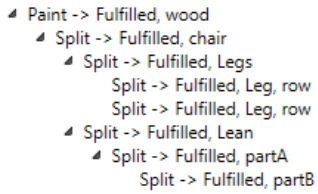
In this chapter, various generated scenes are presented. In further tests, the functionality of the system is evaluated with focus on the effects of user feedback. In addition, the goal sequence algorithm is looked at in detail. The tests are performed in various practical and generic scenarios to verify the expected behavior of the different algorithms.
In the next part, the performance of the framework for basic use cases is evaluated. More specifically, run times of individual algorithms for large amount of data are compared.

6.1 Example scene

This section shows an example of a created scene using this program and aims to give some insight about the possibilities of this tool as well as the connection between grammar scripts, rules and goals.



(a) The created 3D model.



(b) The corresponding parse tree.

Figure 6.1: Example 1: Wooden chair

Figure 6.1a above shows a model created from a simple grammar script (Listing 6.1):

Listing 6.1: Grammar: wooden chair

```
new Axiom("chair", "wood").Size(0.4, 0.8, 0.4);
```

The parsed grammar contains a single axiom with two goals and does not define any new rules. Figure 6.1b shows the parse tree corresponding to the chair model. As we can see, multiple rules were applied to the axiom. The actual rules that have been applied to fulfill the goals `chair` and `wood` originate from different grammars (Listing 6.2)

Listing 6.2: Grammar: wood

```
new Rules.Paint(x => Material.Brown((byte)x.GetInt("wood_color")))
  .Fulfills("wood").IfHas("wood_color");

for (byte i = 0; i <= 9; i++) {
  new Rules.Paint(Material.Brown(i))
    .Fulfills("wood").IfNotHas("wood_color");
}
```

In this grammar script, multiple rules fulfilling the goal `wood` are created. For the application on the axiom in the wooden chair example, one of these was picked at random (weighted based on pre-existing feedback records). In the system were this example was created, a number of matching `chair` rules exist in the cloud. One of these rules is shown in Listing 6.3.

Listing 6.3: Grammar: chair

```
new Rules.Split(Axis.Y,
  Relative(0.3).Goal("Legs").Attribute("LegsRelativeDiameter",
      Rnd(0.05, 0.10)),
      Relative(0.15).Goal("Sit"),
      Relative(0.75).Goal("Lean"))
      .Fulfills("chair")
      .Name("ChairSplit");

new Rules.Split(Axis.X,
      Absolute(@GetDouble("LegsRelativeDiameter")).Goal("Leg").Note("row"),
      Relative(1).Void(),
      Absolute(@GetDouble("LegsRelativeDiameter")).Goal("Leg").Note("row"))
      .Fulfills("Legs")
      .IfIs("chair")
      .Name("LegsHSplit");

new Rules.Split(Axis.Z,
      Absolute(@GetDouble("LegsRelativeDiameter")).Goal("Leg"),
      Relative(1).Void(),
      Absolute(@GetDouble("LegsRelativeDiameter")).Goal("Leg"))
      .Fulfills("Leg", "row")
      .IfIs("chair", "legs")
      .Name("LegsVSplit");

new Rules.Split(Axis.Z,
      Absolute(0.08).Note("partA"),
      Relative(1).Void())
      .Fulfills("Lean")
      .IfIs("chair")
      .Name("LeanSplit");

new Rules.Split(Axis.Y,
  Relative(0.2).Note("partB"),
  Relative(0.1))
  .Fulfills("partA")
  .IfIs("chair", "Lean")
  .Name("LeanYSplit");

new Rules.Split(Axis.X,
  Relative(0.1),
  Relative(0.15).Void(),
  Relative(0.5),
  Relative(0.15).Void(),
  Relative(0.1))
  .Fulfills("partB")
  .IfIs("chair", "lean")
  .Name("LeanXSplit");
```

This code is an example for a set of rules that was selected to created the pictured chair model. It is worth noting, that the goal `wood` was fulfilled before `chair`. This is a direct result of the goal sequences algorithm. Based on recorded feedback, the system deemed the sequence `wood -> chair` to be much more suitable than `chair -> wood`. As a consequence, the chair model uses a single wood color, instead of colors chosen for each part independently. The effects of user feedback on the chosen ordering of goals is evaluated in detail in Section 6.2.2.

6.2 Cloud development

In this section various large scale test results are presented, showing the development of the cloud in theoretic long lasting scenarios. These are generic scenarios with artificial semantics.

