The Rules Behind
– Tutorial on Generative Modeling –

Ulrich Krispel, Christoph Schinko, Torsten Ullrich*
Visual Computing, Fraunhofer Austria Research GmbH
Graz University of Technology, Austria

Abstract

This tutorial introduces the concepts and techniques of *generative modeling*. It starts with some introductory examples in the first learning unit to motivate the main idea: to describe a shape using an algorithm. After the explanation of technical terms, the second unit focuses on technical details of algorithm descriptions, programming languages, grammars and compiler construction, which play an important role in generative modeling. The purely geometric aspects are covered by the third learning unit. It comprehends the concepts of geometric building blocks and advanced modeling operations. Notes on semantic modeling aspects – i.e. the meaning of a shape – complete this unit and introduce the inverse problem. What is the perfect generative description for a real object? The answer to this question is discussed in the fourth learning unit while its application is shown (among other applications of generative and inverse-generative modeling) in the fifth unit. The discussion of open research questions concludes this tutorial.

The assumed background knowledge of the audience comprehends basics of computer science (including algorithm design and the principles of programming languages) as well as a general knowledge of computer graphics. The tutorial takes approximately 120min. and enables the attendees to take an active part in future research on generative modeling.

keywords:
geometry processing; generative, procedural modeling; inverse modeling; modeling applications; shape description; language design

reference:

```
@Article{KSU2014,
  Title    = {The Rules Behind -- Tutorial on Generative Modeling},
  Author   = {Krispel, Ulrich and Schinko, Christoph and Ullrich, Torsten},
  Journal  = {Proceedings of Symposium on Geometry Processing / Graduate School},
  Year     = {2014},
  Pages    = {2:1--2:49},
  Volume   = {12}
}
```

* corresponding author:
mail: torsten.ullrich@fraunhofer.at
www: www.cgv.tugraz.at & www.fraunhofer.at

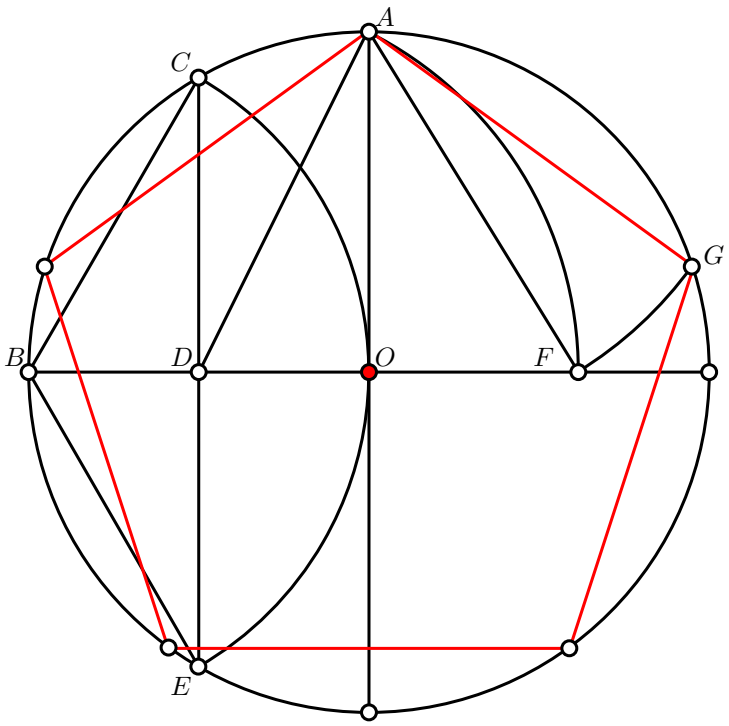
Contents

1 Introduction to "Generative Modeling"

Generative modeling has been developed in order to generate highly complex objects based on a set of formal construction rules. This modeling paradigm describes a shape by a sequence of processing steps, rather than just the end result of applied operations: Shape design becomes rule design. This approach is very general and it can be applied to any domain and to any shape representation that provides a set of generating functions.

1.1 Ruler and Compass

The ruler-and-compass construction is the construction of lengths, angles, and other geometric figures using only an idealized ruler and compass. Geometry from the days of the ancient Greeks placed great emphasis on problems of constructing various geometric figures using only a ruler without markings (to draw lines) and a compass (to draw circles). All ruler-and-compass constructions consist of repeated application of five basic constructions based on Euclid's axioms [39] using the points, lines and circles that have already been constructed. Based on these geometric primitives and a fixed set of operations, the ruler-and-compass constructions – such as illustrated in Figure 1 – are the first algorithmic descriptions of generative models.



Construction of a pentagon:

1. Draw a circle in which to inscribe the pentagon and mark the center point O.
2. Construct a pair of perpendicular lines which intersect in O and mark their intersection with the circle A and B.
3. Let D be the midpoint of $\overline{BO}$. The circle with center D and radius $|\overline{DA}|$ intersects the line defined by the points B and O. Mark the intersection point as F.
4. The length of section $\overline{AF}$ is equal to the edge length $\overline{AG}$ of an inscribed pentagon (red).

Figure 1: *The construction of a pentagon can be performed using compass and a straightedge only. The construction algorithm is based on* Euclid*'s axioms* [39].

Because of the prominent place Greek geometric constructions held in Euclid's Elements [39], these constructions are sometimes also known as Euclidean constructions. Such constructions lay at the heart of the geometric problems of antiquity of circle squaring, cube duplication, and angle trisection. The Greeks were unable to solve these problems, but constructions for regular triangles, squares, pentagons, and their derivatives had been given by Euclid.

It turns out that all constructions possible with a compass and straightedge can be done with a compass alone, as long as a line is considered constructed when its two endpoints are located [123]. The reverse is also true, since Jacob Steiner showed that all constructions possible with straightedge and compass can be done using only a straightedge, as long as a fixed circle and its center have been drawn beforehand. Such a construction is known as a Steiner construction.

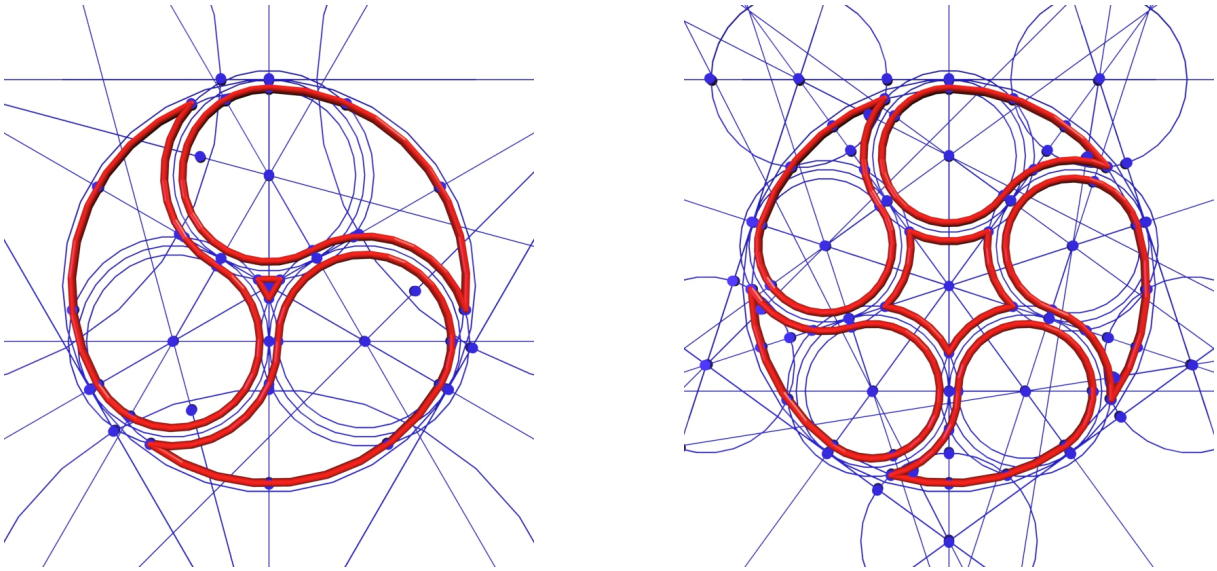


Figure 2: *Compass and ruler operations have long been used in interactive procedural modeling. This Gothic window construction was created in the framework presented by* Wolfgang Thaller *et al. using direct manipulation without any code or graph editing* [103].

The long history of geometric constructions [57] is also reflected in the history of civil engineering and architecture [64]. For example, Gothic architecture and especially window tracery exhibits quite complex geometric shape configurations. But this complexity is achieved by combining only a few basic geometric patterns. Sven Havemann and Dieter W. Fellner present some principles of this long-standing domain, together with some delicate details, and show how the constructions of some prototypic Gothic windows can be formalized using our generative modeling techniques [35]. Using modularization, so that complex configurations can be obtained from combining elementary constructions, different combinations of specific parametric features can be grouped together, which leads to the concept of styles. They permit to differentiate between the basic shape and its appearance, i.e., in a particular ornamental decoration [103]. This leads to an extremely compact representation for a whole class of shapes [11].

1.2 Architecture

Generative modeling techniques have rapidly gained attention throughout the past few years. Many researchers enforced the creation of generative models due to its many advantages. All objects with well-organized structures and repetitive forms can be described procedurally. In these cases, generative modeling is superior to conventional approaches.

A big advantage of procedural modeling techniques is the included expert knowledge within an object description [109], e.g., classification schemes used in architecture, archaeology, civil engineering, etc. can be mapped to procedures. For a specific object, only its type and its instantiation parameters have to be identified [112]; the generative building blocks themselves are fixed and do no change. As a consequence, only their evaluation parameters have to be specified: Figure 3 illustrates variations of the same building blocks.

The usage of generative modeling techniques in architecture is not limited to buildings of the past. Over the last few decades, progressive architects have used a new class of design tools that support generative design. Generative modeling software extends the design abilities of architects by harnessing computing power in new ways. Computers, of course, have long been used to capture and implement the design ideas of architects by means of CAD and, more recently, 3D modeling. But generative design actually helps architects design by using computers to extend human abilities.

An impressive example is the Helix Bridge in Singapore (see Figure 4). The 280 m bridge is made up of three 65 m spans and two 45 m end spans. The major and minor helices, which spiral in opposite directions, have an overall diameter of 10.8 m and 9.4 m respectively. The outer helix is formed from six tubes which are set equidistant from one another, whereas the inner helix consists of five tubes.

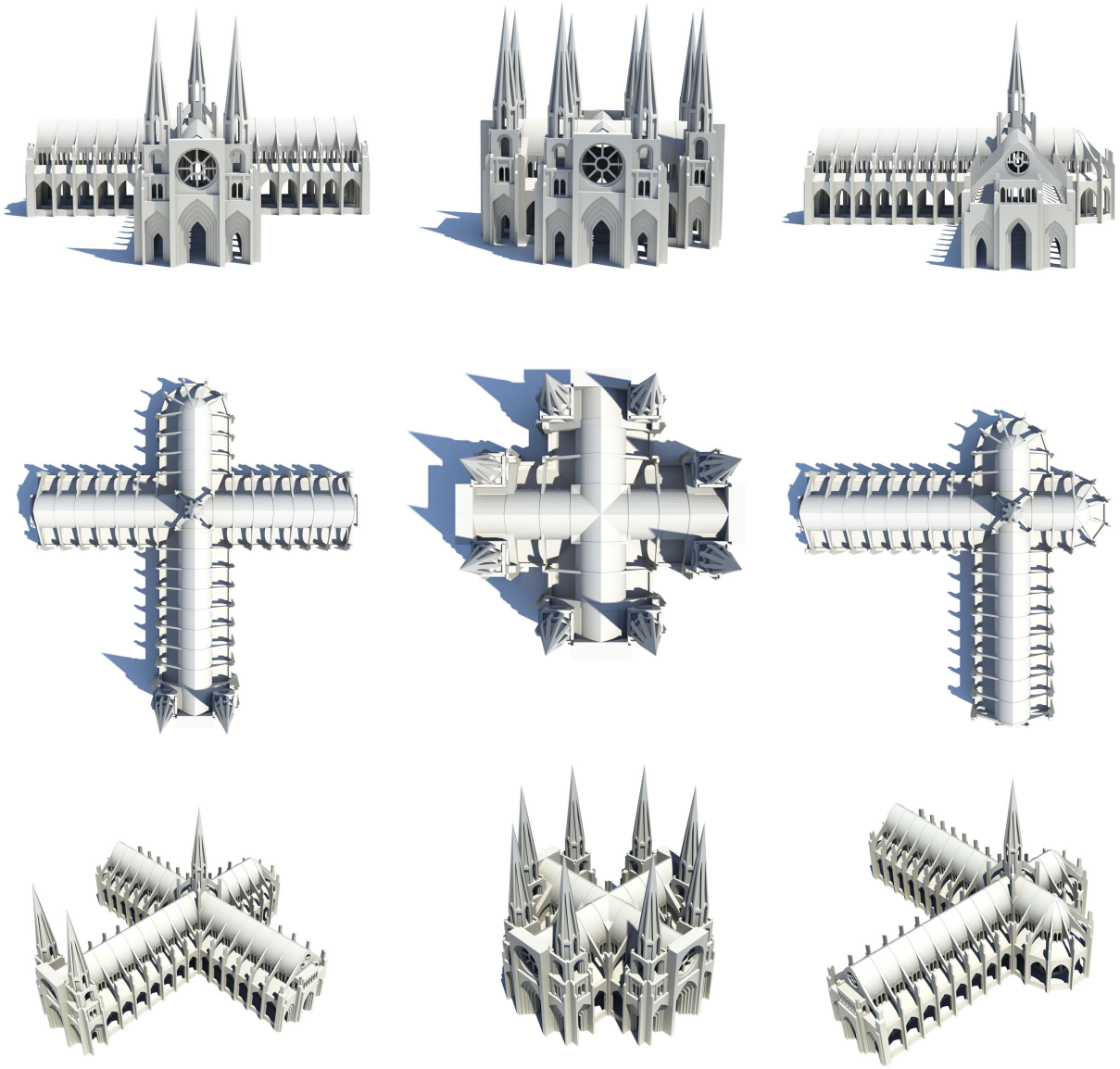


Figure 3: *Gothic architecture flourished during the high and late medieval period. It is defined by strict rules with its characteristics: pointed arcs, the ribbed vaults, and the flying buttresses. These building blocks have been combined in various ways to create great churches and cathedrals all over Europe. The generative description of Gothic cathedrals encodes these building blocks and the rules on how to combine them. The result is an algorithm that takes a few high-level parameters. By modifying these parameters, it is easy to generate a family of Gothic examples (left, middle, right). The building blocks have been created by* Michael Curry, *http://www.thingiverse.com/thing:2030.*

Figure 4: *The Helix Bridge is a pedestrian bridge in the Marina Bay area in Singapore. Its generative design has been optimized numerically. Furthermore, the bridge was fully modeled in order to visualize its form and geometrical compatibility, as well as to visualize the pedestrian experience on the bridge.*

The bridge design is the product of inseparable collaboration between architects (Cox Architecture and Architects 61) and civil engineers (Arup Consultant). For its 280 meter length, the dual helix structure of the bridge utilizes 5 times less steel than a conventional box girder bridge. This fact enabled the client to direct the structure to be constructed entirely of stainless steel for its longevity.

1.3 Civil Engineering

The generative modeling approach is very general. It can be applied to any domain and is not restricted to shape representations [20]. Nevertheless, this tutorial focuses on shape design, computer-aided design and 3D modeling.

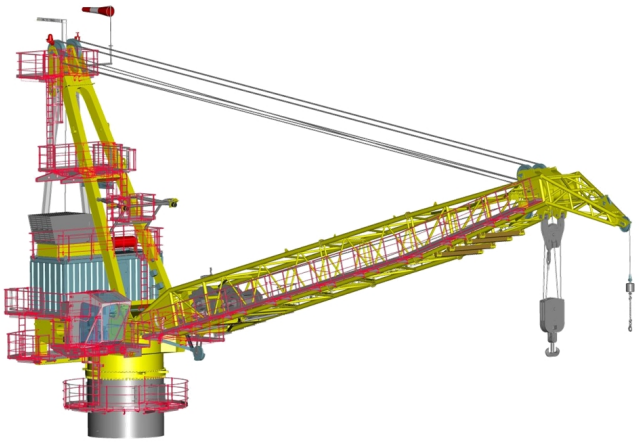


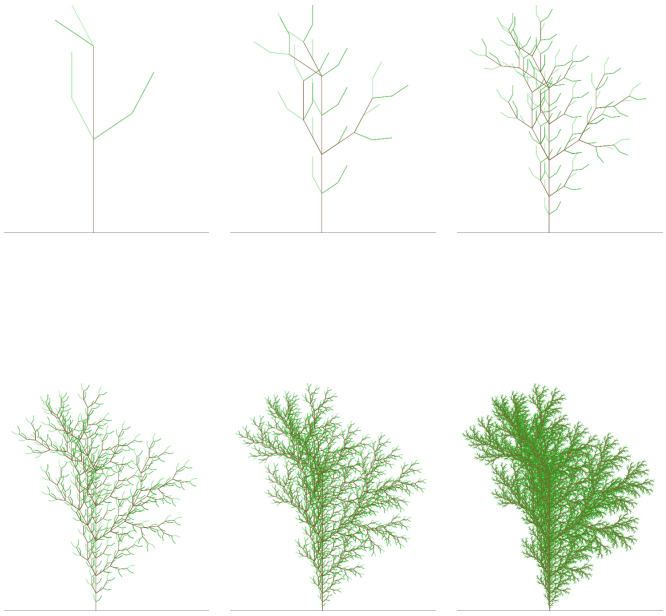
Figure 5: *The design of ascent assemblies for offshore cranes (colored in red) results in high efforts and contributes a major part of the overall engineering costs of a crane. In case of repetitive and nearly identical design processes, the product development processes can be optimized by software driven design automation: the reduction of engineering efforts by modeling design knowledge* [30].

