International Journal on Interactive Design and Manufacturing (IJIDeM)
https://doi.org/10.1007/s12008-022-00872-7

TECHNICAL PAPER

A theoretical reflection on smart shape modeling

Francisco J. Serón[1] · Ángel Zaldívar[1] · Alfonso Blesa[2] · Gabriela Celani[3] · Juan Magallón[1]

Received: 3 September 2020 / Accepted: 11 March 2022

Abstract
This paper presents, as far as the authors are aware, a complete and extended new taxonomy of shape specification modeling techniques and a characterization of shape design systems, all based on the relationship of users' knowledge to the modeling system they use to generate shapes. In-depth knowledge of this relationship is not usually revealed in the regular university training courses such as bachelor's, master's and continuing education. For this reason, we believe that it is necessary to modify the learning process, offering a more global vision of all the currently existing techniques and extending training in those related to algorithmic modeling techniques. We consider the latter to be the most powerful current techniques for modeling complex shapes that cannot be modeled with the usual techniques known to date. Therefore, the most complete training should include everything from the usual geometry to textual programming. This would take us a step further along the way to more powerful design environments. The proposed taxonomy could serve as a guideline to help improve the learning process of students and designers in a complex environment with increasingly powerful requirements and tools. The term "smart" is widely used nowadays, e.g. smart phones, smart cars, smart homes, smart cities... and similar terms such as "smart shape modeling". Nowadays, the term smart is applied from a marketing point of view, whenever an innovation is used to solve a complex problem. This is the case for what is currently called smart shape modeling. However, in the future; this concept should mean a much better design environment than today. The smart future requires better trained and skilled engineers, architects, designers or technical students. This means that they must be prepared to be able to contribute to the creation of new knowledge, to the use of innovations to solve complex problems of form, and to the extraction of the relevant pieces of intelligence from the growing volume of knowledge and technologies accessible today. Our taxonomy is presented from the point of view of methods that are possibly furthest away from what is considered today as "intelligent shape modeling" to the limit of what is achievable today and which the authors call "Generic Shape Algorithm". Finally, we discuss the characteristics that a shape modeling system must have to be truly "intelligent": it must be "proactive" in applying innovative ideas to achieve a solution to a complex problem.

Keywords Shape modelling · Smart systems · Taxonomy · Procedural modeling · Geometrical modeling · Proactive geometrical modeling · Proactive procedural modeling

1 Introduction

Smart Geometry is based on using computational and parametric software tools and it is focused on using of the computer as an "intelligent design" aid in architecture,

✉ Alfonso Blesa
ablesa@unizar.es

Francisco J. Serón
seron@unizar.es
http://webdiis.unizar.es/~seron/

Ángel Zaldívar
zaldivar@unizar.es

Gabriela Celani
celani@unicamp.edu

Juan Magallón
magallon@unizar.es

[1] Department of Informatics, Universidad de Zaragoza Escuela de Ingeniería y Arquitectura (EINA), Campus Río Ebro, Zaragoza, Spain

[2] Department of Electronics, Universidad de Zaragoza Escuela Universitaria Politécnica, Teruel, Spain

[3] Laboratory of Automation and Prototyping for Architecture and Construction, School of Civil Engineering, Architecture and Urban Design, University of Campinas, Campinas, Brazil

Published online: 05 April 2022

design, engineering and construction. An example of gathering of the global community of innovators and pioneers in these fields are the sg20XX conferences [1] that "*promotes the emergence of this new paradigm in which digital designers and craftsmen are able to intelligently exploit the combination of digital and physical media to take projects from design right through to production*". Smart shape modelling is a part of this global knowledge.

In the near future, or perhaps even now, theses smart environments require better educated and trained engineers, architects, designers, or technical students. They must be aware of this challenge and, therefore, they must be prepared to extract the relevant pieces of intelligence from the ever-growing volume of knowledge and technologies accessible today. In doing so, they will be able to use innovations to solve/design complex shapes and also contribute to the creation of new knowledge. As in other fields where the evolution of knowledge is faster and faster, we must take into account how our students learn it; that is, the strategies that they can use for ordering and prioritizing. It is also important to consider how to efficiently generate and incorporate new methods or tools that allow solving the same problems, or even addressing new ones. In this context, we must highlight the following aspects:

1. As theoretical models and tools increase in complexity and power, shape design can be approached from different options, so the student or designer who must know how to classify and categorize the knowledge of shape modeling systems to choose the best one for each project; i.e., users with different awareness or ability levels should use the right tool or methodology, adapted to his/her skills.
   Thus, it is necessary to offer the student a complete shape modeling taxonomy from the outset that allows them to know from the point of view of a global context how to select the correct models and tools to work with shapes. This global context knowing is even one of the most important objective to achieve during the learning process when the engineering or design student faces to, for the first time, a design project with a set complex set of solutions, usually with limited time to deal with them in detail: without right references for the student's level it can generate frustration or lack of motivation, affecting his/her performance.
   In addition, a good taxonomy must allow the inclusion of new options that they can appear in time to advance in the design of smart shapes. For all this, we propose a taxonomy, taking in account the topic "smart" to classify the different modeling categories from the least to the smartest. Furthermore, we introduce as a proposal a new category associated to smart shape modelling (algorithmic generic shape): The genuine coding task to build a textual algorithm. This proposal is compared with all rest on the taxonomic tree.
2. Shape designers' learning process: Depending on their training, a user (shape designer, engineer, architect, student, etc.) can be characterized in terms of the final knowledge and his/her relationship with the system modeling used to generate a shape. In order to make the student clear what level he has reached: This relationship can be described as follows: (a) his/her awareness about the underlying mathematical and algorithm descriptions and their properties and, (b) the user's ability to construct new mathematical descriptions or/and new mental algorithms. We must distinguish between knowing a shape modeling tool and how it is used, and understanding why that tool is so and why it reveals certain properties. The user reaches the most complete knowledge when he acquires the ability to expand it.

1.1 Shape specification

A geometric shape is the manner in which a curve, surface or volume occupies space. The specification (or definition or method to describe an object) and representation of a shape (a data structure in which the object can be instantiated and manipulated by the computer) are significantly different elements in geometric design. Although a shape's specification is often treated as a representation, it is, however, useful to differentiate one from the other.

In the context of computer-aided design, computer graphics, geometry processing and shape description, digitally creating a shape is called modeling. A general 3D shape modeling system should be capable of specifying and representing any physically constructible or imaginable meaningful shape, so that it can be analyzed and queried easily. This is not straightforward or possible without some level of approximation/representation change with current technology, since there is no single way to express the possible shape of an object. Even if we restrict ourselves to man-made shapes, specifying and representing problems are still very difficult. Usual modeling systems "solve" this problem by restricting the shapes they deal with to a narrow domain.

If we adopt a more rigorous view of modeling, we can differentiate between three separate levels:

- Physical objects: using models, our aim is to discuss some real or imagined, yet plausible, things in our three-dimensional real world. Unfortunately, we cannot perceive a real-world object in its full complexity, and if it were possible, we could not represent all aspects of it with a computer.
- Mathematical or algorithmic objects: to have any hope of modeling objects with a computer, we must adopt a suitable idealization of the real three–dimensional physical

objects (shape specification). As we shall see later, such a characterization can be achieved using basic concepts from mathematical and algorithmic theories.
- Representations: the final step of the modeling activity is to assign the representation suitable for computer modeling. These representation scheme properties are:
  - Expressive power and precision: which objects can be modeled? And how accurately can complicated objects be modeled?
  - Validity: Are all admissible representations valid?
  - Unambiguity and uniqueness: Do every valid representations model exactly one shape? Do some shapes have more than one valid representation?
  - Conciseness: How large (in terms of computer storage) do representations of solids become? (This property often contradicts the precision of the representation).
  - Time: can the representation be manipulated in a timely manner?
  - Closure of operations: How general are the manipulations that can be supported?

In any case, it is helpful to divide a shape computer-based modeling system into four parts:

- A shape specification, which defines what a shape is,
- a computer representation to instantiate the shape,
- a user interface that allows shapes to be specified,
- a set of tools for manipulating shape representation, to determine what can be done with shapes.

Although we believe that all four are important, specification is the part of the modeling system that mostly determines the chance of achieving the intended shape. Therefore, without neglecting the others, in this article we will focus primarily on specification.

The structure of the paper is as follows: in this Section the authors explain the reasons why designers need to understand the concepts that underlie the taxonomy chosen to explore the world of shape design and know how to choose the most appropriate one. In addition, the parts that make up a digital shape modeling system are described and guide the paper to the shape specification part. Section 2 presents the taxonomy for shape specification and a comprehensive and detailed description of approaches to shape specification. It also shows the scope offered by each category in relation to the adjective smart. Section 3 presents a suggestion of a possible work methodology to train designers. In Section 4 the authors have selected two of their own examples, which are analyzed from the point of view of the knowledge the shape designer must have. Finally, in Sect. 5, we present the conclusions and future work, and we discuss what features a shape modeling system should have to be called "smart".

2 Shape modeling and its associated specification taxonomy

Taxonomy, in a wider, more general sense, may refer to a classification of things or concepts, and the principles underlying this classification. In this paper, the concepts are approaches to generating a shape, and the principles are the specification modeling techniques. In our taxonomy, the broad specification classes of a geometric shape are: when the modeling system is not based on an algorithmic description, principle 1, it will be termed "geometric modeling" in this paper, and when the modeling system is based on an algorithm description, principle 2, it will be termed "procedural modeling". Figure 1 shows the taxonomy proposed and it is used as guide for this section. The first level is related with main taxonomy branchs: geometrical and procedural modeling global classification.

Geometric modeling (GM) is considered a branch of applied mathematics and computational geometry that studies methods for the mathematical description of shapes as surfaces or solids. These descriptions may be based on mathematical functions used to compute the coordinates of points on or within the shape, or they may be based on discrete samples of points on or within the shape.

Procedural modeling (PM) (The object is defined implicitly by an algorithm, i.e., the shapes emerge from the algorithmic description of a construction process that computes points on or within it). In mathematical and computer sciences, an algorithm is a self-contained step-by-step set of operations to be performed to solve a particular problem. In the context of this paper, we believe there is a clear distinction between the ability to construct an algorithm capable of generating a shape, and the ability to implement an algorithm on a computer. The first ability describes an algorithm, ignoring the implementation details; the second requires a more machine formal description. In this paper we refer to the first capacity.