6.2.1 Rule probabilities

For the evaluation of the rule selection process, we consider a grammar with a single single axiom, containing one goal. For the context of a specific shape, or rather a specific goal, only rules which fulfill this goal will factor into the probability calculations. Therefore, all rules considered in this test scenario fulfill the shape's goal.

We assume that the cloud is filled with a set number of rules, initially with no user feedback provided. In this scenario, the probabilities are equally distributed and each rule has equal chances at the beginning. We now want to observe the changes of the rule probabilities over time in different scenarios.

Growing user feedback

In this scenario, a set of 10 rules R exist in the database which fulfill our generic goal. Each of these rules has 100 feedback records stored in the database. The feedback values as well as the similarity to the grammar's shape are uniformly distributed. In this test we apply feedback to each of the 10 rules in multiple iterations. The applied feedback values are equally spaced from -1 to +1 (i.e. R_1 to R_5 receive ascending negative values, R_6 to R_{10} receive ascending positive values).

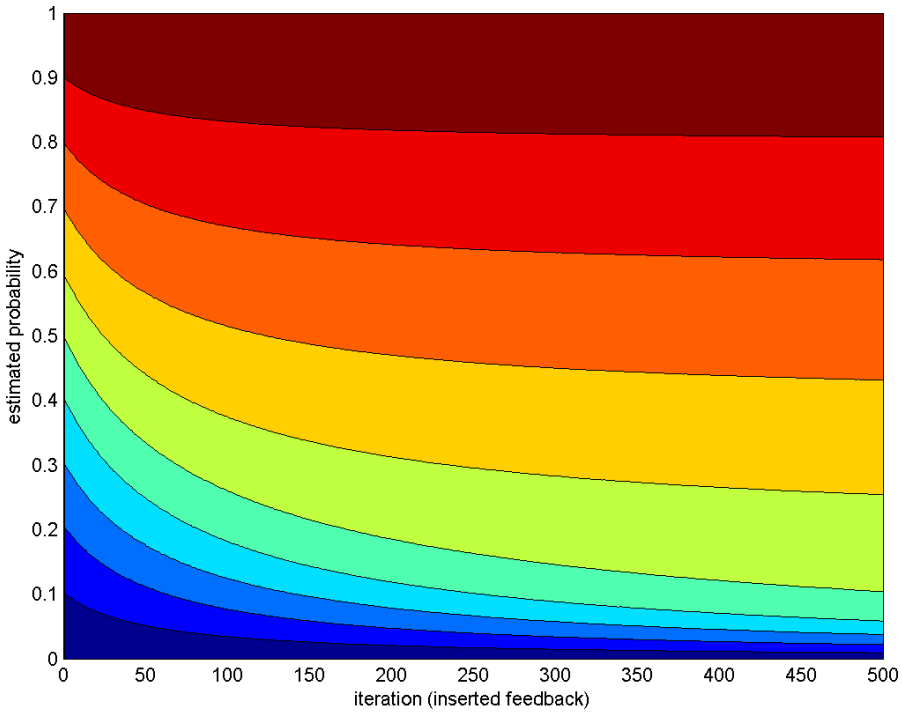


Figure 6.2: Test result visualizing the effects of growing user feedback on the probabilities of individual rules.

The shown area plot diagram (figure 6.2) shows the probabilities for individual rules after n iterations. At n iterations, $n + 100$ feedback entries exist for a particular rule. The diagram shows that additional feedback entries had a higher effect on the probability at earlier iterations, where the total amount of feedback entries is lower, which should be obvious considering the nature of the probability calculation. From a practical standpoint, this makes it clear that for the long time usage of the system, giving additional feedback on very old rules could prove to be a pointless effort for the user. This could potentially cause users to give up on existing rules and create new ones instead. It is noticeable that the probabilities for rules which received very low positive feedback values (yellow) grow almost at the same rate as rules with high values (dark red). The sign of the received feedback is much more important than the actual values.

For a different perspective on the effects of growing user feedback another test was performed (figure 6.3). In this scenario, just two rules exist in the database, initialized with 100 random feedback records each. Over the course of 1000 iterations, positive feedback is added to the first rule (red) and negative feedback to the second one (green). The diagram shows the changes of probabilities of the initially equally weighted rules over multiple iterations.

