Parametric Modelling with GIS

Patrick Janssen[1], Rudi Stouffs[2], Akshata Mohanty[3], Elvira Tan[4], Ruize Li[5]
[1,2,3,4,5]National University of Singapore
[1]patrick@janssen.name [2]stouffs@nus.edu.sg
[3,4,5]{akshatamohanty|elvira.tan|liruizenus}@gmail.com

Existing urban planning and design systems and workflows do not effectively support a fast iterative design process capable of generating and evaluating large-scale urban models. One of the key issues is the lack of flexibility in workflows to support iterative design generation and performance analyses, and easily integrate into design and planning processes. We present and demonstrate a parametric modelling system, Möbius, that can easily be linked to Geographic Information Systems for creating modular workflows, provides a novel approach for visual programming that integrates associative and imperative programming styles, uses a rich topological data structure that allows custom data attributes to be added to geometric entities at any topological level, and is fully web-based. The demonstration consists of five main stages that alternate between QGIS and Möbius, generating and analysing an urban model reflecting on site conditions and using a library of parametric urban typologies, and uses as a case study an urban design studio project in which the students sketched a set of rules that defined site coverage and building heights based on the proximity to various elements in the design.

Keywords: *generative design, urban planning, Geographic Information Systems, parametric modelling*

INTRODUCTION

Existing urban planning and design systems and workflows do not effectively support a fast iterative design process capable of generating and evaluating large-scale urban models.

One of the key issues is the lack of flexibility in workflows to support iterative design generation and performance analyses, and easily integrate into design and planning processes. Especially when such processes are characterized as dynamic, collaborative, and constrained by time, skills and the availability of systems. While various systems exist to support performance analysis within urban planning and design processes, these tend to be neither simple to use, nor flexible in their ways of usage, nor can they be easily integrated with other systems that may serve other parts of the workflow.

Various systems for parametric urban design already exist. Beirão et al. (2011) present a parametric urban design system that provides automatic feedback on a number of urban indicators and density measures. König (2015) presents an open source library for computational analysis and synthesis, denoted CPlan, which supports the optimization of spatial configurations. However, both systems only consider buildings as rectangular blocks and do not support a differentiation in building typologies. Knecht and König (2012) present a tool that, given a street network, generates building lots and buildings, the latter in one of four types: row buildings, courtyard buildings, ribbon buildings and free-standing blocks. However, the tool supports only one building type at a time and treats other parameters, such as building depth, similarly. As such, it mainly focuses on small developments.

Esri CityEngine (Müller et al. 2016) considers a procedural modelling approach for generating 3D city models. Specifically, it adopts a rule-based approach inspired by shape grammars but using procedural rules. While CityEngine is very comprehensive in terms of modelling different building typologies, including their facades, it primarily targets 3D visualization over analysis and assessment. Additionally, much of the work in using CityEngine to generate 3D cityscapes is manual in nature, developing or selecting rules that apply to specific plots in order to generate the relevant buildings. Finally, though CityEngine supports some analysis, this is fairly limited and the data generated within CityEngine cannot easily be exported back into a 2D GIS environment (besides Esri ArcGIS) for detailed analysis and assessment.

Modular Workflows

Modular workflows that can more easily be adapted by the user are of special interest. Such workflows often rely on various software systems that must be seamlessly connected in order to allow the user to switch back and forth between modelling and analysis. Workflows capable of integrating geographic mapping and parametric modelling systems could enable various rapid iterative urban prototyping methods to be developed. Geographic mapping uses Geographic Information Systems (GIS), such as QGIS [1] and Esri ArcGIS [2]. Parametric modelling is generally supported by Computer Aided Design (CAD) systems together with various plugins, with four popular tools being McNeel Grasshopper [3], Bentley GenerativeComponents [4], Autodesk Dynamo [5], and SideFX Houdini [6].

Geographic Information Systems can be used to integrate geospatial data of various types, to generate 2D maps as a starting point for the urban design exploration process, and to perform various types of 2D analysis, including area calculations, buffer analysis, and network analysis. Parametric modelling systems can be used to generate 2D street layouts and parcel subdivisions, to generate 3D urban massing studies and morphologies, and to perform various types of 3D analysis, including view analysis, solar radiation analysis, daylight analysis, and privacy analysis. Furthermore, both types of modelling systems can be linked to other more advanced simulation programs. For example, Geographic Information Systems can export data for transport simulation programs, and parametric modelling systems can export data for Computational Fluid Dynamic (CFD) simulation programs.