In the context of 3D computer-aided design (CAD), each design process that involves repetitive tasks is perfectly suited for a generative approach. Engineering processes can be differentiated in repetitive and creative processes. In contrast to creative processes, repetitive ones consist of nearly identical tasks and are therefore independent of creative decisions. This condition is necessary for modeling them in a system of rules as demonstrated by Gerald Frank [31]: Liebherr manufactures and sells an extensive range of products including ship-, offshore and harbor mobile cranes as well as hydraulic duty cycle crawler cranes and lift cranes. Due to customers' needs each crane has to be partially or fully engineered, but the design process of ascent assemblies is based on repetitive tasks based on a set of invariant rules that can be modeled and stored. In numerous interviews with engineering experts at Liebherr the repetitive design processes have been analyzed and a generative model has been designed. Integrated into the existing CAD pipeline, a construction engineer now only has to determine the defining parameters of an assembly and fill out the corresponding input fields in a user interface. The engineering of ascent assemblies of an offshore crane required up to 150 hours. Using the procedural approach, the efforts have been reduced down to 10%.

1.4 Natural Patterns

In today's procedural modeling systems, scripting languages and grammars are often used as a set of rules to achieve a description. Early systems based on grammars were Lindenmayer systems [81] (L-systems) named after Aristid Lindenmayer. They were successfully applied to model plants. Given a set of string rewriting rules, complex strings are created by applying these rules to simpler strings. Starting with an initial string the predefined set of rules form a new, possibly larger string. The L-systems approach reflects a biological motivation. In order to use L-systems to model geometry an interpretation of the generated strings is necessary.

The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures (see Figure 6). This lead to the introduction of parametric L-systems. The idea is to associate numerical parameters with L-system symbols to address continuous phenomena which were not covered satisfactorily by L-systems alone.



L-System:

- Axiom: FX
- Angle: 28°
- Rules:
  $F \mapsto C_0FF-[C_1-F+F]+[C_2+F-F]$
  $X \mapsto C_0FF+[C_1+F]+[C_3-F]$

whereas F denotes "draw forward" and $+/-$ denote "turn left"/"turn right". The square bracket [corresponds to saving the current values for position and angle, which are restored when the corresponding square bracket] is executed. C_0, C_1, C_2 switch colors and X does not correspond to any drawing action.

This example can be executed online by Kevin Roast's L-Systems-Demo:
http://www.kevs3d.co.uk/dev/lsystems/

Figure 6: *Lindenmayer systems are a simple but elegant "turtle rendering" platform. The recursive nature of L-system rules lead to self-similarity and thereby fractal-like forms. Plant models and natural-looking organic forms "grow" and become more complex by increasing the iteration level i.e. the number of substitutions.*

Combined with additional 3D modeling techniques, Lindenmayer systems can be used to generate complex geometry [106], [107]. In order to generate models of plants, terrains, and other natural phenomena that are convincing at all different scales, Robert F. Tobler et al. introduce a combination of subdivision surfaces, fractal surfaces, and parametrized L-systems, which makes it possible to choose which of them should be used at each level of resolution. Since the whole description of such multi-resolution models is procedural, their representation is very compact and can be exploited by level-of-detail renderers that only generate surface details that are visible.

This kind of data amplification can be found in various fields of computer graphics. E.g. curved surfaces specified by a few control points are tessellated directly on the GPU. This results in low storage costs and allows generating the complex model only when needed, while also reducing memory transfer overheads. Although L-systems are parallel rewriting systems, derivation through rewriting leads to very uneven workloads. Furthermore, the interpretation of an L-system is an inherently serial process. Thus, L-systems are not straightforwardly amenable to parallel implementation. In 2010, Markus Lipp et al. presented a solution to this algorithmic challenge [54].

1.5 Applications

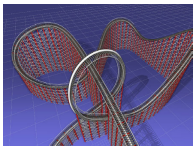
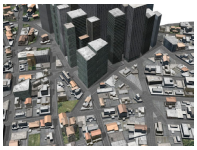
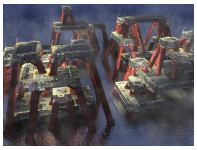
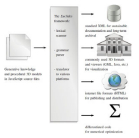
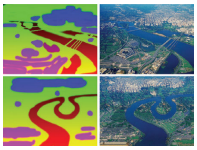
	**Procedural Generation of Road** [32]		**Interactive Architectural Modeling with Procedural Extrusions** [47]
	**Procedural Modeling of Interconnected Structures** [49]		**Interactive Modeling of City Layouts using Layers of Procedural Content** [52]
	**Interactive Visual Editing of Grammars for Procedural Architecture** [53]		**Model Synthesis: A General Procedural Modeling Algorithm** [61]
	**Computer-generated residential building layouts** [62]		**Interactive Furniture Layout Using Interior Design Guidelines** [63]
	**Interactive Coherence-Based Facade Modeling** [68]		**Procedural Modeling of Cities** [76]
	**Modeling Procedural Knowledge – a generative modeler for cultural heritage** [86]		**Scripting Technology for Generative Modeling** [87]
	**Procedural Descriptions for Analyzing Digitized Artifacts** [112]		**Modelling the Appearance and Behaviour of Urban Spaces** [118]

Table 1: *Currently architecture, urban design and civil engineering are the predominant fields of applications for generative modeling, although the main concept can be applied to any domain.*

2 Languages & Grammars

Originally, scripting languages have been designed for a special purpose, e.g., to be used for client-side scripting in a web browser. Nowadays, the applications of scripting languages are manifold. JavaScript, for example, is used to animate 2D and 3D graphics in VRML [18] and X3D [9] files. It checks user forms in PDF files [17], controls game engines [25], configures applications, defines 3D shapes [87], and performs many more tasks. According to John K. Ousterhout scripting languages use a higher level of abstraction compared to system programming languages as they are often typeless and interpreted to emphasize the rapid application development purpose [74]. Whereas system programming languages are designed for creating algorithms and data structures based on low-level data types and memory operations. As a consequence, graphics libraries [72], graphics shaders [70] and scene graph systems [84], [120] are usually written in C/C++ dialects [26], and procedural modeling frameworks use scripting languages such as Lua, JavaScript, etc.

2.1 Language Processing & Compiler Construction

The evaluation of procedural descriptions typically utilizes techniques used for description of formal languages and compiler construction [77]. The range of different concepts of languages to describe a shape is very wide and comprehends all kinds of linguistic concepts [21]. The main categories to describe a shape are

- rule-based: using substitutions and substitution rules to build complex structures out of simple starting structures [75], [50], [67], [96].
- imperative and scripting-based: using a scripting engine and techniques used in predominant programming languages [34], [87], [49], or
- GUI and dataflow-based: using new graphical user interfaces (GUI) and intelligent GUIs to detect structures in modeling tasks, which can be mapped onto formal descriptions [53], [102].

Nevertheless, the general principles of formal descriptions and compiler construction are in all cases the same – independent of ahead-of-time compilation, just-in-time compilation or interpretation [89]. The basic steps are illustrated in Figures 7 and 8. They outline the compilation process and show the main data structures – especially the abstract source tree (AST): In the first stage the input source code is passed to lexer and parser. A first step is to convert a sequence of characters into a sequence of tokens, which is done by special grammar rules forming the lexical analysis. For instance, in some languages only a limited number of characters is allowed for an identifier: all characters `A-Z`, `a-z`, digits `0-9` and the underscore `_` are allowed with the condition that an identifier must not begin with a digit or an underscore. Such lexer rules are embedded in another set of rules – the parser rules. They are analyzing the resulting sequence of tokens to determine their grammatical structure. The complete grammar consists of a hierarchical structure of rules for analyzing all possible statements and expressions that can be formed in a language, thus forming the syntactic analysis.

For each language construct available a set of rules is validating syntactic correctness. At the same time actions within these rules create the intermediate AST structure that represents the input source code. The resulting AST is the main data structure for the next stage: semantic analysis. Once all statements and expressions of the input source code are collected in the AST, a tree walker analyzes their semantic relationships, i.e., errors and warnings are generated, for instance, when symbols are used but not defined, or defined but not used.

Having performed all compile-time checks, a translator uses the AST to generate platform-specific files. In other words, this task involves complete and accurate mapping of the AST to constructs of the target platform.

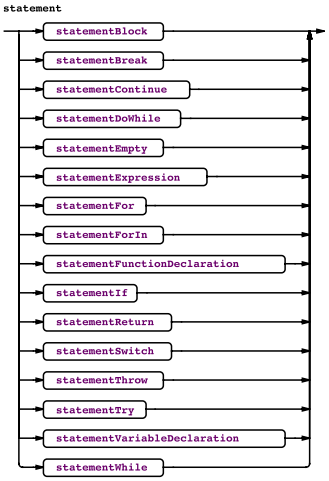
The example in Figures 7 and 8 shows a compilation process of JavaScript. In JavaScript, the top-level rule of an AST is always a simple list of statements – no enclosing class structures, no package declaration, no inclusion instructions, etc. Each statement contains all included substatements and expressions as well as associated comments. During the validation step, this tree structure is extended by reference and occurrence links; e.g., each method call references the method's definition and each variable definition links to all its occurrences. Having assured that all compile-time checks are carried out, symbols are stored in a so called namespace. During validation, this data structure is used to detect name collisions (e.g. redefinition of variables) and undefined references (e.g. usage of undeclared variables).

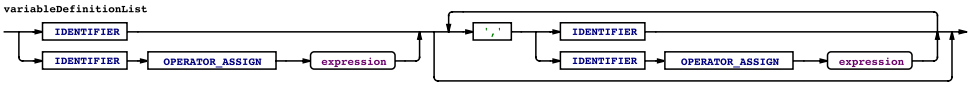
```
var pi = 3.14159;

function circle_area(radius) {
   return pi * radius * radius;
}

for(var i=1; i<10; ++i) {
   var radius = 10+2*i;
   var area   = circle_area(radius);
   var text   = "circle #" + i + ": "
      + "r = " + radius + " cm, "
      + "A = " + area   + " cm^2.";
   IO.print(text);
}
```

1. The first step in a compiler pipeline is performed by a *lexical analyzer*. It converts source code (top left) into a sequence of tokens, i.e. a string of one or more characters that is significant as a group. Tokens are identified based on specific rules of the *lexer*.

2. The stream of tokens (middle right) is processed by the *syntactic analyzer* based on the grammar rules of the language to parse. This example is written in JavaScript. Its entry point into the grammar rules is a `statement`-rule.

```
var pi = 3.14159 ; function circle_area
( radius ) { return pi * radius *
radius ; } for ( var i = 1 ; i
< 10 ; ++ i ) { var radius = 10
+ 2 * i ; var area = circle_area
( radius ) ; var text = "circle #"
+ i + ": " + "r = " + radius +
" cm, " + "A = " + area + " cm^2." ;
IO . print ( text ) ; }
```





3. The result of the parsing step is an *abstract source tree* (AST) (see Figure 8 right). Afterwards, the *sematic analyzer* constructs the table of symbols (see Figure 8 left) and generates all references needed to resolve names (see Figure 8 red pointers).

4. In an optional step, an *optimizer* may perform changes in the AST in order to speed up the final code; in this example, the variable `pi` and the function `circle_area` are only assigned once. Therefore, they might be resolved and inlined in order to reduce the number of look-ups and function calls.

5. In the final step a *code generator* produces object code of the target platform.

Figure 7: *A compiler consists of three main components:* a front-end *reads in the source code and constructs a language-independent representation – a so-called abstract source tree (AST). The* middleware *performs normalization and optimization steps on the AST. Finally, the* back-end *generates platform-specific object code, i.e. executables, libraries, etc.*

Table of symbols:

GLOBAL.IO
[...]

GLOBAL.area
[...]

GLOBAL.circle_area
[...]

GLOBAL.circle_area.msg
[...]

GLOBAL.circle_area.radius
[...]

GLOBAL.i
[...]

GLOBAL.pi
declaration statement:
`var pi = 3.14159;`
declaration file & line:
example.ecs:1
references:
1. example.ecs:1
   `var pi = 3.14159;`
2. example.ecs:4
   `return pi * radius * radius;`

GLOBAL.radius
[...]

GLOBAL.text
[...]

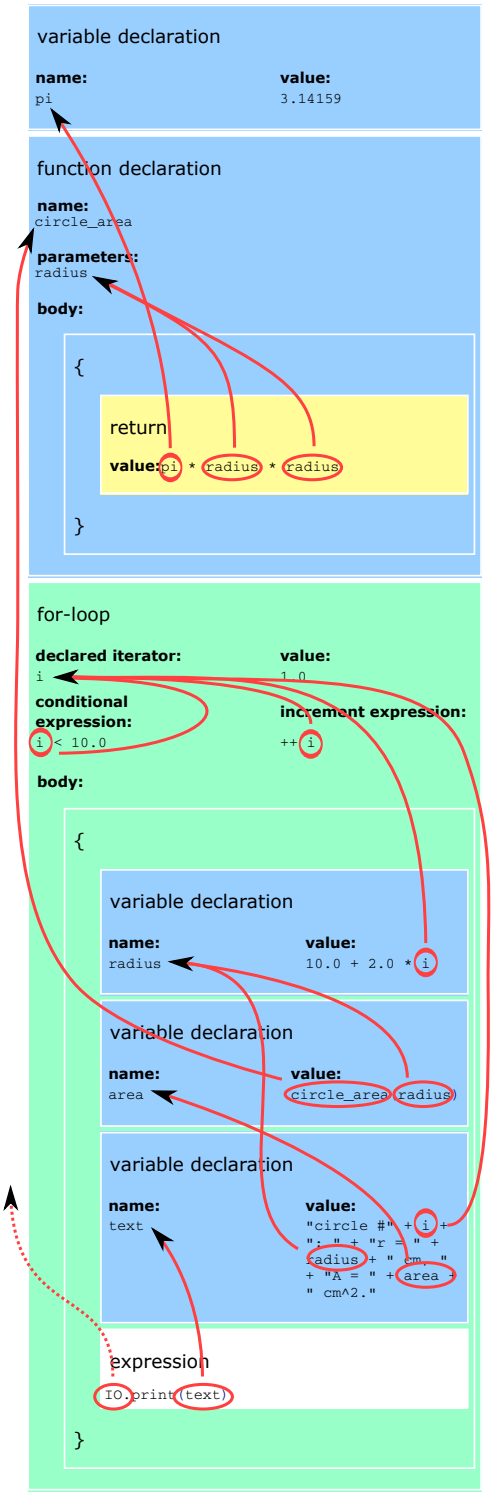


Figure 8: *The most important data structure within a compiler (suite) is the abstract source tree (AST), which represents the input source code in a language-independent way. It consists of a tree structure to encode the hierarchical, nested statements (right) enriched by references to resolve symbols (visualized in red). Additional data structures – such as a table of symbols (left) – simplify the work performed by the compiler's middleware.*

From a historical point of view, the first procedural modeling systems were LINDENMAYER systems [81], or L-systems for short. These early systems, based on grammars, provided the means for modeling plants. The idea behind it is to start with simple strings and create more complex strings by using a set of string rewriting rules. The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures.

Later on, L-systems are used in combination with shape grammars to model cities [76]. YOGI PARISH and PASCAL MÜLLER presented a system that generates a street map including geometry for buildings given a number of image maps as input. The resulting framework is known as *CityEngine* – a modeling environment for *CGA Shape*. Also based on *CGA Shape*, MARKUS LIPP et al. presented another modeling approach [53] following the notation of PASCAL MÜLLER [67]. It deals with the aspects of more direct local control of the underlying grammar by introducing visual editing. Principles of semantic and geometric selection are combined as well as functionality to store local changes persistently over global modifications.

SVEN HAVEMANN takes a different approach to generative modeling. He proposes a stack based language called *Generative Modeling Language* (GML) [34]. The postfix notation of the language is very similar to that of *Adobe Postscript*.

Generative modeling inherits methodologies of 3D modeling and programming [110], which leads to drawbacks in usability and productivity. The need to learn and use a programming language is a significant inhibition threshold especially for archaeologists, cultural heritage experts, etc., who are seldom experts in computer science and programming. The choice of the scripting language has a huge influence on how easy it is to get along with procedural modeling. *Processing* is a good example of how an interactive, easy to use, yet powerful, development environment can open up new user groups. It has been initially created to serve as a software sketchbook and to teach students fundamentals of computer programming. It quickly developed into a tool that is used for creating visual arts [83].

Processing is basically a Java-like interpreter offering new graphics and utility functions together with some usability simplifications. A large community behind the tool produced libraries to facilitate computer vision, data visualization, music, networking, and electronics. Offering an easy access to programming languages that are difficult to approach directly reduces the inhibition threshold dramatically. Especially in non-computer science contexts, easy-to-use scripting languages are more preferable than complex programming paradigms that need profound knowledge of computer science. The success of *Processing* is based on two factors: the simplicity of the programming language on the one hand and the interactive experience on the other hand. The instant feedback of scripting environments allow the user to program via "trial and error". In order to offer our users this kind of experience, we enhanced our already existing compiler to an interactive environment for rapid application development.

