P. Janssen, P. Loh, A. Raonic, M. A. Schnabel (eds.), *Protocols, Flows and Glitches, Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2017*, 407-417.

EMPOWERING URBAN DESIGN PROTOTYPING

A Case Study in Cape Town with Interactive Computational Synthesis Methods

YUFAN MIAO[1], REINHARD KOENIG[2], PETER BUŠ[3], MEI-CHIH CHANG[4], ARTEM CHIRKIN[5] and LUKAS TREYER[6]
[1,3,4,5,6]*ETH Zürich, Switzerland*
[1,3,4,5,6]*{miao|bus|chang|chirkin|treyer}@arch.ethz.ch*
[2]*Austrian Institute of Technology, Austria*
[2]*Reinhard.koenig@ait.ac.at*

Abstract. Although Cape Town city in South Africa is generally regarded as the most stable and prosperous city in the region, there are still approximately 7.5 million people living in informal settlements and about 2.5 million housing units are needed. This motivates the so-called Empower Shack project, aiming to develop upgrading strategies for these informal settlements. To facilitate the fulfillment of this project, urban design prototyping tools are researched and developed with the capabilities for fast urban design synthesis. In this paper we present a computational method for fast interactive synthesis of urban planning prototypes. For the generation of mock-up urban layouts, one hierarchical slicing structure, namely, the slicing tree is introduced to abstractly represent the parcels, as an extension of the existing generative method for street network. It has been proved that our methods can interactively assist the urban planning process in practice. However, the slicing tree data structure has several limitations that hinder the further improvement of the generated urban layouts. In the future, the development of a new data structure is required to fulfill urban synthesis for urban layout generation with Evolutionary Multi-objective Optimization methods and evaluation strategies should be developed to verify the generated results.

1. Introduction

Modern urban design is characterized by increasing complexities and dynamics. However, traditional urban design methods still rely heavily on the static and sectoral approaches, which cannot fully fulfill such complex and dynamic requirements. At the same time, from modelling to manufacturing, computers are playing an increasingly important role in the design process. Many phases in design once

carried out by hand are nowadays automated (Bolognini et al. 2011). The automation of urban design enables the fast exploration of different design possibilities so that the designers could have a comprehensive overview and understanding of different design choices through their interaction with the computing tool. This method is what we call Urban Design Prototyping (UDP), which employs computational synthesis methods, aiming at facilitating designers' task through the automated generation of optimal urban design solutions.

Urban Design Prototyping is deeply rooted to the so-called Virtual Prototyping (VP) which has been widely studied and implemented for engineering designs (Reinhardt et al. 1993; Malik et al. 2006; and Bringmann et al. 2015). From the definition of Song et al. (1999), VP refers to "the process of simulating the user, the product, and their combined (physical) interaction in software through the different stages of product design, and the quantitative performance analysis of the product", which is emphasized on the interaction between human and product, and the role of VP in the design process (Wang 2002). As a subset of Virtual Prototyping, Urban Design Prototyping adapts this philosophy to the context of urban design, where the footstones are the interaction between urban designers and urban layout, and the functions of the prototyping tools for analysis and prediction. Inherited from VP, UDP also has the advantage of reducing time, saving costs, and increasing quality (Rix et al. 2016). More specifically speaking, it reduces the time for urban designers to explore different possibilities of mock-up urban layouts; it saves the costs of manpower and fortifies the quality of the design solutions by testing out different possibilities.

The present research evolves within the context of the Empower Shack project (Urban ThinkTank 2016), taking place in the Cape Town city in South Africa where there are still approximately 7.5 million people living in informal settlements and about 2.5 million housing units are needed. The Empower Shack project aims to develop upgrading strategies for these informal settlements. Our research is alight to meet one principle of this project, the "scalable methodology" principle which is to enhance the design process with new digital planning tools and evaluation framework to ensure the scalability and relocability. To fulfill that, our proposal is to develop one portable UDP tool that could be fast adaptive to changes of the requirements from urban designers and stakeholders. Besides, our tool is "designer-centered" with two major characteristics that should be highlighted. The first one is the capability of interaction between the designers and the program whereas the second one the capability of generating urban layouts based on parameters popularly used in urban design rather than in computer science. They are important because in reality, computers in many cases are not as creative as the human designers. The interaction helps improving the design and parameters in the "right language" help urban designers to specify their needs naturally. For the interaction, it is realized with an existing software platform, Rhino together with its add-on Grasshopper, which is relatively familiar to urban designers. For the parameters, they are determined through discussions with urban designers and stakeholders of the project. In this way, with this paper, we present a computational method for fast interactive synthesis of urban design prototypes by concentrating on i) the description of the methods for the generation of urban fabric and ii) the