The ability to easily exchange data between Geographic Information Systems and parametric modelling systems could potentially result in an ecosystem of flexible workflows that could be modified and adjusted to tackle a wide variety of urban modelling scenarios. The problem is that, currently, exchanging data between these systems is difficult at best. The main reason for this is differences in how data is represented within these systems.

Modelling with Attribute Data

When comparing the representations used by Geographic Information Systems and parametric modelling systems, many differences can be identified beyond the 2D/3D distinction. For example, parametric modelling systems typically support a wide variety of geometric types including spline-based curves and surfaces. Geographic Information Systems on the other hand have extensive tools sets for working with both raster and vector data, but the geometric types supported by the latter are typically very restricted, often consisting only of three types: points, polylines, and polygons. While such differences are important, for this research the key difference relates to how non-geometric data can be associated with geometric entities in the model.

In Geographic Information Systems, attribute data can be attached to geometric entities such as points, polylines and polygons. This attribute data can represent any arbitrary information that the user wishes to associate with the geometric entities. The ability to work with attribute data is a fundamental feature of geographic mapping and can be very powerful.

The attribute data can be displayed using attributes tables, where each row in the table represents a geometric entity and each column represents an attribute. Users can define attribute values explicitly or by using formulas that calculate values based on the geometry or on other attributes. Attributes are typed, typically being integer numbers, floating point numbers, text strings, or dates. Geometric entities and their associated attributes can be exported in a number of file formats, including Shape files and GeoJSON files.

In contrast, most CAD systems either do not allow attribute data to be attached to geometric entities, or only support it in limited ways. One example is Trimble SketchUp [7], which allows data to be added to groups or components. However, this means that separate groups or components need to be created for every geometric entity that needs to have data attached to it. For large models, this can become very cumbersome.

Parametric modelling systems also do not tend to support attribute data. Instead, this type of data needs to be handled as a separate dataflow, which imposes a significant burden of data management on the user. For example, in Grasshopper, there are various plugins for reading and writing GIS data. An example is the Heron plugin [8], which is able to import GIS data from various sources. The plugin provides a node that can read GIS data, but the output produces disconnected data sets for the geometric entities and the attribute data. Managing these separate data sets and keeping them synchronised can become very complex.

An interesting exception is the Jeneratiff plugin for Grasshopper (Dritsas 2016). This plugin provides a library of various advanced computational design algorithms supported by an infrastructure that allows arbitrary data to be associated with geometric entities. The data is stored in dictionaries, with the possibility of nesting dictionaries inside one another, thereby allowing more complex data structures to be created. However, due to the limitations of the underlying Rhino platform, the association between geometry and data only exists within the Grasshopper environment. Once the model is transferred back to Rhino, geometric entities lose all their data.

One parametric modeller with more direct support of attribute data is Houdini. Furthermore, geometric entities are represented using a topological data structure and attributes can be added at three levels: primitives, vertices and points, in addition to the global level representing all geometries. Primitives are geometric entities such as polylines and polygons. Each primitive is defined by a set of vertices, and each vertex has a position in space that is defined by a point. Attribute data can be added at any of these three levels. This means that a polygon can have different data added to the points, the vertices, and the polygon itself. The attribute data is displayed as tables in a similar way to GIS systems.

The authors have previously developed various parametric GIS workflows using QGIS for geographical mapping and Houdini for parametric modelling [9]. The ability of Houdini to be able to handle attribute data in a similar way to GIS systems allowed data to be easily exchanged. Moreover, the combination of the topological data structure with the attribute data in Houdini also proved to be very powerful, even though this was not supported in QGIS. For the geographic mapping, the flat data structure was found to suffice, but for the parametric modelling the topological data structure was very useful.

Despite these advantages, Houdini is still found to be lacking for more advanced modelling tasks. First, the parametric modelling approach is predominantly a dataflow approach with rather weak support for procedural modelling (Janssen and Stouffs 2015). Second, the topological data structure focuses on individual geometric entities but does not include more complex objects consisting of multiple geometric entities. Therefore, a new type of parametric modelling system was developed, called Möbius [10], which can easily be linked to Geographic Information Systems for creating modular workflows.

Paper Overview

This paper describes the Möbius system and uses a case study to demonstrate how Möbius can support complex parametric GIS workflows. The paper consists of three main parts, with the first part describing the Möbius system, the second part focusing on the case study, and the third part discussing future challenges.

