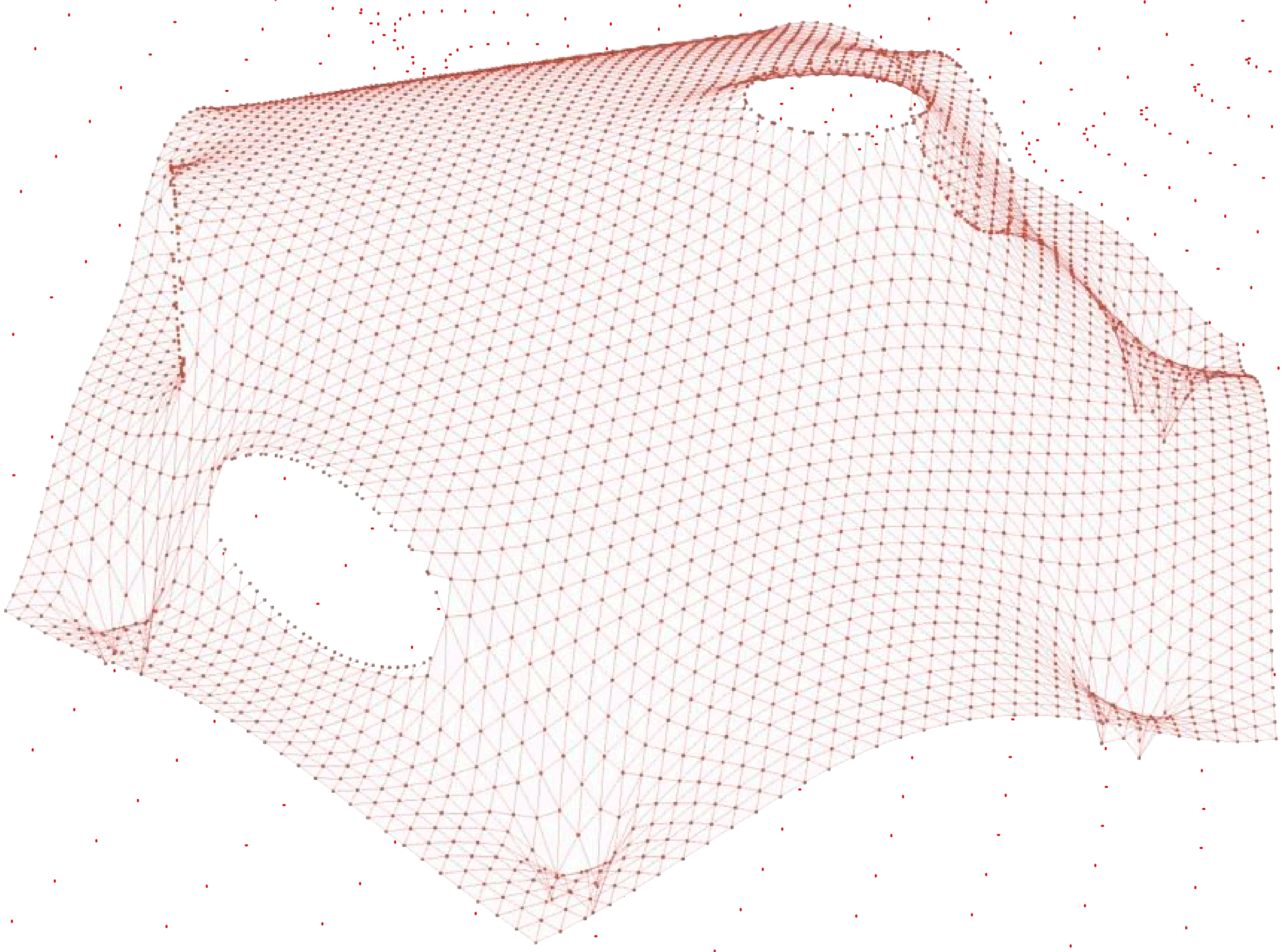
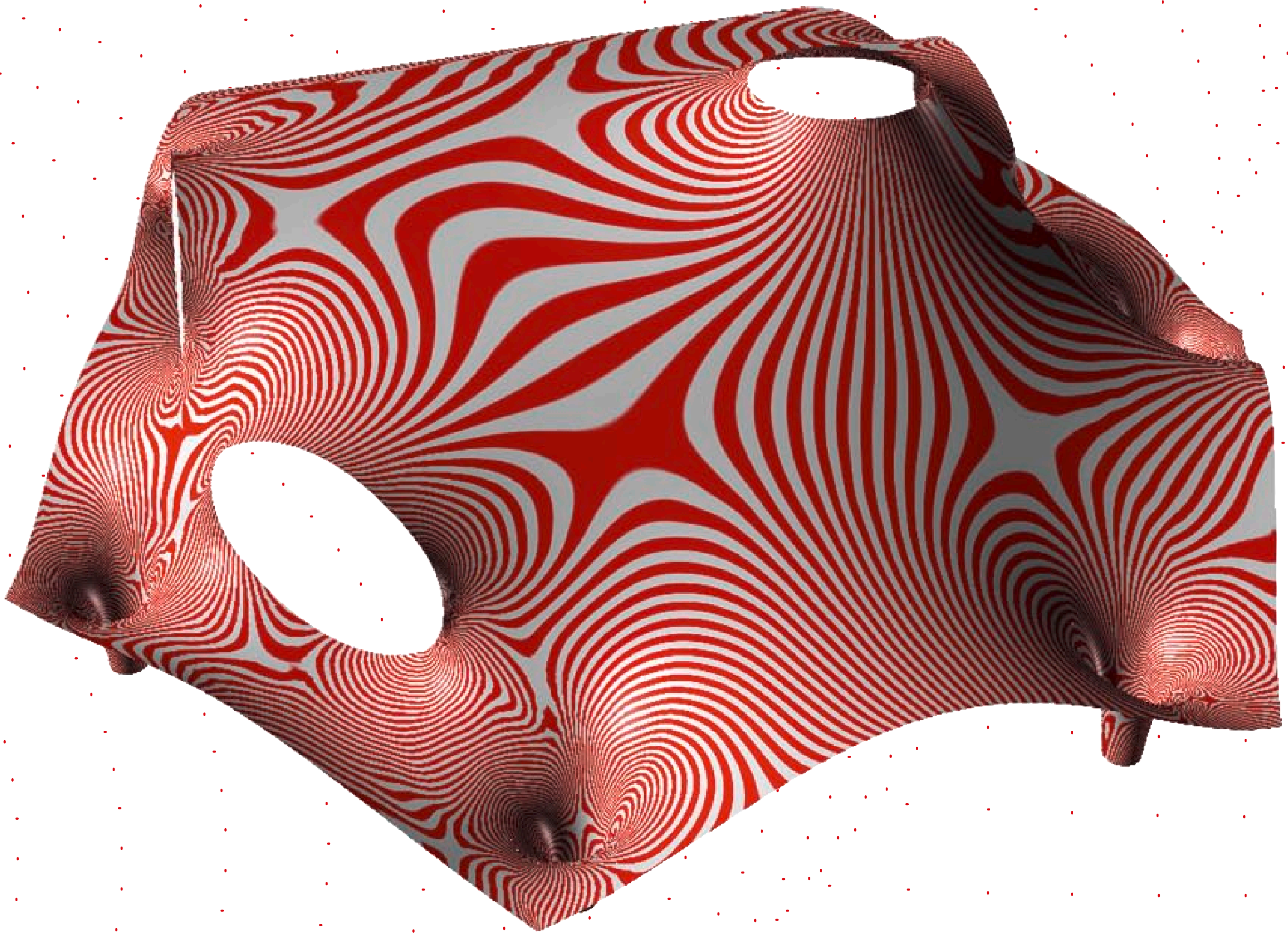
Fabian Scheurer
Hanno Stehling



