Computational Implementation of a Tool for Generative Design of High-rise Residential Building Facades

Guo Fei ¹, Eduardo Castro e Costa², José Duarte³, Shadi Nazarian⁴

¹ The Pennsylvania State University
fxg20@psu.edu

² The Pennsylvania State University
erc15@psu.edu

³ The Pennsylvania State University
jxp400@psu.edu

⁴ The Pennsylvania State University
sun14@psu.edu

Abstract. We propose a computational design tool that aims to provide more variety to the design of high-rise residential building facades. In contemporary cities, the pressure to build many high-rise residential buildings leaves little time to focus on facade design, resulting in repetitive facades that impart a monotonous appearance to cities. We propose a computational tool that can help to improve facade variety, based on shape grammars and parametric modeling. Shape grammars are used to analyze facade composition and to structure design knowledge. Subsequently, the grammars are converted into parametric models, which are implemented using the Python programming language that can be used to generate designs in CAD software. The resulting tool encodes a general parametric model that manipulates the rules of formal composition of building facades. Without limitations from software, the program takes advantage of the high-processing power of the computer to provide many design solutions from which architects can choose.

Keywords: Variety, Facades, Computational Design, Parametric Modeling, Shape Grammar.

1 Introduction

High-rise residential buildings can fulfill the housing needs of rapidly growing populations (Cohen 2003). Consequently, in recent years numerous high-rise residential buildings have been constructed in a short time. The resulting pressure to design more buildings in less time leaves architects with little time to focus on facade design, which in turn results in a tendency to design buildings with similar facades, rendering cityscapes monotonous (Salingaros 2011) (Figure 1).
Some designers attempt to recreate the designs of classical buildings, but simply copy the heads and bases and repeat the shaft layers without changes. The resulting facades correspond to taller versions of traditional ones (Figure 2). Architects are responsible for designing original facades instead of simply reusing previous or existing designs. In order to avoid such a level of repetition, new design methods should be explored.

This paper explores the use of computation to generate various facade options based on contemporary design schemas. The use of computational technology in architectural design can enable a better understanding of building types through the systematic analysis of facades compositions. The technology also enables architects to generate a variety of options during the design of facades, as well as allowing the architects to focus on other design issues.

Figure 1 Residential buildings in Singapore

Figure 2 Taller version of classical buildings

2 https://baijiahao.baidu.com/s?id=1605391885189284122&wfr=spider&for=pc
2 Related Works

Earlier initiatives have used computational techniques towards facade generation, adopting several different approaches. CADaFed (Computational Architectural Design approach based on Fractals at the Early Design) uses existing buildings as reference to generate alternative facade designs according to traditional architectural patterns (Guerbuez et al. 2010). Haegler et al. (2010) use a rule- and texture-based method to encode facade structures into grammars for quickly generating building designs. Bao et al. (2013) and Musialski et al. (2012) apply segmentations and labels to find design rules in existing buildings. Duarte (1995) uses modules to develop new facade compositions, namely a system of anthropomorphic proportions. Yet, these solutions are considered passive in the sense that they are limited to the analyzed buildings avoiding further explorations of new styles. Computer Generated Architecture (CGA) presents a particularly relevant approach by implementing a parametric split grammar that enables real-time generations of urban design featuring detailed geometry (Wonka et al. 2003, Müller et al. 2006, Halatsch et al. 2008). Yet this flexible tool does not solve the facade variety. Steinø (2017) argues in favor of defining a facade syntax supported by empirical facade analyses. However, such analyses are often limited to visual observations for the purpose of city planning.

The gaming industry also promotes the development of computational applications capable of generating buildings in three-dimensional city models. Although extensive models can be created efficiently, such applications are limited to reproducing existing buildings for the purpose of city planning instead of architectural design. The design of high-rise residential buildings is often neglected in these approaches. Wu et al. (2013) generated stochastic high-rise building facades with split grammars for given facade layouts; however, every floor remained the same and the problem of monotonous facades was not solved from an architectural viewpoint. Matcha and Quasten (2009) developed a research program that plugs into Autodesk Revit to generate customized mass single family housing units through a parametric system; however, the options exposed to users lack detailed facade customization. Fast processing repetitions of monotonous designs should evolve from random generations into proper approaches to easy programming and architectural design rules. New designs should not only rely on image processing but also on understanding the composition principles of facades.