2.2 Scripting Languages for Generative Modeling

There exists a broad variety of tools and techniques for procedural modeling. We provide an overview of a collection of generative modeling techniques (see Table 2) under the following aspects:

application domain: Generative modeling tools often incorporate prior knowledge of a specific application domain, e.g. generative modeling of architecture [99], or modeling of organic structures [55], [81], which is reflected in this aspect.

programming category: Some methods are built on top of conventional programming languages, or scripting languages. On the contrary, some techniques are built using proprietary languages, such as rule-based systems for buildings [125], or [67] for urban modeling. Some systems can be used even without any scripting, e.g. graph-based languages [103], or the visual interactive editing of split grammars [53].

environment: This aspect covers the tool set that provides geometric entities and operations, for example the geometry kernel of a 3d modeling software, e.g. the open source modeling suite blender or a proprietary system such as shape grammars on convex polyhedra [104].

Tool Name	Application Domain	Programming Category	Environment
Blender Scripting	general purpose modeling	python scripting	open source modeling software *blender*
CityEngine [67]	urban modeling	CGA shape	commercial integrated development environment *CityEngine*
Generalized Grammar G^2 [50]	scientific	python scripting	commercial modeling software Houdini
Generative Modeling Language (GML) [34]	CAD	postscript dialect	proprietary, integrated development environment for polygonal and subdivision modeling
Grasshopper 3D	visual arts, rapid prototyping, architecture	visual programming based on dataflow graphs, Microsoft .NET family of languages	commercial modeling software *Rhinoceros3D*
HyperFun [78]	scientific	specialized high-level programming language	proprietary geometry kernel FRep (Function Representation)
Maya Scripting	general purpose modeling	Maya Embedded Language (MEL) and python scripting	commercial modeling software *Autodesk Maya*
OpenSCAD	CAD	OpenSCAD language	open source, based on CGAL geometry kernel
PLaSM	scientific	python scripting, Function Level scripting	integrated development environment *Xplode*
Processing	visual arts, rapid prototyping	Java dialect	open source, integrated development environment *Processing*
PythonOCC	general purpose modeling and CAD	python scripting	Open CASCADE Technology
Revit Scripting	architecture	Microsoft .NET family of languages	commercial modeling software *Autodesk Revit*
siteplan [47]	rapid prototyping, architecture	interactive GUI-based modeler	open source, integrated development environment *siteplan*
Sketchup Scripting	architecture, urban modeling and CAD	Ruby scripting	commercial modeling software *SketchUp*
Skyline Engine [79]	urban modeling	visual programming based on dataflow graphs, python scripting	commercial modeling software Houdini
speedtree	plants/trees	interactive GUI-based modeler, SDK for C++	standalone modeler and integration into various game engines
Terragen	landscape modeling	interactive GUI-based modeler	free and commercial, integrated development environment *Terragen*
XFrog [24]	plants/trees	interactive GUI-based modeler	integrated development environment, standalone and plugins for *Maya* and *Cinema4D*

Table 2: *Overview on generative / procedural 3D modeling tools and approaches.*

There are many different programming paradigms in software development. Therefore, they also apply to the field of generative modeling, where some paradigms emerged to be useful for specific domains.

imperative: In many cases, generative modeling is carried out using classical programming paradigms: A programming language is used to issue the commands that generate a specific object using a library that utilizes some sort of geometry representation and operations to perform changes. An example are compass and ruler systems used by an imperative language. Furthermore, any modeling software that is scriptable by an imperative language or provides some sort of API falls into this category. Note that the resulting geometry is often produced as side effects.

dataflow based: The program is represented as a directed graph of the data flowing between operations. The graph representation also allows for a graphical representation; Visual Programming Languages (VPL) allow to create a program by linking and modifying visual elements, many VPL's are based on the dataflow paradigm. Examples in the domain of generative modeling are the Grasshopper3D plug-in for the Rhinoceros3D modeling suite, or the work of Gustova Patow et al. [79] built on top of the procedural modeler Houdini.

rule based systems: Another different representation that proved useful for generative modeling are rule-based systems. Such systems provide a declarative description of the construction behavior of a model by a set of rules. An example are L-Systems, as described in the Introduction. Furthermore, the seminal work of George Stiny and James Gips [99] introduced shape grammars, as a formal description of capturing the design of paintings and sculptures, in the sense of "design is calculating". Similar to formal grammars, shape grammars are based on rule replacement.

shape grammars In the classical definition [99], a shape grammar is the 4-tuple $SG = (V_T, V_M, R, I)$, where V_T a set of shapes, ${V_T}^*$ denotes the set of the shapes of V_T with any scale or rotation. V_M is a finite set of *non-terminal* shapes (markers) such that ${V_T}^* \cap V_M = \emptyset$. R denotes the set of rules, which consists of pairs (u, v), such that $u = (s, m)$ consists of a shape $s \in {V_T}^*$ combined with a marker of $m \in V_M$, and v is a shape consisting of either

- $v = s$
- $v = (s, \tilde{m})$ with $\tilde{m} \in V_M$
- $v = (s \cup \tilde{s}, \tilde{m}$ with $\tilde{s} \in {V_T}^*$ and $\tilde{m} \in V_M$

Elements of the set ${V_T}^*$ that appear in and rules of R are called *terminal shapes*. I is called the *initial shape*, and typically contains an $u \in (u, v) \in R$. The final shape is generated from the shape grammar by starting with the initial shape and applying matching rules from R: for an input shape and a rule (u, v) whose u matches a subset of the input, the resulting shape is another shape that consists of the input shape with the right side of the rule substituted in the matching subset of the input. The matching identifies a geometric transformation (scale, translation, rotation, mirror) such that u matches the subset of the input shape and applies it to the right side of the rule. The *language* defined by a shape grammar SG is the set of shapes that will be generated by SG that do not contain any elements of V_M.

split grammars The work of Peter Wonka et al. [125] applied the concepts of shape grammars to derive a system for generative modeling of architectural models. This system uses a combination of a spatial grammar system (split grammar) to control the spatial design and a control grammar, which distributes the design ideas spatially (e.g. set different attributes for the first floor of a building). Both of these grammars consist of rules with attributes that steer the derivation process. The grammar consists of two types of rules: *split* and *convert*. The *split* rule is a partition operation which replaces a shape by an arrangement of smaller shapes that fit in the boundary of the original shape. The *convert* rule replaces a shape by a different shape that also fits in the boundary of the original shape. A simple example is shown in Figure 9.

This system has further been extended by the work of Pascal Müller et al. [67], which introduced a *component split* to extend the split paradigm to arbitrary 3d meshes, as well as occlusion queries and snap lines to model non-local influences of rules. For example, two wall segments that intersect each other should not produce windows such that the window of one wall coincides with the other wall, therefore occlusion queries are used to decide if a window should be placed or not.

The derivation of a split grammar, starting from an initial shape, yields a tree structure, which suggests that the derivation can be speed up by a parallel implementation. However, the non-local influences are a problem because they introduce dependencies between arbitrary nodes of the derivation tree. Recent work by Markus Steinberger et al. [98] shows how to overcome this problem in an GPU implementation.

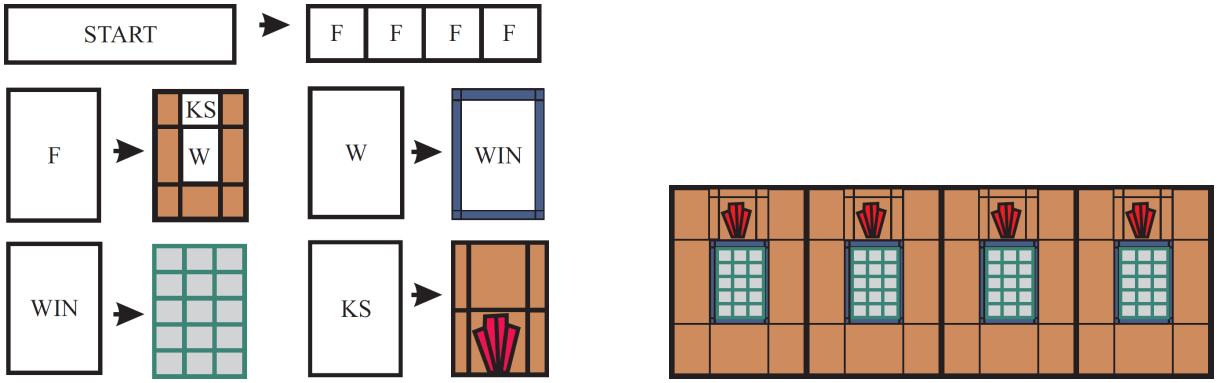


Figure 9: *The tiles of a floor that contains windows with decorative elements (keystones) are generated by a split grammar. The set of rules (left) will yield the final instance of the floor if applied to a start image (right). The derivation process is guided by an additional control grammar (e.g. which keystone is selected), which is not shown in this figure* [125].

3 Modeling by Programming

3D objects, which consist of organized structures and repetitive forms, are well suited for procedural description, e.g. by the combination of building blocks or by using shape grammars. We discuss the problems in conjunction with the definition of a shape:

What is a suitable interface for a building block?

Especially within a pipeline of different tools, this question is gaining in importance.

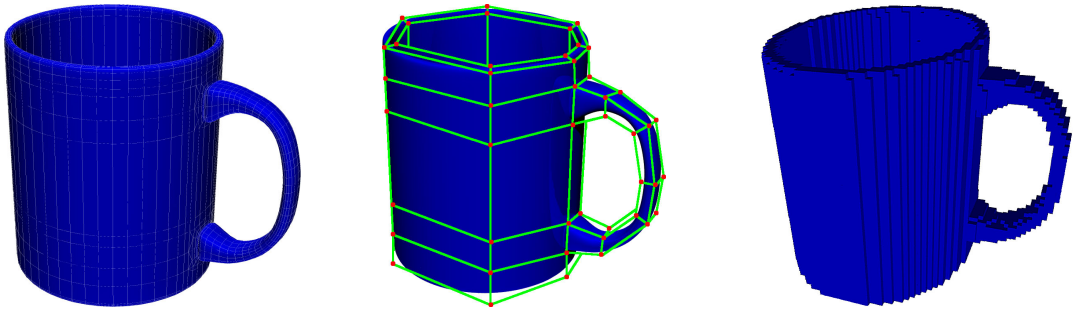


Figure 10: *In this Figure, three different data structures are used to represent a mug. A polygonal mesh is shown in the left image. The image in the middle shows a subdivision surface with the corresponding control mesh. On the right hand side, the mug is represented by a* $64 \times 64 \times 64$ *grid of voxels.*

3.1 Building Blocks & Elementary Data Structures

Several elementary data structures are commonly used in computer graphics (see Figure 10).

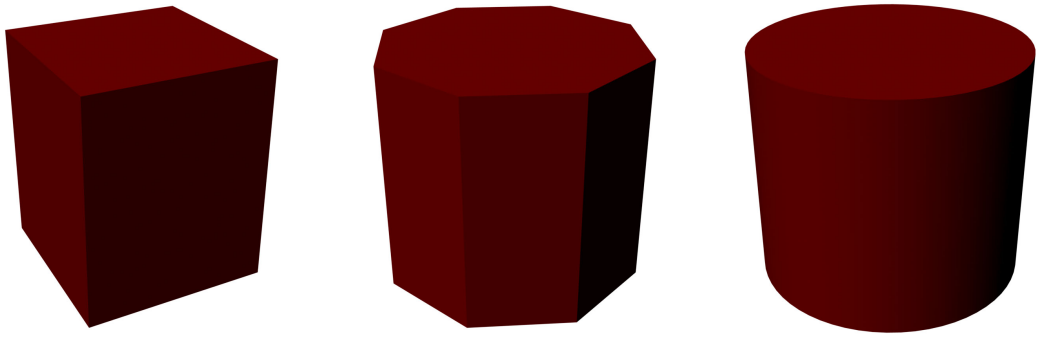


Figure 11: *The quality of the polygonal approximation of a cylinder highly depends on the number of primitives. By increasing the number of primitives a better approximation of a cylinder can be created. In our case we have n-gonal prisms, where n is 4 (left), 8 (middle), 64 (right). Once the approximation is selected and saved as a mesh, the semantic information: "This is a cylinder." is lost.*

Polygonal Meshes A polygonal mesh representation is well-suited for real-time computer graphics. Their inherent structure consisting of vertices, edges and faces can be – more or less – directly mapped onto consumer graphics cards. Due to numerical problems with floating point numbers, modeling with polygons can lead to undesirable results. During many mesh operations, even within a simple intersection routine, the trade-off between range and precision may cause inaccuracies. Another drawback of polygonal meshes can be seen when trying to model free-form shapes. The quality of the approximation of an object's surface with polygons highly depends on the number of primitives and the modeling operations used. Once an approximation is found, all information about the approximated surface is often lost, see Figure 11.

Apart from approximation problems, all operations are well-defined, e.g. boolean mesh operations always yield a polygonal mesh. However, important questions remain unanswered: How does an interface for binary mesh operations on polygonal meshes look like? Are half-edges in this context a suitable interface?

Non-Uniform Rational B-Splines This problem gets worse when modeling with non-uniform rational B-splines (NURBS). Even the question for a simple intersection of two NURBS curves is not easy to answer. Apart from the cases where two curves do not intersect or are identical, there can be at most 9 intersection points, when looking at cubic curves. Calculating the intersection leads to numerical issues. The problems get worse when intersecting two NURBS surfaces. There can be various intersection components such as curve segments, points, loops and singular points. These components are approximated with respect to given tolerances. Given the intersection curve, how are the two surfaces stitched together? An interface for this operation has to deal with all these aspects.

Subdivision Surfaces Subdivision surfaces are defined recursively starting with a given polygonal mesh (typically a quadrilateral, or a triangle mesh), as illustrated in Figure 10 (middle). A refinement scheme is applied to this mesh creating new vertices and faces converging to the limit subdivision surface (which is the surface produced by applying the refinement scheme infinitely many times). Intersection operations are often carried out on a per-facet basis. Therefore, the surfaces are subdivided into a great number of facets and the intersection of surfaces is approximated by the intersection of the facet pairs. Although, the stitching problem remains and undesired artifacts around the intersection curve may occur.

For ensuring manufacturability, additional constraints have to be considered. For 3D printing purposes, the geometry has to be water-tight and free of self-intersections. The former can, for example, be ensured by using topology preserving Euler operators for mesh creation. They operate on the connection graph of the mesh representation and obtain a valid boundary. The latter property can be ensured by careful use of modeling operations.

Convex Polyhedra However there are data structures that do not have these problems. Shapes can be defined by the intersection of half-spaces. In general, the intersection of arbitrary half-spaces need not be bounded. So called convex polyhedra can be defined algebraically as the set of bounded solutions to a system of linear inequalities. An important topological property is that convex polyhedra are homeomorphic to a closed ball. All operations are well-defined, leaving the problem with the finite precision of floating point numbers.

Voxel Representations A voxel represents a data point on a regular, three-dimensional grid – similar to a pixel in an image. Depending on the area of application, the data point can be multi-dimensional, e.g. a vector of density and color. Due to the fact that position and size of a voxel are pre-defined, voxels are good at representing regularly sampled spaces. The approximation of free-form shapes suffers from this inherent property, as can be seen in Figure 10 (right). Nevertheless, voxel representations do not suffer from numerical instabilities as they are typically defined on an integer grid.

Implicit Surfaces Implicit surfaces are defined as isosurfaces by a function $\mathbb{R}^3 \to \mathbb{R}$. Therefore, similar to voxels, a surface is only indirectly specified. Also, with a function it is hard to describe sharp features and it is difficult to enumerate points on a surface. However, this representation has several advantages. There is an efficient check, whether a point is inside a shape, or not. Surface intersections as well as binary operations can be implemented efficiently. Since the surface is not represented explicitly, topology changes are easily possible.

Creating shapes with the presented elementary data structures requires the definition of modeling operations. Depending on the underlying representation, certain modeling operations are difficult or impossible to implement. The selection of operations for these data structures are manifold and can be grouped as follows:

- *Instantiation* are operations for creating new shapes.
- *Binary Creations* are operations involving two shapes such as constructive solid geometry (CSG) operations.
- *Deformations* and *Manipulations* stand for all deforming and modifying operations like morphing or displacing.

Also building blocks can be regarded as modeling operations. Complex shapes typically consist of subparts – so called building blocks. We have already mentioned the open problem of defining a suitable interface on a data structure level. This problem still exists when describing shapes on a more abstract level. When creating an algorithmic description of a shape, one has to identify inherent properties and repetitive forms. These properties must be accounted for in the structure of the description. Identified subparts or repetitive forms are best mapped to functions in order to be reusable. However, the true power of an algorithmic description becomes obvious when parameters are introduced for these functions. As little as the possibility to position a subpart at a different location using input parameters makes the difference. From that point on, the algorithmic description no longer stands for a single object, but for a whole object family.

Within a composition of modeling functions, where each function is attached via its parameters to topological entities defined in previous states of the model, another problem occurs. Referenced entities must then be named in a persistent way in order to be able to reevaluate the model in a consistent manner. In particular, when a reevaluation leads to topological modifications, references between entities used during the design process are frequently reevaluated in an erroneous way, giving results different from those expected. This problem is known as "persistent naming problem" [56].

3.2 Advanced Techniques

Besides classical, geometric operations – such as CSG – procedural and functional descriptions offer novel, additional possibilities to describe a shape.

