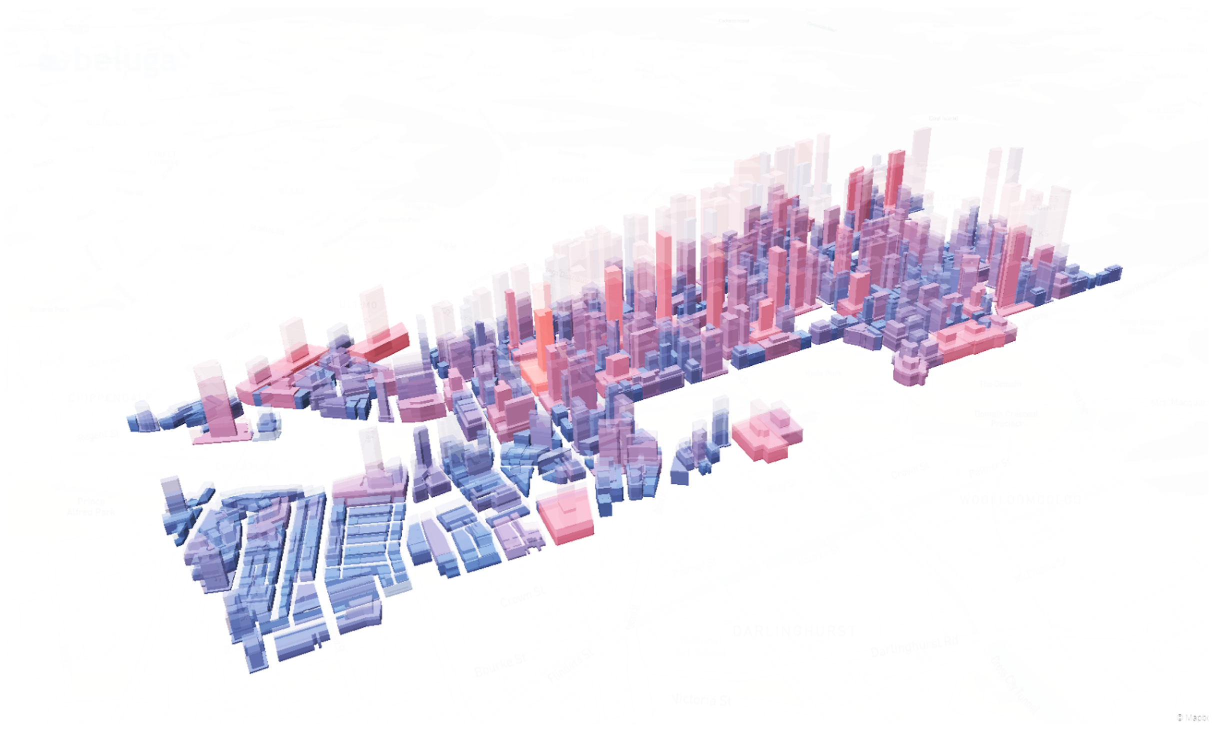
Adjudicating by Algorithm

Scripting building regulations to generate permissible forms



Thesis submitted to the Faculty of the Built Environment in fulfilment of the requirements for the degree of Computational Design (Honours).

Nazmul Khan z5018132

Computational Design (Honours)

University of New South Wales

2017

PLEASE TYPE

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: KHAN

First name: NAZMUL | Other name/s:

Abbreviation for degree as given in the University calendar: Computational Design (Honours)

School: University of New South Wales | Faculty: Built Environment

Title: Adjudicating by Algorithm: Scripting building regulations to generate permissible forms

Abstract 350 words maximum: (PLEASE TYPE)

This paper outlines Law as Code as a concept, and demonstrates how it can be applied to urban planning through several prototype scripts. By doing so, processes such as testing new urban schemes, and preliminary design analysis can become automated. Beluga is presented as an example of converting regulations to code to generate permissible envelopes; Humpback is presented as tool to convert between GIS and computational tools; and Minke presented as a versioning tool. Through these case studies we speculate on how expressing more urban planning laws through computer code might be beneficial to stakeholders in decision making processes.

Supervisors: Dr. M. Hank Haeusler, Ben Doherty and Rob Asher

Examiners: Nicole Gardner and Alessandra Fabbri

SCIENTIA
MANU ET MENTE
UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Originality Statement

‘I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.’

Signed …………………………………………………........

Date ………………………………………………............

List of Publications

Throughout the year of research, the following papers were written and presented at the following conferences:

The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA 2017), Hong Kong

Urban Pinboard *- Establishing a Bi-directional Workflow Between Web-based Platforms and Computational Tools*

Architecture is heading towards a future where data is collected, collated and presented in a dynamic platform. Urban Pinboard is a web-based GIS platform that promises to establish a bi-directional workflow between web data repositories and computational tools. This paper presents a workflow between websites and computational tools to allow users with minimal experience in computational processes to be engaged in the utilisation of these large datasets.

Urban Planning and Property Development Conference (UPPD 2017), Singapore

Humpback *- Introducing and evaluating a GeoJSON constructor tool for Grasshopper*

This paper presents Humpback, a plugin for the computational tool Grasshopper. In our background research we identified that due to limited workflow connection between Geographical Information Systems (GIS) and computational design tools, a gap of informed urban design outcomes exists. Humpback extends Grasshoppers capability to read and write GeoJSON; a geographic data format. This enables a new approach to urban planning by allowing computationally generated forms to be visualised in GIS software. These tools produce iterative, smarter urban solutions which can be shared on a wider platform. This paper introduces and details Humpback's creation, evaluates the tool, as well as its application to Mapbox, a GIS platform.

Urban Design Conference (UD 2017), Surfers Paradise, QLD

Encoding Building Regulations *- Translating Regulations into Code to Inform Better Urban Outcomes*

This paper outlines computational law as a field, and describes how it can be applied to urban design through several examples. We present Dugong; a maximum envelope generation tool. Beluga as an example of converting regulations to code and Minke as a versioning tool. Though these case studies we speculate on how expressing more urban planning laws through computer code might be beneficial to stakeholders in decision making processes.

ABSTRACT

This paper outlines Law as Code as a concept, and demonstrates how it can be applied to urban planning through several prototype scripts. By doing so, processes such as testing new urban schemes, and preliminary design analysis can become automated. Beluga is presented as an example of converting regulations to code to generate permissible envelopes; Humpback is presented as tool to convert between GIS and computational tools; and Minke presented as a versioning tool. Through these case studies we speculate on how expressing more urban planning laws through computer code might be beneficial to stakeholders in decision making processes.

Keywords: Urban Planning, Computational Design, Geographic Information Systems, Design Tool, Web App, Data Interoperability, Building Regulations, Policy, Building Law, Grasshopper

TABLE OF CONTENTS

DEFINITIONS

This research was part of a two-part theses, completed in collaboration with Cox Architecture and fellow research partner Madeleine Johanson

Although these terms enjoy a rich and interesting life outside of this study, for the purpose of this paper, the following terms will be defined as follows:

Term	Definition
Algorithm	A process or set of rules to be followed, by a computer, usually to calculate or perform problem-solving operations.
Computational Design	Refers to the application of computational strategies to the design process.
Modularisation	The process of dividing up a script into its separate functions, so that the modules can be reused.
Parametric Design	The application of algorithmic methods to define design relationships and produce a response to apply to buildings and urban organisation.
Urban Planning	The process of design and organisation of urban space. Deals with the development and use of land, permissions, protection of the environment, public welfare and infrastructure.
Geographical Information Systems (GIS)	A computer system where spatial and geographic data can be stored, managed, analysed and maintained.
Visual Scripting	Any programming language that lets users create programs by manipulating elements graphically rather than textually.
Grasshopper	A visual scripting environment used to create programs that generate and manipulate geometry and data. It is a plugin for Rhino.
Scripting	Any programming language that automates the execution of tasks, carried out by the computer.
JavaScript	A programming language, commonly used to create interactivity on websites.

Application Programming Interface (API)	A set of functions that allow a website to interact with an application to provide an additional service or functionality. For example, in this project we use Mapbox's API to provide the map for the application.
Mapbox	Provides custom online maps for websites and application, using open source information from OpenStreetMaps and NASA. Some applications that use Mapbox include Uber and Airbnb.
Open Format	A file format for storing digital data, defined by a published specification, which can be used and implemented by anyone. E.g. Portable Document Format (PDF), widely used to transfer and present documents.
Open Source	Software that distributes it source code, that anyone can inspect, modify and enhance. The opposite to this is 'closed source' software, where only the organisation who created the source code can make modifications (Opensource, 2017).
GeoJSON	An open file format for encoding geographic data.
Building Envelope	The space a building occupies on a site.
Permissible Building Envelope	A 3D form defining the maximum space a building occupies on a site.
Local Environment Plan (LEP)	Legislation for planning decisions for local government areas.
Height of Building (HOB)	The maximum height of the building allowed by the LEP.
Floor Space Ratio (FSR)	The Floor Space Ratio is the ratio of the total area of a building's floors to the area of the site, as dictated by the LEP.
Git	Git is a distributed version control system that can detect and keep track of changes in documents.
Github	GitHub is an web-based platform that uses Git to allow developers to store, manage and host projects online.

Gist	Gist is a service provided by GitHub that allows you share code with other people. You can share small snippets of code, particular files, or an entire project.
Gross building Area (GBA)	The total area in square meters of all the floors in a building.
Gross Floor Area (GFA)	The total area of usable floor space inside a building

List of Figures

List of Tables

INTRODUCTION

Laws are a codification of a society's norms. They capture the current state of those norms in a way that can be communicated and interpreted with some degree of consistency. Programming languages do a very similar job; they capture a set of operations and decisions in a way that can be consistently executed. This paper explores the benefits of capturing law in programming languages, and demonstrates how it can be applied to urban planning laws through several prototype scripts. These scripts were developed with an aim to automate tasks within urban planning, such as testing new policies and urban schemes.

The following chapter discusses the current role technology has played in law, and defines features of laws that allows and prevents them from being captured as code.

Computational Law

Computational Law is a field that has been developed through the integration of computer systems with law. It is a "branch of legal informatics concerned with the mechanization of legal analysis (whether done by humans or machines)" (Genesereth, 2015). The field uses conditions and relationships between the statements of the law and automates the language into a computational script. Currently, this is through computer systems which automate legal decisions such as "compliance checking, legal planning, and regulatory analysis" (ibid.). A distinguishing feature of these systems is that they have the capability to interpret the content of laws, rather than just search for them. By automating these laws into a system, decision making is made easier. As the field develops, "Governments have increasingly sought to utilise automated processes which employ coded logic and data-matching to make, or assist in making, decisions." (Perry, 2014). For Example:

"Intuit's Turbotax is a simple example of a rudimentary Computational Law system. Millions use it each year to prepare their tax forms. Based on values supplied by its user, it automatically computes the user's tax obligations and fills in the appropriate tax forms. If asked, it can supply explanations for its results in the form of references to the relevant portions of the tax co"

(Genesereth, *Computational Law*, 2015)

Tax systems are an ideal candidate for Computational Law systems because the rules that they compose are logical and systematic. Computational Law demonstrates a fundamental advantage of integrating technology with law; to automate time consuming process.

Law as Code

Law as code is the concept of translating laws into programmable logic which can then be processed by a computer. It differs from Computational Law as it is an alternative to the current documentation of laws as legal prose. The code itself becomes the law. This is particularly ideal with laws that are based on a series of statements, as they are more straightforward to translate into conditions within a computer system. Documenting laws as code allows for new opportunities, such as better maintenance, testing and simulation methods of laws (Genesereth, 2015).

The translation of arguments into code was explored as early as the 14th Century by Majorcan philosopher Ramon Llull, "who tried to make logical deductions in a mechanical rather than a mental way" (Dalakov, 2017). He invented a paper 'machine' as a means to convert people to Christianity through logical statements. It "proposed eighteen fundamental general principles [...] accompanied by a set of definitions, rules, and figures in order to guide the process of argumentation, which is organised around different permutations of the principles" (Gray, 2016)

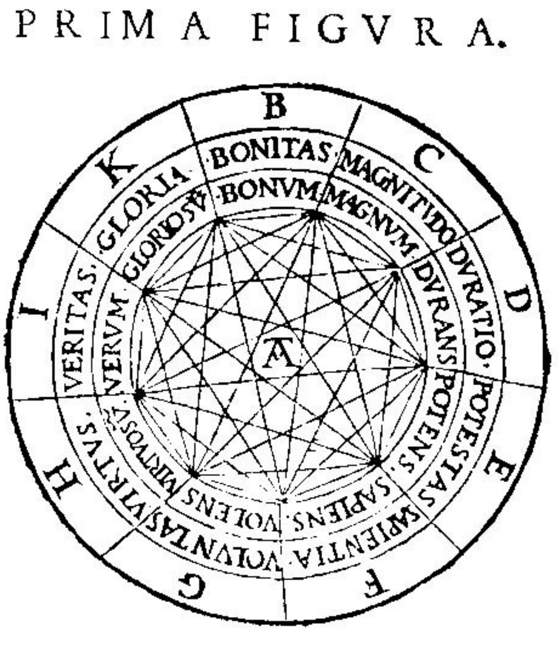


Figure 1: Prima Figvra (Dalakov, 2017)

This is an early attempt to capture norms through explicit rules—a defining characteristic of computation. Similar to laws, it is a tool of reasoning.

Ambiguity in Law

The wording of laws are often left ambiguous to allow for freedom of interpretation in future situations. Real world situations are unpredictable and can be judged differently based on context. Therefore, "many legal decisions are made through case-based reasoning, bypassing explicit reasoning about laws and statutes" (Love, 2005). This lack of absolute specificity makes it challenging to translate laws into computer language.

Some laws are more clear-cut. If a driver exceeds the speed limit, they have committed an offence, and many speed cameras will autonomously issue a fine (Hartzog et al, 2015). Extenuating circumstances can be considered in an appeal, but this still reduces the total amount of work done by humans. Enumerating the extenuating circumstances is a potentially infinite task. What defines a medical emergency? How injured does a person need to be to allow speeding? This is dealt with human intervention in an appeal by a judge or magistrate. Similarly, automated systems could handle the more logical, rule based systems and then pass the more complex decisions to an expert. These human-machine "centaur" teams have had some significant success in chess (Cassidy, 2015), and are endorsed as a likely future scenario for work.

The creation of new areas of law is likely to remain the domain of skilled humans for the foreseeable future. Incomplete contract theory (Bolton and Dewatripont, 2005) describes the problems of planning for the future; situations arise that none of the parties could have imagined leading to discussions to resolve the extra-contractual event. "Whoever designed the computer system cannot gather good data on all of these factors so that the program can take them into account. The only way to do that is to have a much more comprehensive model of the world than any computer system has" (McAfee and Brynjolfsson, 2017). The complete contract theory conceptualises that every possible state has been agreed upon, eliminating the need for legal discussion or courts (Bolton and Dewatripont, 2005). This theory is mirrored in the debate of autonomous cars, where almost every conceivable situation are identified, with the vehicle's response pre-planned. If the programming fails, human lives are placed at risk. This also draws attention to Thompson's (1985) classic 'Trolley Problem,' with the ethical debate of whose life to prioritize in an emergency. The debate is centred around whether killing one person is worth saving the life of five others. The answer is dependant on the morals of each individual. After 32 years, there is still no clear cut answer to this question. If humans are unable to resolve this debate, then it becomes less likely that the responsibility would ever be passed onto a machine.

However, with the progression of Artificial Intelligence (AI), Musk (2015) predicted in a tweet “when self-driving cars become safer than human-driven cars, the public may outlaw the latter”. Interpretation of nuanced laws is also likely to take a long time before machine intelligence can outperform humans, however, the large proportion of laws that are already somewhat procedural is likely to be automatable in the very near future. The relative magnitude of laws of this type is unknown, but is likely to be significant. If those could be automated it would enable commensurate productivity gains.

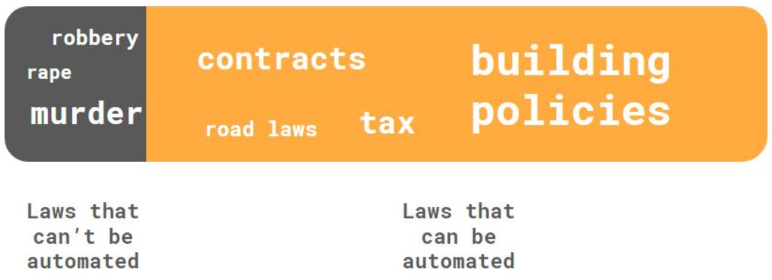


Figure 2: Laws that can’t and can be automated

Many of urban planning’s law falls into the category of largely procedural. For example, planning controls that define the maximum buildable space on a site. Checking for compliance involves a procedural test of each planning control. With the more ambiguous laws set to the side, we can define what can be programmable.

Translating Language to Code

Mathematical expressions are easy to translate into code as they are made up of precise statements. Language, whether spoken or written, is harder to translate into code due to its arbitrary nature. This poses the question of whether computer code can capture laws the same way that language can.

The history of Artificial Intelligence (AI) illustrates how technology has attempted to capture rules to make decisions that typically require human intelligence. There have been several times where AI system have outperformed humans.At the outset of AI, Allen Newell, J.C. Shaw and Herbet Simon created the “Logic Theorist” program in 1956. Hailed as a ‘thinking machine,’ the program used rule-based logic to prove mathematical theorems (McAfee and Byrnjolfsson, 2017). British philosopher and mathematician, Bertrand Russell responded with delight as the program was able to solve and eloquently express 38 of this theorems from the book “Principia Mathematica”. Logic Theorist used symbolic logic, with numbers, words, and symbols that humans can compute.

AI stalled in the 80s, known as the ‘AI winter,’ as funding dropped (ibid). During this time, a machine was capable of mimicking simple human functions, solve theorems and mathematical problems, and play chess (Faggella, 2015). Interest ignited again with IBM’s ‘Deep Blue,’ a computer programmed to play chess, that in 1997 beat the world chess champion. The architecture in deep blue could be applied to financial modeling, data mining and molecular dynamics (IBM 100, 2017).