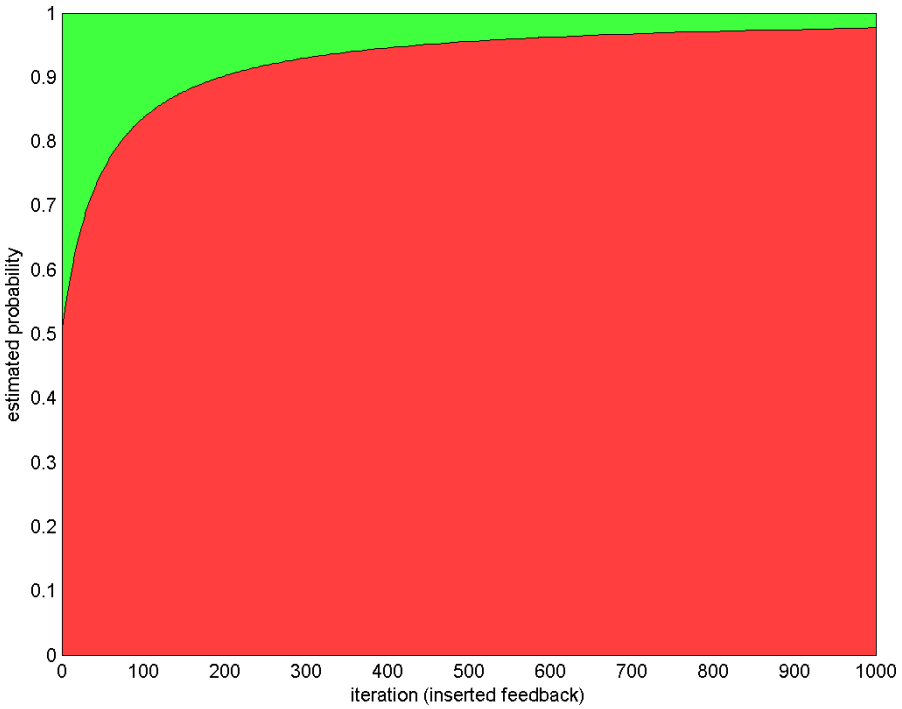


Figure 6.3: Test result visualizing the effects of growing user feedback on the probabilities of individual rules.

Growing rule database

The second test scenario shows a growing number of rules in the database and its effects on the probabilities of existing rules. In this scenario, 3 rules (R_1, R_2 and R_3) exist in the database. R_1 initially has 100 records of random positive feedback, while R_3 received 100 random negative entries. R_2 has no existing feedback records. Over 1000 iterations, one additional rule fulfilling the grammar's goal including random user feedback is added into the system in each step. The diagram in Figure 6.4 shows the probabilities of R_1 (red), R_2 (green) and R_3 (blue) after n iterations on a logarithmic scale. The white area represents the summed probabilities of the n additional rules that have been added.

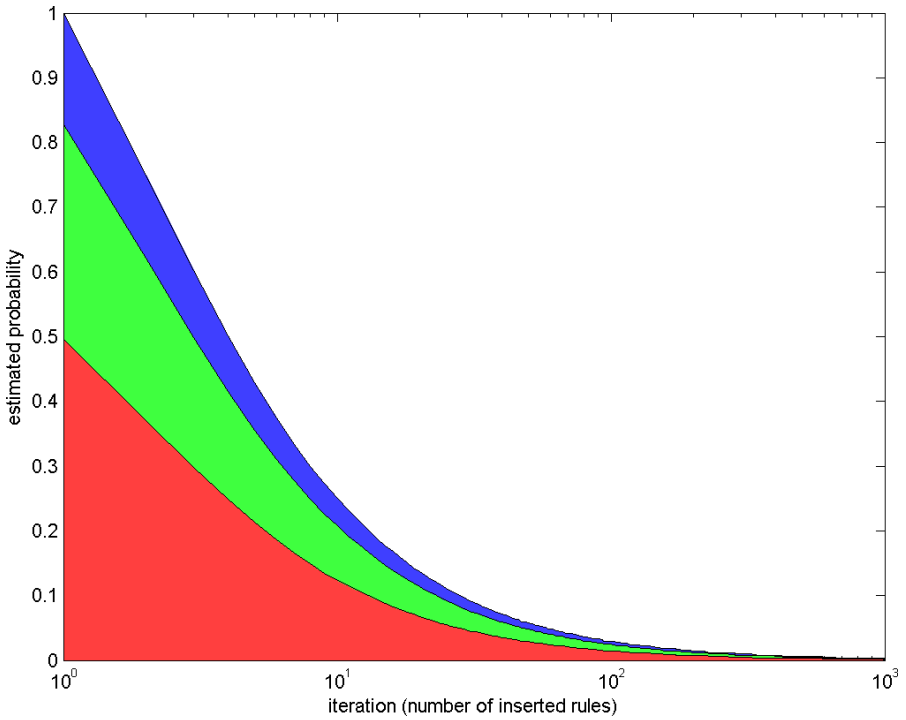


Figure 6.4: Test result visualizing the effects of a growing rule database on the probabilities of individual rules.