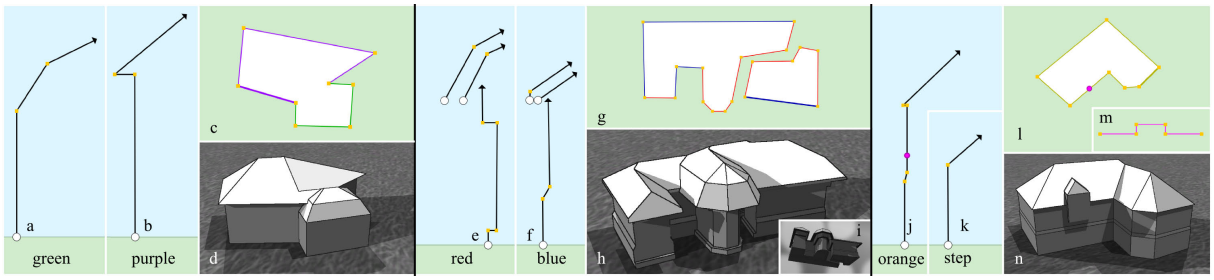


Figure 12: *The work of* Tom Kelly *and* Peter Wonka [47] *offers a framework to specify the geometry of a building by extrusion profiles. The segments of footprint polygons (e.g.* c*) are associated with extrusion profiles, e.g. the green segments are associated to the profile* a*, the purple segments to the profile* b*. The resulting geometry can be seen in* d*.*

Architectural Modeling with Procedural Extrusions This method utilizes the paradigm of footprint extrusion to automatically derive geometry from a coarse description. Input to this system are polygons whose segments can be associated with an extrusion profile polygon. The system utilizes the weighted straight skeleton method [6] to calculate the resulting geometry. Examples can be seen in Figure 12. An implementation is available under the name *siteplan*, see also the tools section and Table 2.

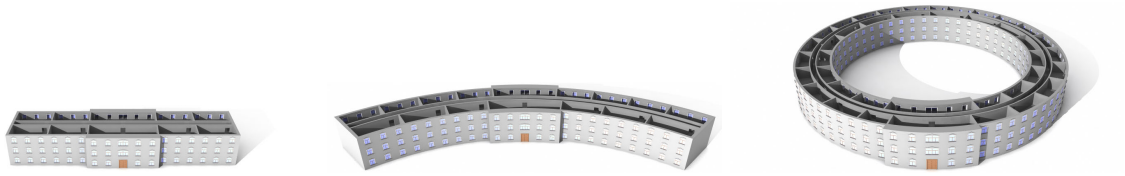


Figure 13: *Deformation aware shape grammars allow the integration of free-form deformation into a grammar-based system that is based on planar primitives and splits. Measurement about the available space for placing objects are taken in deformed space while splits are carried out in undeformed space. An undeformed building with rooms (left image) is deformed using two different deformations (middle, right). It can be observed that the amount of windows and rooms adapts automatically to the available space* [129].

Deformation Aware Shape Grammars Generative models based on shape and split grammar systems often exhibit planar structures. This is the case because these systems are based on planar primitives and planar splits. There are many geometric tools available in modeling software to transform planar objects into curved ones, e.g. *free-form deformation* [91]. Applying such a transformation as a post-processing step might yield undesirable results. For example, if we bend a planar facade of a building into a curved shape, the windows inside the façade will have a curved surface as well. Another possibly unwanted property arises when an object is deformed by scaling, e.g. the windows on a façade would have different appearances. Therefore, René Zmugg et al. introduced *deformation aware shape grammars* [129], which integrate deformation information into grammar rules. The system still uses established methods utilizing planar primitives and splits, however, measurements that determine the available space for rules are performed in deformed space. In this way, deformed splits can be carried out, the deformation can be baked at any point to allow for straight splits in deformed geometry. An example is shown in Figure 13.

Variance Analysis Analyzing and visualizing differences of similar objects is important in many research areas: scan alignment, nominal/actual value comparison, and surface reconstruction to name a few. In computer graphics, for example, differences of surfaces are used to validate reconstruction and fitting results of laser scanned surfaces. Scanned representations are used for documentation as well as analysis of ancient objects revealing smallest changes and damages. Analyzing and documentation tasks are also important in the context of engineering and manufacturing to check the quality of productions. Christoph Schinko et al. [90] contribute a comparison of a reference / nominal surface with an actual, laser-scanned data set. The reference surface is a procedural model whose accuracy and systematics describe the semantic properties of an object, whereas the laser-scanned object is a real-world data set without any additional semantic information. The first step of the process is to register a generative model (including its free parameters) to a laser scan. Then, the difference between the generative model and the laser scan is stored in a texture, which can be applied to all instances of the same shape family as illustrated in Figure 14.

As generative models represent an ideal object rather than a real one, the combination of noisy 3D data with an ideal description enhances the range of potential applications. This bridge between both the generative and the explicit geometry description is very important: it combines the accuracy and systematics of generative models with the realism and the irregularity of real-world data as pointed out by David Arnold [5]. Once the procedural description is registered to a real-world artifact, we can use the fitted procedural model to modify a 3D shape. In this way we can design both low-level details and high-level shape parameters at the same time.

3.3 Semantic Modeling

In some application domains, e.g. in the context of digital libraries, semantic meta data plays an important role. It provides semantic information that makes up the basis for digital library services: indexing, archival, and retrieval. Depending on the field of application, meta data can be classified according to the following criteria [113]:

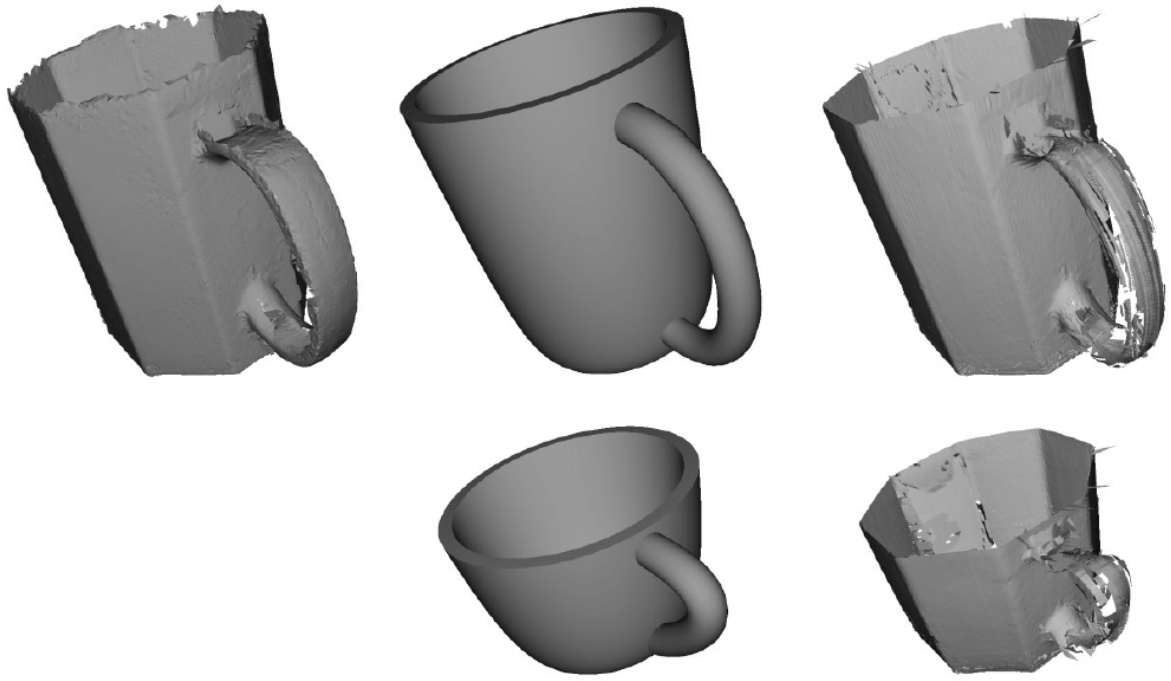


Figure 14: *This figure shows the scanned model (top left), the procedural reference model (top middle), as well as the output of the combined representation (top right). The combined version consists of a static instance of the procedural model with details stored in a texture. The details are applied to the procedural base geometry via shaders. The procedural reference model has been defined by a set of parameters obtained in a fitting process applied on the scanned model. Having modified the procedural parameters, new procedural cups can be generated (bottom middle). If one of these new cups is combined with an already existing texture, previously captured details can be transferred (bottom right).*

Data Type The data type of the object can be of any elementary data structure (e.g. Polygons, NURBS, Subdivision Surfaces, ...).

Scale of Semantic Information This property describes, whether meta data is added for the entire data set or only for a sub part of the object.

Type of Semantic Information The type of meta data can be descriptive (describing the content), administrative (providing information regarding creation, storing, provenance, etc.) or structural (describing the hierarchical structure).

Type of creation The creation of the semantic information for an object can be done manually (by a domain expert) or automatically (e.g. using a generative description).

Data organization The two basic concepts of storing meta data are storing the information within the original object (e.g. EXIF data for images), or storing it separately (e.g. using a database).

Information comprehensiveness The comprehensiveness of the semantic information can be declared varying from low to high in any gradation.

An important aspect of semantic enrichment are standards. Many concepts for encoding semantic information can be applied to 3D data, unfortunately only a few 3D data formats support semantic markup [93]:

Collada The XML-based Collada format allows storing meta data like title, author, revision etc. not only on a global scale but also for parts of the scene. This file format can be found in Google Warehouse where meta data is, for example, used for geo-referencing objects.

PDF 3D PDF 3D allows to store annotations separated from the 3D data even allowing annotating the annotations. An advantage is that the viewer application is widely spread and PDF documents are the quasi standard for textual documents.

Due to the persistent naming problem, a modification of the 3D model can break the integrity of the semantic information. Any change of the geometry can cause the referenced part of the model to no longer exist or being changed.

There are a lot of examples for semantic modeling in various contexts [33], [105], [117], [126]. Here, one representative from the field of geospatial modeling is selected to illustrate the topic:

Input parameters for a generative description can be either artificial or derived from real-world measurements, like survey or satellite images. In the domain of geospatial modeling, data exported from geospatial databases is used. The two file formats GeographyML (www.opengeospatial.org) and CityGML (www.citygml.org) are wide-spread. Erick Mendez et al. [59] generate models using data exported from geospatial databases, typically available in a vector format, to transcode it into 3D models suitable for standard rendering engines. The transcoding process involves information loss, therefore the right point in the pipeline has to be found to perform transcoding – this is called transcoding trade-off. Their modeling framework lets developers optimize the transcoding trade-off to create 3D interactive visualizations. The example augmented reality application shown in Figure 15 displays underground infrastructure created out of geographic information systems (GIS) data.

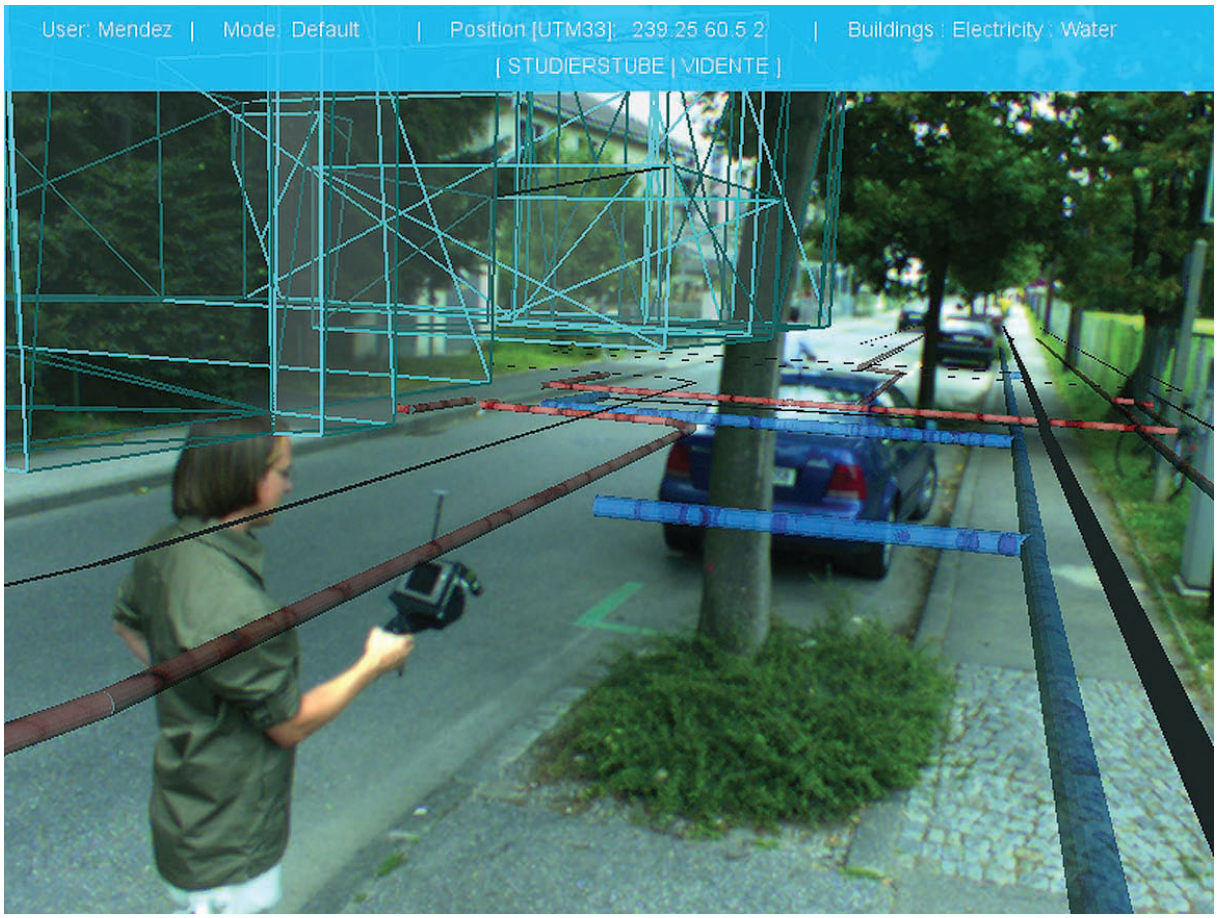


Figure 15: *An augmented reality application displays underground infrastructure created out of GIS data together with wireframe building models that help retain spatial context* [59].