3 Methodology

A computer program was developed based on rules extracted from the analysis of precedents that identified facade types. Such types were then encoded into a generic parametric grammar (Beirão and Duarte 2018) that enables generating many different instances of the type. The paper begins by presenting examples of different facade
types and then describes the design rules extracted from these precedents. Finally, it shows how the precedents can be used to generate new facade designs.

These rules were aggregated and encoded into a parametric shape grammar (Stiny, 1980) allowing for an informed development of the generative design system – through detection of redundancies for example – thus making it more efficient. In order to make it accessible to a larger audience of designers, the design knowledge encoded in the shape grammar was later implemented into a parametric model (Barros, Chaparro, & Duarte, 2014). In this model, the dimensions of shapes are expressed by parametric values so that different parameter ranges generate different shapes. Such parametric shapes can generate various new shapes by changing the parameter values in a range that is pre-determined by the tool designer according to the type of component. The resulting program can create numerous facade designs beyond the design solutions analyzed as precedents by combining programming components corresponding to generic shape grammar rules.

4 Analyzing precedents and inferring design rules

This section documents the development of a parametric model that encodes the rules for designing a facade. Such rules were inferred from four selected facade styles (Figure 3). The first style corresponds to a basic grid facade, as exemplified by the Jianwai Soho development (Riken Yamamoto, 2004, Beijing, China). The second style, represented in the Sukhothai Residences (Kerry Hill, 2010, Bangkok, Thailand), features a unique arrangement of bays. The third style is based on Kristall Tower Holzhafen (ASTOC, 2011, Hamburg, Germany), which presents a variation in the width of bays. Finally, the fourth style present variations in depth, exemplified by the Mill Owners’ Association building (Le Corbusier, Ahmedabad, India, 1954). The developed parametric model departs from these four styles of facade. In the model, a facade is generated through four consecutive steps: the division of the whole facade into floors, the division of floors into arrangement of bays, the division of bays into components, and the definitions of each component. Different styles follow different design rules in each of these steps.

Figure 3 Four styles of facades

Figure 4 shows how a facade can be divided into floors and bays. Steps 1 and 2 use \( R2 \) to generate the floors according to their story heights. Steps 3 and 4 generate bays according to spans with \( R1 \). This simple division schema corresponds to Style 1.

Figure 5 illustrates the arrangement of bays in a Style 2 facade. Figure 6 displays the generation of the four different bay types using rule \( R1 \). Rule \( R2 \) provides the
vertical division into different floors. In Style 2, the facade design repeats every two floors instead of every floor, the two repeating floors being different from each other. The repeating pattern is different from the 6th floor up. When identical bays are vertically aligned, the separation between floors is blurred. Table 1.1 and Table 1.2 show the arrangements of bay types in the representative floors, using symbols $a$, $b$, $b'$, and $c$ to represent the different bay arrangements. Figure 7 shows how the facade can be generated step by step.

The rules inferred from Style 2 can generate many different facade designs, and therefore this example demonstrates how a few simple rules can generate many complex facades.

Analogously, Figure 8 shows the bay arrangements of Style 3 using symbols $a$, $b$, $oa$, and $ob$ to differentiate among four bay types. Types $oa$ and $ob$ feature opaque panels, which have the same sizes as windows $a$ and $b$. The composition of even and odd floors is presented in Table 2. Figure 9 shows how the facade is generated.