Initially, the feedback has a clearly visible impact on the probabilities. After more and more rules are added to the database, the individual rules get less likely to be chosen. Relatively the probabilities of the three original rules stay the same over multiple iterations. Generally, the system allows all rules to be chosen and assigns equal probability to rules with neutral feedback. This behavior could however lead to a 'flood' of newly created rules dominating existing positively received rules even with a large amount of feedback records.

To show some practical results, we consider a simple grammar to generate a colored chair model. In this grammar, an axiom with the goal `coloredchair` is used, which is fulfilled by a range of material rules which apply a color to this shape. In addition, these rules assign a new goal `chair` to the shape, in order to continue with the generation of the chair geometry. We would like to generate a wooden chair model, and therefore want the system to choose the rule for brown color. An axiom with the goals `coloredchair, wood` is used, but no rule in the system exists which would fulfill the goal `wood`. Therefore, we need to apply positive feedback on applications of the brown color rule, in order to teach the system, that *wood* equals **brown** in this particular context.

In this test scenario, 7 different material rules (which fulfill `coloredchair`) exist, each with random initial user feedback. To evaluate the expected behavior, positive feedback

for the brown color rule and a shape with the goals `coloredchair, wood` is applied over 1000 iterations.

Figure 6.5 shows the probabilities of the 7 available material rules. Starting with equal weights, additional feedback applied to the brown color rule (deep blue) increases its chances to be selected for a shape with the `coloredchair` and `wood` goals.

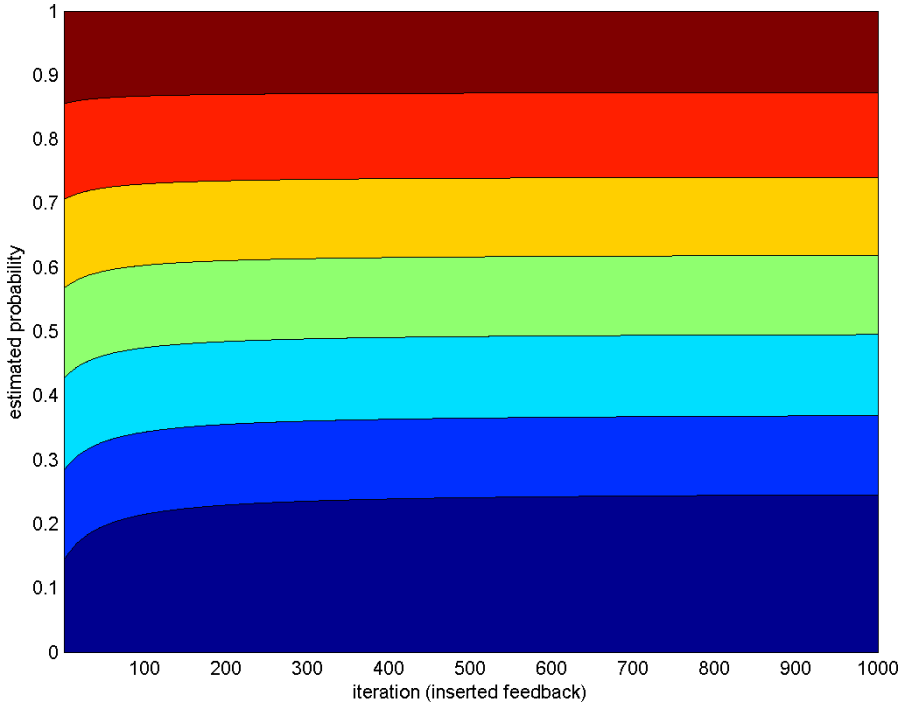


Figure 6.5: Test result visualizing the probabilities of rules depending on feedback and context.

The results show, that we can influence the rule selection via context specific feedback. The system learns that shapes with the goal `wood` are supposed to be brown. The probabilities for the other material rules are still very high, which makes it obvious that this feedback alone is not enough to actually enforce specific rules for specific goals. Controlling the selection process via feedback is no direct replacement for the use of goals.

6.2.2 Goal sequences

The next set of tests aims to evaluate the goal sequence algorithm (see Section 4.3), which is executed when a non-terminal shape of a grammar that is derived has multiple active goals from the same derivation step. In the following test scenarios, the algorithm is applied on a shape with a set of generic goals. In each test run the goal sequence is determined a set number of times to evaluate the distribution between the possible

sequences. We perform these runs in multiple iterations where in each step additional feedback is added.