OVERVIEW OF MÖBIUS

Möbius is a web-application that runs in the browser on the client side. Figure 1 shows the Möbius user interface. Compared to existing systems, Möbius offers two key advantages. First, it provides a novel approach for visual programming. Second, it uses a rich topological data structure that supports objects consisting of multiple geometric entities. Each of these are described in more detail below.

A Novel Visual Programming Approach

The parametric modelling approach in Möbius integrates associative and imperative programming styles. The associative style of programming is supported through dataflow programming, where the user constructs networks consisting of nodes and wires. The imperative style of programming is supported through blocks-based programming, where the user constructs procedures by creating sequences of code blocks (Resnick et al. 2009). Each node in the dataflow network has an imperative procedure that is defined using the blocks-based ap-

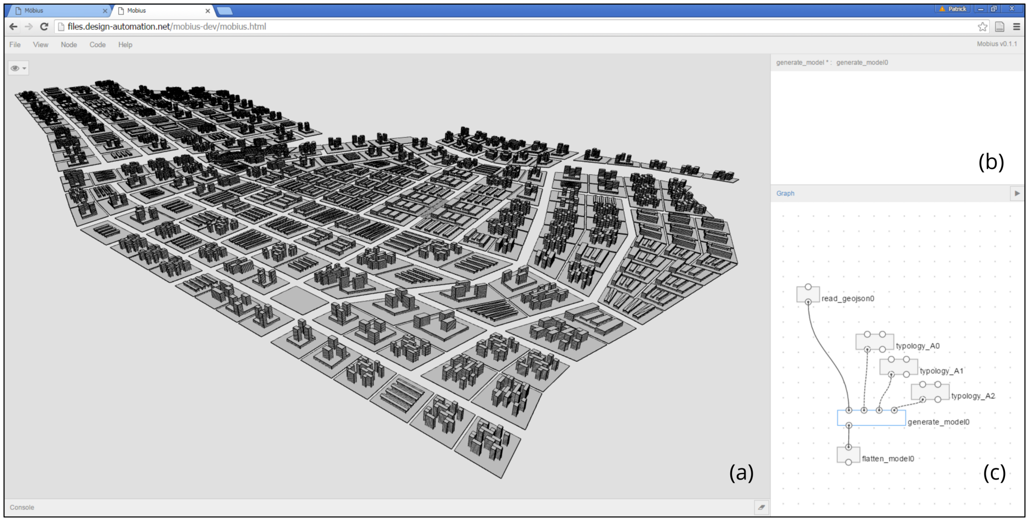


Figure 1
The Möbius interface.

proach. This combined approach allows for the dataflow network to be conceived at a higher level of abstraction, allowing for smaller dataflow networks that can more easily be read and understood.

The node network is defined with the associative programming style, using panes (b) and (c) in Figure 1. Pane (b) is the node parameters pane, and is used to set certain input values for the node instance selected in pane (c). Pane (c) is the network pane, and is used to define the dataflow network. The inputs to a node instance can either be defined by connecting a wire to the input port, or by entering a specific parameter value through the user interface widget. In the example in Figure 1, the *generate_model0* node instance is selected, and the parameters shown are therefore for this node.

When the dataflow network is run, the network is converted into a single JavaScript program that is then executed in one go. This represents a synchronous type of execution, where the nodes are first topologically sorted and then mapped to code in a fixed order. This differs from most other dataflow system, where nodes are executed asynchronously in response to changes to their inputs.

The generation of an execution program from the dataflow network has the advantage that it allows for a wide range of more advanced programming mechanisms to be used. Two such mechanisms are iterative loops and higher-order functions. For a more detailed description of these mechanisms, see Janssen et al. (2016).

Iteration refers to mechanisms for repeating a sub-procedure, and includes for-loops, while-loops and recursive function calls. Visual Programming Language (VPL) systems for parametric modelling typically use a graph-based programming approach consisting of nodes and wires, and as a result do not support iteration very well. Nevertheless, with dataflow (e.g. Grasshopper) and procedural modelling systems (e.g. Houdini), it is possible to create models that incorporate iterative procedures (Janssen and Stouffs 2015). However, creating sophisticated types of iterations in these types of systems (such as nested loops) remains prohibitively difficult (Janssen and Stouffs 2015).

