MeasureIt-ARCH:

A Tool for Facilitating Architectural Design in the Open Source Software Blender

by

Kevan Cress

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Architecture

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis discusses the design and synthesis of MeasureIt-ARCH, a GNU GPL licensed software add-on developed by the author in order to add functionality to the Open Source 3D modeling software Blender that facilitates the creation of architectural drawings. MeasureIt-ARCH adds to Blender simple tools to dimension and annotate 3D models, as well as basic support for the definition and drawing of linework. These tools for the creation of dimensions, annotations and linework are designed to be used in tandem with Blender's existing modelling and rendering toolset. While the drawings that MeasureIt-ARCH produces are fundamentally conventional, as are the majority of the techniques that MeasureIt-ARCH employs to create them, MeasureIt-ARCH does provide two simple and relatively novel methods in its drawing systems. MeasureIt-ARCH provides a new method for the placement of dimension elements in 3D space that draws on the dimension's three dimensional context and surrounding geometry order to determine a placement that optimizes legibility. This dimension placement method does not depend on a 2D work plane, a convention that is common in industry standard Computer Aided Design software. MeasureIt-ARCH also implements a new approach for drawing silhouette lines that operates by transforming the silhouetted models geometry in 4D 'Clip Space'.

The hope of this work is that MeasureIt-ARCH might be a small step towards creating an Open Source design pipeline for Architects. A step towards creating architectural drawings that can be shared, read, and modified by anyone, within a platform that is itself free to be changed and improved. The creation of MeasureIt-ARCH is motivated by two goals. First, the work aims to create a basic functioning Open Source platform for the creation of architectural drawings within Blender that is publicly and freely available for use. Second, MeasureIt-ARCH's development served as an opportunity to engage in an interdisciplinary act of craft, providing the author an opportunity to explore the act of digital tool making and gain a basic competency in this intersection between Architecture and Computer Science.

To achieve these goals, MeasureIt-ARCH's development draws on references from the history of line drawing and dimensioning within Architecture and Computer Science. On the Architectural side, we make use of the history of architectural drawing and dimensioning conventions as described by

Mario Carpo,[1] Alberto Pérez Gómez[2] and others, as well as more contemporary frameworks for the classification of architectural software, such as Mark Bew and Mervyn Richard's BIM Levels framework,[3] in order to help determine the scope of MeasureIt-ARCH's feature set. When crafting MeasureIt-ARCH, precedent works from the field of Computer Science that implement methods for producing line drawings from 3D models helped inform the author's approach to line drawing. In particular this work draws on the overview of line drawing methods produced by Bénard Pierre and Aaron Hertzmann,[4] Arthur Appel's method for line drawing using 'Quantitative Invisibility,'[5] the techniques employed in the Freestyle line drawing system created by Grabli et al. [6] as well as other to help inform MeasureIt-ARCH's simple drawing tools.

Beyond discussing MeasureIt-ARCH's development and its motivations, this thesis also provides three small speculative discussions about the implications that an Open Source design tool might have on the architectural profession.

We investigate MeasureIt-ARCH's use for small scale architectural projects in a practical setting, using it's toolset to produce conceptual design and renovation drawings for cottages at the Lodge at Pine Cove. We provide a demonstration of how MeasureIt-ARCH and Blender can integrate with external systems and other Blender add-ons to produce a proof of concept, dynamic data visualization of the Noosphere installation at the Futurium center in Berlin by the Living Architecture Systems Group. Finally, we discuss the tool's potential to facilitate greater engagement with the Open Source Architecture (OSArc) movement by illustrating a case study of the work done by Alastair Parvin and Clayton Prest on the WikiHouse project, and by

[1]Mario Carpo, "Building with Geometry, Drawing with Numbers," in *When Is the Digital in Architecture?*, ed. Andrew Goodhouse and Canadian Centre for Architecture (Montréal: Canadian Center for Architecture, 2017), 35–44

[2]Alberto Pérez Gómez and Louise Pelletier, *Architectural Representation and the Perspective Hinge* (Cambridge, Mass: MIT Press, 1997)

[3]BSI Standards Limited, "PAS 1192-2:2013," n.d., http://www.hfms.org.hu/web/images/stories/PAS/PAS1192-2-BIM.pdf.

[4]Pierre Bénard and Aaron Hertzmann, "Line Drawings from 3D Models: A Tutorial," *Foundations and Trends® in Computer Graphics and Vision* 11, nos. 1-2 (2019): 1–159, https://doi.org/10.1561/0600000075.

[5]Arthur Appel, "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," in *Proceedings of the 1967 22nd National Conference on* - (The 1967 22nd national conference, Washington, D.C., United States: ACM Press, 1967), 387–93, https://doi.org/10.1145/800196.806007.

[6]Stéphane Grabli et al., "Programmable Rendering of Line Drawing from 3D Scenes," *ACM Transactions on Graphics* 29, no. 2 (March 1, 2010): 1–20, https://doi.org/10.1145/1731047.1731056.

highlighting the challenges that face OSArc projects as they try to produce Open Source Architecture without an Open Source design software.

Figure 1: *A Plan Drawing Made in Blender, With and Without MeasureIt-ARCH (Left) Plan created with MeasureIt-ARCH. (Right) Plan Created without MeasureIt-ARCH*

Acknowledgements

There are many people without whose help and support this thesis would not have been possible. Firstly a thank you to my committee. To my supervisor Philip Beesley, for his enthusiasm, knowledge, and for always inspiring new connections and possibilities as I worked. To Maya Przybylski, my committee member, for her support and guidance, especially during the early stages of exploration. To my internal reader, Craig Kaplan, for his eye for detail and for bringing a Computer Science perspective to the work, with feedback and references that will be immeasurably valuable as I continue forward. And to my external reader Patrick Harrop, for his insightful questions on the nature of digital craft and its relation to the work.

Secondly, at this academic milestone, I'd like to take a moment to thank everyone who has contributed to my education over the past six years — especially the faculties of both the McEwen and Waterloo Schools of Architecture. Beyond the academic, I'd also like to give special thanks to Alex Strachan and Daniel Viirtela for the opportunities they've provided over the years.

Lastly, but certainly not least, a heartfelt thank you to all of my friends and family. You have given me more support than you could ever know.

A Note on Production

In keeping with the core principles of this thesis work, as much of this document as possible has been produced using Open Source software. The development of MeasureIt-ARCH has been tracked and made publicly available through GitHub; it's history can be found at (https://github.com/kevancress/MeasureIt-ARCH). The thesis text was written in the markdown language using the Atom text editor. The raw markdown was converted to any necessary formats using Pandoc while typesetting and formatting were controlled and automated with the LaTex document preparation system to produce the PDF or print publication you are currently reading. Citations and references were managed using the Zotero reference management software.

Unless explicitly stated, all original diagrams presented in this thesis are produced using Blender and MeasureIt-ARCH.

This thesis would not be the work that it is without the passionate work of countless Open Source developers who have volunteered their time and effort to produce highly versatile tools for the benefit of the broader public. They cannot be thanked enough.

A Note on Performance Metrics

In a few places throughout this thesis, timing benchmarks are provided as comparative metrics for the performance of certain operations in MeasureIt-ARCH. All of these tests were performed on the same computer, with the following specifications;

Operating system: Windows-10 Version 10.0.18363 Build 18363

Graphics card: NVIDIA GeForce GTX 1050

Processor: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, 2304 Mhz, 4 Core(s), 8 Logical Processors

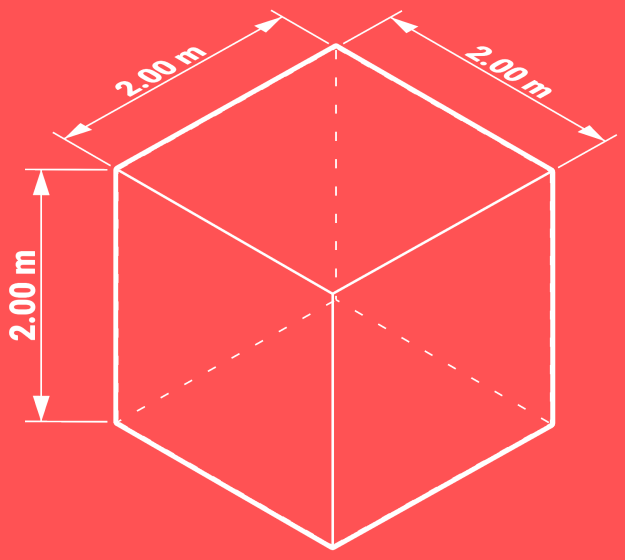
Installed Physical Memory (RAM): 16.0 GB

Table of Contents

List of Figures

**”Invent your own tools.
Adopt new technologies,
rewrite the code,
and collaborate in
unexpected ways.”**

- PARTISANS [7]



[7]PARTISANS, *Rise and Sprawl: The Condominiumization of Toronto*, ed. Hans Ibelings and Nicola Spunt (Montreal Amsterdam: The Architecture Observer, 2016).

Introduction

This thesis explores the craft of code and digital tool making by developing a specialized software tool MeasureIt-ARCH and by framing this practical work within a discussion of related aspects of dimensioning and drawing in architectural practice. . MeasureIt-ARCH was developed in response to two key goals.

First, to help myself learn and gain competency in the craft of digital tool making.

Second, to create a functioning piece of free and Open Source software to allow for the creation of architectural drawings, that could be made available to anyone.

The Craft of Digital Tool Making

As a student of architecture, I've always been fascinated by the digital tools we increasingly rely on for our everyday design work. I have been intrigued by how they function, and how they could improve. I have been amazed by their efficiency, but frustrated at their 'black box' nature, their source code and methods hidden away and guarded by the companies that license them to us. This interest in our digital tools has led to a desire to truly engage with and design these tools for myself, and to share that process. The exploration of craft in this work tries to embody the ethos of the craftsperson as described by Richard Sennett in his appropriately titled book 'The Craftsman.'[8] Sennet describes craft as;

> *"the special human condition of being engaged."*[9]

The text evokes an expanded state of engagement that is achieved not only through the act of physical making, but also through cyclical acts of problem solving and problem finding, exploring often-ambiguous areas of study and seeking an intuitive understanding through practice and creation. The thesis represents a modest participation within the vast craft that is digital tool making.

[8] Richard Sennett, *The Craftsman* (New Haven: Yale Univ. Press, 2008)
[9] ibid., 44

Creating a Functioning Open Source Tool for Architectural Drawing.

The exploration and learning of the first goal was tempered and guided by the concrete goal of creating a functioning tool for the creation of architectural drawings. This tool is MeasureIt-ARCH. MeasureIt-ARCH's contributions include a simple set of tools for generating linework, annotations and dimensions in the Open Source 3D modeling software Blender, designed to allow for the creation of conventional architectural drawings. MeasureIt-ARCH contributes to the culture of distributed Open Source development in the architectural profession, providing a freely available Open Source software platform that is capable of producing architectural drawings.

MeasureIt-ARCH implements two novel technical methods in its drawing system. The first of these methods allows for the dynamic placement of dimensions in 3D space, and the second is a method for drawing object silhouette lines by transforming a duplicate of the object's geometry in 4D Clip Space.

MeasureIt-ARCH attempts to solve the problem of placing dimensions in 3D space in an intuitive and legible way . A common practice in today's industry-standard Computer-Aided-Design (CAD) or Building Information Modeling (BIM) tools is to rely on a manually defined two dimensional 'work plane' upon which the dimension is placed. That practice tends to remove dimension elements from 3D space, reducing them to two dimensions. MeasureIt-ARCH explores an alternative to conventional 'work plane' based methods by drawing on contextual information about the object that the dimension is attached to in order to determine a placement within 3D space that optimizes the dimension's legibility. This placement system makes use of the model's surface normals and the user's current viewpoint to determine a dimension's location dynamically whenever the scene is drawn.

MeasureIt-ARCH also presents a novel approach to silhouette line drawing. This method is similar to the Inverse Hull method commonly used for Non-Photorealistic Rendering (NPR) in games and animation.[10] The Inverse Hull method creates an inverted copy of the model's geometry, expanded along its surface normal to produce a silhouette effect. In contrast, MeasureIt-ARCH offsets the duplicated geometry along the Camera Space Z-Axis by a user specified threshold. This allows the user to vary the types of silhouette lines being drawn by MeasureIt-ARCH's silhouette system.

[10] Junya Christopher Motomura, "GuiltyGear Xrd's Art Style: The X Factor Between 2D and 3D" (Game Developers Conference, Moscone Center - San Francisco, 2015), 24, http://www.ggxrd.com/Motomura_Junya_GuiltyGearXrd.pdf

Typically this method would function poorly in a perspective view, as perspective projection would distort the duplicated geometry resulting in a distorted silhouette. MeasureIt-ARCH works around this problem of perspective distortion by conducting its transformation of the silhouette geometry in 4D Clip Space. This technique allows the method to obtain consistent silhouettes in orthographic and perspective views.

In addition to discussing in more detail how these two technical challenges were overcome in MeasureIt-ARCH, this thesis also explores motivations that inspired MeasureIt-ARCH's creation, the history and analysis that defined its scope, the affordances that Blender might have for architectural design, and potential implications that an Open Source tool might have for small practices and Open Source architectural movements. The thesis describes what MeasureIt-ARCH does and how it functions, the reasons for its design, and its development process.

MeasureIt-ARCH is shared publicly on GitHub.[11]

[11] Antonio Vazquez and Kevan Cress, *MeasureIt-ARCH*, 2018, https://github.com/kevancress/MeasureIt-ARCH.

Sections

The thesis is divided into six sections. The first provides an exploration of the motivations that inspired the software tool. The second describes the specification and scope that guide MeasureIt-ARCH's development. The third explores why Blender was chosen as a base for MeasureIt-ARCH and what affordances are offered for architectural design in Blender's existing feature set. The fourth discusses features, development and benchmarks. The fifth reviews specifications previously identified in section one in order to evaluate MeasureIt-ARCH's current state and explore what lies ahead for future development. The final section includes three investigations that explore use cases and future implications.

Section 1: Motivating an Open Source Architectural Software.

This section provides a brief overview of three distinct areas of interest that have motivated my desire to create MeasureIt-ARCH. This section briefly discusses the culture of distributed development found in the animation industry, the Open Source Architecture movement, and the author's own experience using the Open Source software Blender for architectural design.

Section 2: Requirements and Specifications

The Requirements and Specifications Section provides a short history of the development of architectural drawing conventions. Included is a review of standards that govern the style and legal importance of numeric dimensions in current architectural practice.

In addition to understanding the general requirements of architectural drawings, the thesis integrates the BIM classification system authored by Mark Bew and Mervyn Richards as a framework to identify broad categories of software used in architectural practice. These categories, along with a comparative analysis of the feature sets of other industry standard AEC software packages, are used to situate MeasureIt-ARCH's intended scope and feature set and to prepare a detailed specification of features for MeasureIt-ARCH.

Section 3: Blender and Architecture

Section three provides a discussion of the aspects of the Open Source software Blender that make it suitable as a base for the development of the features outlined in our specification. We discuss Blender's development model, notable features, and cite a review of the software conducted by Theodoros Dounas and Alexandros Sigalas in 2009. This discussion illustrates both potential affordances and shortcomings of Blender for architectural projects, providing a practical context for the development of the new features of Measureit-ARCH.

Section 4: MeasureIt-ARCH

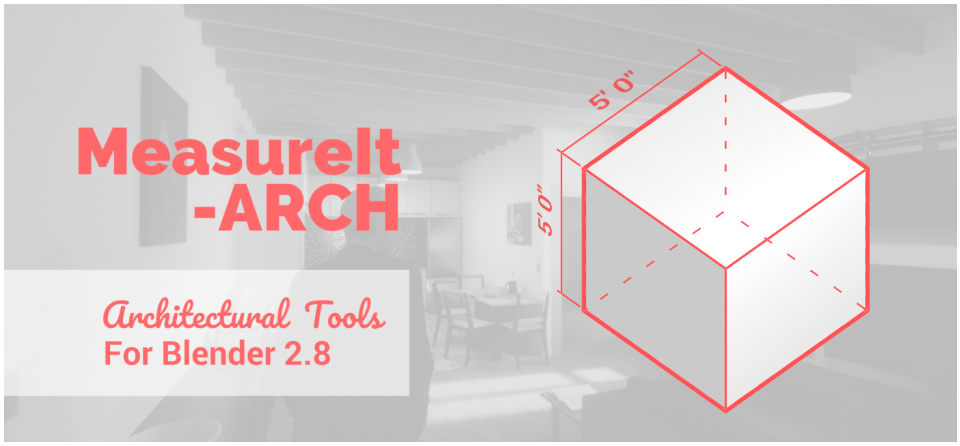


Figure 2: *MeasureIt-ARCH's GitHub Release Banner Image*

Section four describes how MeasureIt-ARCH builds on the foundation of Antonio Vazquez's MeasureIt add-on to develop a more comprehensive toolset for architectural drawing, and explains how MeasureIt-ARCH integrates with and utilizes the 3D nature of Blender's modelling environment to produce a workflow for small design projects. We provide a detailed analysis of MeasureIt-ARCH's linework and dimensioning implementations, along with precedent research in the field of computer science that informed these features.

In the discussion of MeasureIt-ARCH's dimensioning tools, we present an analysis of the conventional work plane based dimensioning systems present in industry-standard tools, identifying their challenges and failure cases in their implementation in Rhino and Revit. Providing an

alternative that integrates dimensions within three-dimensional design space, MeasureIt-ARCH uses an algorithm that allows the user to place multiple dimensions simultaneously, without the need to predefine a work plane. It accomplishes this by calculating the dimension's optimal positioning related to the user's viewpoint and the topology of the geometry that the dimension is attached to.

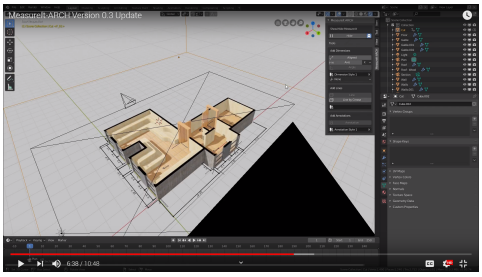


Figure 3: *MeasureIt-ARCH in Action*
Still from the 'MeasureIt-ARCH V0.3 Update Video', published on YouTube

Finally, in the spirit of Open Source development, and to foster discussion and testing around the tool, we discuss the 'Version 0.1' and 'Version 0.3' releases of MeasureIt-ARCH. These open releases and the resulting user feedback helped contribute to solutions for significant issues in the cross-operating system compatibility of the tool, and helped to identify bugs, leading to the improved stability of the tool.

Section 5: Specification Evaluation

We evaluate how MeasureIt-ARCH's current implementation measures up to the specification presented in the first section. For each identified feature, we discuss the successes and shortcomings of its current state and show performance benchmarks. We also provide brief notes on how features might be improved in future development.

Section 6: Testing and Implications

We look beyond the creation of MeasureIt-ARCH, exploring impacts that the Blender and MeasureIt-ARCH, as an Open Source design platform, might have on the architectural profession. We demonstrate MeasureIt-ARCH's application in two widely varied practical implementations. We illustrated MeasureIt-ARCH's implications for small scale architectural projects, in a case-study design of several small cottages developed for the Lodge at Pine

Cove located in French River, Ontario. In parallel, data tied experimental lightweight digitally fabricated constructions are explored in collaboration with the Living Architecture Systems Group's recent installation entitled 'Futurium Noosphere', installed in 2019 at the Futurium facility in Berlin, Germany. This collaboration explores the potential of MeasureIt-ARCH and Blender to integrate with external data sources used in the Futurium installation, and visualize this data through MeasureIt-ARCH's annotations. This is accomplished by combining MeasureIt-ARCH with an additional add-on created for the handling of external data through the Open Sound Control (OSC) message protocol. Finally we provide some reflection on how an Open Source tool like MeasureIt-ARCH might impact OSArch projects.

SECTION 1:

Motivating an Open Source Architectural Software

Motivating an Open Source Architectural Software.

As a profession, Architecture's relationship with the software that enables its creation is often one-sided. We are reliant on those that produce the software we use, and change and innovation in this software generally come from outside the AEC industry, from the technology companies that craft the tools we use in our day to day work.

Now, more technologically invested readers may find this statement off-putting. No doubt, the architectural profession has its share of innovators. They cannot be denied and should not be undervalued. There is a vibrant community of academics and cutting-edge practices that develop new methods and innovate *within the bounds of existing software*, to embrace the technological revolution that is currently overturning our conceptions of fabrication and construction. One doesn't need to look farther than the conference proceedings that emerge from institutions like eCAADe, CADRIA or ACADIA, or the fabrication labs of most schools of architecture, to see evidence of the experimentation and the potential for technological advancement in the architectural research that is carried out by these dedicated pioneers.

And yet, despite this dedicated and robust community of innovators and academics, the translation of these innovations from research to practice is slow in the collective professional entity that is the Architecture, Engineering and Construction (AEC) Industry. Of course, there are a multitude of factors that contribute to the pace of development in this profession. Short deadlines, tight budgets, and high risk have made it difficult for the average architectural Practice to experiment or innovate on the job. More challenging still, the interdisciplinary nature of the AEC Industry necessitates fluent exchange and smooth communication, leading to an almost darwinian selection of a few software packages which dominate across major segments of the industry. While having a few dominant 'industry standard' tools provides stability for the industry it can make it difficult for smaller, more experimental tools to integrate into larger workflows, and expecting the level of technological skill necessary for every architect to be capable of wrangling experimental software into service in their day to day work seems unreasonable. Especially in a profession whos education already demands a multidisciplinary understanding of design, philosophy, art history, statics, and sustainability, not to mention the practicalities of creating buildings that comply with local requirements and regulations.

Where though, does a modest Open Source tool for architectural drawing sit in all this?

MeasureIt-ARCH, as a simple tool, cannot answer this challenge, this gap between architectural innovation and architectural practice, but it does draw its inspiration from it. In response to this challenge, it pulls motivations from the culture of innovation and distributed development present in the animation industry, the growing enthusiasm behind the Open Source Architecture (OSArc) movement, and my own experience using the Open Source software Blender in small scale design work. And while the body and scope of this thesis deal predominantly with details of MeasureIt-ARCH's development and its modest toolset, it's worth taking a moment to explore these motivations to set the stage with the grander aspirations, to which MeasureIt-ARCH is only a small step.

Distributed Development and Innovation.

How might an Open Source architectural design software start to mobilize a change in architecture's relation to software, in a way that helps to close this gap between architectural innovation and practice? If we aren't expecting the average architect to have the software literacy necessary to read and write code, then how might an Open Source software motivate bottom-up software development and change in the AEC industry? To see an image of how this change might play out, it is useful to look at the culture of innovation and distributed development at play in the animation industry, and how Open Source development, and the more recent shift to Open Source tools has affected it.

The animation industry itself bears some passing resemblance to the early phases of an architectural design. Both involve the spatial design and composition of environments, that will be inhabited by actors. Both require the active co-ordination and exchange of assets across multiple departments with different focus and scope, and although animators are not limited by the fetters of having to physically construct their creations, they face other technical limitations imposed by the available computing resources that must be overcome in order to ensure that projects are completed.

The necessity to deliver products that are increasingly computationally intense means that studios involved in animation are constantly innovating to make the best use of new hardware resources or to produce more appealing visuals than their competitors. This intense innovation has resulted in a company structure at many studios that includes a dedicated

research and development department, responsible not only for keeping pace with ever-evolving hardware but also for developing artist-friendly software tools. The work of this technical team allows the creative departments at the studio to focus on the design work they do best. Allowing artists to take advantage of new developments and software without burdening them with the need to have the technological literacy required to create and maintain that software themselves.

These tools can be reserved for in house use, due to their bespoke nature or to keep a competitive edge. But occasionally large studios, like Disney or Pixar, will release key developments and software libraries for the good of the industry as a whole under Open Source Licenses (Disney's Principled shading model and Pixar's Open Subdivision technology are prime examples of this)[12][13]. This allows other studios R&D departments to incorporate these advances into their own tools, and provides an opportunity for other Open Source software platforms, to keep pace with the 'industry standard' that is being actively developed within the industry itself.

However, some studios have taken this approach a step further. Rather than focusing on bespoke in-house tools, they've adopted fully Open Source software platforms as their primary design tool. This change can make it simpler for new tools to be developed and shared with the rest of the industry. As each studios R&D department develops new tools and improves those that already exist, their new developments can be folded back into the shared Open Source platform and made available for others to use. Since these developments are folded into a shared platform, other studios can make use of these developments as soon as they are committed to the code base. This type of development model can be seen in Toronto's Tangent Animation Studios. Tangent Animation uses Blender, an Open Source 3D modelling tool that plays a key role in this thesis work, in conjunction with many other tools like Houdini and Substance Painter to realize their animated features. To improve their workflow, Tangent's Senior Software Engineer, Stefan Werner has worked in tandem with the Blender Development team, to add several improvements to the Open Source platform, including but not limited to, support for the Intell Embree raytracing kernel, Cryptomatte compositing masks, and early support for the OpenVDB library to better

[12] Brent Burley and Walt Disney Animation Studios, "Physically-Based Shading at Disney," n.d., 26, https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf.

[13] Tony DeRose, Michael Kass, and Tien Truong, "Subdivision Surfaces in Character Animation," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '98* (The 25th annual conference, Not Known: ACM Press, 1998), 85–94, https://doi.org/10.1145/280814.280826.

facilitate the exchange of fluid and smoke simulation data between Houdini and Blender.[14]

This culture of distributed 'in-house' development is not dependant on the existence of a monolithic Open Source software platform like Blender, the sharing of innovative developments as smaller software libraries, released under Open Source licenses was, and is still, commonplace, not only in the animation industry but in the Open Source software world at large. What an Open Source Platform adds to this mix is the opportunity to distribute these developments and make them available to a wider audience. It moves the burden of implementation off of the individual or studio that wishes to use the released library and makes it a shared responsibility of the development team and the maintainers of the Open Source platform. This makes each successive development more accessible, as their inclusion in a common platform makes them ready for use across the industry.

Adopting this style of distributed development in the AEC industry would require a substantial reimagining of how architectural firms are structured, but has the potential to shift how innovation occurs in the industry. Investing solely in software from external vendors means relying on those external vendors to deliver innovations from the top-down. Adopting a free Open Source design platform, on the other hand, could provide architectural firms with the opportunity to redirect their financial investment, from licensing commercial software, towards creating dedicated research and development teams within their organization. A change like this could help drive bottom-up innovation from within the industry itself. Of course, for this shift to be feasible, there must first be an Open Source software platform that is capable of meeting the needs of the AEC industry. Once such a platform can meet the basic needs of the industry, or a small segment of the industry, it may serve as a base on which further bottom-up, architect driven development could occur.

And yet, even without an Open Source software platform on which to work, some architects are still drawn to the ideals of the Open Source movement. Though their interest in the Open Source comes from another perspective entirely.

[14]Stefan Werner, "Developer.Blender.Org Stefan Werner's Account History," accessed October 3, 2019, https://developer.blender.org/p/swerner/

The Open Source Architecture Movement

The original Open Source movement, can largely be attributed to the work of Linus Torvalds and his Linux operating system kernel, which he released for free in 1991. The free release of Linux would prove to be a spark for change in the practices of software development and beyond. Most notably Netscape (now known as Mozilla Firefox) would release their formerly proprietary web browser and its source code to the public in 1998 and coin the term "Open Source". This Open Source spark would inspire other developers to begin producing and releasing their software for free as well. Creating an 'Open Source' movement and methodology that would spread far beyond software development.

There is more to Open Source than simple the 'free' release of software, however. Typically Open Source refers to the practice of producing and releasing software *and its source code* to the public. This means that not only can anyone use the software, but they can modify and redistribute it as well. Often Open Source works are also protected legally by 'copyleft' licences. These copyleft licences, like the GNU GPL licence, are a manipulation of traditional copyright law, used to ensure that the original Open Source work, as well as any derivative works that modify or build on the original must be Open Source as well. Utilising copyleft licenses helps ensure that the work of Open Source developers isn't exploited for commercial gain, a protection that these works would not have if they were released into the public domain.

This radical form of protected sharing has no doubt inspired interest even outside the realm of software development, reaching as far as the realm of architectural theory. The Open Source Architecture (OSArc) movement has drawn inspiration from the collaborative authorship of Open Source software and is trying to co-opt some of the movement's ideas in order to re-ignite the ideals of Participatory Architecture championed by the likes of Christopher Alexander, Giancarlo DeCarlo and Cedric Price in the mid-1900s. Facilitating their ideals of a participatory design process with new information technologies.

> "*[the] traditional approach to the reconciliation of social values and individual choice is to entrust de facto decision-making to the wise and knowledgeable professional experts and Politian's. But whether one finds that ethically tolerable or not, we hope we have made it clear that even such a tactic only begs the question, for there are no value-free, true-false answers to any of the wicked problems*" Horst W. J. Rittel and Melvin M. Webber[15]

[15] "Dilemmas in a General Theory of Planning," *Policy Sciences* 4, no. 2 (June 1, 1973):

Horst Rittel and Melvin Webber in their 1973 work *Dilemmas in a General Theory of Planning* define problems of urban design and architecture as *wicked problems*, characterized as having no singular conception, no perfect solution, no ethical opportunity for trial and error, no clear endpoint, and resulting consequences that ripple far beyond the original scope of the problem. Not to mention that these already significant challenges are compounded by the cultural complexity present in a post-industrial, diverse, heterogeneous society.