4 Inverse Modeling

4.1 Problem Description

In order to use the full potential of generative techniques, the inverse problem has to be solved; i.e. what is the best generative description of one or several given instances of an object class? This problem can be interpreted in several ways. The simplest way to create a generative model out of a given 3D object is to store it in the geometry definition file format called OBJ. A simple cube may result in a file with content:

```
# definition of vertices
v  1.0 -1.0 -1.0
v  1.0 -1.0  1.0
v -1.0 -1.0  1.0
v -1.0 -1.0 -1.0
v  1.0  1.0 -1.0
v  1.0  1.0  1.0
v -1.0  1.0  1.0
v -1.0  1.0 -1.0

# definition of faces
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

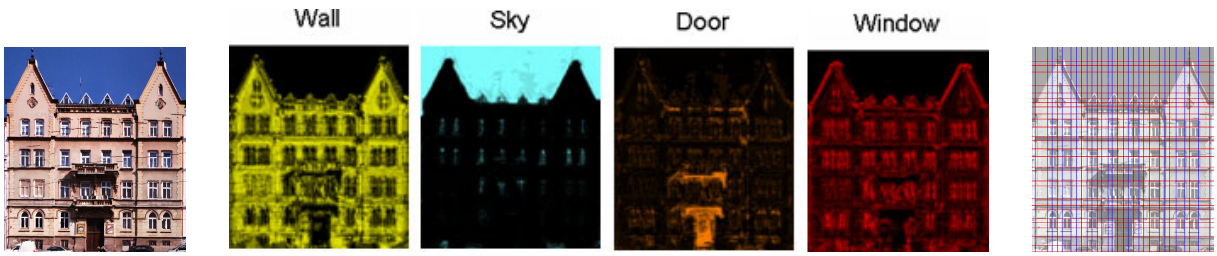
This file format structure can be interpreted as a simple language in Polish prefix notation. Prefix notation is a form of notation for logic, arithmetic, and algebra. Its distinguishing feature is that it places operators (`v`, `f`, ...) to the left of their operands / parameters. Obviously, this is not the desired result as the generative model can only represent a shape families, which are composed of only one member.

4.2 Overview on Current Approaches

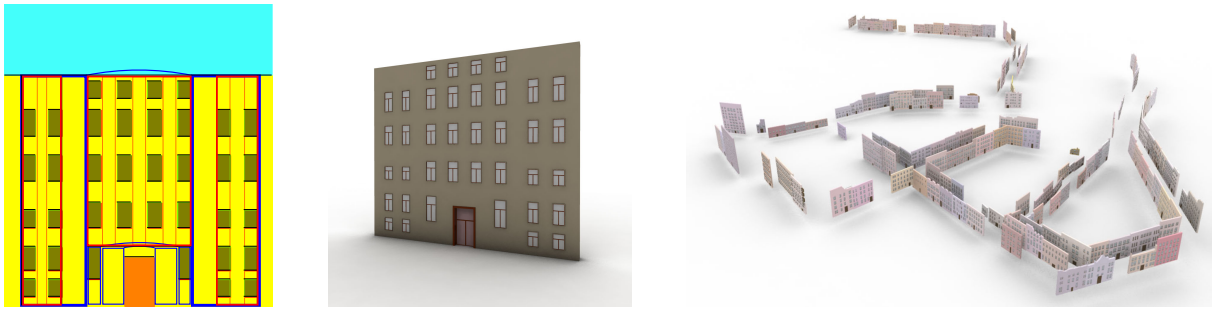
Parsing shape grammars Shape grammars can be used to describe the design space of a class of buildings / façades . An interesting application is therefore: given a set of rules and measurements of a building, typically photographs or range scans, which application of rules yields the measurements? In this context, the applied rules can also be seen as *parse tree* of a given input.

The work of Hayko Riemenschneider et al. [85] utilizes shape grammars to enhance the results of a machine learning classifier that is pre-trained to classify pixels of an orthophoto of a façade into categories like windows, walls, doors and sky. The system applies techniques from formal language parsing, a variant of the CYK algorithm, to parse a two-dimensional split grammar that consists of horizontal and vertical splits, as well as repetition and symmetry operations. In order to reduce the search space, an irregular grid is derived from the classifications, and the parsing algorithm is applied to yield the most probable application of rules that yields a classification label per grid cell. Such a parse tree can easily be converted into a procedural model, as can be seen in Figure 16.

Model synthesis The work of Paul Merell and Dinesh Manocha [60] is set in the context of automatic generation of a variation of models. The task is that given an exemplaric object (i.e. a mesh) and constraints, derive a locally similar object. The method was inspired by texture synthesis methods. These methods generate a large two-dimensional texture from a small input sample, where the result is locally similar to the input texture, but should not contain visible regularities or repetitions. The method computes a set of acceptable states, according to several types of constraints, and constructs a set of parallel planes that correspond to faces orientations of the input model. Intersections of these planes yield possible vertex positions in the output model. The system proceeds by assigning an acceptable state to a vertex and remove incompatible states in its neighborhood. The system terminates, if every vertex has been assigned a state. This process is illustrated in Figure 17.

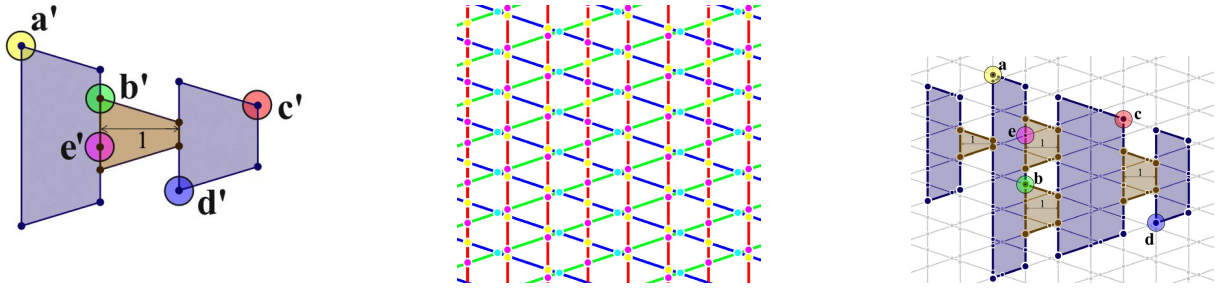


(a) *orthophoto* **(b)** *classification result* **(c)** *irregular grid*



(d) *parsed structure* **(e)** *3d rendering* **(f)** *reconstruction*

Figure 16: *A building façade (a) is classified into pre-trained categories using a machine learning classifier (b). From these classifications, a irregular grid is derived (c), and a two-dimensional split grammar is parsed (d). It can be seen that the system was able to detect horizontally repeated columns (red rectangles) and two side parts, symmetric around the middle part (connected blue rectangles). The resulting parse tree can be transformed to a generative description, which can be evaluated to geometry for rendering (e). Parts of the city of Graz were reconstructed from photographs and range scans using this technique (f).*



(a) *input model* **(b)** *parallel lines* **(c)** *acceptable output shape*

(d) *input model* **(e)** *generated variation*

Figure 17: *The work of* Paul Merell *and* Dinesh Manocha [60] *uses a mesh with constraints as input (a). The input model is analyzed, and a set of identical lines is identified. Parallel translation of these lines yields a discretization of space (b), from which a new model is synthesized that locally satisfies the constraints of the input model (c). The bottom row shows an example in 3D, where many complex buildings (e) are generated from four simple ones (d). The output contains vertices that have been constrained to intersect in four faces, some of them are circled in red.*

(a) *procedural tree model*

(b) *input model and variations generated from the procedural model*

Figure 18: *The method of* Ondrej Stava *et al.* [97] *utilizes a statistical growth model of trees that is able to generate a variety of different tree species, as can be seen in the top row (a). The system uses a statistical optimization method to find the parameters of the model, given an input exemplar. The bottom row shows an input model and three variations generated from the procedural model using the parameters that resulted from the optimization process.*

Inverse procedural modeling of trees The work of Ondrej Stava et al. [97] proposes a method that estimates the parameters of a stochastic tree model, given polygonal input tree models, such that the stochastic model produces trees similar to the input. Finding such a set of parameters is a complex task. The parameters are estimated using Markov Chain Monte Carlo (MCMC) optimization techniques. The method uses a statistical growth model that consists of 24 geometrical and environmental parameters. The authors propose a similarity measure between the statistical model and a given input mesh that consists of three parts: *shape distance*, which measures the overall shape discrepancy, *geometric distance*, which reflects the statistics of geometry of its branches, and *structural distance*, which encodes the cost of transforming a graph representation of the statistical tree model into a graph representation of the input tree model. For some examples see Figure 18.

The MCMC method has also been applied by other methods to find parameters of a statistical generative model: [101], [119], [127].

Parameter Fitting and Shape Recognition The approach presented by Torsten Ullrich and Dieter W. Fellner uses generative modeling techniques to describe a class of objects and to identify objects in real-world data e.g. laser scans [109]. The input data sets of the algorithm are a point cloud P and a generative model M. Then, the algorithm answers the questions

1. whether the point cloud can be described by the generative model and if so,
2. what are the input parameters x_0 such that $M(x_0)$ is a good description of P.

The implementation uses a hierarchical optimization routine based on fuzzy geometry and a differentiating compiler; i.e. the complete generative model description $M(x_1, \ldots, x_k)$ (including all possibly called subroutines) is differentiated with respect to the input parameters. This differentiating compiler offers the possibility to use gradient-based optimization routines in the first place. Without partial derivatives many numerical optimization routines cannot be used at all or in a limited way.

The example data set shown in Figure 19 consists of laser-scanned cups and a generative cup description. The algorithm is able to detect an instance of the generative cup. In these cases the cups' properties (position, orientation, radius, height, handle shape) are determined with only a small error.

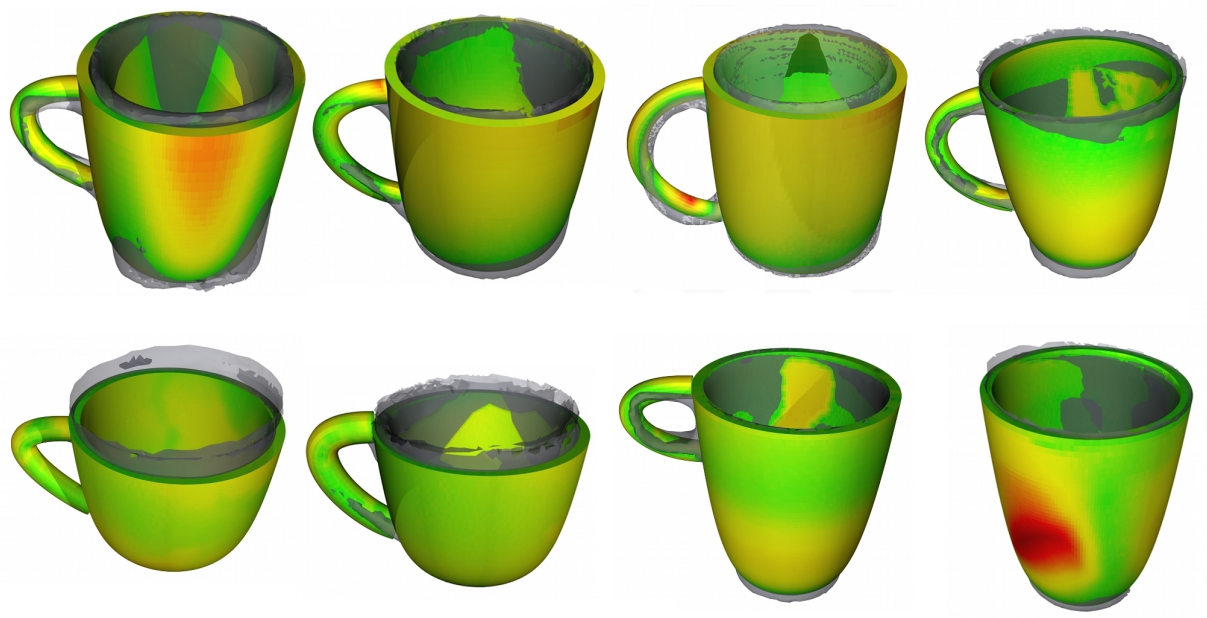


Figure 19: *The scanned cup (rendered in semi-transparent gray) have been identified as instances of the generative cup description.In these cases the cups' properties (position, orientation, radius, height, handle shape) are determined successfully.*

5 Applications

Real-world applications demonstrate the power and versatility of generative modeling techniques.

5.1 Procedural Shape Modeling

First, we demonstrate the effectiveness of procedural shape modeling for mass customization of consumer products [12]. A generative description composed of a few well-defined procedures can generate a large variety of shapes. Furthermore, it covers most of the design space defined by an existing collection of designs – in this case wedding rings, see Figure 20. The challenge is whether it is actually possible to find a suitably general generative description.

Figure 20: *The challenge: Samples from the JohannKaiser wedding ring design space. This is the input for the creation of a generative description, which is both an abstraction and a generalization of the given individual designs.*

Careful analysis of the existing collection of designs reveals that the basic shape of most rings can be defined using the following parameters:

1. a profile polygon,
2. the angular step size defined by the number of supporting profiles to be placed around the ring's center,
3. the radius, and
4. a vertex transformation function.

The idea is to decompose the design variations into a set of transformation functions. Each function transforms selected in a certain way. Effects can be combined by calling a sequence of different transformations. Figure 21a shows an example of a profile polygon, copies of which are placed radially around the origin (Figure 21b). These polygons are first converted to double-sided faces. Then, corresponding back and front sides are connected (Figure 21c) and a subdivision surface is created (Figure 21d). The profile polygon, the rotation angle, and a set of custom parameters are the input to this transformation function (Figure 21f). In a post-processing step selected faces can be colored differently (Figure 21h).

The creation of the basic shape is separated from optional steps to create engravings, change materials, or add gems. Engravings are implemented as per-vertex displacements (to maintain the option for 3D-printing) and can be applied on quadrilateral parts of the ring's mesh using half-edges to specify position and spatial extend, or projectively by using a parametric cylindrical projection.

Typically noble materials like gold, silver, and platinum are used for wedding rings. Their surfaces can be treated with various finishing techniques like polishing, brushing, or hammering. In order to account for these effects, a per-pixel shading model using an approximation of the Fresnel reflection term

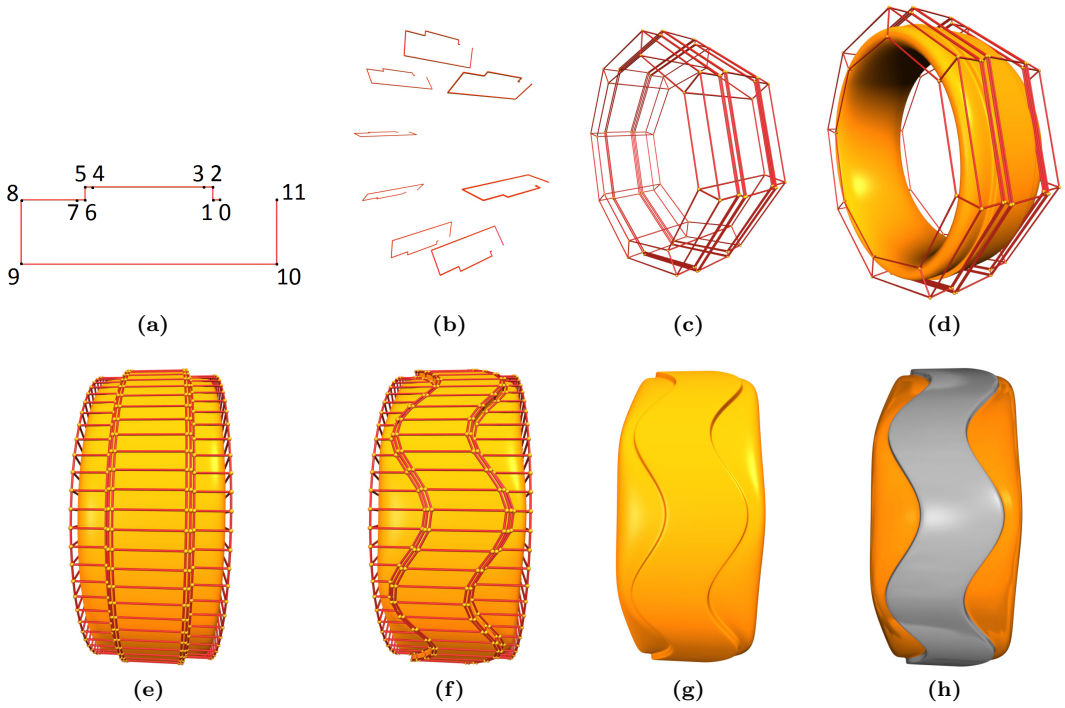


Figure 21: *Parametric wedding ring construction: n copies of the the profile polygon (a) are radially placed around the center (b). Consecutive profiles are connected using a newly-designed operator to form a control mesh (c) that has a simple toroidal structure. It defines a subdivision surface forming the actual ring shape (d). Switching the profile transformation function from identity (e) to a selective sine transformation (f) creates a decorative wave on the surface (g). Using a post-processing step the wave faces can be colored differently (h).*

is used to model anisotropic highlights. Additionally, a cube map is deployed to create visually appealing reflections. Predefined surface finishes can be applied using normal mapping techniques.

The gem model is of the round brilliant cut type. The mathematical model consists of a convex polyhedron representing the surface of the gem. Only five parameters are needed to define the shape. Procedural gem instances can be placed by specifying the apex point, an upvector, and a scale factor.

The presented approch is used in a hardware accelerated server-side rendering framework, which has been included in an online system called *REx* by *JohannKaiser*. It addresses specialist retailers like jewelers and wedding ring studios and offers an intuitive web interface for configuring and visualizing wedding rings (Figure 22). The rendering is done on a dedicated server greatly reducing the hardware requirements on the client-side. This approach is feasible for a small number of clients offering the advantage of intellectual property protection, because rendering on the client-side is reduced to displaying images (renderings coming from the server) and proxy geometry for interaction purposes. The actual 3D content and its generative description is not transferred to the client.

This work demonstrates the efficiency of procedural shape modeling for the mass customization of wedding rings. The presented generative description is able to produce a large variety of wedding rings. Figure 23 shows a few results of the parametric toolkit. The approach is considered to be used for visualization, information purposes only.

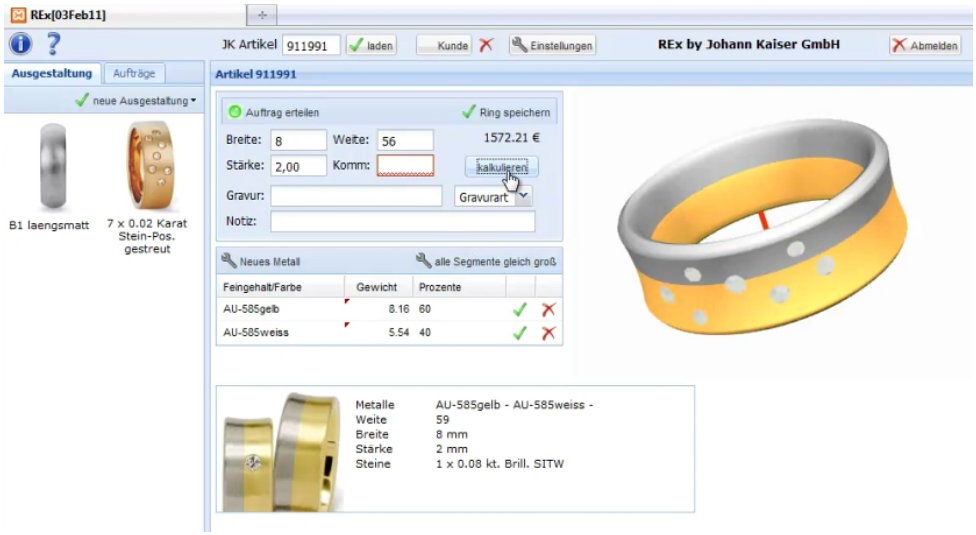


Figure 22: *The* REx *wedding ring configurator: The user can specify the various parameters of the wedding rings like profile of the ring, placement and number of diamonds, etc. The interactive 3D rendering gives an immediate feedback to the user how the choices will influence the final appearance.*

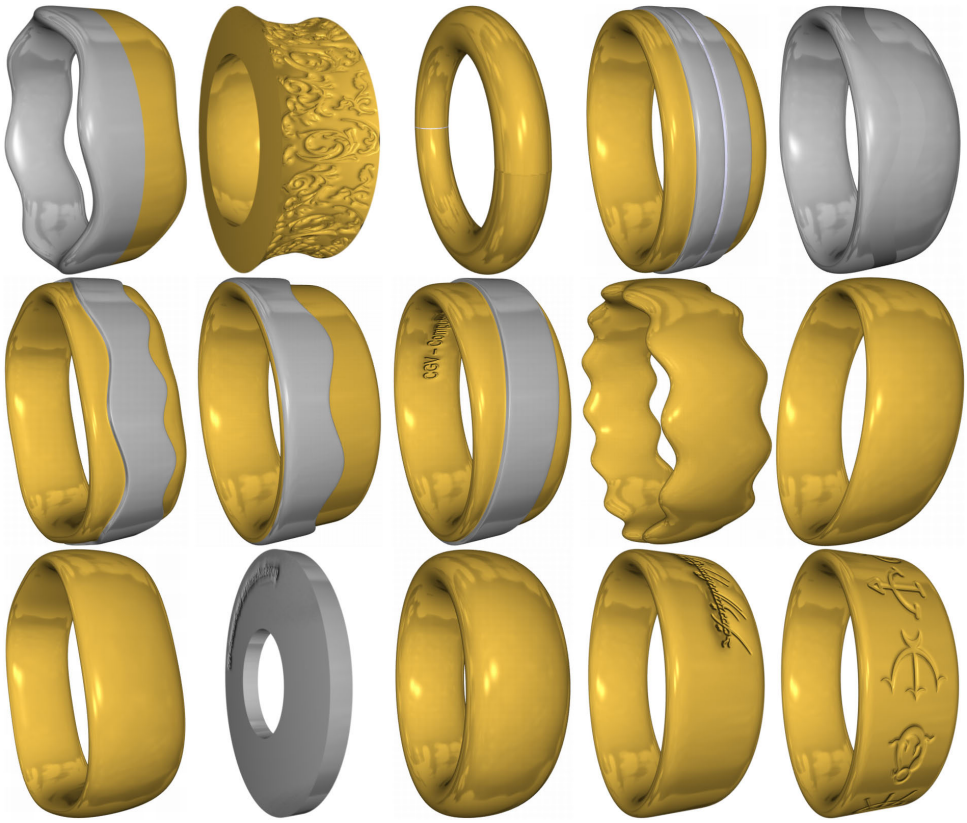


Figure 23: *The presented generative description is able to produce a large variety of wedding rings. Features like engravings, recesses, different materials, unusal forms and gems can be created and customized.*

5.2 Semantic Enrichment

With increasing number of (3D) documents, digital library services become more and more important. A digital library provides markup, indexing, and retrieval services based on metadata. In the simplest case, metadata is of the Dublin Core type [45] (title, creator/author, time of creation, etc). This is insufficient for large databases with a huge number of 3D objects, because of their versatility and rich structure. Scanned models are used in raw data collections, for documentation archival, virtual reconstruction, historical data analysis, and for high-quality visualization for dissemination purposes [94]. Navigation and browsing through the geometric models should be possible not only in 3D, but also on the semantic level. This requires higher-level semantic information. Semantic questions ("How many windows does this facade have?", "How many steps do these stairs have?", etc.) cannot be answered, if the library simply treats 3D objects as binary large objects (BLOB). The need for semantic information becomes immediately clear in the context of electronic data exchange, storage and retrieval [27], [29]. The problem of 3D semantic enrichment is closely related to the shape description problem [58]:

How to describe a shape and its structure on a higher, more abstract level?

The traditional way of classifying objects, pursued both in mathematics and, in a less formal manner, in dictionaries, is to define a class of objects by listing their distinctive properties:

> cup – a small, open container made of china, glass, metal, etc., usually having a handle and used chiefly as a receptable from which to drink tea, soup, etc.
>
> http://dictionary.reference.com

This approach is hardly realizable because of the fact that definitions cannot be self-contained. They depend on other definitions (e.g., container, handle, receptable, ...), which leads to circular dependencies that cannot be resolved automatically by strict reasoning, but rely on intuitive understanding at some point.

An alternative, non-recursive approach for describing shape uses examples. Each entry in a picture dictionary is illustrated with a photo or a drawing. This approach is widely used, for example in biology for plant taxonomy. It avoids listing an exhaustive list of required properties for each entry. However, it requires some notion of similarity, simply because the decision whether object x belongs to class A or B requires measuring the closeness of x to the exemplars $a \in A$ resp. $b \in B$. This decision can be reached by a classifier using statistics and machine learning [14], [116]. A good survey on content-based 3D object retrieval is provided by Benjamin Bustos et al. [19]. Statistical approaches clearly have their strength in discriminating object classes. However, feature-based object detection, e.g., of rectangular shapes, does not yield object parameters: width and height of a detected rectangle must typically be computed separately.