LOST IN PARAMETER SPACE?



Rather than eradicating the need for mathematics in architectural practice, computation has intensified it. As **Fabian Scheurer and Hanno Stehling** of designtoproduction explain, the uneven flow of a complex design from computer-aided design (CAD) to computer-aided engineering (CAE) and through to computer-aided manufacturing (CAM) necessitates an understanding of abstract mathematical concepts that facilitate communication, precision and an accurate assessment of quality throughout the process.

Figure 1. Shigeru Ban, Centre Pompidou, Metz, France, 2010
The roof surface as triangle mesh (left) and as NURBS surface (right). Only the latter allowed for the fabrication of smoothly curved girders, but its definition took some help from specialists who usually work for the automotive industry.

A few years ago the introduction of ever more powerful computer-aided design (CAD) systems seemed to have almost eradicated mathematics from architectural practice. At least when looking at the curricula of architecture schools one could have the impression that this subject – anyway unloved due to its 'uncreative' formal rigidity – was happily replaced by CAD courses and the belief that somewhere in the background the software would take care of all the calculations. Under this cover, even highly sophisticated mathematical concepts like non-uniform rational b-splines (NURBS) managed to sneak into architecture, understood by very few but happily applied nevertheless.

Recently this seems to have changed again. Architects have suddenly shown an increasing interest in mathematics[1] and its abstract concepts. Surprisingly at first glance, the very same process of digitalisation that once marginalised mathematics in architecture now actuates its re-establishment. On closer inspection, however, this is a consequent progression: it turned out that the complex shapes unleashed by digital design tools did not smoothly flow down the process chain from CAD to computer-aided engineering (CAE) to computer-aided manufacturing (CAM) until automatically materialising inside some digital fabrication machines, and that the implementation of a passably seamless digital workflow requires more than the flick of a button in the designer's CAD software. To understand why this is the case we need to take a look at mathematics and the closely related theory of computation.

First of all, architectural design is a process of communication. It is a long way from the designer's initial idea to the built result, necessitating means to describe a design in ways that give sufficient and unambiguous instructions to the builders.

Traditionally and despite all digitalisation, this is still achieved mainly by 2-D drawings. Mathematically speaking, the projection of a three-dimensional object onto a two-dimensional sheet of paper is a mapping transformation that must be set up carefully in order to deliver not only correct but also meaningful results. Ideally, a projection plane is defined so that most of the object's edges keep their length and inscribed angles, allowing measurements in the drawing to give valid information about the real thing.

Unfortunately, such a projection plane does not now exist for a complex shape with no planar faces. Subsequently, any 2-D plan might transport parts of the topology (how certain features are related to each other) but no reliable metrics any more (how far those features are from each other). This loss of information, in the end, makes it impossible to reconstruct the 3-D object based on a set of 2-D drawings – the traditional language of architecture becomes insufficient.

Luckily, CAD systems have evolved from a stage where they merely were simulating 2-D drawing boards. Nowadays CAD models can – contrary to a sheet of paper – unambiguously store the actual 3-D information of an object. Consistent 2-D plans can then be derived from those 3-D models on request. So complex designs consequently have to be modelled in three dimensions before they are flattened to drawings; the model becomes the core of communication.