> "*We have neither a theory that can locate societal goodness, nor one that might dispel wickedness, nor one that might resolve the problems of equity that rising pluralism is provoking*." - Horst W. J. Rittel and Melvin M. Webber[16]

This loss of trust in the idea of the wise *Promethean Architect* [a] in the mid 20th century set the stage for participatory design discussions. It was a sobering realization that professional Architects might, in fact, not have all the answers, accompanied with a growing sense of existential dread in the face of the true complexity of urban and architectural design problems.

[a] Referring to the modernist ideal of the architect, characterized by Ratti and Claudel in Open Source Architecture

If designs that benefited the general public could not come from the mind of the expert alone, then new methods of design would need to be tested.

in 1969, Giancarlo De Carlo proposed that the only effective way to stop the architect from being an "*operative appendage*" of the dominant societal powers was to engage in a sequential design process that revolved around continuous feedback and revision from the inhabitants of an architectural work. That it was only through this cyclic engagement that the architect could overcome their alienation from the public and restore their credibility in the eyes of the people.[17]

While the notions of cyclic feedback and revision are compelling to discuss in theory, they are significantly more difficult to implement in practice when it comes to architecture. Historically, buildings are, notably, static, and an architect's active engagement in their life typically ends shortly after construction. Might then, an architect utilizes new technologies to empower and engage the broader public, or at least the buildings prospective occupants in cycles of user feedback and revision, to create buildings that better suit the public that occupies them?

This certainly seems to be the future proposed by the Open Source

155–69, https://doi.org/10.1007/BF01405730.

[16] Ibid.

[17] Giancarlo De Carlo, "Architecture's Public," in *Architecture and Participation*, Digit. print (London: Taylor & Francis, 2009), 3–22

Architecture Movement, initiated by Carlo Ratti and Mathew Claudel in their manifesto "Open Source Architecture", they define the OSArc movement as;

> "*an emerging paradigm [...] drawing from references as diverse as open source culture, avant-garde architectural theory, science fiction, language theory, and others, it describes an inclusive approach to spatial design, the* ***collaborative use of design software****, and the transparent operation throughout the course of a buildings [...] life cycle*".[18]

The movement began life as a collaborative Wikipedia page in 2011. This collaborative online document would serve as the kernel for an article published in Domus[19] in June of 2011. However, after the article's publication, the text would continue to grow, eventually becoming the full-fledged manifesto that was published in 2015. Ratti and Claudel (and a host of many other collaborative authors) trace a lineage of participatory methods that span from the historic vernacular, through the gothic cathedrals, briefly interrupted by international modernism before being revitalized in the mid to late 19th century by the post-modernists and cyberneticists. They propose that the next evolution of participatory design is one that will build on the success of the Open Source software movement by creating an 'Open Source Architecture'. Architectural designs that can be shared modified and redistributed but that will ultimately be composed by what they refer to as the "Choral Architect" who weaves together the disparate ideas of the populous into a harmonious whole.[20] However, the models and methods that have proved successful in the world of Open Source software development may not be so easy to translate to architectural Design.

Theodora Vardouli and Leah Buechley provide a critical assessment of the assumptions and problems that exist within the conception of an Open Source Architecture. At the core of their critique is the fundamental differences in medium and methods of creation between architecture and software. The Open Source software movement can flourish because its software is reliably reproducible by anyone with a computer and an internet connection through the compiling of source code. In basic terms, a software's source code is the plain text written in a (more or less) human-readable programming language that can be compiled to produce the computer-readable set of instructions that are the software. Because a

[18] Carlo Ratti and Matthew Claudel, *Open Source Architecture* (New York, New York: Thames & Hudson, 2015)

[19] Carlo Ratti, "Open Source Architecture (OSArc)," Domus, June 15, 2011, https://www.domusweb.it/en/opinion/2011/06/15/open-source-architecture-osarc-.html.

[20] Ratti and Claudel, *Open Source Architecture*

software's source code is simple plain text, the code can be shared modified and reproduced without error by thousands of participants with minimal financial investment and without the need for any commercial tools. The process of "compiling" or building architecture from its plans and constituent virtual representations is significantly more nuanced, involves many more actors, often with competing interests, and requires significantly more capital investment. The idea that architecture has a "source" to be open is a problematic one as well, and according to Vardouli and Buechley, this lack of a clear 'Open Source' can lead to hollow and meaningless appropriations of Open Source ideals.[21]

Most OSArc projects treat the virtual representations of a building (3D models, BIM Data, CAD drawings) as the 'source' of a work of architecture and utilize common Open Source platforms like GitHub to share and manage collaborative editing of this data. However, unlike software source code, which is comprised mainly of plain text, architectural data currently requires the use of specialized commercial software to read and edit. OSArc projects have excelled at the sharing of architectural data over the web; however, the industry-standard tools available for editing this data are commercial, closed source, and proprietary, which imposes a financial barrier to entry on any who wish to engage with these projects. Without an open platform to open and edit what the OSArc movement views as the 'source code' of an architectural work, its difficult to argue that these works are Open Source at all. To put it simply, the realization of a truly "Open Source Architecture" requires more than just the sharing of plans or building models in proprietary formats. For a project to be Open Source in the truest sense of the term, then the set of tools necessary to interact with that project should be Open Source as well to ensure that its information is accessible, legible and editable for those who wish to engage with it. Otherwise, the OSArc movement runs the risk of becoming yet another "hollow appropriation"[22] of Open Source methodologies.

But how far can one get in producing an architectural project using only Open Source software?

[21] Theodora Vardouli and Leah Buechley, "Open Source Architecture: An Exploration of Source Code and Access in Architectural Design," *Leonardo* 47, no. 1 (August 3, 2012): 51–55, https://doi.org/10.1162/LEON_a_00470

[22] ibid.

Early Attempts at Architectural Design with Open Source Software.

In the summers of 2016 & 2017, I worked with the Lodge at Pine Cove, a "luxury rustic" vacation resort located on the French River in northern Ontario, to design two 1,000 sq. ft. modular cottages. Investing in a commercial software package, such as Autodesk's Revit, as an individual designer working on a relatively small scale project seemed impractical, and a significant financial burden. As an alternative, I attempted to use the Open Source 3D modelling package Blender as my primary tool for the design of the Pine Cove Cottages.

Figure 4: *The Toni Harting Cottage at the Lodge at Pine Cove. Design by Kevan Cress, Interiors by The Lodge at Pine Cove, Photo by Braeden Martel*

The early design work done with Pine Cove consisted primarily of conceptual design exploration. 3D models of several of the existing cottages on-site at Pine Cove were created and used as a rough guide for the development of the pre-fabricated modular system that would be used for the construction of the new cottages. Renderings and visualizations were produced and shared with guests of the Lodge for feedback. Blender was capable of performing all of these tasks, and its modelling tools allowed for the quick creation of the detailed models necessary to explore possible modular construction options. However, once the design was complete and a construction system decided on, Blender's lack of available dimensioning tools made it impossible to create the necessary Engineering, Manufacturing and Construction drawings without transferring the model to a commercial

software, in this case, McNeil's Rhino. While Rhino was used to create the 2D Technical Drawings, the Blender model was maintained to produce 3D renders and visualization. Attempting to manage two representations of the same project in independent software packages and trying to keep them in sync resulted in inconsistencies between the 3D Blender model and the 2D Rhino drawings. These inconsistencies, the result of human error in maintaining synchronization between the two forms of representation, along with a tight timeline for final specifications, resulted in some small discrepancies between the fabricated panels and the 3D model.

While small discrepancies like this are not uncommon in architectural projects where 3D models and 2D drawings are not synchronized, this experience attempting to utilize Blender in a production environment helped to identify Blender's current strengths and limitations when applied to architectural projects. It has shown that even though Blender's modeling and rendering toolsets are valuable assets for design and client consultation, it is lacking the functionality required to produce working drawings, and currently this makes it impractical for use in the later stages of architectural production. This experience has also demonstrated that attempting to work with multiple representations of a project in separate software packages introduces more opportunities for errors and inconsistencies to accumulate in a project.

But what if we could augment Blender to a level where it was capable of facilitating the entirety of an architectural workflow? Could it provide a cost-effective option for small firms and individual designers? Could it allow Open Source Architecture projects to truly meet their mandate, allowing them to produce their drawings and models in a format that is available to be read and modified by anyone? Could it serve as an Open Source platform upon which the AEC industry might start to build a culture of distributed development, providing opportunities for the passionate community of architectural technology innovators to fold their advancements back into a common platform for all the industry to use?

Blender has a long way to go before we can begin to see answers to these questions. As we will see in the following work, even updating Blender to the level where it can be used for small scale architectural drawings will be a significant task, but it is these questions, these ideas of a future where the AEC industry has the agency to direct the development of the software it relies on, that truly motivate the modest developments made in this thesis.

SECTION 2:

Requirements and Specifications

Requirements Analysis; Why do Architects Dimension

A Brief History of Architectural Descriptions

Architecture deals fundamentally with the description of objects to be built, but this act of description has not always relied on drawings with numeric dimensions as we are familiar with them today. In ancient times the architect would physically mark the site itself. Drawing a plan in-situ. Classical architects could describe their forms as a series of geometric relations. Mario Carpo once described how the western cannons 'first true architect' Filippo Brunelleschi occasionally made use of carved radishes to convey construction details to craftsmen during the construction of the Florence cathedral. However we could consider the popularization of paper and ink drawing in the 14th century[23] to be the first of many technological developments that would lead to our modern mode of dimensioning. With paper and ink drawings the conception of a work of architecture became fundamentally separate from the physical work itself, even though the creation of these early drawings was essentially analogous to the physical marking of the site in times past. Early paper and ink drawings were constructed geometrically, utilizing a compass and unmarked ruler[24] in processes that would be mechanically similar to the marking out of a building in situ, only with smaller tools.[25] These geometric methods for design, persistent since the era of Vitruvius, provided several affordances for architects of the time. They allowed for the construction of curvilinear forms without the need for complex calculus and decimal notation (which wouldn't make its way to Europe until the early 17th century),[26] They could be transmitted as a set of verbal instructions or written in plain text (which was essential as there wouldn't be a way to reliably copy drawings until the 16th century),[27] and they were inherently scale-less (because the description was

[23]Marco Frascari, "An Age of Paper," in *When Is the Digital in Architecture?*, ed. Andrew Goodhouse and Canadian Centre for Architecture (Montréal: Canadian Center for Architecture, 2017), 25–31.

[24]Mario Carpo, "Drawing with Numbers: Geometry and Numeracy in Early Modern Architectural Design," *Journal of the Society of Architectural Historians* 62, no. 4 (2003): 450, https://doi.org/10.2307/3592497.

[25]Carpo, "Building with Geometry, Drawing with Numbers," 40.

[26]George Sarton, "The First Explanation of Decimal Fractions and Measures (1585). Together with a History of the Decimal Idea and a Facsimile (No. XVII) of Stevin's Disme," *Isis* 23, no. 1 (1935): 153–244, http://www.jstor.org/stable/225223.

[27]Mario Carpo, "How Do You Imitate a Building That You Have Never Seen? Printed Images, Ancient Models, and Handmade Drawings in Renaissance Architectural Theory,"

a set of geometric relationships that would produce proportionally identical results regardless of the scale at which they were carried out at).[28] These methods of geometric representation were not unique to architecture of course and were utilized in mathematics and physics as well, as they were the best techniques of the time. Galileo, for instance, discerned that the path of a projectile could be described geometrically as a section through a cone (a parabola), but it would be centuries still before this same phenomena could be defined algebraically through the quadratic equation.[29]

The era of geometric construction however, would only last for so long. The ability to mass copy drawings through woodcuts or copperplate etchings on a printing press (widespread in the 1600's),[30] paired with the advances in numeracy brought about by Simon Stevin's introduction of decimal numbers to the west (1585)[31] and the application of al-Khwarizmi's algebra to describe continuous lines by Fermat and Descartes (1637)[32] ushered in the 'Age of Architectural Numeracy'[33] in which the conventions of the number based, scaled, and dimensioned drawings that we are familiar with today were developed. While these number based drawings are visually similar to their geometric counterparts, they lack a geometric procedure, which means that in order to translate from the small scale drawing, to the full scale building you need to know the drawing's scale factor, and you need to be able to measure each element in order to conduct the appropriate scaling. This leads to what Mario Carpo describes as the Iron Law of Transference, in which;

> " *'We can only measure what we can draw, and we can only build what we can measure in a drawing. In short, if you cannot draw it, you cannot measure it, and if you cannot measure it, you cannot build it.'*.

If one were to take the Carpo's Iron Law of Transference at face value, you might imagine that we would not need dimensioned drawings at all. That builders and craftspeople would simply measure directly from the architects scaled drawing themselves, with written numeric dimensions

Zeitschrift Für Kunstgeschichte 64, no. 2 (2001): 227, https://doi.org/10.2307/3657211.

[28]Carpo, "Building with Geometry, Drawing with Numbers," 40.

[29]Victor J. Katz and Bill Barton, "Stages in the History of Algebra with Implications for Teaching," *Educational Studies in Mathematics* 66, no. 2 (September 18, 2007): 195, https://doi.org/10.1007/s10649-006-9023-7.

[30]Carpo, "Drawing with Numbers," 454.

[31]Sarton, "The First Explanation of Decimal Fractions and Measures (1585). Together with a History of the Decimal Idea and a Facsimile (No. XVII) of Stevin's Disme."

[32]Katz and Barton, "Stages in the History of Algebra with Implications for Teaching," 195.

[33]Carpo, "Building with Geometry, Drawing with Numbers," 42.

provided merely as a courtesy or to save time. However in practice it is generally the written dimensions that are the primary description of the buildings form, and, quite importantly, these written numeric dimensions are the basis of the contracts and legal agreements between the Architect, and Client or Contractor. In fact the Canadian Construction Documents Committee, a joint committee with representation from the Association of Consulting Engineering Companies-Canada (ACEC), Canadian Construction Association (CCA), Construction Specifications Canada (CSC), and the Royal Architectural Institute of Canada (RAIC), notes clearly in its standard contracts that:

> "If there is a conflict within the Contract Documents: dimensions shown on Drawings shall govern over dimensions scaled from Drawings" (CCDC 2 – 2008 Stipulated Price Contract GC 1.1.7.3)

This language in contract is necessary because, in practice, there are many opportunities for errors to accumulate through the transmission, printing, and physical measuring of the drawn elements in architectural document packages, and to make matters worse these accumulated errors are all multiplied by the drawings scale factor. It is for this reason that the written dimension takes legal precedent over the scaled measurement in todays architectural profession, as numeric dimensions are immune to the potential distortions that reproduction inflicts on drawn representations. We should then amend Carpo's Iron Law of Transference to include this fundamental step of producing an architectural drawing; describing our drawn representation with written numeric dimensions.

> *We can only dimension what we can measure, we can only measure what we can draw, and we can only build what we have dimensioned in a drawing. In short, if you cannot draw it, you cannot measure it, if you cannot measure it you can not dimension it, and if you cannot dimension it, you cannot build it.*

So, our Open Source architectural software must be capable not only of producing drawn representations of works of architecture (3 dimensionally or otherwise), but it also needs to be capable of carrying out this fundamental step of measuring our design, and displaying those measurements as numeric dimensions. It must be able to Measure It.

Dimensioning Conventions

How should we display and draw these numeric dimensions then? What are the conventions for numeric dimensions and where did they come

from. Here the history is less clear cut. We might deduce that some of the conventions would have arisen along side the early applications of orthographic drawings for precise measurement in military architecture in the 16th century,[34] however any clear documentation of this I have not found. Many of the drafting symbols and conventions found today seem to have arisen from the engineering study of Geometric Dimensioning and Tolerances, a field of research whos origin is generally attributed to Stanley Parker and his 1956 text Dimensions and Drawings, however a reputable source tracing this history has proven illusive.

While the history of dimensioning conventions may be somewhat murky, the standards that define and dictate these conventions in current day practice are crystal clear. "ISO 129 Technical Drawings - Indication of Dimensions and tolerances" produced by the International Standards Organization lays out the standard description of how numeric dimensions should be drawn. In general the dimension elements drawn with MeasureIt-ARCH have been designed to comply with ISO 129-1, and the relevant sections of the standard have been noted in the Requirements Specification below, as well as the evaluation of that specification in Section 5 of this thesis.

[34]Pérez Gómez and Pelletier, *Architectural Representation and the Perspective Hinge*, 266.

Classifying Architectural Software

With an understanding of the importance of numeric dimensioning in architectural practice in mind, it is useful here also to take a moment to consider the types of software that are often used in a current day architectural practice, and the workflows in which these software packages are used for dimensioning and architectural representation, in order to help define the scope and features that should be included in our Open Source architectural software. A comprehensive analysis of current architectural workflows would be a massive undertaking, as architectural workflows often contain nuances and idiosyncrasies that vary from firm to firm, architect to architect and project to project. That being said, there are existing frameworks in place that define broad categories of software functionality that can be helpful when attempting to determine a more general class of use cases. For this analysis, Mark Bew and Mervyn Richard's BIM Levels framework serves as a basis on which to build. Bew and Richard's framework provides differentiation between software packages based on the level of connection and synchronization maintained in a project. The framework first considers the synchronization maintained between the many forms of representation (drawings, details, renderings, schedules) produced to describe the building in question. Second, it considers the degree of synchronization maintained between the different actors throughout the AEC industry involved in the production of a building. This framework by Bew and Richards has been used to inform BIM standards quite extensively in the UK and to a lesser degree internationally (as outlined in PAS 1192-2 and ISO 19650 respectively).[35] Each of Bew and Richards BIM levels can be defined briefly as such:

[35]Limited, "PAS 1192-2."

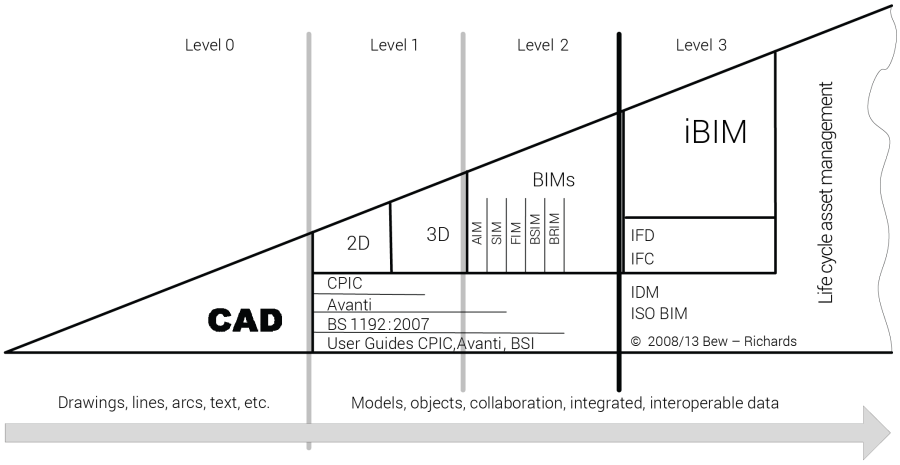


Figure 5: *Mark Bew and Mervyn Richards BIM Wedge*

Level 0: Unlinked 2D drawings; All building information is represented in two-dimensional drawings, and all synchronization needs to be maintained manually. Hand drawn drawing sets or early 2D CAD workflows would be considered BIM level 0. There are very few CAD applications available today that only offer Level 0 functionality.

Level 1: Hybrid 3D-2D approach with 'non-federated' [b] 3D models with limited Metadata. BIM level 1 workflow practices are characterized by the use of single-discipline 3D models[36] with some basic capacities for generating 2D drawings and other data exports (schedules, etc.). Rhino, Sketchup, and AutoCAD offer this level of functionality.

[b] a 'federated' model in the context of Bew and Richards framework refers to a building model whose various layers (mechanical electrical, structural, architectural) are distributed, to be worked on by their respective profession, but remain linked and synchronized to a commonly available master model

Level 2: Partially 'Federated' 3D Models with imbedded metadata are used to automatically generate 2D drawings and other data exports that remain linked to, and are informed by the 3D model. BIM level 2 models can be linked within a common data environment and used to resolve conflicting information across disciplines. BIM level 2 is what we commonly consider 'Building Information Modeling' in North America. software like Revit, Vectorworks, and ArchiCAD facilitate this level of workflow.

Level 3: In a level 3 software a single fully federated 3D model is accessible at all times to all stakeholders through an online common data environment

[36] Bilal Succar, "Building Information Modelling Maturity Matrix," *Concepts and Technologies*, 65–103, accessed August 18, 2019, https://www.academia.edu/186259/Building_Information_Modelling_Maturity_Matrix

using a non-proprietary file format such as the IFC (Industry Foundation Class) open standard. No software platform available today currently supports this level of functionality.

These three levels outlined by Bew & Richards deal predominantly with how architectural information in CAD and BIM software is exchanged among the various stakeholders involved in the creation of a building, and how this information is transformed into 2D drawings for the production of contract documents. While these levels encompass the dominant tools that are the main workhorses of architectural firms, they don't capture many of the more aesthetically focused tools that are often used in parallel with the firm's CAD or BIM software. To more fully encapsulate the software functionality utilized in architectural work, a fourth category of 'supplementary tools' should be defined.

Supplementary Tools: Supplementary tools augment the typically limited visualization capacities of CAD and BIM software, and are often used to produce realistic or stylized renderings of the aesthetic or ephemeral characteristics of an architectural work. Software such as Lumion, 3DsMax, Unreal Engine, V-ray, and Maxwell are often used in this capacity.

An Open Source Architectural Software Specification

Scope

With our software classifications in mind, we can make some decisions about the scope of our Open Source architectural software. To keep our scope manageable, we will work under a few additional assumptions.

Firstly, we assume that developing a new 3D modelling tool from scratch that is capable of meeting the needs of architects and the AEC industry is an unreasonable goal for a single novice programmer to realize. Instead, an existing Open Source modelling tool should be selected and used as a basis upon which to add features to facilitate an architectural workflow that fits within one of the level classifications outlined by Bew and Richards. Blender has been chosen as the basis upon which to build for this thesis.

Second, we need to identify which of Bew and Richards levels of functionality, our proposed solution is aiming to sit within. Based on my own previous experience using Blender as a design tool, as discussed in the motivations of this thesis, we can make some statements about Blender's current capabilities and where it might fit into Bew and Richards framework, with some minor modification. We can comfortably say that Blender's capacities for rendering and architectural visualization already give it a reasonable standing within our **Supplementary Tools** category. We can also say, quite confidently, that Blender's 3D modelling ability sets it above the limited 2D drawing workflows of a Level 0 classification. We can not, however, say that Blender fits into a level 1 classification, as it lacks drafting tools necessary to translate the architectural information contained in its 3D models into dimensioned drawings that can be shared with other stakeholders. Other Level 1 Tools such as Rhino, Sketchup, and AutoCAD include tools to create annotations and dimensions to communicate this architectural information. These tools also provide options to display architectural elements as simple line drawings, relying on line weight, style and colour to communicate form, instead of rendered shading.

To gain a clearer understanding of what functionality is present in Blender, and how it compares to the features on offer in other software prevalent in the AEC industry, the following Matrix and Radar charts were produced. The matrix takes a sampling of common software packages and scores the degree to which a given feature (tool or operation) has been implemented. The features selected for ranking are those that are present in more than one

of the software packages being evaluated. Scores are assigned numerically from 0-6 based on the degree to which the feature has been implemented.

A score of 0 is assigned if no support is present for the stated feature.

A score from 1-3 is assigned if the feature is available in the software through the use of an add-on.

A score from 4-6 is assigned if the feature is present natively within the software package itself.

Evaluation of the features degree of implementation was kept quite coarse. A simple qualifier of minimal, basic or comprehensive support was provided. These qualifiers correspond to scores of 1,2,3 or 4,5,6, respectively, depending on whether or not the implementation is internal or external. This coarseness in the ranking system was necessary, as a full evaluation of what constitutes a 'high-quality' implementation any of these features could quickly become a substantial research endeavour of its own, which would be contrary to the intent of this analysis. This particular assessment aims simply to provide a high-level overview of which features are commonly present in software used in the AEC industry.

To display these rankings in a visually meaningful way, the features being scored were sorted into five major categories, and an average score was calculated for each category. These categories are;

- Drafting
- Rendering
- General modelling
- Interoperability,
- BIM capabilities.

The average scores for each category is presented in a radar chart for each software. Blender's current state is represented in the darker orange, while its desired future state is proposed in light orange.

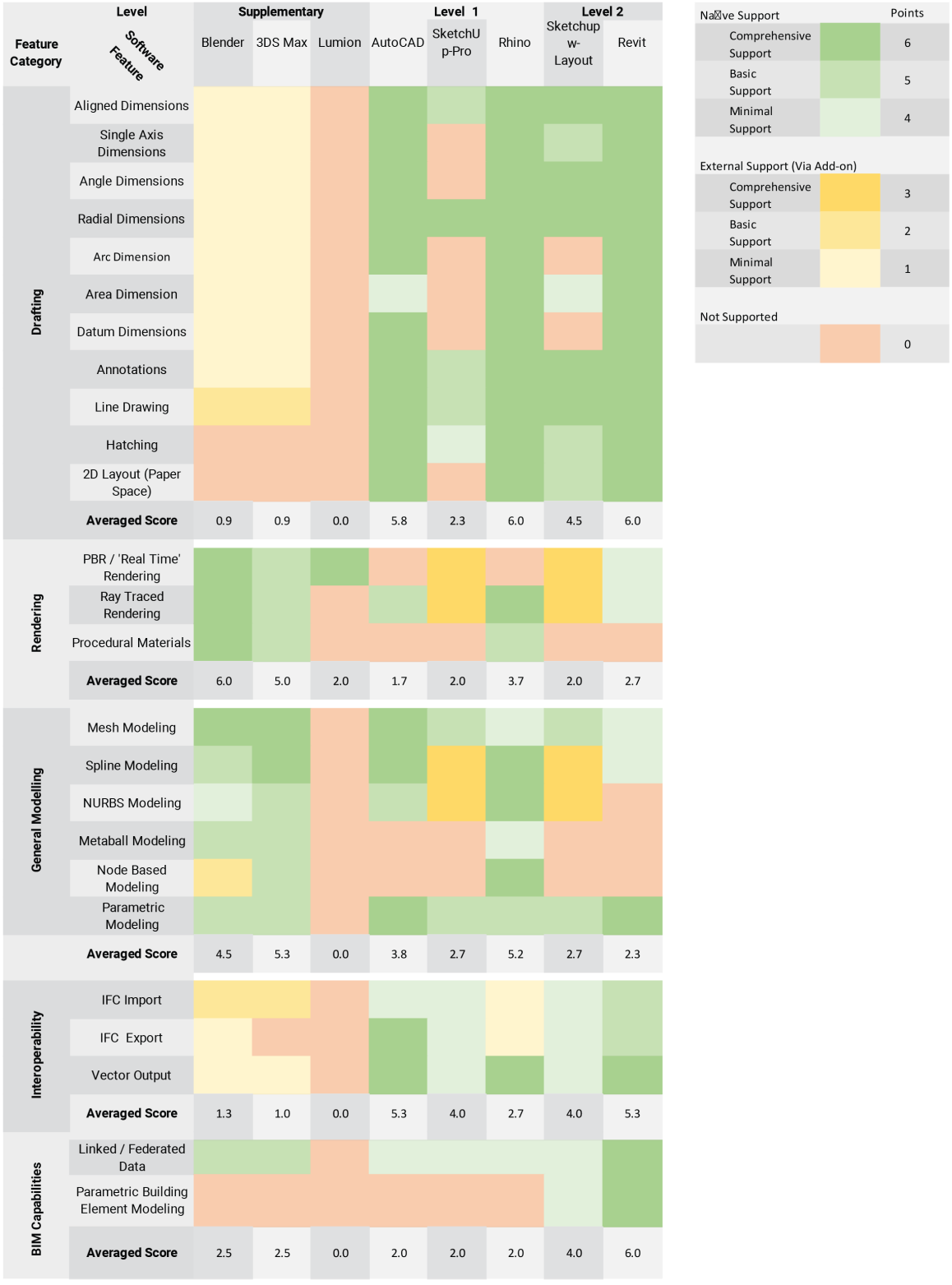


Figure 6: *Architectural Software Analysis Matrix.*

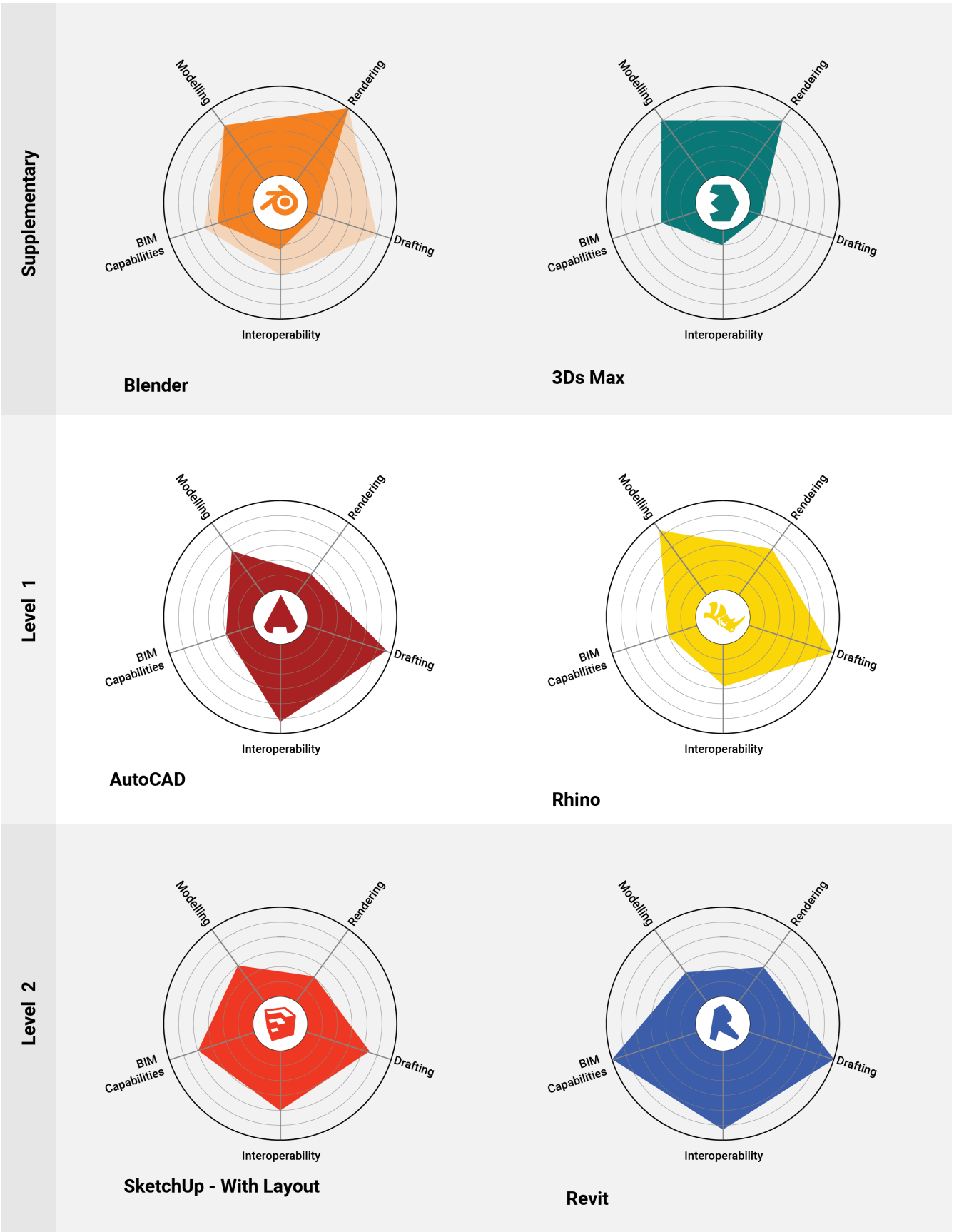


Figure 7: *Architectural Software Analysis Radar Charts.*

The results of the matrix and radar graph fit well with the previous practical experience of using Blender in a production environment during my work designing cottages for The Lodge at Pine Cove. In this practical application, Blender's modelling and rendering functionality was suitable for the schematic design phase of the project and for representing the aesthetic elements of the design, but when it came time to communicate the technical information required by building code officials, manufacturers and contractors, Blender's available toolset fell short.