The first test scenario considers two different goals with multiple matching rules for each. This scenario applies to the wooden chair example that was described earlier (Section 6.1). The executed grammar script contains a single axiom with the goals `wood` and `chair`. The grammar contains several rules for the generation of a chair model as well as rules to apply a brown material. The database is initialized with 100 positive feedback entries, equally distributed for the sequence orders `{wood, chair}` and `{chair, wood}`. In each iteration, positive feedback with the sequence `{wood, chair}`. The diagram (Figure 6.6) shows the probabilities of the possible sequences to be chosen (red = `wood` before `chair`, blue = `chair` before `wood`).

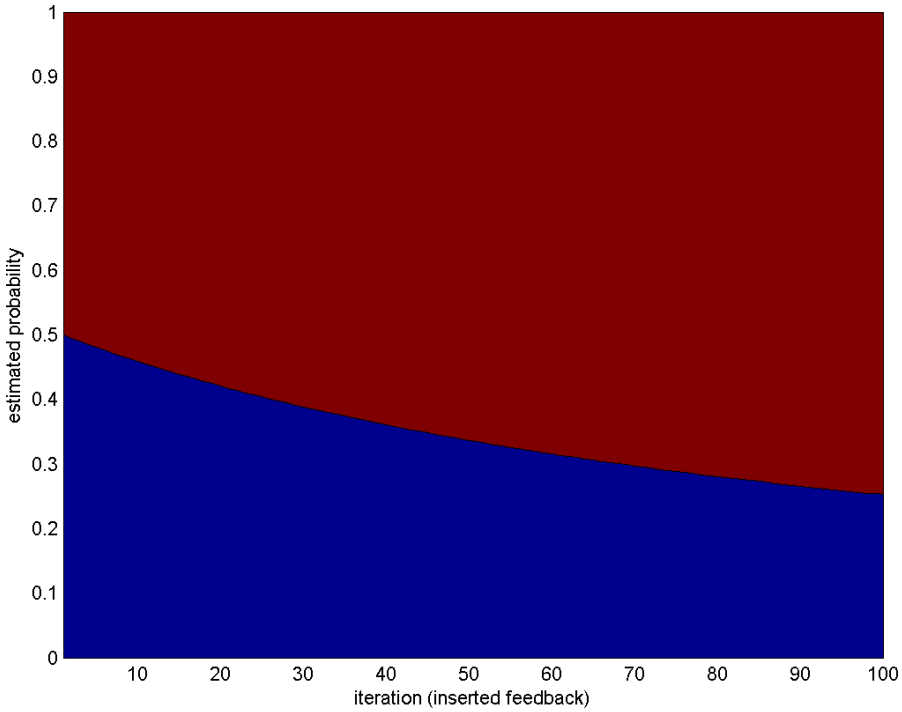
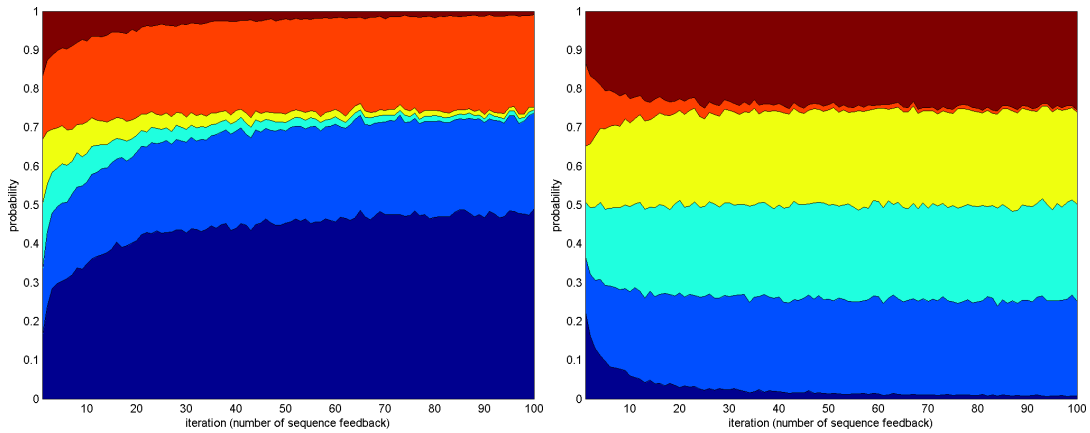


Figure 6.6: Test results visualizing the probabilities of goal sequences adapting to user feedback (Red = *wood* before *chair*).

