DrAFT: an Algorithmic Framework for Facade Design

1 Facade: The Outer Layer of Architecture

The architectural facade has been the canvas of each architectural style. Although the arrival of the Modernism made it lose its status, since Post-modernism, we watched an increasing interest in facade composition and, nowadays, designing a facade is reassuming an important role in architecture practice due, in part, to the support of digital technologies (Pell, 2010).

This trend of highly textured building envelopes celebrates again the ornament in architecture and the composition of architectural facades. There are cultural and technical reasons for this renewed interest. On the cultural side, we find the reinterpretation of Modernist aesthetics, the reintroduction of symbolism and historical precedent by Post-Modernism (Venturi, 1966), and the revisitation of vernacular precedent proposed by Critical Regionalism (Frampton, 1983). On the technological side, we find the algorithmic approaches that simplified the design of complex and intricate architectural surfaces. Unfortunately, algorithmic approaches do not make facade design trivial. On the contrary, they require the rigorous specification of all algorithmic steps, a task that requires specialized knowledge and that, in many cases, can be quite complex.

In this thesis, we propose a well-defined and systematic methodology designed to simplify the algorithmic specification of facades. The methodology is based on the classification of the intended facade using a set of orthogonal dimensions which, then, guide the selection of the algorithms that best address the design problem. As we will show, the methodology promotes the exploration of facade designs and simplifies the adaptation of the generated models to the ever-changing design process conditions.

2 Algorithmic Approaches to Design

The computer changed, and is still changing, the way architects design (Kolarevic, 2003). As Terzidis states, computers "do not eradicate human imagination but rather extend its potential limitations...it provides the means for exploration, experimentation, and investigation in an alternative realm" (Terzidis, 2003). Generative design (GD) is a computer-based approach to design that creates shapes through algorithms (Terzidis, 2003). GD makes it possible to generate several different design solutions in a short period of time, while avoiding the tedious and repetitive tasks needed when the modeling work is done manually. Through GD, instead of going directly from the idea to the design, the architect produces an intermediate algorithmic-based description of a design (Leitão A., 2013). Parametric Design is a specific GD approach in which the parameters of a particular design are declared, rather than its shape (Kolarevic, 2003). This approach has the ability to generate different instances of a design, where each instance represents a particular set of values for the design
parameters (Barrios, 2005), allowing the designer to freely explore a large solution space of the design briefing/program. Ultimately, this leads to the assessment of solutions that would be difficult to generate with traditional design methods. An algorithmic-based design method can easily accommodate changes in the proposed solutions, as the dynamics of the design process alter the state of the design brief and its programmatic nature.

In spite of their advantages, algorithmic-based design methods require a disciplined approach which, in many cases, is difficult to follow. It is important, then, to develop strategies that help designers implement these methods. In this dissertation, we address this problem and we contribute to the state-of-the-art by proposing a strategy for the development of algorithmic-based solutions for the generation of facades. More specifically, we propose a computational framework for the design of facades.

3 A Framework for the Generation of Contemporary Facades

To develop a computational framework for the design of facades, we started by analyzing a large corpus of contemporary facades. The current variety of facades has already inspired several different classifications based on different concepts, such as Depth and Affect (Moussavi & Kubo, 2006) and facade’s Articulation (Pell, 2010). However, as we wanted a framework that helped designers with the generation of facades, we propose a different classification of facades that is more useful to the designer that intends to use a computational approach.

Our classification is divided into different categorical dimensions that we consider computationally relevant. This multidimensional classification guides the designer towards a library of functional operators that address the generation of different designs of facades. In practical terms, the designer matches his ideas for a particular facade with the categorical dimensions which, in turn, guide him in the selection of the most appropriate algorithms for the generation of the idealized facade. This guiding process is not intended to replace the role of the designer, as he is still responsible for the division of the whole design into parts, for establishing the dependencies between them, for instantiating and combining the different algorithms that handle each design part, and for any additional scripting that might be needed to handle specific circumstances of the design brief. However, the framework significantly reduces the programming effort.
3.1 Design Stages and Categorical Dimensions

Our framework for facade design takes into account the several stages that typically occur in such design, namely, (1) the definition of the facade’s geometry, (2) the generation of the facade’s elements, (3) the distribution of the elements (the mapping and rotation of the elements) and (4) the generation of the facade’s final appearance (materiality or color). Each of these stages is involved in one or more dimensions of our classification.