Abstraction

A model, by definition, is always an abstraction of reality. Building a model means reducing the infinite complexity of the real world to a level where it can be described with manageable effort. What is obvious in the workshop of a model builder sometimes gets forgotten when almost infinite digital storage space is at hand: a perfect model does not contain as much information as possible, but as little as necessary to describe the properties of an object unambiguously. Any extra bit would be meaningless for the given purpose and only impede comprehensibility. In information theory, this is known as 'Kolmogorov complexity' or 'descriptive complexity': the complexity of an object is defined by the length of the shortest possible description. While modelling starts with gathering data, it is far more important to then throw away everything that turns out to be superficial. This task requires quite some (human!) intelligence, because it involves finding patterns and defining general cases.

This is easily done for planar faces and regular grids: details can be defined once and then multiplied; local changes do not induce re-evaluations of the whole structure. But again, the fun stops as soon as the shapes get curvy.

Abstracting Shape

A straightforward approach to describe a non-planar shape would be to define a large number of points – for instance by laser-scanning a physical model – and connect them by straight lines to form a mesh. Meshes are an easy way to define complex shapes, but they have a severe disadvantage: the planar facets of a mesh can only approximate a curved shape, which is usually acceptable for rendering an image, but certainly is not for digital fabrication as the approximation errors quickly exceed the machine precision (typically some 1/10 millimetre for large-scale fabrication equipment) and are duly and visibly reproduced.

Fortunately, there is a mathematical model for precisely describing curved surfaces. Developed in the 1950s and 1960s, the computational complexity of NURBS meant it took almost 50 years until they started their impressive career in architecture. NURBS allow the precise definition of complex shapes through control points. When used properly, significantly fewer control points are needed for a NURBS surface than vertices for a similar mesh, while at the same time NURBS allow the precise calculation of all in-between points on the

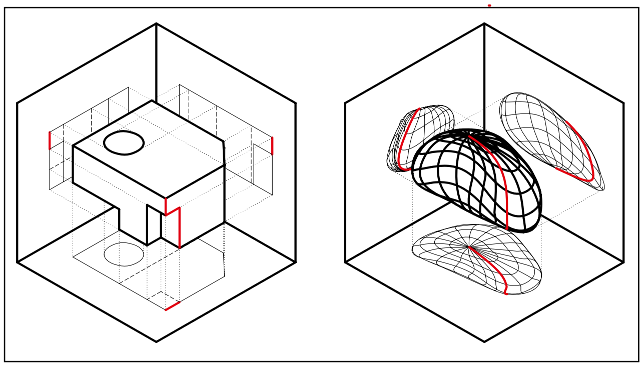


Figure 2. Planar projection
Simple 3-D objects with planar faces (left) can be unambiguously described by a small set of 2-D plans preserving lengths and angles for all edges running parallel to the projection planes. For curved surfaces (right), this approach fails because no projection plane would preserve the metrics.

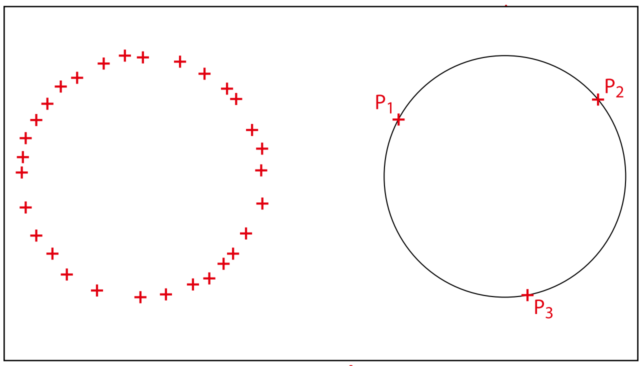


Figure 3. Abstracting a circle
A circle is unambiguously defined by only three points. After discovering the shape behind those 30 points we can throw away 27 of them (90 per cent of the data) and still have the same figure defined in the drawing. Additionally, the geometric definition of a circle lets us now identify the exact location of infinitely many more points than the 30 we started with.

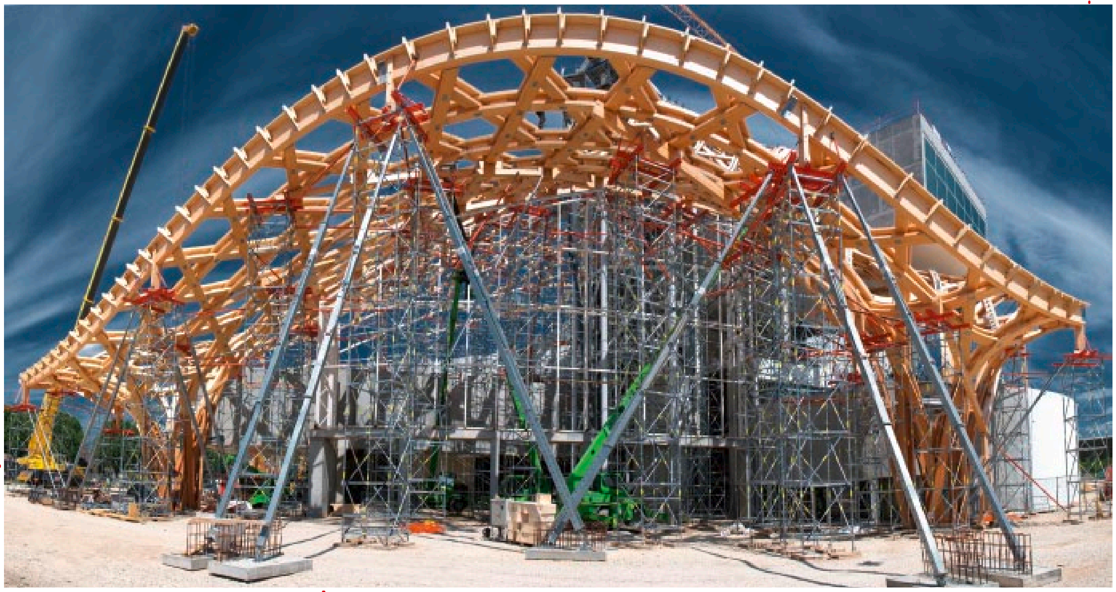


Figure 4. Shigeru Ban, Centre Pompidou, Metz, France, 2010
The roof during erection. The structure is composed of six layers of double-curved girders that were precisely pre-cut on a computer-controlled machine.

