VISUAL PROGRAMMING FOR GEO-COMPUTATION

Towards Tools for Tool Makers

PATRICK JANSSEN¹, DEREK PUNG² and KIAN WEE CHEN³
¹,²National University of Singapore
³Andlinger Center for Energy and the Environment, Princeton
¹patrick@janssen.name ²derek.pung@u.nus.edu
³chenkianwee@gmail.com

Abstract. The paper presents the Möbius Geospatial Modeller, a web-based visual programming tool developed by the authors for creating interactive 3D geospatial datasets. As an evaluation of the modeller, the results of a five-day workshop are presented. The paper ends with a discussion, proposing the creation of customised visual programming environments that provide users with the ability to create their own customised high-level domain-specific functions, as opposed to trying to create hundreds of functions to cater for all possible use cases.

Keywords. Visual programming; geo-computation; geospatial visualization; dataflow and control flow.

1. Introduction

Geospatial data is abundant. Currently, interrogating and visualising such data is still the domain of experts. However, current technologies are making it possible for citizens from varying backgrounds to build interactive 3D geospatial web mashups to suit their own needs, creating visualisations that others can explore and interrogate. When the curiosity and ingenuity of vast numbers of citizens are unleashed on these datasets, important new discoveries may be able to influence the long-term evolution of cities in novel and unexpected directions.

A number of existing platforms provide support for end users to build geospatial narratives, consisting of web mashups that integrate various geospatial datasets (Baker 2015). An example is Esri’s Story Maps, which allow end users to combine 2D maps with narrative text, images, and multimedia content to tell your story (Strachan 2014). However, a key downside of these existing technologies is that they do not support any type of advanced data generation through procedural geo-computation. Users are limited to manually editing datasets with GIS modelling tools, which makes it difficult to make novel discoveries. Such discoveries would require scripting or software programming skills that are beyond the reach of most citizens (Murgante et al. 2009). In this paper, we explore the use of visual programming as a possible solution.
In computer-aided architectural design, a parametric model is a user-defined algorithm that generates 3D model instances by varying input parameters. Parametric modelling requires the designer to define this algorithm, possibly using some type of visual programming language. Parametric modelling is typically used either for quickly generating design options or for generating a single design with a high-level of complexity.

Despite numerous advantages, designers face difficulty in translating their design ideas into such an algorithm. In response, design schools have introduced computational thinking courses to provide students with the conceptual foundations for overcoming this difficulty. For design students, computational thinking involves idea formulation, algorithm development, and solution exploration, with a focus on the manipulation of geometric and semantic datasets. Through this process, students also become familiar with higher-level computational concepts, such as decomposition, encapsulation, and abstraction.

Visual programming is a useful method for learning and practising computational thinking. Visual programming is usually based on either of the two approaches: flow-based or blocks-based programming. Flow-based programming gives more importance to the state of data at each stage of the program. Blocks-based programming emphasises the order of operations in a program and is process-oriented. The former is usually visually represented as a network of nodes and wires, the latter as code blocks sequences stacked from top to bottom.

Parametric modelling can be categorised into four types: object-based modelling, associative modelling, dataflow modelling, and procedural modelling (Janssen & Stouffs, 2015). The latter are the more advanced, offering greater control and flexibility. Examples of such dataflow and procedural systems include Bentley GenerativeComponents (GC), McNeel Grasshopper, Autodesk Dynamo, and Sidefx Houdini. These systems all use a visual programming approach, where the designer defines algorithms by building networks of nodes and wires (Janssen & Chen 2011).

Numerous researchers have argued that the limited ability of flow-based visual programming to capture control-flow logic means that it is unable to fully support computational thinking (Celani & Vaz, 2012; Leitão, Santos, & Lopes, 2012). On the other hand, blocks-based visual programming has achieved considerable success in helping even elementary and high-school students do simple programming tasks (Brennan & Resnick, 2012; Weintrop & Wilensky, 2017). However, no existing parametric modelling tools have used the blocks-based approach.

We hypothesise that both flow-based and blocks-based approach can complement each other. We propose that the flow-based approach can be used at a higher level, to visually define the overall structure of the algorithm. At the lower level, inside each node, the algorithmic procedure is defined using the blocks-based approach. In this integrated approach, we disallow any control-flow logic at the flow-based level.
2. Möbius Modeller

A web-based parametric modelling platform called Möbius Modeller has been implemented that integrates these two visual programming approaches within a single language (Janssen, Li, & Mohanty 2016; Janssen, 2014). Previous research has presented broad benefits of this approach, the primary one being the significant reduction in the number of dataflow nodes required to achieve a particular modelling task (Janssen, Li, & Mohanty 2016). As a result, the complexity of the visual program can remain the commensurate with the complexity of the modelling task, even when highly complex models are created.