Based on these experiences, Bew and Richard's classification system, and the provided analysis, we can see that the improvements required to augment Blender to the point where it could be considered a **Level 1** software lie predominantly in our categories of Interoperability and drafting, while some substantial improvements to the BIM capabilities category would be necessary for it to approach a **level 2** categorization.

It seems feasible then for the scope of this thesis to focus on augmenting Blender to a point where it could be considered a **Level 1** tool. To tighten the focus further still, we can see that Blender's substantial deficiency in the drafting category, when compared to the other level 1 software considered, makes this an ideal feature set to focus on for our planned augmentations.

The specification below outlines how these drafting features might be implemented in Blender, and assigns each feature a priority as it relates to achieving a level 1 categorization for Blender. The priorities are as follows.

Necessary; Necessary features are those such as like line drawing, dimensioning and annotating tools, which are present in all Level 1 packages.

High; High priority features are present in at least two level 1 packages.

Medium; Medium priority features are present in at least one of the Level 1 packages discussed and represent 'quality of life' or automation improvements that would allow for more efficient workflows. Medium priority features can generally be achieved manually using some combination of the Necessary or High priority features in combination with Blender's existing feature set (for example, a dedicated room area tag feature would speed up the production of plan drawings, but could be set up manually in Blender with the existing driver system and MeasureIt-ARCH's annotations).

Feature Requirements

Line Drawing

Priority: Necessary

Description:

- Users specify edges of their 3D Geometry to be drawn as simple lines
- Lines have their weight, colour, and style (solid or dashed) specified by the user
- Lines occluded by geometry should be visually distinct, differentiated by colour and dash style
- Lines that represent the silhouette of an object should be visually distinct, indicated by line weight
- Lines created in the same operation should be collected as a line-group, and share common properties (colour weight etc.)

Stimulus and Response:

- The user selects the desired edges of their 3D Geometry within the 3D view space
- Once all edges are selected, the user selects the UI button (or keyboard shortcut) for the line group operation to create lines along those edges and collect them in a line group
- Once a line group is created its properties should be adjusted through a corresponding UI panel
- Edges should be able to be added or removed from a line group after its initial creation.

Automated Line Group Creation

Priority: Necessary

Description: As Blender's mesh-based 3D geometry will often consist of significantly more edges than will be necessary to produce clean line drawings, and manual selection of the relevant edges will often be a tedious process for the user, an automated system should be in place to automatically initialize line groups with edges that meet some user-specified factor, such as;

- Geometry crease angles
- Material boundaries
- Non-manifold edges

Stimulus and Response:

- The user selects the object they wish to run the operation on
- The user selects the relevant selection criteria
- The operation generates the line group from edges that pass the selection criteria

Annotations

Priority: Necessary

Description:

- Annotations display user-entered text
- Annotations should be anchored to an objects Vertex, Origin, or an empty object
- Annotation Leader Lines should conform to ISO 129-1 Section 5.5
- Annotations should have user-defined:
  - Font
  - Font Size
  - Color
  - Alignment (Left, Right, Top, Bottom)
  - Line Weight (for the leader line

connecting the text to the anchor point)
    - End Cap (for the end of the leader attached to the anchor point)
- Annotations display critical data about the object they are anchored to such as;
    - Assigned Material
    - User defined Metadata

Stimulus and Response:

- The user selects the Vertex, Object or Empty Object they wish to anchor the annotation to
- The user selects the UI button for the Annotation operation to create a new Annotation
- The Annotations properties should be edited via their corresponding UI panel
- Annotations Placement and Rotation should be manipulated in the 3D viewport

Aligned Dimensions

Priority: Necessary

Description:

- Aligned Dimensions measure the distance between two points in 3D space. These two points could be;
    - Object Origins
    - Vertices
    - Empty Objects
    - Light Objects
    - Camera Objects
- Aligned Dimensions extensions should always be perpendicular to the line they measure (ISO-129-1 Section 5.4)
- Aligned Dimensions should have user-defined:
    - Font
    - Font Size
    - Color
    - Rotation (taking the line being measured as the axis of rotation)
    - Line Weight (for the leader lines)
    - Terminations (According to ISO 129-1 Section 5.3.2)

Stimulus and Response:

- The user selects two or more points (vertices, Objects, Empties etc.) they wish to dimension
- The user selects the UI Button (or Keyboard Shortcut) for the Aligned Dimension operation
- Aligned Dimensions are created for each pair of points selected (if 3 points are selected then a dimension is created between points 1 and 2, and a sperate dimension is created between points 2 and 3)
- Dimensions are automatically positioned based on the user's current viewpoint
- A Dimension's distance from its line of measure can be adjusted in the 3D viewport
- Dimension properties should be adjusted through a UI panel

Single Axis Dimensions

Priority: Necessary

Description:

- Single Axis Dimensions measure the distance between two points in 3D space, but only along a specified Axis. These two points can be;
    - Object Origins
    - Vertices
    - Empty Objects
    - Light Objects

    - Camera Objects
- Single Axis Dimensions are always placed perpendicular to the axis along which they measure
- Single Axis Dimensions can have user-defined:
    - Font
    - Font Size
    - Color
    - Line Weight (for the leader lines)
    - End Cap (arrows or dashes at the ends of the dimension line)
- The axis of measure for a Single Axis Dimension can be:
    - Any cardinal Axis (X, Y, Z)
    - Any user-defined Axis

Stimulus and Response:

- The user selects two or more points (vertices, Objects, Empties etc.) they wish to dimension
- The user selects the UI Button (or Keyboard Shortcut) for the Single Axis Dimension operation
- Single Axis Dimensions are created for each pair of points selected (if 3 points are selected then a dimension is created between points 1 and 2, and a sperate dimension is created between points 2 and 3)
- Dimensions are automatically positioned based on the user's current viewpoint
- A Dimension's distance from its line of measure can be adjusted in the 3D viewport
- Dimension properties can be adjusted through a UI panel
- The Axis of Measure can be selected in the UI panel

Angle Dimensions

Priority: High

Description:

- Angel Dimensions measure the angle between two lines, as defined by three points.
- Angle Dimensions are aligned in-plane with the 3 points that define them.
- Angle Dimensions can have user-defined:
    - Font
    - Line Weight
    - Color
    - Font Size
    - Radius

Stimulus and Response:

- The User selects the 3 points that define the angle they wish to measure. Points 1 & 2 represent the first line of the angle, points 2 & 3, the second line.
- The User selects the UI button (or keyboard shortcut) for the Angle Dimension Operation, which creates a new Angle Dimension
- The Angle Dimension's properties can be adjusted through a UI panel

Arc Dimensions

Priority: High

Description:

- Arc Dimensions should measure the Radius (ISO 129-1 Section 7.3) and Arc Length (ISO 129-1 Section 7.6) of an Arc, defined by 3 Points, where points 1 and 3 are the extremes of the arc.
- Arc Dimensions are aligned in-plane with the 3 points that define them.
- Arc Dimensions can have user-defined:

- Font
- Line Weight
- Color
- Font Size
- Radius
- Endcaps

Stimulus and Response:

- The User selects the 3 points that define the arc they wish to measure.
- The User selects the UI button (or keyboard shortcut) for the Arc Dimension Operation, which creates a new Arc Dimension
- The Arc Dimension's properties can be adjusted through a UI panel

Room Area Tags

Priority: Medium

Description:

- Room Area Tags measure the area defined by a planar polygon defined by an arbitrary number of points
- Room Area Tags are placed in-plane with the polygon that they measure
- Room Area Tags can display a room name in addition to the area of the room.
- Room Area Tags can have user-defined:
  - Room Name
  - Font
  - Font Color
  - Font Size
  - Fill
  - Fill Color

Stimulus and Response:

- The user selects either:
  - the points that define the polygon they wish to measure
  - or the existing faces of the geometry that they want to measure
- The user selects the UI Button (or keyboard shortcut) for the Room Area Tag operation, which creates a new room tag
- The Room Tags Properties can be adjusted through a UI panel

Schedules & Reporting

Priority: Medium

Description: Architectural Schedules are tables containing user-specified attributes about a group of objects or materials present in a model. Schedules are valuable tools for reporting information contained in a digital model. They can be used to produce lists of necessary material quantities, Door and Window Specifications and amounts, and Furniture and Equipment lists, to name a few of their many uses.

- Schedules should be generated to include information based on the set of user-defined attributes such as;
  - Assigned material
  - Name
  - Instance
  - Any User-defined metadata fields
- Schedules should allow for the grouping of data by attribute to allow for reporting on quantities and amounts. Each grouping represents a column in the table:
  - e.g. All four items that are instances of the 'Door 36" x 72"' object are grouped to a single row and report their number in a 'Quantity' attribute
- Schedules should have the following user-defined properties:
  - Main Font
  - Column Title Font
  - Row Title Font

    - Color
    - Line weight
    - Position
    - Size
    - Attributes
    - Groups
    - Camera Visibility
- Schedules should be exportable in a format that can be read and modified by other tools created to edit spreadsheets:
    - .csv (comma delimited)
    - .ods (the open document spreadsheet format)

Stimulus and Response:

- The user creates a Schedule item via a UI button
- The user specifies their desired attributes and groupings in that Schedules UI Panel
- The user runs the populate schedule operation to analyze the model and produce the completed schedule.
- The Schedule is then drawn in the view of the specified camera

Title Blocks

Priority: Medium

Description: Title Blocks Identify a drawing or page and keep track of revision numbers and other key project information.

- Title Blocks should contain the following user-defined information
    - Author name
    - Revision Number
    - Revision History
    - Date
    - Description
    - Disclaimer/ Legal info
    - Project Name
    - Project Address
    - Project Number
    - Scale
    - Drawing Title
    - Drawing Number
- Title Blocks Should Automate the following Information:
    - Date
    - Scale
- Title Blocks should have the following user defined visual characteristics:
    - Orientation (Top, Bottom, Side)
    - Line Weight
    - Color
    - Font
    - Font Size (definable per information category)

Stimulus and Response:

- The user selects the camera to which they wish to add a title block
- The user selects the UI button to add a title block to that camera
- The user adjusts the title blocks settings and fields in its UI panel
- The title block is drawn in the view of the specified camera

General Requirements

Style System

Priority: Necessary

Description:

- Styles should available for all elements created through this add-on:
  - Annotations
  - Dimensions
  - Line Groups
  - Title Blocks
  - Schedules
- Styles should define the relevant visual properties for that element type. Typically:
  - Color
  - Font
  - Line Weight
- If a style should be used, and which style to be used should be user-defined on a per-element basis.
- Styled properties should be able to be overridden on a per-property basis, without breaking other properties connected to the style.
- The Styles UI should be visually similar to their corresponding elements UI panel

Integrated User Interface

Priority: Necessary

Description: MeasureIt-ARCH's user interfaced should be designed to minimize clutter and to fit with the design principles of Blender's UI. MeasureIt-ARCH elements should be organized into lists, and the properties of the currently selected item should be displayed below in a single column layout. This UI design is based on Blender's representation of other Object Properties such as Vertex Groups and Shape Keys.

3D Integration

Priority: High

Description: All elements created by MeasureIt-ARCH should be placed in 3D space, independent of any 2D work planes or paper space.

Gizmo Implementation

Priority: Medium

Description: Although the properties of MeasureIt-ARCH drawing elements can be edited through the UI, and Blender's non-overlapping non-modal UI design still allows for users to see responsively how changes to these properties impact the drawing, this method of editing dimensions is still not as intuitive as directly manipulating drawing elements in the 3D view.

For many of Blender's standard tools, 'gizmos' are used as a visual cue that allows for direct manipulation of objects in the 3D viewport. Connecting gizmos to the properties of MeasureIt-ARCH elements would make them significantly more intuitive to adjust.

Adaptive Behaviour

Priority: High

Description: All elements created by MeasureIt-ARCH should attempt to adjust certain properties, such as position or orientation, to remain coherent, visible and legible as the user adjusts their viewpoint in the 3D scene. Some behaviours to facilitate this might include:

- Line Groups should draw with distinct Line weight, Color, or Dash style depend-

ing on their occlusion or silhouette state.
- Text should automatically adjust its orientation to remain legible to the user. (See ISO-129-1 Section 5.6.2 Figure 23)
- Dimensions and Annotations should adjust their placement in the 3D space to remain visible to the user.

Animated Properties

Priority: High

Description: Building MeasureIt-ARCH within Blender should expose all of it its properties to Blender's animation system. Any MeasureIt-ARCH property should able to be animated. Animated properties could be used for simple tasks, like toggling the visibility of drawing elements, or for creating more dynamic forms of representation, such as walk throughs and animations.

Hybrid Rendering

Priority: High

Description: All elements created by MeasureIt-ARCH should be drawn as an overlay, drawn overtop of the rendering system currently in use in Blender's viewport. This overlay should be conceived as a separate layer of visual information, even though it utilizes the same 3D space and information for Occlusion purposes. Treating the MeasureIt-ARCH elements as an independent layer of visual information allows it to assist in the production of more than just technical line drawings. As an overlay, MeasureIt-ARCH can facilitate hybrid forms of representation that make use of Blender's diverse options for stylistic and realistic rendering, with this additional layer of technical information composited in.

Instancing Support

Priority: Medium

Description: Taking advantage of Blender's Linked data capabilities, MeasureIt-ARCH annotations, line groups and dimensions should be able to be instanced along with their host object and linked across Blender files. A proper instancing system could begin to facilitate workflows closer to that of a BIM Level 2 capable software, allowing users to create linked libraries of objects that would function similarly to Revit's families.

Interoperability

Priority: Medium

Description: In order for this tool to be properly integrated within an architectural workflow, it should be possible to export the information produced within it to other software packages, and to import models and information from other sources. Key import and export formats should include:

- Import:
  - .dxf for AutoCAD data
  - .ifc for the import of BIM data
  - .obj , .fbx , .3ds , and .dae for the import of general purpose 3D models
- Export:
  - .svg for the export of vector linework, text and dimensions
  - .ifc for the export of BIM data
  - .obj , .fbx , .3ds , and .dae for the export of general purpose 3D models

Scaled Camera Space

Priority: Medium

Description: Typically, in CAD style applications, the inclusion of a 'Paper Space' is provided to set up the desired views, accommodate the flattening or cutting of the 3D model into 2D representations, and facilitate the addition of 2D annotation text and dimensions. Our proposed Open Source architectural software aims to keep all of this information positioned in 3D space and use automated placement systems to ensure that it remains legible as the user's viewpoint shifts. However, there must still be a feature that allows for this 3D view to be composed onto a page with meaningful real-world size and scale for this software to be useful in conventional architectural workflows. Currently, Blender uses 'Cameras' to define the specific view to be rendered however, the camera settings are optimized for the composition of digital images and video and lack clear physical definitions. Some Adjustments to Blender's native camera settings could be beneficial here.

- Cameras should have an option for render sizes specified in physical units (Centimeters, Inches, etc.) with a specified resolution (Pixels Per Inch)
- Stored presets should be available for common paper sizes
- Each Camera should be able to have an independent size and resolution
- Orthographic Scale information should be defined as a ratio with physical units:
    - e.g. 1cm on the defined paper size is 1m in the model space
- Stored presets should be available for common architectural scales.

SECTION 3:

Blender and Architecture

Why Blender?

Before we dive into MeasureIt-ARCH's development itself, it is important to understand why Blender was chosen as a base for this tool. At its face, the choice may not seem immediately intuitive; There are several Open Source CAD tools available (BRL-CAD, FreeCAD, and LibreCAD, to name a few). However, each of these packages faces unique challenges when it comes to meeting the goal chosen for this thesis work. That is, creating an Open Source software that would allow for the creation of architectural drawings, and could be made available to anyone.

For this goal to be met, the platform should;

1. Have a stable and supported user and developer base.
2. Be approachable.
3. Provide usable tools for creating dimensioned and annotated design drawings.

As an additional decision factor, an Open Source design platform must be more than a 'free AutoCAD clone', or a purpose-built design tool for single specific design style or solution. An Open Source design platform has the potential to question the existing conventions of CAD and BIM software and should leverage that potential.

While most existing Open Source CAD tools successfully meet our third criteria, they fair significantly worse when evaluated on the first two. This is somewhat problematic, as these first two criteria are arguably much more difficult targets to achieve, especially when considered in the scope of a masters thesis. Though some Open Source tools like FreeCAD certainly have aspirations of improving their approachability and feature set to surpass conventional CAD tools, the lack of concrete funding and full-time developer base makes these aspirations especially challenging for an Open Source platform.

Blender, on the other hand, meets all but our third criteria quite well.

Stability & Support

While many Open Source software tools operate as a sort of pseudo-anarchic collaboration without much of a defined structure to speak of, Blender's development structure lies somewhere between the chaos of true Open Source, and the structure of more traditional software development. Much like Netscape, Blender started development as a closed source project.

Created by Ton Roosendaal in 1995. Blender started life as a proprietary tool for NeoGeo a Dutch animation studio co-founded by Roosendaal. From 1998 to 2002 Blender continued to be developed by Roosendaal under the company Not a Number. In 2002, following Not a Number's shutdown, the community surrounding Blender, initiated by Roosendaal, raised 100,000 Euros to purchase Blender's licensing from Not a Number's investors and re-release it as an Open Source software under the GNU GPL license.[37] Since 2002 Blender has seen stable development and a robust community, largely due to the support of the two organizations that facilitate the software's development, the Blender Institute, and the Blender Foundation.

The Blender Institute is responsible for supporting Blender by offering professional training courses at their offices in Amsterdam, testing and demonstrating Blender's capabilities through Open Projects (typically short films produced entirely with Open Source software), and managing the Blender Cloud service. The Blender Cloud is a paid monthly subscription service that provides access to more advanced training tutorials, as well as texture and material libraries, and asset and project management solutions geared towards the management of Animated films. The Institute is primarily funded through these Blender Cloud subscriptions.

Figure 8: *A Still from the Spring Open Movie. Produced by the Blender Institute*

[37] Blender Foundation, "Blender 2.8 Design Document," Blender Developers Blog, accessed August 9, 2019, https://code.blender.org/2017/10/blender-2-8-design-document/.

The Blender Foundation, on the other hand, is a not-for-profit organization whose primary purpose is to provide the financial and organizational infrastructure necessary to manage a successful software project. Most importantly, they provide grants and funding that allow the Blender communities most dedicated and talented developers to work on Blender either full or part-time. The Foundation gives these select developers that have the desire and skill to improve Blender the financial freedom to do so and ensures that there is always a committed core team available for maintenance, bug fixing, planning, and to offer some limited user support.

The Foundation's grants are funded by both community and corporate donations to the Blender Development Fund. As of July 26th, 2019, the development fund provides an income to the Blender Foundation of 82,089 US Dollars per month[38]. The Foundations 29 corporate members make up 67% of its funding, while the 2716 individual donors account for 33%. This money funds grants to 17 developers (3 part-time), as well as supporting Ton Roosendaal and three other support staff. [c]

[c] it is almost certain that these figures will have changed by the time you are reading this; however, The Blender Foundation regularly updates its current status and grants publicly at https://fund.blender.org/

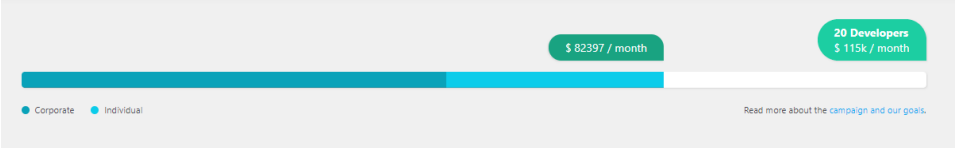


Figure 9: *Status of the Blender Development Fund as of July 30th, 2019*
Image from https://fund.blender.org/

Industry Support It's worth noting that at the beginning of 2019, the development fund sat at roughly 25,000 US Dollars per month[39]. Its substantial growth in the intervening six months is largely due to a 1.2-Million-dollar grant (paid over four years) from Epic Games (creators of Unreal Engine), following the release of the Blender 2.8 Project.[40] This is a major milestone for Blender both financially and in terms of Industry recognition. The Epic grant and smaller grants from other notable companies like Ubisoft, Tangent Animation, Google, Intel, and others illustrate that the Blender Foundation and its developers are recognized, respected and trusted by major players in the Computer Graphics industry.

[38] Blender Foundation, "Blender 2.8 Highlights," Blender Developers Blog, accessed August 9, 2019, https://code.blender.org/2018/03/blender-2-8-highlights/

[39] Ton Roosendaal and Blender Foundation, "Development Fund Report, July 2019," Blender Developers Blog, July 20, 2019, https://code.blender.org/2019/07/development-fund-report-july-2019/

[40] ibid.

Approachability

Approachability has been a challenge for Blender in the past, but improving the User Experience, and taming its notoriously steep learning curve has been a significant focus for the development team. The Blender Foundation is just reached the completion of a major design overhaul of the software with its 2.8 release. An explicit target of the Blender 2.8 project was improving the User Experience of the software. Redesigning the user interface to enhance feature discoverability, and implementing an 'Industry Standard Keymap' to make Blender's keyboard shortcuts and interaction feel more comfortable to those coming from commercial applications like Autodesk's 3DsMax. The Blender Institute is also responsible for producing nearly 2 hours worth of free introductory tutorials to acquaint new users with the software. This approach is notably different from those of other Open Source CAD applications, which either lack the resources to produce this degree of introductory educational content or explicitly take an" experts only" approach to the design of their tools.[41] [d]

Like any 3D design software, Blender does have a steep learning curve, however, the abundance of tutorials produced by the Blender Institute and the Blender Community, as well as the development team's ongoing efforts to improve the User Experience for new users, are helping ease the difficulty of approaching the software.

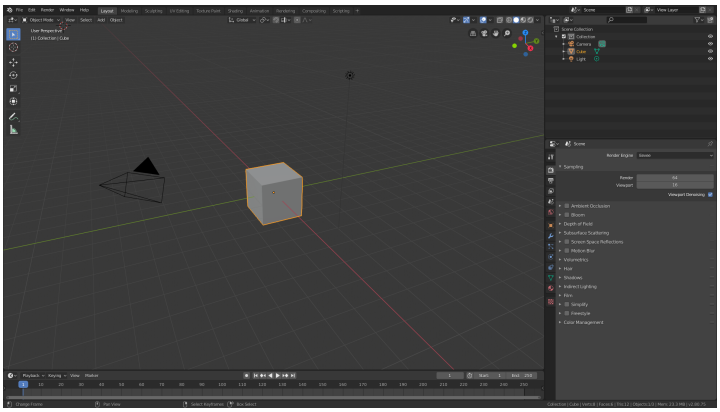


Figure 11: *Blender 2.8's User Interface*

[d] From BRL-CAD's documentation; "MGED does not provide a discoverable graphical user interface. Going through the available tutorials and documentation is required to be proficient. MGED is expert-friendly with minimal documentation and feedback inside the application itself".

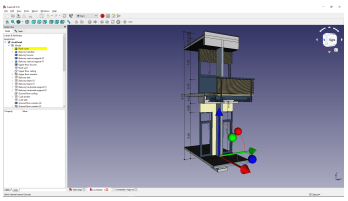


Figure 10: FreeCAD User Interface

[41] BRL CAD, "Mged - BRL-CAD," accessed July 26, 2019, https://brlcad.org/wiki/Mged

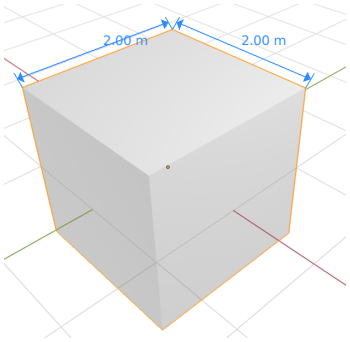


Figure 12: A Cube Dimensioned with the Original Measure It

Architectural Drawing tools

In the past, the Blender community has made efforts to add tools for basic dimensioning. Antonio Vazquez, one of Blender's more active volunteer developers, produced the MeasureIt add-on. Which added tools to create and display simple, screen space (meaning that the dimensions are drawn as a 2D overlay on the 3D scene, rather than existing in the 3D space of the model), dimensions in Blender. However, the add-on lacks support for dimension styles, and line drawings, both features necessary to produce architectural drawings efficiently. While MeasureIt was still being maintained by Antonio, with periodic updates to ensure its functionality in the latest version of Blender, his primary focus for the last two and a half years has been on the development of Blender's Grease Pencil tools.

Thanks to the Open Source nature of Blender and its add-ons, Antonio's work on MeasureIt could be used as a solid foundation on which fully featured dimensioning tools could be built. MeasureIt would serve as an open base on which to create a tool that would meet our scope and specifications defined in the previous section.

Based on the past efforts of Antonio Vazquez, the technical challenge of creating useful dimensioning tools in an already thriving Open Source Platform seemed an accomplishable scope of work. Significantly more manageable in scope than attempting to resolve the issues of approachability, financial stability, and community support present in other existing Open Source CAD applications. For these reasons, improving MeasureIt for Blender to meet the requirements of our specification was identified as the primary goal of this thesis.

Blender's Notable Features

Building architectural dimensioning tools into Blender provides opportunities to draw on the rich toolset already present in the software. As Blender is primarily used is as a Digital Content Creation tool for VFX, Animation and Games, the majority of its tools are focused on providing artist friendly solutions to create high-quality visuals. Meaning that when applied as an architectural tool, Blender brings new opportunities for design thinking to the table rather than merely being a 'free autoCAD clone'. Some of the most exciting features that relate to the production of architectural design are presented here;

Responsive Physically Based Rendering

Figure 13: *'Minimalist Kitchen' by Augusto Cezar rendered in Blender with EEVEE*

Blender 2.8 features the EEVEE (Short for 'Extra Easy Virtual Environment Engine[42]) rendering engine, primarily developed by Clement Foucault. EEVEE can simulate near-photorealistic lighting and material characteristics (including reflection and refraction), in 'real-time'. The exact framerate varies depending on the complexity of the scene and your specific hardware, of course. Still, the EEVEE engine is typically responsive enough to be used during design and modelling tasks. As a renderer, this functionality is similar to packages like Lumion, or Unreal Engine, but because EEVEE is built directly

[42]Blender Foundation, "Eevee Roadmap," Blender Developers Blog, accessed August 9, 2019, https://code.blender.org/2017/03/eevee-roadmap/

into Blender, and not as an external rendering application, it can provide users with a fully rendered scene while they design.