Both GM and PM are produced using a computer and for computer-based applications, but the user's cognitive approach to each differs greatly. Usually, end users of a modeling system (architects or engineers) have a rich interface that they interact with by selecting 2D/3D actions that correspond to specific internal computer codes involved in different achievable options. Normal users do not have a background in algorithm techniques and do not have in-depth knowledge of the geometric/mathematical foundations involved in the achievable options. Therefore, user interface (interoperability) remains the main issue between common architects or engineers and commercial modeling systems.

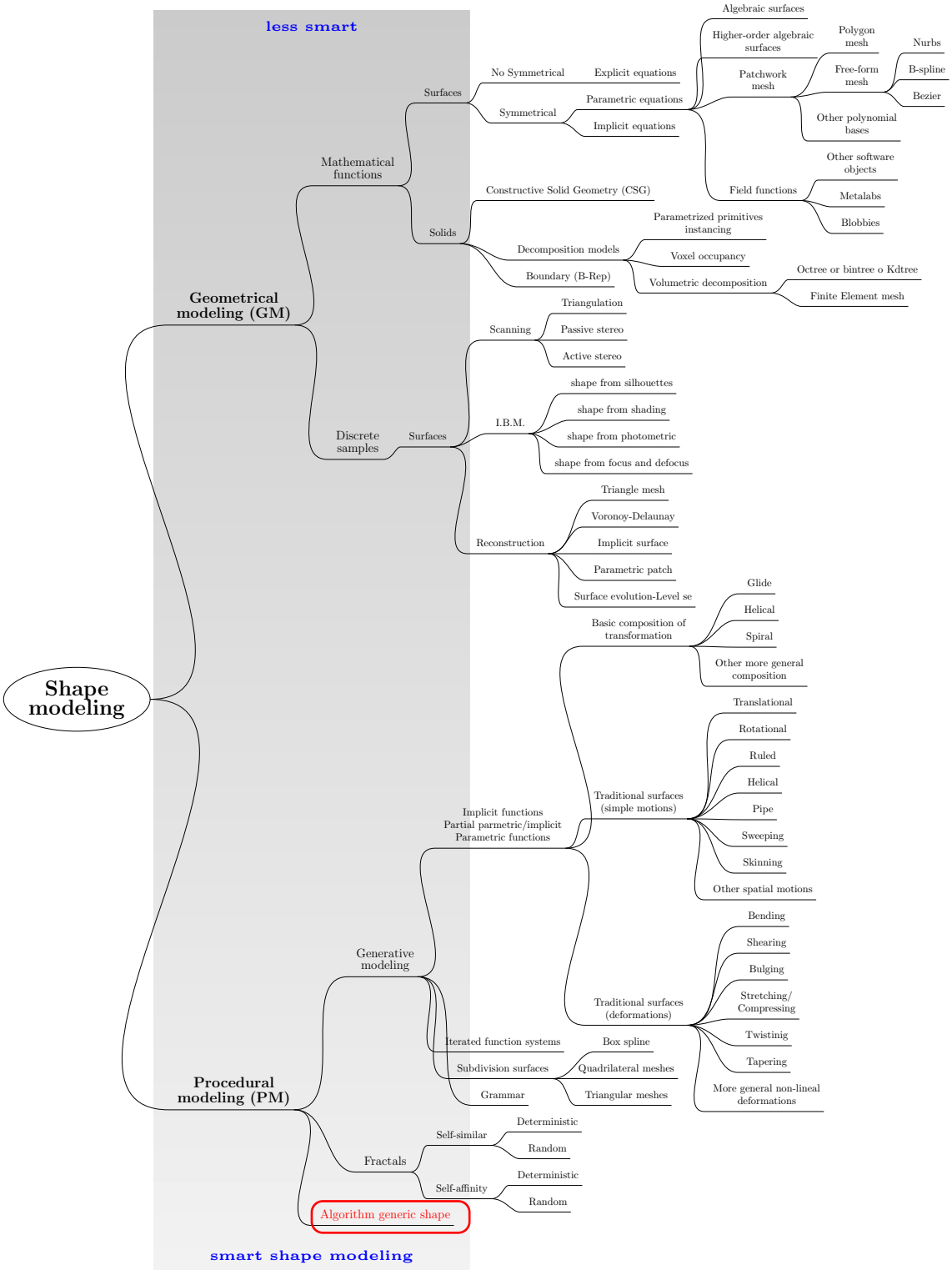


Fig. 1 Shape modeling taxonomy

In this kind of approximation, users often ignore the essential differences between geometry and computer science in defining/describing a shape.

Slowly but increasingly, the ability to design an algorithm to describe a shape is favored by the emergence of systems that offer both textual and visual scripting capabilities. A script language is a programming language that supports scripts, programs written for a special run-time environment that can interpret (rather than compile) and automate the execution of tasks that could alternatively be executed one-by-one by a human operator. The spectrum of scripting languages ranges from very small and highly domain-specific languages to general-purpose programming languages. In these kinds of approximation users need to have a background in algorithm techniques and understand the geometric/mathematical foundations involved in achievable shapes [2].

2.1 Geometric modeling (GM) shape specification based on mathematical functions

This section surveys several techniques for shape specification based on geometric modeling using mathematical functions. Application areas of these techniques are central to computer-aided-design and manufacturing (CAD/CAM) and widely used in many applied technical fields, including shipbuilding, aircraft and automotive industries, as well as architectural and industrial design and computer graphics.

Identifying relationships between the coordinates (x, y and z) of a point on or within a geometric object is helpful to further develop the taxonomy. This relationship may differ depending on how the coordinates are treated [3]:

- Non-symmetrical approach
  - Explicit equations: $z = g(x, y)$
- Symmetrical approach
  - Parametric equations, where the coordinates are treated as functional values: $x = f_x(u, v)$, $y = f_y(u, v)$, $z = f_z(u, v)$
  - Implicit equations, where the coordinates are treated as functional arguments: $F(x, y, z) = 0, F(x, y, z) = c$ (isosurfaces)

The functions g, $f_{x,y,z}$ or F may contain any mathematical expressions. An expression that is only polynomial (the expression may be rational) is called algebraic; an expression that is non-polynomial (trigonometric, exponential, logarithmic and hyperbolic) is called transcendental.

Traditionally, computer graphics has favored the polynomial parametric over the implicit equation because the former is simpler to draw, render, tessellate, subdivide, bound and perform operations requiring detailed knowledge of the surface, and it is more convenient for certain geometric differential operations.

Implicit surface functions naturally describe whether a point is inside, outside, or on a surface, represent blends of volumes and provide an alternative to the often difficult process of filleting and rounding parametric surfaces.

Since parametric and implicit equations have complementary advantages with respect to certain geometric operations, converting from one to the other can be useful. Conversion from the parametric to the implicit form is known as implicitization and it is not always tractable and is often computationally demanding; conversion from the implicit to the parametric form is known as parameterization and it is not always possible. The implicitization of parametric curves and surfaces, parameterization of implicits, and techniques used to circumvent conversions between implicit and parametric representations are discussed in [4].

2.1.1 Geometric modeling (GM) surface shape specification based on algebraic functions

As mentioned, algebraic surfaces [5], may be defined parametrically or implicitly. There are some principal means to define algebraic objects that are more complex than low-order surfaces such as planes, quadric surfaces, and tori:

- Algebraic surfaces.
- Higher-order algebraic surfaces. Designing surfaces with this type of shape modeling is very difficult, except in some situations (i.e., super quadric).
- Patchwork mesh is a technique that allows users to define a shape obtained by joining fragments of other simpler shapes. Patch specification can be *implicit* or *parametric* [3]. These fragments are usually named patches consisting of, piecewise, low-order parametric or implicit algebraic surfaces. With this kind of approximation, the patched may need to be carefully joined to their neighbors to maintain continuity along patch boundaries. Patches are desirable because they provide a compact and highly continuous surface specification and an easy representation based on the specification itself or surface tessellation. Important issues in the free-form polynomial patch modeling of shapes depending of the type of piecewise rational/non–rational bases used [6].
  - Polygon mesh, based on triangle faces or quadrilateral linear faces; they are the simplest algebraic primitives. Used as specifications of polygonal shapes or as a representation of tessellating other types of surfaces.

  - Free-form mesh based on interpolation or approximation splines (uniform or non-uniform) bases (tensor product schemes): Bezier, B-spline and Nurbs.
  - Other Polynomial bases.
- Previous specifications make it difficult or inefficient to represent some computer-based 3D geometric models for smooth and deformable objects. A possible and popular approach has been the use of implicit algebraic surfaces called field functions in 3D that can generate isosurfaces through the field. Three commonly used techniques, each differentiated by their field functions, are [7]: Blobs, Metaballs and Soft objects.

2.1.2 Geometric modeling (GM) volume shape specification based on mathematical functions

The division of space into inside and outside is the basis for the form of geometric modeling called solid modeling. A solid model is defined as a surface and it's interior. The theoretical underpinnings of solid modeling are found in point-set topology: a solid is defined as a bounded closed regular set of points. Representations achieving this may be divided into three large classes as follows [8]:

- Constructive solid geometry (CSG) representing a point set as a combination of primitive point sets. Each of the primitives is represented as an instance of a primitive solid type. Constructive models include more general construction operations than mere gluing (union), namely rigid motions and set Boolean operations (union, intersection, and difference).
- Decomposition models representing a point set as a collection of simple objects from a fixed collection of primitive objects, combined with a single gluing operation.
  - Parameterized primitive instancing.
  - Voxel occupancy (also called spatial enumeration).
  - Volumetric decomposition:
    - Octree or bintree or Kd-tree
    - Finite element mesh
- Boundary (B–Rep) representing a point set in relation to the boundary. The boundary of a three-dimensional solid point set is a two dimensional surface usually represented as patches.