To date, AI has made a lot of progress, but not from symbolic logic. In an interview in 2012, Chomsky, a linguist, was sceptical of the way AI has progressed “...it was assuming you could achieve things that required real understanding of systems that were barely understood” (Katz, 2012) whereas the ‘new AI’ — focused on using statistical learning techniques to better mine and predict data — is likely to yield general principles about the nature of intelligent beings or about cognition.’ Technology like Google Home[1] operates within the behaviourist theory—mimic what they have been programmed to do, however, cannot think for themselves when the command lies outside their scope.

A major difference between computer code and law, is that a script can only be interpreted in one way. It is read by a computer, which processes exactly what it is told to do. Law, on the other hand, “depends on the intricacies of natural language, whether spoken, written or printed” (Hildebrandt, 2013). In the procession of translating laws into computer executable code, or even the process of translation in general, "shades of meaning may be lost or distorted" (Perry, 2014). Translation becomes more difficult with laws that have many statements and clauses which alter the consequence of the condition.

Deconstructing how each word contributes to a law allows an understanding of its intention. However, this is easier said than done, there are many words in the English language that change definitions based on context. For example, ‘a set of keys on the table’ can translate to a group of keys on the table, while ‘set the keys on the table’ means to place or lay the keys on the table. In this case, ‘set’ is a word that has multiple definitions, however, there are also words that have one meaning but change based on context. The phrase ‘chuck the mug’ could mean both to throw the mug, or to simply pass the mug. ‘Chuck’ by definition is to throw or toss something carelessly. In this case a mug isn’t an object you would typically throw, which suggests that the phrase is most likely suggesting to gently hand over the mug. To make it even more difficult, ‘chuck the mug’ could also refer to

[1] Google Home is a voice-controlled smart speaker designed for home automation

throwing away or discarding the mug, it could even refer to a person named Chuck, whose peers disparagingly deem to be a mug.

In 1979, a research team gave their translator AI program the phrase “The spirit is willing, but the flesh is weak” to convert to Russian. The program produced the Russian equivalent of “The whisky is agreeable, but the meat has gone bad” (McAffee and Brynjolfsson, 2017). Humans can naturally figure out the correct definition of the word based on context and inflection, however a computer can not. These sorts of wordplays are analogous to Winograd Schemas (Winograd, 1972) in that the ambiguity needs context and domain knowledge to resolve.

Symbolic programming languages can understand context and domain knowledge in human language. Wolfram Language is an example of a symbolic programming lanauge, that uses rewritable symbols, like in algebra, rather than values to perform calculations. This means inputs “don’t immediately have to have “values”; they can just be symbolic constructs that stand for themselves" (Wolfram, 2016). This allows the programming language to understand constructs in human language. “The Wolfram Language has, for example, a definition of what a banana is, broken down by all kinds of details. So if one says “you should eat a banana”, the language has a way to represent “a banana”.” (ibid.)

To make the terms in the language explicit, each word is defined with one meaning. That way there is only one interpretation for each word. Once a word has been defined the programming language can understand its meaning in a sentence and “do explicit computations with it.” (ibid.)

Michael Poulshock is a Legal Knowledge Engineer who translates laws into computer programs and has experimented with Wolfram Language. In 2015 he started a project called Hammurabi Project, with the aim “to make law-related determinations - that is, to apply legal rules to a given factual situation and report the result.” (Poulshock, 2016). The project is named after King Hammurabi, who is believed to be the first leader to document laws by having them etched onto stone tablets. Poulshock’s concept behind the Hammurabi Project is not only to make law as code, but to compile legal rules across many areas of the law, such as taxation law and contract law, to then make them freely available for open use. It is written in Wolfram Language, meaning it is in an executable format that can understand constructs such as ‘tort’. “Once executable, it can be embedded into our computing infrastructure where it can drive other applications.” (Poulshock, 2016). While it is an experiment, the Hammurabi Project can be considered an example of law as code.

Chapter Summary

Current programing languages are not sufficient enough to capture the arbitrary nature of all human language. Symbolic languages, such as Wolfram language make it more probable for more explicit sentence structures to be written in code. If laws were more explicit, they could written as code.

RESEARCH AREA

'...urban planning needs to be flexible and dynamic with a view to the future.'

Verebes, *Masterplanning the Adaptive City*, 2014

Urban planning regulations are a type of law that shape our cities by defining the opportunities and constraints that allow buildings to be made. In a simplified view, there are two main stakeholders - the councils who define the regulations, and the designers who plan within them. Councils are challenged by defining regulations that need to be applied on a large scale, without being able to evaluate in detail, the efficiency of the proposed regulations. While designers have to then comply with several documents of regulations and translate them into a built form.

The concept of law as code can be applied to the domain of urban planning to automate processes and resolve problems these councils and developers face. This paper, investigates how planning policies can be made explicit by interpreting them as logic within software. If a building regulation such as the maximum height of a building became a module within a script, it could be used to create a permissible building envelope. Creating a series of these modules, each of which represent a different building law, allow the complex relationship between building regulations to be visualised as a city. In this sense, the script allows an explicit translation of urban planning laws, leading to a better understanding of the intent behind specific rules, as well as help inform better urban outcomes.

Methodology

The research was conducted using an action based approach. Action based research typically applies to real life situations, and is a method of "learning by doing" (O'Brien, 1998). It involves a cycle of planning, doing, observing and reflecting to reach a certain outcome. This method was well suited with the framework set by the industry partner embedded in the research who desire tangible outcomes.

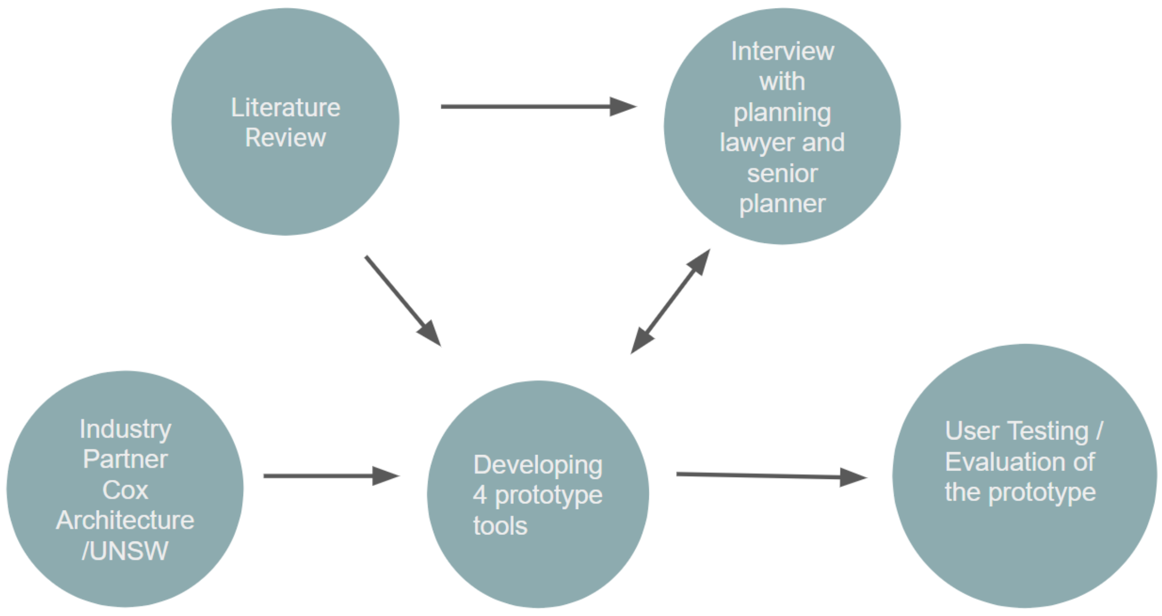

Literature Review
Interview with planning lawyer and senior planner
Industry Partner Cox Architecture /UNSW
Developing 4 prototype tools
User Testing / Evaluation of the prototype

Figure 3: Overall methodology diagram

As shown in the diagram above, there are several factors that contributed to the outcome of this research. The main elements include:

Cox Architecture

Cox Architecture works at the intersection of technology, design and legal frameworks. This thesis extended Cox's understanding of all three of those spheres and provided promising directions for commercial application. Important to Cox was the way the thesis was presented. Web technology will be core to the way architects communicate content and this thesis substantially advanced Cox's ability to use this medium.

Interviews with Experts

To gain insight of the planning industry in regards to this research, two Planners were interviewed. Both interviewees were representing themselves and not their place of work. Key quotes from the interviews are embedded throughout this paper, while the full transcripts can be found in the appendix. The interviews helped to define the scope and problem that the prototypes address.

The interviewees were:

David Haron, Senior Planner at NSW Department of Planning & Environment

Peter Holt, Specialist in environmental, planning and local government law at Holding Redlich

Literature Review

This research consists of three overall disciplines - law, urban planning and computational design. Therefore, a review of literature was used to gain an understanding of each field, which was then applied when asking questions for the interviews. A study of precedents allowed an understanding of current problems and how they have been solved as is. The study of literature and precedents both helped shape the prototypes developed as part of this research.

Research Question, Aims and Objective

Computational Law has had success in automating procedural processes within law. Based on this observation, a similar framework can be applied within urban planning. This leads to the core research question:

How can the application of 'law as code' benefit urban planning?

With the aim to:

- Improve transparency and effectiveness when interpreting the law
- Allow an intuitive understanding of complex zoning laws through technology
- Use an interdisciplinary approach to solve problems in urban planning
- Streamline testing and simulation of new urban schemes

This resulted in the objective to:

Create a script that encodes building regulations, and use it to generate permissible building envelopes.

In a parallel study by Madeleine Johanson, the building envelopes were used to to test the compliancy of buildings. This is documented in the thesis *Adjudicating by Algorithm: Creating an open web platform to inform preliminary urban design stages.*

Prototype tools (Marine Park)

This research resulted in the creation of 4 prototype computational tools between the two theses.The motivations and a more detailed explanation of what each tool is will be explained further on in the thesis.

Humpback: Grasshopper Plugin *(Nazmul Khan)*

A tool responding to the problem of data interchange formats between GIS and computational tools. Humpback allows Grasshopper to read and write GeoJSON, a GIS file format used by urban planners.

Beluga: Building Generation Script *(Nazmul Khan)*

A script which generates permissible building envelopes in accordance with the Local Environmental Plan (LEP) and Development Control Plan (DCP) standards of Sydney.

Minke: Versioning Tool for Grasshopper *(Nazmul Khan/Madeleine Johanson)*

Minke is a tool to detect changes in documents, and manage multiple versions using GitHub Gists.

Dugong: Interactive website *(Madeleine Johanson)*

Dugong is an open source web GIS platform. It is a decision support tool for urban planning, acting as a visual communication platform. This will be used as a method to represent the building generation script, allowing users to design within the various clauses of the legislation for any site in Sydney through an interactive and responsive interface.

UNDERSTANDING THE PROBLEM

Planning Controls

In Sydney, the current Local Environmental Plan (LEP) is provided online comprised of in excess of 100 documents, mostly zoning maps. Like the majority of zoning maps that exist, they are “two-dimensional and fail to provide a clear picture of the code’s impact on development” (Didech, 2015). These maps are available openly online, however the sheer amount of documents makes the process of obtaining and collating them a challenge. This is apparent when you take into consideration that a council is split up into a grid of small maps, and that each regulation is documented on a separate map.

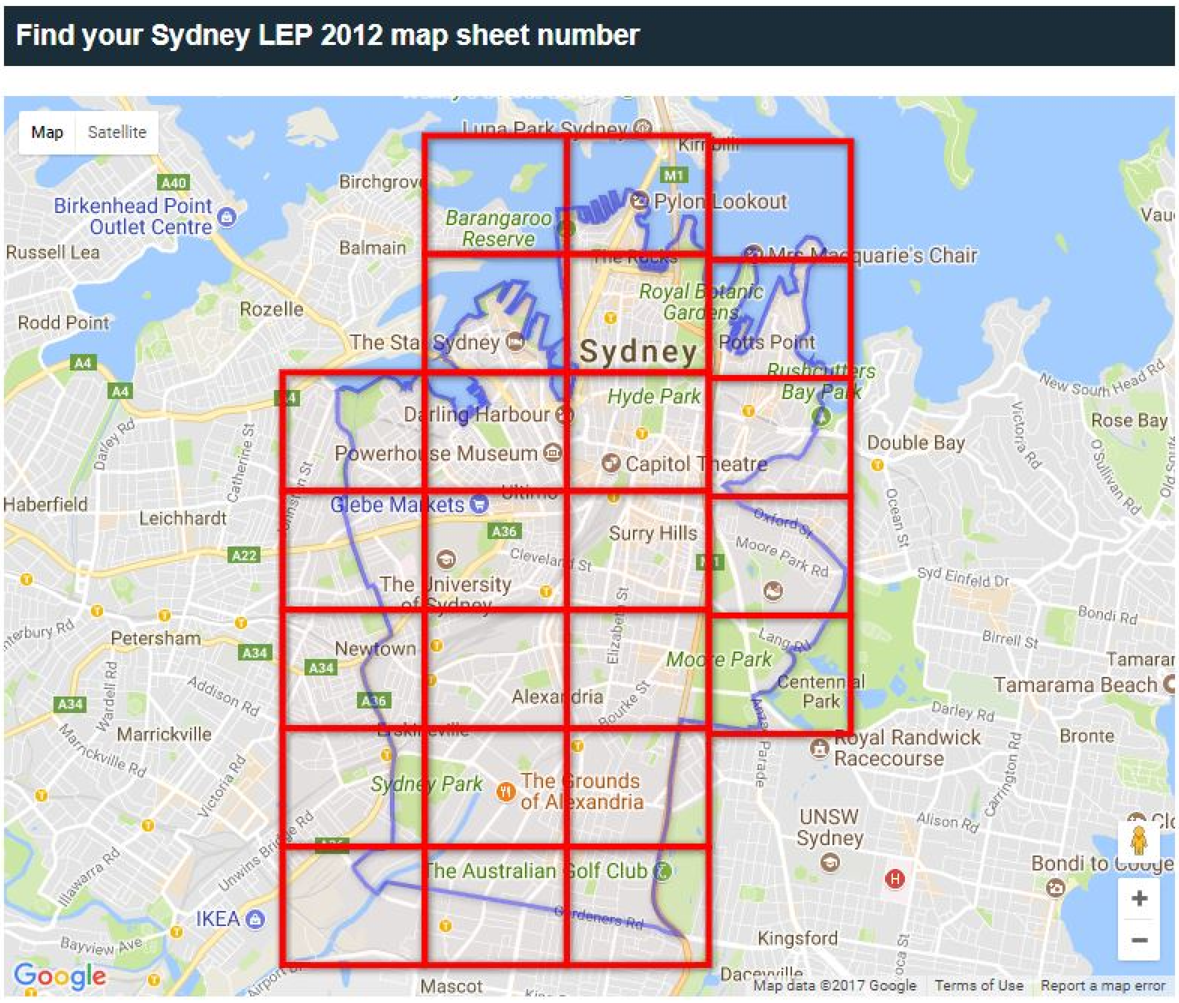


Figure 4: Sydney LEP are divided into 24 sheets per planning control.

They can also be downloaded as GIS datasets.
Developers not only have to consider the LEP, but also the Development Control Plan (DCP), which documents more detailed regulations such as setbacks. When working with multiple sources of information it becomes even more difficult to obtain an understanding of what regulations apply to a property. Developers could be applying their expertise within the boundaries defined by zoning laws, rather than spend time searching and collating them. The complexity of zoning hinders development, making building construction more costly and time-consuming.

Redundancy in Regulations

Redundancies in regulation makes it difficult for architects and property developers to understand the different regulations that are set. For example, the LEP includes the maximum Height of Building (HOB), and the Floor Space Ratio (FSR) for each lot, which both factor into the maximum allowed height of a building.

In an interview with Planning Lawyer Peter Holt, he expressed his opinion on why Local Environmental Plans generally become complicated. Using the City of Parramatta [2]as an example, he said that they wanted to have a control that protected the amount of light that was available to certain areas along the Parramatta River, but did not "reconcile the heights and FSR's with the sun access plane" (Holt, 2017). Height of Building gives a rough indication of what building can be built on a site. For example, a HOB of 80 suggests a tower could be built 80m tall, however there could be a provision that says you can't go through the sun access plane, which lowers its maximum height.

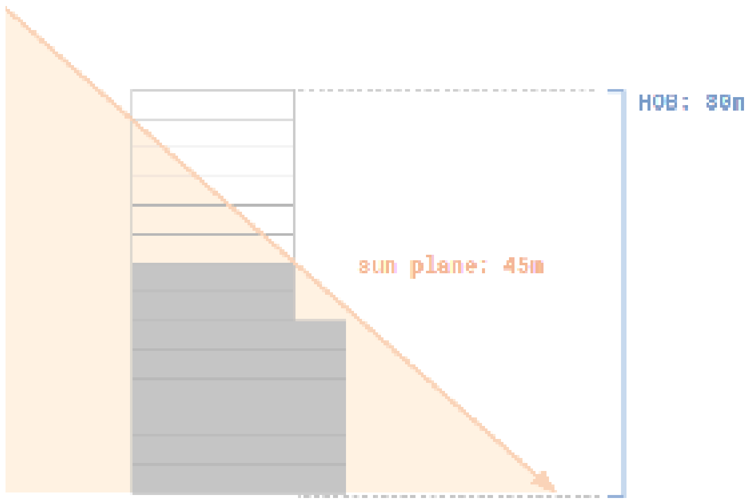


Figure 5: HOB and sun plane defining the buildings height

[2] The City of Parramatta Council, is a local government area in the western suburbs of Sydney

Both of these regulations coexist with one another and both determine how tall a building can be made. If this sun access plane intersects a site before the specified HOB, does the HOB regulation becomes redundant? The precedence of planning policies can change constraints of a building. “So of course when somebody is preparing an application [...] they're stuck with an envelope that doesn't make any sense. [...] They're stuck with that height and that FSR, but with a provision that says, by the way you can't actually go higher than that, because of the overshadowing caused on the park” (ibid.).

The redundancy in regulations can be traced back to the process of reconciling controls. If the process of updating existing controls was easy, than perhaps they could be more consistent and coherent.