Procedural Modeling

Procedural Modeling, more commonly referred to as parametric modelling in the field of architecture, generates forms based on a series of rules and input parameters. In Blender, this can be accomplished through a variety of methods. Blender's modifier system takes a simple input mesh, and a set of parameters, and performs a specified operation to generate new geometry or deform existing geometry. These modifiers can be stacked to create complex forms from simple inputs. Blender's driver system allows the user to link nearly any property of any object in the scene to any other. This allows for the creation of complex parametric relationships, although the user interface for defining and managing these relationships is not particularly intuitive. Jacques Lucke, a recent Blender Foundation grant recipient, is currently in the process of overhauling many of Blender's internal systems, such as the modifier system, to allow for a node-based interface. This should allow for node-based parametric modelling workflows similar to Rhino's Grasshopper. Similar, but less intuitive, workflows are already possible in Blender using either the animation-nodes add-on (also created by Lucke) or the Sverchok add-on.

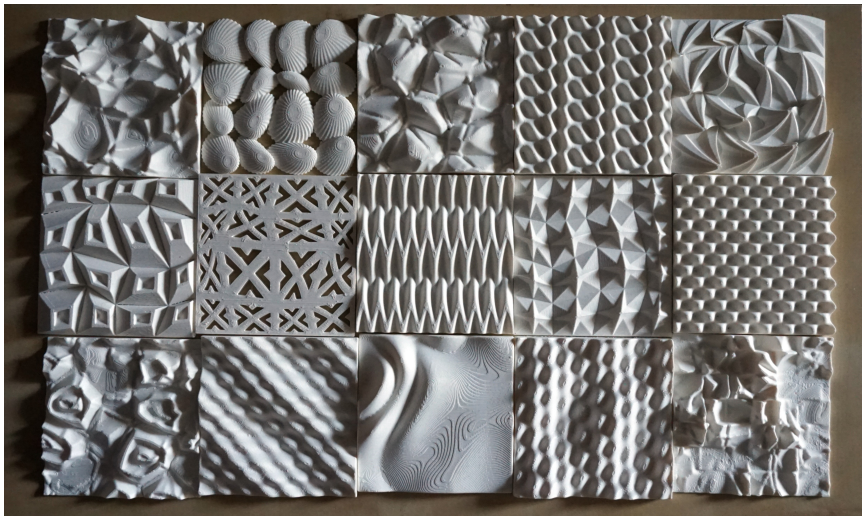


Figure 14: *'Modulations' by Alberto Giachino.*
Parametric tiles generated with Blender and the Sverchok add-on then 3D printed

Custom Object Metadata & Linked Data Workflows

While it is beyond the scope of this thesis to add BIM-like functionality to Blender, it is important to note that Blender's data structures are already well situated to handle adaptations to manage the kinds of object metadata required to facilitate a BIM workflow. Users can already define custom properties that are attached to an object and use these custom properties in conjunction with Blender's driver system to alter an object's geometry, materials or animation. Blender also provides a 'linked data workflow'. Meaning that objects can be linked across Blender files, allowing a user to create collections of objects with customizable parameters that can then be linked into a central working file, in a workflow similar to Revit's Families, or SketchUp's Components. The user interface available to set up and manage this linked data workflow is not particularly intuitive at the moment; however, the basic functionality does exist, and an Asset Manager designed to facilitate working with linked collections is a key target for an upcoming release.[43]

Grease Pencil Sketching

Blender's Grease Pencil started life as a simple tool to sketch notes to animators in the 3D viewport, but thanks to recent work by Antonio Vazquez, Charlie Jolly, Daniel Lara, and Matias Mendiola, the Grease Pencil has been developed into a robust 2D drawing tool that exists within the 3D space of a Blender scene. This was intended to allow for the production of 2D animated films.

When paired with an appropriate input device like a graphics tablet, the Grease Pencil gives users the ability to sketch naturally into the 3D scene. For architects, the Grease Pencil could be used to develop architectural ideas quickly by sketching in-situ over simple 3D massing models. The Grease Pencil toolset makes Blender feel like a unique hybrid of sketchbook and modeling software.

[43] Blender Foundation, "Asset Manager," Blender Developers Blog, accessed April 13, 2020, https://code.blender.org/2020/03/asset-manager/

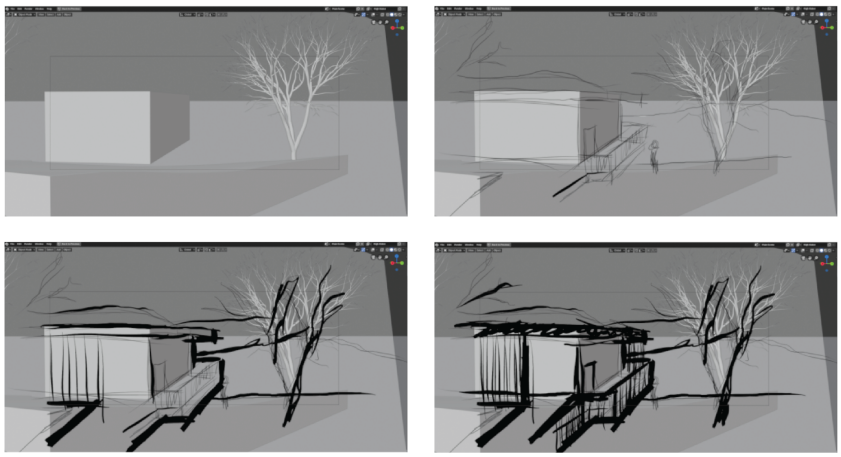


Figure 15: *Using Grease Pencil to sketch in-situ over a simple massing model*

Robust API

Blender features an incredibly robust Application Programming Interface (API) that gives access to low-level utilities within the software through the Python programming language. Application Programming Interface's are a relatively common feature of large software applications. They provide users with the ability to access certain aspects of the software through a programming language. Revit, Rhino, and SketchUp all feature API's; however, their depth and usefulness vary. Typically, the scope of an API allows users to write add-ons for the automation of repetitive tasks, and the importing and exporting of various filetypes[44],[45] however, some software API's provide deeper access that allows for the creation of complex tools. Rhino's Grasshopper plug-in is a prime example of this, although Grasshopper is now a core part of the Rhino software, it started life as an Open Source add-on built with Rhino's API.[46] Blender's API provides low-level access to some of the software's core functionality, such as the ability to access the OpenGL drawing functions that allow add-on's to create and

[44] Autodesk, "Understanding Revit Terms | Revit Products | Autodesk Knowledge Network," accessed April 1, 2019, https://knowledge.autodesk.com/support/revit-products/getting-started/caas/CloudHelp/cloudhelp/2014/ENU/Revit/files/GUID-2480CA33-C0B9-46FD-9BDD-FDE75B513727-htm.html

[45] Trimble, "SketchUp Ruby API — SketchUp Ruby API Documentation," accessed April 4, 2019, http://ruby.sketchup.com/

[46] McNeel & Associates, *RhinoCommon Is the .NET SDK for Rhino5 / Grasshopper: Mcneel/Rhinocommon* (2010; repr., Robert McNeel & Associates, 2019), https://github.com/mcneel/rhinocommon

draw new content to the 3D scene, the ability to alter existing User Interface (UI) elements, as well as functions that give add-on's access to Blender's Dependency Graph and mesh creation and modification systems. This functionality is essential for MeasureIt-ARCH, and other add-ons, as it allows for the creation of new tools and elements that integrate directly with the 3D environment.,

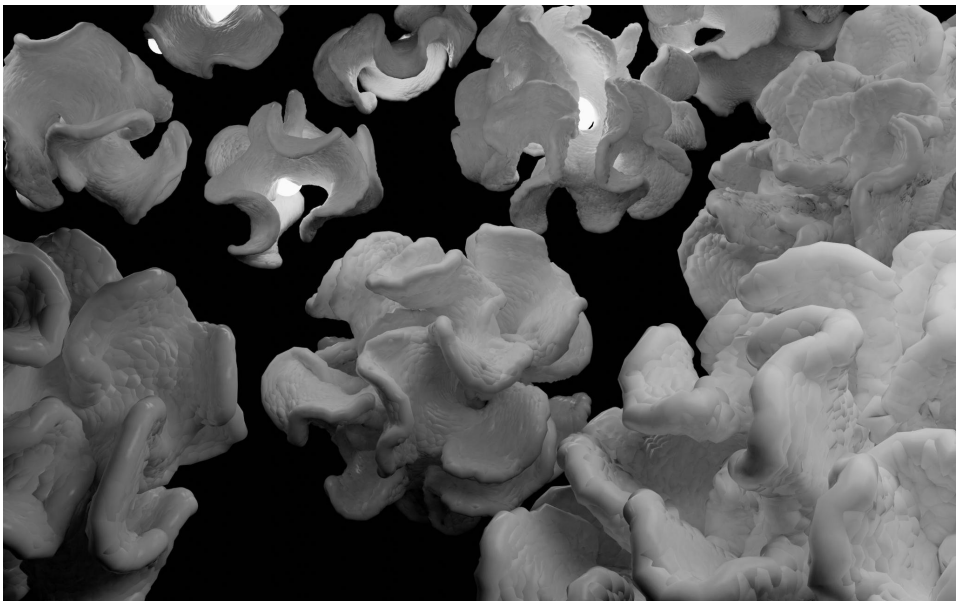


Figure 16: *'Differential Growth Collection 00 (aka: cabbages)' by Alex Martinelli. Generated in Blender via the Python API and Grease Pencil.*

These features provide new opportunities for designers. While BIM software development has put a focus on attempting to optimize the design process for maximum efficiency in technical precision and design documentation, a design tool like Blender puts its focus on functionality that allows designers to explore the more experiential aspects of a design quickly. It offers designers faster visual feedback on their designs, lighting, materiality, and atmosphere, through real-time rendering, which provides designers with more opportunities to iterate and interact with those features of their design work.

If this feature set were to be paired with tools for the creation of technical drawings, Blender could serve as a powerful design tool for architects.

Previous Architectural Investigations

> *"Although most users will be faced with a steep learning curve when using Blender, scaling that curve will bring many benefits in terms of design thinking and concept development in the architectural workflow of the user. In the end, in our experience in mastering a software like Blender the user becomes more comfortable with design tool making,"*[47]

In 2009 Alexandros Siglas and Theodoros Dounas published an analysis of Blender's capabilities in architectural workflows, examined through its use in a two-year design seminar at the Department of Architecture in Volos, Greece.[48] While much of their analysis is somewhat out of date, as Blender has undergone 25 releases and 2 major redesigns (the 2.5 and 2.8 projects) since 2009, their discussion of Blender's modular data approach, and how it impacts design thinking, remain consistent to today.

Modular Data Approach

> *"Modeling and other workflows in Blender don't try to hide the way the computer handles the data in 3D computer graphics. The designer is exposed to the mechanics of computer graphics, (…). The modular approach of the modelling tools(…), helped the students to develop "design thinking" and use Blender as a decision-making tool for concept design and not only as a visualization tool."* - Alexandros Siglas and Theodoros Dounas[49]

Dounas and Siglas note that Blender's modular approach to data storage and geometry manipulation helped students build an understanding of what they refer to as the 'mechanics of computer graphics.'[50] This modularity can be seen in the Blender's modifier system, it's linked data approach, and the driver system, which, as we have discussed, allow for operations to be stacked in a flexible manner to produce complex results from simple input geometry. Managing the balance between this exposure to the underlying systems that make up a 3D modelling application and providing intuitive easy to use tools has been a significant task for Blender developers over the intervening decade. While the modular approach has been maintained, the way it is presented is currently much less raw than it was in 2009.

[47] Theodoros Dounas and Alexandros Sigalas, "Blender, an Open Source Design Tool: Advances and Integration in the Architectural Production Pipeline," 2009, 8

[48] ibid.

[49] ibid.

[50] Ibid.

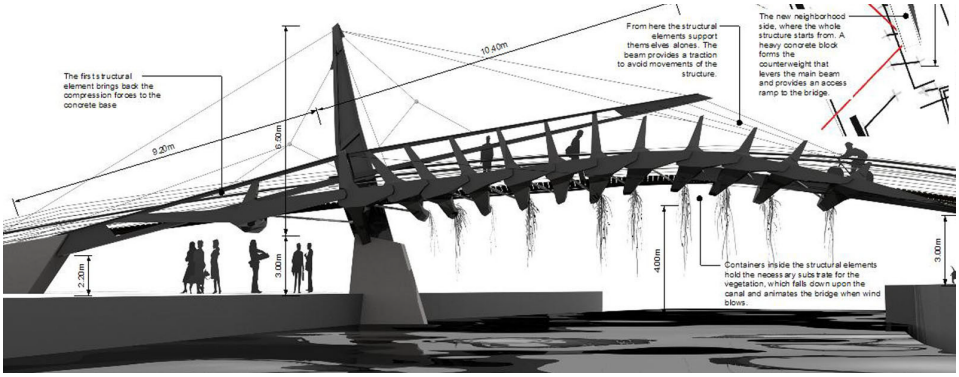


Figure 17: *Yorik Van Havre - Pennington, Road Bridge Case Study at Leeds-Liverpool A parametric bridge derived from a Bezier curve, as seen in, 'Blender, an Open Source Design Tool', by Alexandros Siglas and Theodoros Dounas*

Barriers to Adoption

In their conclusion, Dounas and Siglas propose that:

> "*Blender has still a long way to go in terms of percentage of adoption among architects, not because of lack of features or integration capabilities but because it is an Open Source software, downloaded free of charge and usually perceived as a mere toy.*"

As we've discussed, this toy-like perception is beginning to change in the computer graphics and animation industry, as more studios begin to utilise Blender in professional work and larger industry players such as Google, Intel, and Epic begin to contribute financially to Blender's development fund. Whether or not architects response to the software will improve as well remains to be seen, but the most achievable avenue to encourage adoption may lie in bridging the gap in Blender's architectural functionality that has persisted over the decade since Dounas and Sigalas's paper, Blender's lack of adequate dimensioning tools.

SECTION 4:

MeasureIt-ARCH

This section provides an overview of MeasureIt-ARCH's workings and development. First we explore how Antonio Vazquez's original add-on, MeasureIt was expanded and updated to be able to facilitate the development of new features for MeasureIt-ARCH. Second we take a deeper dive into the new linework and dimensioning tool's developed for MeasureIt-ARCH's, and discuss the precedent research that their development is based off of. Finally we discuss how MeasureIt-ARCH was shared, and how issue reports from the Blender community were essential to bug fixing, testing and improving MeasureIt-ARCH.

The codebase of MeasureIt-ARCH currently sits at more than 8000 lines of code, split over 12 files, written in Python and GLSL, made up of 308 Commits since December of 2018. The discussion in this section however, is not an exhaustive walkthrough of the content of MeasureIt-ARCH's code, but rather an overview of the process, methods, and precedent used in MeasureIt-ARCH's design. Through this overview, the reader should gain a high-level understanding of MeasureIt-ARCH's structure and operation.

For those interested in exploring MeasureIt-ARCH's code, Appendix A of this thesis provides a simple roadmap of MeasureIt-ARCH's code base, with an overview of the purpose of each python file, and MeasureIt-ARCH's source code itself, along with its revision history and reported issues are documented in their entirety in the projects GitHub repository.[51]

[51] Vazquez and Cress, *MeasureIt-ARCH*.

Updating MeasureIt

MeasureIt-ARCH's development started first with an assessment of MeasureIt to identify how its systems worked, which systems worked well, and which systems needed to be revised. The aim of the discussion here is to provide an overview of the thought process that inspired the four primary changes to Antonio Vazquez's original tool as well as to take a brief look at how each of these changes were implemented.

An Open Base to Build From

The MeasureIt add-on produced by Antonio Vazquez provided a fantastic foundation on which to start building MeasureIt-ARCH. It provided a substantial structure to work with, as well as some clear areas to improve.

MeasureIt added tools that allowed users to add aligned and single Axis dimensions to your Blender model, as well as angle, arc, area, and origin dimensions, annotations, and labels. Each of these measurement types was stored in a single list attached to the object they referenced. Each measurement had its own unique set of properties, all of which were set individually through a UI that displayed all an objects 'measurements' in a sequential list. Measurements were drawn in screen space, which means that while the distances themselves were calculated in 3D Space, the positions and placement of the measurements were converted to 2D pixel co-ordinates before being drawn. [e]

Through early tests with MeasureIt, a few fundamental changes became immediately apparent. To improve MeasureIt to meet our specification we needed to:

1. Add the ability to add new element types (like linework) to the add-on.
2. Redesign the User interface to avoid clutter, and clearly communicate features and properties
3. Develop a system to allow dimensions of a similar type to share visual properties (i.e. a dimension style system)
4. Redesign MeasureIt's draw system to work in 3D-space to allow for proper depth ordering and occlusion. [f]

Each of these tasks would require a substantial redesign of MeasureIt's code.

[e] all images drawn on a screen are translated to screen space co-ordinates at some point, but the point at which this occurs in the drawing pipeline impacts the spatial relationships of the object being drawn to its surrounding scene

[f] MeasureIt-ARCH's shaders use the 4D Homogeneous co-ordinates system, which is common practice for glsl shaders, however the python side that interfaces with Blender's API uses Blender's scene space 3D co-ordinates

How do MeasureIt & MeasureIt-ARCH Work?

Before we can understand how these changes were made, we need to understand, at least at a high level, the basic structure of how MeasureIt (and by extension MeasureIt-ARCH) work. Most importantly, we need to have a grasp of the steps that are taken in MeasureIt's code to store dimensions, annotations or line groups, set their properties, and finally draw them to the screen. For this discussion we'll use the term element to refer to any dimension annotation or line group created by the add-ons, since all three types follow the same general procedure. At this high level, both the original MeasureIt and the MeasureIt-ARCH follow, more or less, the same process. However, MeasureIt-ARCH adds some intermediary steps which we will discuss in relation to the features they improve.

Property Groups

MeasureIt and MeasureIt-ARCH store all the elements associated with a particular object inside a list attached to that object called a 'Generator'. Each element stored in the generator is an instance of a **Property Group**. Property groups contain all the properties needed to draw that element accurately, and determine its look and style. These properties include information about the element's line weight, colour, location, anchor point, or any other critical information that the software needs to draw that particular element. The properties on each element instance are initially set when the user creates that element.

Operators

In Blender, any action that the user can trigger through a button or a keyboard shortcut is called an **Operator**. When the user presses a button to add a MeasureIt element, it triggers an operator to create that element. This operator creates a new element instance, adds it to the relevant object's Generator list, and sets the properties in its property group according to the state of the scene at that moment (i.e., checking which vertices the user has selected as the element's anchor points etc.).

Once elements have been created, they need to be displayed in two ways.

Panels

First, we need to expose the properties in each element's property group to the user so they can be edited. We can do this through Blender's user interface. In Blender, segments of the user interface that display properties are called **Panels**. To present an element's properties to the user interface

we need to define a panel that describes which properties from the property group will be displayed, how they should be formatted, and where they should appear in Blender's UI.

Draw Methods

Second, we need to draw the element to the screen in the 3D scene. The drawing process involves checking the generator list of each object in the scene, working out the placement of all of the elements stored in the generator, and updating those elements whenever anything in the scene changes. This process is carried out in the element's **Draw Method**. An element's draw method is run every time Blender detects that *anything* in the scene has changed. This means that that a draw method is ideally being run upwards of 60 times a second since every change of the users viewpoint, or adjustment of a property triggers the scene to be re-drawn. To keep the 3D scene responsive we need to ensure that all of our elements draw methods are quick and efficient to run, since the speed of our draw methods will determine how many times a second the scene can be updated, directly impacting our 3D views frame rate.

Once the draw method calculates where an element should be situated in the 3D scene, and processes the properties that impact it's appearance, it sends this information to the appropriate **shader**.

Shaders

While MeasureIt and MeasureIt-ARCH run on the computers Central Processing Unit (CPU) **Shaders** are smaller programs that run on the dedicated hardware of the computer's Graphics Processing Unit (GPU). Shaders are written in the GLSL programming language and are responsible for taking the basic properties of an element provided by the draw method, and calculating how they should be drawn to the actual pixels of the screen.

These five aspects of MeasureIt and MeasureIt-ARCH, **Property Groups, Operators, Panels, Draw Methods** and **Shaders** are key to understanding the changes that were made to Antonio Vazquez's original MeasureIt and how these changes address the four issues identified above.

1. Adding New Element Types

In the original MeasureIt, every drawn element, (be it an aligned dimension or annotation or any other element type) was an instance of the same property group and used the same draw method. A property called 'gltype' was used to identify what type of element the instance was and what parts of the draw method should be applied. While this solution was functional for the existing element types in MeasureIt, this structure resulted in code that was somewhat challenging to understand and add to. Adding support for linework into this system would have meant adding more properties to the already extensive single property group and extending the existing plethora of branching cases in the draw method.

To help make MeasureIt-ARCH structure more readable, the way data was handled in the add-on needed to be re-thought. A system for managing data was implemented that would not only facilitate adding linework support, but also make it easier to add new types of drawing elements in the future. Rather than using one property group to store all possible properties, three categories of MeasureIt-ARCH elements, each with their own unique Generator list, Property Group, and Draw Method, were defined.

- **Dimensions** deal with the display of spatial properties like length, angles etc.
- **Annotations** deal with the display of user-defined text or the metadata of an object.
- **Line Groups** deal with the drawing of a collection of lines with similar characteristics (line weight, colour, etc.).

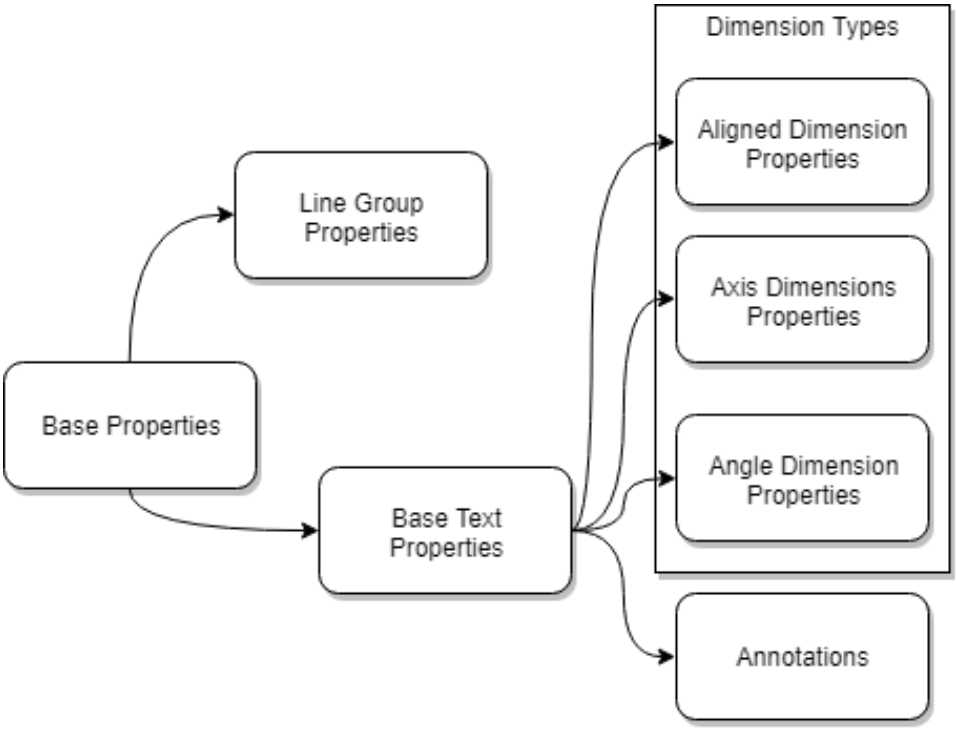


Figure 18: *A Schematic Diagram of MeasureIt-ARCH's Inheritance Structure*

While each of the three types is unique, they do share many similar properties. All these types have a colour and a line weight property, for example. To streamline the code, each type's Property Group was set up to inherit its common properties from a Base Property Group. This inheritance structure helps reduce duplicate code, and it helps ensure that standard operators (like deleting an item or updating its text) don't need to be rewritten for each type. This helps simplify the process of defining and managing each of our types and makes adding new subtypes in the future more straightforward.

The full Inheritance structure for MeasureIt-ARCH in its current state looks like this:

Data Definitions

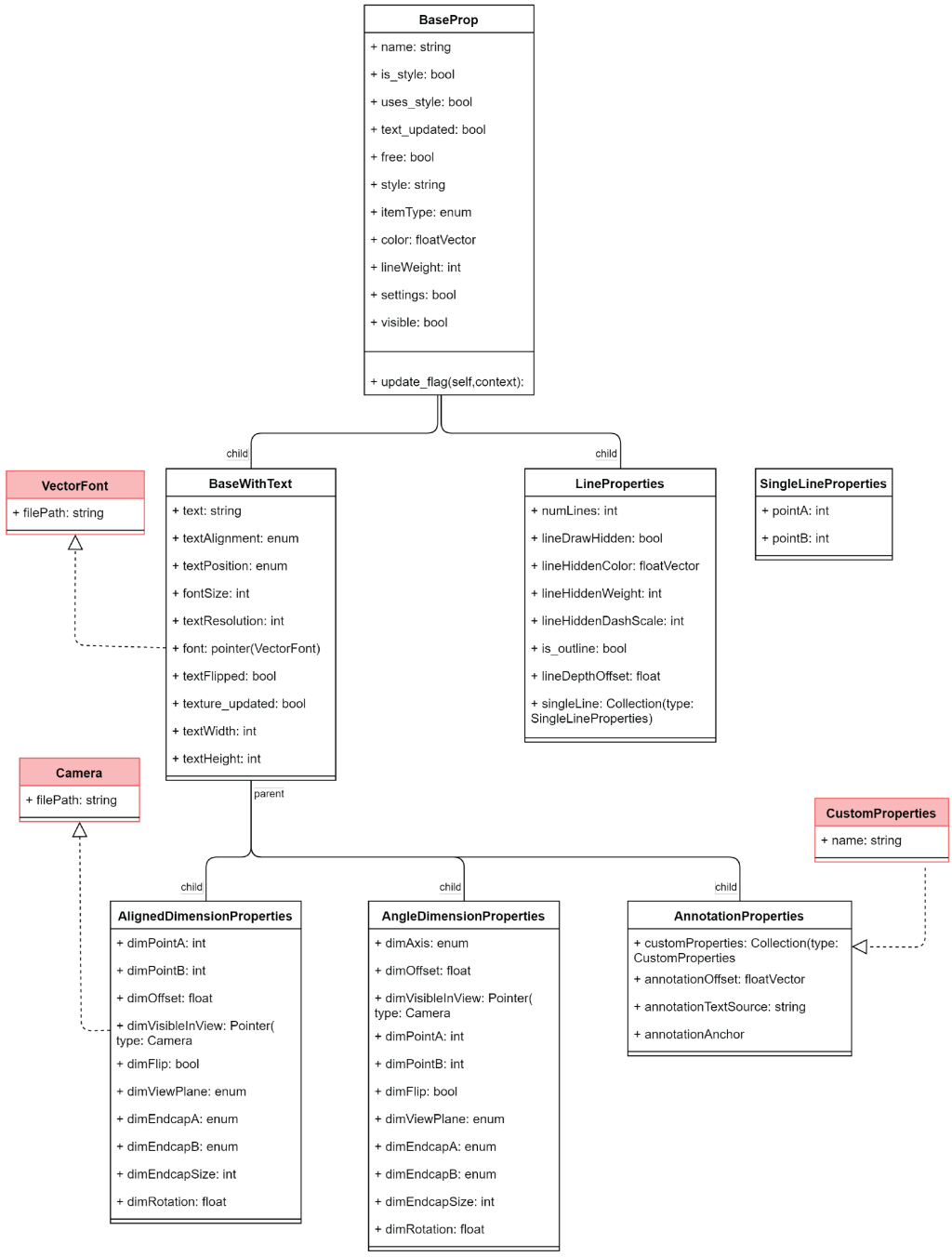


Figure 19: *Diagram of MeasureIt-ARCH's Inheritance Structure.*

2. Redesigning the User Interface

MeasureIt's original user interface, though functional, left a bit to be desired. The entirety of the user interface was put into the right toolbar of the 3D view, leaving little visual distinction between Operator buttons, configuration settings and element properties. The element properties themselves were clustered with a logic that was not immediately understandable to unfamiliar users and not consistent with Blender's general UI conventions. To make MeasureIt-ARCH feel more like an integrated part of Blender, it's UI needed to be redesigned to fit with the UI principles outlined by Blender core developers.