2.1.3 Geometric modeling (GM) surface shape specification based on discrete samples

When designing products with free form shapes, the shape information can be acquired as a set of data points and then the surface patches have to be designed from the same. This modeling is named Point-based shapes. Point primitives have experienced a major renaissance in recent years. Modern three-dimensional digital photography and 3D scanning systems acquire both geometry and appearance of complex shapes, generating point clouds of a 3D shape of real-world objects. An overview of acquisition techniques [9]:

- Scanning system: triangulation-based 3D scanners, Passive stereo and Active stereo.
- Image-based modeling (IBM). This is a set of methods relying on a set of 2D images of a shape that are used [10].
  - Shape from silhouettes.
  - Shape from shading and photometric stereo.
  - Shape from focus and defocus.
- Converting a point-cloud shape representation of an object into a more explicit representation is known as surface reconstruction. Usual representations are [9]:
  - Triangle mesh.
  - Voronoi and Delaunay triangulations.
  - Implicit surface methods.
  - Parametric patches.
  - Surface evolution or level set method. This is a type of geometric modeling that adds dynamics to implicit surfaces. They are characterized by several useful properties allowing shape specification that involves topology or time-varying shape objects or problems in which sharp corners and cusps are present.

2.2 Procedural modeling (PM) shape specification based on algorithms

This section surveys several shape specification techniques based on procedural modeling using algorithms. Application areas of this technique are central to computer graphics, industrial design, architectural shapes, city generation, terrain modeling, and so on.

Algorithms are often accompanied by a list of parameters. The term parameter (sometimes called formal parameter) is an intrinsic property of the algorithm included in its definition.

The procedural modeling design focuses on creating a shape based on a selected algorithmic procedure and the set of parameters that characterize it. Moreover, the specific derived shape depends on parameters that are explicit in the information structure and must be evaluated to obtain a specific solid shape. We must be aware that a procedural model comprises all the specific shapes derivable from the representation. Furthermore, the supported design paradigm allows the shape designer to define entire families of shapes, not just specific instances. This added flexibility can be exploited in many

ways, and forms an important advance in shape modeling and its applications. Such possibilities are certainly not suitable for all users; they require a fairly sophisticated mathematical background, the ability to design and modify algorithms and substantial training investment.

The principal ideas of procedural modeling (PM) are:

- The information necessary to generate a given shape is recoverable from the shape's algorithmic definition. The procedural modeling captures the evolution of the shape from the building blocks, rather than merely specifying the end result of a creative process. The standard geometric model approach has no memory, since the shape's generation is ignored rather than stored or regenerated for later recovery.
- PM offers the ability to encapsulate a design and opens up new approaches and possibilities.
- PM replaces complicated formulas with simple procedures. Thus, it is more in tune with modern computer science than classical mathematics.
- PM can generate a larger class of shapes.
- The complexity of a procedure, when measured in terms of the length of the shortest computer program that can generate it, is very small.

When a designer writes a geometric algorithm to solve a shape design and manufacture problem, the algorithm is both a description of the problem and the solution. Geometric algorithmic thinking means:

- Taking on an interpretative role to understand the result of the algorithm.
- Understanding how to modify the algorithm to explore new options.
- Speculating on further design potential.

An algorithmic design enables the designer to define relationships between elements or groups of elements, and to assign values of complex expressions to organize and control those definitions.

As with any design tool, there are positive and negative aspects. The primary advantage is that, once the relationship has been established, the system may run autonomously within its parameters and explore novel solutions that may not be apparent to the designer. The main disadvantage of the parametric-design process is that it is very time-consuming, particularly for inexperienced designers. In the algorithmic approach, designers spend time and effort "designing the design". Understanding this way of thinking requires considering algorithm-related knowledge, in its broadest sense, in the design process [11].

Consequently, the algorithm is about rationalization, reasoning, logic, deduction, induction, extrapolation, exploration, and estimation of processes. In its manifold implications, it involves problem solving, mental structures, cognition, simulation, and rule-based intelligence, to name a few. It is well known that to find the algorithm that solves a problem, you first have to know how to solve the problem. Therefore, to paraphrase what tradition says was engraved on the door of Plato's Academy, the school he founded in Athens, to produce an algorithmic design, "let no one ignorant of geometry enter". We believe this sentence highlights the need for geometric knowledge to work in this field of design.

The most common types of geometry, and, therefore, necessary for algorithmic design, are: Euclidean geometry, computational geometry, differential geometry, differential topology, algebraic geometry, and fractal geometry. However, designers without much formal training in geometry can rely on the computer to help with geometry teaching, since it allows geometry to be manipulated dynamically and interactively. The proposed taxonomy can be seen in Fig. 1.

2.2.1 Generative modeling

In this article, the representation of a shape obtained by a generative model is considered a shape described by continuous arbitrary transformation of an initial shape called the generator. Generators and transformations may be embedded in a space of any dimension [12]. Typically, a generative model is formed by transformations of a lower-dimensional generator. As an example, consider a generative surface consisting of all points generated by transformation acting on each point of a 3D curve.

- Allows a generator be represented by the parametric function $F(t) : R^l \rightarrow R^m$.
- Allows a continuous set of transformation be represented as a parametric transformation $T(p; q) : R^m \times R^k \rightarrow R^n$ where $p \in R^m$ is a point to be transformed and $q \in R^k$ is an additional parameter defining a continuum set of transformations.
- The shape is the parametric function $T(F(t); q) : R^{l+k} \rightarrow R^n$.

A generative model allows arbitrary transformation of generators. Usually the base for representing generative models (generators and transformations) is functions of any number of parameters. In this approach, shapes are represented as multidimensional, continuous, piecewise differentiable parametric functions. Ultimately, the shape is a set of points that are the image of a mapping.

The taxonomy proposed split Generative modeling on these branchs: Parametric/Implicit functions [12,13], Iterated function systems (IFSs were conceived in their present form by John E. Hutchinson in 1981 [14] and was popularized by

Michael Barnsley's book [15]), Subdivision surfaces [16,17] and Grammars [18].

2.2.2 Generative modeling. Parametric/implicit functions

The use of parametric generators and transformations yields a closure property because transformations of a generator can be expressed as a simple composition of parametric functions, resulting in another parametric function. In fact, the use of parametric generators and transformations blurs the distinction between generator and transformation, both are parametric functions:

- Parametric functions are not the only basis for representing generative models that can be chosen. Generators and transformations can also be determined by implicit functions. An implicit function describes a shape as the set of points that solves a system of equations rather than as the set of points that is the image of a mapping. Although an implicit representation is valuable in many circumstances, rendering general implicit shapes is a costly computation.
- Partly parametric and partly implicit generative shape:
  - Allows a generator to be represented by an implicit function $F(x) : R^n \rightarrow R^m$
  - Allows a continuous transformation set to be represented as a parametric transformation $T(p; q) : R^n \times R^k \rightarrow R^n$ where $p \in R^n$ is a point to be transformed and $q \in R^k$ is an additional parameter defining a continuum set of transformations.
  - The shape can be expressed as the set $T(x; q)|F(x) = 0, q \in R^k$
- Totally implicit generative shape
  - Allows a generator to be represented by an implicit function $F(x) : R^n \rightarrow R^m$.
  - Allows a continuous transformation set to be represented implicitly as $T(p; q) : R^n \times R^k \rightarrow R^k$ can be seen as a transformation of p by determining a q that solves $T(p; q) = 0$, where $p \in R^n$ is a point to be transformed and $q \in R^k$ is an additional parameter defining a continuum set of transformations.
  - The shape can be expressed as the set $T(x; q)|F(x) = 0, q \in R^k$

The transformation set utilized may be linear (affine transformations), fraction of linear functions (projective transformation) or non-linear (deformation transformation).

In general, generative models are easy to control and edit, since they encourage building high-dimensional shapes from low-dimensional components. Generative models are natural for specifying many man-made shapes and digital manufacturing processes. These digital fabrication approaches are known by the names: sectioning, interlocking, contouring, tessellation, and folding.

The usefulness of the generative modeling approach is not limited to mimicking the manufacturing processes. Totally synthetic generators and transformations can be used. Generative modeling can be seen as a device for human modelers to conceptualize shapes. Very complex shapes can be built with a series of transformations that act on simpler generator shapes.

Generative models are not limited to rigid 3D shapes. They can represent shapes deforming in time, shapes that are functions of manufacturing variables, or shapes composed of non-uniform materials. Such shapes can be represented using generators and transformations that are functions of desired parameters. Generative models allow meta-shapes through parameterized generators and transformations [19].

Usually, as a user interface, a set of symbolic operators on such functions are used to build complicated shapes by simple composition of symbolic operators. All operators in the generative modeling approach are recursive, i.e., their results can be used as inputs for other operators. This recursive property together with the closure property gives the designer the use of any reasonable combination of operations to specify shapes.

An overview of the more usual techniques relationship with Generative Modeling is [13]:

- Basic spatial affine and projective transformations are: translation, rotation, reflection, scaling (uniform and non-uniform), shear, perspective.
- Basic composition of transformation: glide reflection (reflection and a special translation), helical (rotation and translation), spiral (rotation, translation and scaling), other more general composition.
- Traditional surface classes obtained with simple motions are: translational, rotational, ruled, helical, pipe, sweeping, skinning, other more general spatial motions.
- Traditional surface classes obtained with deformations are: tapering, twisting, stretching/Compressing, bulging, shearing, bending, more general non-linear deformations.

2.2.3 The deterministic fractal shapes based on iterated function systems (IFS) as a special case of generative modeling

Conceptually, building a deterministic fractal based on IFSs requires two ingredients: a space of geometric shapes q_k and a transformation T that maps a coarse shape q_{k-1} to a fine shape p_k. Given an initial shape q_0, a fractal method defines an infinite sequence of shapes by iterating S according to $q_k = T(q_{k-1})$. If the transformation T is chosen appropri-

ately, the limit of q_k as $k \to \infty$ is a well-defined shape q_∞ satisfying the fixed-point property $q_\infty = T(q_\infty)$.

In the standard fractal algorithm, we begin with a compact set C_0 of points and iterate over a collection of (m) contractive transformations W_m each with a respective contractivity factor (r) to generate a sequence of compact sets $C_{n+1} = W_m(Cn)$ that converge in the limit to a fractal shape $C_{m,\infty} = \lim_{x\to\infty} C_{m,n}$; the final shape is $q_\infty = \cup_{(i=1)}^{n} W_m(C_i)$.