In the practical example, these results show that with positive feedback on results where `wood` was fulfilled before `chair`, the system is more likely to consider `wood` first.
In the next test runs, a set of three different goals is used (`{A, B, C}`). Three goals could be applied in six different sequences (`{A,B,C}`, `{A,C,B}`, etc.). The diagrams in Figure 6.7 show the relative number of times these sequences were chosen. The sequences starting from the bottom in the diagrams are ordered alphabetically (i.e. `{A,B,C}` corresponds to the first area at the bottom of the diagrams).



(a) positive feedback added (b) negative feedback added

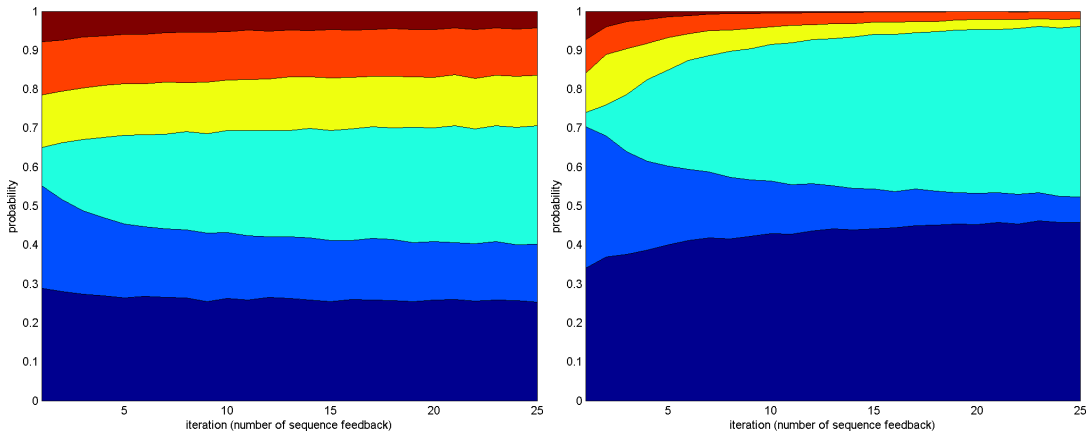
Figure 6.7: Probabilities of goal sequences.

For the first run (Figure 6.7a), in each iteration, a positive feedback record is added for the goal sequence `{A, B}`. With more feedback being added, the number of occurrences for the sequences where `A` comes before `B` increases (`{A,B,C}` and `{A,C,B}` in deep blue and light blue as well as `{C,A,B}` in orange), while those where `B` comes before `A` become less likely.

In the second run (Figure 6.7b), negative feedback records are added on the sequence `{A, B}`. According to the definitions in 4.3, negative feedback penalizes only directly succeeding goals. As a result, the occurrences of the sequences `{A,B,C}` (deep blue), and `{C,A,B}` (orange) decrease over the course of multiple iterations.

Global vs. local mode

In this test the differences between the global and local mode for goal sequence calculations are evaluated. In this scenario, sequences for the goals `{A, B, C}` are calculated. In practice, the system only uses the local mode when a certain number of goals is reached, but for the purpose of this test, this threshold was set to 1. The test scenario is initialized with the same random sequence feedback for both runs. In each iteration, manual user feedback is simulated and specific positive feedback is applied. The test shows how the two calculation modes calculate the probabilities. For the proposed calculation model we consider the global solution to be optimal, as all possible sequences are evaluated. The purpose of this test is therefore to evaluate the local results in comparison. The plots in Figure 6.8 show the approximate probabilities for each of the 6 possible sequences.



(a) (optimal) global result (b) (approximate) local result

Figure 6.8: Probabilities of goal sequences.

The most obvious differences in the results is that the local mode puts a stronger weight on applied feedback. This leads to the sequence `{C,B,A}` (dark red) being almost completely eliminated. In the current system, the local mode is seen more as a safeguard than an actual alternative to the global solution. Cases where a sequence has to be calculated from an extremely high number of goals are seen as very unlikely. However, for future improvements and to allow better scaling, the global approximation could be optimized.

6.3 Performance

In this section the run time of the system is evaluated in various scenarios. All tests were performed locally on a PC with the following specification:

- CPU: Intel Core i7-4770K @ 3.5GHz
- RAM: 16 GB
- GPU: RADEON HD 7970

6.3.1 Example scenes

The grammars for the example scene shown for the previous tests as well as some more complex grammars were executed multiple times and their average run time was recorded. As an additional metric the average number of matched rules is noted. This is calculated from the total number of rules which were considered for selection (i.e. each rule that a probability was calculated for) in each derivation process.