presentation of a case study.

The remainder of the paper is composed of four sections: methods, case study, results and discussion, and conclusion and future work. In the methods section, the data structure and algorithm are explained for the representation and the generation of the three basic urban elements, namely, street network, parcels and buildings. In the case study section, specific requirements of the Empower Shack project concerning our tools are introduced. In the results and discussion section, the generated urban layouts are presented and current problems and limitations, received from the stakeholders concerning the data structure are discussed. In the conclusion and future work section, the paper is summarized and concluded, and research and development of the tool in next phase are proposed.

2. Methods

In our study, urban features have been simplified to four basic elements, namely, streets, blocks, parcels and buildings. For the representation of the street network, the instruction tree (Koenig et al. 2013) is employed, which has been implemented and tested in earlier versions of CPlan (Koenig 2015) and will not be introduced in detail in this paper. For blocks, they are nothing else but enclosed street segments. Buildings, on another hand, are independent on the generation of the street network and blocks because they are just polygons placed inside the parcels. Obviously, we lack one data representation of the parcels as the bridging feature between street network (blocks) and buildings. Therefore, in this section, the data structure for the parcels are majorly introduced.

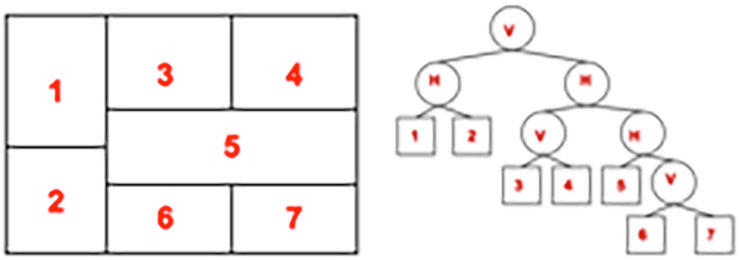

Figure 1. The slicing tree structure. On the left side, it is the geometric representation of the sliced parcels. On the right side, it is the tree representation of the sliced parcels. 'V' indicates vertical slicing whereas 'H' horizontal. Indices of the parcels on the left are corresponding to the ones on the right.

For the representation of the parcels, the subdivision method with slicing tree data structure is employed due to its efficiency to be created and searched (Koenig & Knecht. 2014), for example, for nearest neighbor queries. The data structure is relatively straight-forward to understand as can be seen in figure 1. On the left side, it is the geometric representation of the parcels sliced within a block. On the right-hand side, it is the tree representation of the parcels where the rules to slice are encoded in the nodes. The whole slicing process is depicted step by step as in figure

2. This is a recursive process and the program stops slicing after reaching certain threshold. From the paper of Koenig & Knecht. (2014), the stopping criteria is the specified threshold for the area of the individual leaf (in their cases, rooms for floorplan). In this case, the width of the parcel is more important according to the stakeholders. Therefore, the stopping criteria is the threshold for the width of the parcels, which can be interpreted as the minimum width of the parcels.

In the urban layout generation process, the slicing is only performed on the "street" to ensure that all the parcels are along the streets. One example is as in figure 1, parcel 5 is longer than the others but not sliced further because there is only one edge on the street, whose length is not long enough for further slicing. In reality, the shapes of the blocks are usually more complicated than those in figure 1 and figure 2 and the slicing might not work consistently. To deal with this problem, Koenig & Knecht (2014) sliced vertically or horizontally on the rectangular bounding box of the space as in the left depiction of figure 3. However, it also generates parcels with irregular shapes, which are not suitable for urban design. Instead, in our implementation, the minimum bounding box is calculated and slicing is performed on the minimum bounding box as can be seen in the right depiction of figure 3. Moreover, to keep the slicing perpendicular to the street segment, the shapes are rotated before being sliced as can be seen in figure 4. After being sliced, the shapes are rotated back to their original positions.