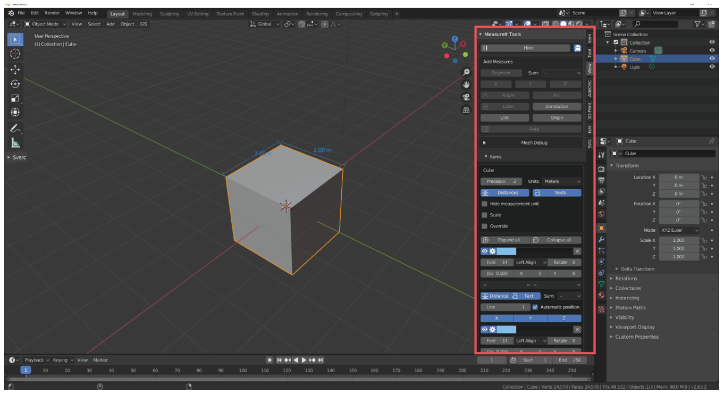


Figure 20: *MeasureIt's Original Location in Blender's UI*

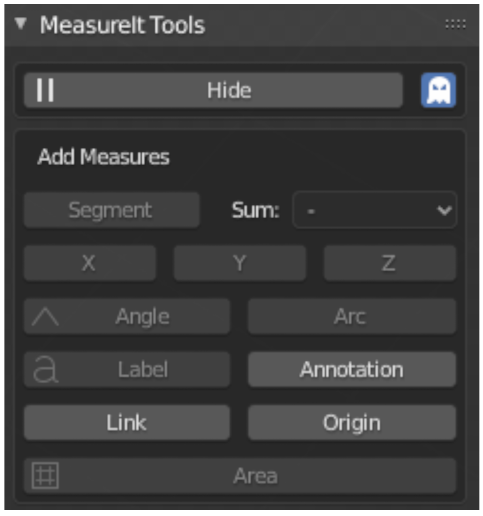


Figure 21: *MeasureIt's Original Panel for Operators.*

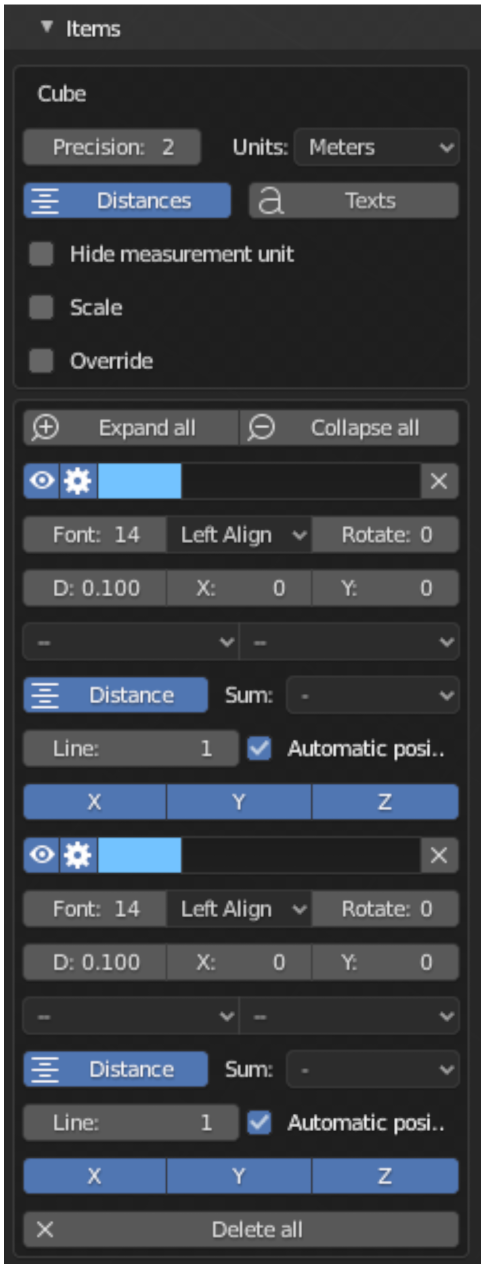


Figure 22: *MeasureIt's Original Panel for Dimension Settings*
Settings for two aligned dimensions are shown here

Blender's User Interface Design Principles

Blender User Interface, as started by Ton Roosendaal and developed by William Reynish and the rest of the UI team, [g] draws heavily on Jef Raskin's, *The Humane Interface*[52]. In 2008 Reynish published a design document for the Blender 2.5 UI redesign project outlining the fundamental principles drawn from Raskin's work that are embodied in Blender's UI;

[g] Blender's full UI team at time of writing consists of; Pablo Vazquez, William Reynish, Campbell Barton, Brecht Van Lommel, Julian Eisel, and Harley Acheson

- **Non-Modal;** The user interface should never "*pull the user's locus of attention away from the content she wishes to create*."[53] The designed object always maintains a privileged place in Blender's UI, ensuring that the designer is aware of how their actions in the software are impacting it.
- **Non-Overlapping** This quality of non-modality is achieved by implementing a non-overlapping interface. Rather than relying on a plethora of pop up windows and dialogue boxes, the likes of which have become the norm in complex software applications, Blender's UI is subdividable, to allow for all of the necessary UI panels to be made available in the same window. Each subdivision can be stretched to the user's desired size and used to represent a particular set of information (Object Properties, Materials, Scene Hierarchy, UV data, etc.). Meaning that the software never puts you in a position where you're editing properties in a floating window and unable to see their effect until after your changes are applied.

During the Blender 2.8 UI redesign, another principle was added to this mix, the **Single Column Layout**. The single column layout was advised so that all properties would be clearly legible in a single list, rather than clustered together as we see in MeasureIt's original UI design.

Organizing the User Interface Panels

Keeping these principles in mind, the first step in redesigning the UI involved sorting the various panels into appropriate areas of Blender's user interface.

Operators: The operator buttons for the creation of MeasureIt-ARCH elements are located in the right toolbar of the 3D View window. All of these operators are called from the 3D View after the user has selected the objects or vertices they wish to dimension. In order to distinguish them from other tools, the operators have been moved to their own tab in this toolbar. The

[52] Jef Raskin, *The Humane Interface: New Directions for Designing Interactive Systems* (Reading, Mass: Addison Wesley, 2000)

[53] William Reynish, "The Evolution of Blenders User Interface," n.d., 25

operator buttons were re-ordered to a single column layout and their text was paired with a representative icon from Blender's existing icon set.

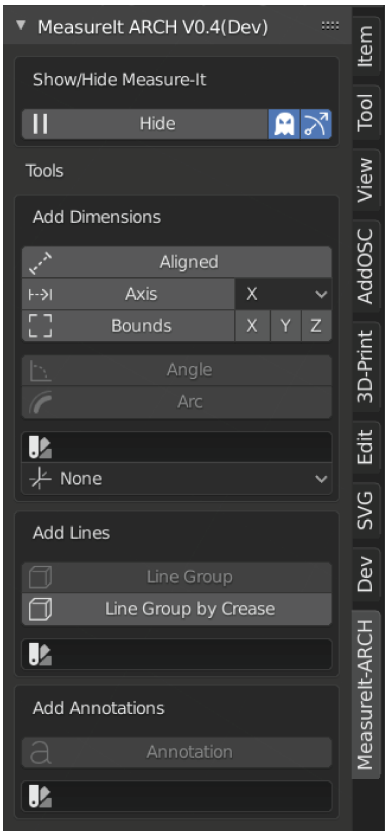


Figure 23: *MeasureIt-ARCH Main Tool Panel*

Element Properties: The properties for MeasureIt-ARCH's various element types were moved to the Object tab of Blender's Properties Editor. Each element type was given their own panel, to make them easier to quickly access , and visually distinguish.

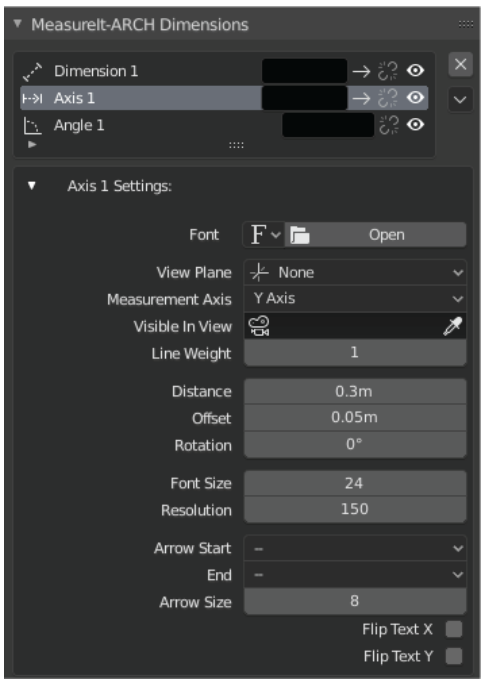


Figure 24: *MeasureIt-ARCH's Dimension Properties UI*

Configuration Settings: The majority of the original MeasureIt's configuration settings were eliminated in the update to MeasureIt-ARCH. The original MeasureIt used Unit settings independent of those defined in Blender's scene settings. To avoid confusion, these separate unit configuration options were removed, and MeasureIt-ARCH instead uses the units specified in Blender's scene settings. The remaining MeasureIt-ARCH unit configuration settings, which define the level of precision used in dimension text, were appended directly onto Blender's existing units UI panel in the Scene tab of Blender's properties editor.

Currently, a MeasureIt-ARCH Settings panel is also present in the Scene tab of Blender's properties editor. However, this panel only contains a series of toggle options that are useful for development debugging. These will be completely removed in an upcoming release.

Styles: The new Styles panel is also presented in the Scene tab of Blender's properties editor, since the style instances are stored as custom data attached to the scene.

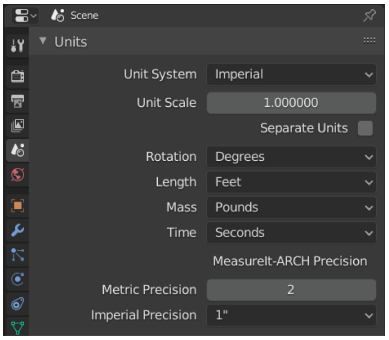


Figure 25: *MeasureIt-ARCH Unit Settings UI*

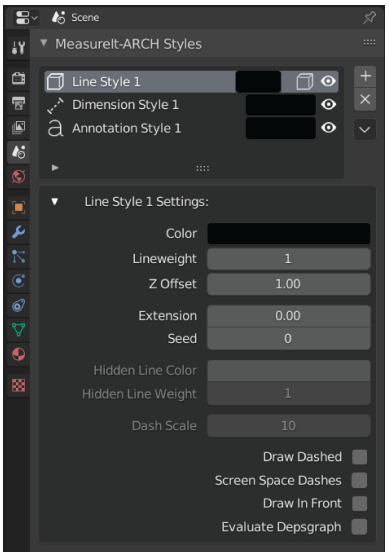


Figure 26: *MeasureIt-ARCH Style Settings UI*

Displaying MeasureIt-ARCH Properties; The List UI

The second step was finding a way to present the MeasureIt-ARCH element properties using an approach that is more compact and less cluttered. De-cluttering the UI was necessary to make it faster for users to find the particular element they're looking for.

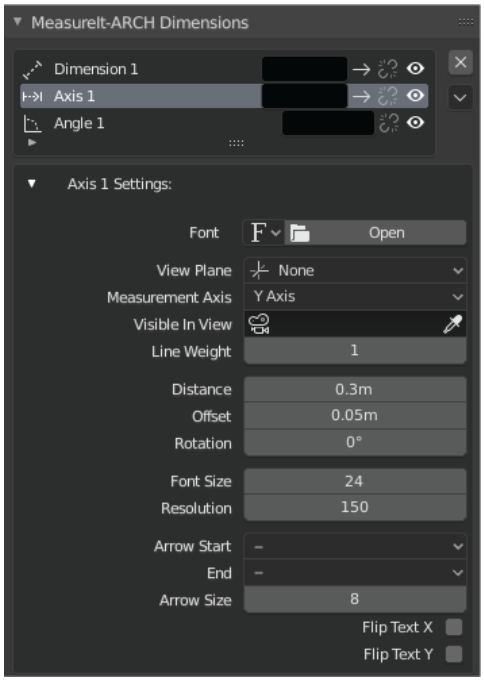


Figure 27: *MeasureIt-ARCH's List Style UI*

To accomplish this, a list-style user interface was implemented. The design of the list UI was based on Blender's existing UI for displaying other mesh data such as shape keys and vertex groups, as well as a discussion between Pablo Vazquez, a Member of Blender's UI Design team, and twitter user Fin around a potential list based redesign of Blender's modifier system UI.[54] The UI list displays all of the elements of a particular type, and the properties of the actively selected list item are displayed below the list in a collapsible UI box. This allows users to easily search through the list of elements, and compare their settings in a visually simple way.

[54]Fin O'Riordan and Pablo Vazquez, "Fin. On Twitter: "@PabloVazquez_ Soooooo..... More Like This Then? https://t.co/WcHnuNplBd" / Twitter," Twitter, accessed July 29, 2019, https://twitter.com/FinEskimo/status/1097551783327645699

Some technical tricks were required to allow multiple style types, or multiple dimensions types to appear in their respective lists, as Blender's current API for creating UI lists won't accept lists with more than one type of element. To circumvent this a dummy list that points to the relevant elements is used as a stand in for the list element type. This dummy list is automatically generated behind the scenes and should not impact the user experience in any way.

A schematic diagram of MeasureIt-ARCH's UI code can be seen on the following page.

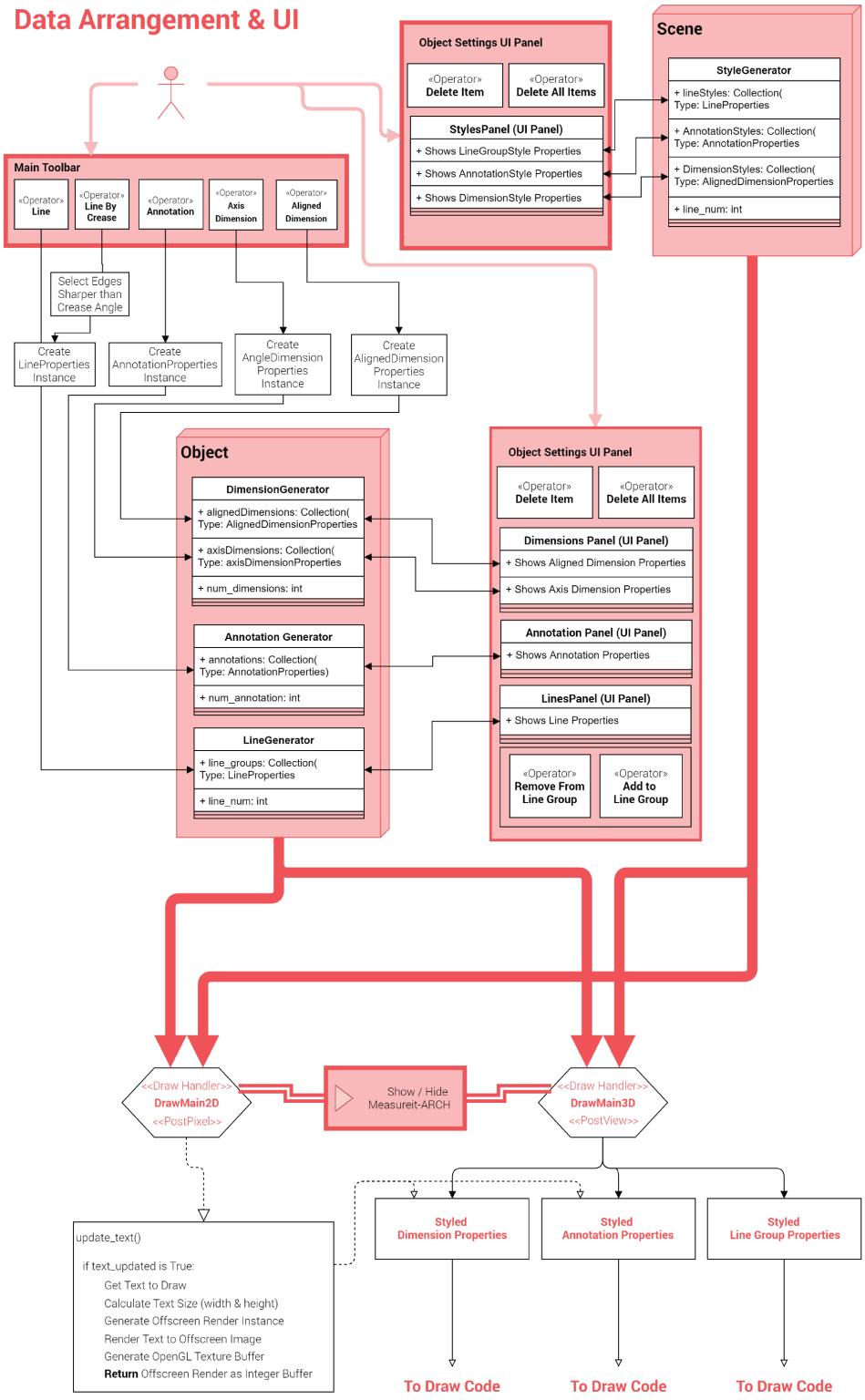


Figure 28: *Schematic Diagram of MeasureIt-ARCH's UI Code.*

3. Adding a Style System

To understand how MeasureIt-ARCH styles were implemented, let's consider an annotation style as an example. We can think of an annotation style as an instance of the annotation property group just like any other annotation. However, unlike a regular annotation, we need our annotation style to be stored in a place where all the annotations in the scene can reference it, and access its properties. This means that instead of storing our annotation style instance inside of an object's annotation Generator list as we would for a regular annotation, we need to store our annotation style somewhere more general. Luckily, Blender doesn't only support the storing of custom data lists on objects, but on all its major data types. Which means we can create a 'Style Generator' list as a custom property of the 3D scene itself in order to store our style instances in a more communal location. Now that we have a place to store our styles we can create a Styles UI Panel to allow our style properties to be displayed to be edited by the user.

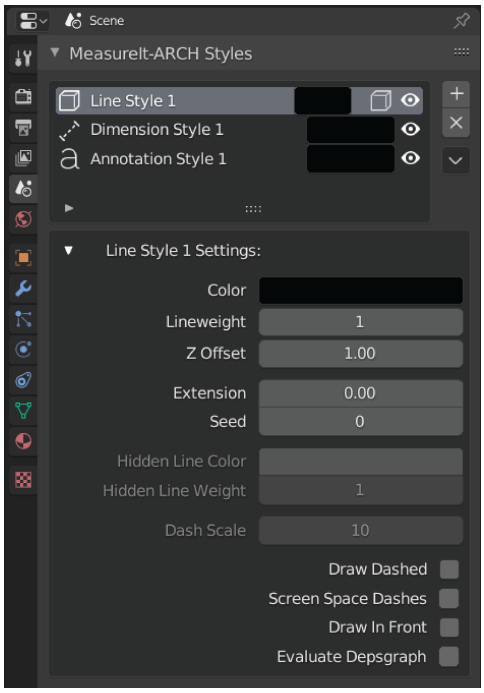


Figure 29: *MeasureIt-ARCH's Styles User Interface*

With our style stored in an accessible location, we need to edit our regular annotations property group to give it properties to identify and store a

reference to a style. First, we need to add a True or False property called 'is_style' which tells MeasureIt-ARCH if this particular instance of the annotation property group is a style or a true annotation instance to be drawn in the scene. Second we add another True or False property called 'uses_style' that tells the code if our annotation should look for a Style to use. Third, we need to add a property called 'style' to store a reference to the name of style we want the annotation to pull its properties from. Lastly in the annotation's draw method we need to check 'uses_style' to see if the annotation being drawn uses a style, and if it does, we override the annotation's visual properties, using the properties of style referenced in the 'style' property.

In our actual implementation, the 'is_style', 'uses_style' and 'style' properties are all added in the base property group, and inherited by all MeasureIt-ARCH elements. This allows every MeasureIt-ARCH element type to be instanced as a style, and to use styles.

4. Redesigning the Draw System to Work in 3D Space.

MeasureIt's original draw system worked in 'Screen Space'. Screen space is a two dimensional co-ordinate system that only contains information about an elements x and y position relative to the screen it is being drawn on. The way MeasureIt elements were being converted to screen space caused any information about the elements 3D location and depth to be lost before the element was drawn. Without this depth information all elements in the original MeasureIt would draw over top of the 3D scene objects, causing even simple scenes to become visually cluttered and confusing to read.

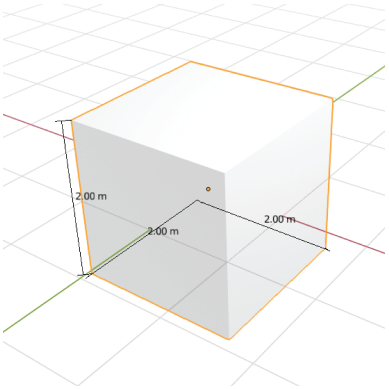


Figure 30: *MeasureIt's Original Dimensions*
Note these dimensions draw in screen space in front of a cube they should be occluded by

Converting MeasureIt's 2D screen space draw methods into 3D draw methods for MeasureIt-ARCH required a complete re-write of MeasureIt's original drawing system. These new 3D space draw methods not only allowed for proper depth ordering and occlusion of MeasureIt-ARCH elements, but they opened up the possibility to add new features like MeasureIt-ARCH's dynamic linework, and its topologically aware dimensions. Both of these features require 3D data and depth information that was discarded in the old drawing method. We'll discuss how each of these features makes use of this 3D information in more detail later in this section, but first we need to understand how we implemented a 3D draw method in Blender.

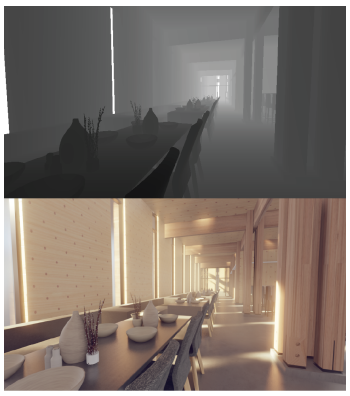


Figure 31: A Depth Buffer (Mapped to Greyscale), and its Corresponding Rendered Scene.

When writing a draw method for 3D space using Blender's API, there are a few essential steps.

1. Ensure that the draw method is being called from a Post-View draw handler: In Blender's API a variety of handlers are available that can be used to call methods in your add-on code after a particular event has occurred (i.e., saving, loading, frame changes, drawing etc.). Blender's Draw Handlers, which call a user-specified method every time Blender redraws the scene, are perfect for calling our draw methods, because we want our draw methods to run every time the scene has changed and is redraw by Blender.

Blender provides two types of draw handlers 'Post Pixel' and 'Post View', Post Pixel is used for methods that draw in screen space, and Post View is used for methods that draw in 3D space. [h]

[h] Although the differences between the two handlers isn't particularly well documented in the Blender API, I believe difference in implementation with these handlers results from their automatic configuration of the appropriate 3D or 2D projection Matrix used by any glsl shaders that are used within the handlers called method.

2. Enable drawing to, and testing against a Depth Buffer. The depth buffer can be thought of as a record of the distance of every drawn element from the current viewpoint. Using a depth buffer allows objects (or MeasureIt-ARCH elements) to be drawn in any arbitrary sequence, and then be checked against the depth buffer to see if any closer objects occlude them.

3. Ensure all co-ordinates output from the draw method are 3D: The draw method needs to determine the co-ordinates of every line, arrow, or circle that gets drawn. For lines, this is very straightforward as only the user specified start and end points of the line need to be sent to the shader. For dimensions and annotations, on the other hand, all of the components that make up the element (leader lines, extension lines, terminations etc.) and their positioning need to be calculated by the draw method for the shader to use.

In the original MeasureIt, after the distance between the two 3D points was calculated, the points were projected down to 2D screen space co-ordinates before calculating the positions of the rest of the elements components. This projection to 2D simplifies the calculations necessary to generate the rest of the drawing, but results in the elements depth information being sacrificed.

MeasureIt-ARCH on the other hand calculates the position of all of an elements components in 3D space. In some instances, this requires some extra information about the scene to ensure that elements like dimensions are placed reasonably and predictably. While this adds some computational complexity, it allows for elements to be placed more coherently in the 3D scene, and opens up the potential for more responsive features like MeasureIt-ARCH's dynamic dimension placement.

Once the placement of all an elements components have been calculated in 3D space, their co-ordinates are passed to a shader.

4. Use a shader that accepts 3D co-ordinates: Shaders are the programs that run on the GPU (Graphics Processing Unit) that draw the co-ordinates they receive to the pixels of the screen. While the bulk of a Blender Add-on is written in the Python programming language, shaders are written in the programming language GLSL. GLSL is used to access the OpenGL API, an API that Blender uses to communicate with the GPU hardware.

Blender's API does provides a set of basic built in shaders for drawing in 2D and 3D. The original MeasureIt made use of Blender's built in 2D shaders in its draw method for nearly all of its drawing. However, Blender's API also

allows you to define custom shaders. MeasureIt-ARCH makes use of this functionality to define it's own 3D shaders for its drawing. These custom shaders allow MeasureIt-ARCH to achieve effects such as; drawing lines thicker than one pixel, smoothing jagged edges of lines, drawing dashed lines, drawing silhouette lines, and drawing dimension and annotation text from a generated image texture.

Each shader is comprised of either two or three parts (depending on its desired outcome), which run in sequence.

1. **A Vertex shader:** The Vertex shader converts the 3D World Space (x,y,z) co-ordinates provided by the draw method, to 4D Homogeneous Clip Space co-ordinates (x,y,z,w) which are a projection of the 3D co-ordinates based on the 'model view projection matrix'. In short, Clip Space co-ordinates allow shaders to calculate perspective projections based on the user's current viewpoint through the mathematical operation of matrix multiplication, which the GPU' hardware is optimized for. For a detailed and intuitive discussion of why homogeneous co-ordinates are used Squirrel Eiserloh's 2015 GDC presentation provides an excellent summary.

In the vertex shader, after our co-ordinates have been projected, we can also offset their depth, in Clip Space, from the current viewpoint. MeasureIt-ARCH does this occasionally to fix rendering errors with overlapping elements, or to achieve silhouette outlines.

2. **A Geometry shader:** The Geometry shader is optional, but it lets us generate new lines or shapes based on the input co-ordinates we received from the draw method. MeasureIt-ARCH uses a Geometry shader to control a lines thickness. The Geometry shader expands the start and end points of the line received from the draw shader, into a rectangle whose width is determined by the line weight property. Calculating this expansion in the geometry shader makes it easier to manage lines in the main body of the add-on, and, since this particular expansion is conducted in screen space, it allows for line thickness to stay consistent regardless of the screen's aspect ratio or the distance of the line from the viewer.

3. **Fragment shader:** The Fragment shader converts the shapes defined in the Vertex or Geometry shaders into pixels on the screen and describes how these pixels should be coloured. MeasureIt-ARCH's Fragment shaders assign an elements colour based on its user defined colour property. In MeasureIt-ARCH's line shader, the Fragment shader also fades the opacity of the line close to its edges, to reduce jagged

pixilated artifacts on diagonal lines. Fragment shaders can also colour a shape by mapping an image texture to it based on the UV co-ordinates associated with the shape. [i]. MeasureIt-ARCH uses image textures to display dimension and annotation text. The text itself is rendered to a 2D texture, earlier in the drawing process, and then mapped onto a simple rectangular card located in the appropriate position and orientation in 3D space. Rendering the text as a simple UV mapped card makes it simple to re-orient an elements text to maintain a legibility, as the 4 UV mapping co-ordinates of the rectangle can be flipped according to the user's viewpoint.

[i] UV co-ordinates are a set of 2D co-ordinates that describe how a point in 3D space could be unwrapped to lie in a 2D image plane, and are typically used for mapping textures to a 3D surface

A schematic diagram of MeasureIt-ARCH's draw methods and shaders can be found on the following page.