In Möbius, iteration is supported by defining imperative procedures using the blocks-based programming. Each node is an instance of a type, which is defined by specifying a procedure with a set of inputs and outputs. This procedure can include looping control-flow constructs such as *for-loops* and *while-loops*. In Figure 1, the procedure in the *generate_model0* node iterates over all the plots in the GeoJSON file and generates a building on each plot.

Higher-order functions refer to any functions that take another function as an argument or return a function as a result. In parametric modelling, such functions can be very useful as they allow components in the design to be represented and manipulated using functions (Leitão 2014). Most VPL systems do not support higher-order functions. Two recent exceptions are Autodesk Dynamo and Autodesk 3DS MCG [11]. However, the dataflow interface makes it difficult to leverage the full power of higher-order functions.

In Möbius, higher-order functions are supported by an additional output port, which returns a reference to the function that wraps the procedure for that node. When this output is wired into another node, it results in a dashed wire, which indicates to the user that it contains a reference to a function rather than actual data. These wires can be used to provide a function as an input to another downstream node. That node can then have a procedure that executes that function. In the case study, alternative building typologies are conceived as higher-order functions. In this way, selecting a particular building typology only requires the function reference to be provided as input to the generator node, rather than embedding the procedure for the function node into the generation network. For example, in Figure 1, the three 'typology' nodes define procedures that generate buildings with alternative typologies. These are linked as higher-order functions into the *generate_model0* node, which uses these to generate the urban model.

A Topological Data Structure with Attribute Data

Möbius uses a topological data structure consisting of six levels: points, vertices, edges, wires, faces, and objects. Points are similar as in Houdini, and represent positions in space. Geometric entities are then defined using the various topological levels. For example, a polygon is a face defined by a closed wire, a wire is defined by a series of edges, an edge is defined by two vertices, and a vertex is defined by a point. These topological levels are similar to those found in the OpenCascade modelling kernel [12].

Custom data attributes can be added to the geometric entities at any of the topological levels. When creating procedures, users are able to use functions that directly read and write attribute data at specific entries in the topological hierarchy.

For example, a polygon representing an urban plot may have some face attributes that define the target site coverage and building height. In addition, one of the edges of that polygon may also have attribute data that specify that the edge is adjacent to a road. This allows the parametric procedure to orient the buildings that are generated on that plot, so as to ensure that they are facing the appropriate direction relative to the road.

CASE STUDY

The case study stems from an international, collaborative design studio (Winter School) involving over 170 students and 30 design tutors, organized by the International Forum on Urbanism (IFoU). Participants from all over the world came together to work for 12 days on proposals for the Jurong Industrial Estate (JIE), a 5000 ha industrial area in the west of Singapore. The brief was to develop proposals for the transformation of JIE from an almost monofunctional, segregated and fragmented, polluted industrial area into a major catchment area for future population growth that integrates clean(ed) industrial plants with green lungs, attractive housing and vibrant urbanity for one million people.

Some of the results of the IFoU Winter School were further developed in an urban planning studio within the Master of Urban Planning program at the Department of Architecture, National University of Singapore. The case study focuses on one of these projects, called Ecotopia. As part of the proposal, the students developed a set of rules that defined urban parameters based on the proximity to various elements in the design.

The site was divided into three zones as shown in Figure 2: the residential zone, the industrial zone, and the commercial zone. A set of rules was then diagrammed, as shown in Figure 3. The rules define how site coverage and building heights vary in relationship to roads (classes A, B and C), parks and other boundaries. The leftmost rules apply to the residential zone, the second set of rules to the industrial zone, and the third to the commercial zone. The rightmost rules apply to all zones and relate to the proximity to parks.

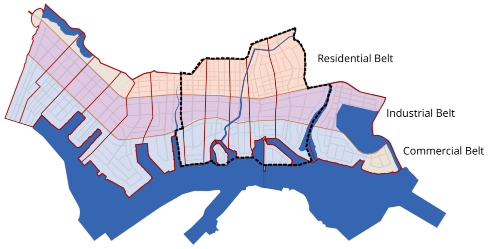
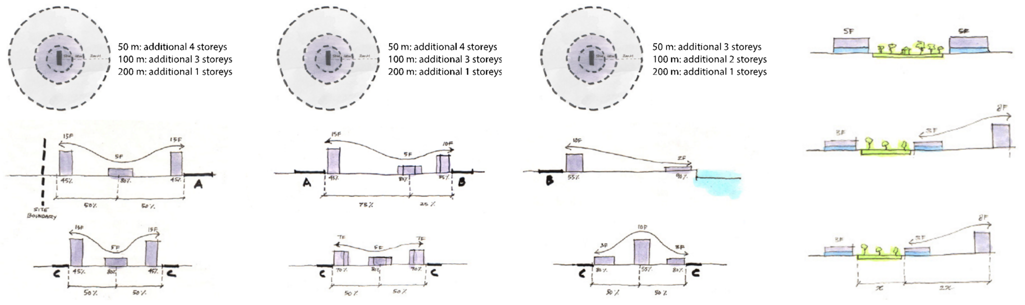