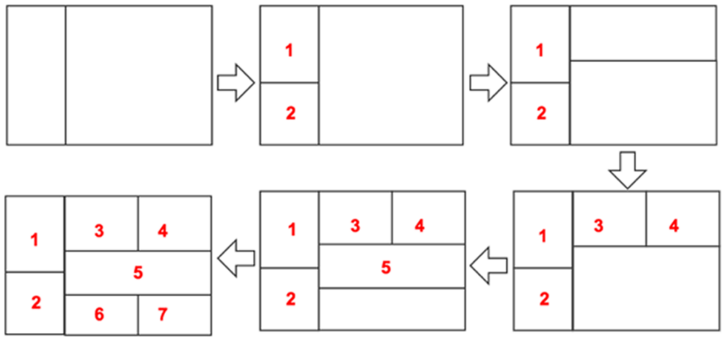

Figure 2. The slicing process. Indices in red are the parcel indices corresponding to the ones in figure 1.

For the development of the components in Grasshopper, each step in the generative process is implemented as one component, including the generation of street network, extracting blocks from the street network, slicing the blocks into parcels, and placing buildings in the parcels as can be seen in figure 5. Grasshopper is currently used as an interface for the CPlan framework, given its popularity among urban designers and architects and the existing abundant and versatile add-ons. This tool is adapted to the case study of the Empower Shack project which will be introduced in the following section.

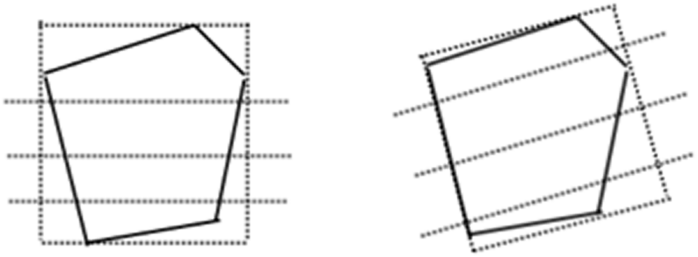

Figure 3. Slicing on the boundary box of irregular shapes. On the left, the slicing is performed on the normal bounding box whereas on the right, the slicing is on the minimum bounding box.

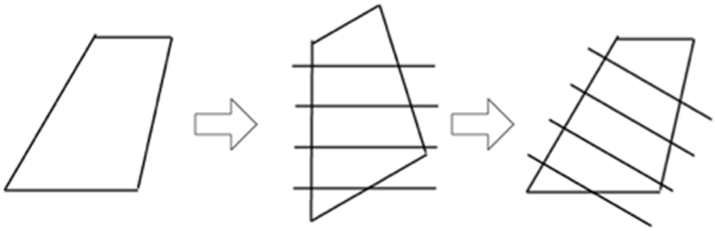

Figure 4. Slicing on the rotated shape so that the slicing lines are always perpendicular to the longest street.

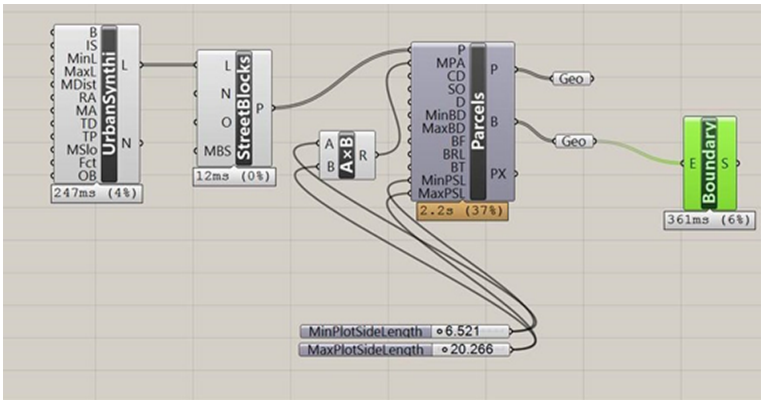

Figure 5. The Grasshopper components for street network, street blocks, parcels and buildings.