Style 4 provides an example of a frame with a horizontal tilt transformation, which will be shown in Figure 11.
Figure 6 Generation of bay types in Style 2

Figure 7 Derivation of Style 2

Table 1.1 6th floor up bay arrangement in Style 2

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Even</td>
<td>b'</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b'</td>
<td>a</td>
<td>b'</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>Odd</td>
<td>b'</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>

Table 1.2 6th floor down bay arrangement in Style 2

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Even</td>
<td>b'</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b'</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>Odd</td>
<td>b'</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b'</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>

Table 2 Bay arrangement in Style 3

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Even</td>
<td>ob</td>
<td>b</td>
<td>ob</td>
<td>oa</td>
<td>a</td>
<td>ob</td>
<td>a</td>
<td>ob</td>
<td>a</td>
</tr>
<tr>
<td>Odd</td>
<td>b</td>
<td>ob</td>
<td>oa</td>
<td>a</td>
<td>ob</td>
<td>oa</td>
<td>b</td>
<td>oa</td>
<td>ob</td>
</tr>
</tbody>
</table>
5 Implementation of the parametric model

In the computation implementation of the parametric model, buildings are generated according to the hierarchy mentioned earlier: bays are made of components; floors are made of bays; buildings are made of floors. This hierarchical order is similar to how traditional design processes approach facade design. Each of these hierarchical levels corresponds to a set of Python functions. A first set of generic functions generates the different facade components individually. A second set of functions combines specific combinations of components into bay types according to particular styles. Such specific functions can easily be combined into floors by aggregating them into lists. The functions are executed automatically by iterating such lists, generating the facade floors. Finally, floors are combined into whole facades, reflecting the rules inferred in the previous section.
5.1 Facade components

Facade components include structural framework, opaque panels, and windows. In the implementation of the parametric model, a generic function generates each component. For example, the windows on the facade are generated by a function called `windowmodel`. Its parameters include window width, the number of horizontal and vertical divisions of the glazed areas, the span, the side distance to the boundary, and the distance to the top (Figure 10). The encoded function can generate window models according to the above parameters. Figure 12 shows the `windowmodel` function code.

The `frame` function is used to generate the structural components or the ornament outside the windows (Figure 11). The parameters are the frame height, middle poles, span, width and depth of frames. The parameter `depthX` is used to produce a horizontal tilt transformation, as style 4 shows. It is defined by the tilt distance rather than by an angle, making it easier to express in architectural dimensions. The `frame one`, `frame two` and `frame H` are the elements within the frame. The default value is `False`, meaning these elements will not be generated until the parameter is activated. This allows the frame function to generate different bay designs.

---

**Figure 10** `windowmodel` function parameters

**Figure 11** `frame` function parameters

**Figure 12** `windowmodel` function code
The `opaquePanel` function is a versatile generic component that can generate several architectural components, such as walls, floor slabs, parapets etc. Its parameters are width, height, distance from the top, and thickness.

In the Python implementation, each function returns a value. Such values are used in the iterative generation of floors using `for` loops and used to find subsequent origin points where new bays can be positioned.

### 5.2 Defining bay types

The generic component functions are combined in order to define differentiated bay types according to parameters input from users. Such parameters will be passed into each generic component function to generate specific designs for components thus defining styles. For example, in the code of Style 1, the `windowmodel` function is specified into functions `wa` and `wb`, which correspond to different window types. These functions have only one or two parameters, reduced from the six parameters that define `windowmodel`. The arrangement of different types is encoded into a list in Python, composed of specified functions like `wa` and `wb`. Floors can thus be generated directly by iterating the list of specified functions with a `for` loop. This method allows the user to easily set up a new style based on the original ones.

Lists are executed by loops to generate floor designs (Figure 13). Each iteration within the loops determines the origin’s coordinates where the component should be generated. The coordinate is calculated from the value returned by the component functions, which corresponds to the component’s width. This calculation allows the user to generate components automatically one by one.

```python
for i in range (0,frameFloors):
    sum=0
    for j in range(0,totalBays):
        origin=[sum,1*frameHeight+upBaseHeight,0]
        if (i%2==1 and (i>6):
            spanOFBays=oddFloors[j%10](origin,frameHeight)
            sum=sum+spanOFBays
        if (i%2==0 and (i>6):
            spanOFBays=evenFloors[j%10](origin,frameHeight)
            sum=sum+spanOFBays
        if (i%2==1 and (i<=6):
            spanOFBays=bottomOddFloors[j%10](origin,frameHeight)
            sum=sum+spanOFBays
        if (i%2==0 and (i<=6):
            spanOFBays=bottomEvenFloors[j%10](origin,frameHeight)
            sum=sum+spanOFBays

Figure 13 List execution
```