Our ongoing research aims to investigate to what extent the Möbius visual programming language can support computational thinking concepts required for constructing complex parametric models.

In Möbius, the flow-based approach is implemented as a flowchart of nodes with links. Inside each node in the flowchart, the blocks-based approach is then used to visually create a procedure. The user builds the procedure by inserting control-flow statement and functions from a library. The user interface is shown in Figure 1.

![Figure 1. An example of a Möbius flowchart (left) and procedure (right).](image)

Version 0.1 of Möbius was developed in 2015. After a year of further testing and evaluation, version 0.2 was developed in 2017, integrating a wide range of improvements. This second version was then used, between January and April 2018, for teaching a 12-week module called “Spatial Computational Thinking”, for 135 second-year students in the undergraduate Architecture course at National University of Singapore. The module was designed to develop capabilities in spatial computational thinking, in order to help students translate their design ideas into executable algorithms.

Following the experience of teaching Möbius, version 0.3 was developed, this time focusing on geospatial modelling. This version allows users to code geo-computational procedures for manipulating and visualising geospatial data. It integrated Cesium.js as a viewer, and the turf.js modelling library. One of the key features introduced in this third version was the ability to publish models as interactive web pages that others could visit and explore. The aim was to allow...
students to develop specific data visualisations and narratives and to then share these with a wide audience.

This paper focuses on this third version of Möbius, called the Möbius Geospatial Modeller. Version 0.4 of Möbius, currently under development, is discussed in section 3 of this paper.

2.1. GEOSPATIAL WORKSHOP

Möbius Geospatial Modeller was used for a five-day workshop teaching students how to create and publish geospatial narratives and models on the web. The workshop was conducted in collaboration with the Beijing City Lab at Tsinghua University. Students did not have any prior scripting or programming knowledge. During the workshop, students were taught to build geo-computational procedures for generating interactive 3D narratives, consisting of web mashups built on top of diverse open data sets.

The modelling was all performed using the geojson file format, using 2D geometry consisting of three types of entities: points, polylines and polygons. The geometric entities were created as a way of representing and visualising data at particular geospatial locations. Various types of data were then added to the geometric entities by adding attributes. Such attributes consisted of key-value pairs, where the key was the name of the attribute and the value was any type of data. In many cases, student created simple grids of square polygons, with some data attributes capturing how certain types of data varied over a geospatial area.

Once the 2D geojson data had been generated, the visualisation of this data was then automatically mapped to 3D, using the Cesiumjs viewer. This visualisation could be performed interactively by anyone visiting the published website. A customised version of the Cesiumjs viewer was developed that allowed two attributes to visualise. One attribute would be visualised as an extrusion height while the other was visualised as a colour. For example, in the case of the 2D grid of square polygons, each polygon would first be extruded to represent the first attribute, and the colour of the extruded column would then be changed to represent the second attribute. For the colour, a blue-to-red false colour range was used. Attributes could be either discrete or continuous. For both extrusion height and colour, the attributes minimum and maximum values could be set.

This meant that a visitor to the website could play with the settings to explore the data in more depth. Initially, on opening the website, the default settings defined by the student would typically display two specific attributes. But the student could also have added other data sets to the geometric entities. The visitor to the website would then be able to change the data that was being visualised, and to play with the minimum and maximum data ranges to discover other data patterns and relationships.

For the workshop, everything was completed in only five days. The first day focused on learning basic programming skills, the second day on searching for and manipulating geospatial data, and the third day on learning a variety of geo-processing functions available in turf.js. Finally, during the last two days, the students developed geospatial narratives focusing on specific issues and then...
collected a variety of geospatial data sets from open online sources.

In total 16 students attended the workshop. Students worked either individually or in small groups, developing 13 different geospatial narratives and associated data visualisations. The narratives engaged with a variety of different issues. In order to give a sense of the diversity of narratives and data visualisations, some typical topics were as follows:

- Tourist density versus rates of consumption.
- Air quality versus GDP.
- Building height versus open space ratio.
- Level of urbanisation versus mental health.
- Food rating versus food price.
- Subway versus bus accessibility.
- Salaries versus house prices.
- Urban demolition versus urban life.
- Land price versus population density.

In most cases, the process of manipulating the geospatial data was decoupled from the process of visualising that data. This was mainly done in order to make the visualisation process faster and more interactive. For the data manipulation, one or more Möbius models were created for analysing, mapping, and merging various existing geospatial datasets. The resulting data was then saved as a reduced geojson dataset optimised for visualisation. For the visualisation, a second Möbius model was then created, which was then used for publishing the interactive website.