The principal algorithms used are termed [15]: deterministic and random iteration.

Each W_m is associated with a probability $p_i > 0$, where $\sum_{i=1}^{n} p_i = 1$. p_i indicates the probability of occurrence of the event $C_n = W_m(C_{n-1})$. Despite the use of probabilities, the final shape is always deterministic [20,21].

The goal of fractal procedures is to construct extraordinary shapes like the patterns that can be found in nature [22], unconventional forms as the obtained in [23], from conventional origins.

One of the simplest methods for constructing fractals involves iterating a set of affine transformations. However, conceptually, it is possible to apply any of the previously presented transformations, provided that the selected transformation turns into contractive property. The problem of intentionality, which consists in finding the necessary transformations from a certain form, is a problem that has not yet been resolved. For example, in the case of IFS, the feasibility is found in the Collage theorem [24]. However, the exhaustive search of an IFS to approximate a given image is beyond the capabilities of current computing systems. An IFS like that of the fern consists of 24 parameters, so finding their respective values by "brute force" would require a computation time that is impossible to quantify.

The question could be, so what are they useful for, a possible answer could be, so that the designer can find reasons or causes for inspiration.

2.2.4 Subdivision surfaces as another special case of generative modeling

Subdivision surfaces as another special case of generative modeling Subdivision algorithms are similar to fractal IFS procedures. In the standard fractal algorithm, we begin with a compact set C_0 and iterate over a collection of contractive transformations W_m to generate, in each case, a sequence of compact sets $C_{n+1} = W_m(C_n)$ that converge in the limit to a fractal shape $q_\infty = \cup_{(i=1)}^{n} W_m(C_i)$.

In subdivision procedures, we start with a set P_0 (usually either a control polygon or a control polyhedron, or a quadrilateral or triangular mesh) and recursively apply a set of rules S to generate a sequence of sets $P_{n+1} = S(P_n)$ of the same general type as P_0 that converge in the limit to a smooth curve or surface $P_\infty = \lim_{n\to\infty} P_n$.

Although classical subdivision algorithms are essentially fractal (IFS) procedures, the goals of subdivision algorithms and fractal (IFS) procedures are, nevertheless, fundamentally different. The goal of fractal (IFS) procedures is to construct extraordinary shapes, unconventional forms from conventional origins; the goal of subdivision algorithms is to construct smooth shapes, differentiable functions from discrete data.

The subdivision algorithms are interesting for three reasons:

- They are easy to understand and simple to implement.
- They can generate a large class of smooth functions, not just polynomials and piecewise polynomials.
- Subdivision algorithms on polyhedral meshes can produce shapes with arbitrary topology, unlike tensor product schemes that can only generate surfaces topologically equivalent to a rectangle, or by identifying edges to a cylinder or a torus.

There are three distinct paradigms for subdividing surfaces [16]:

- Box spline (a generalization of uniform tensor product B-spline surfaces).
- Quadrilateral meshes, or arbitrary quadrilateral meshes.
- Triangular meshes or arbitrary triangular meshes.

2.2.5 Grammars as another special case of generative modeling

A shape grammar consists of shape rules and a generation engine that selects and processes rules [25]. The foundation of shape grammars has been defined in [18]. A shape rule defines how an existing (part of a) shape can be transformed. A shape rule consists of two parts separated by an arrow pointing from left to right. The left part of the arrow is termed the left-hand side (LHS); it depicts a condition in terms of a shape and a marker. The right part of the arrow is termed the right-hand side (RHS); it depicts how the LHS shape should be transformed and where the marker is positioned. The marker helps to locate and orient the new shape.

A shape grammar minimally consists of three shape elements: an initial shape, at least one transformation rule, and a termination rule. The initial shape is necessary to start the shape generation process. The termination rule is necessary to stop the shape generation process. The simplest way to stop the process is by a shape rule that removes the marker.

Parametric shape grammars [26] are an extension of shape grammars. The new shape in the RHS of the shape rule is defined by parameters so that it can consider more of the context of the already existing shapes. This typically affects

internal proportions of the new shape so that a greater variety of forms can be created.

Despite their popularity and applicability in academic circles [27], shape grammars have not found widespread use in generic computer-aided design applications, in part due to difficulties in implementing the identification of sub-shapes for applying rules, and of new emerging shapes that are not coincident with the geometric representation of the existing ones.

2.2.6 Fractal shapes in general

Since Euclid, mathematicians have developed the theory of smooth curves and surfaces. These shapes may have a very complicated structure globally, but in small neighborhoods they are just straight lines or planes. The discipline that deals with these objects is differential geometry. Fractals feature just the opposite of smoothness. While the smooth objects do not yield any more detail on smaller scales, a fractal possesses infinite detail at all scales no matter how small they are.

How are fractals different from the usual Euclidean shapes?

- Whereas Euclidean shapes have one or at most a few, characteristic sizes or length scales (i.e., the size of a cube), fractals possess no characteristic sizes.
- Euclidean geometry provides a concise accurate description of man-made objects but is inappropriate for natural shapes. Fractals, however, provide an excellent description of many natural shapes.
- Finally, whereas Euclidean shapes are usually described by mathematical formulas, fractals, in general, are the result of a construction procedure or algorithm that is often recursive and ideally suited to computers. The complexity of a fractal, when measured in terms of the length of the shortest computer program than can generate it, is very small.

Fractal shapes are said to be self-similar and independent of scale or scaling; in other words, self-similar shapes repeat (statistically or exactly) under magnification with uniform scaling.

When the magnification is with non-uniform scaling, the shapes are (statistically or exactly) invariant under transformations that scale coordinates by different amounts, and the fractal is known as self-affinity.

The distinction between similarity and affinity is important. By way of summary, a self-similar object comprises N copies of itself (with possible translations and rotations), each of which is scaled down by the ratio r in all coordinates from the whole. On the other hand, a self-affinity object is the union of N distinct (non-overlapping) subsets, each of which is scaled down by different ratios in each coordinate from the whole. Fractals can be Self-similar and Self-affine (see Fig. 1).

Fractals come in two major variations termed [15]:

- Deterministic. The procedure requires the use of a particular rule that is then repeated over and over in a usually recursive scheme. Application of the same procedure with the same input data, always generates the same shape.
- Random. The procedure is similar to a deterministic procedure but there is an additional element of randomness. The same procedure, with the same initial data but with different random data, generates shapes of the same family but different from each other.

2.2.7 Generic shapes = algorithms + data structures

Many objects and environments contain repetitive or self-similar structures that can be modeled easily using some types of procedural modeling, generative and fractal techniques. This kind of procedural modeling as a mature technology in contemporary commercial systems has pervaded several architectural and engineering domains. After the advent of this paradigm, designers are now endowed with tools to construct new entities. In accordance with its philosophy, any design is viewed as a geometric object resulting from specific algorithmic computations. The type of procedural modeling we are referring to is also known in the literature as "smart geometry" [28].

These approaches have been used to generate many complex shapes, but each method is mainly limited to a specific class of model or requires considerable user input or guidance. If the new objective is to shift from an instance design to a generic one, the procedural modeling paradigm is broadened by using the "bricks" that allow geometric algorithms to be built. The algorithm may employ conventional mathematical functions, arithmetic operators, vector and matrix operators, integration, differentiation, constraint solutions, constrained minimization, numerical solvers, widely used numerical techniques, data tables, or a rule-based or graph-based approach. It may also employ randomness or stochastic methods, and conditionals and iterations as well. In algorithmic science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently.

In general, a procedural definition is not expressible in closed form. Consequently, geometers cannot understand the surface analytically. In this case, geometric properties can be deduced only through the function's numerical evaluation. Designers, however, may find considerable flexibility in procedural methods.

The constraint-based algorithmic design is important in CAD/CAM applications. It allows the algorithm used to design the shape to be a function of a given set of parame-

ters and constraints on them. A constraint specifies a relation on or between entities in a model that must be maintained. The following classes of constraints arise naturally in shape design:

- Geometric relationship as well as metric dimensional constraints.
- Equational constraints that express relations between dimensional parameters and/or technological variables, for instance (mechanical properties).
- Semantic constraints that define validity conditions on a shape.
- Topological relations between entities in a model, such as incidence or connectivity.

Although it would be impossible to describe all the ways algorithms are used to solve problems in a few lines, there are four strategies regularly employed by computer scientists at the core of this modern style of problem solving:

- Careful problem definition.
- Logical reasoning.
- Decomposition techniques, also called divide and conquer. strategies. The idea is to approach a problem by separating it into its constituent sub problems, and then solve each sub problem individually.
- Abstraction using anything that allows us to concentrate on important characteristics while deemphasizing less important, perhaps distracting, details.

Describing algorithms requires a notation to express a sequence of steps to be performed. The most common option is pseudocode, and the control structure is the mechanism for specifying the proper order the five steps must be performed in: sequential, selection, repetition, control abstraction and concurrency.

There are many different programming paradigms used in the field of algorithmic modeling. From the highest to the lowest capacity to generate shapes they are:

- Imperative and oriented object: Using classical programming paradigms [29].
- Dataflow based: A shape description can be represented by a direct graph of the data flowing between operations [30].
- Ruled-based systems: These systems provide a declarative description of the construction behavior of a model by a set of rules. An example is L-systems [25].

3 A suggestion for a possible methodology

The basic guideline of all good training is that our brain behaves in the following way: what is heard is forgotten. What is seen is remembered. What is done is learned. As we are now in the century of information and technology: What you hear, see and do at the same time is what really matters, what you learn and what you enjoy. And therefore, it is the best that can be taught. In addition, a reasonably complete training capable of preparing a designer must offer 1. Basic knowledge of the fundamentals on which they are based. 2. Seeing not only academic examples but learn real examples of application. 3. Learning how to use the tools available at the time to meet the challenges of design.

It is evident that there are different basic formations of form designers, such as industrial designers, engineers of different types, architects, etc. who are usually dedicated to a specific set of form modeling. This fact requires a different training as the different conditions of each type of student have to be taken into account. We all know that there is a lot to teach and that the time for training is finite, and that the designer must look for the moment and find opportunities. But as William Ward (1921–1994) said: "*The mediocre teacher says. The good teacher explains. Senior teacher demonstrates. The great teacher inspires*". Finally, and in relation to existing professionals, there is no way to access new knowledge without retraining.