Towards 3D Planning Controls

Councils are tasked with defining regulations that need to be applied on a large scale, without being able to evaluate in detail, the effects of the proposed regulations. The first problem is that 2D maps make it difficult to communicate 3D information. However, planners have grown accustomed to working in 2D.

“A good planner, and this is what planners do all day every day, knows based on this height and the FSR that this is the kind of built outcome that could be achieved, but beyond that, that sort of real, really penetrating analysis of the data to try and drive insights, there's not much there” (Holt, 2017). This a limitation of working within a 2D environment, most 3D spatial relationships don’t get considered working in this format. This is shown when planners “do a reasonable job of extruding an envelope based on a set of controls, but it never really fits the bill because it's been conceived in plan view based on a particular approach and then you're extruding it as 3D”(ibid.). This leads to problems in the real, 3D world, such as overshadowing, or even the collision of buildings.

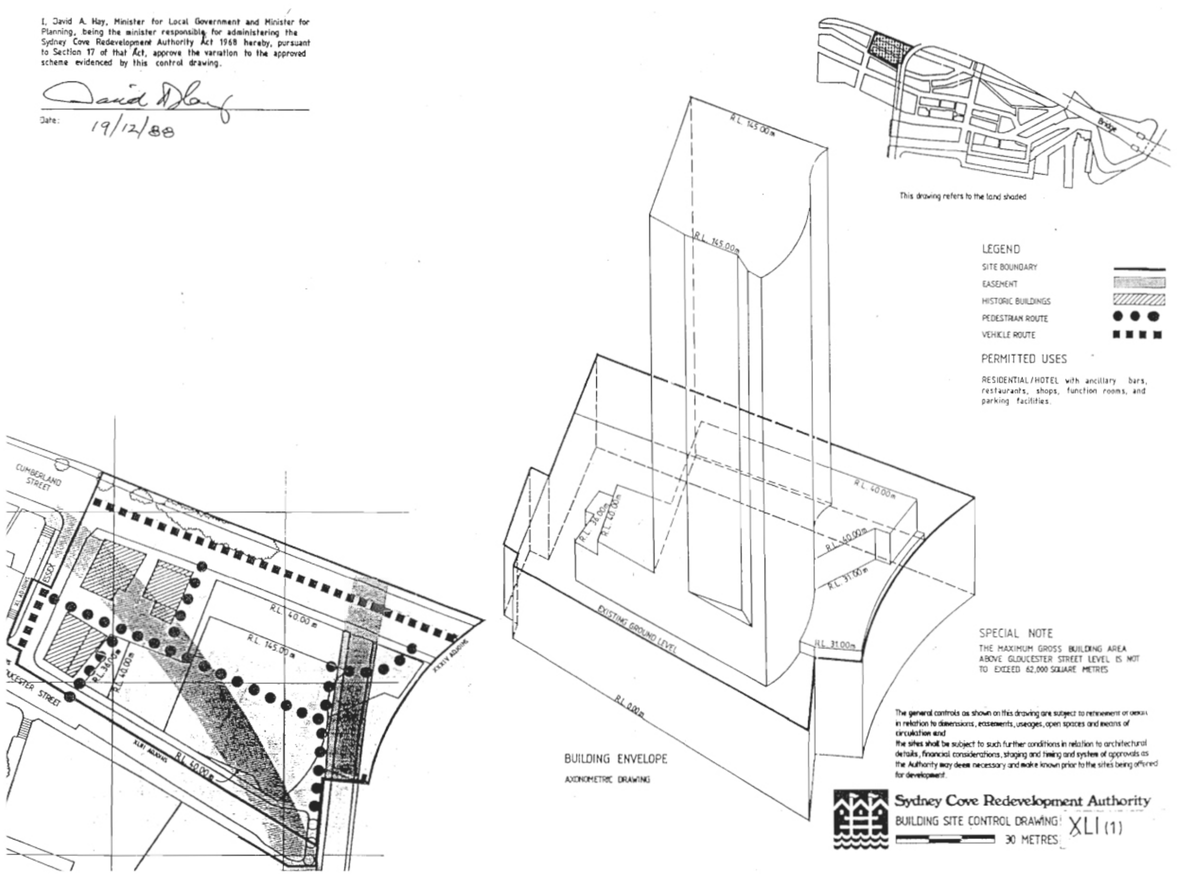
Aside from this there are many more issues associated with 2D planning controls. In Sydney, arcades and small retail stores that are located underground are technically listed as ‘parks’ because the plans that define their land use are documented on ground level.

‘Essentially, if you want a modern planning system, you need modern tools and a modern way of thinking about it, and that's digital.’ (Haron, 2015). 3D planning controls involve the documentation of regulations in a 3D format to allow more consideration when planning building regulations. “The way that you produce the information ultimately dictates the outcome.” (Holt, 2017). With 3D envelopes as an outcome of planning controls you can “assess overshadowing, bulk and scale, impact issues, [...] you can show the relationship to the existing streetscape, you can have a little person walk around and look up” (ibid.)

Sydney Cove Redevelopment Authority Scheme

The Sydney Cove Redevelopment Authority Scheme in 1984 has drawn 3D envelopes as an outcome of planning controls. Building envelopes were hand drawn as axonometric diagrams, which represented the maximum buildable area of a property.

With this scheme “you knew what you were gonna get. You could price it and from a public perspective they knew what they were gonna get. (Holt, 2017). Developers were able to know what Sydney Cove would look like 20 years from now.



I, David A. Hay, Minister for Local Government and Minister for Planning, being the minister responsible for administering the Sydney Cove Redevelopment Authority Act 1968 hereby, pursuant to Section 17 of that Act, approve the variation to the approved scheme evidenced by this control drawing.
Date: 19/12/88
This drawing refers to the land shaded
LEGEND
SITE BOUNDARY
EASEMENT
HISTORIC BUILDINGS
PEDESTRIAN ROUTE
VEHICLE ROUTE
PERMITTED USES
RESIDENTIAL/HOTEL with ancillary bars, restaurants, shops, function rooms, and parking facilities.
R.L. 145.00m
R.L. 40.00m
R.L. 31.00m
R.L. 0.00m
EXISTING GROUND LEVEL
CUMBERLAND STREET
GLOUCESTER STREET
BUILDING ENVELOPE
AXONOMETRIC DRAWING
SPECIAL NOTE
THE MAXIMUM GROSS BUILDING AREA ABOVE GLOUCESTER STREET LEVEL IS NOT TO EXCEED 62,000 SQUARE METRES
The general controls as shown on this drawing are subject to refinement of detail in relation to dimensions, easements, useages, open spaces and means of circulation and the sites shall be subject to such further conditions in relation to architectural details, financial considerations, staging and timing and system of approvals as the Authority may deem necessary and make known prior to the sites being offered for development.
Sydney Cove Redevelopment Authority
BUILDING SITE CONTROL DRAWING: XLI (1)
30 METRES

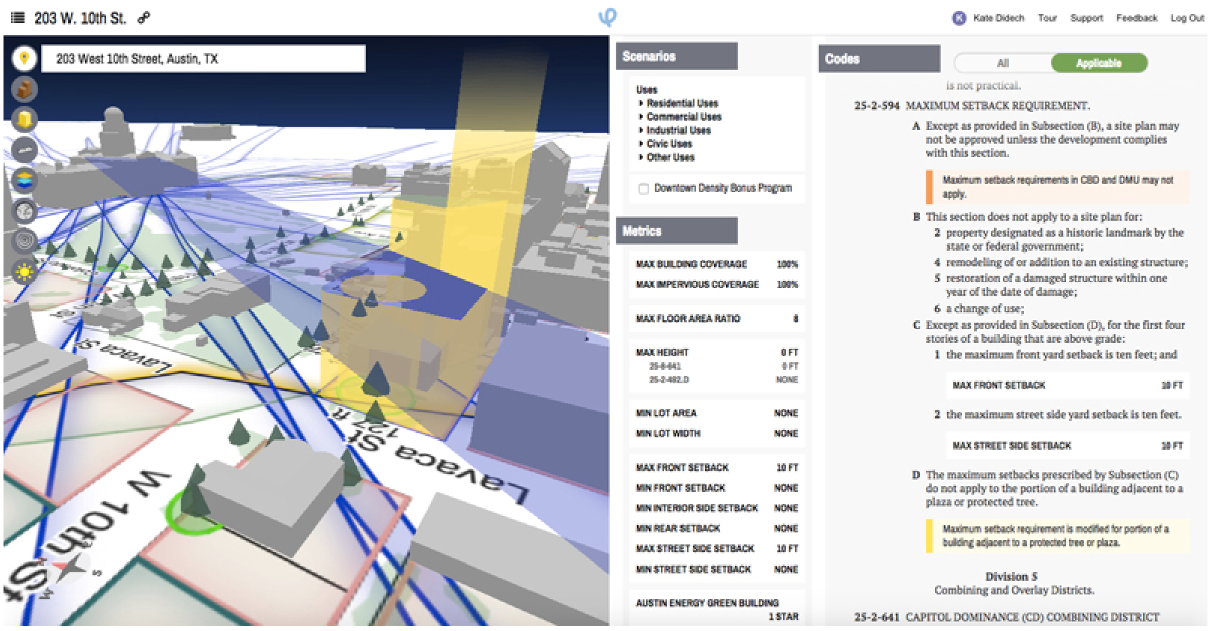
Figure 6: Sydney Cove Redevelopment Authority Scheme

The diagrams included a site plan, and site specific information such as roof levels, permitted land use types, maximum floor area and exceptions to the envelope. This provides a developer with an instant boundary to work within. “The Rocks is a great outcome, and I suspect that part of that is the effort that went into checking and cross checking the 3D scheme and the fact that they stuck with the 3D scheme over a long period of time" (ibid.). Considering it was 1984, the envelopes were not available in a digital format. However, considering the abundance of technology at our disposal today, the creation of these envelopes could be made a less laborious process.

Precedent Studies

Flux Metro

Flux Metro was a web app that combined multiple urban planning sources, both private and public, and translated them into built form. The application focused specifically on the city of Austin, and visualised permissible building envelopes in accordance to zoning regulations rather than a copy of what exists. Created in 2014, the web app's interface was a 3D map which users could navigated through to view specific sites. By doing so developers could easily see a site's opportunities and developmental risks. The motivation behind the project was that "the complexity and opacity of zoning hinders development, making building construction more costly and time-consuming" (Didech, 2015). Using computational techniques, Flux Metro was able to "automatically analyze the multitude of regulations imposed by the [Land Development Code's] base and overlay zones, watersheds, Heritage Tree Ordinance, and the like, instantly providing users key metrics like maximum building height and minimum setbacks" (ibid.). A fee was charged per property for indepth information.



MAX BUILDING COVERAGE	100%
MAX IMPERVIOUS COVERAGE	100%
MAX FLOOR AREA RATIO	8
MAX HEIGHT	0 FT
25-8-641	0 FT
25-2-482.D	NONE
MIN LOT AREA	NONE
MIN LOT WIDTH	NONE
MAX FRONT SETBACK	10 FT
MIN FRONT SETBACK	NONE
MIN INTERIOR SIDE SETBACK	NONE
MIN REAR SETBACK	NONE
MAX STREET SIDE SETBACK	10 FT
MIN STREET SIDE SETBACK	NONE
AUSTIN ENERGY GREEN BUILDING	1 STAR

is not practical.

25-2-594 MAXIMUM SETBACK REQUIREMENT.

A Except as provided in Subsection (B), a site plan may not be approved unless the development complies with this section.

Maximum setback requirements in CBD and DMU may not apply.

B This section does not apply to a site plan for:

2 property designated as a historic landmark by the state or federal government;

4 remodeling of or addition to an existing structure;

5 restoration of a damaged structure within one year of the date of damage;

6 a change of use;

C Except as provided in Subsection (D), for the first four stories of a building that are above grade:

1 the maximum front yard setback is ten feet; and

MAX FRONT SETBACK 10 FT

2 the maximum street side yard setback is ten feet.

MAX STREET SIDE SETBACK 10 FT

D The maximum setbacks prescribed by Subsection (C) do not apply to the portion of a building adjacent to a plaza or protected tree.

Maximum setback requirement is modified for portion of a building adjacent to a protected tree or plaza.

Division 5
Combining and Overlay Districts.

25-2-641 CAPITOL DOMINANCE (CD) COMBINING DISTRICT

Figure 7: Flux Metro Austin Preview renders the maximum buildable envelope for a parcel that is impacted by different building regulations (Didech, 2015)

Flux's initial ambition was to first focus on the city of Austin, and then expand to other cities. However, the project was shut down in February 2017, due to the company's interest in shifting focus towards data exchange plugins (Carlile, 2017).

Envelope Beta

Envelope (envelope.city, 2017) is a web application similar to Flux Metro that simulates and analyses the potential of a property on an urban scale. The app was developed with SHoP Architects and is targeted at the real estate industry. Currently, it is focused on the city of Manhattan, New York, with more than 35000 properties available for analysis. After unlocking a property with a bought key, users can see what can be built based on zoning laws. The website allows the users to manipulate key characteristics of a property, such as its land use, which depending on the zoning laws will generate a different form. This form is available for download in a 3D file format (OBJ); designers can continue to develop the envelope in external software.

Envelope automates searching and interpreting zoning laws to generate draft analysis, a process that typically takes 5-10 hours (ibid.). Through Envelope this process is essentially automated to the click of a button. It is able to confirm existing land use and zoning designations in seconds and run preliminary 3D scenarios in minutes (ibid.).

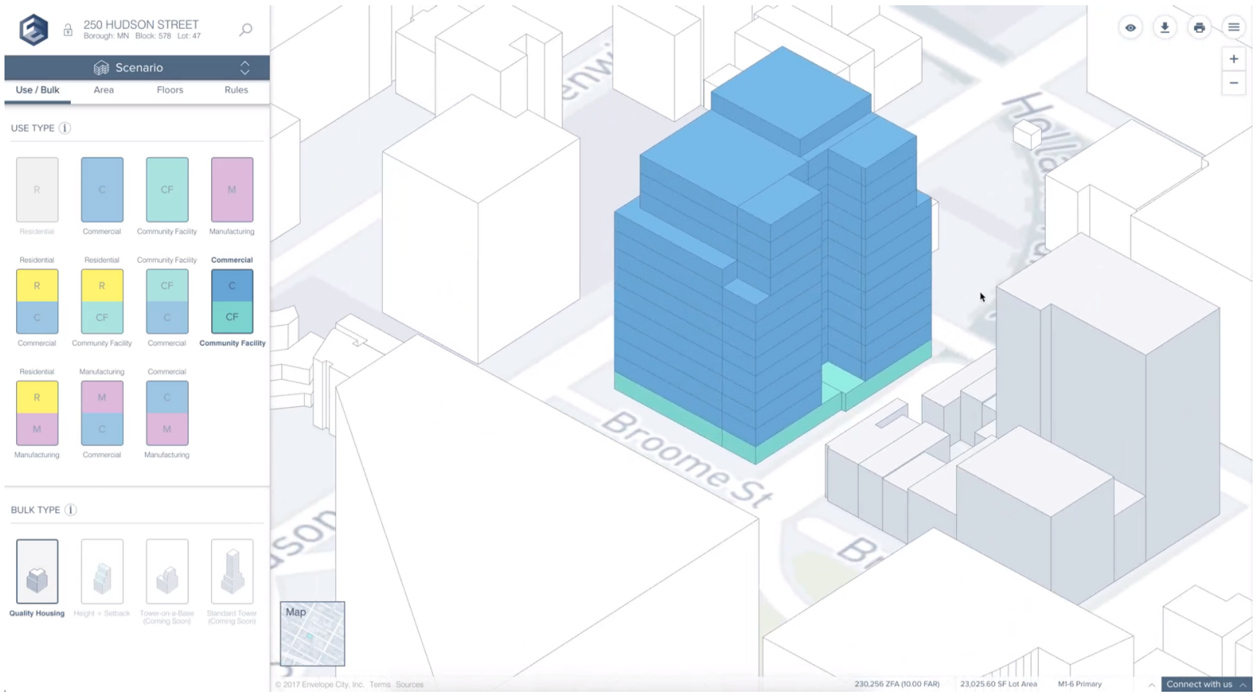


Figure 8: The interface of Envelope Beta, showing the maximum building envelope for the selected site (ibid.)

Using Envelope, users are able to download floor area metrics and reports which contain appropriate zoning information according to the site specified.

The web app is still in its beta phase and the company “just secured a $2 million funding round to develop its platform and launch marketing efforts in New York City and beyond.” (Wheatley, 2017). Envelope’s next step is “to build new products based on its software algorithms that will allow real estate professionals to ‘search for and visualize potential at a neighborhood or urban scale.’” (ibid.).

COMPUTATIONAL METHODOLOGY

This chapter discuss what computational design is and the tools used in the discipline. This is followed by a overview of what tools were used to reach the objective of this research and the relationships between them.

What is Computational Design?

Computational design is a discipline that focuses on embedding computational techniques into the design process. It derives from a combination of architecture, computer science and engineering. A common misconception is that computational design involves being the end user of computer software to produce design outcomes. This is typically referred to as a "computer-aided" approach, rather than "computational" (Menges and Ahlquist, 2011). The key difference from a computer-aided approach to design is that it is not taking advantage of the computational power of the computer (Terzidis, 2006). A key part of computational design involves creating custom tools throughout the design process, which drive design decisions. This often involves using programming languages to rationalise 3D forms, that would be rather difficult to design otherwise.Meaning that the outcome is not necessarily determined by the designer beforehand. This makes computational design explorative by nature, and involves searching and finding solutions to complex problems.

While designers who use visual scripts are often referred to as "amateur programmers" (Woodbury, 2010), they are able "to actively and intuitively engage with analysis, this approach has the potential to bring about a change in the way that analysis data is understood and applied within the design process" (Aish, 2013).

Computational Design tools

Computation designers use scripting languages to create programs which generate forms. Grasshopper 3D is the most commonly used computational design tool in architecture. It is a plugin for Rhino 3D, a common 3D computer aided drafting package used by architects to model built forms. In Grasshopper, data is manipulated through a visual scripting language where a network of relationships to shape a parametric model (Woodbury, 2010).