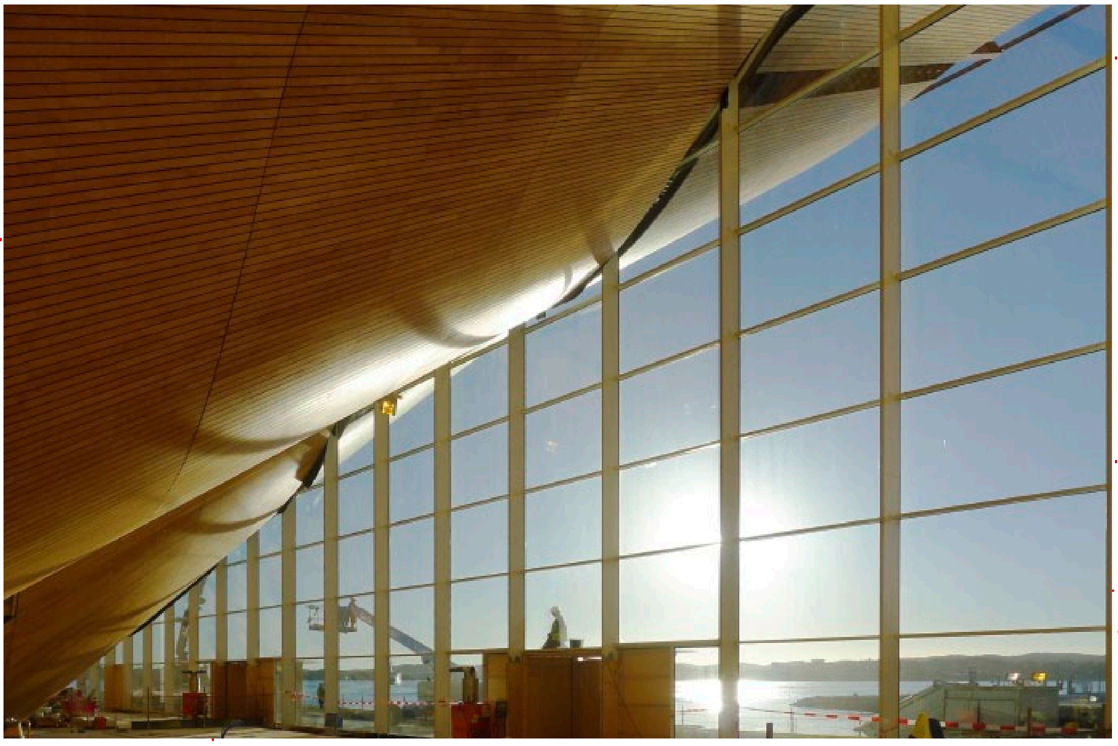


Figure 5. ALA Arkitekter AS, Kilden Performing Arts Center, Kristiansand, Norway, due for completion 2012
The facade towards the waterfront is clad by straight oak boards, only twisted around their longitudinal axis.

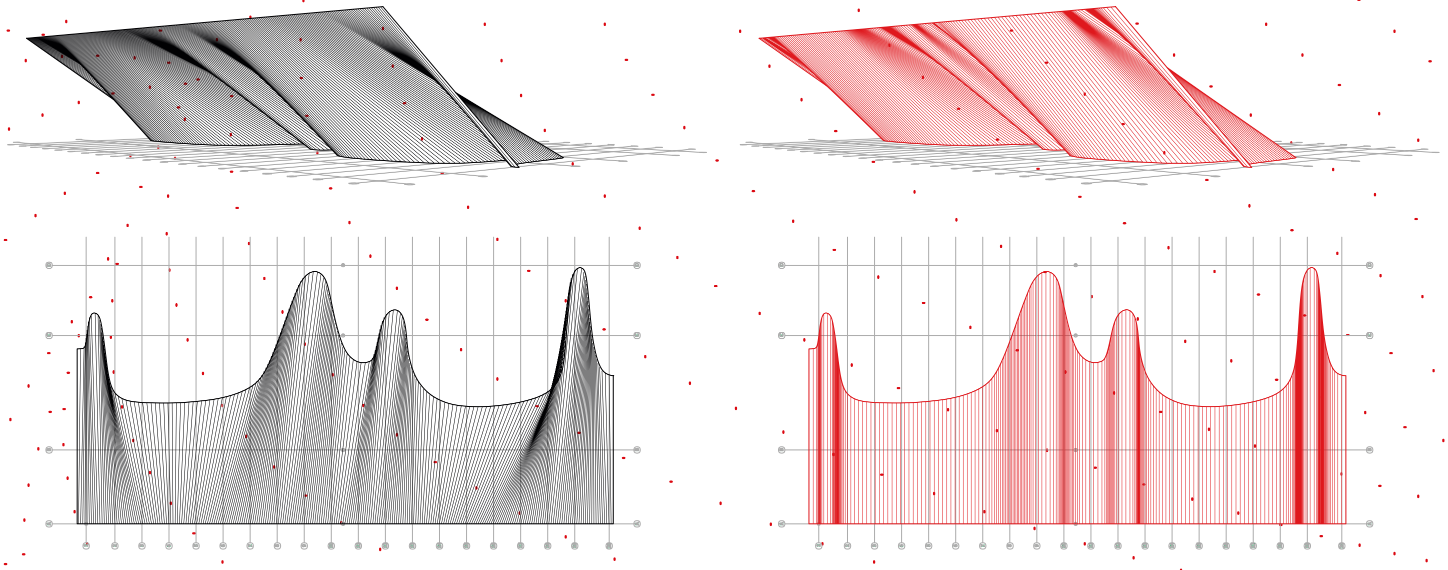


Figure 6. ALA Arkitekter AS, Kilden Performing Arts Center, Kristiansand, Norway, due for completion 2012
The facade's shape is defined by a ruled surface with a straight upper and a curved lower edge. For the intended prefabrication concept all generatrices had to be aligned with the building axes, a demand that could not be met with the default 'loft' method found in standard CAD packages (left), but needed a custom NURBS-definition (right).