When asking what to teach, in what order shape modelling methods could be presented, and what references might be appropriate to cite, this document provides, as a script, an overview of the current possibilities and indicates the minimum knowledge required.

If one looks at the taxonomy presented in Fig. 1, the methods are presented from least to most intelligent, and the last line highlighted, which is called Algorithm generic shape. Any teacher specializing in form design can teach any of the above techniques, leaving it up to his or her discretion (and the time available for training) to go deeper into the aspects mentioned in the previous paragraph, referred to in (1–3). However, it is very rare to advance in the teaching of the *Algorithm generic shape*. For this reason, we are going to propose a reflection on the minimum knowledge necessary to do so.

The procedural modeling cited are: Generative modeling, fractals and generic algorithmic shapes. Although all shapes are based on the concept of algorithm, the algorithm based generative modeling considers that a shape is described by the arbitrary continuous transformation of an initial shape called generator. In the case of fractals, they are based on procedures that use a particular rule that is then repeated over and over again in a deterministic way or with an additional element of randomness. Therefore, both techniques have a limitation in terms of the set of all the shapes that can be obtained with

them. This is not the case for the technique we have generically called Algorithm generic shape . To the question of "what are the limits of this technique?" The answer lies in theoretical computer science, and in the closely related fields of algorithm analysis and theory of computation. In our case, the question is "what kinds of shape problems can, in principle, be solved algorithmically?". Without going into further depth in this paper, it is clear that all of the above techniques shown in Fig. 1 are obviously algorithmically solvable. But there are shapes that do not meet the necessary requirements either by generative or fractal means.

The academic answer to the academic question "what is to be taught" is to perform the equality:

$$generic\ shapes = algorithms + data\ structures \quad (1)$$

This expression is understood in the context that it is not necessary for the designer to become a computer scientist, but we are stating that he needs to broaden his language to expand his ability to model forms through a slightly deeper knowledge of what algorithms are, how they are expressed and the types of data that are most commonly used.

The original shape and aesthetics that the designer intends to model will depend on the designer's creativity, bearing in mind that, at the moment, creativity cannot be taught, only trained.

Algorithm generic shapes opens the way to future design systems defined more accurately as smart ones. The concept of "smart" (in the sense of intelligent) has many different perspectives. In general, something is said to be intelligent when is proactive in applying innovative ideas to achieve a solution of a complex problem.

In Sect. 4 of this paper we show two examples of what we are saying. The example 1. Is a virtual exhibition stand design based on generative modeling and visual scripting. The example 2 show soap bubbles based on a textual algorithm.

This last example, quite complex to implement, has been chosen because, is not obtainable by currently existing generative modeling tools. The algorithm is general, in the sense that it can generate any resulting soap bubbles shape. The algorithm is very concise and only collects the three experimentally geometric conditions introduced by Plateau [31]. The final result will depend on the initial data.

The two examples show that using general techniques it is possible to generate families in different ways. With generative techniques, such as those shown, the potential for work is very large. And with the same example algorithm, any bubble soup could be modeled. Indeed, the examples seem to support the fact that the proposed methods require ad hoc solutions for each general design problem, which is the case with all the current existing design approaches.

The reason is that, from a theoretical point of view, problem solving is considered the most complex of all intellectual functions. Problem solving has been defined as a higher level cognitive process that requires modulation and control of more routine or fundamental skills. Although problem solving has existed since the beginning of human evolution, the nature of human problem solving processes and methods have led to the fact that, for the time being, no successful attempts have been made to derive a comprehensive theory of general problem solving. Therefore, to solve concrete shape modeling problems, we have to settle for approximations to bound sets of shapes that have some way of expressing themselves in common.

In the Sect. 5 we reflect about the future of shape modeling taxonomy. But at this point, we just reflect on algorithms that have not yet been used in a general way to model shapes: neuronal networks, deep learning, convolutional neuronal networks, recurrent neuronal networks, genetic algorithms, cellular automata, ...

4 Examples

4.1 Example 1. A virtual exhibition stand design based on generative modeling and visual scripting

The idea of this example is to show a suggestive application. Therefore, we decided to design a virtual exhibition stand based on algorithmic modeling and generative strategies which took into account fabrication capabilities that are offered by a fab lab (fabrication laboratory) [32]. Fab labs are small-scale workshops providing a (personal) digital fabrication service. They are generally equipped with a variety of flexible computer-controlled tools for several length scales and materials, with the aim of making almost anything. Their main feature is that they are strongly linked to society. Developing and disseminating this exploration of algorithmic design and digital fabrication were, therefore, central objectives of this example [33,34].

Digital fabrication techniques generally fit into four main categories: Cutting, Subtraction, Addition and Formation, and the modeling techniques implemented to generate the designs are: contouring, folding, forming, sectioning and tiling [35].

- Contouring: much of the material used to produce shapes is processed in or from a sheet format. Contouring changes this physical materiality by using an incremental subtractive technique to provide intricately patterned three-dimensional features through a series of contours. Specifically, the use of CNC milling can quickly produce non-standard or repetitive elements. We have used three

different algorithms in the design of the stand modules. The first is based on the concept of fields and attractor points, the second on contours and the last on Voronoi mesh.

- Folding: a much more familiar process of developing two-dimensional surfaces into three-dimensional forms is folding. By folding, sheet material rigidity can increase substantially. While the material operation may seem relatively finite, digital technologies enable the calculation and setting out of complex fold patterns, furnishing the practitioner with a greater spectrum of design options.
- Tiling: this process, also referred to as tessellating, involves developing figures or shapes that when assembled together form a coherent plane without any gaps or overlaps. One of the many advantages of digital design and fabrication methods is that they can effectively overcome the previous investment of time and also provide ways in which patterns may be generated and optimized to gain maximum impact both visually and materially especially to reduce waste.

We used tiling and folding techniques for the stand to design a complex surface. We designed a collection of individual pieces of cut and folded sheet to act as tesserae. These tiles are joined by bolts.

- Forming and Interlocking: Forming method is an effective and relatively economical way of making a significant number of equal components that are usually created using CNC-milled mold but occasionally uses additive manufacturing o vacuum-forming techniques. In the case of this stand we have used the Forming technique to design a sculpture formed by the union (Interlocking) of equal regular pentagonal elements, in turn generating a sculpture formed by dodecagons joined together.
- Sectioning: this technique is a method of profiling components in relation to surface geometry. By taking a series of sectional cuts through a digital model, it offers a quick and effective way of gathering the necessary data to inform a CAD/CAM process. The stand structure was designed using the Sectioning technique. Areas where the sections are either separated or combined together were differentiated during construction. In each case, we considered the manufacturing, assembly, formal development and aesthetic appearance of the stand itself.

Figure 2 shows the full stand and Fig. 3 explains the digital fabrication techniques used.

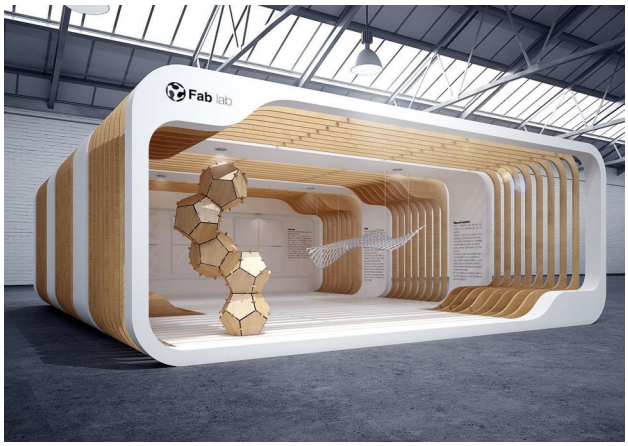


Fig. 2 The stand

Fig. 3 Digital fabrication techniques used: *1* sectioning, *2* interlocking, *3* contouring, *4* folding

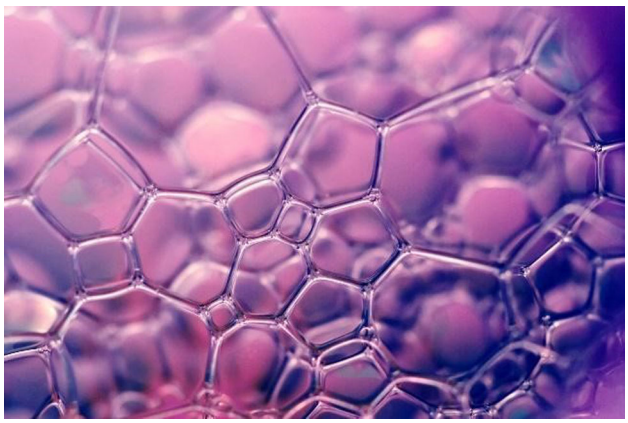


Fig. 4 Soap bubbles

4.2 Example 2. 2D and 3D shapes of soap bubbles based on a textual algorithm

The idea of this example is to design something that would show what we found practically impossible to model accurately applying any of the techniques presented so far, unless an algorithm is used. Therefore, we decided to design a general algorithmic modeling of soap bubbles, inspired by the external walls of Beijing National Aquatic Center.

For centuries, soap bubbles (see Fig. 4) have captivated biologists (Thompson with coalescence patterns [36], physicists (Plateau [31], Boys [37], and Isenberg [38] with soap films), mathematicians (Euler with *area minima* [39]) and architects (Frei Otto and his experiments with real bubbles [40], the consortium of Beijing's Box of Bubbles [41], and so on).

Bubbles seek an arrangement that minimizes surface area and energy. After Plateau's work on this, the topic was studied mathematically by Sophie Germain, and it is currently an area of great development in molecular biology and chemistry. The fact that bubbles must meet in the way they do was only proved in the 1970s.

This principle can be seen not only in soap bubbles but more generally in nature. A dragonfly's wings are a good example, as are bee hives.