![Image of the classification's categorical dimensions.](image)

Fig. 3 – Image synthesis of the classification's categorical dimensions. The eight dimensions are organized in four different sets, which correspond to design stages: 1- definition of the facade’s geometry; 2- definition of the facade’s elements; 3- distribution of the elements and 4- facade’s final appearance.

The main objective of the classification is the identification and implementation of a set of algorithms and strategies that address the needs of different designs of facades. Some facade designs might directly match a specific computational approach in this multi-dimensional space but less common designs might not have a specific computational solution and, thus, might not have a match in some of the dimensions. In this case, some additional algorithmic development might be needed. This is intentional, as the goal of the framework is not to limit the facades that can be produced but, instead, to speed up the development of facades. Moreover, when additional algorithms are developed, they can be incorporated in the framework and, thus, further improve the matching process of subsequent facade designs.
3.2 The Application of the Framework

We called this framework DrAFT – Draft Algorithmic Facades Tool. To use this framework we start by selecting an operation from each dimension and, then, we implement the selected operations by combining them into a single expression. To make this framework more flexible we use higher-order functions, which are functions that receive other functions as arguments and/or compute other functions as results (Leitão A. M., 2014).

The Facade’s Geometry is in charge of providing the points that define a certain surface and the Element’s Geometry for providing an element with a certain shape. The last one is also combined with the Element’s Size dimension, which provides several types of size variations to the elements, and with the Element’s Distortion, which is in charge of producing several types of distortions on the elements. As we already have the surface points and the elements, the next dimension Element’s Distribution is in charge of mapping the elements on the surface points. This dimension has available several ways of mapping elements on a surface, starting with mappings in just one direction to mappings in two (creating grids of points) and three directions (including a temporal coordinate).

Regarding the placement of the elements, the Element’s Rotation dimension controls the positioning angle of the elements. The final facade can result from the subtraction of the elements from the facade’s surface, from the union of the elements with the surface or simply by the union between the elements themselves. This feature is controlled by the Facade’s Articulation dimension. Finally, the last dimension, Material & Color, is in charge of giving the materiality to the facade’s model. If the facade has the material in sight, the layer of the facade will have the name of the chosen material and will also present the chosen materiality. If the facade has colors in its final appearance, the process is the same but with the color name.

It should be noted that it is not mandatory to select an operation from all the available dimensions. As an example, if the elements do not have any size or rotation variation, we do not select any operation from the dimensions Element’s Size and Element’s Rotation.

In practical terms, DrAFT is a library of functional primitives and functional operators usable in different programming languages and also a set of guidelines that helps a designer select and combine the most useful operators to implement a design for a particular facade.

4 Exploring Several Design Solutions

By combining the operators available within DrAFT, we can generate several different facades. In this section we will explain how to generate the possible solutions.

As an example, consider we want a facade with straight geometry and with squared elements. For this, we select from the dimensions Facade’s Geometry and Element’s Geometry the operations which respectively correspond to a (1) straight surface and (2) squared elements. Let us also assume that we want the size of the elements to vary along the facade’s length, i.e. the elements have an increasing size variation. Our framework also provides an operation, increasingSize, to produce this type of size variation, which is available inside the dimension Element’s Size. Given this information,
we can define the function that generates each element as the functional composition of the functions that implement the selected algorithms, i.e., we combine both squaredGeometry and increasingSize.