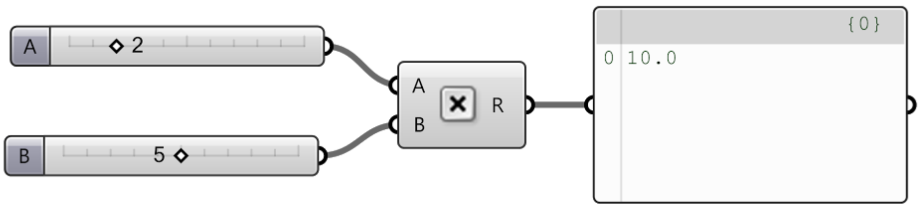
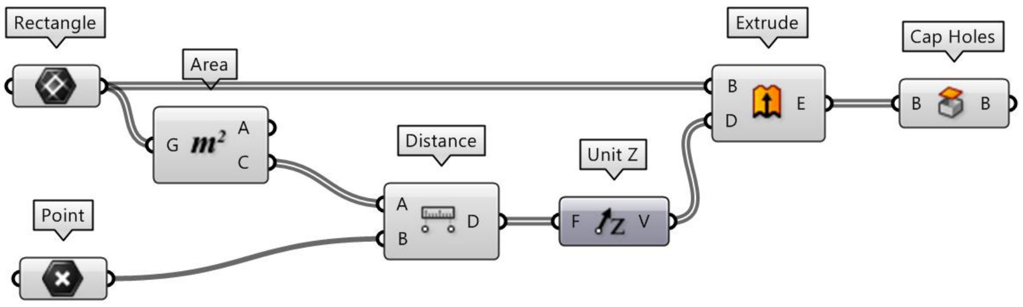


Figure 9: A simple function in Grasshopper with basic transformation of data

The visual scripting program has a library of components which perform specific tasks. For example, a component can be used to draw a sphere given a point in space and a radius. Parametric workflows allow designers to iterate and analyse many configurations of design much faster than traditional methods (Woodbury, 2010). Designs can be programed to respond to real-world stimuli, generating a specialised form. This is done through the construction of relationships within the script. As a basic example, the height of box can be generated from its distance to a point in space.



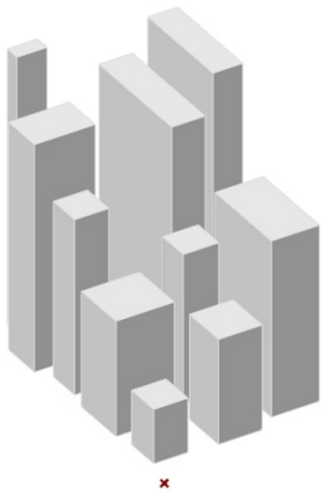


Figure 10: The figure shows parametric relationship between a point and multiple boxes, and the script used to generate it.

Grasshopper comes with the nodes needed to perform common operations. Also users can make their own specialised components, which can contribute to the functionality of the software. These components are often packaged as plugins, and are available online for download. For example, Ladybug is an open source environmental plugin for Grasshopper. It extends Grasshoppers ability to perform environmental analysis.

Addressing the Objective

Objective: to create a script that encodes building regulations, and use it to generate permissible building envelopes

What is a permissible building envelope?

In this thesis a permissible building envelope has been defined as a 3D form that determines the maximum buildable space on a property. It Includes appropriate setbacks and height restrictions, and is a boundary that developers should generally work within to ensure that they are complying with the fundamental building regulations. It's important to note that this form can change depending on different types of landuse. As an example, a commercial building may be allowed to be built taller than a residential building. For the purpose of this research, we haven't considered bonuses.

"I don't think [bonuses] are rational in the sense that you could convert them to program language, they're just a bit kind of like, here's a bucket of six or eight good things and if you do those six or eight great things, then we'll give you even more goodness."

Peter Holt, 2017

Instead, we have considered the permissible building envelope the first phase of testing whether a building is compliant.

The Script – Beluga

Building regulations are laws that dictate how a physical form is generated. Grasshopper allows the scripting of geometric operations in Rhino. Based on this, and its popularity in the discipline of architecture, Grasshopper will be used as the computational scripting tool to generate permissible building envelopes.

In a computer program there are generally three elements:

1. the input
2. the process
3. and, the output

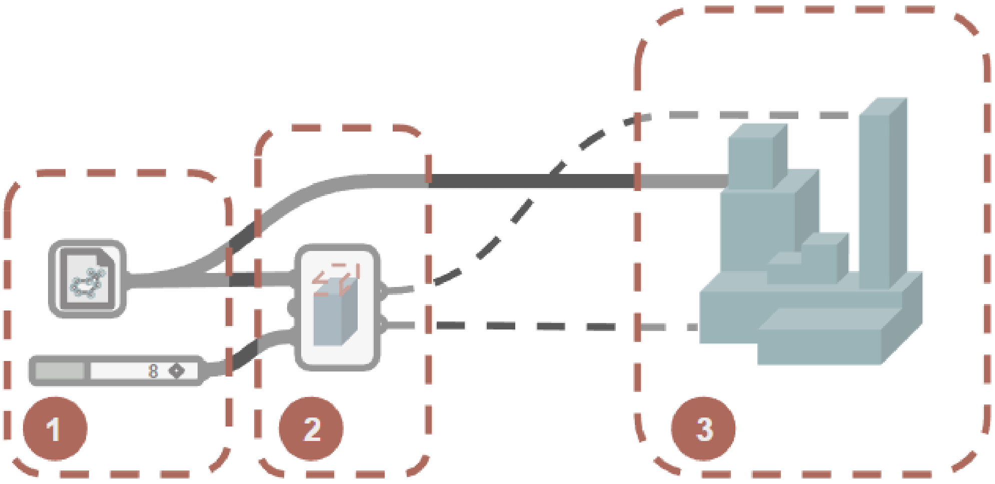


Figure 11: diagram of beluga script on a fundamental level

Beluga's main function is to create building envelopes based on planning controls. With this information we can make some basic assumptions of what the inputs of the script will be:

- Cadastral lots - the boundary of a property as a 2D polygon
- Height of Building (HOB) Control
- Floor Space Ratio (FSR) Control
- Solar planes as a 3D geometry
- Setback rules

Computational Workflow

While HOB and FSR are typically documented in PDF's that are available online, they are also available for download as GIS datasets. Australian Urban Research Infrastructure Network (AURIN) is an initiative of the Australian Government that "brings together and streamlines access to more than 1,800 datasets previously difficult, time consuming or costly to obtain" (AURIN, 2010) through a web platform. The cadastral lots, HOB and FSR policies of Sydney are available for download on the website, however only in a format that Grasshopper can't read. This software limitation lead the research in the following computational workflow.

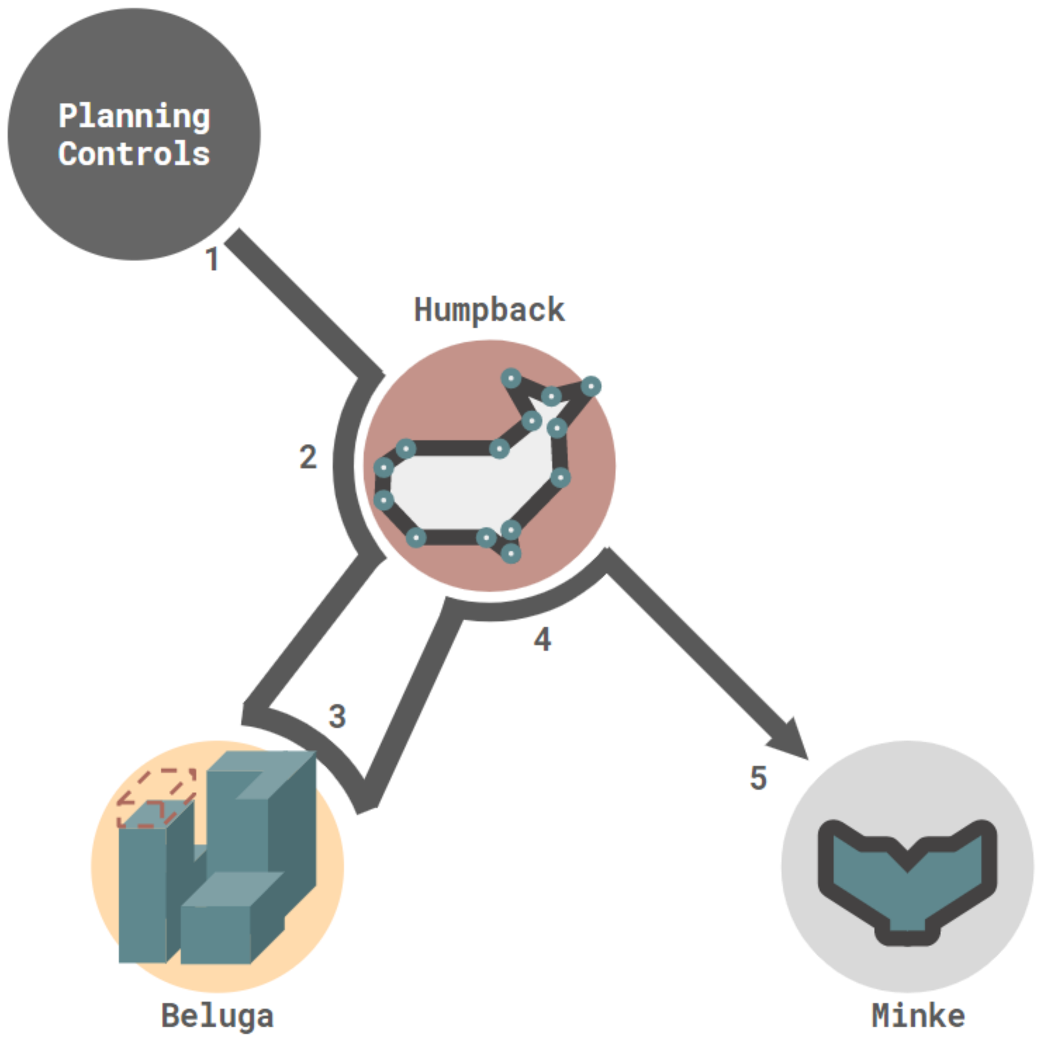


Figure 12: Relationship between prototype tools in this research

1. The LEP planning controls were downloaded as GeoJSON files from Aurin.org.
2. Humpback was created as a tool to convert the GeoJSON file into geometry in Grasshopper.
3. Grasshopper interprets the planning controls and uses them to generate permissible building envelopes. This script is called Beluga and creates a visualisation of urban planning policies.
4. These building forms then get sent to Humpback, which converts the geometry back into GeoJSON.
5. The GeoJSON is stored using Minke, where changes and previous versions can be seen.

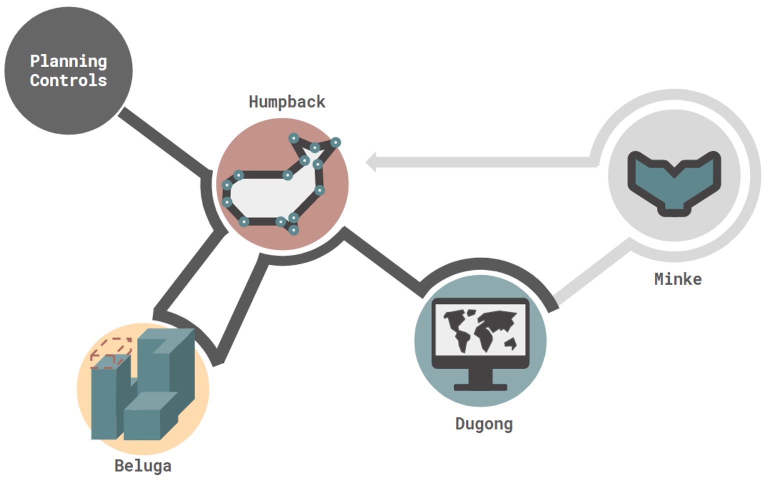


Figure 13: Alternate methodology diagram

As a parallel study, the building envelopes get turned in GeoJSON, and then visualised on an interactive website developed called Dugong, where it is used to test whether building designs are compliant. This research is documented in *Adjudicating by Algorithm: Creating an open web platform for interaction with the law.*

Modularisation as a Scripting Philosophy

Although computational design in architecture is a relatively new field, it has already established scripting practices that tend to be project specific as a result of being rapidly developed.

Architects that program "do enough to accomplish the task at hand, but they don't have the time or inclination to do it strategically" (Davis, 2015). This lack of strategy leads to computational solutions that are only applicable to very specific projects, preventing its re-application to other scenarios. This results in functions needing to be remade, when they could be reused. While visual scripts are generally easier to read than written code, "it can be difficult to know what to erase, where to make the edit, and what needs relating and repairing, because the visual tangle of relationships within the script can obfuscate–rather than reveal–the function and relations of operations" (Davis et al., 2011). This makes it especially difficult when reading scripts that are written by another author.

Creating a script that encompasses various building regulations requires a great deal of complexity. Therefore, a strategy for developing the script was considered to maximise efficiency in long term maintenance.

Software engineering is a more established discipline that has resolves many of the problems computational designers face. In the 1960s software developers feared that unstructured programs were becoming too complex for humans to read and write (Davis et al., 2011). This is a result of GOTO statements, which were used to jump to another line of script based on a condition. A series of these statement resulted in scripts that were convoluted and hard to maintain. As a response, structured programming methods became more popular which involved the reuse of certain functions by containing them within modules. Each module within a script can be tested in isolation, allowing software developers to flag and fix errors in their scripts more easily.

From this, we can extract that there is need for improvement in scripting practices within the computational design discipline. Perhaps rather than “accomplishing the task at hand” (Davis, 2015), scripts should be written as functions that can be abstracted and used for multiple purposes. Structured programming is a method that can be adopted by computational designers. A series of modules can be collated to create a toolset, which performs flexible small tasks.

If a single module captured a building regulation, it could have its own inputs which determine how it influences a building lot. A series of these modules, each capturing a different building regulation, would allow a complex relationship between each policy and how they influence the permissible building form. This strategy was used when developing Beluga and Humpback, making the flow of data between modules easy to understand.

HUMPBACK

Immediately, we are faced with the challenge of importing the cadastral lots and planning controls into Grasshopper. While these datasets are available openly online, they are only available for download in a limited number of file formats. This includes Shapefile, CSV, and GeoJSON. Humpback is a tool developed as a response to this problem. It reads and writes GeoJSON files in Grasshopper. Rather than“accomplishing the task at hand” (Davis, 2015), Cox Architecture as an industry partner desired a toolset that could be used for commercial application outside of this research.

This chapter argues that there is evidence that the incompatibility of data between software potentially creates a barrier between computational design and urban planning technologies. This observation is shared in general terms by Kuus (2002) who states that “The lack of universally adopted standards has produced a situation, where the communication between different systems is made hard.” When translating this argument to urban design, one could argue that this potentially inhibits a computational designer’s ability to analyse and generate urban datasets.

An investigation into the relationship between software and file formats between computational design and urban planning, allow us to understand what is limiting the integration of computational design into the urban planning industry.

Data Interchange Formats for GIS

Urban planners have been strong advocates for the technological development and deployment of Geographical Information Systems (GIS) (Drummond, 2008). GIS allows urban planners to build planning support systems, where spatial and geographic data can be stored, managed, analysed and maintained. These data can be stored in various file formats, for example ESRI’s Shapefile, File Geodatabase (GDB) and many others. The file types differ depending on the program it was developed for, and the format used to store information. Some are interchangeable between GIS platforms, but not all. The current industry standard file format for GIS software is ESRI’s Shapefile (SHP). However, 3 files make up this format, one with the feature geometry, one with the shape index position and one with the attribute data (GISGeography, 2017).

“GeoJSON is a format for encoding a variety of geographic data structures” (GeoJSON.org, 2017). It is an extension of JavaScript Object Notation (JSON). The main differences between this format and Shapefile is that GeoJSON contains all geometry and attributes in one file, and is “easy for humans to read and write... and easy for machines to parse and generate” (json.org, 2017). “JSON is an open standard data-interchange format that uses plain text to transmit data objects" (Hickok, 2014). A defining characteristic of GeoJSON is its use of a key and value system to parse information. An element can have keys and values attached to it.

```
{
        "type"  : "polygon"
       "colour" : "grey"
       "height" : "80"
}
```

In this example the key "type" defines what type of geometry it is, "colour" defines what colour the geometry gets displayed as, and "height" determines how many metres the geometry gets moved off the ground. The specific key names are not predetermined, it is up to the entity that creates the GeoJSON. The keys and values are just text, they don't really mean anything until software interprets those specific keys and values. This makes GeoJSON the ideal file format for web GIS applications, such as ArcGIS Online and Mapbox. In its creation, GeoJSON has been 'derived from preexisting open geographic information system standards and have been streamlined to better suit web application development using JSON' (GeoJSON.org, 2017). GeoJSON is typically used for data interchange in GIS software as an 'open standard and has been widely adopted' (Jones, 2013). Popular GIS systems that use GeoJSON include ArcGIS, Aurin, QGIS and Mapbox.

Motivations for Humpback

Computational tools allow us to create dynamic iterative architectural forms, whereas GIS platforms enable us to display and analyse spatial data on a geographical map, often to solve spatial urban functions.

Yet, the incompatibility of data formats between computational tools and GIS remains a challenge. Plugins for the Grasshopper environment extend the program's toolset, but currently there are no plugins for Grasshopper which support GeoJSON conversion, a popular file format within GIS. This limits Grasshopper's compatibility with GIS systems, Humpback aims to remedy this lacuna.

In 2002, Kuss acknowledged the gap between computational tools and GIS in 'The Interoperability of Geographical Information Systems.' "[New standards of practise] may increase the usage of geospatial data, bringing new people to the world of GIS by enabling them easily supplement their databases". If a simple, powerful workflow connected the two, better informed urban possibilities can be explored. This is where Humpback contributes as it allows computational tools and GIS platforms to speak the same digital language, creating a bi-directional workflow by extending

Grasshopper’s capability to read and write JSON files. Janssen (2016) outlined this relevance stating that “Workflows capable of integrating geographic mapping and parametric modelling systems could enable various rapid iterative urban prototyping methods to be developed.”