Figure 2
Division of the site into three zones. The case study focuses on the area indicated by the dotted line.

In the proposed parametric GIS workflow, QGIS is linked to Möbius in order to support fast iterative generation and evaluation of large-scale urban models. The workflow uses parameter fields and relies heavily on the ability to attach attribute data to geometric entities in the model in both QGIS and Möbius. The workflow consists of five main stages that alternate between QGIS and Möbius, with data exchange using the GeoJSON file format.

Stage 1: Geographic Mapping in QGIS

QGIS is used to create a map of the site area, including existing buildings and infrastructure. Geographic data is collected and manually integrated into a single geospatial dataset. A series of large parcels are


50 m: additional 4 storeys
100 m: additional 3 storeys
200 m: additional 1 storeys
50 m: additional 4 storeys
100 m: additional 3 storeys
200 m: additional 1 storeys
50 m: additional 3 storeys
100 m: additional 2 storeys
200 m: additional 1 storeys

(a) Residential (b) Industrial (c) Commercial (d) All Zones

Figure 3
Ecotopia sketched rules.

defined in QGIS, indicating the areas of the site where future development is proposed. The map is shown in Figure 4. The parcels are exported from QGIS and imported into Möbius.

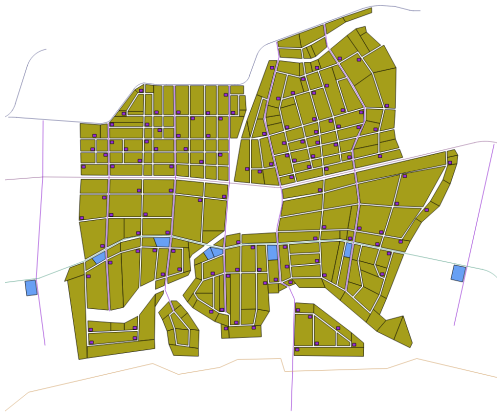


Figure 4
A map of the area for future development in QGIS. The shaded area indicates the parcels where road networks and building typologies need to be developed.

Stage 2: Parcel Subdivision in Möbius

Möbius is used to recursively subdivide each parcel into similar size plots, with tertiary roads also inserted between the plots. The subdivision is performed by a parametric procedure that attempts to create plots that are as evenly sized as possible. The parameters allow the designer to specify the target size of the plot. The transformation from large parcels to smaller plots is shown in Figure 5. The plots and tertiary roads are exported as a GeoJSON file and imported back into QGIS.

Stage 3: Parameter Generation in QGIS

QGIS is used to create parameter fields through a combination of proximity functions and custom formulas that capture the spatial rules shown in Figure 3. A series of additional attributes are defined in the attribute table for the plots for this purpose, in three steps. First, proximity attributes are created that calculate proximity to roads, parks, and the waterfront. Second, parameter attributes are created that calculate the building height and plot coverage for each of the different rules shown in Figure 3 based on the proximity attributes. This results in each plot being assigned two building heights and plot coverages. Third, a final pair of attributes is created that calculates the final value for both parameters using formulas that give priority to certain rules or conditions. The plots with the attributes are exported from QGIS and imported into Möbius.