2.2. SINGAPORE NARRATIVE

As an example of one of the narratives, an investigation focusing on Singapore and developed by the authors will be described in more detail. The model originally created for this narrative will then be analysed in more detail, and various alternative approaches and improvements to the workshop workflows and the process will then be discussed.

The Singapore narrative investigated the relationship between the population density around the island in relation to two factors: hotspots for dengue fever outbreaks, and distance to the Mass Rapid Transit (MRT) stations. The data for the visualisations were downloaded from a government data portal website, data.gov.sg.

For the data manipulation part, one Möbius model was created consisting of three nodes, as shown in Figure 1. First, two relatively simple nodes were created for extracting the centroids of the dengue hotspots and MRT station locations. Figure 1 shows the procedure for the dengue hotspots. In both cases, the original data was downloaded as geospatial data sets from the government portal. A third node was then used to perform the main processing of the data.

This third node creates an analysis grid, consisting of 50m x 50m square polygons covering the whole of Singapore. The population density was then read from another geojson data source and mapped onto the analysis grid. It then performs various distance calculations between points in the analysis grid and the
centroids of the dengue hotspots and MRT stations. All the resulting data is saved as attributes attached to the polygons in the analysis grid. A view of the developer version of the Möbius model showing the relationship between population density and dengue hotspots is shown in Figure 2.

Finally, a second model is created for the data visualisation. As already mentioned, this visualisation could have been created directly in the first model. However, this would have made the visualisation very slow, and as a result, it was decided to split visualisation into its own model. The visualisation of the data as a published interactive website is shown in Figure 3. All the complexity of the flowchart and the procedures are hidden from the website visitor.

3. Critical Analysis

A critical analysis of the workshop results highlighted two issues. The first issue will only be briefly discussed as it is not the focus of this paper. It relates to the decoupling of the data manipulation models from the data visualisation models. This creates a significant problem, in that the data that is displayed in the published interactive web visualisation may quickly become outdated. A simple solution is being investigated for this, in which the data manipulation models are uploaded into the cloud and executed as a scheduled task (a ‘cron’ job). This can be achieved using an online cloud platform such as Travis CI.

This paper focuses on the second issue. Despite the effort to develop user-friendly visual programming tools, creating geo-computational procedures remained difficult for students with no programming experience. Two key challenges were identified. First, the number of functions provided to the students was found to be overwhelming and confusing. In total, the modeller included
160 functions, most of which were geoprocessing functions from the *turfjs* library. Second, the blocks-based procedures that the students created quickly became very long. These procedures became difficult to understand and maintain, especially when they included multiple nested loops.

An analysis of the models showed that in most cases, the models included only a small number of nodes but with long procedures. In addition, the procedures only used a very small subset of the geoprocessing functions available in the library. The third data manipulation model for the Singapore narrative was fairly typical. It consisted of just three nodes, with the third node creating the analysis grid and performing all the distance calculations. This node had a procedure 42 lines long, making use of 11 functions.

Further analysis of the student procedures revealed significant commonality in programming patterns. For example, out of the 13 models, 12 made use of an analysis grid. For all these models, significant parts of the code had similar patterns for distances between sets of features. Analysing these patterns in more detail then suggested that the procedures could have been created using just a few high-level functions.

In order to test this, a new set of high-level functions were implemented for the models that required analysis grids. In these high-level functions, the looping behaviour was built into the function itself, further reducing the code that needed to be written. Thus, students would not be required to write blocks-based code to explicitly loop over the collections of entities.

The required number of functions to implement all 12 analysis grid models was found to be just 8 functions. As an example, the Singapore narrative can be considered. For this narrative, just five high-level functions would be required. This would reduce the 42-line procedure to just 12 lines.

For the Singapore model, the five functions consisted of the following:
- `id()` - creates an id for each entity.
- `area()` - calculates the area of an entity and adds it as a new attribute.
- `attribTransfer()` - transfer an attribute from one entity to another.
- `attribMath()` - create a new attribute by performing mathematical calculations with one or more existing attributes.
- `many2manyDistance()` - calculate the distance from every entity in one collection to every entity in another collection, and stores the results in attribute arrays.

Although these functions are more high-level that the previous `turf.js` functions, they nevertheless remain fairly generic. These functions could be used in many different scenarios, even in models that did not necessarily require an analysis grid.

### 3.1. USER-DEFINED FUNCTIONS

It is clear that the number of functions required in order to perform a certain geo-processing task can be massively reduced by providing just a small set of higher-level functions. In the case of the Singapore model, the number of lines of code was reduced by over 71%, from 42 lines to just 12 lines. Such a reduction would make the task of visual programming much easier for a student, for two reasons. First, the number of functions they would have to become familiar with could be far fewer. Second, the blocks-based procedure itself would be much shorter and would require fewer loops, making it easier to understand.