A simplified flow of data between computational tools and GIS platforms will cultivate a symbiotic relationship. Enabling the adoption of computational tools into GIS will allow iterative, smarter designs to expand into mainstream architecture. GIS allow the designer to understand the context of the site and make informed design decisions based on data (Kuus, 2002). Computational tools add another dimension of data, the dimension of optimised possibility.

In regards to this research, Humpback was developed in order to import in planning control boundaries from GIS to Grasshopper, and vice versa. This allowed the computational things to happen within within a computational environment, and then be put back into a GIS environment that urban planners are more familiar with.

Open Formats as a Method

Independent development in one field may leave the other behind in terms of technology and methodology (Drummond, 2008). Consequently, disciplines must share their emerging advancements if they are to grow and develop (Senske, 2005). One way of sharing this knowledge is through the use of open file formats to enable transparency between disciplines. Using a common format to interchange data between software allows communication between the platforms. An open file format is described as, “a freely available published specification which places no restrictions, monetary or otherwise, upon its use” (Open Definition, 2017).

An example of a common open file format is docx for Microsoft Word. The move to open formats “allow[s] developers to more easily create applications that can access data within Word documents…to be more open and more transparent” (Zetlin, 2010).

Programs can interpret and use data that may otherwise be incompatible (Kuus, 2002). Humpback converts geometry and its attributes to a GeoJSON file, an open format (GeoJSON.org, 2017). Formatting information in this way makes it simpler for different tools to access and use the data. In the built environment, open datasets and formats allow us to make more informed urban design strategies on a bigger platform. Combining GIS and Computational Design tools enables an alternative workflow, using flexible tools to inform the design.

Evaluation of Existing Tools

Currently, there are methods for urban analysis within Grasshopper, each which have their own limitations.

In 2011, a feature request for importing GIS data to Grasshopper to make the “data as accessible as possible for scripting” (Golder, 2011) was made on the Grasshopper forums. No conclusion was reached, but it indicated the need in the Grasshopper community for this tool.

Meerkat

In 2014, Meerkat was released on Food4Rhino, an online platform to distribute Grasshopper plugins. Meerkat is a set of tools used generate Grasshopper geometry from GIS shape files, developed by Nathan Lowe up to 2015 (last entry). In this tool, Shapefiles can be batch 'geolocated and cropped' in a web browser (Grasshopper MeerkatGIS, 2017). For mapping urban trends, Webb (2014) used Grasshopper and the plugin MeerkatGIS, which “not only facilitates GIS shapefile importation, but also trims multiple GIS shapefiles to limit the data to a selected region.” This produces ‘.mkgis’ files which is no longer compatible with GIS programs. Users report that “Formats like these [Shapefiles] are a pain to work with. You can't just open them up in a editor and make changes. In addition, the requirement for multiple files is a problem for web applications where you want to download a single file into the browser, access its data and render them in JavaScript” (Jones, 2013). Hence, in an ideal workflow, information can flow smoothly in both directions. Formats like GeoJSON facilitate this.

_EXPORT

Also in 2015, a Grasshopper component called ‘_EXPORT’ by Guillaume Meunier was published on the discussion forums (Alliages, 2017). However, this required multiple steps to install, and depended on external plugins and Python script to run in Grasshopper.

Mobius

Janssen’s web parametric modeling system, Mobius, can be used to create modular workflows to GIS programs, such as QGIS. (Janssen, 2016). However, this is not specific to a Grasshopper workflow. Janssen’s work showed how through an open source file format, GeoJSON, information can be exchanged between different tools. This creates a bi-directional pipeline that can receive, transform, and send information. The paper demonstrates how computational tools can be incorporated in and urban planning workflow.

Based on the evaluation of work by others we identified the following principles that helped us to develop Humpback. With Grasshopper you are able to create your own tools that can be applied to geographical datasets. While this may be more complicated and time consuming to set up, it can lead to more flexible analysis, where parameters can be adjusted to suit the specific task. “The standardization of spatial data formats and the methods by which they are exchanged is cultivating an environment of geospatial interoperability, and allows various software components to communicate more easily” (Kuus, 2002). This shows relevance of breaking the current barriers between GIS software and computational tools like Grasshopper.
From existing examples, the predecessors attempt to close the gap between GIS and computational tools. Currently *Meerkat* and *_EXPORT* only support a one-directional flow of data. Mobius allows the bi-directional flow from computational tools. Humpback contributes a solution, supporting a bi-directional flow from Grasshopper and any GIS application that uses GeoJSON. Humpback is a translator for computational forms. Forms are converted to an open source file type that can be read and edited by both GIS software and computational tools. By cutting out complicated procedural steps, a simple functional workflow is created (Janssen, 2016).

Development of Humpback

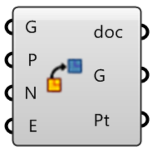
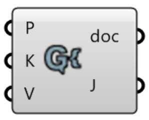
Humpback was developed in anticipation to be used by a broader audience than just the authors. Since GeoJSON is an open format, writing a script to interpret the format was set out to be an achievable goal.

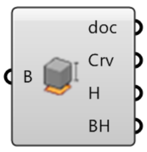
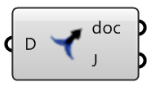
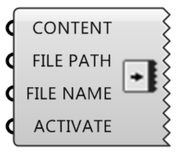
The components created were based on a series of text-based manipulations, such as extracting the list of coordinates in a GeoJSON file, and then using them to create points which can be connected together to make a polyline in Grasshopper. To create GeoJSON, the script does the reverse of this and deconstructs polylines into points that are turned into coordinates, which are then formatted in the correct GeoJSON syntax.

Humpback is easy to find and install from Food4Rhino (http://www.food4rhino.com/app/humpback) and contains thorough documentation with sample files and examples. The plugin contains 6 components, outlined in the following chapter. It works with both 2D and 3D forms. This means that GIS platforms capable of displaying 3D forms, such as Mapbox, are able to visualise urban variations in an open format.

SInce its initial release, a second version of Humpback has been released. Humpback 1.1 includes basic bug fixes as well as faster working components.

Humpback Components

Component	Description	Input	Output
Orient (N, E) 	Orients geometry to specified Northing and Easting Coordinates.	**Geometry (G)** Geometry to move. **Initial Plane (P)** Plane to move geometry from. **Northing (N**) Specified Northing coordinates as a number **Easting(E)** Specified Easting coordinates as floating point eg. 151.203655	**Geometry (G) -** Reoriented geometry
Polygon to GeoJSON	Writes a polygon to a GeoJSON file, where keys and values can be defined.	**Polyline Curve (P)** Polygon or list of polygons to convert to GeoJSON. **Key (K)** Properties used to describe geometry. To specify height, base height and colour, use the following keys: height, base_height, colour **Value (V)** Values of properties.	**GeoJSON (J)** Produces a GeoJSON Feature Collection.
GeoJSON to Polygon 	Converts GeoJSON into Grasshopper geometry.	**GeoJSON (J)** FeatureCollection containing geometry type objects. Only accepts	**Polyline Curve (P**) Closed Polyline

		easting, northing (N,E) coordinates.	**Keys (K)** Outputs keys found in properties **Values (V)** Outputs values found in properties
Deconstruct Extrusions 	Deconstructs simple extrusions into polylines, heights, and base heights for later use in 'Polygon to GeoJSON component'.	**Brep (B)** Extruded or capped Breps. Must be extruded in the z axis.	**Polyline (Crv)** Projected base polyline of Brep **Height (H)** Height of Brep. The distance of extrusion. **Base Height (BH)** The height of the base of the brep from the ground XY Plane.
GeoJSON Merge 	Merges multiple GeoJSON FeatureCollection files as one FeatureCollection	**GeoJSON Datasets (D)** Accept multiple Feature Collections. Created for merging the outputs of *Polygon to GeoJSON* components.	**GeoJSON(J)** Merged GeoJSON file as one FeatureCollection
File Write 	Exports text files	**Content** text for export. In this case, geoJSON. **File Path** destination of file. eg.	Exports text file to specified file path

		C:\Users\YourName\Desktop **File Name** Name of file to export. Includes the file type. eg. Filename.JSON **Activate** Boolean toggle, activates export. If True, the file will update live.	

Table 1: Humpback Components

Example workflow

The following example documents the process of converting a simple 3D building into GeoJSON, which will then be used to render on the popular web GIS platform, Mapbox. The Rhino model contains a building that has been constructed from vertically extruded closed curves. The model is constrained to vertically extruded forms because GeoJSON only documents 2D geometry. However, Mapbox can extract defined heights from the GeoJSON file and use them to render 3D geometry.

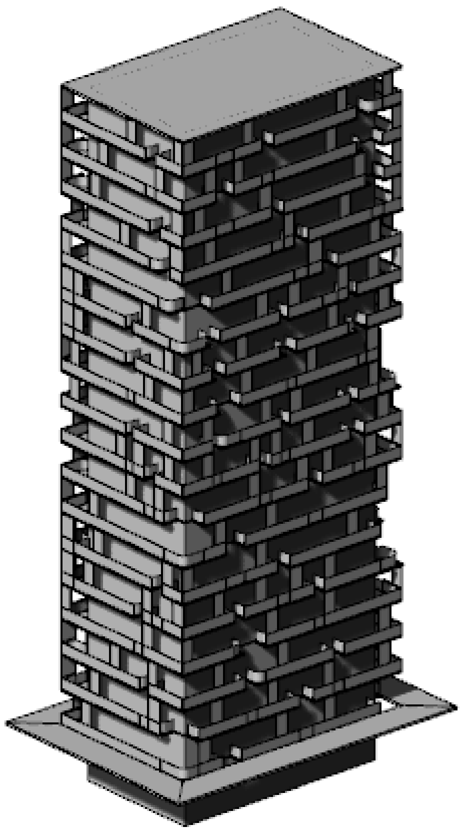


Figure 14: Rhino model of building to be converted into GeoJSON

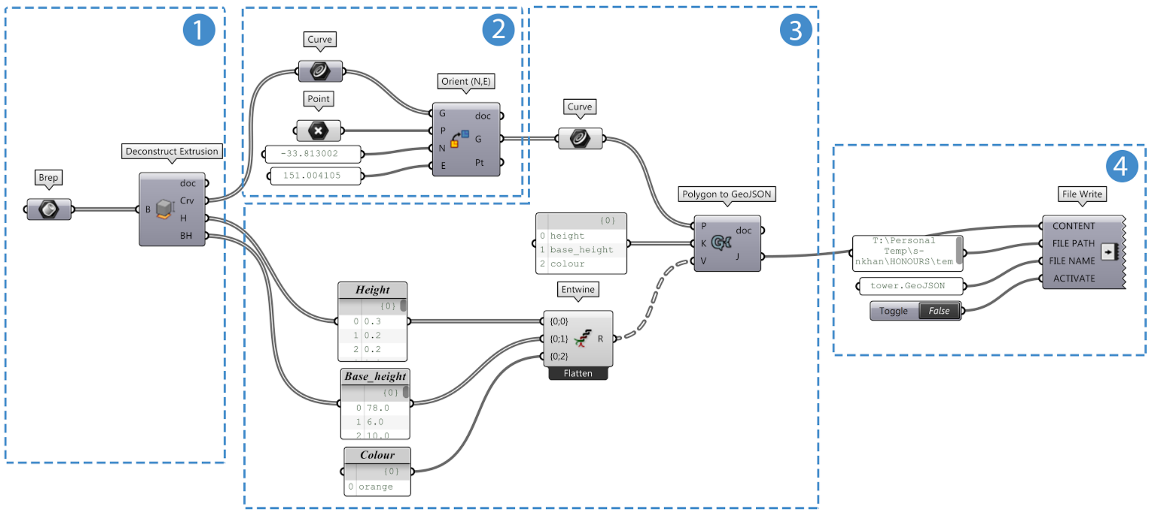


Figure 15: Example Grasshopper script for the workflow below

1) The first step is to deconstruct the extrusions into keys and variables that can be associated to them. This can be done using the *Deconstruct Extrusion* component, which

extracts the form into polylines, heights, and base heights for later use in ‘Polygon to GeoJSON component’.

2) Since the building was modeled at the center of the document (0,0) it needs to be moved to its correct coordinates in relation to the world. This involves using the *Orient (N,E)* component, which moves any geometry to specified Northing and Eastings from a reference point. The base curves that were generated in the previous step will be oriented using this component.

3) Now that the base curves are in the correct location, the next step is to use the *Polygon to GeoJSON* component, which writes polylines into GeoJSON where key and values can be defined. In order to render the buildings in Mapbox, the keys of `height`, `base_height`, and `colour` were specified. The values for height, and base_height are outputs of the *Deconstruct Extrusion* component, while the colour orange was set using a text panel. The list should be structured so that the first branch of data is associated with the first key. The key can be any characteristic that is to be associated with the geometry, (eg. colour, layer, tag) which can then be interpolated and read by the GIS program. A key can either have a 1:1 relationship to a key, or a relationship to many, allowing values to be associated with multiple polygons. The component will now output a valid GeoJSON string.

4) The final step is to export the GeoJSON string to a valid file. This is a simple process using the *File Write* component, where the user must define a file path and filename for the GeoJSON. A boolean toggle activates the export.

Once the file has been saved, the GeoJSON can be viewed on a GIS platform. A web GIS platform like Mapbox can extract and interpret the information in the GeoJSON file and render it in the server. Such a platform has been developed for this project, UrbanCoDe, which can be found at https://madeleinejohanson.github.io/UrbanCoDe. This web application visualises the data contained, and has the ability to turn layer tags within the GeoJSON on and off. This shows the flexibility of GeoJSON within web GIS applications.

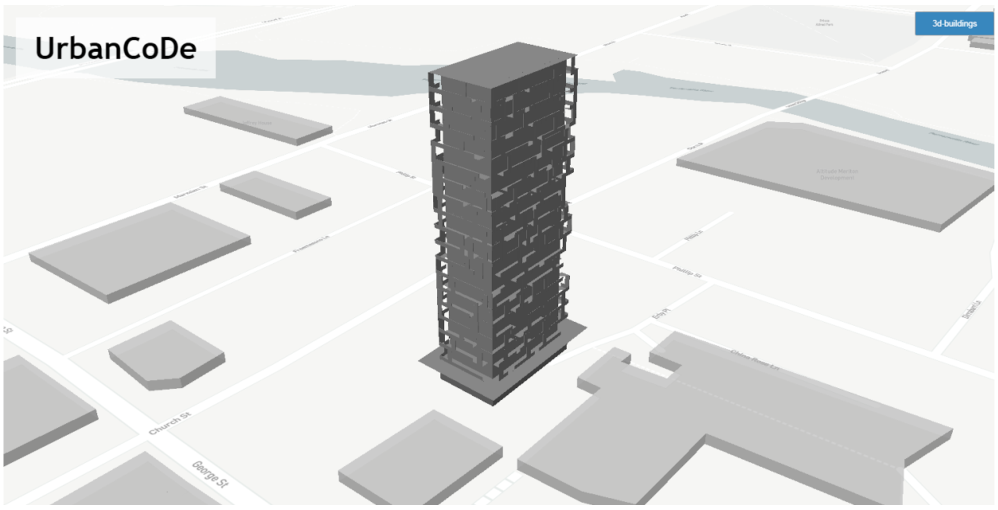


Figure 16: Rendered building in Mapbox

In the same way, the outcomes of the building generation scripts were rendered onto Mapbox, allowing a visualisation of permissible building forms within its geographic context. This can be found at: *https://nazmulazimkhan.github.io/beluga/*

Chapter Summary

Humpback is a bridging tool for GIS and computational tools to share information and functionality. Open source data has enabled software to speak the same digital language. Cultivating a symbiotic relationship, Humpback allows a simplified flow of data between platforms. Based on this data, informed design decisions about the context of the site can be made. Analysing urban datasets is made difficult without this standardization of data (Kuus, 2002). Mainstream architecture can benefit from the smarter designs generated from this iterative workflow. Using this new method with Humpback, the barrier between GIS packages and computational software can be broken.

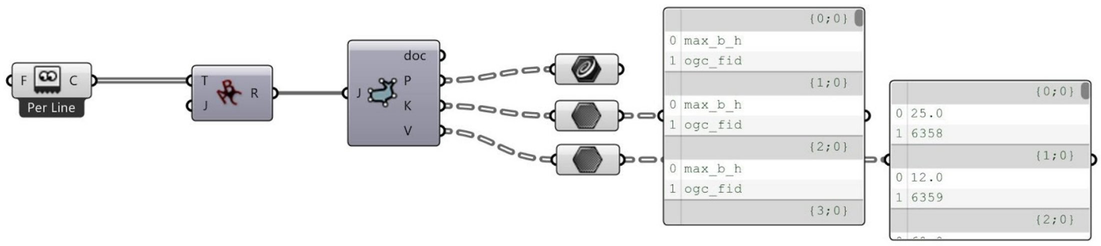
BELUGA

Beluga was developed as part of this research to show how law as code can support a better experience for both designers and policy makers. Using LEP controls and associating them with cadastral lot geometry, 3D legal forms can be generated. The tool uses building regulations to generate 3D forms which represent the maximum buildable space with no bonuses. We refer to these as permissible building envelopes. The LEP controls specified are changeable parameters within the script, allowing the impact of alterations to be seen in real time.

Using the cadastral lot geometry and LEP controls as inputs, the script generates a permissible building envelope as an output. Creating a script that encompasses various building regulations requires a great level of complexity. Beluga captures each building regulation as single module within the script. Each with its own specific processes that determine the transformation of a building lot.

Importing Planning Controls Using Humpback

The first step in creating Beluga is to bring in GIS datasets from Aurin that contain the cadastral lot geometry as well as planning control boundaries. Humpback streamlines this process. The *GeoJSON to Polyline* component converts GeoJSON files into geometry, and outputs the associated keys and values. When developing the script a small section of Sydney’s CBD was chosen. Sydney was chosen as a case study city by Cox Architecture, who could use a visualisation of sydney's planning laws to inform decisions within the practice. However, Beluga could be potentially applied to any city.



```
{0;0}
0 max_b_h
1 ogc_fid
{1;0}
0 max_b_h
1 ogc_fid
{2;0}
0 max_b_h
1 ogc_fid
{3;0}
```

```
{0;0}
0 25.0
1 6358
{1;0}
0 12.0
1 6359
{2;0}
```

Figure 17: Grasshopper script converting HOB GeoJSON file into geometry. In this case the HOB data is stored in a key called max_b_h

Assigning Controls

With the datasets in Grasshopper, the next step is to associate the correct planning control to the cadastral lots. This is achieved by testing whether the boundary of a planning control is colliding with a cadastral lot. The script does this by testing whether the centre point of a cadastral lot is inside a HOB or FSR boundary. If the point is within a boundary then the cadastral lot inherits the property of the planning control.