3. Case Study

Our case study was conducted within the Empower Shack project, which is led by the Urban ThinkTank (UTT) group (2016). One important aim of the project was to develop a layout of streets and parcels that would allow a maximum density for the two-story houses, which densify an area that was used for informal one story buildings. The study area is Enkanini in Cape Town City, South Africa as can be seen in figure 6. Our methods and tools provide practical avenues for the planners to obtain prototyping suggestions in a fast and interactive way. Moreover, the generated urban fabric could be used as an input for various simulations measuring its performance concerning, for example, security, fire protection, and traffic potential.

Figure 6. The study area selected from the Empower Shack project.

The requirements from UTT and the stakeholders were defined as follows: the tool should be able to fit as many regular shaped (close to rectangles but not necessarily to be) parcels as possible in the study area; the streets should be possible to be moved manually so that the generation of the parcels could adapt automatically; the dimensions of the parcels should be able to be specified and generated precisely; boundaries can be specified and distinguished from street segments; and the Grasshopper interface can be connected to a web interface. With our method, we were able to fulfill these requirements and generate layouts as in figure 7. For the generation of such layouts, the user only needs to specify the dimensions of the individual parcel (width and length), which is simple for the urban designers to use. Moreover, further spatial analysis can be performed based on the generated urban layouts and displayed on web, as the example in figure 8 shows.

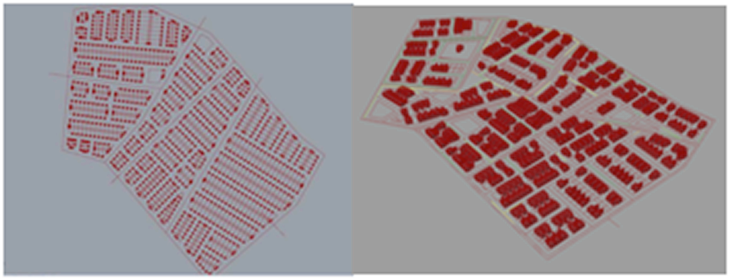

Figure 7. The urban layout for the sliced parcels and buildings within the parcels.

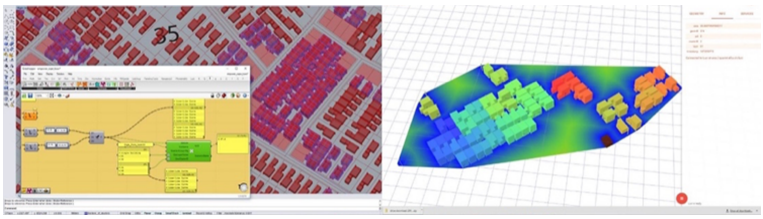

Figure 8. Different variations for synthesized neighbourhoods in Cape Town using a parametric modelling approach. The colours in the layouts show a distance analysis. In this case the area that can be seen from one location is encoded by the colours (blue nearest distance area, green maximum distance). The results can be visualised in a web browser.

4. Discussion and Future Work

From the results shown in the case study, the requirements from the Empower Shack project can be fulfilled with our tool. The urban layouts can be automatically generated based on simple parameters, namely, width and length of the individual parcel, specified by the users. Moreover, the urban designers can interactively revise the urban layout by moving the street segments to achieve satisfying results.