In implementing bay types, conditional clauses are used to evaluate whether a specific component should be placed. The option of stairs and random arrangements are examples of such conditions. Conditional clauses are also used to evaluate whether floor number is even or odd, or if it corresponds to an exception, such as a
ground or top floor. Different conditions execute different lists, such as even lists and odd lists. Currently, the arrangements of different kinds of components correspond to separate loops. This can be changed in future developments of the program. As a result, different sequences can generate top floors, bottom floors, windows, frame generations or side facade creations.

High-rise residential buildings typically feature similar spans and heights. Spans and heights will define the grid, allowing loops to generate facades automatically according to the numbers of floors and bays. The process is very similar to creating grids. The loops also perform repetitive tasks. Repetitions still exist even if facades are different from each other. These repetitions result from economic considerations, since standard units can save money through mass production.

5.3 Generation

This section illustrates the program’s generative potential with a set of facades resulting from different parameter values. We begin with nine designs of Style 2 facades (Table 3). Table 5 shows the corresponding input parameters. The first example in each style is generated using the default values for that style and it corresponds to the precedent design used to infer the rules. Changed values will produce a new facade. Table 3 shows that a random arrangement of bays can generate a new facade design featuring the same bay types as in the original facade (Design 2, 3, 5 and 7). The component shapes are also changed by the width and heights. Design 8 (Table 5) changes the heights of frames to 9000mm from the original 3000mm to create an original facade, while keeping the same component arrangements. Iterating different lists can change the arrangement of components, corresponding to new designs. Table 4 and Table 6 show facade designs corresponding to Style 3 and the corresponding parameters respectively. Table 7 and 8 show the generation of Style 4. By introducing other style components into the program, Style 1 can generate different designs as shown in Table 9 and Table 10.

Table 5 Input parameters for examples of Style 2

<table>
<thead>
<tr>
<th></th>
<th>1 (default)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>total numbers of floors</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total numbers of bays</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the width of frame</td>
<td>600</td>
<td>300</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of bottom floors</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The height of frames</td>
<td>3000</td>
<td>6000</td>
<td>6000</td>
<td>9000</td>
<td>12000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random arrangement of bays</td>
<td>X ✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3 Generated facade designs in Style 2

#### Table 4 Generated facade designs in Style 3

### Table 6 Input parameters for examples of Style 3

<table>
<thead>
<tr>
<th></th>
<th>1 (default)</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>total numbers of floors</td>
<td>8</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total numbers of bays</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the width of frame</td>
<td>100</td>
<td>300</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>number of bottom floors</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change in Code (question window using default value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 frame height is 6000 mm</td>
</tr>
<tr>
<td>6 (a=1200)</td>
</tr>
<tr>
<td>7 (a=600)</td>
</tr>
<tr>
<td>8 (a=600, b=1200)</td>
</tr>
</tbody>
</table>

### Table 7 Input parameters for examples of Style 4

<table>
<thead>
<tr>
<th></th>
<th>1 (default)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>total numbers of floors</td>
<td>3</td>
<td>9</td>
<td>13</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total numbers of bays</td>
<td>11</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the width of frame</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lift distance</td>
<td>2000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>span of bays</td>
<td>2000</td>
<td>3000</td>
<td>1000</td>
<td>1500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6 Discussion

We argue that the program presented in this paper constitutes a tool that can efficiently support architectural design as follows, since it can be used to rapidly generate ideas for preliminary facades designs, similar to “white box” models used in city planning. As a result, designers can quickly progress to the next design phases.
Generated facade designs can be compared, offering the user the opportunity to gain insights for improving the design further. Then, additional facades can be generated quickly to further test design ideas. In this way, the program contrasts with conventional design techniques that might cause architects to refrain from exploring design iterations requiring cumbersome, time-consuming changes.