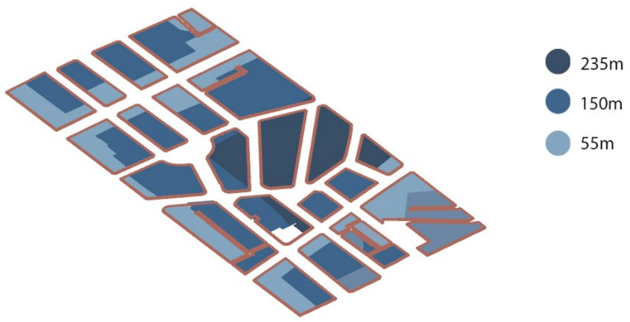


Figure 18: HOB data associated with cadastral outlines

Evaluating Lots

The Sydney Development Control Plan (DCP) contains ambiguous terms such as “front and side setbacks”. The benefit of designing a system where the law is code, is that you can program it to figure out what defines a front face, back face or side face.

The next module in the script does exactly this, it determines the faces on a cadastral lot.

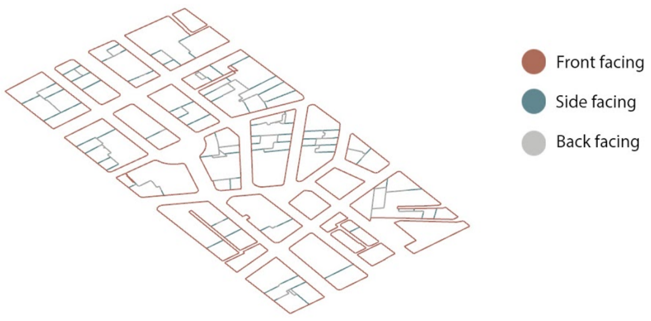


Figure 19: Evaluation of street facing edges

In the current planning control system "you just set the one height and the one FSR across all of them, and then you use clause 4.6 to vary the standard in order to get the outcome on the corner block. Whereas in a more sophisticated system you could run the analysis of that, and the controls that were appropriate for the corner block would be quite different controls from the controls that are applied to all of the blocks that have the straight up frontages."

(Holt, 2017)

Embedding intelligence into a script to understand ambiguous terms allows for less room for interpretation, and exploitation by developers of existing planning controls. This way the law becomes explicit and it's easy to determine whether a developer is complying with building code.

Setbacks and Height of Building

Given that we know what the HOB of each cadastral lot is, and the front, side and back face for each, we can use this information to generate a primary permissible envelope.

The next step is to input the correct setback distances, such as a setback of 7 metres on a street facing facade after 45 metres of height. This is defined in a module that allows users to input a setback for a specified height and side.

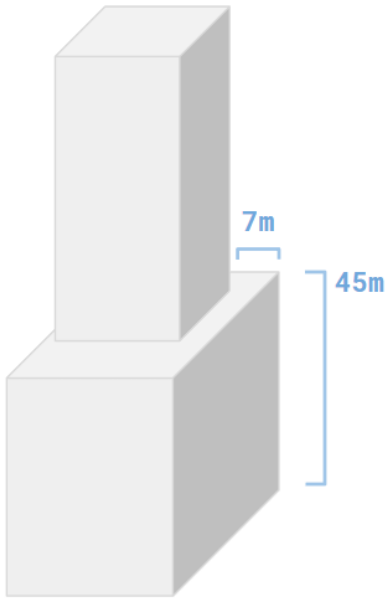


Figure 20: Setback rules defined in Beluga

After the setbacks have been defined, the cadastral lots are extruded up to a height of 45m, then each face gets set back by the correct distance. The building continues to extrude until it reaches its maximum height defined by the HOB control. However, if no setback is specified then the HOB module will disregard setback and create a simple extruded form.

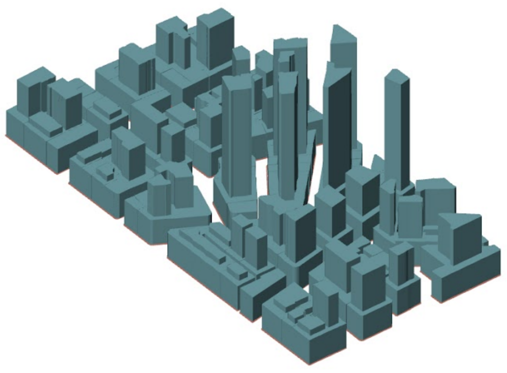


Figure 21: Permissible building envelopes based on HOB and setback data

Solar Access Planes

As different planning controls are added into the script, the logic determining how each permissible building envelope becomes more complex. The building geometry is passed on from module to module, starting from cadastral lots and resulting in permissible building envelopes. This strategy was used to understand how each building regulation impacts the building lots. Each process that generates the permissible building envelope can be seen explicitly. As a consequence, it becomes easier to test whether the modules are doing the correct transformations.

As an addition to the HOB module, a solar access module can be used to determine the maximum height of a building. This was introduced as a response to the Central Sydney Planning Strategy (Central Sydney Planning Strategy, 2016) which would allow buildings to “soar to heights of 310 metres, up from the current restriction of 235 metres” (Saulwick and Visentin, 2016) based on solar access to parks and specified public spaces. These solar panels were modelled manually in Rhino from plans found in the Central Sydney Planning Strategy

The solar access module is used after the the preliminary building envelope is created. If the envelopes intersects with this solar plane, the maximum building height gets lowered to this new height.

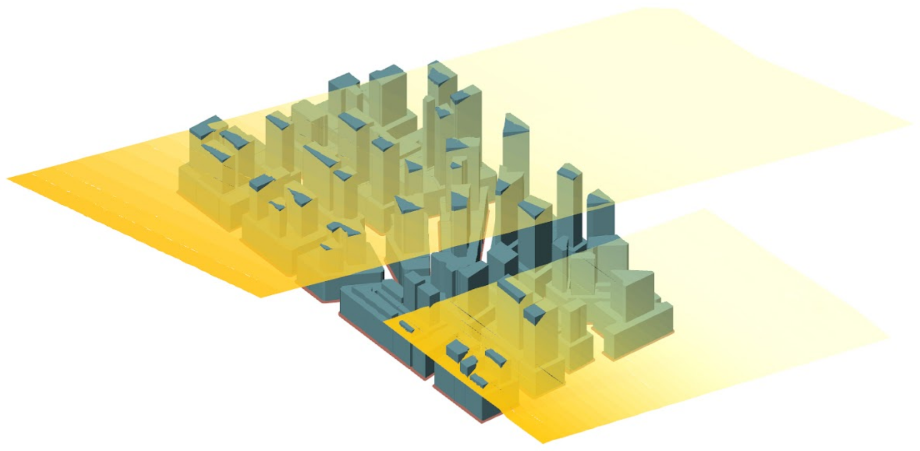


Figure 22: Solar Access controlling the maximum height of building

Floor Space Ratio

Floor space ratio or FSR defines the relationship between the area of a cadastral and the total floor area of what's built on it. Beluga is scripted to make the largest volume possible. If the FSR doesn’t allow the building to reach its maximum height defined previously, the floorplate of the buildings become smaller.

An issue when developing this script was how the FSR relates to the HOB. If a FSR allows a building to be 10 levels high, but the HOB only allows 8 levels, the permissible building envelope can only become 8 levels high. Similar to the HOB and sun access policies, precedence plays an important role to the generation of a building envelope. To resolve this, the FSR can only be applied once a maximum height is already established, whether that be from Height of building controls, or solar access.

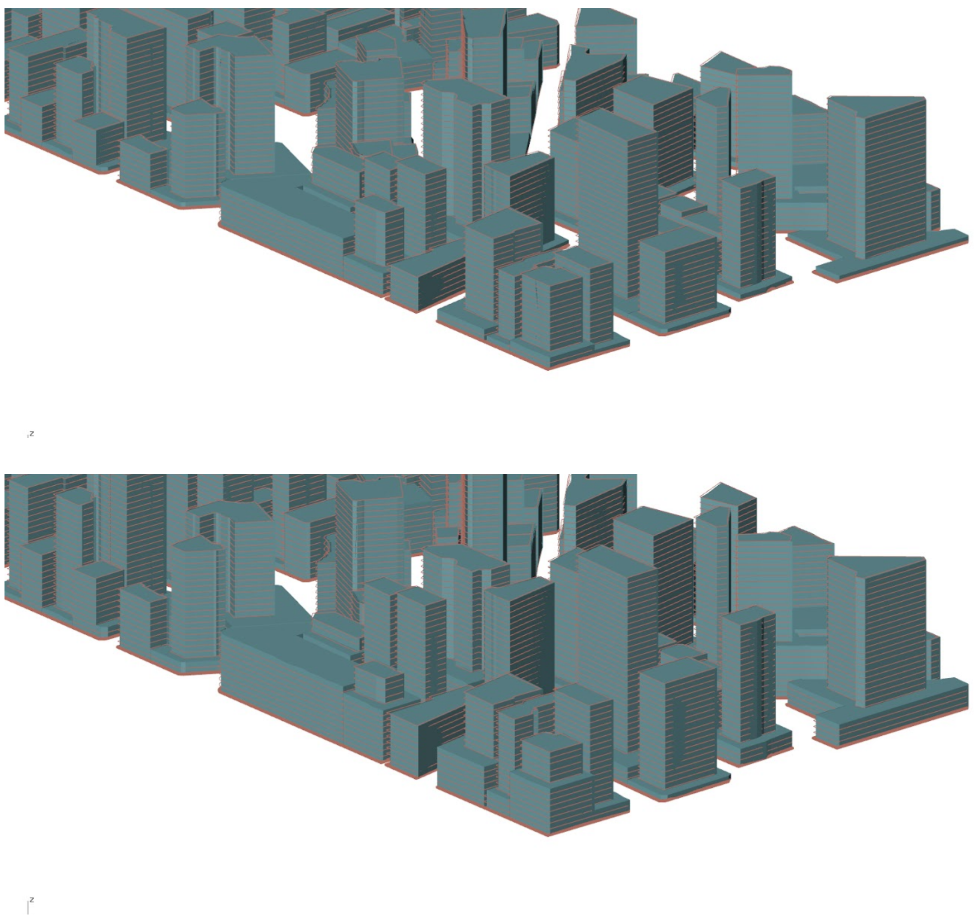


Figure 23: (Top) FSR of 8 (Bottom) FSR of 12. If the HOB control has already defined the height, then the script can’t exceed that.

Chapter Summary

Beluga allows an instant visualisation of how changing the parameters to building policies affects the city on a large scale. By modifying regulations, users can explore how different urban schemes or outcomes can be made. For example, what the city would look like if every lot was built based on solar access to parks and public spaces.

Currently Beluga consists of the following modules: HOB, FSR, Setback, and Solar Access. There are many more modules that could be scripted for future work, such as modules that modify the applied regulations based on land use.

In the future future increasingly complex processes and clauses can be added into the program. Potentially, if a building as 3D geometry were to be put into the program, it could test the buildings feasibility and compliance with the law. A report could be made of all the building regulations and constraints that contribute to that lot, as well as indicate which laws the building is currently complying and not complying with. The fundamental idea behind this is explored in Madeleine Johanson’s thesis *Adjudicating by Algorithm: Creating an open web platform to inform preliminary urban design stages.*

MINKE

This chapter argues that the process of versioning laws could be automated through version control systems such as GitHub. A prototype of a plugin for Grasshopper was developed called Minke which allows a bi-directional workflow between Grasshopper and Gists.

Versioning of the Law

Laws have been documented as early as 1754 BC when Babylonian King Hammurabi had his 282 laws set into stone. “The Code of Hammurabi includes many harsh punishments, sometimes demanding the removal of the guilty party’s tongue, hands, breasts, eye or ear” (History.com Staff, 2009). As humanity has evolved over time, so have our morals. Therefore it’s inevitable for our laws to stay constant. “Each time a new U.S. law is enacted, it enters a backdrop of approximately 22 million words of existing law. [...] Seeing these changes in context would help lawmakers and the public better understand their impact” (Hershowitz, 2015). From the eyes of a programmer, there are many ways to achieve this outcome through version control systems, that have yet to be adopted in practice.

Version Control Systems

“Programming is a three-way relationship between a programmer, some source code, and the computer it's meant to run on” (Shirky,2012). This relationship is simple when there’s only one programmer, however add multiple programmers and you could have potential chaos.

In programming, it is common for the input of one function to depend on the output of others. All of the functions of a script tie together in a complex relationship to perform the overall task of a program. When multiple programmers start editing the same script, the change of one function can impact others. Without coordination, functions within scripts are overwritten causing errors in the overall output of the program.

As a response, programmers have developed ways to manage large projects when working with multiple programmers. The standard solution to this problem is to use of a version control system, which “provides a canonical copy of the software on a server somewhere. The only programmers who can change it are people who've specifically been given permission to access it, and they're only allowed to access the sub-section of it that they have permission to change” (Shirky, 2012).

Git

Git is a distributed version control system that can detect and keep track of changes in documents. It is able to append changes made by multiple people, making it ideal for open source collaboration. It was created in 2005 by Linus Torvalds, the creator of the open-source operating system Linux. There are a few factors that make Git different to regular version control systems. The most distinguishing feature is its ability to branch and merge independent versions of a document. This allows developers to clone an entire copy of a document, make their own changes, and then merge back with the source document, even if it has been altered by someone else since it was cloned. It also gives the programmer the choice of what changes they want to push back to the source.

GitHub

GitHub is an web-based platform that uses Git to allow developers to store, manage and host projects online. It makes Git accessible to a larger audience through its web interface. After creating an account, users can create a repository that contains the contents of a project.

GitHub's versioning system can be applied to the process of creating and updating laws; there are several GitHub repositories that attempt to do so. Stefan Wehrmeyer is a German software developer who downloaded the German federal government's complete laws and regulations and then uploaded them onto GitHub. This was done with the intent to “make it easy for German voters to track changes to the laws – and to also give lawmakers a vision of the future” (McMillan, 2012). Wehrmeyer periodically downloaded the complete German legal code and then used Git and custom tools to figure out what had changed (ibid). Those changes were then submitted into the repository for other users to see. This workflow was made achievable through the German government posting their laws publicly on the internet in an XML format. The repository allows users to download the current set of laws, and be able to see where specific amendments have been made throughout time.

As laws get more complex, the amount of versions increase, documenting how laws change over a large time period. From 2012 to 2013, 77 branches of the repository were made, meaning there are 77 versions of the master version, each branch marking new changes and amendments to the documents.

As mentioned earlier in this paper, Hammurabi Project is also stored in GitHub, allowing users to see how the executable code is developed overtime.

Development of Minke

Gist is a service provided by GitHub that allows you share code with other people. You can share small snippets of code, particular files, or an entire project. A Gist is a mini repository meaning users can clone and make changes to the code, just like in GitHub.

The motivation behind Minke was to use Gists as a way to store, share and version the law. However, if a script is determining the law, what should be stored? The inputs of the script, the process to create it, or the output? These could all potentially be stored using Gists, however for the purpose of this research the building envelopes were versioned.

Using Humpback, the permissible building envelopes can be converted into GeoJSON, which can be posted as a Gist. As building policies change, the permissible envelopes get updated, and a new geojson can be added to the repository. Gist will automatically version and detect changes in the document, allowing you to see what has changed.

GitHub's Application Programming Interface (API) allows software developers to incorporate the services of Gist in websites that use Javascript. The same API was used to script components in Grasshopper to POST, PATCH and GET Gists.

POST is a function in the API which allows you to create a Gist. It returns a URL containing the contents of the code which can be shared and cloned. It requires a filename, the content of the file, and a description of what the content is.

PATCH allows you to edit an existing Gist. When a Gist is edited it can start to version changes in documents, allowing you to see how the Gist has developed over time. The PATCH function requires the same inputs as POST.

GET is a function that allows you to get the contents of an existing Gist. It can be any version that is stored within the Gist's history. It requires the URL of a GIST.

These three functions were turned into modules in Grasshopper. They all require a 'metadata' input, which is the users GitHub account name, followed by a token created by GitHub which acts as a password by authenticating the user. Information Security is not considered in this prototype, but will be addressed in the production version.

There are three main workflows for using Minke to POST, PATCH AND GET Gists.

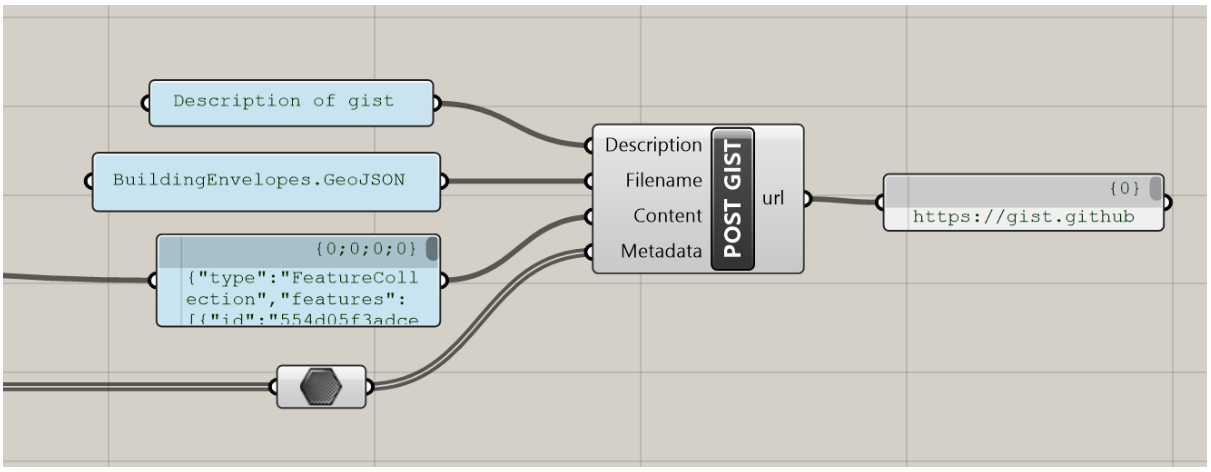


Figure 24: Posting a Gist, the process of patching Gists is similar.