However, all of this does, of course, come at a cost of reduced flexibility. If a visual programming system was created that provided only these 8 functions, then it could only be used for generating visualisations similar to the Singapore model, consisting of analysis grids and performing distance and area calculations. This would be highly limiting.

The ability for end users to define their own customised functions would offer a way out of this dilemma. We may imagine different types of end user. For example, in a scenario such as the workshop, two types of end-user are the teacher and the students. The teacher may be more experienced with visual programming, while the students may be complete novices. In such a case, the teacher can create a set of high-level functions for the students, while at the same time hiding many of the low-level functions. In this way, the students have an easier learning curve. If during the workshop, one of the students requires a different high-level function, then the teacher can implement that function on the spot and add it to Möbius for the students to use. For the students who are eager to explore the lower level functions, the teacher may also show them how those functions can be enabled again. This allows for a flexible approach, with different sets of functions being available to different types of users.

A key challenge with this approach is how to allow the high-level functions to be easily defined by the end users. This functionality is now being implemented in the latest version of Möbius. In the new version, the flowchart includes a `start` node, an `end` node, and zero or more `procedure` nodes. This flowchart can now be mapped to a single function, containing a set of nested functions for each of the `procedure` nodes. With the new functionality, a user is able to import any existing
Möbius file into their current model, and each file is then transformed into a single function. This allows any user to define their own higher-level functions, and to share these functions with others.

As an example, a flowchart could be defined for the `many2manyDistance` function that calculates the distances between all the entities in two collections. The `start` node is used to define a set of global variables which all the procedures can read. In this case, three global variables are defined. The first two variables define the names of the two collections to process. The third variable defines the name of the attribute to create. A number of procedure nodes could then be defined to calculate the distances and create the necessary attributes. When this flowchart is imported into another Möbius model, it will appear as an imported function, called `many2manyDistance()`, with three arguments. The function can then be inserted into any procedure, in exactly the same way as the built-in functions.

This functionality includes the ability to create a description of the model, together with descriptions of all the input parameters. When the model then gets used as a function, these descriptions automatically become the help of the function that other users are able to view. The ability to easily define the help for such user-defined functions is essential if they are to be used by a large number of other users. Eventually, the function may even be shared on online platforms.

Finally, the ability to hide selected low-level functions is being developed. This will be enabled through Möbius configuration settings that can be saved and also shared between users. When working inside Möbius, certain functions can be disabled through the user interface. Once the desired combination of functions has been set, the setup can then be saved as a configuration file which can later be imported into a different Möbius session. The configurations will be saved in the browsers’ local storage, thereby allowing the changes to remain persistent.

4. Conclusions

The research project is developing a visual programming platform and language that combines flow-based and blocks-based visual programming approaches called the Möbius Modeller. The flow-based approach is used at a higher level, to visually define the overall structure of the algorithm. At the lower level, inside each node, the algorithmic procedure is defined using the blocks-based approach.

The paper focuses on the results of a workshop conducted using one version of this platform, the Möbius Geospatial Modeller. This version aims to support geo-computational modelling and visualisation. These models are typically dealing with large data sets, and the data processing requires nested looping and iteration. Through numerous iterations of software development and user testing, the research is trying to discover conceptual frameworks and constructs that will make it easier for end users to develop complex geo-computational models.

A critical analysis of the workshop results highlighted that, despite the visual programming tools, creating geo-computational procedures remained challenging for end-users with no programming experience. One of the key reasons for this was the fact that the modeller provided 160 low-level functions that were often confusing and overwhelming. An initial analysis of the student procedures
revealed significant commonality in programming patterns. Further investigation then revealed that the procedures could have been created with far fewer high-level domain-specific functions. Based on this analysis, the latest version of the Möbius is now being developed, incorporating the ability of end users to define their own high-level functions.

Acknowledgements

We thank Professor Long Ying from the Beijing City Lab at Tsinghua University for his support in running the Extruded Cities workshop. We also thank the workshop participants for their contributions: Fan Zekun, Feng Wenhan, Hou Luyao, Huang Zidong, Li Bayi, Li Xintian, Liao Hui, Liu Jiandie, Liu Shang, Tang Zixiao, Wang Pengyu, Wu Shuyun, Xu Bingjun, Yang Li and Zhang Xinrui. Finally, we thank our team members for helping to run the workshop: Akshata Mohanty, Sun Huiyao, Joie Lim and Wang Likai.

References


Papert, S.e.y.m.o.u.r.: 1980), Mindstorms: Children, computers, and powerful ideas, Basic Books, Inc.