Stage 4: Urban Model Generation in Möbius

Möbius is used to generate urban models using a library of parametric urban typologies. The parameters for each plot are extracted from the attributes attached to the plot polygon; namely the building height and the plot coverage. These parameters can be viewed in Möbius in an attribute table, as shown in Figure 6. The urban model is then generated by se-

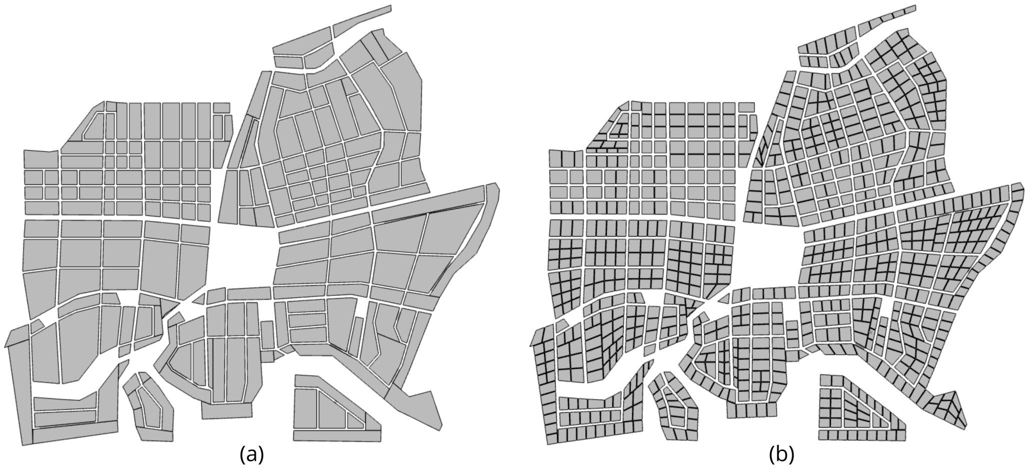
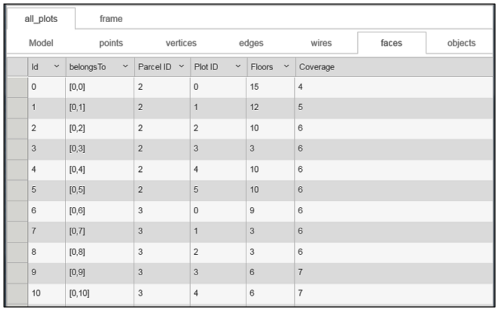


Figure 5
The process of subdividing large parcels into smaller plots in Möbius. (a) The large parcels. (b) The subdivided plots.



Id	belongsTo	Parcel ID	Plot ID	Floors	Coverage
0	[0,0]	2	0	15	4
1	[0,1]	2	1	12	5
2	[0,2]	2	2	10	6
3	[0,3]	2	3	3	6
4	[0,4]	2	4	10	6
5	[0,5]	2	5	10	6
6	[0,6]	3	0	9	6
7	[0,7]	3	1	3	6
8	[0,8]	3	2	3	6
9	[0,9]	3	3	6	7
10	[0,10]	3	4	6	7

Figure 6
The attribute table for the plots in Möbius.

lecting and instantiating a parametric urban typology on each plot, as shown in Figure 7(a).

Each plot together with the urban model on that plot is treated as a single object. The polygons in that object have a 'type' attribute that defines the types of

building elements that they represent, such as *plot*, *footprint*, *floor*, *wall*, *roof*, and *window*. Floors are further categorised into three types: *residential*, *commercial* and *industrial*. Möbius is then used to calculate the floor areas for each of the different functions. These values are then added as attributes to the footprint polygons. For example, if a plot has two buildings on it, then it will have two footprint polygons. For each building, the floor areas will be calculated and added as attributes to the respective footprint polygon.

Finally, in order to be able to export the model back to QGIS, the model needs to be flattened to 2D. The critical information that needs to be transferred is the building footprints together with the floor area attributes. These attributes are important as they will be used for the QGIS analysis in the next stage. Möbius exports the building footprints as a 2D model which can then be imported back into QGIS. Figure 7(b) shows the resulting 2D model, and Figure 8 shows the attribute table for the plots and building footprints.

Stage 5: Spatial Analysis in QGIS

QGIS is used to analyse the flattened map of the urban model. First, a set of additional attributes are added to the building footprint polygons representing the number of people in each building for each of the functions. These values are calculated using a formula that divides the floor area for each function by average 'area-per-person' values. The attributes are then used to calculate the percentage of people within a certain walking distance of transport nodes. This is performed using a simple buffer analysis, which is based on the straight-line distance from