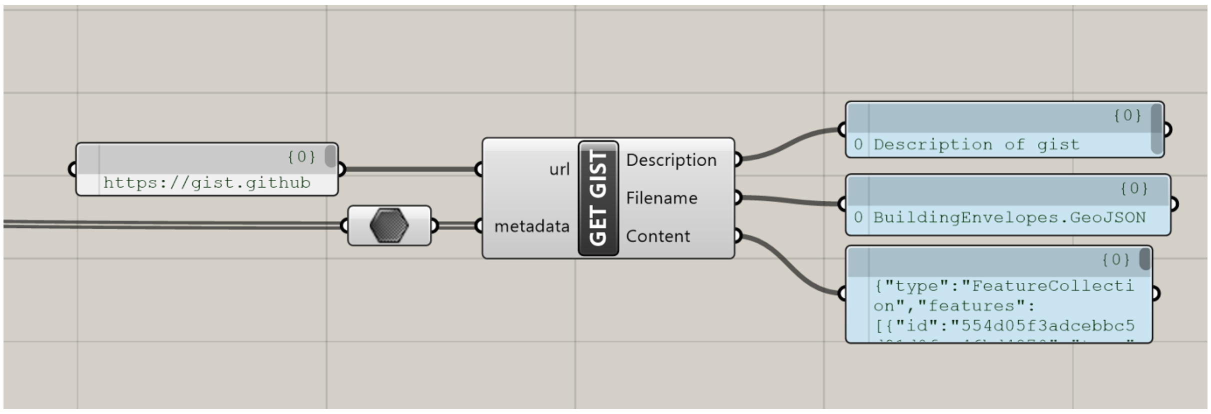


Figure 25: Getting Gists from a URL

Why hasn't distributed version control already been adopted in Law?

Git has been around since 2005, and soon after GitHub was developed in 2008. So with this knowledge, what has been preventing distributed version control to be adopted in legal practice?

This topic was raised on the question and answer website Quora.com, where a user asked “Public Policy: What are the nontechnical barriers to adopting a version control system for use in writing bills and new laws?”

Ari Hershowitz, Director of Open Government, responded with the following venn diagram:

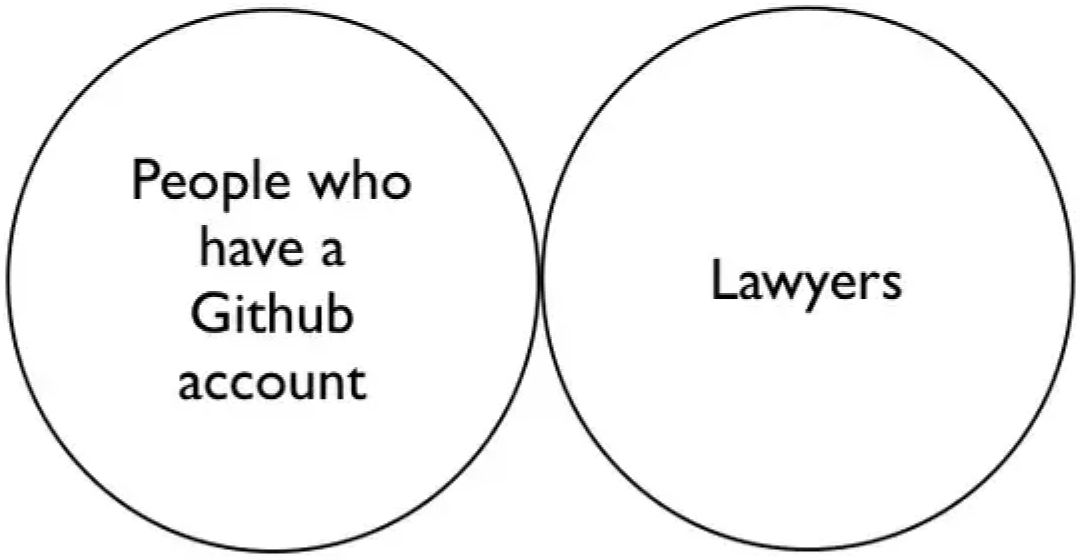


Figure 26: People who have a Github Account vs Lawyers

“If you squint, you might be able to find a couple of intersections, but not many. [...]The legal community is unaware of the powerful text-based tools that could make legal work more accessible to the public and more efficient. Meanwhile, there is no ‘version control’ lobby in Congress. So although adding version control would make a tremendous difference to the efficiency of the legal process, few people understand the value that it would bring.”

Hershowitz, 2011

Lawyersongithub (https://github.com/dpp/lawyersongithub) is a GitHub repository, which as the name suggests, documents the lawyers that have GitHub accounts. To join the lawyer makes a pull request[3] proving you're a lawyer with bar membership. To date there are 28 lawyers on GitHub, 3 lawyers *at* GitHub, and 6 law students on GitHub.

[3] Pull requests let you tell others about changes you've pushed to a GitHub repository

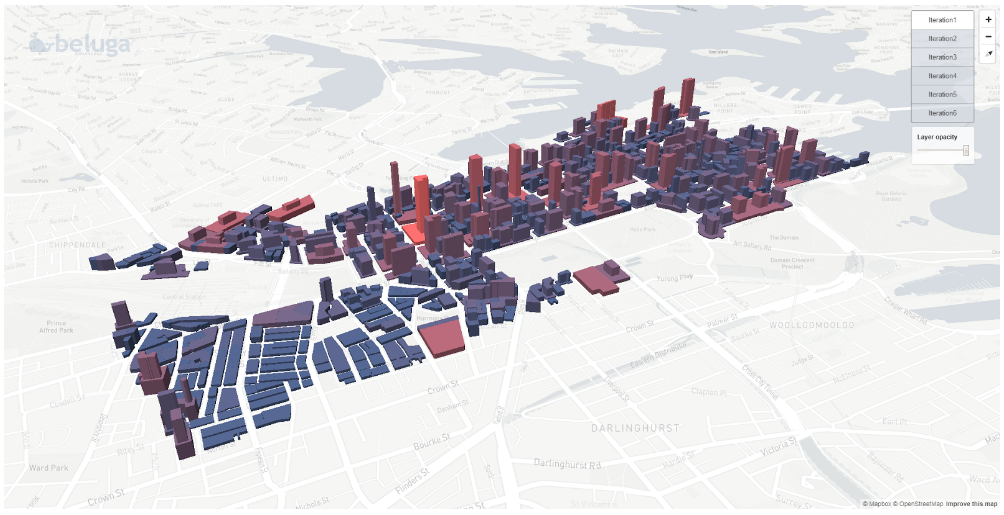
SIGNIFICANCE OF SCRIPTING REGULATIONS

This chapter explores the significance of documenting laws as code by highlighting potential uses of Beluga.

Mass Iterations

The advantage of creating a parametric script, is that those parameters can be changed at will. Beluga allows users to change the values contained by planning controls. This is done through the interface of Rhino, where a user selects a boundary of a planning control and changes its attached value.

To demonstrate the capabilities of Beluga, the following iterations of the CIty of Sydney were made:

Iteration	**Policies**
 beluga Iteration1 Iteration2 Iteration3 Iteration4 Iteration5 Iteration6 Layer opacity CHIPPENDALE WOOLLOOMOOLOO DARLINGHURST Prince Alfred Park Ward Park © Mapbox © OpenStreetMap Improve this map	**FSR:** From LEP **HOB:** From LEP **Sun Access** **Setbacks**

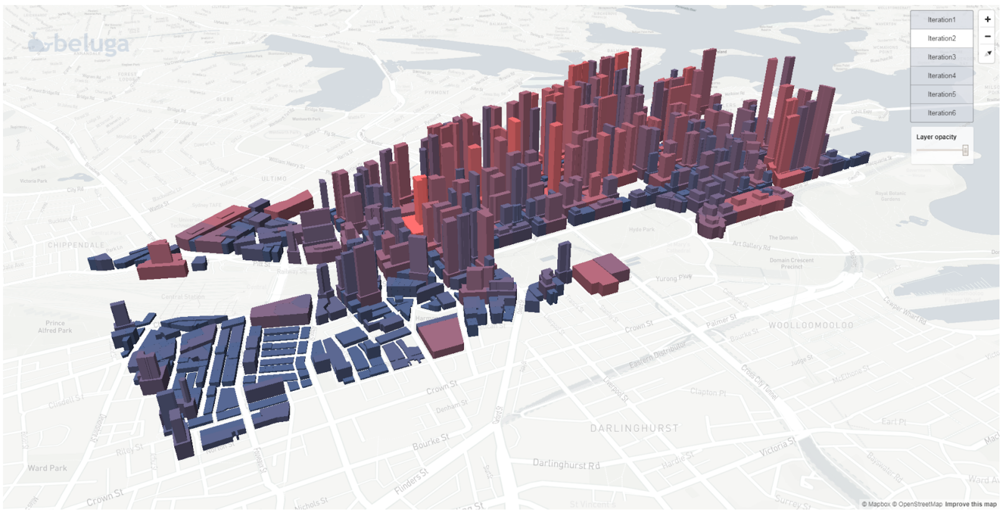
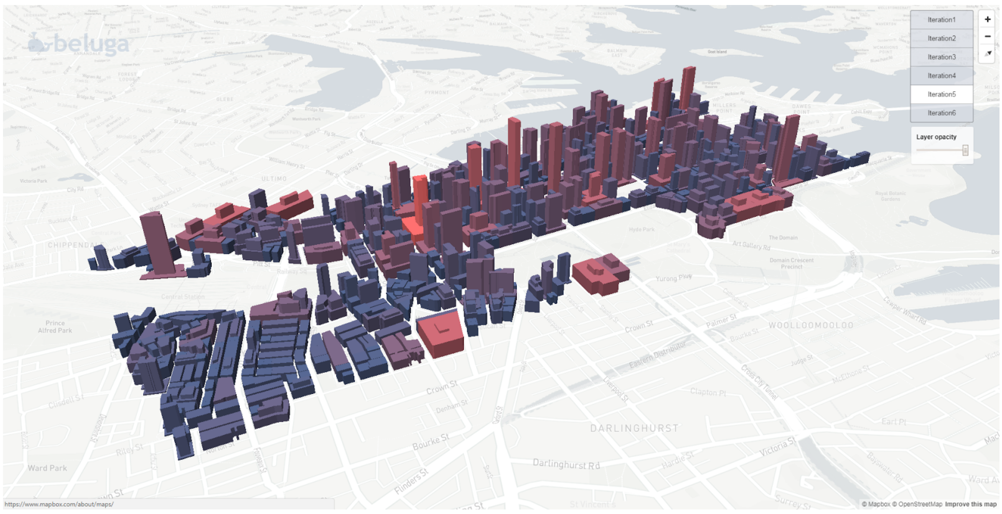
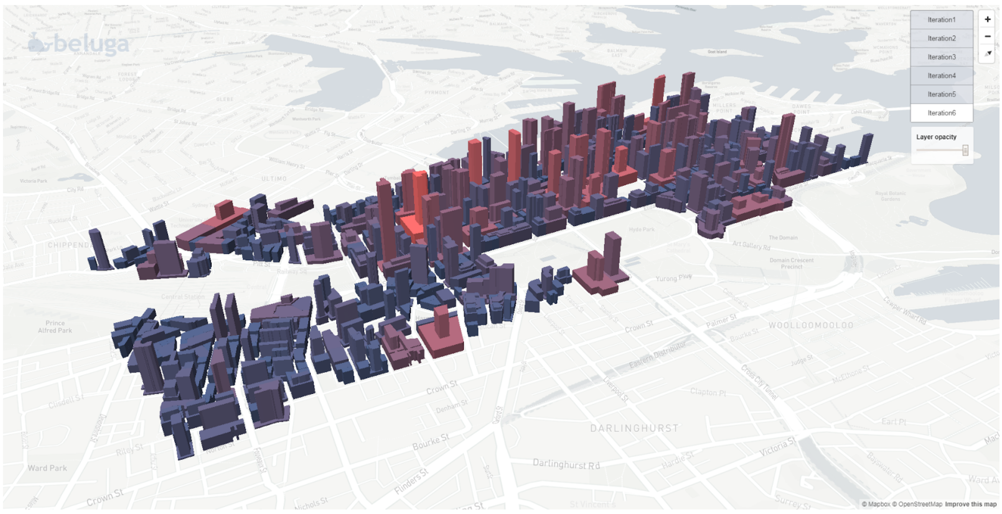
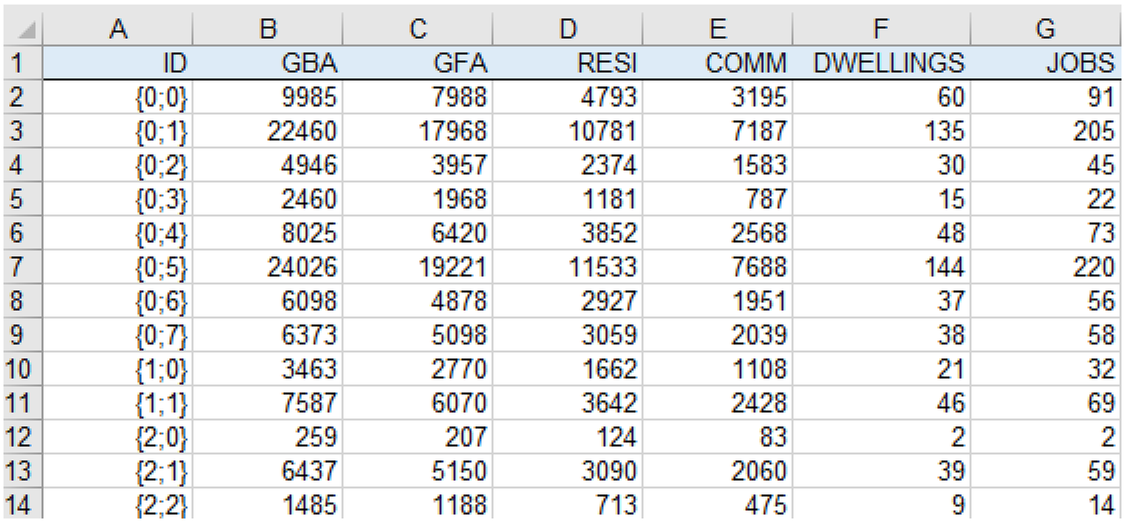
 beluga Iteration1 Iteration2 Iteration3 Iteration4 Iteration5 Iteration6 Layer opacity ULTIMO CHIPPENDALE WOOLLOOMOOLOO DARLINGHURST Darlinghurst Rd © Mapbox © OpenStreetMap Improve this map	**HOB:** From LEP **Sun Access** **Setbacks**
 beluga Iteration1 Iteration2 Iteration3 Iteration4 Iteration5 Iteration6 Layer opacity WOOLLOOMOOLOO DARLINGHURST Darlinghurst Rd https://www.mapbox.com/about/maps/ © Mapbox © OpenStreetMap Improve this map	**FSR:** LEP +5 **HOB:** LEP +30m **Sun Access** **Setbacks**
 beluga Iteration1 Iteration2 Iteration3 Iteration4 Iteration5 Iteration6 Layer opacity WOOLLOOMOOLOO DARLINGHURST Darlinghurst Rd © Mapbox © OpenStreetMap Improve this map	**FSR:** Randomly generated **HOB:** Randomly generated **Sun Access** **Setbacks**

Table 2: Mass Iterations using Beluga

These iterations can be found on https://nazmulazimkhan.github.io/beluga, and show how individual planning control effect a city as a whole making it useful when testing new building policies to foresee any consequences and opportunities.

Statistical Output

Beluga produces floorplates for every building envelope, there are a number of statistics that can be automatically generated from this. On an urban scale, there is leeway for general assumptions in the calculations. Beluga generates the following values which can be exported as a .CSV file that can be opened in any spreadsheet program.



	A	B	C	D	E	F	G
1	ID	GBA	GFA	RESI	COMM	DWELLINGS	JOBS
2	{0;0}	9985	7988	4793	3195	60	91
3	{0;1}	22460	17968	10781	7187	135	205
4	{0;2}	4946	3957	2374	1583	30	45
5	{0;3}	2460	1968	1181	787	15	22
6	{0;4}	8025	6420	3852	2568	48	73
7	{0;5}	24026	19221	11533	7688	144	220
8	{0;6}	6098	4878	2927	1951	37	56
9	{0;7}	6373	5098	3059	2039	38	58
10	{1;0}	3463	2770	1662	1108	21	32
11	{1;1}	7587	6070	3642	2428	46	69
12	{2;0}	259	207	124	83	2	2
13	{2;1}	6437	5150	3090	2060	39	59
14	{2;2}	1485	1188	713	475	9	14

Figure 27: Statistical Output

Gross Building Area (GBA)

The total area in square meters of all the floors in a building. This is prior to architectural considerations such as the setbacks of exterior walls.

Gross Floor Area (GFA)

The total area of usable floor space inside a building. The GFA was calculated to be 80% of the GBA. This is a percentage that can be changed within the script to provides a rough estimate of what the GFA would be.

Residential floor area (RESI) and Commercial floor area (COMM)

Beluga allows users to set a percentage that determines the total area of residential and commercial floor space. The example above calculates the residential floor area as 60% of the GFA, and the commercial area as 40%.

Number of Dwellings and Number of Jobs

Number of dwellings is calculated based on the assumption that for every 80m^2 of residential area there is 1 dwelling. Number of jobs is calculated based on the assumption that for every 35m^2 of commercial area there is 1 job.

While being heuristic, these calculations can help guide policy makers in the process of making laws. They can also help developers understand what sort of statistics to expect when developing a property.

Virtual Reality and Facade generator

The representation of a 3D model plays an important role as to how it's perceived. Typically 3D models in urban planning are represented through diagrams and 2D drawings. The web GIS platform developed for the permissible envelope script, Beluga, (https://nazmulazimkhan.github.io/beluga) demonstrates how an interactive 3D model can give a greater understanding of context than still images.