Table 6.1: Grammar execution performance

grammar	avg. run time	avg. rules
Chair	256.0 ms	33
Room	1871.6 ms	320
Table set	590.8 ms	93
4x4 chairs	2647.6 ms	501

The results show run times between 0.25 and 3 seconds. In a system with the purpose of creating visual content, these run times are borderline acceptable for the user. While a quarter of a second is arguable fast enough, the run times for the derivation of complex grammars is very high. In the current implementation, users should preferably work on smaller pieces (i.e. rules for single objects) of a large scene and put them together in the last step. This is obviously not an optimal solution and the system's overall performance should be top priority for future improvements.

6.3.2 Rule selection

To evaluate the performance of rule selection in a large scale scenario a test database was filled with large amounts of generic data. For these tests, a grammar containing a single axiom with a single goal was executed. Before each of the test runs the cloud was filled with a number of generic rules (which fulfill the provided goal) and randomized feedback on these rules.

Table 6.2: Performance in large scale scenarios

matching rules	feedback per rule	total feedback records	average runtime
10	1	10	0.025 s
10	10	100	0.044 s
10	100	1000	0.323 s
100	1	100	0.454 s
100	10	1000	3.112 s
100	100	10000	30.775 s
1000	1	1000	30.384 s
1000	10	10000	315.071 s
1000	100	100000	>30 min

The results from Table 6.2 show very high run times for high numbers of rules and feedback entries. In a practical application these scenarios would occur when a very large number of rules fulfill a common goal and match the required constraints. This means that the system needs to calculate the probability which in turn requires all the relevant feedback records to be evaluated. In this test scenario all applied feedback was relevant to the derived shape, which may not be the case in most situations. Nonetheless, this

shows that the current implementation of the database is not suited for such large scale queries. For future work it would be wise to either optimize the probability algorithm or limit the amount of rules that can be matched at the same time by adding additional constraints.

6.3.3 Goal sequences

In this section the performance of the goal sequence algorithm is evaluated. The tests solely focus on the performance of the sequence calculations and do not contain any database queries. This test is therefore not an indicator for the performance in practice but instead shows the differences between the local, global and hybrid modes of the algorithm. In multiple tests, the run time for the calculation of a sequence of 5 to 10 goals with 100 feedback entries was recorded.

Table 6.3: Goal sequence performance

Goals	Global	Local	Hybrid (T=7)
5	0.525 ms	0.243 ms	0.491 ms
6	2.545 ms	0.237 ms	0.288 ms
7	24.10 ms	0.246 ms	2.418 ms
8	238.7 ms	0.252 ms	2.413 ms
9	∼2.3 s	0.283 ms	2.492 ms
10	-	0.288 ms	2.350 ms

Table 6.3 shows the extremely high increase in run time of the global algorithm with increasing number of goals and makes it very clear why the local solution is required for a high amount of goals. The local algorithm has almost constant run time for 5-10 goals. As for the hybrid approach it is easy to see that the local run time is simply added to the global values once the set threshold is reached. This makes it very easy to adjust the threshold depending on the run time that is required in a real scenario.

CHAPTER 7

Conclusion & Future Work

The proposed system is a versatile framework for procedural content generation. The used ACGAX modeling language allows users to create a huge variety of different scenes from grammar scripts. Combined with the concepts of recommendation systems, the cloud based rule selection system responds to the user's preferences The application of explicit user feedback is integrated into the actual modeling work flow. When executing a grammar, its axioms are stepwise derived. For this task, the system matches the shape's goals with the stored rules from the cloud. Based on recorded feedback, probabilities are assigned to the matching rules and one of them is chosen. The core element of the derivation process is the goal notation. Axioms in a grammar are initialized with goals. Each rule of a grammar fulfills at least one of these goals and in turn may add new goals to the resulting shape, which become relevant in the next derivation step. This continues until no more goals exist or no more rules are applicable. The end result of a derived grammar is a scene of 3D objects. The system visualizes the derivation process in a tree structure and provides an interface to perform locking operations. These actions provide feedback for the rule selection and directly control the next derivation process by preserving certain rules of the grammars' previous derivation.

7.1 Limitations

While the system in its current state allows users to create a variety of scenes, there are some limitations in the modeling language and in the framework which possibly hinder the user's design process.

One limitation is the **preservation of details**. The provided locking operations allow users to provide specific feedback on detailed parts of a scene. With the current features however, a user can only effectively lock or regenerate outer details of the scene. Locking or rejecting an inner node of the parse tree (corresponding to an earlier derivation step) will also affect any succeeding nodes. This means that if a user is unhappy with a rule

that was applied early, the whole scene starting from with that node is newly generated. Likewise, it is not possible for a user to lock details of an object, without locking the preceding rules which led to the generation of this detail. To provide a practical example, we assume a user is writing a grammar to model a house. The grammar is made to build up the house from the ground up. First its outer walls are created, then its rooms are modeled and finally the interior is generated. Now the user might be totally satisfied with the created interior, but might dislike structural parts of the house. Since these elements were created in an earlier derivation step and the interior is a successor within the same axiom, it is not possible to regenerate the outer elements of the house while preserving the interior. To overcome this limitation, the modeling language needs to be further extended.