surface. In this respect, a mesh can be abstracted by a NURBS surface like a polygon can be abstracted by a circle. But finding a proper NURBS representation for a given mesh or point-cloud requires quite a bit of knowledge of the underlying mathematical methods.

So it is more efficient to work with NURBS from the beginning, which is exactly what modern 3-D modelling software offers. But modelling a free-form surface means more than just tweaking control points; in order to come up with something buildable in the end, it means understanding the mathematical concepts behind those surfaces and relating them to the material world.

Abstracting Material

Curved surfaces have many geometrical properties that directly influence the options to actually build them (like developability and curvature radii). Most CAD programs can visualise those surface properties, but the designer has to interpret the colourful images and either match the design to the available material or find a suitable material for the given design. Speaking from experience, the latter approach is chosen far too often, frequently resulting in awkward and inefficient solutions.

For the development of smart solutions, the properties of both shape and material have to be known in detail. And, in order to precisely describe them in a 3-D model, the mathematics behind their physical behaviour have to be known too. If, for example, a curved surface is to be clad with thin strips of wood, it is easy to map a 'pinstripe' texture to the respective NURBS model and render a realistic looking but physically wrong image. In order to find out what really happens, one needs to base the stripe pattern on the bending characteristics of real-world wood strips. Only with this knowledge is it possible to create a valid geometry for all the slats on the surface and tell a fabricator how many to order and how to pre-cut them. Also, it enables the designer to optimise both surface and pattern for fabrication as well as for visual impression.

Abstracting Detail

Architectural design does not stop at defining an overall shape; a large number of components have to be joined to create a building. And, as soon as the underlying grid becomes non-regular, both components and joints must be adjusted to the geometrical situation at every grid position, rendering every piece unique. To save designers from manually modelling thousands of components, the concept of 'parametric modelling' was introduced: instead of describing the final result as a model, the process of modelling itself is described. A sequence of instructions (an algorithm) generates output (a detailed model) based on input (a set of parameters). By varying the input values, different output can be generated.

Abstraction in this context means to systematically develop a general solution suiting all individual components. This usually starts with finding the extreme cases – for example, the joints with extreme angles or the members with highest loads – and developing a parametric solution that can handle those as well as all intermediate cases. But since the (conflicting) requirements usually define a multidimensional solution space it is not always obvious whether all occurring cases are within the boundaries. Verifying the validity of a parametric solution might still require testing every single case. So, the challenge of building a parametric model is to untangle the interdependencies created by different requirements and find a set of rules that is as simple as possible while remaining flexible enough to accommodate every occurring case. In other words: to pinpoint the view to the exact level of abstraction where no important point is lost and no one gets distracted by unnecessary detail.

Reduction

Reduction, in contrast to abstraction, is not about reducing the amount of information but rather about finding the optimal way to transport it, hence rewriting the description without altering the content.

In the CAD domain, reduction can happen on different levels. Low-level reduction is about optimal descriptions of single geometric entities that save resources such as memory and disk space.

Fortunately, reduction on this level happens deep within the CAD system. It is higher-level reduction that is more interesting to the designer. Here we are mainly talking about two different procedures: elimination of redundancies and optimisation of descriptions and processes.

Normalisation

Redundancies (information that is present more than once) increase the weight of the model without adding detail and, more importantly, lead to update anomalies: the model can become inconsistent if only parts are updated. In database theory, the process of eliminating such anomalies is called normalisation. But it comes at a price: while changing information (writing) is made safer and quicker, extracting information (reading) becomes more complicated and generally slower, because it must be compiled from several spots throughout the dataset. Therefore, databases that are significantly more read than written are often kept partly redundant on purpose.

Carried over to CAD, this means that when creating 1,000 parametric components on one reference surface it could be sufficient to save one single point per entity and define a set of geometric operations to re-create its actual shape. However, this might render the model unusable as those operations have to be repeated on every information request. So the shortest possible description is not necessarily the best one. It might be worthwhile to keep some redundancies while carefully respecting update consistency.

Parametric modellers like McNeel's Grasshopper generally produce largely normalised models. When setting up a model, the user builds a hierarchical graph rooted in the input geometry. Grasshopper achieves a great deal, representing the graph visually and letting the user interact with it in a fairly intuitive way. Still, designers should be aware that the resulting geometry at every stage is volatile and immediately dependent on the input. While this eliminates the risk of update anomalies it also suppresses the possibility of deliberate redundancies or manual intervention.

Refactoring

The second flavour of high-level reduction could be described as cleaning up a model. Again, it can be rooted in computer science, where it is known as refactoring; that is, changing the source code of a program without changing its functionality in order to ensure maintainability and extensibility. This can be mapped directly to CAD: by throwing away superficial parts and simplifying parametric dependencies, a model can be kept sleek and efficient. This is especially important when it is used by more than one party.

However, it is important to note that reduction is irreversible: once we reduce a circle's description from three points to centre and radius, there is no way to get our initial points back – the information is retained, but not its history. This means that designers have to make sure the parts they eliminate are truly superficial; otherwise reduction becomes further abstraction that affects the functionality of the model.