To describe a shape and its construction process, its inner structure must be known. Structural decomposition is well in line with human perception. In general, shapes are recognized and coded mentally in terms of relevant parts and their spatial configuration or structure [48]. One idea to operationalize this concept was proposed, among others, by Masaki Hilaga [42], who introduces the Multiresolution Reeb Graph, to represent the skeletal and topological structure of a 3D shape at various levels of resolution. Structure recognition is a very active branch in the field of geometry processing. The detection of shape regularities [80], self-similarities [15] and symmetries [65], [66] is important to understand a 3D shape. To summarize, structural decomposition proceeds by postulating that a certain type of general regularity or structure exists in a class of shapes. This approach clearly comes to its limits when very specific structures are to be detected, i.e., complicated constructions with many parameter interdependencies.

A possibility to describe a shape is realized by the generative modeling paradigm [75], [111]. The key idea is to encode a shape with a sequence of shape-generating operations, and not just with a list of low-level geometric primitives. In its practical consequence, every shape needs to be represented by a program, i.e., encoded in some form of programming language, shape grammar [67], modeling language [34] or modeling script [7].

The implementation of the "definition by algorithm" approach is based on a scripting language [109]: Each class of objects is represented by one algorithm M. Furthermore, each described object is a set

of high-level parameters x, which reproduces the object, if an interpreter evaluates $M(x)$. As this kind of modeling resembles programming rather than "designing", it is obvious to use software engineering techniques such as versioning and annotations. In this way, model M may contain a human-readable description of the object class it represents.

This encoding of semantic information can be used to enrich 3D objects semantically: the algorithm starts with a point cloud P and a generative model M. Without user interaction it determines a parameter set x_0, which minimizes the geometrical distance between P and $M(x_0)$. This distance d can be interpreted as a multidimensional error function of a global optimization problem. Therefore, standard techniques of function minimization can be used. Having found the global minimum x_0, the geometric distance $d(P, M(x_0))$ can be interpreted. A low value corresponds to a perfect match; i.e. the point cloud P is (at least partly) similar to $M(x)$, whereas a high value indicates no similarity. Consequently, the algorithm is able to semantically recognize instances of generative objects in real data sets.

As the computational complexity of global optimization depends on the dimensions of the error function, Torsten Ullrich et al. use a hierarchical optimization strategy with coarse model descriptions and few parameters at the beginning and detailed model descriptions at the end. This multi-step optimization determines free parameters successively, fixes them and introduces new parameters. This process stops, if the end of the hierarchy is reached, or if high error values indicate no object similarity.

In contrast to other related techniques using fitting algorithms, such as "Creating Generative Models from Range Images" by Ravi Ramamoorthi and James Arvo [82], the approach by Torsten Ullrich et al. can classify data semantically. Although Ravi Ramamoorthi and James Arvo also use generative models to fit point clouds, they modify the generative description during the fitting process. As a consequence the optimization can be performed locally with a computational complexity, which is significantly reduced. But starting with the same generative description to fit a spoon as well as a banana does not allow to generate or preserve semantic data.

An example illustrates this process. The generative model to describe a vase takes 13 parameters: $R(r_x, r_y, r_z)$ is the base reference point of the vase in 3D and $T(t_x, t_y, t_z)$ is its top-most point. The points R and T define an axis of rotational symmetry. The remaining seven parameters define the distances $d_0, \ldots, d_6$ of equally distributed Bézier vertices to the axis of rotation (see Figure 24). The resulting 2D Bézier curve defines a surface of revolution – the generative vase.

In our example, the result of the registration step is a particular generative vase out of a family of vases. A small selection of vases is shown in Figure 25 to demonstrate the versatility of the shape template. It is the best generative vase to describe the digitized artifact it has been registered to. Figure 26 illustrates this result. Please note, the registration algorithm can only modify the input parameters of the generative description. It cannot modify the description itself. As a consequence, features, which cannot be generated by the script, cannot be included in the generative result.

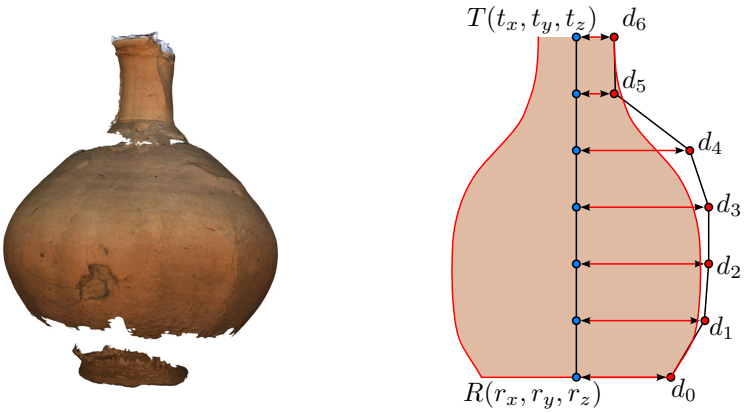


Figure 24: *A digitized artifact is often represented by a list of triangles with additional information such as textures (to describe the visual appearance) and meta data (to describe its context); e.g. the vase on the left hand side is a digitized artifact of the "Museum Eggenberg" collection. It consists of 364 774 vertices and 727 898 triangles.*
A generative model describes a 3D shape by an algorithm and a set of high-level parameters. This example of a procedural shape on the right hand side takes two points R and T in 3D and some distance values, which define the control vertices of a Bézier curve. The result is a surface of revolution to describe a vase.

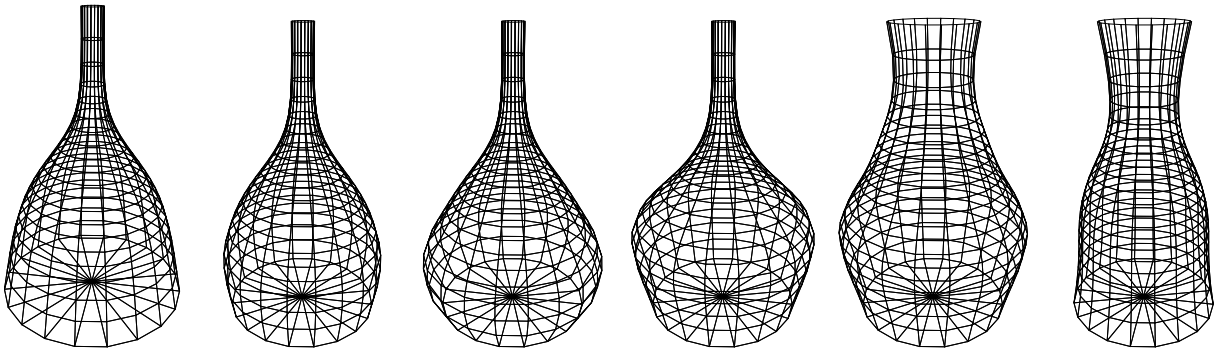


Figure 25: *The generative vase is defined by two reference points of an axis of rotation (top and bottom) and seven distances $d_0, \ldots, d_6$, which define a surface of revolution. Changing only the parameters $d_0, \ldots, d_6$, keeps the vases' height fixed and modifies the outer shape.*

Figure 26: *The digitized artifact (left) has been registered to a generative description of a vase. The result is the best-fitting vase of the complete family of vases defined by the generative description – i.e. the generative vase (right; rendered in red) is very similar to the digitized artifact (right; rendered in beige).*

5.3 Form Follows Function

The last application involves design procedures and optimization methods used in inverse modeling as well as civil engineering techniques – namely thermal analysis.

The adjustment of architectural forms to local and specific solar radiation conditions is a fundamental study. When discussing energy consumption and solar power harness in buildings, important aspects have to be taken into account, e.g., the relation between a building form and its energy behavior, and the local weather conditions on an all-year basis. Several studies were published so far, trying to answer these questions. "Form follows energy" has become an omnipresent dogma in architecture, but its realization is difficult. The manual analysis of the various relations between form, volume, and energy consumption has to face many – not only numerical – problems. Christina R. Lemke points out these difficulties [51]. Already existing, common building forms (64 different shapes in total) are analyzed and discussed.

In "Impact of building shape on thermal performance of office buildings in Kuwait" [4] a simplified analysis method to estimate the impact of the building shape on energy efficiency of office buildings is presented. This method is based on results obtained from a comprehensive whole building energy simulation analysis. The simplified method, presented in the form of a correlation equation, is, according to the authors, suitable for architects during preliminary design phase in order to assess the impact of shape on the energy efficiency of office buildings. Although a building's shape has a significant impact on the energy costs, and although these maintenance costs surpass the initial costs in several orders of magnitude, no general guidelines are available for architects on the impact of a building's shape on its energy efficiency.

Several studies try to reduce the complex relationship of "Form follows Energy" to a few equations. Naresh K. Bansal and Amitabh Bhattacharya[8] as well as Ramzi Ourghi et al. [73] derived simplified analysis methods. Although some simplifications are acceptable for special scenarios, all of them may introduce a significant, systematic error as discussed by Patrick Depecker et al. [23]. According to them many parameters are often missing, such as orientation or climate in order to produce reliable results. The climate aspects are usually simplified to the season with the harshest weather, often forgetting that temperatures in cities at certain latitudes can drop below thermal comfort limits in winter and that temperatures in cities at other latitudes often raise above thermal comfort limits in summer [71].

Similar to our example, Veerparkash P. Sethi analyzes five of the most commonly used single span shapes of greenhouses (even-span, uneven-span, vinery, modified arch and Quonset types) [92]. Hence, in this study an attempt has been made to select the most suitable shape and orientation of a greenhouse for different climatic zones (latitudes) on the basis of total solar radiation availability and its subsequent effect on the greenhouse air temperature. From this study important conclusions were taken regarding greenhouse shapes and their relation to maximum solar radiation extraction by using several different simulations.

Using generative modeling techniques we perform an optimization within a configuration space of a complete family of buildings. The numerical optimization routine used in "Generative Modeling and Numerical Optimization for Energy-Efficient Buildings" [115] has to tackle several problems which occur in many complex, nonlinear optimization tasks: the choice of initial values and the problem of local minima. Both problems are addressed by a modified differential evolution method. Differential evolution is a heuristic approach for minimizing nonlinear, non differentiable space functions. It is described in "Differential Evolution: A simple and efficient heuristic for global optimization over continuous spaces" [100], [16]. The differential evolution method for minimizing continuous space functions can also handle discrete problems by embedding the discrete parameter domain in a continuous domain. It converges quite fast and is inherently parallel, which allows an execution on a cluster of computers.

The greenhouse to optimize is based on an extruded, quartic Bézier curve: the first and the last control point have distance $width$, which may vary from $3m$ to $50m$. The second control point is located directly above the first one at a fixed height of $2m$. Symmetrically, the second to last control point is directly above the last one. The third control point can be chosen freely. It is parameterized by its absolute height $height \in [4m - 75m]$ and its relative position $ratio \in [0, 1]$. Finally, the greenhouse has a geographical orientation $\alpha \in [0° - 180°]$ measured towards northern direction. These parameters define the upright projection and the vertical section as illustrated in Figure 27.

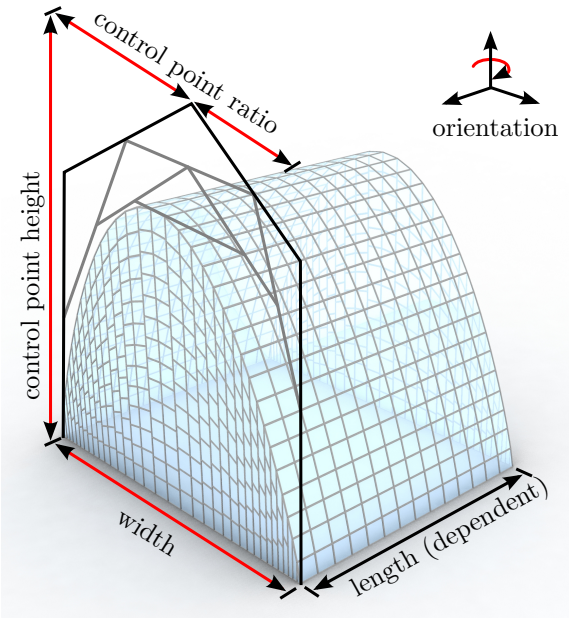


Figure 27: *This greenhouse is defined by four parameters: its width, a free control point of a Bézier curve (with two free coordinates), and an overall orientation within its environment. This image sketches the construction process and shows the optimization result at the same time – the optimal solution according to energy needs.*

The surface area under the defined Bézier curve has been calculated analytically. Its symbolic representation is included as a function in the scripted model in order to determine the greenhouse's length – a non-free parameter. If the length were a free parameter, the minimization would set it to zero. The resulting greenhouse would have no volume and no energy need. To exclude such trivial solutions the volume has to be a constant.

Having scripted the construction process our framework generates a population of parameter vectors $\mathbf{x}_{1,1}, \ldots, \mathbf{x}_{1,n}$, evaluates the script with each parameter and passes the corresponding geometry to Autodesk Ecotect$^{(\mathrm{TM})}$, which performs an energy-efficiency analysis. This analysis returns an approximation for the greenhouse's annual energy needs – the back coupling for a gradient-free optimization method. The outcome of the numerical optimization process is the best building (within the family of buildings described by its script) concerning energy-efficiency.

The generative models are scripted in a JavaScript dialect. The geometry is encoded in a simple mesh structure, which contains additional, semantic information and markup. For example, each polygon is annotated with

- its meaning (floor, roof, wall, window, etc.),
- its material (a reference name of the material data base of Autodesk Ecotect$^{(\mathrm{TM})}$: concrete, glass, etc.), and
- its hierarchy level (e.g. each window references its surrounding wall).

Furthermore, the generative model contains global meta data about energy respectively heat sources; especially

- its location (a geographically referenced climate data file in WEA format according to Drury B. Crawley [22]) in order to include solar energy effects and the building's environment as well as
- occupancy and activity data to include waste heat into the energy efficiency calculation.

Each trial vector of a new generation $\mathbf{x}^{\star}_{k+1,i}$ has to be interpreted as a parameter set of a generative model in order to create the geometric model described above. For performance and usability reasons the procedural model description source is translated and compiled from JavaScript to Java byte code, which can be loaded into the optimization framework dynamically.

While the optimization routine and the script evaluations are performed on a server, several clients perform the energy efficiency calculation. Each client fetches a geometric model (the geometry created by a single script evaluation) and passes it to Autodesk Ecotect$^{(TM)}$ using its public API. Ecotect Analysis's is based on the admittance method described in "Thermal performance of buildings" [2], [3] and comprehends parts of "Transmission heat loss coefficient calculation method" [1]. According to Jan L. M. Hensen et al. the underlying assumption of the admittance method is that the internal temperature of any building will always tend towards the local 24-hour mean outdoor temperature [40]. The admittance of a building element represents its ability to absorb and release heat energy and defines its dynamic response to cyclic fluctuations in temperature conditions. Its unit is power per area and difference in temperature: $\frac{W}{m^2 \cdot K}$.

Any incident solar or internal heat gains will first act to raise internal air and surface temperatures, the effect of which is to increase conductive heat losses through the fabric and air infiltration losses through windows and other openings. When the total of all heat losses equals the total gain, the internal temperature stabilizes. The admittance method encapsulates the effects of conductive heat flow through building fabric, infiltration and ventilation through openings, direct solar gains through transparent materials, indirect solar gains through opaque elements, and internal heat gains from equipment, lights and people. Using a two-pass algorithm, it also accounts for the effects of inter-zonal heat flow [41].