7.2 Future work

The conclusions in the previous paragraphs and in various parts of this thesis make room for a lot of future improvements. While possible extensions of the modeling language have been discussed already, I want to focus on the possible improvements for the implementation of the system as well as the database.

7.2.1 Performance

As the results in Section 6.3 have shown, the system's performance has lots of room for improvements. As the general performance for the creation of scenes is lacking, there are multiple areas where optimization is required.

Parallelization

In multiple parts of the system, different tasks are performed independently. While some level of parallelization is already present, the run times for a lot of processes could potentially be shortened with multi threading. The following is a list of some noteworthy examples:

- **Derivation**: When a grammar is derived, it potentially consists of multiple axioms where each could have more than one active shape. The rule matching processes for these shapes are generally independent from each other and could be performed in a parallel manner.
- **Rule probability**: The evaluation of probability for matching rules is one of the most expensive tasks in the system. It would seem possible to speed up this task by splitting the set of matching rules into multiple parts to perform the probability calculation in multiple threads.
- **Goal sequences**: The global method for the calculation of goal sequences iterates over all permutations of a set of goals. To improve the performance, these permutations could be split up and evaluated in a parallel manner.

Database optimizations

One of the biggest bottlenecks within the system is the database connection. While the use of more efficient database systems or clients might improve the performance, in a lot of cases the actual queries might need to be revised. As the results from Section 6.3.2 have shown, the amount of data that is queried at once has a huge impact on the performance. The growing nature of the cloud creates additional issues. The amount of feedback that has to be evaluated for the application of a rule will only increase over time, as there is no expiration for feedback.

For future work a solution for these problems should be found. As mentioned above, one simple way to prevent feedback from flooding the database would be an expiration timer which invalidates feedback after a set amount of time or a number of parsing steps have passed. For a more elegant solution to reduce the overall amount of entries that need to be evaluated, it could be possible to merge similar feedback entries together.

These database optimizations could also be applied to the rule matching process in a similar way. Reducing the number of rules that are matched, could greatly limit the amount of feedback records that need to be processed. However, with any of these potential improvements it is important to keep their limitations on the possibilities of the system in mind.

7.2.2 Multi-user support

The end goal for this project is a collaborative modeling system for a large number of users. The proposed system is currently operating with only one client system with a single user. While connecting multiple clients to the same database should not be a huge difficulty, there are various additional challenges that emerge with a multi-user system. For one the GUI would need a lot of improvements to provide some level of user account management and allow the use of collaborative features. Second, the system needs to identify the responsible user and keep track of his actions in various parts of the system. With multiple users involved, the dynamic scaling of the cloud would be even more important. Therefore, the performance improvements for database queries would have an even higher priority in a multi-user scenario.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Bis06] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[FBY92] William B Frakes and Ricardo Baeza-Yates. Information retrieval: data structures and algorithms. 1992.

[FJN+14] Alexander Felfernig, Michael Jeran, Gerald Ninaus, Florian Reinfrank, Stefan Reiterer, and Martin Stettinger. Basic approaches in recommendation systems. In *Recommendation Systems in Software Engineering*, pages 15–37. Springer, 2014.

[IW16] Martin Ilčík and Michael Wimmer. Collaborative modeling with symbolic shape grammars. *Complexity & Simplicity - Proceedings of the 34th eCAADe Conference*, 2:417–426, aug 2016.

[Jac12] Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912.

[Lin68] Aristid Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280 – 299, 1968.

[MWH+06] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *Acm Transactions On Graphics (Tog)*, volume 25, pages 614–623. ACM, 2006.

[OK+98] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83, 1998.

[PM01] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[RHM97] Y. Rui, T. S. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in mars. In *Proceedings of International Conference on Image Processing*, volume 2, pages 815–818 vol.2, Oct 1997.

[Roc71] Joseph John Rocchio. Relevance feedback in information retrieval. 1971.

[RRS11] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[SG71] George Stiny and James Gips. Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress (2)*, volume 2, 1971.

[STN16] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[TV08] Johan W Tangelder and Remco C Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia Tools and Applications*, 39(3):441–471, 2008.

[WWSR03] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. *Instant architecture*, volume 22. ACM, 2003.