Algorithms

Theoretical computer science is definitely unlisted on the average architecture student's agenda. But when parts of the design are delegated to computer programs (as in computational optimisation) or new computational tools are developed within a design process (as in parametric modelling), some knowledge about algorithms becomes key to understanding their influence both on the process and its result.

Determinism

First and foremost, contrary to a design problem, an algorithm has to be well defined. Since computers cannot guess based on experience and intuition, every step in a computer program has to be completely and unambiguously determined by the previous steps. Any decision making on how to proceed has to be already embedded in the program, and randomness is only simulated by numerical methods. Even computer programs that seem to exhibit experience (like expert systems) or random behaviour (like evolutionary systems) are running on deterministic hardware that can only switch currents on or off in a silicon chip (non-deterministic algorithms do exist, but they are mainly of interest for computational theory due to the lack of appropriate non-deterministic hardware).

Thus, defining an algorithm to solve a class of problems means to already know a general solution for those problems and describe a step-by-step process to derive an output from the given input. The first step usually is to assert that the input matches the problem specifications and can be processed (so the range of allowed inputs – the so-called 'parameter space' – has to be already well defined). From there on, the algorithm deterministically proceeds step by step, until it presents always the same final result for the same given input. Incidentally, evolutionary methods are no exception to this rule; they merely lift it to a different level of abstraction. The evolutionary method as such has to be well defined and deterministic, only the results are probabilistic.

Termination

Unfortunately it is not at all given that even a deterministic algorithm will eventually deliver a result. As soon as an algorithm contains some sort of loop it becomes hard to prove that it never gets lost in perpetual orbit for any given input. Consequently, parametric modellers like Grasshopper do not allow loops in their models. The data-flow diagram set up by the user always forms a directed loop-free graph assuring that data passes through without ever reaching the same point twice. On the other hand, iteratively executing the same step many times or even recursively calling an algorithm from within itself are very powerful and indispensable methods for efficient programs. And as a matter of fact, loops are used in parametric models, albeit only within the encapsulated components provided by the modeller and carefully hidden from the user to rule out infinite loops.

Computational Complexity

Sometimes even finite loops are too much. There are problems that can be solved by perfectly well-defined and provably terminating algorithms – only it takes far too long to wait for the result. A striking example is the so-called 'Travelling Salesman Problem' of finding the shortest route through all cities on a given list. An algorithm just has to generate all possible permutations of the listed cities, calculate the respective route lengths and find the shortest one. Since the number of cities is finite, so is the number of routes that can therefore be tested in finite time by a deterministic algorithm. The only problem is that for n cities the number of permutations accounts to ½×(n-1)!, a term that grows by the factorial (that is, the product of all positive integers

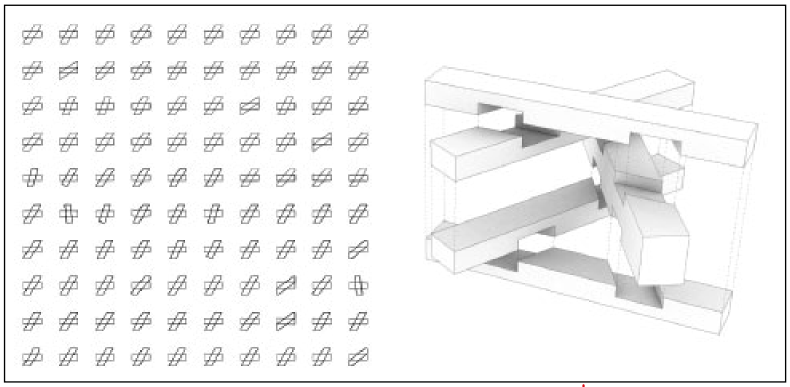


Figure 7. Shigeru Ban, Heasly Nine Bridges Golf Club, Yeoju, South Korea, 2010
To allow for continuous girders in all three directions, they are split into five layers with two lap joints at every crossing. The complete roof contains some 3,500 curved timber components with almost 15,000 lap joints. Even though many parts are similar, 467 individual components with over 2,000 different joints had to be described in detail. This was only possible by formally describing the whole structure in a parametric system that automatically generated the detailed models from a reference surface and some numerical parameters.

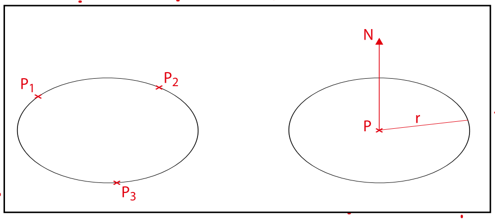
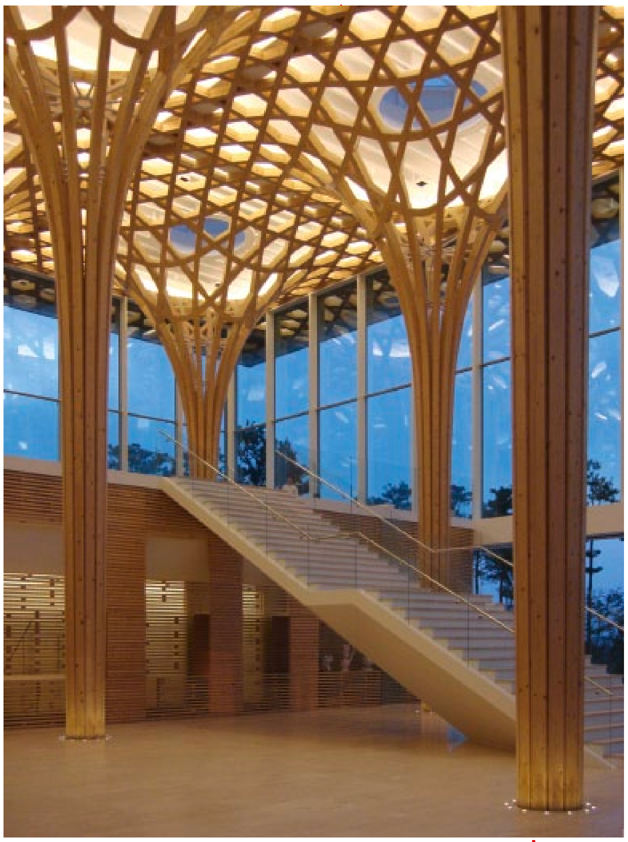