Having the elements defined, we now move our attention to the distribution of the elements on the facade’s surface. We will consider that the elements are distributed in a regular-grid and, simultaneously, with a horizontal rotation. The elements’ rotation angle also increases along the facade’s length as it happened with their size. To implement these considerations, we select the operation `regularGrid` from the dimension Element’s Distribution, as this operation produces a regular distribution, and we select from the dimension Element’s Rotation an operation that gives a rotation to the elements. To actually distribute the elements on a certain surface, we have to combine these functions with all the functions that we selected so far. The `regularGrid` function is a higher-order function, which receives other functions as argument:

1. The function `element` — the function `regularGrid` knows how the distribution is done but it needs to know the element to distribute.
2. The function `straightGeometry` — the function `regularGrid` requires the set of points on which the distribution will be done.
3. The function `horizontalRotation` — the function `regularGrid` needs to know if there is some kind of rotation, when distributing the elements and how the rotation is done.

Lastly, we will assume that the elements — squares — are applied on the facade’s surface, which corresponds to the function `applied` in the Facade’s Articulation dimension, and the materials used are glass for the surface and black metal for the elements. For each of these classifications, we receive the most appropriate functions, which then we combine using the functional operators.

<table>
<thead>
<tr>
<th>FACADE’S GEOMETRY</th>
<th>ELEMENT’S GEOMETRY</th>
<th>ELEMENT’S SIZE</th>
<th>ELEMENT’S DISTRIBUTION</th>
<th>ELEMENT’S ROTATION</th>
<th>ARTICULATION</th>
<th>MATERIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRAIGHT</td>
<td>SQUARED</td>
<td>INCREASING</td>
<td>REGULAR GRID</td>
<td>HORIZONTAL</td>
<td>APPLIED</td>
<td>GLASS &amp; BLACK METAL</td>
</tr>
</tbody>
</table>

Fig. 4 – An image of a pattern produced by the classification in the table above.

In the first example, both size variation and elements rotation vary along the facade’s length. Now, if we change the type of element's distribution to become an alternated-grid, we generate the following facade example (Fig.5). We could also change the distribution to become a chess-grid and the final result would also be different (Fig.6). These changes do not require changing the rest of the structure, which corresponds to the functions provided by the other dimensions, but simply changing the name of the function in charge of the elements’ distribution:
Lastly, imagine that this facade has now a pictorial size variation, which produces the image selected by us: an image with a characteristic pattern of the Portuguese stone pavement. For this, we exchange the function provided by the Element’s Size dimension, increasingSize, for the function pictorialSize and the elements – the squares - will vary their sizes according to the color intensity of the pixel that they represent.
As another completely different example, imagine we want a facade with pictorial elements with a shape similar to a diamond. We want the elements to vary their sizes according to a distance to a point – the attractor point. In addition, we want a regular distribution on a horizontally sinusoidal surface. For its final appearance we want a surface made by the juxtaposition of the elements, which are characterized by the color light gray.

<table>
<thead>
<tr>
<th>FACADE’S GEOMETRY</th>
<th>ELEMENT’S GEOMETRY</th>
<th>ELEMENT’S SIZE</th>
<th>ELEMENT’S DISTRIBUTION</th>
<th>ARTICULATION</th>
<th>MATERIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>HORIZONTALLY SINUSOIDAL</td>
<td>PICTORIAL</td>
<td>ATTRACTED</td>
<td>REGULAR GRID</td>
<td>JUXTAPOSED</td>
<td>LIGHT GRAY</td>
</tr>
</tbody>
</table>

Fig. 8 - An image of the pattern produced by the classification in the table above – with a Horizontally Sinusoidal Surface.

To change the shape of the surface we simply have to select another operation from the Facade’s Geometry dimension. The implementation and the operations of all the other parts remain equal and they automatically adapt to the selected new shape. We experimented the pattern on a sin-cosinusoidal surface and on a more complex geometry, a cylindrical sinusoidal surface.

<table>
<thead>
<tr>
<th>FACADE’S GEOMETRY</th>
<th>ELEMENT’S GEOMETRY</th>
<th>ELEMENT’S SIZE</th>
<th>ELEMENT’S DISTRIBUTION</th>
<th>ARTICULATION</th>
<th>MATERIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN-COSINUSOIDAL</td>
<td>PICTORIAL</td>
<td>ATTRACTED</td>
<td>REGULAR GRID</td>
<td>JUXTAPOSED</td>
<td>LIGHT GRAY</td>
</tr>
</tbody>
</table>

Fig. 9 - An image of the pattern produced by the corresponding classification in the table above – with a Sin-cosinusoidal surface.