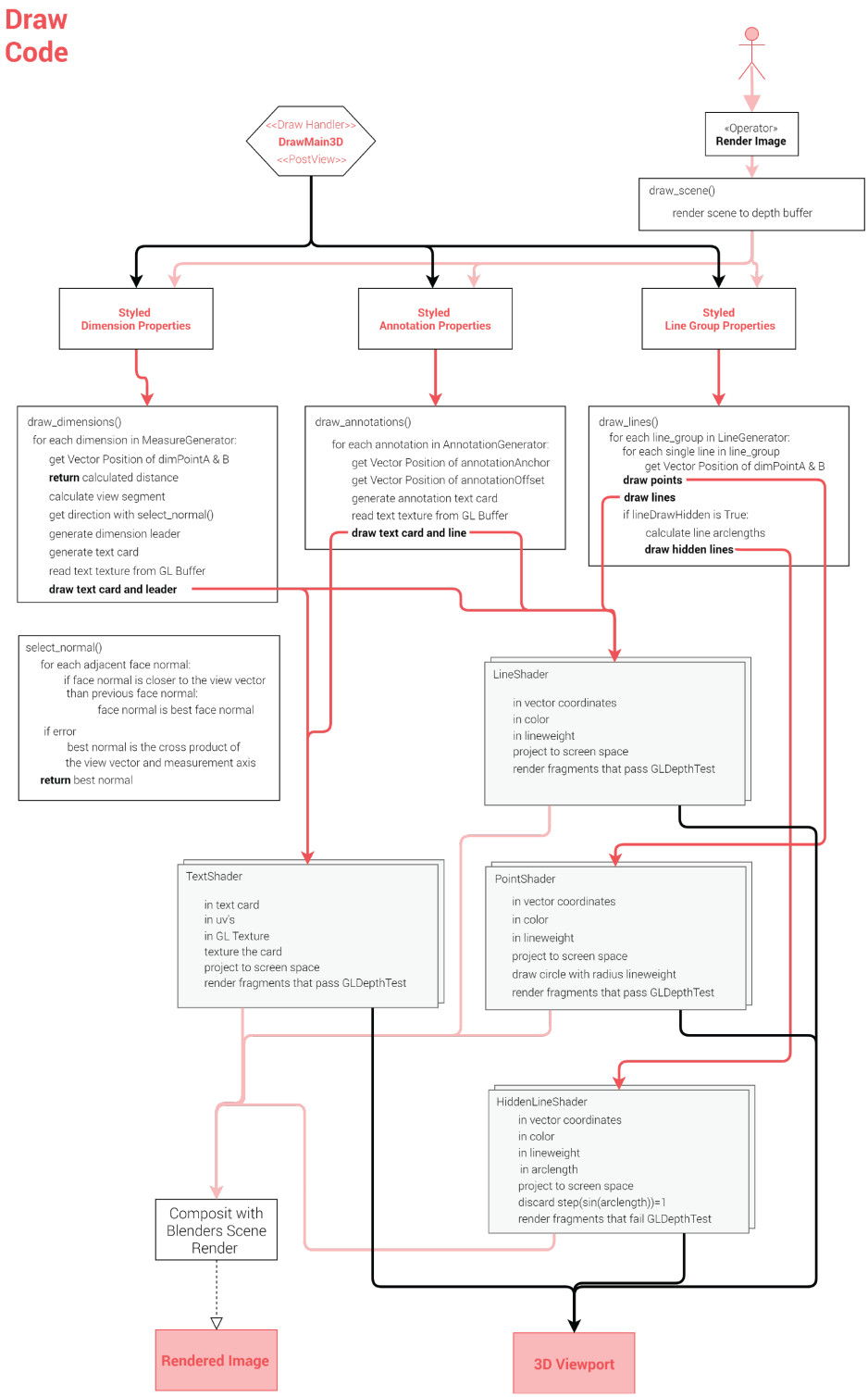


Figure 32: *Schematic Diagram of MeasureIt-ARCH's Draw Code.*

Implementing Linework and Dimensions

Limitations and Design Constraints

Now that we have a high level understanding of how Vazquez's MeasureIt was adapted and updated to become MeasureIt-ARCH, lets take a moment to explore in more depth how MeasureIt-ARCH implements its main features. Namely the drawing of linework and dimensions. Each of these area's has a rich history of research and development and numerous algorithms and implementations developed by computer graphics researchers associated with it. This section of the thesis provides a brief look at some of the key developments in the history of each of these features, and how these precedent approaches have influenced MeasureIt-ARCH's implementation of line drawing and dimensioning tools. MeasureIt-ARCH itself does not implement any of these techniques verbatim however, opting instead for implementations that are, as a general rule, unique but much less sophisticated.

These unique but simple implementations are necessitated by a few factors. Foremost, my own novice experience as a programmer means that simpler solutions are always more achievable. There are however other technical constraints on MeasureIt-ARCH's implementation of linework, and dimensioning tools that are imposed by its nature as an add-on for Blender, rather than a standalone application. As an add-on written in Python MeasureIt-ARCH is constrained by Python's comparatively slow execution speed, as Python code typically runs several time slower than lower level languages like C or C++. For most aspects of MeasureIt-ARCH's code this difference in speed is imperceptible, however, in certain areas, such as the draw methods, where MeasureIt-ARCH may potentially loop over hundreds of thousands of drawing elements, and needs to do so in a fraction of a second, Python's slower execution can be a challenge. In these loop intensive areas Python's slower execution necessitates some creative solutions to ensure MeasureIt-ARCH remains responsive.

This technical constraint prevents more computationally intensive drawing techniques from being implemented directly in MeasureIt-ARCH (by a programmer of my skill level). There is however another constraint that influences how MeasureIt-ARCH implements its line drawing and dimensioning systems that is worth noting. This constraint is not technical or skill based in nature but rather one imposed by the stated goal of this thesis work. That is, that the end product of this work, MeasureIt-

ARCH, needs to be a **functioning** Open Source platform for the creation of architectural drawings within Blender that is publicly, and freely available for all to use. The methods employed in MeasureIt-ARCH are not intended to be primary research, new developments, or even implementations of cutting edge best practices in the field of computer science, and by no means would they qualify as such. The methods employed by MeasureIt-ARCH are however, functional. They meet the majority of the requirements outlined in the specification presented in the first section of this thesis, and as we will show in the testing and implications section of this thesis they are capable of producing simple architectural drawings within Blender. To this end, MeasureIt-ARCH focused on simple methods and algorithms, especially in the area of line drawing, that were tailored to the specific subset of necessary features needed to meet our defined specification, rather than attempting to utilize more sophisticated processes with broader feature sets.

That being said, to achieve functional implementation MeasureIt-ARCH borrows and combines methods from across the existing body of research in its solutions, to arrive at hybrid techniques. The following describes in more detail how these hybrid methods were derived from the background research that MeasureIt-ARCH draws on.

Line Drawing

An Overview of MeasureIt-ARCH's Line Drawing

MeasureIt-ARCH Line Groups are responsive to their context in the 3D scene. If a Line Group is hidden by other geometry, or if it represents the silhouette of an object, it's appearance is adjusted, based on its user-defined Line Group properties. This means that easily readable line drawings and diagrams can be produced without the need for the user to define each line's state, and resulting visual properties, individually. Line Groups contain a subset of a Blender object's mesh edges. The edges contained in a Line Group can either be selected manually by the user, allowing for fine grained control of the lines drawn, or selected algorithmically through the use of the Line Group by Crease operator. This operator adds all mesh edges that define a crease greater than a user specified angle to a new Line Group. The edge's crease angle is determined by the rotational difference between the edge's adjacent face normals.

Background Research

MeasureIt-ARCH's Line drawing system is a hybrid of many early techniques in the area of 'Non-photorealistic Rendering' (NPR for short). Research into NPR algorithms for the production of 3D computer graphics dates back to the early 1960's and can generally be divided into two key areas of study based on the desired visual outcome. First are NPR algorithms aimed at the production of technical illustrations and line drawings. The second would be NPR algorithms aimed at imitating or replicating the look of images produced by hand with more traditional artistic media, (pencil, pen and ink, charcoal etc.). As may be expected, MeasureIt-ARCH focuses predominantly on the first area of research, as the second introduces a large degree of complexity, and lies outside of our previously established scope. [j]

Before we look at any specific algorithm that deals with the generation of linework for technical drawings, lets establish what types of problems these algorithms need to solve. In general, existing algorithms for computing and rendering linework deal with 3 distinct problems;

1. **Which lines do we want to render?**
2. **For each line; how do we determine if all or part of this line is visible?**
3. **What visual characteristics do we give these lines based on their determined visibility?**

[j] Though, it is worth a brief note here however to draw attention to the fact that Blender already comes well equipped features to generate traditionally styled works. The previously discussed Grease Pencil tools allow for direct hand sketching within the 3D environment, with textured strokes similar to a 2D raster graphics package, and the Freestyle render engine allows for the algorithmic generation of stylized and textured strokes from 3D models. However, neither of these tools is efficiently suited to the production of responsive, technical Illustrations.

Although there are a myriad of technical line drawing methodologies, we can categorize them in to two distinct classifications based on how they approach their solution to these three problems.[55]

1. **Object Space algorithms**
2. **Image Space (or Screen Space) algorithms**

Object Space algorithms conduct their calculation in 3D space. They make use of the three dimensional geometric relationships between objects, such as occlusion, face adjacency, and differences in face normals, in order to compute the relevant lines to be drawn, and their visibility.

One of the most commonly known object space line drawing methods is Arthur Appel's Quantitative Invisibility algorithm.[56] Developed in 1967, Appel's Quantitative Invisibility uses a combination of techniques and geometric information such as a face's, adjacent faces, its implied vorticity, as well as the parametric representation of its edges, in order to calculate geometric intersections, or piercing points. These piercing points represent where a line intersects or passes behind a surface, and each piercing point marks changes in the visibility state of the line. Determining where these changes in visibility occur allows for a quantitative measure of the number of times a line is occluded by a front facing surface, hence 'Quantitative Invisibility'. Using this measure of Quantitative Invisibility, decisions can be made about how, or if, the invisible line segment should be drawn. The maths that drive Appel's algorithm are quite elegant, and the algorithm produces line drawings that are very accurate and consistent across multiple views. Quantitative Invisibility's key drawback is that, like all object space line drawing algorithms, the computational cost required to draw each view scales proportionally with the geometric complexity of the scene.

Image Space algorithms on the other hand, generally have a computational cost that scales with the resolution, in pixels, of the image being produced. At their simplest, Image Space line drawing algorithms make use of buffers; images that store key information about the scene such as depth, or surface normal direction, as pixel values. These buffers can be processed and utilized in a variety of ways. In some cases, 2D edge detection filters, such as a Sobel filter, can be applied to a depth or normal buffer to calculate linework based on discontinuities in the pixel values of the buffer.[57] In others, the entire wire frame of the object can be rendered after the solid view of

[55] Nadia Magnenat Thalmann and Daniel Thalmann (Tokyo: Springer Japan, 1990).

[56] Appel, "The Notion of Quantitative Invisibility and the Machine Rendering of Solids."

[57] Philippe Decaudin, "Cartoon-Looking Rendering of 3D-Scenes," Research Report (INRIA, June 1996), http://phildec.users.sf.net/Research/RR-2919.php.

the object, and tested against the scene's depth buffer to determine it's visibility. Line segments with a pixel depth value larger than the value stored in the depth buffer are determined to be occluded and can be discarded. Many algorithms are based on these simple principles, but use different manipulations, or combinations of buffers in order to achieve different styles.[58]

More complex Image Space solutions, such as those described by McGuire et al,[59] and the Freestyle system developed by Grabli et al[60] parametrize the lines extracted from their image space calculations to allow for finer stylized control over properties such as line thickness, opacity, colour and texture.

While Image Space algorithms can be calculated quite quickly, when compared Object Space algorithms, they often struggle at obtaining the same degree of accuracy, and consistency demonstrated by their Object Space counterparts. This measure of accuracy, and consistency in image space line drawing is generally referred to as 'temporal coherence.'[61] The challenge with achieving good temporal coherence in Image Space algorithms arises from the fact that the lines being drawn are not derived directly from properties of the 3D geometry itself, which are generally consistent regardless of the users view point, but from Image Space pixel buffers, who's discontinuities may change from view to view. Issues with temporal coherence manifest as a 'flickering' of the linework as the viewpoint changes.

[58] Bruce Gooch et al., "Interactive Technical Illustration," in *Proceedings of the 1999 Symposium on Interactive 3D Graphics - SI3D '99* (The 1999 symposium, Atlanta, Georgia, United States: ACM Press, 1999), 31–38, https://doi.org/10.1145/300523.300526.

[59] Morgan McGuire and John F. Hughes, "Hardware-Determined Feature Edges," in *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering - NPAR '04* (The 3rd international symposium, Annecy, France: ACM Press, 2004), 35, https://doi.org/10.1145/987657.987663.

[60] Grabli et al., "Programmable Rendering of Line Drawing from 3D Scenes."

[61] Ibid.

The Freestyle Line Drawing System.

Of particular interest when discussing a technical line drawing system for Blender, is the Freestyle line rendering system. As mentioned above Freestyle is an image space line drawing system, developed by Grabli et al. Although Freestyle is just one of many current day line drawing algorithms, it is of direct relevance to us here because the Freestyle system is already implemented within Blender, and MeasureIt-ARCH's line drawing system has been developed in direct response to Freestyle's existing capabilities and methods. As such we should understand what Freestyle is, how it functions, and what use cases it is intended for.

Freestyle is a programmable, modular system for the generation of line drawings from 3D models. Freestyle works in image space, but instead of extracting its lines using image filters applied to pixel buffers and drawing these lines directly to the screen, Freestyle makes use of a two stage approach. First Freestyle generates a 2D representation of the scene called a view map. The view map is a data structure that contains a parametric representation of the scenes feature lines and vertices as they appear at that particular viewpoint. The view map also contains additional metadata about each of these feature lines, such as their type (Border, Silhouette, Crease, etc.) and visibility. The parametric nature of this data contained in the view map is the key to Freestyle's flexibility as a line drawing system, as it allows for the view map data to be interpreted in a variety of ways. Once the view map has been calculated, different style modules can operate on its data to produce a variety of different styles of line art. For Freestyle's implementation in Blender, these style modules can be defined through a plethora of settings present in the user interface, or written manually as a Python script.

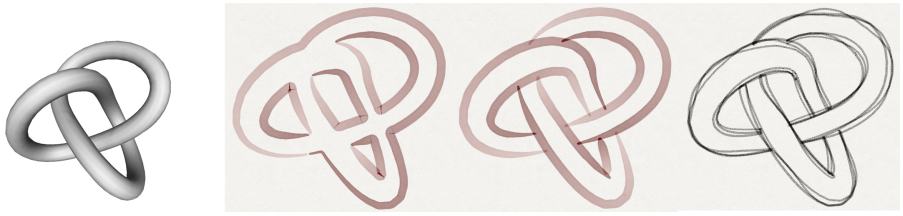


Figure 33: *Multiple Line Styles Rendered with Freestyle*
A torus knot rendered with three unique Freestyle Style Modules.

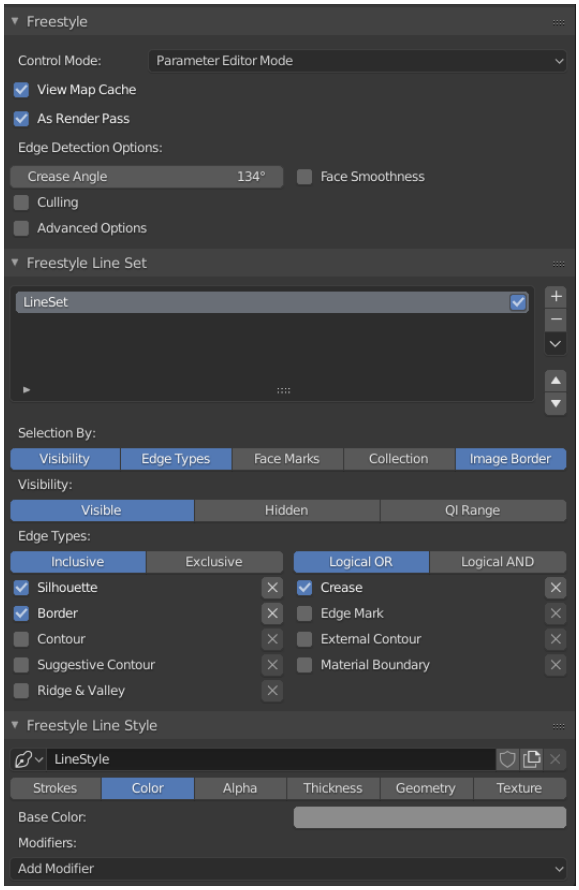


Figure 34: *Blender Freestyle's User Interface*
Freestyle's UI within Blender provides a plethora of options for specifying the edge types to be rendered, as well as the desired styling for those lines.

Freestyle's flexibility and large range of potential styles is ideal for its main use case; the creation of stylized non photorealistic renderings that mimic traditional artistic mediums. However, when it comes to our desired use case; the creation of technical illustrations for architectural drawings, the cost of Freestyle's complexity can start to negatively impact its user experience and efficiency, when applied to this task. There are a few key aspects of Freestyle's implementation and user experience that we should note to avoid when developing MeasureIt-ARCH.

1. Freestyle is a post process; Freestyle runs only after Blender renders a scene. This means that for a user to see the effect of their changes to a style module, the entire scene must be re-rendered. For complex scenes, this makes tweaking linework incredibly time consuming.

2. Freestyle offers limited opportunities for direct user intervention when selecting lines; Freestyle's view map handles all calculations regarding which lines will be drawn in the scene, however nearly all of these lines are generated algorithmically, and there is no ability for the user to tweak the view map after its creation. There is one line type, the Edge Mark type, that can be added to the view map based on user defined edges, but there is no ability to remove algorithmically generated lines that the user doesn't want to see rendered. The only option available for tweaking the view map is to adjust its calculation parameters and re-render the scene. This means that the fastest and most practical way to make changes to the generated linework from Freestyle is often to export it to an external image editing software after the linework has been rasterized and edit the image manually.

3. Complex interface for defining styles; Freestyle predominantly relies on python scripting for the definition of style modules. Though Blender's implementation does provide a basic user interface to create style modules without the need for scripting, this interface is still quite complex, with a huge variety of options and settings available in order to facilitate Freestyle's vast range. Many of these settings are highly technical in nature and not intuitive for an end user. The level of mastery required to efficiently navigate and use this interface can be daunting when one is only attempting to achieve simple results.

For our purposes, these workflow challenges can be seen as symptoms of Freestyle's complexity. While one solution might be to find optimizations in Freestyle's algorithms or implementation that could maintain its flexibility while resolving these issues, our approach however will be one of optimization through reduction. A line drawing system for the creation of architectural drawings does not require the stylistic flexibility that Freestyle offers, but it does need to be responsive, direct, and intuitive.

MeasureIt-ARCH's Implementation

With the benefits and pitfalls of Object Space and Image Space line drawing algorithms, and our takeaways from the Freestyle system, in mind, lets review our three key line drawing problems introduced at the beginning of this section. We'll use our insights from the previous discussion to start to derive MeasureIt-ARCH's approach to each problem.

1. Which Lines or Edges do we Want to Render?

To answer this question we need to understand a few things, how do we classify the types of lines that make up a drawing, which line classifications do we actually need to draw to create a readable technical drawing, and then how do we go about selecting and saving these lines for later drawing. As noted in our critique of the Freestyle system, one of our major priorities with MeasureIt-ARCH's line selection process is going to be providing a simple method for the user to add or remove lines from any automated line selection processes.

Line Types

To get a sense of the types of edges that can be calculated by a line drawing system, lets look at the line types that Freestyle can calculate, and see which types make sense to include for our application. (Note: these line type descriptions are take directly from Blender's Freestyle UI descriptions, we'll be improving on these definitions soon).

1. **Silhouette Lines;** Edges at the boundary of visible and hidden faces.
2. **Border Lines;** Open mesh edges (Non-Manifold edges).
3. **Contour Lines;** The outer silhouette of an object.
4. **Suggestive Contour Lines;** Edges that are almost silhouette or contour edges.
5. **Ridge & Valley Lines;** Boundaries between the convex and concave areas of the surface.
6. **Crease Lines;** Edges between two faces making an angle smaller than the crease angle.
7. **Edge Mark Lines;** Edges annotated by Freestyle edge marks.
8. **External Contour Lines;** The outer silhouette of occluding and occluded objects.
9. **Material Boundary Lines;** Edges at material boundaries.

The differences between some of the Freestyle line types are subtle and can often be difficult to intuit from their written descriptions. The figure below

illustrates the 5 most common Freestyle line types, rendered without styling on two Utah Teapots.

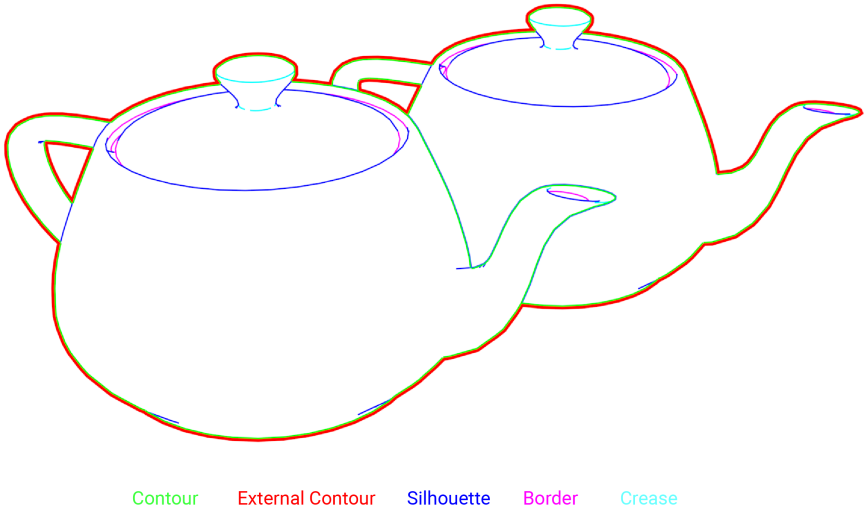


Figure 35: *Freestyle's Line Types*
Contour Lines(Green), External Contour Lines(Red), Silhouette Lines (Blue), Border Lines(Magenta), Crease Lines (Cyan)

Each of Freestyle's 9 Line types can be included in Freestyle's view map calculation, and its worth noting that not all of these line types directly correspond with the mesh edges of a 3D model. Some of these lines, such as the Ridge and Valley, or Suggestive Contour lines are interpolated surface features that may not have a corresponding mesh edge.

Now, our technical line drawing system is meant to deal predominantly with the depiction architectural forms, where hard edge creases are the most common and most essential for describing form. These crease lines should be our predominant focus for the MeasureIt-ARCH system. The second most important, given our priority of user intervention, should be something akin to the Edge Mark line type that Freestyle provides, but with the ability to not only add but also remove lines. We also want the MeasureIt-ARCH system to be responsive and work in 'real time' in Blender's 3D view port. To achieve this, we should stick with a relatively simple set of line types. With these considerations in mind our proposed list of edge types for MeasureIt-ARCH looks something like this;

1. **User Defined Lines;** A user should be able to select any pair of vertices, and create or remove a line defined by those vertices.

2. **Crease Lines;** A user should be able to specify a crease angle, and any edge whose adjacent faces form a crease greater than that angle should be marked as a line to draw.
3. **Variable Silhouette Lines;** The Variable Silhouette line type is MeasureIt-ARCH's naïve approximation and amalgamation of Freestyle's External Contour, Contour, Silhouette, and Border line types. Which of these analogous Freestyle line types the Variable Silhouette Line behaves like, is determined by a single user editable property. Although hard edged forms are our primary focus, variable silhouette lines should provide a basic option to clearly depict simple organic forms through linework.
4. **Hidden Lines;** Hidden lines can be either user defined lines, or crease lines that are occluded in the current view by other geometry.

With our basic desired line types defined, we need to devise a method to calculate and store these lines. A data structure for MeasureIt-ARCH that could be considered analogous to Freestyle's view maps.

Line Groups; Storing Lines to be Drawn

Now that we have an idea of which line types we would like to be able to draw, We can start to work on the sequencing of how these lines, once selected, are stored and represented in memory for later styling and drawing. To use Freestyle as a comparative example again, once Freestyle's various line types have been calculated, they are stored in the two dimensional View Map data structure, before the style modules are applied and the result is drawn to an image. There are two main pitfalls of this approach that we'd like to avoid. First, because the view map is a two dimensional data structure, the calculations that select which lines should be drawn need to be re-run each time our viewpoint in the scene changes. Second, because the style modules and drawing process are applied *immediately* after the view map is calculated, there is no opportunity to add or remove unwanted algorithmically generated lines after the view maps creation.

To address these two issues, MeasureIt-ARCH calculates and stores the lines that need to be drawn in Object Space, and separates the line selection and calculation process from the drawing process entirely. This object space storage of lines to be drawn is what MeasureIt-ARCH refers to as a Line Group. Line Groups are a type of property group that stores the selected line segments, as well as their user defined properties. Line Groups, like all drawn property group instances in MeasureIt-ARCH, are stored in a Generator list, in this case the 'Line Generator', attached to the Object who's geometry they reference.

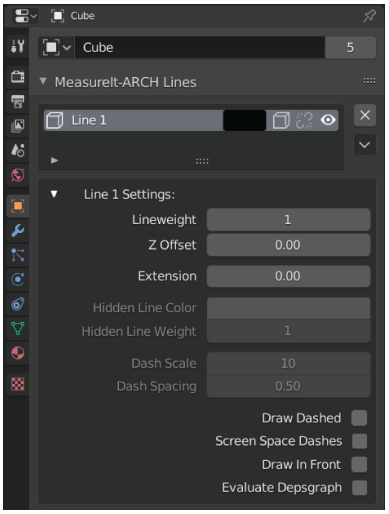


Figure 36: *A Line Group's User Interface Settings*

Utilising Line Groups makes it so that the line selection and calculation process, generally the slowest part of a line drawing algorithm, needs to only be run once for a given set of geometry. Once the Line Group has been populated with the relevant lines, these lines can be drawn from any perspective, with the relevant perspective transforms being applied to the Object Space data in hardware through the shader program running on the GPU, to render the image to the screen. This separation of line selection and line drawing means that the two tasks can be conducted in parallel, allowing the user to modify the contents of a Line Group at any time, while the draw system draws the Line Group's current contents to the screen.

How does a user populate and edit a Line Group though? MeasureIt-ARCH offers two main methods, one algorithmic, one direct.

The Line Group By Crease Operator

The Line Group By Crease Operator is currently MeasureIt-ARCH's only algorithmic solution for adding lines to a Line Group. In principle, the Line Group by Crease operator checks, for each edge in the mesh, if the faces adjacent to that edge form an angle greater than some user specified threshold. If the angle formed by the faces is greater than the threshold, then the edge is added to a Line Group created by the operator, if not then the edge is ignored. The Line Group by Crease operator also gives the user the option to decide if non-manifold edges (edges with more or less than two adjacent faces) should be added to the newly created Line Group.

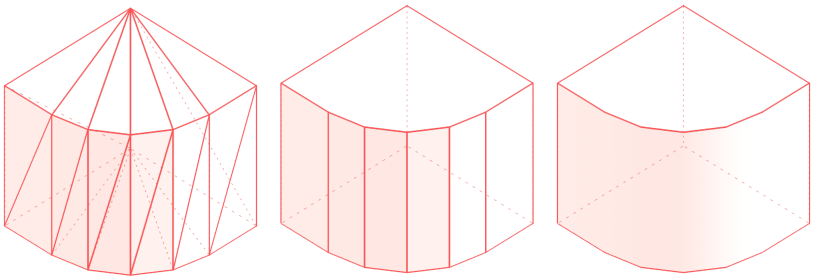


Figure 37: *The Line Group by Crease Operator's Behaviour. A cube with one rounded corner drawn with (left) MeasureIt-ARCH linework on all mesh edges. (center) MeasureIt-ARCH linework generated by the Line Group by Crease Operator using a crease threshold angle of 10 degrees. (right) MeasureIt-ARCH linework generated by Line Group by Crease Operator using a crease threshold angle of 30 degrees*

Direct Selection

MeasureIt-ARCH's primary method for creating and editing Line Groups is through the direct selection of mesh edges. A new Line Group can be created with the Line Group operator. This newly created Line Group will be populated with the mesh edges that the user has selected at the time the operator is run.

MeasureIt-ARCH also provides two additional Operators that allow for direct line selection. The add and remove line operators allow for the modification of existing Line Groups. Running either of these operators will respectively add or remove the mesh edges currently selected by the user to the active Line Group, allowing for easy adjustment of existing Line Groups. The add line, and remove line operators can be added to Blender's quick favourites menu or assigned a keyboard shortcut for even faster workflows.

MeasureIt-ARCH Line Groups; Benefits and Limitations

While the Object Space approach of MeasureIt-ARCH's Line Groups allows for users to fine tune the selection and storing of lines, and ensures that selection calculations don't need to be run every time the users viewpoint changes, these benefits do come with certain downsides. When compared to the selection and storage methods of a sophisticated Image Space algorithm such as Freestyle, MeasureIt-ARCH's Line Groups contain much less useful metadata about a lines condition. Lines in a MeasureIt-ARCH Line Group have no concept of their visibility state, or their relation to other

lines in Image Space.

This lack of relation to other lines in Image Space means that some of the more advanced features for stylizing lines offered by a system like Freestyle are not possible in MeasureIt-ARCH. For example, stylizations such textured strokes in Freestyle rely on chaining together individual line segments based on their proximity in image space. This line chaining allows for the creations of longer strokes along which the desired texture can be mapped, resulting in a more natural looking textured stroke. A lack of Image Space chaining does limit MeasureIt-ARCH's artistic flexibility, but as our primary use case is the creation of technical drawings and diagrams, this seems to be an acceptable trade off for the improved control over line selection, and the added ability to draw lines directly in the 3D viewport in real time.

MeasureIt-ARCH's Line Groups face one other major limitation. That is that the lines included in a Line Group are tied directly to an objects mesh geometry, that is to say that a line in MeasureIt-ARCH Line Group cannot be defined by any arbitrary pair of points in 3D space. Instead a line segment within a Line Group has to be defined by a pair of vertex indices in the mesh. Having lines directly tied to mesh geometry does mean that certain line types, that we deemed non essential for MeasureIt-ARCH, such as suggestive contour or ridge and valley lines cannot be drawn directly with MeasureIt-ARCH. It does however give us some performance benefits that help with MeasureIt-ARCH's real time performance. If MeasureIt-ARCH were to define it's line segments through pairs of 3D co-ordinates, this would mean that every time a mesh object was moved, edited, or deformed, the user would need to re-define the lines that should be drawn for that object, as the modified object would no longer match with the co-ordinates defined in the Line Group. Instead by defining line segments as a reference to a pair of vertex indices the current position of the vertices can be checked at draw time. This allows Line Groups to adapt to modifications in the model, without needing to re-calculate line selection. This ability to adapt to mesh modifications also allows MeasureIt-ARCH linework to be compatible with many of Blender's toolsets for mesh deformation, such as Shape Keys, which allow the user to store and blend between different variations of a mesh, or Blender's modifier system, which allows for parametric deformation of mesh objects. By defining lines using pairs of vertex indices rather than absolute 3D coordinates these deformations can be accounted for by MeasureIt-ARCH's Line Groups in real time.