Figure 8. Reducing a circle
A circle can be unambiguously described by three points. However, if the notation is changed into one centre point plus normal vector and radius, the description size can be reduced from nine values (three points at three coordinates each) to seven values (two points and a number), saving 22 per cent.



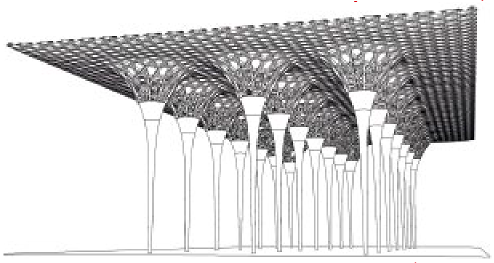


Figure 9. Shigeru Ban, Heasly Nine Bridges Golf Club, Yeoju, South Korea, 2010
The timber roof structure is defined by a regular tri-fold grid that is vertically projected to a curved surface. Girders are created on every projected grid line. Their orientation follows the surface, rendering them curved and twisted. The girders intersect at almost 7,500 crossing points.

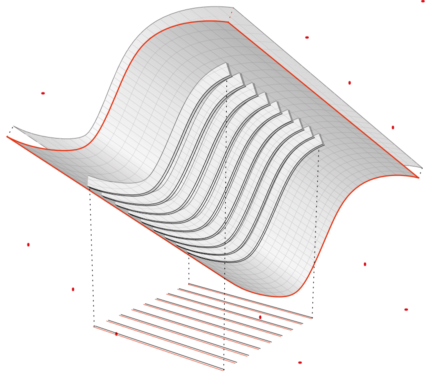


Figure 10. Curved beams in the Kilden facade
All timber beams are derived by projecting straight lines onto offsets from the same reference surface. The shortest possible description of each beam would therefore only contain one line, a width, a surface offset distance and a reference to the original surface. But the laborious offset and projection operations would have to be repeated whenever information about the beam geometry is needed. So it is reasonable to save the projected beam edges in the model, as long as they are updated when the reference surface changes.

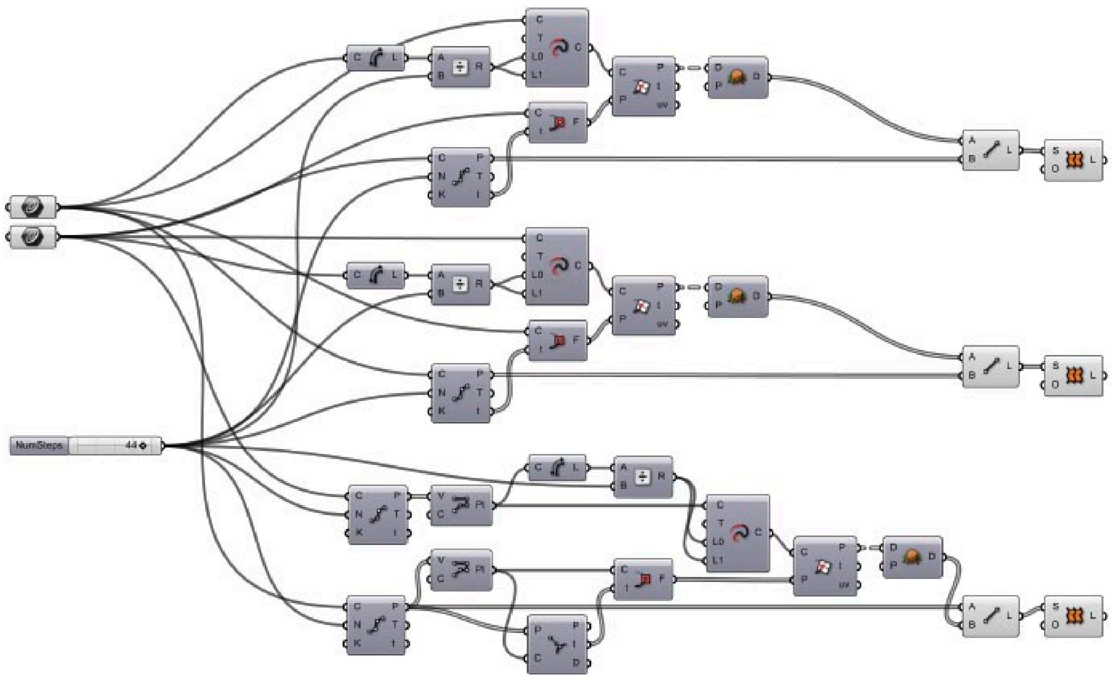
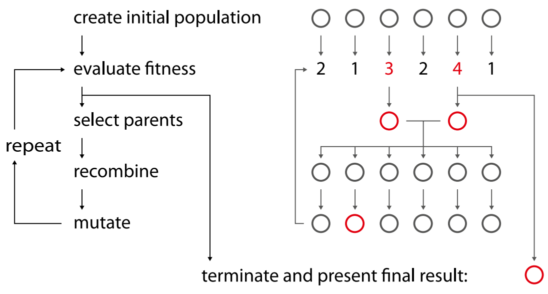


Figure 11. Parametric Grasshopper model
Parametric models describe the relations between different parts of a model as a graph where each node defines a (geometric) entity. The properties of one entity can be passed on to dependent entities, influencing their behaviour. This visually explains the flow of data and the hierarchy of entities in the model. Shown here is a parametric model that takes two curves and a number and generates three different NURBS surfaces.