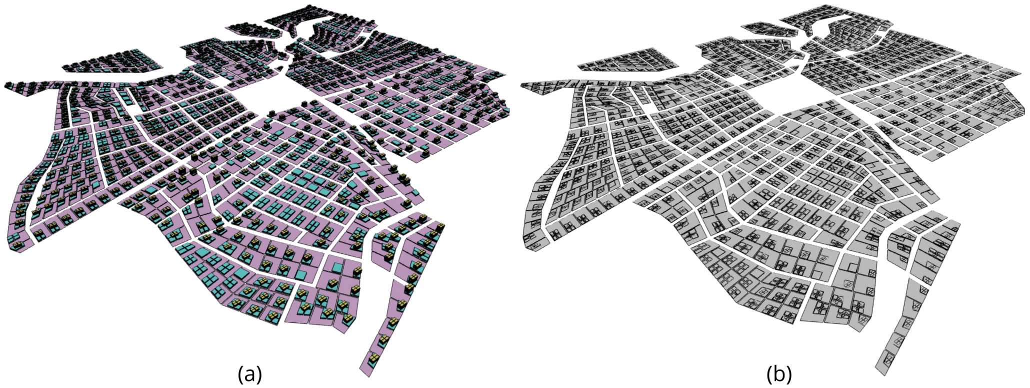
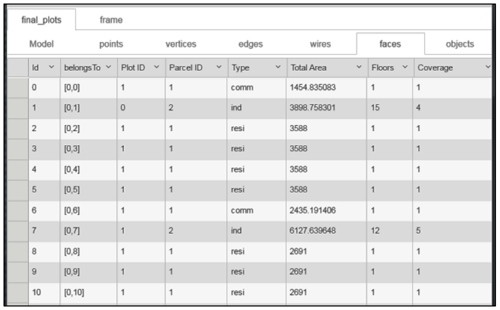


Figure 7
The models in Möbius. (a) The 3D urban model. (b) The 2D flattened model, for export to QGIS.

the centre of the building footprint to the closest transport node. Population densities are also visualised by colouring the polygons. Figure 9 shows the resulting map in QGIS.



Id	belongsTo	Plot ID	Parcel ID	Type	Total Area	Floors	Coverage
0	[0,0]	1	1	comm	1454.835083	1	1
1	[0,1]	0	2	ind	3898.758301	15	4
2	[0,2]	1	1	resi	3588	1	1
3	[0,3]	1	1	resi	3588	1	1
4	[0,4]	1	1	resi	3588	1	1
5	[0,5]	1	1	resi	3588	1	1
6	[0,6]	1	1	comm	2435.191406	1	1
7	[0,7]	1	2	ind	6127.639648	12	5
8	[0,8]	1	1	resi	2691	1	1
9	[0,9]	1	1	resi	2691	1	1
10	[0,10]	1	1	resi	2691	1	1

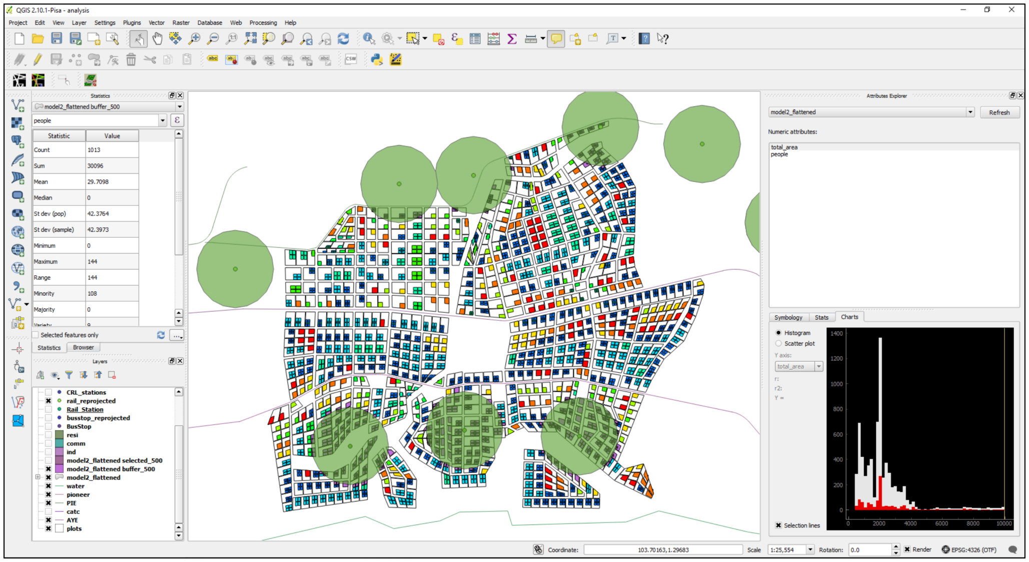
Figure 8
The attribute table for the building footprints in Möbius.