Of these limitation two in particular lead us nicely into our second problem of line drawing. Firstly we mentioned here that the lines stored in Lines Groups

have no defined visibility, how then do we determine which lines are visible to be drawn? Second, we've provided no solution yet for the drawing of our third and fourth desired line type for MeasureIt-ARCH, the Variable Silhouette line type and the Hidden line type. Given the fact that Line Groups are not re-calculated for each view, have no visibility information and are directly tied to the mesh geometry of an object, view dependent line types like these might seem difficult to achieve in this system. Both of these challenges are addressed however, in MeasureIt-ARCH's solution to the second problem of line drawing.

2. For Each Line; How do we Determine if All or Part of this Line is Visible?

While MeasureIt-ARCH handles the process of selecting lines to draw in Object Space. It handles all of it's visibility calculation in Image Space. This switch is again to help improve performance to allow for the real time display of MeasureIt-ARCH linework. Object Space visibility calculations, like Appel's Quantitative Invisibility, rely on an early form of a technique called ray casting. In Appel's method, first a series of calculation are carried out to determine where the visibility on the line might change (the piercing points where it intersects a surface for example). Then a ray is projected from the users viewpoint, to each line segment who's visibility is in question. This ray can then be evaluated to determine the number of front facing surfaces it intersected on its way to the line segment in question. The number of intersections this ray cast experiences is equal to the number of surfaces occluding the line who's visibility we are calculating, and this number of occluding surfaces is that line segments quantitative invisibility. There are two main drawbacks for this approach if we were to apply it in MeasureIt-ARCH. First it requires that all lines be defined in their parametric form, whereas MeasureIt-ARCH defines it's line as pairs vertex indices, and later at draw time as pairs of cartesian co-ordinates. This limitation could be worked around, but the second, more significant limitation, is that the number of calculations required, and therefore the speed of these Object Space visibility algorithms, is directly tied to the amount of geometry in the scene.

Instead MeasureIt-ARCH uses Z-buffering for its visibility calculations. Z-buffering is an Images Space technique for determining visibility that operates per-pixel to produce a raster image of the scene. Since Z-buffering operates on a per-pixel basis, the drawing speed is most directly influenced by the resolution of the produced raster image, instead of the amount of geometry in the scene. This is ideal for achieving the real-time viewport performance we are aiming for with MeasureIt-ARCH. Z-buffering does of course come with some limitations, but before we describe them, we should understand how Z-buffering works to determine basic visibility calculations, and how MeasureIt-ARCH manipulates this basic functionality to draw its Hidden, and Variable Silhouette line types.

How does Z-Buffering Work

Z-buffering determines a drawn element's visibility by checking each rendered pixel against the Depth Buffer. The depth buffer, as we briefly introduced in the previous section, is a per pixel record of the distance each drawn pixel is from the users viewpoint. When using Z-buffering to determine basic visibility, each pixel to be drawn has it's distance from the users viewpoint checked against the current value in the depth buffer for that pixel. If the pixel to be drawn has a distance greater than what is already present in depth buffer then it is discarded, as it must be occluded by some closer object which has already been drawn. If however, the pixel to be drawn has a distance that is less than the value currently in the depth buffer, then the pixel to be drawn overwrites the value in the depth buffer with its depth value, and overwrites the pixel the rendered image with its color value, as it must be closer viewpoint than anything that had been drawn to that pixel previously.

Why do we call this Z-buffering? Well, if we think of the width and height of our image as its X axis and Y axis respectively, then we can think of this 'distance from the users viewpoint' as our images Z axis, stretching out backwards perpendicular to the image plane. A sort of 3D version Image Space where the cartesian X, Y and Z Axes are aligned with the users viewpoint. This '3D Image Space' is generally referred to as View Space or Camera Space.

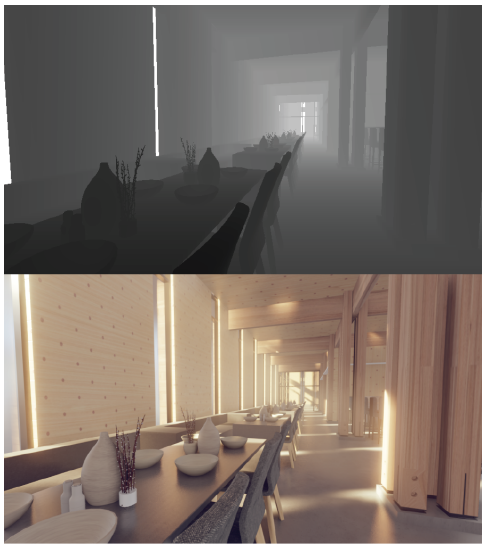


Figure 38: *Depth Buffer Example.*
A Depth Buffer (mapped to greyscale), and its corresponding rendered scene.

Setting up Z-Buffering in Blender's API

Enabling depth buffer testing though Blender's API requires only a few lines of code.

'*bgl.glEnable(bgl.GL_DEPTH_TEST)*'; Enables depth testing when drawing

'*bgl.glDepthMask(True*)'; Allows new draw passes to be written to the depth buffer.

After depth testing is enabled, you can define what test function should be used (When using an OpenGL depth buffer, smaller values are closer to the viewpoint)

'*bgl.glDepthFunc(bgl.GL_LEQUAL)*' This is the typical visibility test described above. With this depth test enabled only parts of an object, or MeasureIt-ARCH element, that are not occluded by anything closer to the users viewpoint will be drawn (The test passes if the new Z-value is less than or equal to the current Z-value in the depth buffer for that pixel)

'bgl.glDepthFunc(bgl.GL_GREATER)' This test does the opposite, only parts of an object that *are* occluded by an object closer to the viewport will be drawn. (The test passes if the new Z-value is greater than the current Z-value in the depth buffer for that pixel)

Hidden Line Drawing

MeasureIt-ARCH takes advantage of these different types of depth testing in order to achieve It's Hidden Line type. Line Groups with hidden line drawing enabled are actually drawn not once, but twice. First we draw the Line Group with a standard GL_LEQUAL depth test and a plain line shader. This gives us our standard lines that aren't occluded by anything. Next we draw a second pass using the GL_GREATER depth test and a dashed line shader. This draws any lines in the Line Group that are occluded with a dashed appearance. The result of this two pass approach is that the style of the line that is eventually drawn is determined by its depth in relation to the other objects in the scene. This method also maintains user control over which particular Hidden Lines are drawn, as this two pass approach applies directly to the user defined Line Groups. This two pass approach is quite conventional for Z-buffer based hidden line drawing, making use of readily available functions present in the OpenGL API. Our approach to silhouettes will need to be a bit more creative however.

Variable Silhouette Lines

The key functionality of Variable Silhouette lines in MeasureIt-ARCH is to replicate, at least in a basic way the functionality of Freestyle's External Contour, Contour, Silhouette, and Border line types. Since we are only interested in drawing simple solid lines, we don't need to achieve this in a way that allows for line chaining or any of the more advanced stylization options that Freestyle offers, we simply need a way to draw clean continuous lines that replicate the appearance of these four Freestyle line types. Most importantly to keep our silhouette system fast enough to run in real time we want to create a method that doesn't require visibility calculations or line selection to be carried out on the CPU, but rather a method that runs in our shaders on the GPU. There exists no readymade OpenGL function that makes this easily implementable, like there was for hidden lines, so we'll need to look closer at conventional methods for silhouette drawing in shader as well as each of our four Freestyle line types, to try and derive our own method.

Conventional Silhouette Drawing; The Inverse Hull Method One of the most conventional methods for drawing silhouette lines around an object in shader is commonly known as the 'Inverse Hull Method'. Although I've found little scholarly material on this technique, it sees fairly frequent use in video games[62] and NPR renderings, and seems to be a sort of vernacular approach to silhouette line rendering. The Inverse Hull method works by duplicating the entirety of the mesh's geometry, and offsetting it by some user defined value along the surface's normal direction to create a hull around the original mesh. The surface normals for this new hull are then inverted, and back facing surfaces are culled from the rendered view. This entire method can be done in the Vertex and Geometry Shaders on the GPU, or by manually modeling the Hull, and is quite effective for simple geometry. However the Inverse Hull method often produces unwanted artifacts as the silhouette becomes thicker, or when non-manifold edges are present in the geometry. This method also offers us no control over which *type* of silhouette line we want to draw.

[62] Junya Christopher Motomura, "GuiltyGear Xrd's Art Style."

Figure 39: *Inverse Hull Silhouette Lines.*
(Left) A clean silhouette (Center) Small cracks appear in thicker linework due to non-manifold geometry (Right) Thicker lines produce unwanted artifacts

Our main takeaway from the Inverse hull method, that will be valuable in developing our Variable Silhouette Lines, is that to draw the silhouette of an object, it's not necessary to select only the edges that represent the silhouette in that view, but rather we can draw the entire geometry of the mesh, and manipulate it to give the impression of a silhouette. However since we do want some control over which type of silhouette line we are drawing, let's take a closer look at each of the Freestyle line types we are trying to emulate.

Types of Silhouette Lines In general in the field of line drawing research, a silhouette line is defined as a line marking the edge between a visible, and an invisible face. The cusp of the visible and hidden. Each of the four Freestyle line types we are trying to emulate is some slight variation on this theme, the cusp of the visible and invisible.

Silhouette Lines Are the most basic, they draw every cusp from visible to invisible.

Border Lines Are the next simplest, rather than marking the transition from a visible face to an invisible face, they mark the transition from a visible face to emptiness. In other words, Border Lines mark edges with only one adjacent face. These are a special case of Non-Manifold edges.

Contour Lines Are the outside Silhouette lines of an object. Its outline. Each cusp from visible to invisible, that isn't *in front* of any other part of the object.

External Contour Lines Are only Silhouette lines that have absolutely nothing *behind* them.

Describing the 4 types in this way we can see that all four share the basic quality of silhouette lines, being a cusp between a visible and invisible face,

but each type has some additional conditions regarding the surfaces behind them.

Our takeaway here then is that the key distinction between these line types has to do with the *distance* from the particular visibility cusp to the next surface behind it. Might we then be able to make use of our depth buffer then to differentiate between these line types?

Our Derived Method

MeasureIt-ARCH's Variable Silhouettes work entirely based on these two key takeaways. We draw every edge of the mesh as a thick line wireframe, and then, in the geometry shader we push this wireframe away from the observer along the Camera Space Z-axis. This causes any non silhouette lines to be occluded by the mesh geometry, and these occluded pixels are discarded in the depth buffer test. The amount that the wireframe is moved back along the Camera Space Z-axis determines what type of silhouette is drawn, and we call this amount the Z-offset, and expose it as a property of the Line Group that the user can adjust. Smaller Z-offsets produce linework analogous to Freestyle's Silhouette and Border line types, larger Z-offsets produce linework analogous to Freestyle's Contour line type, and very large Z-offsets produce results similar to the External Contour line type.

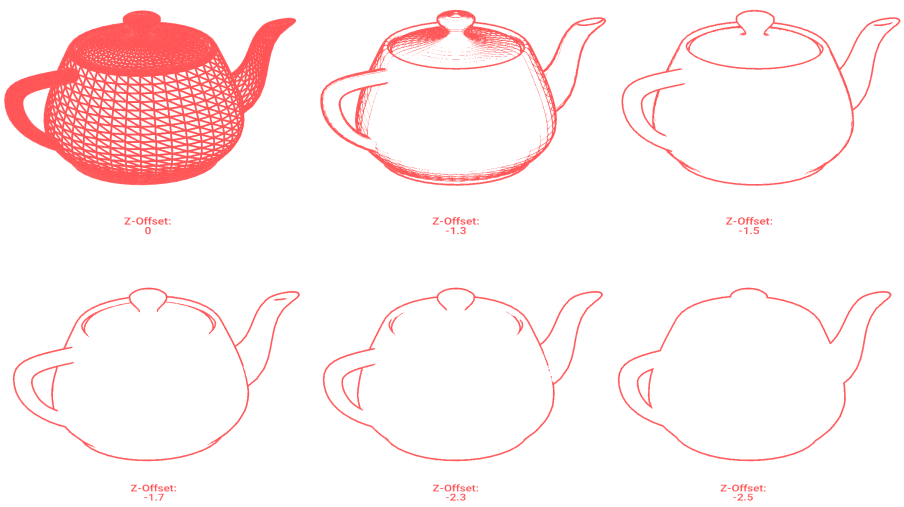


Figure 40: *Variable Silhouette Examples*
A progression showing how the display of MeasureIt-ARCH's Variable Silhouette lines change as the Z-Offset value is adjusted

The implementation isn't quite this simple however. Simply moving the wireframe back along the Camera Space Z-Axis works well for orthographic views, where there is no perspective distortion. However in a perspective view this shift along the Z-axis results in the silhouette being distorted. In the figure below we can see how this might look if our setup was modeled explicitly in the 3D scene.

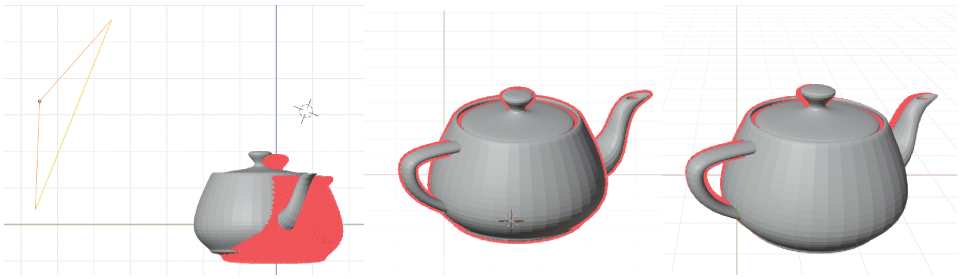


Figure 41: *Perspective Challenges with a Camera Space Z-offset*
How a Z-offset silhouette might appear if created in the 3D scene. (Left) Scene setup (Center) Orthographic camera result (Right) Perspective camera result. Note how this does not result in a proper silhouette

However we can fix this perspective distortion in our shader code. To understand how this works we need to introduce one more co-ordinate system, Clip Space. Clip Space is a homogeneous four dimensional co-ordinate system that is used in OpenGL shaders. The fourth dimension (W) in clip space, allows for perspective view transforms to be carried out through matrix multiplications, which the GPU hardware is optimized for. For our purposes we can think of our X,Y,Z,W co-ordinates in Clip Space as the same as our X,Y,Z Camera Space co-ordinates, just with an additional value that, more or less, represents the amount of perspective distortion which that point will receive when it's mapped to the 2D image on the screen. In our shader, we get a points Clip Space coordinates by multiplying its Object Space position by the 'ModelVIewProjectionMatrix' which returns our Clip Space position. The key here is that our Clip Space coordinates have their perspective distortion baked into the W value. So if we apply our Z-offset only after we've converted our co-ordinates to Clip Space, then we can slide our points backwards along the Camera Space Z-axis, without introducing any new perspective distortion. The results of this are shown in the figure below.

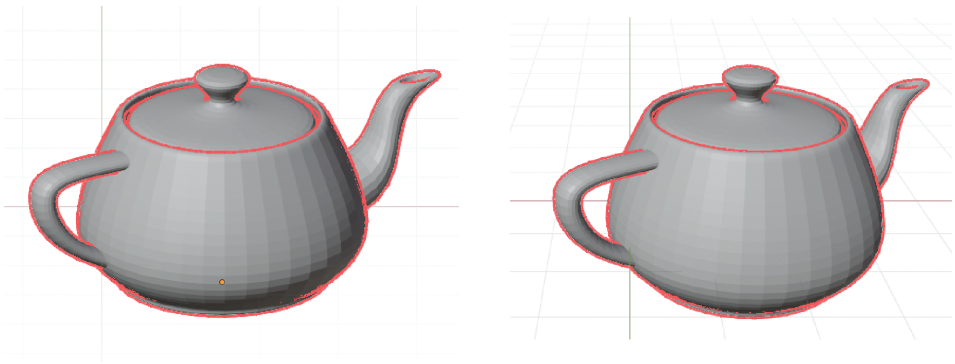


Figure 42: *Clip Space Z-offset in Shader without Perspective Distortion*
(Left) Orthographic camera result (Right) Perspective camera result

Using this Clip Space Z-offset to create Variable Silhouette lines allows MeasureIt-ARCH to emulate Freestyle's most common silhouette line types in real time, with a high degree of user control. By combining multiple Line Groups with different Z-offsets and line weights MeasureIt-ARCH can communicate complex organic shapes through simple line drawings with a visual fidelity similar to that of an un-styled Freestyle render.

Figure 43: *MeasureIt-ARCH Linework Compared with Freestyle Linework on Organic Geometry.*
(From Left to Right) 1. Shaded model and wireframe. 2. MeasureIt-ARCH linework, drawn with two Variable Silhouette Line Groups. 3. Freestyle linework drawn with Silhouette (thin) and External Contour (Thick) lines. 4. MeasureIt-ARCH and Freestyle results overlaid, MeasureIt-ARCH in red, Freestyle in green, overlap in yellow

Z Offset's Weakness; Ground Clipping

The major downside of this method for drawing Variable Silhouette lines is what we'll call ground clipping. Since our Silhouette lines are depth tested against the entire 3D scene, some undesirable clipping of thick

silhouette lines can occur where objects meet. This can be seen in the figure below. This can currently be worked around by rendering multiple passes of the scene (foreground and background) using Blender's view layers and compositor. However in the future this could be solved in shader by using custom depth buffers for silhouette lines

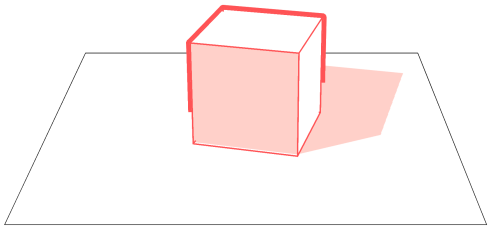


Figure 44: *Silhouette Ground Clipping*
Silhouette lines can be clipped in an undesirable manner when close to another object or ground plane

Z - Buffering, Benefits and Limitations

These Z-Buffering based methods for line visibility, Hidden Line rendering, and Variable Silhouette rendering are ideal for our goal of real time viewport performance, as they keep draw times quite low. The raster images produced by these methods also integrate well into Blender's existing render pipeline allowing them to be overlaid with the output from any of Blender's viewport render engines. A Z-Buffering approach is also functional for rendering out final drawings for print or transmission, as long as the image size and resolution are set to suitable values for the scale of the print. Where these raster techniques do fall short is interoperability. There is currently no option to export MeasureIt-ARCH linework to a vector format for editing in other software packages. Although the prime goal of this thesis is to create an architectural drawing toolset that works *within Blender* first, interoperability is certainly an area that will need to be improved in future development.

3. What Visual Characteristics do we Give These Lines?

Our third and final problem for line drawing is also the simplest to solve. MeasureIt-ARCH provides a few basic properties for Line Groups that the user can edit to change the look of all line segments in that Line Group. Each of these properties is sent to the Line Group Shader at draw time.

These properties are;

Lineweight

The lineweight determines the thickness of the lines drawn. Thickening each line segment is achieved in the Line Group's Geometry Shader. The Geometry Shader takes the start and end point of the line, and calculates its direction and normal in Screen Space. It uses the lines direction and normal to draw a tessellated rectangle of the user defined thickness over the initial line.

Since MeasureIt-ARCH has no line chaining, in order to have a smooth transition between line segments we also draw a circular endcap at the end of each line segment. Without some kind of end geometry, gaps would be visible between adjacent thick lines.

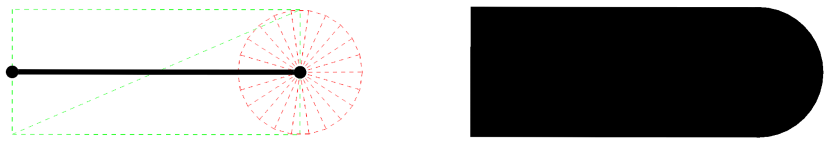


Figure 45: *Thick Line Tessellation*
(Left) Shapes created in the geometry shader. Black dots show the line's start and end points, tessellated rectangle in green, point pass in red. (Right) Resulting thick line

Color

Cotrols the color of the Line Group, applied in the fragment shader.

Z-Offset

As covered in the previous section, the Z-Offset value allows the user to manipulate the Line Groups depth without introducing any perspective deformation, this is used to create Variable Silhouette lines

Extension

Allows for some over extension of MeasureIt-ARCH lines, as is often common in hand drawn architectural drawings. This overextension is calculated in the Geometry Shader, in Object Space.

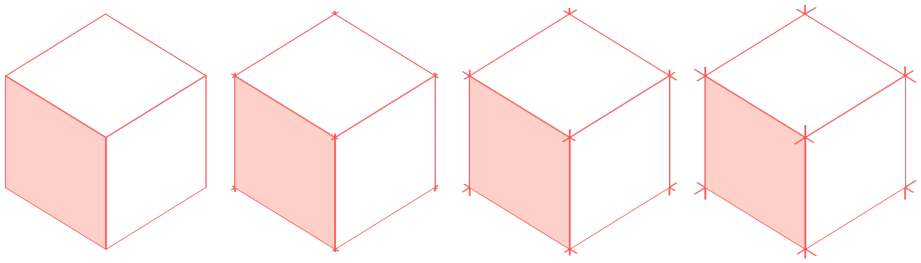


Figure 46: *MeasureIt-ARCH's Line Overextension.*
From left to right, extension values shown are 0.0, 0.05, 0.1, 0.15

Hidden Line Color & Hidden Line Weight

If Hidden line drawing is enabled for this Line Group, these settings allow Hidden Lines to be given their own independent color and line weight

Dash Scale & Spacing

Dashed lines for hidden line drawing are as close as MeasureIt-ARCH gets to a textured stroke. When the Geometry Shader generates the tessellated rectangle that is rendered it also calculates an 'arc length' parameter for each of the points in this rectangle. we can think of the arc length as a linear gradient along the length of the rectangle, whose maximum value is determined by the line's length.

In the fragment shader, for each pixel of the line rendered we evaluate a sine function for the arc length value at that pixel. This gives us a periodic sine wave along the length of the line which we can use to create our dashes. If the value of the sine wave at any point along the line is less than some cut

off threshold, we discard that piece of the line, if its above that threshold, we draw the line as normal. This results in a dashed line, with the space between the dashes determined by the threshold value. The Dash Scale and Dash Spacing properties describe the sine waves frequency and cut off threshold respectively, allowing the user to control the scale and spacing of the dashed lines.

These eight properties allow the user to control the look of each Line Group to some degree. All of these properties can also be controlled through MeasureIt-ARCH's style system, allowing multiple Line Groups to share the same property set.

With that we bring our discussion of MeasureIt-ARCH line drawing system to a close, next, on to Dimensions.

Dimensions; Dissolving the Work Plane with MeasureIt-ARCH

Creating a new Open Source dimensioning tool provides an opportunity to rethink the paradigms and systems that define how dimensioning tools are implemented in industry-standard software packages. Contemporary design tools such as Autodesk's Revit, and McNeil's Rhino provide three-dimensional design environments. However, the dimensioning tools of these software packages are still bound to a two-dimensional "work plane" or "C-plane" (as it's called in Rhino). These work plane based dimensioning systems are legacies of 2D CAD software that have persisted into their three-dimensional successors.

Why Use a Work Plane and When Does it Fail

The 2D work plane approach to dimensioning has two significant advantages;

First, the computations required to accurately locate and display a dimension element on a 2D plane are significantly more straightforward than those required in 3D. Dimensions projected on a 2D work plane are the most efficient solution when the desired end product is a 2D plan, section, or elevation drawing.

Second, the process for the user to place a dimension is very straightforward in a work plane based system. First, the user defines the work plane, often by selecting a surface already modelled in that plane. Then they select the two anchor points of the dimension and drag in the desired direction to determine the dimensions placement. This 'Select Point, Select Point, Drag to Place' workflow for adding dimensions has been nearly ubiquitous since the time of 2D CAD applications, and the definition of a 2D work plane facilitates the use of this same user experience in a 3D context.

However, the 2D work-plane approach does begin to slow down the user's ability to add dimensions to 3D isometric or axonometric views (or any 3D view for that matter). The work-plane needs to be manually redefined by the user to add dimensions to each unique plane, meaning any complex geometric form will require a plethora of unique work-planes to be thoroughly dimensioned.

Things become especially challenging when modifying geometry after a dimension has been placed, as changes to a dimension's anchor points

may cause the dimension to move out of the work plane that was defined for it. How a piece of software responds to this varies, but often it will result in an error in the dimensioning tool. Revit makes some effort to automatically adjust work planes to adapt to changes in a model's parameters. However, these adjustments are not always successful, and often the model's connection to the dimension will break, resulting in an error message prompting the user to delete the dimension or undo the action that caused the error. In Rhino dimension behaviours are a bit more nuanced. As long as the record history functionality is enabled, Rhino's dimension elements will track with their connected geometry, even outside of their original work plane (or C-Plane as it is referred to in Rhino). However, the C-Plane still imposes a limit on the creation of new dimensions and must be redefined before dimensions can be added to a new plane.

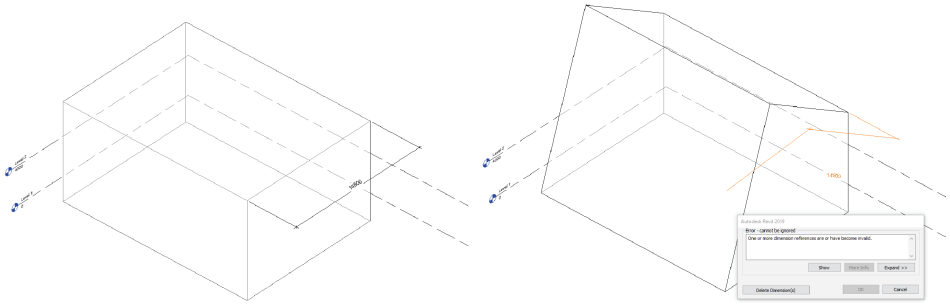


Figure 47: *Adjusting a Dimensioned Massing Object in Revit*

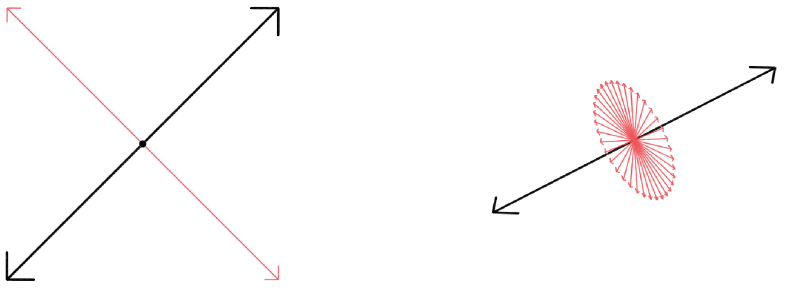
This insistence on defining a flat two dimensional plane in which dimensions sit, impacts the usability and efficiency of these dimensioning tools, for the sake of computational simplicity and a traditional user experience. Can we remove the work plane and its limitations, what new possibilities does this allow, how does it impact the user experience, and what new challenges does it present?

Perpendicularity in Two and Three Dimensions

Let's take a moment to examine how the computations required to place a dimension change when moving from two dimensions to three. Adding a dimension in 2D is relatively trivial. We need to draw some extension lines perpendicular to the line that we are measuring, and a dimension line with some text parallel to this measured line but offset by some amount. The bulk of the computational work here is involved in finding the correct

perpendicular direction. For a 2D line with a direction defined by the vector (x,y), its two perpendicular directions would be (y,-x), a 90-degree rotation counter-clockwise, or (-y,x), a 90-degree rotation clockwise. With only two options to choose from for the direction of the dimension, a piece of software can very simply rely on sampling the user's mouse movement to decide which of the two options to choose, and use the mouse position to determine how long to make the perpendicular extension lines.

Moving up to 3D things become more challenging, while in 2D space, there were only two possible perpendicular directions to choose from, in 3D space, there are an infinite set of perpendicular directions. Trying to select the user's intended option from this infinite set using only mouse input would be unreliable and frustrating for the user. This challenge of perpendicularity leaves us with two options, maintain the status quo and reduce the problem back to 2D space by having the user pre-define a 2D work-plane or utilize more information from the 3D environment to select a reasonable perpendicular direction with as little user intervention as possible.



2D Space
Two perpendicular directions from a point on a given line

3D Space
Infinite perpendicular directions from a point on a given line

Figure 48: *Directions Perpendicular to a Line in Two and Three Dimensions*

Using Mesh Geometry and the Users Viewpoint to Determine a Dimension's Placement.

To select a sensible perpendicular direction without any user intervention, we first need to develop a series of rules defining how dimensions should ideally be placed. Though MeasureIt-ARCH implements simple case by case algorithms (rather than trained learning algorithms) for its dimension

placement system, some of the decision making and rule defining processes for the dimension placement were inspired by the work done by Ian Vollick, Daniel Vogel, Maneesh Agrawala, and Aaron Hertzmann[63] in their work on automatic placement systems for annotating Images. The set of rules that inform MeasureIt-ARCH's dimension placement algorithm are:

1. Dimensions should always be placed perpendicular to the line of measurement
2. Dimensions should never be placed inside of an object
3. Aligned Dimensions should try to remain in-plane with one of their adjacent Faces
4. Dimensions should be as close to parallel as possible with the user's viewpoint to ensure readability.
5. Single Axis Dimensions should always remain in plane with a basis plane (XY plane, YZ plane, XZ plane)

While it is relatively simple to articulate these rules in language, to write them in code, we need to formalize them mathematically. Implementing these rules requires information about the geometry surrounding the dimension and the user's viewing direction. Using this information and a bit of linear algebra, it is possible to produce an algorithm that reliably places dimensions without manual user input. The resulting algorithm, as it is currently implemented in MeasureIt-ARCH can place Dimension elements in a useful and predictable manner and automatically adjust their location as the geometry is modified, or as the user's viewpoint changes.