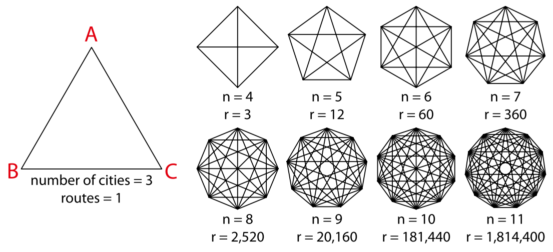


Figure 12. Schematic view of a genetic algorithm (GA)
Evolutionary methods seem to find surprisingly good results in vast solution spaces by chance, but they are based on completely deterministic algorithms. Notably the encoding of an individual's properties into a genome, the recombination of genomes during reproduction, and the selection based on a quantifiable fitness measure have to be formally well defined and unambiguous. Virtual dice are tossed at some steps of the algorithm to draw decisions, but this is also part of the predefined recipe.

Figure 13. The Travelling Salesman Problem
For three cities A, B and C, the six permutations would be [ABC], [ACB], [BCA], [BAC], [CBA] and [CAB]. If we assume that neither starting city nor travelling direction matter, those three routes are effectively the same. So for three cities, there is only one possible route. But that changes quickly for n>3.

less than or equal to a number) of the list length. For n=16 there already are ½×1×2×3×4×5×6×7×8×9×10×11×12×13×14×15 = 653,837,184,000 alternatives to check, which at a rate of one million routes per second takes about 7.5 days; and one single extra city would raise the waiting time to four months.

The amount of resources consumed by an algorithm in relation to the number of inputs is called 'computational complexity'. As the travelling salesman problem shows, the computational complexity of most problems does not scale linearly with the number of inputs. In particular, computational simulations like finite element analysis (FEA) and computational fluid dynamics (CFD) are not easily scalable, which makes it practically impossible to simulate large models in reasonable time, for example to use the results as fitness measures for evolutionary optimisation. Buying faster processors will only help momentarily; building leaner models and applying smarter methods is a much more sustainable approach.

Precision

It is a still common misconception that digital models are infinitely precise. Truth is that every computational operation on real numbers is subject to slight errors due to the fact that those numbers are stored as a combination of a whole number and an exponent, with finite precision (this is called 'floating point' to illustrate that the position of the radix point depends on the variable exponent). While these imprecisions can be neglected in most cases, they can add up and become relevant especially in complex geometric operations. Because of the finite number of digits available for both integral and fractional part, floating point errors are also dependent on the operands' magnitudes, which is why the exact same operation might succeed at the model origin [0,0,0] but fail at [10^{15},10^{15},10^{15}].

Furthermore, many fundamental geometric operations – like finding the intersection of two NURBS surfaces – utilise numerical approximation, which ultimately destroys the notion of infinitely precise CAD models. This is also the reason for CAD modellers to provide a tolerance setting. Increasing the tolerance can help working with imprecise input geometry, but will also lower the resulting quality. While decreasing it raises precision, it also elevates the barrier for geometric operations to succeed and boosts computation time.

While this impreciseness is inherent to geometric operations, it is notably not so to their formal description; formally defined models are precise until they are rendered into geometry. So the transition from formal relations to geometric operations is an important one that should be commenced carefully.

Of course, error-bound geometric operations are impossible to avoid in CAD modelling, but knowing when and why these errors occur can help to improve the construction sequence instead of just readjusting the tolerance settings when Boolean intersection fails again.

Quality

As we have seen, complex shapes can only be handled if digital or even parametric models are an integral part of the architectural design and communication process. Digital models, which aim at describing and simulating aspects of real objects, need to be set up carefully, with the right focus and the appropriate level of abstraction to deliver meaningful results. Especially when using parametric models, the hierarchic dependencies within complex structures have to be thoroughly untangled and precisely described in formal algorithmic and mathematic notations; only then can the output be rebuilt automatically upon changing the input parameters. But when a specific design is just one out of a myriad of possible instances a parametric machine can produce, what is the appropriate level to discuss the quality of design? Clearly, meaningful evaluation cannot stop at the skin-deep layer of the output's visible appearance. Nevertheless, today's architectural discourse rarely dives below this level, even though designers are gradually becoming programmers who design their own, highly sophisticated tools.

We think it is about time to discuss the quality of the processes instead of merely reviewing the end results that can be generated in endless variations. If we do not want to get lost in parameter space, we need to assess and understand the quality of the algorithmic machines we design, not the designs they produce. Which are the defining parameters of a model? Where and how does abstraction strike and why are certain things included and others left out? How are algorithms conceived and rules defined? What are the quantifiable – and therefore optimisable – measures for the quality of a design and how are they weighed against each other? What defines the quality of an algorithm and how does it reflect in its output? And finally, how can we communicate and discuss complex architectural structures in a meaningful way – not between digital machines but between the human minds assembled in a project team? Because in the end 'designing' means drawing decisions and taking the responsibility, not delegating them to a machine. Only this prevents algorithmic design, which is largely based on formal descriptions, from itself becoming formalistic. Δ

Note

1. See, for example, Helmut Pottmann, Andreas Asperl, Michael Hofer and Axel Kilian, *Architectural Geometry*, Bentley Institute Press (Exton), 2007. Also Jane Burry and Mark Burry, *The New Mathematics of Architecture*, Thames & Hudson (London), 2010.