The theoretical work was introduced by Plateau, who used wire frames to explain soap bubble interfaces. Plateau [31] concluded experimentally that soap bubble interfaces always meet three geometric conditions (see Fig. 5):

- Only three interfaces can meet at a point, creating Plateau borders. The amount a border can curve inwards or outwards is determined by the difference in pressure on either side (Young–Laplace equation).
- The tangential angles between the Plateau borders is $120°(2\pi/3)$.
- Four Plateau borders, each formed by the intersection of three surfaces, are joined at vertices creating an angle equal to $109°28'16''(arcs[-1/3])$, called the tetrahedral angle or Maraldi angle. When three or more bubbles meet, they arrange themselves so that no more than three bubbles share a wall, and no more than four share an intersection. At these intersections, each bubble meets the shared wall at a 120° angle and meets every other bubble with an angle of 109.5°.

These elegant rules can be used to explain the interactions among clustered soap bubbles. From a geometric perspective, these interactions can be described quantitatively using three equations (A, B, and C represent centers of the bubbles and r_A, r_B and r_C are their respective radii):

$$\begin{aligned} \frac{1}{r_B} &= \frac{1}{r_A} + \frac{1}{r_C} \\ |AB|^2 &= r_A^2 + r_B^2 - 2 \cdot r_A \cdot r_B \cdot cos(\pi/3) \\ |AC|^2 &= r_A^2 + r_C^2 - 2 \cdot r_A \cdot r_C \cdot cos(2\pi/3) \end{aligned} \tag{2}$$

Equation (2) can be derived from the Young-Laplace equation, whereby excess pressure, surface tension (which is constant for all soap bubbles at an interface) and the principal radii of curvature (which in soap bubbles are equal) relate to each other. The physical phenomenon indicates that pressure is inversely proportional to the radius of curvature. This means that large bubbles contain low excess pressure, while small bubbles contain high excess pressure. Coalescing bubbles are surrounded by three different pressures: inside bubble A, inside bubble B and inside bubble C. The excess pressure between bubbles A and C must be equal to the excess pressures between the two other regions, bubbles B and C and bubbles A and B.

Two cases are considered, one with equal-sized bubbles and one with different-sized bubbles. When equal-sized bubbles meet, no differential is produced, so the Plateau border will be flat. In this case the radius of curvature will be infinite. When unequal-sized bubbles meet, the smaller bubble, which has higher pressure, will push into the larger bubble. The boundary between the bubbles created by the radius of curvature r_C will be curved, with the concave side toward the larger bubble.

4.2.1 2D soap bubble film algorithm

The geometric model that describes the structure of soap films in foams is based on Plateau's laws. The 2D model can be described from a set of simple geometrical routines, which are all detailed in the basic literature of plane geometry. The original geometric functions we used are:

- `CircleCircleIntersection(C1,C2)`: given two circles C1, C2, the function returns the two intersection points between them (if the entities do not intersect, the returned value is null).
- `CircleLineIntersection(C,L)`: given circle C and line L, the function returns the two intersection points between them (if the entities do not intersect, the returned value is null).
- `ParameterValueOfPointInSegment(p,p0p1)`: given point p and the segment defined by points p0 and p1, the function returns the value of the independent variable (scalar) when point p is evaluated in the parametric equation of segment p0p1.
- Point inside/outside circle. The test return if a point is inside or outside a circumference.
- The basic equation of the polar coordinate system.

The first step is to compute the interface arc between two circles. Given two intersected circles C_0 and C_1, the function `ComputeInterfaceArc(C0,C1)` (Algorithm 1 on Fig. 6a) returns the interface arc B' between them. The two input circles are converted to arcs C_0' and C_1', (see Fig. 6b). The procedure works by building an auxiliary circle B with radius according to Eq. (2) and center in one of two intersection points between auxiliary circles A_0 and A_1, which have the same radius and respective centers in the intersection

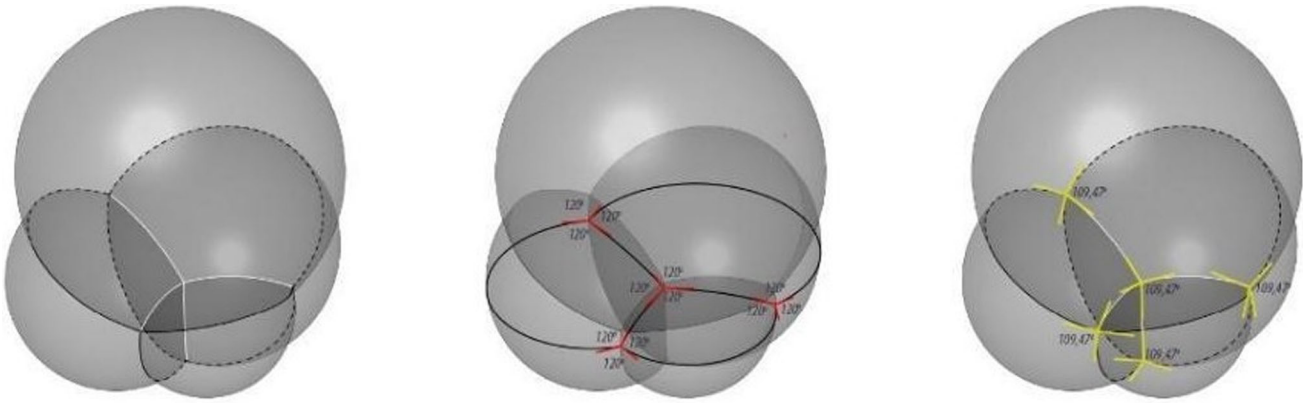
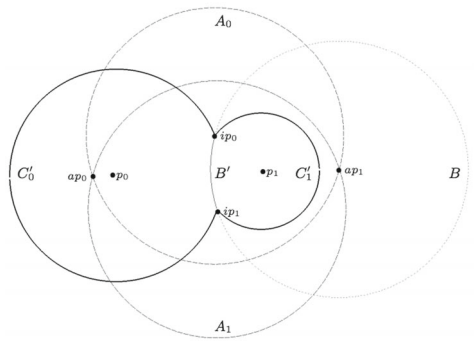


Fig. 5 Tetrahedral angle: **a** all surfaces in a bubble intersection are spherical sectors with equal mean curvature across the entire area. **b** All Plateau surfaces intersect three in 120° angles. In this case, there are 4 internal intersections and 6 external borders, all respecting Plateau Laws. **c** All Plateau Borders intersect three by three in tetrahedral angles. In this case, there are internal vertex and 4 external vertices

Algorithm 1 Compute interface arc between two circles. Given two circles C_0 and C_1 intersected between them, with radios r_0 and r_1, and centers p_0 and p_1 respectively. The function returns the interface arc B' between them. Also, the two input circles are converted to arcs C'_0, C'_1.

function ComputeInterfaceArc(C_0, C_1)
  if $r_0 < r_1$ **then**
    Swap(C_0, C_1) ▷ So, the larger radius circle is C_0
  $(ip_0, ip_1) \leftarrow$ CircleCircleIntersection(C_0, C_1)
  $r_a \leftarrow r_0 * r_1/(r_0 - r_1)$ ▷ Compute auxiliar radius according to ...
  A_0 and A_1 are two auxiliary circles with radius r_a and centers in ip_0 and ip_1 respectively.
  $(ap_0, ap_1) \leftarrow$ CircleCircleIntersection(A_0, A_1)
  v $\leftarrow$ ParameterValueOfPointInSegment($ap_0, \overline{p_0 p_1}$)
  if $v > 1$ **then**
    B is an auxiliary circle with radius r_a and center in ap_0
  else
    B is an auxiliary circle with radius r_a and center in ap_1
  $(tp_0, tp_1) \leftarrow$ CircleLineIntersection($C_0, \overline{p_0 p_1}$)
  Circle C_0 is transformed into arc C'_0 formed by the points ip_0, k_0, ip_1, where k_0 is one of the two point tp_0 and tp_1 with lower parameter value when is evaluated in parametric equation of the segment $\overline{p_0 p_1}$.
  $(tp_0, tp_1) \leftarrow$ CircleLineIntersection($C_1, \overline{p_0 p_1}$)
  Circle C_1 is transformed into arc C'_1 formed by the points ip_0, k_1, ip_1, where k_1 is one of the two point tp_0 and tp_1 with greater parameter value when is evaluated in parametric equation of the segment $\overline{p_0 p_1}$.
  $(tp_0, tp_1) \leftarrow$ CircleLineIntersection($B, \overline{p_0 p_1}$)
  Circle B is transformed into arc B' formed by the points ip_0, k_b, ip_1, where k_b is one of the two point tp_0 and tp_1 with lower parameter value when is evaluated in parametric equation of the segment $\overline{p_0 p_1}$.
  return (B', C'_0, C'_1)

(a) Algorithm 1



(b) Geometrical setup

Fig. 6 Compute interface arc between two circles

points between C_0 and C_1. Here, the circumference arcs are described by three consecutive points over the circumference (initial, middle, final).

The next elementary step is to compute coalescent bubbles among three circles. This procedure is based on similar geometrical principles to the previous step but here an appropriate adjustment of the generated geometrical entities is required. Given three circles C_0, C_1 and C_2 intersected among them.

The function `ComputeCoalescentBubbleAmong3Circles(...)` (Algorithm 2 on Fig. 7a) returns the interface arcs B'_{01}, B'_{02} and B'_{12} between them. The three input circles are also converted to arcs C'_0 , C'_1 and C'_2, see Fig. 7b.

Based on the same constructive principles, the routine that computes the coalescent bubble among four circles (`ComputeCoalescentBubbleAmong4Circles(...)`) can easily be defined. These three geometric routines are enough to generate a bubble field, as we will demonstrate below.

The algorithm to generate a bubble field is based on a smart random generation of circles following a spiral path until the stop condition is reached. By agreement, the proposed procedure follows a counter-clockwise path. Figure 8 shows the first six iterations of the process.

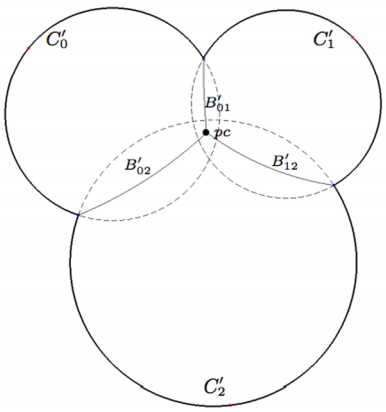
In step 1, a circle only has to be generated with a random radius ($r \in [minRadius, maxRadius]$) and position ($x, y \in [-viewportLimit, viewportLimit]$).