However, when interviewing Peter Holt he expressed that the ‘fly over’ view of a city is not not a real human experience. “Walking along the street looking up, that's a much more realistic representation. [...] The really interesting thing from the street view is that the human consciousness only allows you to really contemplate ten to twelve stories up. Beyond that it's just something else.”

To make a more realistic representation, a Grasshopper script was developed to generate a simple facade on the permissible building envelopes. The script generated floor slabs, mullions and glazing using the floorplates made by Beluga. The image below is a render taken from a street view. The street view, and basic building elements give an indication of how tall the buildings really are.

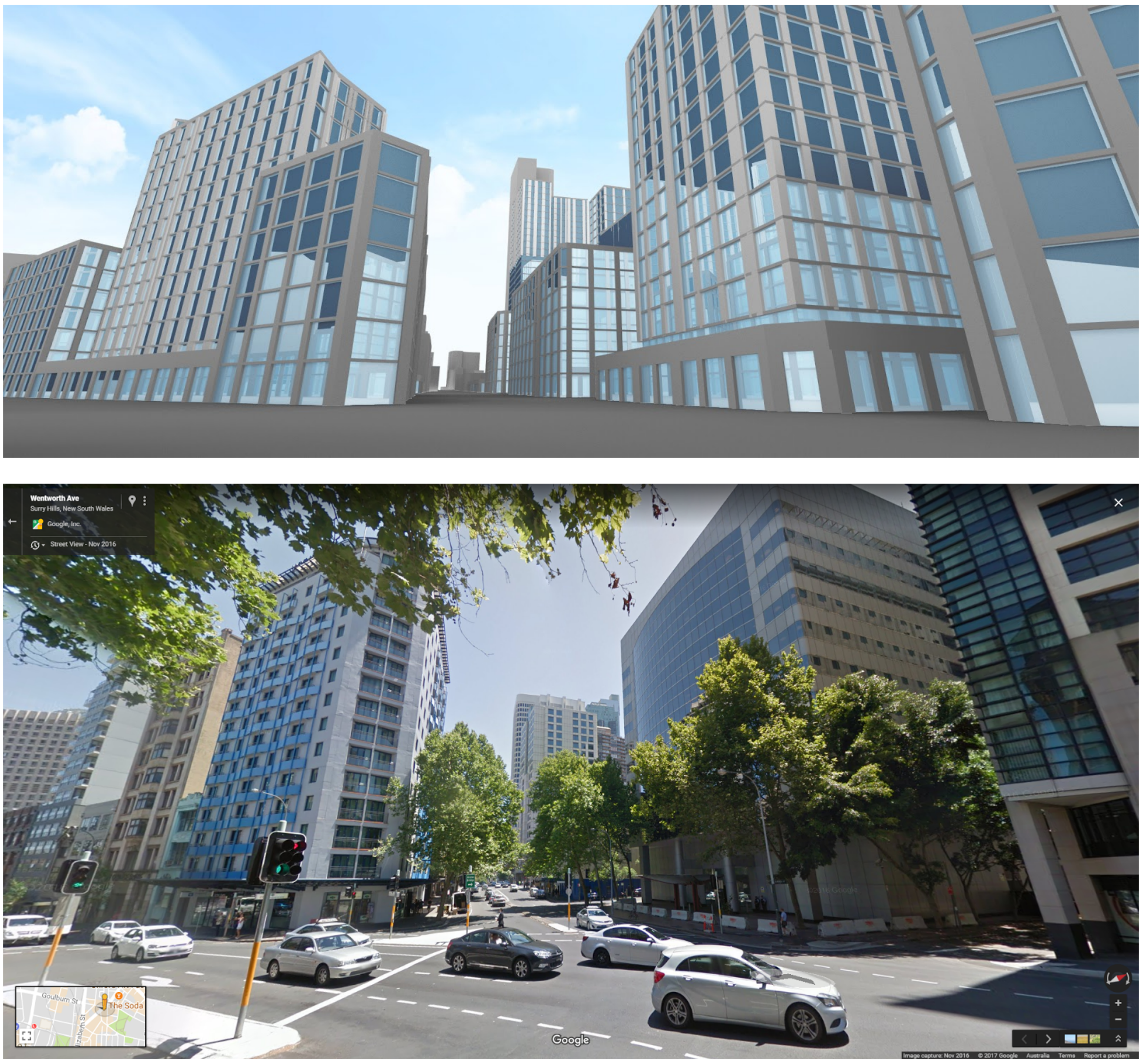


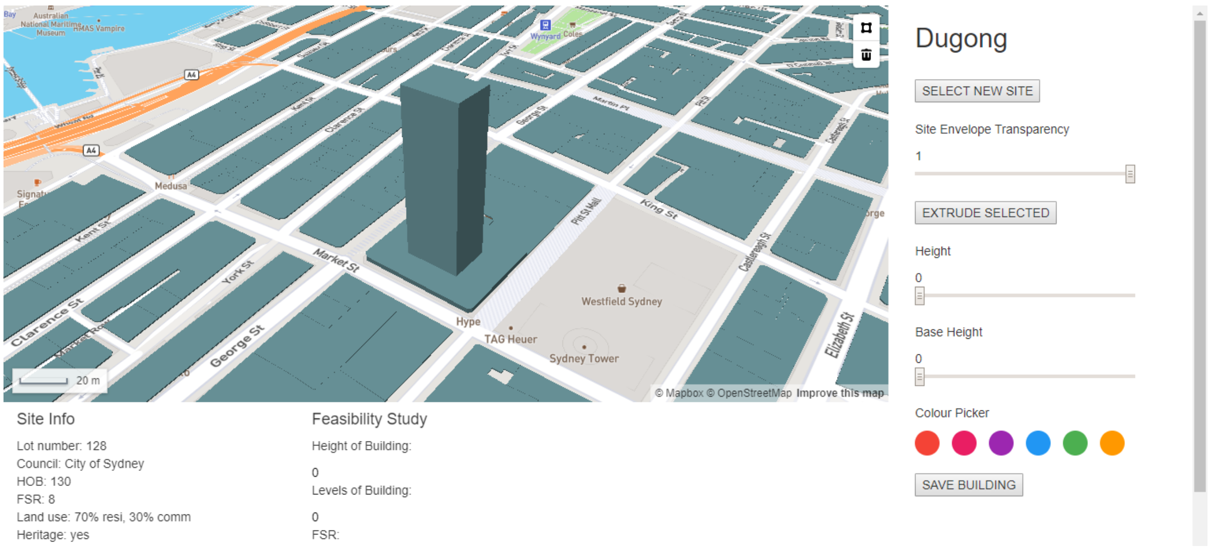
Figure 28: (Top) Facade generator (Bottom) Existing context

To see how the output of Beluga compares to real life, a screenshot was taken from Google Street Maps from the same location. This model could potentially be connected into a Virtual Reality system, where developers and policy makers could explore the space at a 1:1 scale. This could potentially change the way policies are made, as the tool could help analyse architectural problems, that are specific to the ground level, such as circulation from a human perspective. This enables a new type of urban analysis, from a human scale.

Dugong: Decision Support System

Dugong is a interactive website that acts as a decision support tool for urban schemes. It is a platform designed for developers rather than policy makers. The website allows users to ‘sketch’ design proposals and receive immediate feedback of the forms compliance with building policies.

Dugong visualises the permissible building envelopes generated from Beluga to guide users to design within the maximum buildable space.



Dugong

SELECT NEW SITE

Site Envelope Transparency

1

EXTRUDE SELECTED

Height

0

Base Height

0

Colour Picker

SAVE BUILDING

Site Info

Lot number: 128
Council: City of Sydney
HOB: 130
FSR: 8
Land use: 70% resi, 30% comm
Heritage: yes

Feasibility Study

Height of Building:

0

Levels of Building:

0

FSR:

Figure 29: Dugong web app (Johanson, 2017)

Dugong was developed by Madeleine Johanson, in parallel to the research in this paper. It is documented in the paper *Adjudicating by Algorithm: Creating an open web platform for interaction with the law.*

CONCLUSION

Evaluation

This research explores how the concept of ‘law as code’ can benefit the domain of urban planning. The aims of the research were addressed through a script that interprets planning policies and generates permissible building envelopes to effectively communicate the law.

Beluga’s role in this research was to interpret laws to generate permissible building envelopes. By doing so, the script itself can be considered a document of the law. Each and every process that contributes to the generation of the building envelope can be seen. From the perspective of a policy maker, Beluga streamlines testing and simulation of new urban schemes. Parameters within the script can be changed to alter the effects of policies, or new policies can be implemented into the script. This supports a more iterative process to policy making, where potentially hundreds of versions could be tested and evaluated before they are implemented. From the perspective of a developer, Beluga is advantageous because the permissible building envelopes define the buildable space on a site. As policies change over time, the developer will only have to look at the most recent envelope to instantly understand building potential and constraints of a site.

Transparency has been achieved through Minke, the versioning tool, which shows the development and change of building policies over time. As the law is updated, the building envelopes change and are versioned into a repository. This benefits both developers and policy makers as they can see what, when and how something has changed.

Reaching the objective exposed problems in the disciplines of urban planning and computational design. Humpback was developed as a consequence of this, and directly addresses the aim of using interdisciplinary techniques to improve workflows within urban planning. The plugin allows for a bi-directional flow between two softwares, in a streamlined process that wasn't available prior to this research.

Overall, these tools have demonstrated how the integration of law, computational design and urban planning can benefit each other.

Future work

Beluga:

For future development, the following policies could be added to Beluga:

- Heritage - to determine what lots are heritage
- Land Use integration - allowing different envelopes to generate based on its land use
- Ownership - buildings ownerships rules that can merge lots

Topography was not addressed in this research, but topographical considerations could be used to drive how policies are conceived.

Minke:

Minke has not yet been released on Food4Rhino, and further plans for development and functionality are in progress, with an aim to support collaborative workflows. As well as file versioning, Minke could also be used to backup scripts, add comments, history slider, data dam and support multi-file gists. Information Security will be addressed in the production version.

Humpback:

Humpback was released as a plugin for Grasshopper on Food4Rhino in March of 2017. Since its initial release, a new version of Humpback has been uploaded. Humpback 1.1 features minor bug fixes and faster processing components. Currently, Humpback only supports the conversion of polygons. Future work would consider expanding to different geometry types, such as points and lines.

Exposure

Two conference papers have been presented alongside the investigation.

1. **Urban Planning and Property Development Conference (UPPD 2017), Singapore:**
   Presents Humpback as a GeoJSON compatibility tool for Grasshopper and UrbanCoDe as a presentation platform.

2. **Urban Design Conference (UD 2017), Surfers Paradise, QLD:**
   The conversion of law to code, presenting Beluga, Dugong and Minke as computational tools.

Fin

Permissible building envelopes are not the extent of how law as code can benefit urban planning, they are merely one application. This research suggests that while the concept of laws documented as computer code may be considered utopian, adoption of these processes may improve the efficiency of many more disciplines. Although there are many obstacles, with further research, perhaps law as code will no longer be purely speculation.

REFERENCES

AURIN. (2010). The AURIN Journey, Australian Urban Research Infrastructure Network. <https://aurin.org.au/about/the-aurin-journey/> (Accessed: 2nd September 2017)

Bolton, P. and Dewatripont, M. (2005) Contract Theory. MIT Press.

Cassidy, M. (2015) Centaur Chess Shows Power of Teaming Human and Machine, Huffpost. Available at: <http://www.huffingtonpost.com/mike-cassidy/centaur-chess-shows-power_b_6383606.html> (Accessed: 27 September 2017)

Carlile, J. (2017) Flux Metro: What We Learned – FLUX. Available at: https://blog.flux.io/flux-metro-what-we-learned-11fc82b6de03 (Accessed: 7th June 2017).

Central Sydney Planning Strategy (2016) City of Sydney. Available at: <http://www.cityofsydney.nsw.gov.au/__data/assets/pdf_file/0006/260187/160719_PDC_ITEM04_ATTACHMENTA1.PDF> (Accessed: 17th September 2017).

Dalakov, G. (2017) History of Computers and Computing, Dreamers, Ramon Llull. Available at: http://history-computer.com/Dreamers/Llull.html (Accessed: 16 September 2017).

Davis, D., Burry, J., Burry, M. (2011) Untangling Parametric Schemata :Enhancing Collaboration Through Modular Programming. In Designing Together: Proceedings Of The 14th International Conference On Computer Aided Architectural Design Futures

Davis, D., (2015) Why Architects Can't Be Automated, Architect Magazine <http://www.architectmagazine.com/technology/why-architects-cant-be-automated_o> (Accessed 20th March 2017)

Didech, K. (2015) Flux Metro: A Better Way To Visualize Development Code - CodeX - Stanford Law School. <https://law.stanford.edu/2015/03/05/flux-metro-a-better-way-to-visualize-development-code/> (Accessed 7th June 2017)

Drummond, W.J. and French, S.P., 2008. The Future Of Gis In Planning: Converging Technologies And Diverging Interests. Journal of the American Planning Association

Genesereth, M. (2015) Computational Law: The Cop in the Backseat. Available at: http://complaw.stanford.edu/ (Accessed: 1st September 2017)

Geojson (2017) GeoJSON. <http://geojson.org/> (Accessed 4th November 2017)

Golder, B., 2011. RhinoCommon/Grasshopper: A New Class for Geometry with Key/Value Pairs? UserData?. Available at: <http://www.grasshopper3d.com/forum/topics/rhinocommongrasshopper-a-new> (Accessed 21st March 2017)

GISGEOGRAPHY (2017) The Ultimate List of GIS Formats - Geospatial File Extensions, GIS Geography. <http://gisgeography.com/gis-formats/> (Accessed 4th November 2017)

Hershowitz, A. Public Policy: What are the nontechnical barriers to adopting a version control system for use in writing bills and new laws? Available at: <https://www.quora.com/Public-Policy-What-are-the-nontechnical-barriers-to-adopting-a-version-control-system-for-use-in-writing-bills-and-new-laws> (Accessed 4th November 2017)

Hickok, J. A., 2014. A Web GIS Framework for Simulating Pre-incident
Planning, s.l.: s.n.

History.com Staff (2009) CODE OF HAMMURABI. Available at: <http://www.history.com/topics/ancient-history/hammurabi> (Accessed: 3 October 2017).

IBM (2017) IBM100 - Deep Blue, IBM 100. Available at: <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> (Accessed: 3 October 2017).

Jones, R. (2014) Shapefiles, GeoJSON and KML.<http://apprentice.craic.com/tutorials/28> (Accessed 1st May 2017).

Janssen, P., STOUFFS, R., MOHANTY, A., TAN, E. (2016) 'Parametric Modelling with GIS', eCAADe 2016

JSON (2017) JSON. <http://www.json.org/> (Accessed 4th November 2017)

Katz, Y. (2012) Noam Chomsky on Where Artificial Intelligence Went Wrong, The Atlantic. Available at: https://www.theatlantic.com/technology/archive/2012/11/noam-chomsky-on-where-artificial-intelligence-went-wrong/261637/ (Accessed: 1 October 2017).

Kuus, H. (2002) 'Interoperability of Geographical Information Systems'. Tallinn Technical University

Love, N. and Genesereth, M. (2005) ‘Computational Law’, in ICAIL ’05. Available at: http://web.stanford.edu/group/codex/cgi-bin/codex/wp-content/uploads/2014/01/p205-love.pdf (Accessed: 4 September 2017).

Mcaffee, A. And Brynjolfsson, E. (2017) Machine, Platform, Crowd. W. W. Norton & Company, Inc.

Menges, A. And Ahlquist, S. (2011) Computational Design Thinking. John Wiley & Sons.

Musk, E. (2015) Human driven cars may be outlawed because they’re too dangerous, Twitter. Available at: https://twitter.com/elonmusk/status/577946893646364673?ref_src=twsrc%5Etfw&ref_url=https://www.washingtonpost.com/news/the-switch/wp/2015/03/18/elon-musk-human-driven-cars-may-be-outlawed-because-theyre-too-dangerous/ (Accessed: 3 October 2017).

O’brien, R. (1998) Action Research Methodology. Available at: <http://web.net/~robrien/papers/xxar final.html > (Accessed 16th August 2017).

Open Definition (2017) Open Definition 2.0. Available at: <http://opendefinition.org/od/2.0/en/> (Accessed 22nd August 2017).

Perry, M. (2014) ‘iDecide: the legal implications of automated decision-making’. Available at: http://www.fedcourt.gov.au/digital-law-library/judges-speeches/justice-perry/perry-j-20140915 (Accessed: 12 May 2017).

Poulshock, M. (2016) HammurabiProject. Available at: <https://github.com/mpoulshock/HammurabiProject/wiki/Project-rationale > (Accessed 11th August 2017).

Thomson, J (1985) The Trolley Problem. The Yale Law Journal, Vol. 94, No. 6. The Yale Law Journal Company, Inc.

Wheatley, M. (2017) realtybiznews. Available at: <http://realtybiznews.com/envelope-lands-2m-funding-to-build-out-zoning-analysis-tool/98741387/ > (Accessed 30th May 2017).

Woodbury, R. (2010) Elements of Parametric Design, Abingdon: Routledge.

Saulwick,J.,Visentin,. (2016) CBD boom time: City of Sydney says the only way is up. Available at: <http://www.smh.com.au/nsw/cbd-boom-time-city-of-sydney-says-the-only-way-is-up-20160713-gq4vhb.html> (Accessed 4th March 2017).

Senske, N. (2005) ‘Fear Of Code: An Approach To Integrating Computation With Architectural Design’, Massachusetts Institute of Technology.

Shirky, C. (2012) Clay Shirky: How the Internet will (one day) transform government, TED Talk <https://www.ted.com/talks/clay_shirky_how_the_internet_will_one_day_transform_government> (Accessed 3rd October 2017).

Winograd, T. (1972) ‘Understanding Natural Language’, Systems Research and Behavioral Science. Academic Press, Inc.

Zetlin, M. (2010) Doc or Docx? Which Office Format to Use. <https://www.inc.com/software/articles/201002/doc.html> (Accessed 3rd July 2017).