However, there are also limitations of the method. During the study and implementation, three major issues were discovered and partly solved. The first issue was the limitation of the slicing tree data structure for the generation of the parcels. Given the top-down recursive nature of the structure, the generated parcels cannot be guaranteed to be with the precise width and length specified by the users. The slicing stops when the threshold was reached and thus parcels with wider width were generated occasionally. This problem was solved by pre-partitioning the blocks. A slicing tree was still used for the generation of the parcels. However, instead of slicing an irregular block at once, the block was firstly partitioned into several small blocks with regular shapes. The number of the polygons after the partition can be calculated by solving equation 1 below, which is the number of "a" that is equal 1. For clarification, one example is illustrated in figure 9. As can be seen from the figure, the length of the longest edge is 20 m and the width of the parcel is specified as 2 m. To get precise solutions, the edge is firstly partitioned

into two polygons. After the partition, there are two slicing trees that can be used for slicing the region: one is with depth of 3, whereas the other one with depth of 1. However, there are two problems with respect to this method. The first one is that the parcels can only have integer width. The second problem is that it cannot be used to specify precisely the length of the parcel.

$$\lfloor L \rfloor = \sum_{i=0}^{n} a \cdot \lfloor W + 1 \rfloor \cdot 2^n; n = \lfloor \log_2 L \rfloor; a = 0 \quad \text{or} \quad 1 \tag{1}$$

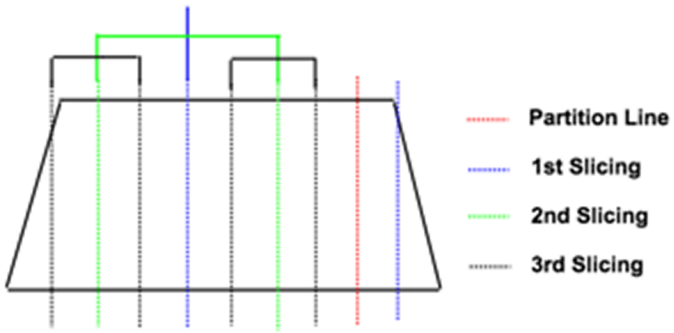

Figure 9. The adapted slicing tree structure. The slicing is based on the bottom edge (longest) with a length of 20 meters. The width of the parcel is specified as 2 meters.

To deal with the above-mentioned issue, a nested method was introduced. In Grasshopper, the blocks were first partitioned into smaller blocks with defined width and then partitioned again into parcels. Although the above-mentioned method could be used to solve the problem, it shows that in reality, the top-down hierarchical slicing structure is not the ideal structure for urban design, especially that it can only be used to fulfill global parameters rather than local ones.

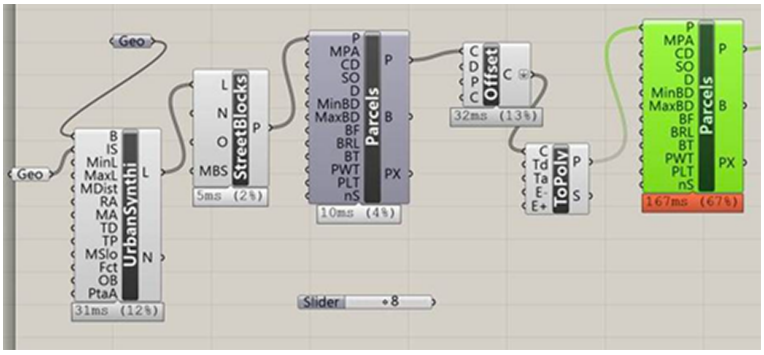

Figure 10. An illustration of the nested method. Blocks are firstly partitioned into smaller ones with the slicing tree and then partitioned into parcels.

Another issue is about the growth of the street network. Given the hierarchical nature of these urban features, the generated solution of the parcels is very much dependent on the generated result of the street network. This leads to a lot of manual revision during the interaction process which spoils the user experience. The ideal solution will be a new data representation for the parcel that could have two characteristics. The first one is that it can take the local specifications into account, say, parameters for individual parcels. The second one is that it can grow with the street network and have mutual influence with each other so that the generated street network can better fit the parcels.

For urban synthesis, Evolutionary Multi-objective Optimization has been proved to be effective (Koenig et al. 2014). However, with the current data structure, the optimization can only be performed hierarchically one urban feature after another, which cannot guarantee the Pareto efficiency for all (Kanbur 2005). A new data structure is required to abstract the geometric representation of the parcels into genotypes so that all urban features can be optimized all together. Moreover, a backcasting strategy (Koenig & Schmitt 2016) can be employed for the next generation tool of computational urban design.