In step 2, the second circle is generated at a random distance d ($d \in [r_0, r_0 + r_1]$) from the center of the existing circle and the random position is computed generating a random angle $\alpha(\alpha \in [0, 2\pi])$ and using the polar coordinate system equations. Once the new circle is defined, the routine

Algorithm 2 Compute coalescent bubble among three circles. Given three circles C_0, C_1 and C_2 intersected among them. The function returns the interface arcs B'_{01}, B'_{02} and B'_{12} between them. Also, the three input circles are converted to arcs C'_0, C'_1 and C'_2.

function ComputeCoalescentBubbleAmong3Circles(C_0, C_1, C_2)
  $(B'_{01}, C'_0, C'_2) \leftarrow$ ComputeInterfaceArc(C_0, C_1)
  $(B'_{02}, C'_0, C'_2) \leftarrow$ ComputeInterfaceArc(C_0, C_2)
  $(B'_{12}, C'_1, C'_2) \leftarrow$ ComputeInterfaceArc(C_1, C_2)
  $(ip_0, ip_1) \leftarrow$ CircleCircleIntersection(B'_{02}, B'_{12})
  $pc \leftarrow$ one of the two point ip_0 and ip_1 that is inside the circle C_2
  The arc B'_{01} is adjusted by substituting the point of the arc that is inside C'_2 by pc.
  The arc B'_{02} is adjusted by substituting the point of the arc that is inside C'_1 by pc.
  The arc B'_{12} is adjusted by substituting the point of the arc that is inside C'_0 by pc.
  The arc C'_2 is adjusted by substituting the points of the arc by one the points resulting from the intersection of C'_2 with C'_0 and C'_2 with C'_1 respectively (The circle-circle intersection returns two resulting points, of which, the point outside the remaining circle is used in each substitution).
  The arc C'_0 is adjusted by substituting the point of the arc that is inside C'_2 by the intersection point between C'_2 and C'_0 used for defining C'_2 in previous step.
  The arc C'_1 is adjusted by substituting the point of the arc that is inside C'_2 by the intersection point between C'_2 and C'_1 used for defining C'_2 in previous step.
  return $(B'_{01}, B'_{02}, B'_{12}, C'_0, C'_1, C'_2)$

(a) Algorithm 2



(b) Geometrical setup

Fig. 7 Compute coalescent bubble among three circles

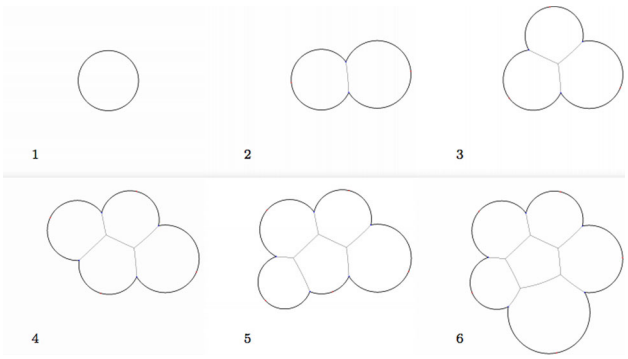


Fig. 8 Initial demonstration steps to generate a bubble field

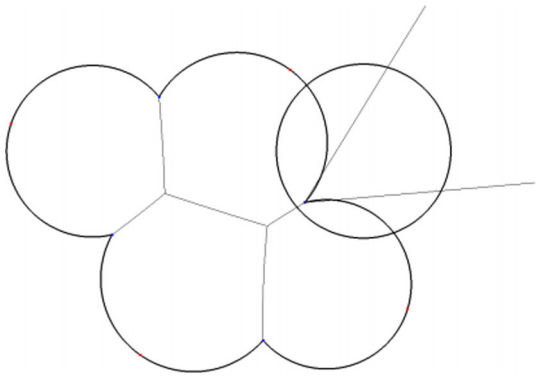


Fig. 9 Reference sector to generate the random circle

Algorithm 3 Generate Bubble Field.

procedure GenerateBubbleField
  lastCircle $\leftarrow$ *NULL*
  repeat
    newCircle $\leftarrow$ GenerateRandomCircleFrom(*lastCircle*)
    if *scene is* ***empty*** **then**
      Add *newCircle* to the scene
    if *scene has* ***one*** *element* **then**
      $C_0 \leftarrow$ *only bubble of the scene*
      $C_1 \leftarrow$ *newCircle*
      $(B'_{01}, C'_0, C'_2) \leftarrow$ ComputeInterfaceArc(C_0, C_1)
      Add B'_{01}, C'_2 and replace C'_0 to the scene
    if *scene has more than* ***one*** *element* **then**
      if *newCircle intersects* ***two*** *circles of the scene* **then**
        $C_0 \leftarrow$ *first circle of the scene that intersects newCircle*
        $C_1 \leftarrow$ *second circle of the scene that intersects newCircle*
        $C_2 \leftarrow$ *newCircle*
        $(B'_{01}, B'_{02}, B'_{12}, C'_0, C'_1, C'_2) \leftarrow$ ComputeCoalescentBubbleAmong3Circles(C_0, C_1, C_2)
        Add B'_{02}, B'_{12}, C'_2 and replace B'_{01}, C'_0, C'_1 to the scene
      if *newCircle intersects* ***three*** *circles of the scene* **then**
        $C_0 \leftarrow$ *first circle of the scene that intersects newCircle*
        $C_1 \leftarrow$ *second circle of the scene that intersects newCircle*
        $C_2 \leftarrow$ *third circle of the scene that intersects newCircle*
        $C_3 \leftarrow$ *newCircle*
        $(B'_{01}, B'_{12}, B'_{03}, B'_{13}, B'_{23}, C'_0, C'_1, C'_2, C'_3) \leftarrow$ ComputeCoalescentBubbleAmong4Circles(C_0, C_1, C_2, C_3)
        Add $B'_{03}, B'_{13}, B'_{23}, C'_3$ and replace $B'_{01}, B'_{12}, C'_0, C'_1, C'_2$ to the scene
    lastCircle $\leftarrow$ *newCircle*
  until (*stop condition*)

Fig. 10 Algorithm 3: generate a bubble field

`ComputeInterfaceArc(...)` is applied to fuse both circles.

Steps 3–5 have in common that the new random circle intersects two existing circles; therefore, the function `ComputeCoalescentBubbleAmong3Circles(...)` is used to obtain the resulting arcs. Each of the new circles is generated counter-clockwise from the intersection point of the last placed circle and its adjoining circle (see Fig. 9). The random parameters are generated as in step 2, but the random angle α remains within the sector defined by the tangent vectors to the radius of two concerned circles in the convergence point (see Fig. 9).

In step 6, the new random circle intersects three existing circles. Therefore, the function `ComputeCoalescentBubbleAmong4Circles(...)` is used to obtain the resulting arcs. Here, the reference point to generate the ran-

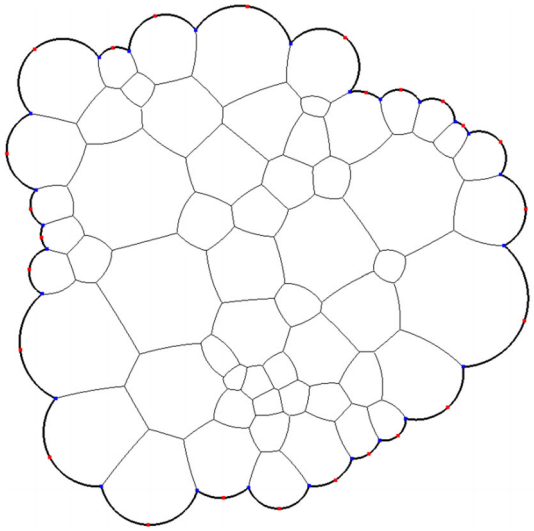


Fig. 11 Example of bubble field

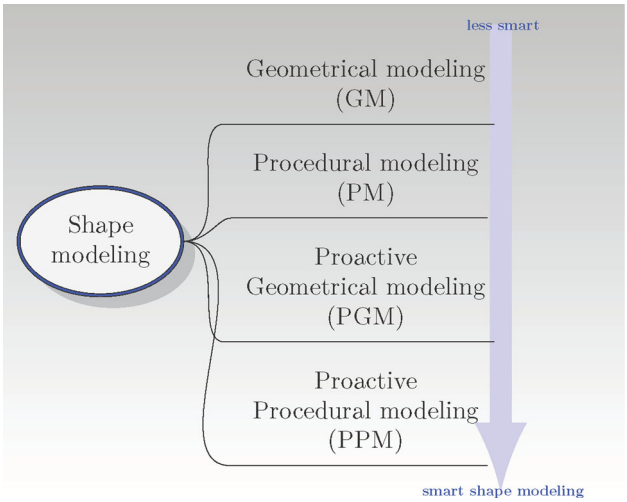


Fig. 12 Future shape modeling taxonomy

dom parameters of the next circle can be the center point of the external arc that will disappear after the fusion operation.

In the successive iterations, the counter-clockwise path is followed with the last convergence point as a reference to fix the random values. If a smart method of generating the next circles is maintained, then some of the last two situations will have to be applied until the stop condition is reached. The ranges and methods for generating the random values in each case can be sophisticated for different visual results.

The pseudocode of this process is shown in Algorithm 3 on Fig. 10. Figure 11 shows an example of a bubble field generated with this method.

5 Conclusions and possible future

Entering general 3D shapes into the computer remains a difficult and time-consuming task. Why do humans find it so hard to specify 3D shapes? What constitutes a good geometric modeling system? What, if anything, is wrong with the current approach to shape modeling? And how can it be improved?

Currently, a designer has multiple strategies to respond to a Shape Design Project. We understand that they must be ordered and presented to the user in a categorized way. Furthermore, this classification must be done during the learning process: It is not just about knowing what tools and procedures are available. It is about understanding in depth its characteristics, establishing a hierarchical organization and choosing the right one for each design.