This program was created as an experiment under the premise that combining basic shapes can lead to a meaningful beginning. The design of high-rise residential buildings is often reduced to the design of simplified boxes. The main purpose of the program is to provide facade variety by using basic shapes. However, it does present some limitations, as the analyzed precedents do not represent all possible facade styles. For example, the analyzed precedents fail to represent curved facades, for example. Also, the selection of precedents did not take aesthetic considerations into account. The precedents were chosen based on the idea of using several basic shapes to generate facades in order to demonstrate the validity of the program, and the criterion used in the selection was that the precedents should be considered paradigmatic or influential of current contemporary high-rise residential buildings. In fact, the construction of most residential buildings is often constrained by economic considerations, and therefore simple shapes can be perceived as more likely to save money.

The program parameters are intricately connected to each other. For example, the window’s height and width are linked to the bay’s height and span. For this reason, the design of the windows can be regenerated automatically when the bay design changes, as opposed to conventional CAD software. This advantage can become a disadvantage when adding new styles to the program. Since every function needs variables to define properties that satisfy the requirement of individual styles, when new styles are added to the program, functions may need to incorporate new properties. For example, the frame of Style 3 has a new tilt feature, so the original function developed for Style 1 and 2 had to be extended to include this attribute. However, this feature then had to become active in the other styles, inevitably, requiring the associated parameter value to be provided when considering other styles, which might have a negative impact. Moreover, to further extend the current program capabilities the user needs to know the Python programming language used to develop the program, although an interface to overcome this limitation could be developed.

The proposed program separates components are from each other. For example, different loops generate windows and frames are generated by. Conventional CAD applications build components together which can render the model rigid. In contrast, considering components separately can make the model more flexible, allowing for more variation in facade design. For instance, while windows are constrained to the bays of each floor, frames can extend across several floors. Such a separation allows each component to have independent parameters. This strategy allows multiple combinations of components. However, coordinating the changes to components also requires an additional effort. For instance, the designer can choose the bay in which the stairs component is located. In this case, the windows and frames should not be generated in these bays. Because the generation of components is automatic, each
component loop should consider stair locations to avoid conflicting with other facade elements.

Currently, the program does not contemplate all the components required in residential buildings facade design. For example, the program excludes louvers and balconies for example, as it is difficult to find individual sample facades that includes all the components. Although two or three components are enough to make facades unique, new incorporated components can provide more facade varieties. New component functions can be added to the program in future iterations to overcome current limitations.

7 Conclusion and perspective

The presented program was shown to have potential to be an effective tool for generating various high-rise residential building facade designs. With this tool, the design of facades is much faster than in conventional design processes because it can reduce time spent in doing repetitive work and help to explore different arrangements of the elements that compose facades. Although the design of facades generated by the program is based on a small sample of existing facades, the program can generate many other possibilities based on the same design rules. Nevertheless, the facade styles currently encoded into the program need to be expanded to include other requirements, not only compositional but also functional, such as structural and environmental.

The proposed program uses algorithms to encode the design of facades. Although different facades may have different compositional structures, they still share elements, such as floors, bays, windows, and panels. Therefore, generic elements can be defined using the same general parameters and later specified by specific parameters. This property enables the development of parametric models of facade designs.

Shape grammars were used as an initial step to structure the algorithm used to define the design of facades. They offer a valuable theoretical foundation to analyze facades and encode facade rules into the program. However, the program developed does not fully reflect the shape grammar. It was not developed as a proper shape grammar interpreter, since it cannot recognize shapes (Chau 2004). Further work is necessary to reflect the shape grammar more closely in the computer program.

Compared with conventional 3D modeling computer programs, this program facilitates the generation of different facade designs through variation of parameter values, thereby avoiding remodeling. Mathematical relationships among facade elements, which are built into the program, permit the user to redesign a facade automatically when parameter values associated with some of its elements are changed, thereby avoiding the need to revise separate elements one by one. The program presented in this paper can be further developed to support the generation of facades that consider functional aspects, such as sunlight harvesting and sound protection.
References