The applied optimization process evaluated the generative model 3,236 times. The evaluations are plotted in Figure 28. For a fixed volume of $1,000m^3$ the parameter vector (21.19, 53.91, −0.002, 94.70) generates the best greenhouse and geometric shape concerning energy efficiency. This final result has been rendered into the construction sketch of Figure 27.

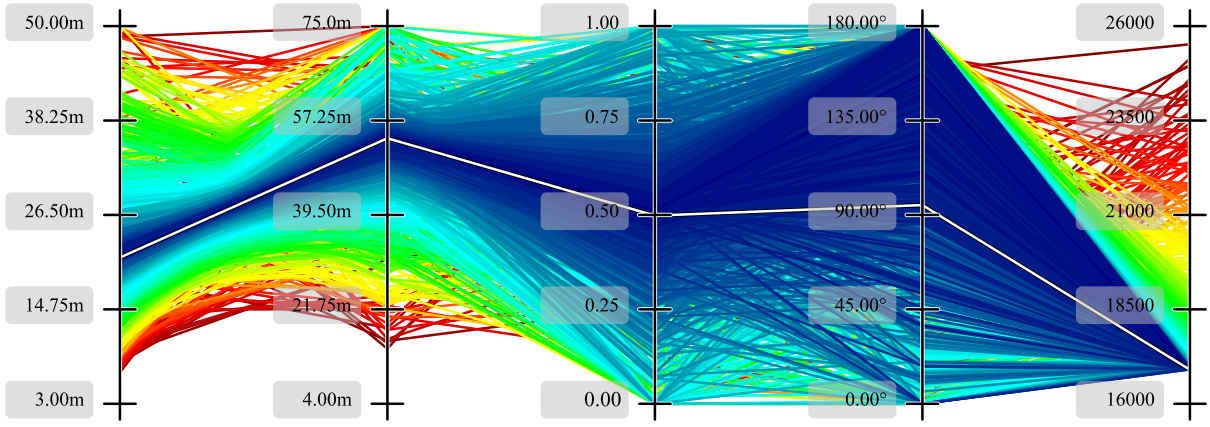


Figure 28: *The optimization algorithm evaluated 3,236 instances of the greenhouse sketched in Figure 27. In this diagram all evaluations have been plotted in parallel coordinates (colored and sorted back-to-front) according to their energy efficiency (last coordinate). While the greenhouse's width (first coordinate) and the height of the free control point (second coordinate) have a big influence on its energy needs, the free control point's horizontal placement (third coordinate) and its orientation (fourth coordinate) play a minor role. The optimal solution is plotted in white.*

In this example we present an optimization framework composed by simulation tools, numerical analysis and procedural modeling techniques. This approach supports architects and engineers when designing new buildings and new products; it offers an innovative and promising combination of design and engineering. The demonstration example shows the immediate applicability for such a framework and its power – not only in the field of architecture but also in many other areas where the need of optimization, combined with complex simulations and procedural modeling is of great importance. The coupling of these processes, instead of successively handling one process after the other, opens a new and better way of integrated optimization possibilities, it saves time and – in these concrete examples – even energy.
This new access to product design is opening the door to new possibilities for the user. It relieves the

user from additional, interdisciplinary burdens: the designer can concentrate on the design, while the civil engineer can focus on engineering aspects. This new approach based on procedural modeling can be used in different fields of interest to solve real world problems.

The automated optimization framework shortens development time significantly compared to conventional methods with separated design and optimization procedures where the creation steps are processed successively and repeated until the desired functional design result has been achieved. There is also a great chance that the optimization framework will propose and check for new unknown and unexpected possibilities, that a user may never had thought of before. Therefore, this approach is opening up new frontiers for the development of more innovative and sustainable ideas.

6 Open Questions

According to Sven Havemann and Dieter W. Fellner [27], [28], [36], [38] several research challenges have to be met:

Classification of shape representations Dozens if not hundreds of digital representations for shape exist, from points and triangles over parametric and implicit (level-set) surfaces to generative and parametric models, and for each there are several sub-representations with different attribute sets. Conversion is usually not possible without loss of information. So far, no exhaustive classification is available that would allow more uniform approaches and algorithms to be formulated in a generic way to cover a whole class of shape representations sharing similar properties.

A sustainable encoding in a 3D file format There is a plethora of different incompatible file formats for storing shape representations in different ways. For many important shape representations there is not even a commonly accepted file format (point clouds, range maps, compressed meshes), because for a 3D software package to support a given specified file format is difficult and expensive. Most commercial systems therefore have their own formats which have become proprietary de-facto standards. A commonly accepted general file format approach would be highly desirable.

Generic, stable, and detailed 3D markup The problem is to attach semantic information to a portion of a shape (point, line, surface, volume) in a sustainable way. The shape markup ("nose of a statue", or even "noise of a statue") should survive simple mesh editing operations ("extracting the head"), so that a minimum of sustainability is guaranteed. Again, the problem is to define this in a generic way compatible to solutions found for the first two problems.

Generic query operations Once a markup is attached, the markup needs to be queried. That means the denoted geometric entities need to be identified, for example by simply highlighting a portion of a surface on the screen. A generic approach is needed that works for many shape representations and encodings. Purely geometric shape query operations that need to be standardized are ray casting/picking, screen-based selection, distance query (how close is a point to an object's surface), and box containment.

Para data, processing history, provenance Three dimensional models are often obtained from combining other 3D models, e.g., by stitching together partial scans or by arranging many objects in a scene. For professional applications it is highly desirable to be able to assess the authenticity of a given set of data by tracing back its digital provenance to see which processing steps it underwent. Unfortunately, most 3D software does not store this information, and even if it does in some rare cases, there is no general standard for it.

Close the semantic gap, determine the meaning of shape The goal is to assign a meaning (car, house, screw) to a given 3D model only by considering its geometry. This entails classical questions such as measures for shape similarity, shape retrieval, and query-by-example. But also more fine-grained questions such as determining dimensions, parameters, part-of relationships as well as symmetries, self similarity (ornaments, patterns) and speculation about deteriorated parts need to be considered.

Maintain the relation of shape and meaning consistent Assuming that the meaning of a shape was determined, how can that information be stored in a sustainable way? Currently there is no commonly accepted, domain independent method to store and exchange the meaning of a shape. Note that this not only requires solving the previous problems, but it additionally requires a common approach for knowledge engineering, e.g., using standardized shape ontologies to express the relations between the different shapes.

All these problems can be illustrated in problems that occur when manufacturing techniques are combined with virtual process descriptions. This connection is vital in many fields of applications, but unfortunately it is still missing. An example demonstrates the problem: the installation of a flushmounted socket. In the real world, this process involves a wall, a socket, and electric cable. Furthermore, some tools are needed to perform operations such as opening and closing cable ducts, etc. Simply speaking, one has to identify the wall, drill a hole in the desired position, run a cable to the hole and install a socket. If

this real-world process should be described in the digital world – e.g. in the context of maintenance and building information modeling – it is normally reduced to its end result. The process itself is usually not described. A digital representation of the process would require means of describing the physical work involved. This reveals two problems: the conversion problem and the documentation problem.

Even the reduction of the process documentation to its end result has to cope with a serious problem. In a CAD model, the wall may be designed using a polygonal representation. The socket is most likely represented as a NURBS model; non-uniform b-splines (NURBS) are a common representation in industrial design. These are only two representations in a zoo of geometric designs created in the field of CAD over the past few decades. In order to perform operations on the representations, they have to be converted introducing conversion errors, numerical inaccuracy and possible loss of data; i.e. they open Pandora's Box of file format conversion. In order to cover n file formats it is necessary to have up to n^2 converters, or with an intermediate representation twice the conversion errors and only $2n$ converters in the ideal case.

Even worse, the end result of the virtually modeled operation has no connection to the real-world process. The process itself is still not represented and the differences between the models are only documented indirectly. Neither the tools nor the industrial parts and processes are documented digitally; i.e. if there are several ways to achieve the same result, the end-result description does not say, which way has been chosen. This problem needs to be solved.

References

[1] **International Organization for Standardization (ISO) 13789:1999 (Transmission heat loss coefficient calculation method)**, 1999.

[2] **International Organization for Standardization (ISO) 13791:2004 (Thermal performance of buildings)**, 2004.

[3] **International Organization for Standardization (ISO) 13792:2005 (Thermal performance of buildings)**, 2005.

[4] Adnan Al Anzi, Donghyun Seo, and Moncef Krarti. **Impact of building shape on thermal performance of office buildings in Kuwait**. *Energy Conversion and Management*, 50:822–828, 2009.

[5] David Arnold. **Procedural methods for 3D reconstruction**. *Recording, Modeling and Visualization of Cultural Heritage*, 1:355–359, 2006.

[6] Franz Aurenhammer. **Weighted skeletons and fixed-share decomposition**. *Computational Geometry*, 40(2):93 – 101, 2008.

[7] Autodesk. **Autodesk Maya API**. *White Paper*, 1:1–30, 2007.

[8] Naresh K. Bansal and Amitabh Bhattacharya. **Parametric equations for energy and load estimations for buildings in India**. *Applied Thermal Engineering*, 29:3710–3715, 2009.

[9] Johannes Behr, Patrick Dähne, Yvonne Jung, and Sabine Webel. **Beyond the Web Browser – X3D and Immersive VR**. *IEEE Virtual Reality Tutorial and Workshop Proceedings*, 28:5–9, 2007.

[10] Bedrich Benes, Ondrej Stava, Radomir Mech, and Gavin Miller. **Guided Procedural Modeling**. *Computer Graphics Forum*, 30:325–334, 2011.

[11] René Berndt, Dieter W. Fellner, and Sven Havemann. **Generative 3D Models: a Key to More Information within less Bandwidth at Higher Quality**. *Proceeding of the 10th International Conference on 3D Web Technology*, 1:111–121, 2005.

[12] René Berndt, Christoph Schinko, Ulrich Krispel, Volker Settgast, Sven Havemann, Eva Eggeling, and Dieter W. Fellner. **Ring's Anatomy – Parametric Design of Wedding Rings**. *Proceedings International Conference on Creative Content Technologies*, 4:72–78, 2012.

[13] Alexander Berner, Martin Bokeloh, Michael Wand, Andreas Schilling, and Hans-Peter Seidel. **A Graph-Based Approach to Symmetry Detection**. *Symposium on Volume and Point-Based Graphics*, 5:1–6, 2008.

[14] Christopher M. Bishop. **Pattern Recognition and Machine Learning**. Springer, 2007.

[15] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. **A Connection between Partial Symmetry and Inverse Procedural Modeling**. *Proceedings of ACM SIGGRAPH 2010*, 29:104:1–104:10, 2010.

[16] Janez Brest, Saso Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. **Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems**. *IEEE Transactions on Evolutionary Computation*, 10:646–657, 2006.

[17] Frank Breuel, René Bernd, Torsten Ullrich, Eva Eggeling, and Dieter W. Fellner. **Mate in 3D – Publishing Interactive Content in PDF3D**. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 15:110–119, 2011.

[18] Don Brutzman. **The virtual reality modeling language and Java**. *Communications of the ACM*, 41(6):57 – 64, 1998.

[19] Benjamin Bustos, Daniel Keim, Dietmar Saupe, and Tobias Schreck. **Content-based 3D Object Retrieval**. *IEEE Computer Graphics and Applications*, 27(4):22–27, 2007.

[20] Amaresh Chakrabarti, Kristina Shea, Robert Stone, Jonathan Cagan, Matthew Campbell, Noe Vargas-Hernandez, and Kirtsin L. Wood. **Computer-Based Design Synthesis Research: An Overview**. *Journal of Computing and Information Science in Engineering*, 11:021003:1–10, 2011.

[21] Noam Chomsky. **Three models for the description of language**. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[22] Drury B. Crawley. **Which Weather Data Should You Use for Energy Simulations of Commercial Buildings?** *ASHRAE Transactions*, 104:498–515, 1998.

[23] Patrick Depecker, Christophe Menezo, Joseph Virgone, and Stephane Lepers. **Design of buildings shape and energetic consumption**. *Building and Environment*, 36:627–635, 2001.

[24] Oliver Deussen and Bernd Lintermann. **Digital Design of Nature: Computer Generated Plants and Organics**. Springer, 2005.

[25] Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli, and Roberto Scopigno. **SpiderGL: a JavaScript 3D graphics library for next-generation WWW**. *Proceedings of the 15th International Conference on Web 3D Technology*, 15:165–174, 2010.

[26] Bruce Eckel. **Thinking in C++: Introduction to Standard C++, Practical Programming**. Prentice Hall, 2003.

[27] Dieter W. Fellner. **Graphics Content in Digital Libraries: Old Problems, Recent Solutions, Future Demands**. *Journal of Universal Computer Science*, 7:400–409, 2001.

[28] Dieter W. Fellner and Sven Havemann. **Striving for an adequate vocabulary: Next generation metadata**. *Proceedings of the 29th Annual Conference of the German Classification Society*, 29:13 – 20, 2005.

[29] Dieter W. Fellner, Dietmar Saupe, and Harald Krottmaier. **3D Documents**. *IEEE Computer Graphics and Applications*, 27(4):20–21, 2007.

[30] Gerald Frank. **Optimization of the Product Creation Process by Automated Design to Cost**. *Proceedings of the International Conference on Intelligent Engineering Systems (INES)*, 15:363–367, 2011.

[31] Gerald Frank and Christian Hillbrand. **Automatic support of standardization processes in design models**. *Proceedings of the International Conference on Intelligent Engineering Systems (INES)*, 16:393–398, 2012.

[32] Eric Galin, Adrien Peytavie, Nicolas Marechal, and Eric Guerin. **Procedural Generation of Roads**. *Computer Graphics Forum*, 29:429–438, 2010.

[33] Simon Haegeler, Pascal Müller, and Luc Van Gool. **ProceduralModeling for Digital Cultural Heritage**. *Journal on Image and Video Processing*, 9:1–11, 2009.

[34] Sven Havemann. **Generative Mesh Modeling**. *PhD-Thesis, Technische Universität Braunschweig, Germany*, 1:1–303, 2005.

[35] Sven Havemann and Dieter W. Fellner. **Generative Parametric Design of Gothic Window Tracery**. *Proceedings of the 5th International Symposium on Virtual Reality, Archeology, and Cultural Heritage*, 1:193–201, 2004.

[36] Sven Havemann and Dieter W. Fellner. **Seven Research Challenges of Generalized 3d Documents**. *IEEE Computer Graphics and Applications*, 3:70–76, 2007.

[37] Sven Havemann, Volker Settgast, Harald Krottmaier, and Dieter W. Fellner. **On the Integration of 3D Models into Digital Cultural Heritage Libraries**. *Proceedings of the 7th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, 1:161–169, 2006.

[38] Sven Havemann, Torsten Ullrich, and Dieter W. Fellner. **The Meaning of Shape and some Techniques to Extract It**. *Multimedia Information Extraction*, 1:81–98, 2012.

[39] J.L. Heiberg, editor. **Euclid's Elements of Geometry**. Fitzpatrick, Richard, 2007.

[40] Jan L. M. Hensen and Marija Radosevic. **Ecotect Methodology**. online, 2004.

[41] Jan L. M. Hensen and Marija Radosevic. **Teaching building performance simulation – some quality assurance issues and experiences**. *Proceedings of PLEA International Conference on Passive and Low Energy Architecture*, 21:1209–1214, 2004.

[42] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. **Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes**. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:203–212, 2001.

[43] Christoph M. Hoffmann and Ku-Jin Kim. **Towards valid parametric CAD models**. *Computer Aided Design*, 33:81–90, 2001.

[44] Bernhard Hohmann, Ulrich Krispel, Sven Havemann, and Dieter W. Fellner. **Cityfit: High-Quality Urban Reconstructions by Fitting Shape Grammars to Images and Derived Textured Point Clouds**. *Proceedings of the ISPRS International Workshop 3D-ARCH*, 3:61–68, 2009.

[45] Dublin Core Metadata Initiative. **Dublin Core Metadata Initiative**. http://dublincore.org/, 1995.