We have presented a complete and extended taxonomy on shape specification modeling techniques; (see Fig. 1), based on two principles: principle 1, when the modeling system is based on mathematical descriptions; and principle 2, when the modeling system is based on an algorithm description.

This taxonomy has these characteristics: its structure is novel, it establishes "smartness" as a classification criterion and it includes a geometric modeling of forms based on the use of the expression of a textual algorithm (algorithmic generic shape), which opens the way to advance the line of smarter modeling systems.

With this approach, a user (shape designer, students, engineer or architect) can be characterized in terms of the knowledge relationship that the user has established with the system modeling used to generate a shape.

- Awareness (or Opacity) for the user about the underlying mathematical and algorithmic descriptions and their properties.
- The user's ability to construct:
  - A new mathematical description.
  - A new mental algorithm.

As original examples, we present and analyze two shape design procedures. The first example is supposed to be accessible to designers used to the generative modeling dataflow type.

The second example shows a shape design that can only be achieved by building an ad-hoc algorithm implemented by pure programming. Based on our knowledge, this algorithm is the first complete implementation published on this subject. This example illustrates a new category, called "algorithmic generic shape" included in the proposed taxonomy. This category covers algorithmic procedures that include paradigms typical of computer sciences (AI, etc.)

Consequently, it is proposed that designer's usual training (i.e. approach to geometry and algorithm learning and thinking) may need to be modified: The focus of algorithmic thinking goes beyond basic knowledge of computing to treat algorithmic science as an independent body of thought that is an essential part of what it means to be educated today. Thinking algorithmically is uniquely important just as scientific investigation, artistic creativity, or proof theory in mathematics are, and yet computational thinking is a distinct form of thought, separate from these other academic disciplines [42].

The increasingly important role of shape modeling has opened a variety of challenging problems centered on accurate geometric modeling computation. The most general algorithmic modeling technique offers an attractive approach to represent the complex shapes and broad shape variability of general shapes. The continued development and refinement of designer training in algorithmic knowledge should remain an important area of research in the foreseeable future of shape modeling.

Algorithm generic shapes opens the way to future design systems defined more accurately as smart ones. The concept of "smart" (in the sense of intelligent) has many different perspectives. In general, something is said to be intelligent when is proactive in applying innovative ideas to achieve a solution of a complex problem.

The effort to create smart shape modeling techniques will require ideas capable to make use of already accessible technologies as; digital connectivity, access to human knowledge for eventually find intelligent solutions in collaboration with others, artificial intelligence, 3D technologies, big data support, cloud computing... in such a way that can face numerous complex problems of the future shape modeling design will require difficult decisions. Based on them, Fig. 12 shows the desired taxonomy evolution. It is presented only the first level and includes both, the proactive procedural and geometrical modeling. Designers daily work and students learning process will inevitably be affected.

Finally, we need to ask when a shape modeling system should be qualified as intelligent/smart. In our opinion, it should take place when it can be proactive, which means that it will perceive information and retain it as knowledge to be applied adaptively within a given environment and when a small set of design principles can be built with useful ways that can reasonably be called intelligent, to be more precise, when the algorithms underlying shape modeling systems can perceive what to design and act accordingly. In general, such software will make a decision at a given time depending on the complete sequence of insights gained so far.

This may seem like science fiction, but bear in mind artificial intelligence (AI) and cognitive systems (CS), whose fields of study are how to create computers and computer software that are capable of intelligent behavior, or as John McCarthy, who coined the term AI in 1955, said, "the science and engineering of making intelligent machines", or as we like to say, "at least the systems seemed intelligent".

Acknowledgements This work is partially supported by Gobierno de Aragón. Departamento de Ciencia, Universidad y Sociedad del Conocimiento. Grant T33_20D.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. Work partially supported by Gobierno de Aragón. Departamento de Ciencia, Universidad y Sociedad del Conocimiento. Grant T33_20D.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. C.M.U.S. of Architecture. sg2020—smartgeometry (2020). https://www.smartgeometry.org/sg2020
2. Aish, R.: in Inside Smartgeometry: Expanding the Architectural Possibilities of Computational Design, vol. 9781118522, pp. 36–49. Wiley Blackwell (2013). https://doi.org/10.1002/9781118653074.ch2
3. Bloomenthal, J.: The Geometry of Implicit Surfaces, pp. 1–51. Morgan Kaufmann Publishers Inc. (1997)
4. Hoffmann, C.M.: Implicit Curves and Surfaces in CAGD. IEEE Computer Graphics and Applications, vol. 13-1, p. 460 (1993). https://doi.org/10.1109/38.180121
5. Iskovskikh, V.: Encyclopedia of Mathematics, chap. Quadric. Springer (2001)
6. Salomon, D.D.: Curves and Surfaces for Computer Graphics. Springer (2006)
7. Velho, L., Gomes, J., de Figueiredo, L.H.: Implicit Objects in Computer Graphics. Springer, New York (2002). https://doi.org/10.1007/b97350
8. Mäntylä, M.: An Introduction to Solid Modeling. Computer Science Press Inc, USA (1987)
9. Gross, M., Pfister, H.: Point-Based Graphics. Morgan Kaufmann (2007)
10. Quan, L.: Image-Based Modeling. Springer (2010)
11. Scheurer, F.: in Inside Smartgeometry: Expanding the Architectural Possibilities of Computational Design, vol. 9781118522479, pp. 186–195. Wiley Blackwell (2013). https://doi.org/10.1002/9781118653074.ch16

12. Snyder, J.M.: Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design using Interval Analysis. Elsevier Science (2014)
13. Pottmann, H., Asperl, A., Hofer, M., Kilian, A., Bentley, D.: Architectural Geometry. Bentley Institute Press (2007)
14. Hutchinson, J.: Fractals and self-similarity. Indiana Univ. Math. J. **30**, 713–747 (1981)
15. Barnsley, M.F., Devaney, R.L., Mandelbrot, B.B., Peitgen, H.O., Saupe, D., Voss, R.F.: The Science of Fractal Images. Springer, New York (1988). https://doi.org/10.1007/978-1-4612-3784-6
16. Warren, J., Weimer, H.: Subdivision Methods for Geometric Design: A Constructive Approach, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (2001)
17. Subdivision Surface—Wikipedia. https://en.wikipedia.org/wiki/Subdivisionsurface
18. Stiny, G., Gips, J.: Shape grammars and the generative specification of painting and sculpture. In: IFIP Congress (2), vol. 3, pp. 1460–1465. North-Holland (1972)
19. Osher, S., Fedkiw, R.: Level Set Methods and Dynamic Implicit Surfaces, Applied Mathematical Sciences, vol. 153. Springer, New York (2003). https://doi.org/10.1007/b98879
20. Peitgen, H.O., Jürgens, H., Saupe, D.: How randomness creates deterministic shapes, the chaos game. In: Chaos and Fractals, pp. 297–352. Springer (1992)
21. Barnsley, M.F., Devaney, R.L., Mandelbrot, B.B., Peitgen, H.O., Saupe, D., Voss, R.F., Fisher, Y., McGuire, M.: The Science of Fractal Images, vol. 1. Springer (1988)
22. Mandelbrot, B.B., Mandelbrot, B.B.: The Fractal Geometry of Nature, vol. 1. W.H. Freeman, New York (1982)
23. Rani, M., Verma, D.K., Sodhi, J.: 3D Fractal Modeling. 2nd International Conference on Issues and Challenges in Networking, Intelligence and Computing Technologies (ICNICT-2012), vol. 1 (2012)
24. Maurer, H.: New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings, vol. 555. Springer (1991)
25. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. The Virtual Laboratory. Springer, New York (1990). https://doi.org/10.1007/978-1-4613-8476-2
26. Stiny, G.: Introduction to Shape and Shape Grammars. Environ. Plann. B. Plann. Des. **7**(3), 343 (1980)
27. Cagan, J.: Engineering Shape Grammars: Where We Have Been and Where We Are Going, pp. 65–92. Cambridge University Press, USA (2001)
28. Peters, B., Peters, T.: Inside Smartgeometry, vol. 9781118522479. Wiley, Chichester (2013). https://doi.org/10.1002/9781118653074
29. Harel, D., Feldman, Y.: Algorithmics: The Spirit of Computing. Pearson Education Ltd. (2004). https://doi.org/10.1007/978-3-642-27266-0
30. Tedeschi, A.: AAD Algorithms-Aided Designed. Le Penseur (2018)
31. Plateau, J.A.F.: Statique Expérimentale et Théorique des Liquides Soumis aux Seules Forces Moléculaires, vol. 2. Gauthier-Villars (1873)
32. Gershenfeld, N.A.: Fab: The Coming Revolution on your Desktop—From Personal Computers to Personal Fabrication. Basic Books (2007)
33. Martínez, H., Serón, F.D.: Estrategias de modelado y fabricación digital basadas en sistemas paramétricos. Caso de un stand. Tech. Rep., Universidad de Zaragoza, Zaragoza (2013). http://zaguan.unizar.es/record/11965
34. Martinez, H., Bruscato, U., Seron, F.J.: Modeling and manufacturing strategies based on smart geometry. Example of a stand. In: Geometrias'14 Proceedings (2014)
35. Dunn, N.: Digital Fabrication in Architecture. Laurence King (2012)
36. Thompson, D.: On Growth and Form, An abridged. Press Syndicate of the University of Cambridge, UK (1961)
37. Boys, C.V. : Soap Bubbles, their Colours and the Forces which Mold Them, vol. 542. Courier Corporation (1959)
38. Isenberg, C.: The Science of Soap Films and Soap Bubbles. Clevedon (Avon), Tieto (1978)
39. Hildebrandt, S., Tromba, A.J.: Mathematics and Optimal Form. Scientific American Library Series, vol. 13. W.H. Freeman & Company (1984)
40. Frei Otto, experimentando con pompas de jabón. https://www.plataformaarquitectura.cl/cl/763573/video-frei-otto-experimentando-con-pompas-de-jabon
41. "Water Cube" designers delight in box of bubbles. https://www.reuters.com/article/us-olympics-beijing-cube-idUSPEK27470620080131
42. Riley, D.D., Hunt, K.A.: Computational Thinking for the Modern Problem Solver. CRC Press (2014)