<table>
<thead>
<tr>
<th>FACADE’S GEOMETRY</th>
<th>ELEMENT’S GEOMETRY</th>
<th>ELEMENT’S SIZE</th>
<th>ELEMENT’S DISTRIBUTION</th>
<th>ARTICULATION</th>
<th>MATERIALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYLINDRICAL SINUSOIDAL</td>
<td>PICTORIAL</td>
<td>ATTRACTED</td>
<td>REGULAR GRID</td>
<td>JUXTAPOSED</td>
<td>LIGHT GRAY</td>
</tr>
</tbody>
</table>
5 Evaluation

In order to evaluate our framework we used it not only to generate some existing facades but also to reproduce some abstract facades. The following images show several examples, each one accompanied by the corresponding classification and generation time. We can conclude that our framework supports the generation of several different complex patterns. Equally important is the effort required to use the framework. Our empirical evaluation shows that the classification step requires between five and ten minutes, while the selection, composition, and testing of the functions suggested by the classification takes between fifteen minutes and one hour, depending on the complexity of the facade.

Fig. 11 – Examples of patterns produced using the framework with the corresponding classification and time of generation a/b: a- the time each model took to classify; b- the time took to test the functions.
The set of examples presented in Fig.11 has its generation time between ten and thirty minutes, including the time required to classify and also to test the functions. Nevertheless, imagine we want to generate a pattern constituted by elements with a complex geometry, a free-form facade that must be implemented by us or a type of elements distribution that is not available in the framework. This example would integrate the set of the most time consuming designs to explore, which would take approximately one hour to implement new functions and test the existing ones.

6 Portability

The current implementation of the framework was done using the Rosetta IDE (Lopes & Leitão, 2011). This has the significant advantage of making the framework portable across the different CAD tools supported by Rosetta. This portability allows us to produce identical models in different CAD tools, such as Rhino, AutoCAD, SketchUp and Revit. In fact, the use of this framework is not restricted to a single CAD tool, as it happens with other similar frameworks, thus liberating the designer from the limitations of any specific CAD software. Moreover, it allows the designer to easily change the CAD tool that he wants to use.

Additionally, Rosetta also promotes portability across the supported programming languages, allowing the exploration of the framework in different programming languages such as Autolisp, Phyton, Processing and Javascript. As a result, in order to use our framework, designers can choose the programming language that they are more familiarized with, without forcing them to learn a new language.

7 Conclusions

The exploration of architectural facades is not new. However, by resorting to recent digital technologies, architects can once again focus on facade design, promoting a growing interest in the exploration of complex patterns and geometries.

In this thesis we developed the DrAFT framework, that helps designers to generate different facade designs through the use of a set of functional operators, to generate different patterns of facades. The current implementation of the framework was done using the Rosetta IDE (Lopes & Leitão, 2011), allowing its exploration in different programming languages.
The framework was also proposed by us and uses a classification of facades based on several categorical dimensions. We demonstrated how the classification is useful to select a set of functional algorithms for each type of facade design. The algorithms might then be used directly, or might be combined using functional operators, promoting a systematic exploration of designs which ultimately aims to a higher productivity by: (1) improving the time of scripting tasks, and (2) adding flexibility to the designers’ workflow. Due to the simplicity of the functional composition, this framework accommodates the ever-changing nature of a design process by facilitating the test of several design concepts, or instantiations of the same idea, in any design stage. In this paper we explored a set of possible designs for facades and we also tried to demonstrate the existing flexibility in each one of the examples.

In the near future, we plan to expand the set of functional algorithms and operators, covering a wider range of facades. In order to make this framework more usable, we are particularly interested in conducting a wider field study of its application, to identify weaknesses of the proposed processes and opportunities for extensions.

References