[46] Hanna Jedrzejuk and Wojciech Marks. **Optimization of shape and functional structure of buildings as well as heat source utilisation example**. *Building and Environment*, 37:1249–1253, 2002.

[47] Tom Kelly and Peter Wonka. **Interactive Architectural Modeling with Procedural Extrusions**. *ACM Transactions on Graphics*, 30:14:1–15, 2011.

[48] Brett D. King and Michael Wertheimer. **Max Wertheimer & Gestalt Theory**. Transaction Publishers, 2005. ISBN 0-7658-0258-9.

[49] Lars Krecklau and Leif Kobbelt. **Procedural Modeling of Interconnected Structures**. *Computer Graphics Forum*, 30:335–344, 2011.

[50] Lars Krecklau, Darko Pavic, and Leif Kobbelt. **Generalized Use of Non-Terminal Symbols for Procedural Modeling**. *Computer Graphics Forum*, 29:2291–2303, 2010.

[51] Christina Rullán Lemke. **ArchitekturForm & SolarEnergie**. Curvillier, 2010.

[52] Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. **Interactive Modeling of City Layouts using Layers of Procedural Content**. *Computer Graphics Forum*, 30:345–354, 2011.

[53] Markus Lipp, Peter Wonka, and Michael Wimmer. **Interactive Visual Editing of Grammars for Procedural Architecture**. *ACM Transactions on Graphics*, 27(3):1–10, 2008.

[54] Markus Lipp, Peter Wonka, and Michael Wimmer. **Parallel Generation of Multiple L-Systems**. *Computers & Graphics*, 34:585–593, 2010.

[55] Benoit B. Mandelbrot. **The Fractal Geometry of Nature**. W. H. Freeman and Co., 1982.

[56] David Marcheix and Guy Pierra. **A Survey of the Persistent Naming Problem**. *Proceedings of the ACM Symposium on Solid Modeling and Applications*, 7:13–22, 2002.

[57] George E. Martin. **Geometric Constructions**. Springer, 1998.

[58] Mark T. Maybury, editor. **Multimedia Information Extraction**. John Wiley & Sons, 2012.

[59] Erick Mendez, Gerhard Schall, Sven Havemann, Dieter W. Fellner, Dieter Schmalstieg, and Sebastian Junghanns. **Generating Semantic 3D Models of Underground Infrastructure**. *IEEE Computer Graphics and Applications*, 28:48–57, 2008.

[60] Paul Merrell and Dinesh Manocha. **Continuous Model Synthesis**. *ACM Transactions on Graphics*, 27:158:1–9, 2008.

[61] Paul Merrell and Dinesh Manocha. **Model Synthesis: A General Procedural Modeling Algorithm**. *IEEE Transactions on Visualization and Computer Graphics*, 17:715–728, 2010.

[62] Paul Merrell, Eric Schkufza, and Vladlen Koltun. **Computer-generated residential building layouts**. *ACM Transactions on Graphics*, 29:181:1–11, 2010.

[63] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. **Interactive Furniture Layout Using Interior Design Guidelines**. *ACM Transactions on Graphics*, 30:87:1–10, 2011.

[64] William J. Mitchell. **The Logic of Architecture: Design, Computation, and Cognition**. MIT Press, 1990.

[65] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. **Partial and approximate symmetry detection for 3D geometry**. *ACM Transactions on Graphics*, 25:560 – 568, 2006.

[66] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. **Symmetrization**. *International Conference on Computer Graphics and Interactive Techniques*, 26:1–8, 2007.

[67] Pascal Müller, Peter Wonka, Simon Haegler, Ulmer Andreas, and Luc Van Gool. **Procedural Modeling of Buildings**. *Proceedings of 2006 ACM Siggraph*, 25(3):614–623, 2006.

[68] Przemyslaw Musialski, Michael Wimmer, and Peter Wonka. **Interactive Coherence-Based Facade Modeling**. *Computer Graphics Forum*, 31:661–670, 2012.

[69] Przemyslaw Musialski, Peter Wonka, Daniel G. Aliaga, Michael Wimmer, Luc van Gool, and Werner Purgathofer. **A Survey of Urban Reconstruction**. *Proceedings of EUROGRAPHICS, State of the Art Report (STAR)*, 31:1–28, 2012.

[70] NVidia. **NVIDIA CUDA C Programming Guide**.

[71] Ahmad Okeil. **A holistic approach to energy efficient building forms**. *Energy and Buildings*, 42:1437–1444, 2010.

[72] Review Board OpenGL Architecture. **OpenGL Reference Manual**. Addison-Wesley Publishing Company, 1993.

[73] Ramzi Ourghi, Adnan Al Anzi, and Moncef Krarti. **A simplified analysis method to predict the impact of shape on annual energy use for office buildings**. *Energy Conversion and Management*, 48:300–305, 2007.

[74] John K. Ousterhout. **Scripting: Higher Level Programming for the 21st Century**. *IEEE Computer Magazine*, 31(3):23–30, 1998.

[75] Mine Özkar and Sotirios Kotsopoulos. **Introduction to shape grammars**. *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH 2008 (course notes)*, 36:1–175, 2008.

[76] Yogi Parish and Pascal Müller. **Procedural Modeling of Cities**. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:301–308, 2001.

[77] Terence Parr. **Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages**. Pragmatic Bookshelf, 2010.

[78] Alexander Pasko and Valery Adzhiev. **Function-based shape modeling: mathematical framework and specialized language**. *Lecture Notes in Computer Science*, 2930:132–160, 2004.

[79] Gustavo Patow. **User-Friendly Graph Editing for Procedural Modeling of Buildings**. *IEEE Computer Graphics and Applications*, 32:66–75, 2012.

[80] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. **Discovering structural regularity in 3D geometry**. *ACM Transactions on Graphics*, 27:#43, 1–11, 2008.

[81] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. **The Algorithmic Beauty of Plants**. Springer-Verlag, 1990.

[82] Ravi Ramamoorthi and James Arvo. **Creating Generative Models from Range Images**. *Proceedings of ACM Siggraph*, 1:195–204, 1999.

[83] Casey Reas, Ben Fry, and John Maeda. **Processing: A Programming Handbook for Visual Designers and Artists**. The MIT Press, 2007.

[84] Dirk Reiners, Gerrit Voss, and Johannes Behr. **OpenSG: Basic concepts**. *Proceedings of OpenSG Symposium 2002*, 1:1–7, 2002.

[85] Hayko Riemenschneider, Ulrich Krispel, Wolfgang Thaller, Michael Donoser, Sven Havemann, Dieter W. Fellner, and Horst Bischof. **Irregular lattices for complex shape grammar facade parsing**. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 25:1640–1647, 2012.

[86] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. **Modeling Procedural Knowledge – a generative modeler for cultural heritage**. *Proceedings of EUROMED 2010 - Lecture Notes on Computer Science*, 6436:153–165, 2010.

[87] Christoph Schinko, Martin Strobl, Torsten Ullrich, and Dieter W. Fellner. **Scripting Technology for Generative Modeling**. *International Journal On Advances in Software*, 4:308–326, 2011.

[88] Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. **Simple and Efficient Normal Encoding with Error Bounds**. *Proceedings of Theory and Practice of Computer Graphics*, 29:63–66, 2011.

[89] Christoph Schinko, Torsten Ullrich, and Dieter W. Fellner. **Minimally Invasive Interpreter Construction – How to reuse a compiler to build an interpreter**. *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 3:38–44, 2012.

[90] Christoph Schinko, Torsten Ullrich, Thomas Schiffer, and Dieter W. Fellner. **Variance Analysis and Comparison in Computer-Aided Design**. *Proceedings of the International Workshop on 3D Virtual Reconstruction and Visualization of Complex Architectures*, XXXVIII-5/W16:3B21–25, 2011.

[91] Thomas W. Sederberg and Scott R. Parry. **Free-form Deformation of Solid Geometric Models**. *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, 13:151–160, 1986.

[92] Veerparkash P. Sethi. **On the selection of shape and orientation of a greenhouse: Thermal modeling and experimental validation**. *Solar Energy*, 83:21–38, 2009.

[93] Volker Settgast. **Processing Semantically Enriched Content for Interactive 3D Visualizations**. *PhD-Thesis, Technische Universität Graz, Austria*, 1:1–233, 2013.

[94] Volker Settgast, Torsten Ullrich, and Dieter W. Fellner. **Information Technology for Cultural Heritage**. *IEEE Potentials*, 26(4):38–43, 2007.

[95] John M. Snyder. **Generative modeling for computer graphics and CAD**. Academic Press Professional, Inc., 1992.

[96] John M. Snyder and James T. Kajiya. **Generative modeling: a symbolic system for geometric modeling**. *Proceedings of 1992 ACM Siggraph*, 1:369–378, 1992.

[97] Ondrej Stava, Sören Pirk, Julian Kratt, Baoquan Chen, Radomir Měch, Oliver Deussen, and Bedrich Benes. **Inverse Procedural Modelling of Trees**. *Computer Graphics Forum*, page to appear, 2014.

[98] Markus Steinberger, Michael Kenzel, Bernhard Kainz, Jörg Müller, Wonka Peter, and Dieter Schmalstieg. **Parallel Generation of Architecture on the GPU**. *Computer Graphics Forum*, 33:73–82, 2014.

[99] George Stiny and James Gips. **Shape Grammars and the Generative Specification of Painting and Sculpture**. *Best computer papers of 1971*, 1:125–135, 1971.

[100] Rainer Storn and Kenneth Price. **Differential Evolution: A simple and efficient heuristic for global optimization over continuous spaces**. *Journal of Global Optimization*, 11:341–359, 1997.

[101] Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomir Mech, and Vladlen Koltun. **Metropolis Procedural Modeling**. *ACM Transactions on Graphics*, 30:11:1–14, 2011.

[102] W. Thaller, U. Krispel, S. Havemann, and D. Fellner. **Implicit Nested Repetition in Dataflow for Procedural Modeling**. *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, 3:45–50, 2012.

[103] Wolfgang Thaller, Ulrich Krispel, René Zmugg, Sven Havemann, and Dieter W. Fellner. **A Graph-Based Language for Direct Manipulation of Procedural Models**. *International Journal on Advances in Software*, 6:225–236, 2013.

[104] Wolfgang Thaller, Ulrich Krispel, René Zmugg, Sven Havemann, and Dieter W. Fellner. **Shape Grammars on Convex Polyhedra**. *Computers & Graphics*, 37:707–717, 2013.

[105] Wolfgang Thaller, René Zmugg, Ulrich Krispel, Martin Posch, Sven Havemann, and W. Fellner Dieter. **Creating Procedural Windowbuilding Blocks using the Generative Fact Labeling Method**. *Proceedings of the ISPRS International Workshop 3D-ARCH*, 5:235–242, 2013.

[106] Robert F. Tobler, Stefan Maierhofer, and Alexander Wilkie. **A Multiresolution Mesh Generation Approach for Procedural Definition of Complex Geometry**. *Proceedings of the Shape Modeling International*, 6:35 – 44, 2002.

[107] Robert F. Tobler, Stefan Maierhofer, and Alexander Wilkie. **Mesh-Based Parametrized L-Systems and Generalized Subdivision for Generating Complex Geometry**. *International Journal of Shape Modeling*, 8:173–191, 2002.

[108] Torsten Ullrich. **Reconstructive Geometry**. *PhD-Thesis, Technische Universität Graz, Austria*, 1:1–322, 2011.

[109] Torsten Ullrich and Dieter W. Fellner. **Generative Object Definition and Semantic Recognition**. *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, 4:1–8, 2011.

[110] Torsten Ullrich, Ulrich Krispel, and Dieter W. Fellner. **Compilation of Procedural Models**. *Proceeding of the 13th International Conference on 3D Web Technology*, 13:75–81, 2008.

[111] Torsten Ullrich, Christoph Schinko, and Dieter W. Fellner. **Procedural Modeling in Theory and Practice**. *Poster Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, 18:5–8, 2010.

[112] Torsten Ullrich, Christoph Schinko, Thomas Schiffer, and Dieter W. Fellner. **Procedural Descriptions for Analyzing Digitized Artifacts**. *Applied Geomatics*, 5(3):185–192, 2013.

[113] Torsten Ullrich, Volker Settgast, and René Berndt. **Semantic Enrichment for 3D Documents: Techniques and Open Problems**. *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, 14:374–384, 2010.

[114] Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. **Semantic Fitting and Reconstruction**. *Journal on Computing and Cultural Heritage*, 1(2):1201–1220, 2008.

[115] Torsten Ullrich, Nelson Silva, Eva Eggeling, and Dieter W. Fellner. **Generative Modeling and Numerical Optimization for Energy Efficient Buildings**. *Proceedings of IEEE / OCG Energy Informatics*, 2:123–128, 2013.

[116] Ilkay Ulusoy and Christopher W. Bishop. **Generative versus Discriminative Methods for Object Recognition**. *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:258 – 265, 2005.

[117] Luc Van Gool, Andelo Martinovic, and Markus Mathias. **Towards Semantic City Models**. *Proceedings of Photogrammetric Week*, 1:217–232, 2013.

[118] Carlos A. Vanegas, Daniel G. Aliaga, Peter Wonka, Pascal Müller, Paul Waddell, and Benjamin Watson. **Modelling the Appearance and Behaviour of Urban Spaces**. *Computer Graphics Forum*, 29:25–42, 2010.

[119] Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. **Inverse Design of Urban Procedural Models**. *ACM Transactions on Graphics*, 31:168:1–, 2012.

[120] Gerrit Voß, Johannes Behr, Dirk Reiners, and Marcus Roth. **A multi-thread safe foundation for scene graphs and its extension to clusters**. *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, 4:33–37, 2002.

[121] Somlak Wannarumon. **An Aesthetics Driven Approach to Jewelry Design**. *Computer-Aided Design and Applications*, 7:489–503, 2010.

[122] Benjamin Watson and Peter Wonka. **Procedural Methods for Urban Modeling**. *IEEE Computer Graphics and Applications*, 28(3):16–17, 2008.

[123] Eric Weisstein. **MathWorld – A Wolfram Web Resource**. Wolfram Research, 2009.

[124] Emily Whiting, John Ochsendorf, and Fredo Durand. **Procedural Modeling of Structurally-Sound Masonry Buildings**. *ACM Transactions on Graphics*, 28:112:1–9, 2009.

[125] Peter Wonka, Michael Wimmer, Francois Sillion, and William Ribarsky. **Instant Architecture**. *International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2003*, 22(3):669 – 677, 2003.

[126] Liu Yong, Zhang Mingmin, Jiang Yunliang, and Zhao Haiying. **Improving procedural modeling with semantics in digital architectural heritage**. *Computers & Graphics*, 36:178–184, 2012.

[127] Lap-Fai Yu, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley Osher. **Make it Home: Automatic Optimization of Furniture Arrangement**. *ACM Transactions on Graphics*, 30:86:1–11, 2011.

[128] René Zmugg, Wolfgang Thaller, Ulrich Krispel, Johannes Edelsbrunner, Sven Havemann, and Dieter W. Fellner. **Deformation-Aware Split Grammars for Architectural Models**. *Proceedings of the International Conference on Cyberworlds*, 11:4–11, 2013.

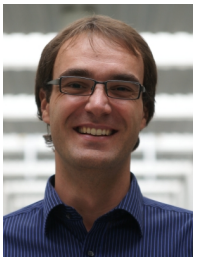
[129] René Zmugg, Wolfgang Thaller, Ulrich Krispel, Johannes Edelsbrunner, Sven Havemann, and Dieter W. Fellner. **Procedural Architecture using Deformation-Aware Split Grammars**. *The Visual Computer*, 12:1–11, 2013.

About the Authors

Ulrich Krispel

Ulrich Krispel received his Master's degree in Telematics in 2008 from Graz University of Technology, Austria. He has been a research assistant at the Institute of Computer Graphics and Knowledge Visualization (CGV), Graz University of Technology, and at Interactive Graphics Systems Group (GRIS), TU Darmstadt. Currently, he is a research assistant at Fraunhofer Austria Research GmbH. His fields of interest cover geometry processing for procedural descriptions and (inverse) procedural modeling with a focus on shape grammars.

Christoph Schinko



Christoph Schinko, born in 1982, studied Telematics at Graz University of Technology. He received his Master's degree in June 2009 and worked as a researcher in the FIT-IT project MetaDesigner at the Institute of Computer Graphics and Knowledge Visualization (CGV) at Graz University of Technology. His research focuses on generative modeling, rapid prototyping and 3D Web technologies. Since March 2013 he works for Fraunhofer Austria Research GmbH, Visual Computing.

Torsten Ullrich (presenter)

Dr. Torsten Ullrich studied mathematics with a focus on computer science at the University of Karlsruhe (TH) and received his doctorate in 2011 with his work "Reconstructive Geometry" on the subject of reverse engineering at the Graz University of Technology. Since 2005 he has been involved in the establishment of the newly formed "Fraunhofer Austria Research GmbH". He coordinates research projects in the field of "Visual Decision Support", "Virtual Engineering" and "Digital Society" and is Deputy Head of the business area *Visual Computing* in Graz. His research in the field of generative modeling languages and geometrical optimization has received several international awards.

Contact information: torsten.ullrich@fraunhofer.at

Acknowledgement

The authors gratefully acknowledge the generous support by the Austrian Research Promotion Agency (FFG) for the research project "Procedural Fitting Service (ProFitS)", #840254.