CONCLUSIONS AND FUTURE RESEARCH

A parametric modelling system called Möbius has been described that supports smooth exchange of data with Geographic Information Systems. In particular, Möbius provides a novel approach for visual programming that integrates associative and imperative programming styles and uses a rich topological data structure allowing custom data attributes to be integrated with geometric entities at any topological level. This facilitates the creation of flexible workflows that can be modified and adjusted to tackle a wide variety of urban modelling scenarios.

The case study demonstrates how Möbius can support complex parametric GIS workflow. In the case study, a workflow is developed that switches between the Geographic Information System and the parametric modelling system multiple times, using each tool for what it is best at, mainly analysis and generation, respectively.

Through this research, some potential benefits of rule-based modelling have started to emerge. Future research will explore the integration of a rule or grammar-based data synthesis method within the visual programming approach. This will require the conception of a relevant description model for the specification, selection and execution of grammar rules within Möbius. This will further improve workflow flexibility allowing users to choose whether to express rules, e.g. for determining building height and plot coverage from the proximity attributes, within Möbius or the Geographic Information System, as procedural expressions, higher-order functions, or grammar rules. Ideally, workflow flexibility should allow users to select and develop their own preferred workflow in support of existing design processes and preferential working methods.



model2_flattened buffer_500

people

Statistic	Value
Count	1013
Sum	30096
Mean	29.7098
Median	0
St dev (pop)	42.3764
St dev (sample)	42.3973
Minimum	0
Maximum	144
Range	144
Minority	108
Majority	0

Layers: CRL_stations, rail_reprojected, Rail_Station, busstop_reprojected, BusStop, resi, comm, ind, model2_flattened selected_500, model2_flattened buffer_500, model2_flattened, water, pioneer, PIE, catc, AYE, plots

Coordinate: 103.70163,1.29683 Scale 1:25,554 Rotation: 0.0 EPSG:4326 (OTF)

Figure 9
The 2D map with the buffer analysis showing the percentage of people within a certain walking distance from transport nodes.

REFERENCES

Beirão, JN, Nourian, P and Mashhoodi, B 2011 'Parametric urban design: An interactive sketching system for shaping neighborhoods', *Proceedings of eCAADe 2011*, Ljubljana, pp. 225-234

Dritsas, S 2016 'An advanced parametric modelling library for architectural and engineering design', *Proceedings of CAADRIA 2016*, Melbourne, pp. 611-620

Janssen, P, Li, R and Mohanty, A 2016 'Möbius: a parametric modeller for the web', *Proceedings of CAADRIA 2016*, Melbourne, p. 157–166

Janssen, P and Stouffs, R 2015 'Types of parametric modelling', *Proceedings of CAADRIA 2015*, Daegu, pp. 157-166

Knecht, K and König, R 2012, 'Automatische Grundstücksumlegung mithilfe von Unterteilungsalgorithmen und typenbasierte Generierung von Stadtstrukturen', *Arbeitspapiere (Working Papers) Informatik in der Architektur*, 15, pp. 3-21

König, R 2015 'CPlan: an open source library for computational analysis and synthesis', *Proceedings of eCAADe 2015*, Vienna, pp. 245-250

Leitão, M 2014 'Improving generative design by combining abstract geometry and higher-order programming', *Proceedings of CAADRIA 2014*, Kyoto, p. 575–584

Müller, P, Wonka, P, Haegler, S, Ulmer, A and van Gool, L 2006, 'Procedural modeling of buildings', *ACM Transactions on Graphics*, 25(3), pp. 614-623

Resnick, M, et al. 2009, 'Scratch: programming for all', *Communications of the ACM*, 52(11), pp. 60-67

[1] http://www.qgis.org/

[2] https://www.arcgis.com

[3] http://www.grasshopper3d.com/

[4] https://www.bentley.com/en/products/product-line/modeling-and-visualization-software/generativecomponents

[5] http://www.autodesk.com/products/dynamo-studio/overview

[6] https://www.sidefx.com/

[7] http://www.sketchup.com/

[8] http://www.food4rhino.com/project/heron

[9] https://gis4design.wordpress.com/

[10] https://gis4design.wordpress.com

[11] http://mobius.design-automation.net/

[12] http://www.opencascade.com/doc/occt-7.0.0/overview/html/technical_overview.html