5. Conclusion

From our case study, it has been proved that our methods can efficiently and effectively assist the urban planning process in practice. Planners can benefit from the interactive fast prototyping which saves a lot of manual work and improves the planning efficiency. It has been found that interaction between urban designers and computer can not only save urban designers' troubles, but also reveal the problems concerning the computer programs themselves, which is inspiring for the computational scientists and software developers.

However, it has also been found in practice, that the data representation for the parcels has its own limitations to fully fulfill the needs of urban designers. In the Empower Shack project, the problem lies in the ability to precisely represent the parcels with specified dimensions. Although remedies have been proposed and implemented as intermediate solutions, a new data structure is required to solve the problem fundamentally.

In future, abstract representations as inputs for optimization methods and evaluation strategies should be developed to better fulfill the urban synthesis. It is also a future expectation for the computer to learn from the urban designers and from their revised urban layouts so that better urban layout solutions can be generated.

References

"Empower Shack Project" : 2016. Available from Urban ThinkTank<http://u-tt.com/project/empower-shack/> (accessed 25th December 2016).

"Urban ThinkTank" : 2016. Available from Urban ThinkTank<http://u-tt.com/> (accessed 25th December 2016).

Bolognini, F., Shea, K. and Seshia, A.: 2011, Advanced applications of a computational design synthesis method, *DS 68-9: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 9: Design Methods and Tools pt. 1*, Lyngby/Copenhagen, Denmark, 15-19.

Bringmann, O., Ecker, W., Gerstlauer, A., Goyal, A., Mueller-Gritschneder, D., Sasidharan, P. and Singh, S.: 2015, The next generation of virtual prototyping: Ultra-fast yet accurate simulation of HW/SW systems, *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* , 1698-1707.

Kanbur, R.: 2005, *Pareto's revenge*, Cornell University.

Koenig, R.: 2016, Backcasting and a New Way of Command in Computational Design, *CAADence in Architecture – Proceedings of the International Conference on Computer Aided Architectural Design*, Budapest.

Koenig, R. and Knecht, K.: 2014, Comparing two evolutionary algorithm based methods for layout generation: Dense packing versus subdivision, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **28**(3), 285-299.

Koenig, R. 2015, CPlan: An Open Source Library for Computational Analysis and Synthesis, *in* B. Martens, G. Wurzer, G. Lorenz and R. Schaffranek (eds.), *Real Time- Proceedings of the 33rd eCAADe Conference*, Vienna: Vienna University of Technology, 245-250.

Koenig, R., Standfest, M. and Schmitt, G.: 2014, Evolutionary multi-criteria optimization for building layout planning-Exemplary application based on the PSSA framework., *Proceedings of the 32nd eCAADe Conference - Volume 2*, Newcastle , 567-574.

Koenig, R., Treyer, L. and Schmitt, G.: 2013, Graphical smalltalk with my optimization system for urban planning tasks, *eCAADe 2013: Computation and Performance–Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe*, Delft.

Lai, M. and Wong, D.: 2001, Slicing tree is a complete floorplan representation, *Proceedings of the conference on Design, automation and test in Europe*, 228-232.

Malik, S.M., Lin, J. and Goldenberg, A. A.: 2006, Virtual prototyping for conceptual design of a tracked mobile robot, *2006 Canadian Conference on Electrical and Computer Engineering* , 2349-2352.

Reinhardt, S.K., Hill, M.D., Larus, J.R., Lebeck, A.R., Lewis, J.C. and Wood, D.A.: 1993, The Wisconsin Wind Tunnel: virtual prototyping of parallel computers, *ACM*, 48-60.

Rix, J., Haas, S. and Teixeira, J.: 2016, *Virtual prototyping: Virtual environments and the product design process*, Springer.

Song, P., Krovi, V., Kumar, V. and Mahoney, R.: 1999, Design and virtual prototyping of human-worn manipulation devices., *Proceedings of the 1999 ASME Design Technical Conference and Computers in Engineering Conference* , 11-15.

Wang, G.G.: 2002, Definition and review of virtual prototyping, *Journal of Computing and Information Science in engineering*, **2**(3), 232-236.