But how is the Information about the user's viewpoint and object geometry obtained, processed and used in the placement algorithms? The bulk of this calculation is conducted in two processes;

View Segmentation

Rather than constantly adjust the dimension for every movement of the user's viewpoint, the 3D space is divided into six segments (Top, Bottom, East, West, North, South) each with a defined view vector ((0,1,0) for Front, (1,0,0) for East, (0,0,1) for Top etc.). We determine the segment of space our user's viewpoint is currently in by comparing its position vector with a series of thresholds. This lets the placement of the dimension be influenced by the segment of the 3D space the user viewpoint is currently occupying, rather than the viewpoints exact position. Meaning that only when the user's viewpoint crosses a predefined threshold from one segment to another does the dimension re-orient itself. In the case of Axis Dimensions, this also allows

[63] Ian Vollick et al., "Specifying Label Layout Styles by Example," n.d., 10

us to finetune unique thresholds that feel more natural for each of the three axes.

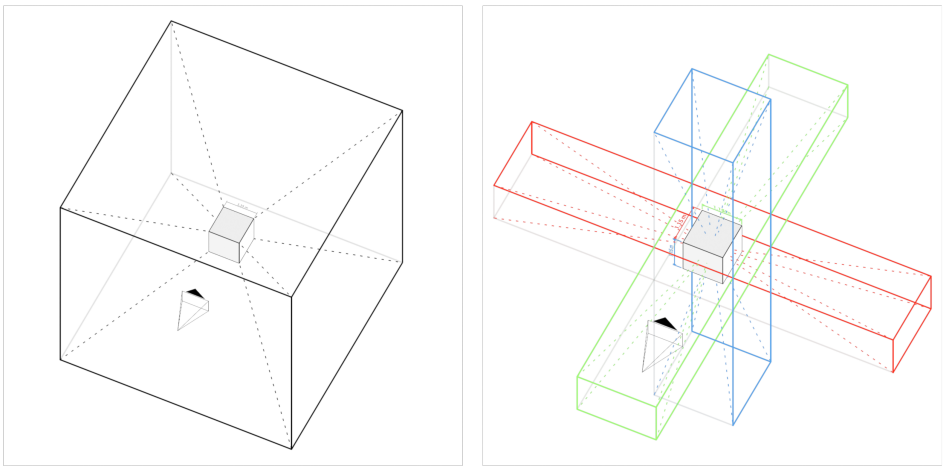


Figure 49: *Visualization of View Segmentation Thresholds in 3D Space (Left) Basic segmentation (Right) Per-axis segmentation*

Using the Geometry's Face Normals.

To ensure that the dimension points away from the object in a sensible direction, MeasureIt-ARCH considers the geometry of the object like so;

1. The algorithm finds two faces adjacent to the line being measured.
2. The normals of the adjacent faces are stored in a list. [k]
3. The sum of the normals is calculated and converted to a unit vector.
4. This new Unit vector is returned as our outward direction.

[k] A faces normal is the direction perpendicular to the plane defined by the face, in Blender and other 3D applications that use mesh modelling the normal is calculated to point outward from the meshes surface for manifold objects

Implementation in Aligned Dimensions For Aligned Dimensions, this consideration of the face normals is most of the placement algorithm. Once a normal is selected, it can either be used as the direction for the dimension right away or be projected down onto the plane of either of the adjacent faces or onto the basis plane, who's normal is most similar to the current viewpoint segments View Vector. Depending on the user preference as indicated in the dimension's properties.

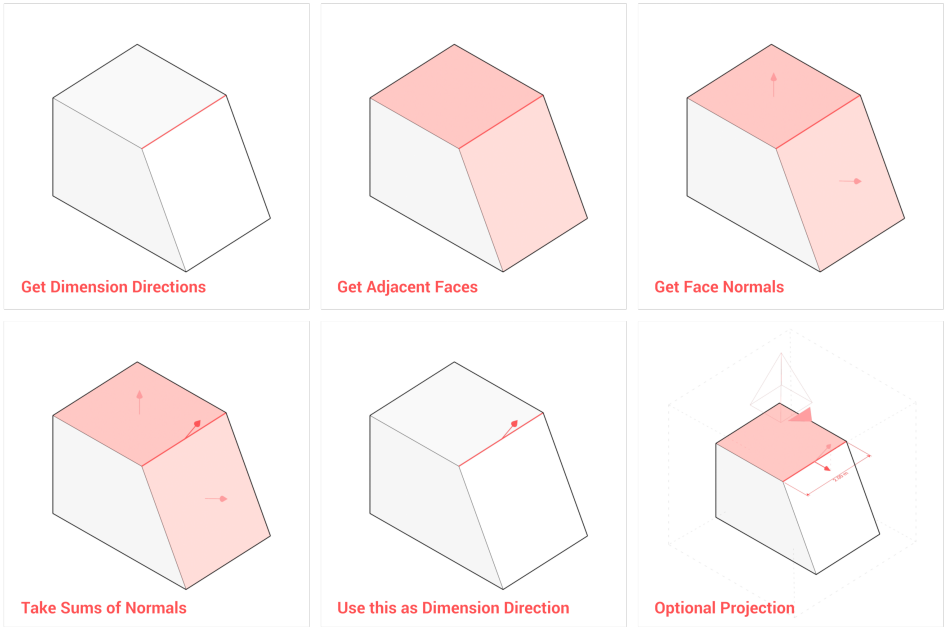


Figure 50: *How Adjacent Face Normals are Used to Evaluate Dimension Placement*

Implementation in Single Axis-Dimensions Single Axis Dimensions are slightly more complex and make greater use of view segmentation. Here's how the Single Axis placement algorithm works;

1. Determine which segment thresholds to use based on the axis being measured.
2. Determine what segment of 3D space the user's viewpoint is currently in based on the segment thresholds.
3. Get that segment's corresponding view Vector ((0,1,0) for Front, (1,0,0) for East, (0,0,1) for Top etc.)
4. Calculate the distance between the two points being measured on the specified axis (x, y, or z) This is our Distance Vector.
5. Calculate the cross product of our Distance Vector and our View Vector (In 3D the cross product of two vectors is always a third vector, perpendicular to the plane defined by the first two)
6. The resulting vector obtained by the Cross Product operation is our dimension placement direction.

Since the dimensions are ultimately being projected onto a 2D plane in both of these cases, one could argue that the end result is not significantly

different from a work plane based system, however the key distinction here is that MeasureIt-ARCH's system does not require the user to manually define these basis planes or projections before the dimension is created, the dimension placement system works out which plane is most suitable automatically based on the users viewpoint.

These steps presented represent only the basic principles of these algorithms. In MeasureIt-ARCH's implementation, both axis and aligned dimension placement algorithms also provide additional features such as allowing the user to lock dimensions to a desired basis plane, and the ability to manually tweak the direction of the dimension by rotating it using the line being measured as the rotation axis. Beyond these additional features, the algorithms also need to be able to deal with edge cases to ensure they behave as expected even when some of the required information is not available. The consideration of adjacent faces, in particular, needs to be able to handle cases with non-manifold geometry (more than two adjacent faces to a single edge), as well as objects without any geometry such as Blender's empties. In these cases, the general fallback is to place the dimension in a plane that is normal (perpendicular) to the View Vector of the view segment that the users viewpoint is currently in.

The User Experience; Adding Multiple Dimensions Simultaneously.

A significant result of this automated dimension placement system is its change to the user experience of adding dimensions. The traditional, 'Select anchor point, select anchor point, drag to place' process of placing dimensions is not possible with this system. Instead to add a dimension with MeasureIt-ARCH the user selects the two anchoring vertices first, and then runs the 'Add Dimension' operator, which creates a new dimension instance that utilizes those two selected vertices as it's anchor points, and the draw system determines this new dimensions placement automatically. Once the dimension is placed, the user can tweak the dimensions distance from the line being measured either by dragging the dimension in the 3D viewport using MeasureIt-ARCH's gizmos, or by adjusting its distance property in the User Interface. For adding a single dimension this process isn't too dissimilar from the traditional mode of adding dimensions in industry standard packages. The strength of this method however, is that it is capable of adding multiple dimensions with a single click of the 'Add Dimension' operator.

If the user selects more than two mesh vertices, then when the 'Add Dimension' operator is called it will add dimensions between each of the selected vertices in the order that they were selected by the user. This allows the user to quickly define sequences of dimensions, without needing to place each dimension individually, or redefine a work plane when changing plane. As an example of MeasureIt-ARCH's current workflow, calculating the position of, and adding three aligned dimension elements, (all on what would be considered independent work planes in other software), with the aligned dimension tool in MeasureIt-ARCH takes only a fraction of a second. When compared to the user input time required to specify three unique work planes and draw in each of their dimensions individually in standard software packages, this seems like an improvement.

Automated Dimension Placement; Benefits and Limitations.

The added complexity in MeasureIt-ARCH's dimension placement algorithm does make it more computationally intensive than it's work plane based alternatives. Querying an object's faces, running vector calculations, determining viewport segments, as well the conditional checks for edge cases, each of these operations takes time, and because dimension placement is calculated in the Draw Method, these operations are running every single time the scene updates, for every dimension in the scene.

That being said, modern computing power can handle this added complexity quite well, and although MeasureIt-ARCH's performance when it comes to dimension drawing is still slower than industry standard packages, there is significant room in MeasureIt-ARCH's code to optimize and improve the basic principles described for the dimension placement system, and the ability to add multiple dimensions simultaneously, through a single use of the dimension tool, also helps to offset the slower viewport performance by providing a potentially more efficient user experience.

Currently, however, MeasureIt-ARCH does show some noticeable reductions in viewport framerate in scenes where 500+ aligned dimensions are being drawn. MeasureIt-ARCH takes approximately 218ms per viewport update (roughly 4.5 frames per second(fps)) to draw 512 dimensions, compared to 6.9ms (140 fps) per viewport update to draw the same number of dimensions in Rhino. Although further timing has revealed that, in general, the common performance bottleneck in MeasureIt-ARCH is not solely due to the dimension placement system, but is also a symptom of the number of unique draw calls being sent to the GPU shaders from the dimension's draw

method.

The speed of the dimension placement system is also proportional to the amount of mesh edges in the object being measured. The system takes roughly 1ms to compute for a single dimension measuring an object with 768 edges, and this timing scales proportionally as the mesh complexity increases. The process of batching our dimensions co-ordinates and sending them to the various shaders, on the other hand, consistently takes between 0.4ms and 1.2ms to compute. Which of these processes ends up being the main performance bottleneck for a scene depends on the average complexity of the objects being dimensioned.

This performance still isn't ideal; however, given that MeasureIt-ARCH is currently only suited for small projects, it is likely that an upper limit of 500+ dimension elements per scene is enough for the small scale workloads we expect the tool to be used for. However, for MeasureIt-ARCH to ever be considered for use in large-scale projects in the future, some work will need to be done to improve the bottlenecks in the dimension placement, and dimension drawing systems.

Sharing MeasureIt-ARCH

> "*Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging. The power of this effect is easy to underestimate.*"[64]

Creating MeasureIt-ARCH as an Open Source add-on for an Open Source software, called for some direct and intentional sharing of the tool throughout its development process. Rather than solely relying on traditional academic modes of information collection and dissemination, this thesis took inspiration from the Cathedral and the Bazaar by Eric S. Raymond,[65] as well as the sharing methodologies of the Blender Foundation and Blender community, since for any Open Source project to survive and grow, it is essential for it to engage with a community. [l][66]

MeasureIt-ARCH takes advantage of new media platforms like YouTube, GitHub, and the 'Blender.Today' forums, to make the work publicly available and create opportunities for user feedback. Although this feedback never dramatically changed the primary design goals of MeasureIt-ARCH outlined in our specification, it was invaluable for troubleshooting bugs, and for giving focus to small, simple changes that helped to improve MeasureIt-ARCH's user experience.

The Cathedral and the Bazaar

Raymond's "The Cathedral and the Bazaar" outlines two philosophies for the development of software. Cathedrals are produced in isolation. Guarded and protected until a final release is ready. This is similar to the conventional mode of software development, closed and hidden with infrequent releases. Bazaars, on the other hand, are a cacophony of chaotic voices and opinions where individual contributions come together to create a new whole. The Bazaar style is reminiscent of the extreme ideals of Open Source development as a pseudo-anarchic, egalitarian community [m]. However, as

[l] This is not to say that effort was not put into traditional modes of academic research sharing, to date versions of this work have been accepted for presentation and publication at eCAADe + SIGraDi 2019, and the Blender Conference

[m] Of course, whether Open Source development, in reality, is the ideally egalitarian process many idealize it as, is up for serious debate. Open Source communities, like most communities today, are struggling to overcome a series of systemic attitudes and unconscious biases that pose problematic barriers to genuinely fair and equitable participation

[64] Eric S. Raymond, *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, 1st ed (Beijing ; Cambridge, Mass: O'Reilly, 1999).

[65] ibid.

[66] Kevan and Beesley Cress, "Architectural Design in Open-Source Software - Developing MeasureIt-ARCH, an Open Source Tool to Create Dimensioned and Annotated Architectural Drawings Within the Blender 3D Creation Suite." in *Sousa, JP, Xavier, JP and Castro Henriques, G (Eds.), Architecture in the Age of the 4th Industrial Revolution - Proceedings of the 37th eCAADe and 23rd SIGraDi Conference - Volume 1, University of Porto, Porto, Portugal, 11-13 September 2019, Pp. 621-630* (CUMINCAD, 2019), http://papers.cumincad.org/cgi-bin/works/paper/ecaadesigradi2019_561.

we've seen in our analysis of Blender's development structure, Open Source projects can benefit from a balance between the two approaches. Raymond too notes that;

> "*It's fairly clear that one cannot code from the ground up in bazaar style. One can test, debug and improve in a bazaar style, but it would be very hard to originate a project in bazaar mode. […] Your nascent developer community needs to have something runnable and testable to play with*" - Eric S. Raymond[67]

MeasureIt-ARCH, still in its early development, is much closer to a Cathedral style of development, with only two key contributors (first Antonio and now Myself). However, steps are being taken to 'Share Early & Share Often' as Raymond recommends. Although beyond the end of this thesis, efforts to foster greater engagement with the tool must be pursued if MeasureIt-ARCH is to continue to grow and improve.

Licensing

To facilitate sharing and ensure that MeasureIt-ARCH remains free and Open Source, it has been released under the GNU GPL v3 license. This is the same license as Blender's source code. GPL v3 is what is commonly referred to as a 'viral copyleft' license, as it employs modern-day copyright law to enforce terms that ensure that for any software released under this license its source code must be made available along with the compiled software itself. GNU GPL is referred to as a 'viral' license because It provides clauses stating that any derivative works created from the software's source code must also inherit the same license. This helps protect free and Open Source works released under the GPL license from being bought or absorbed by larger commercial applications and ensures that they remain Open Source.

Leveraging New Media

To date, YouTube has been the primary outreach vehicle for MeasureIt-ARCH. In the time since serious development was started on MeasureIt-ARCH, two YouTube videos documenting MeasureIt-ARCH's features and development have been published. The first on March 24th, and the second on June 27th of 2019. Alongside these documentation videos, one time-lapse, showing a rough demo of the MeasureIt-ARCH being used to produce a set of drawings for a simple residential massing model, was also published.

[67] Raymond, *The Cathedral & the Bazaar*

As of July 30th, 2019, these videos have been viewed a total of 1,168 times and garnered 34 comments. By YouTube's standards, this is a relatively limited scope of engagement, however as a student working on a piece of software as part of a master thesis, having this degree of exposure and feedback on the project throughout its development has been invaluable, and these cycles of release, testing, and bug fixing have helped keep MeasureIt-ARCH's development focused as and act of craft, reinforcing what Richard Sennett refers to as 'An Experimental Rhythm of Problem Solving and Problem Finding.'[68] The engagement that these videos have fostered both through their existence on YouTube and through their sharing on the 'Blender. Today' forums have led to meaningful improvements to MeasureIt-ARCH, suggested by the community, that would not have occurred otherwise.

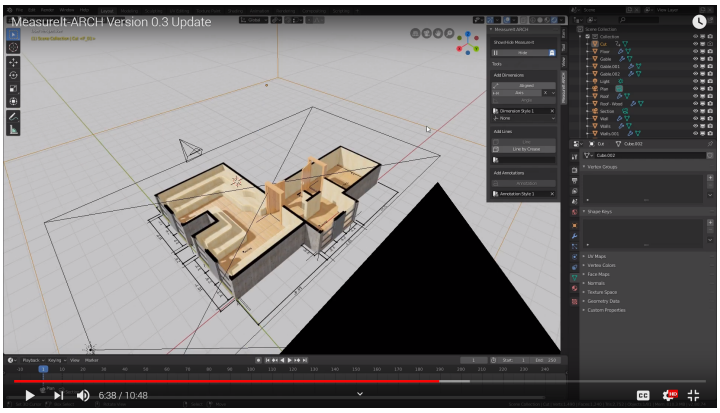


Figure 51: *A Still from the 'MeasureIt-ARCH V0.3 Update Video' Published on YouTube*

[68] Sennett, *The Craftsman*

Community Bug Fixing

> ""*Somebody finds the problem," he says, "and somebody else understands it. And I'll go on record as saying that finding it is the bigger challenge.*"" - Eric S. Raymond[69]

> "*The open relation between problem solving and problem finding, as in Linux work, builds and expands skills, but this can't be a one off event. Skill opens up in this way only because the rhythm of solving and opening up occurs again and again.*" - Richard Sennett[70]

While the majority of engagement with the project was generated through YouTube and 'Blender.Today', both of these sources directed users who experienced any issues with the software to report their problems via MeasureIt-ARCH's GitHub page.[71] This GitHub page has served as the central hub for bug reports and feedback on MeasureIt-ARCH. Utilizing GitHub has helped to keep the management of the project as open and transparent as possible, by allowing those interested in the project to access the latest development version at any time. It has also provided a central location for the management of submitted bug reports, issues, and feature requests. Testers of MeasureIt-ARCH's public releases who reported and documented the following bugs and were essential to their identification and eventual fixes.

Cross Operating System Compatibility

The first release of MeasureIt-ARCH would not run at all on the Apple operating system 'macOS'. Thanks to several issue reports by early users the cause of this incompatibility was quickly identified and addressed. The issue was due to macOS's lack of support for several of the OpenGL functions that the first release of MeasureIt-ARCH depended on. Namely the 'glLineWidth' utility was not available. When used on operating systems that do support it's functionality 'glLineWith' provides a simple method to draw lines thicker than one pixel. The need to accommodate macOS's lack of OpenGL support to ensure compatibility eventually led to the development of MeasureIt-ARCH's Geometry Shader which handles the drawing of line thickness. As a sole developer working on a Windows PC, this bug would not have been identified without the help of community testers.

---

[69] Raymond, *The Cathedral & the Bazaar*

[70] Sennett, *The Craftsman*

[71] Kevan Cress, "MeasureIt-ARCH Issues," n.d., https://github.com/kevancress/MeasureIt-ARCH/issues?utf8=%E2%9C%93&q=

MeasureIt-ARCH's Text Drawing System

One of the most significant bugs identified by public testing was a flaw in MeasureIt-ARCH's text drawing system. MeasureIt-ARCH draws text in 3D by rendering the text first to a 2D texture in an offscreen pixel buffer, and storing this texture data as array in the MeasureIt-ARCH element's properties. When it this element is drawn in 3D the texture data is read back to an OpenGL buffer, and mapped to a card placed in the 3D scene. In MeasureIt-ARCH's first release however, instead of storing the texture data as a property of the MeasureIt-ARCH element, it was written directly to an indexed OpenGL texture on the GPU. This implementation worked but would fail sporadically and unexpectedly, as some of MeasureIt-ARCH's indexed textures would occasionally be overwritten by elements of the Blender User Interface, which also makes use of indexed OpenGL textures to display components of the UI such as colour wheels or the top header bar. When these textures were overwritten users would find their dimension text replaced either by black boxes, or by oddly distorted chunks of Blender's UI. Since the bug was sporadic however, it was difficult to reliably reproduce on my own, but the thorough reporting and documentation of this bug by an early tester of MeasureIt-ARCH was essential to tracking down its cause, and developing its eventual solution.

Figure 52: *Image from an Error Report Submitted by a MeasureIt-ARCH User This image shows corrupted MeasureIt-ARCH text textures. Note one dimension's text has been overwritten with the UI texture that displays Blender's colour wheel UI element*

Modifier Instability

As we have discussed previously MeasureIt-ARCH elements generally reference an object's vertex indices to determine their location and how they are anchored to the geometry, rather than storing their absolute position in 3D space. While this allows dimensions and linework to adapt to changes in the object they reference, it also means that conducting an operation in Blender that re-orders the vertex indices of an object can cause MeasureIt-ARCH elements to break or shift unexpectedly. In past versions of MeasureIt-ARCH, this meant that some of Blender's procedural modifiers, like the Boolean modifier, would result in erratic behaviour from MeasureIt-ARCH elements. However, through community testing we were able to identify which of Blender's modifiers would recalculate an objects vertex indices and provide an option that users could enable to prevent the problematic modifiers from being evaluated when determining the position of a MeasureIt-ARCH elements anchor points. The effect of these modifiers can still be evaluated for the mesh object itself, but this no longer impacts the placement of MeasureIt-ARCH elements. Enabling this option makes working with Boolean modifiers and other generative modifiers much more stable.

While this has solved the issue of erratic behavior of MeasureIt-ARCH elements when using procedural systems like Blender's modifiers, there is still work to be done to develop a solution to maintain consistent and reliable dimension positions during more general non-procedural mesh editing operations that cause a re-ordering of an object's vertex indices.

In addition to the bug fixes, user feature suggestions have also led to the implementation of better unit formatting, and the unification of Blender and MeasureIt-ARCH's unit systems. Community engagement has resulted in the creation of a public project planning board on the GitHub platform to manage and prioritize community feature suggestions, as well as my own development plans for the MeasureIt-ARCH tool.[72]

[72] Kevan Cress, "MeasureIt-Arch Project Board," n.d., https://github.com/kevancress/MeasureIt-ARCH/projects/2.

SECTION 5:

Specification Evaluation

Requirements Specification Evaluation

While many of MeasureIt-ARCH's implemented features have been mentioned throughout our discussion of MeasureIt-ARCH's development, this section provides a complete summary of all of MeasureIt-ARCH's current features, and evaluates it's current state as compared to our specification outlined in section one. Each feature's implementation has been rated, based on how it compares with the original specification, with one of the following terms:

Fully Implemented: All features, as identified in the specification are currently implemented.

Substantially Implemented: The majority of features identified in the specification are currently implemented, with a few remaining improvements to be made.

Partially Implemented: A portion of features, or a proof of concept, is currently implemented in MeasureIt-ARCH.

Not Yet Implemented: No portion of this feature, as defined in the specification, has yet been Implemented.

In Active External Development: No portion of this feature, as defined in the specification, has yet been Implemented in MeasureIt-ARCH, but is under active development in another add-on, or by another member of the Blender community.

Images sequences illustrating the feature's current performance on simple geometry have been provided. To see the behaviour of MeasureIt-ARCH in video form, please see the two video releases that have been published alongside MeasureIt-ARCH's public test releases.,[73],[74][75] To experiment with MeasureIt-ARCH's features and limitations yourself, please see the installation guide and documentation presented in Appendix B.

[73] *MeasureIt-ARCH Introduction - YouTube*, 2019, https://www.youtube.com/watch?v=QL_ArANpsVU&t=90s.

[74] *MeasureIt-ARCH Rough Demo Timelapse - YouTube*, 2019, https://www.youtube.com/watch?v=lHI78SDB8bs.

[75] *MeasureIt-ARCH Version 0.3 Update - YouTube*, 2019, https://www.youtube.com/watch?v=MWo87QvcEPk&t=2s.

Feature Requirements

Line Drawing

Priority: Necessary

Status: Fully Implemented

Current Features:

- Users can specify edges of their 3D Geometry to be drawn as simple lines
- Lines can have their weight, colour, and style (solid or dashed) specified by the user
- Lines occluded by geometry can be visually distinct, differentiated by colour and dashes
- Lines can be specified to represent the silhouette of an object and can be made visually distinct with unique colour and line weight
- Lines created in the same operation are collected as a line-group, and share common properties (colour weight etc.)
- Lines can have a specified "overextension". Overextension extends a line beyond the mesh edge it references. This effect provides a simple imitation of hand drafted linework if that style is desired. (**Figure 58**)

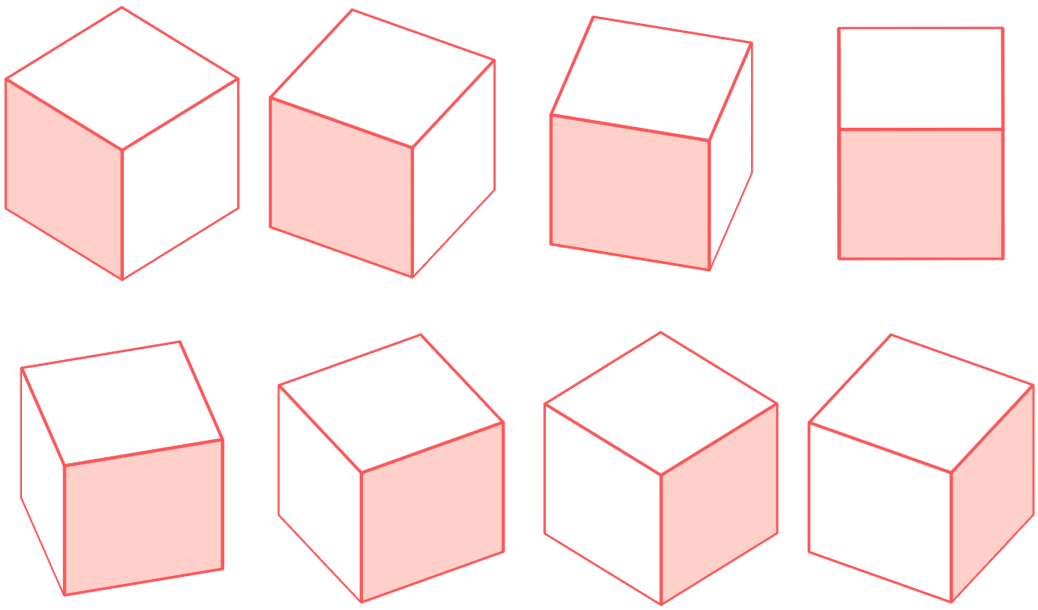


Figure 53: *Image Sequence of MeasureIt-ARCH Linework, on a Rotating Cube*

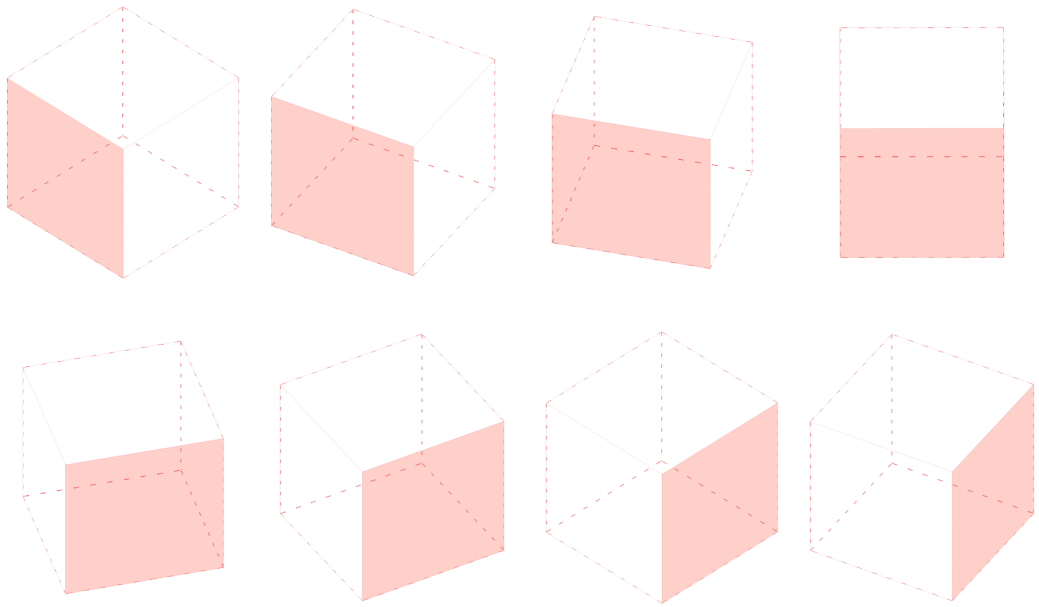


Figure 54: *Image Sequence of MeasureIt-ARCH Hidden Linework, on a Rotating Cube*

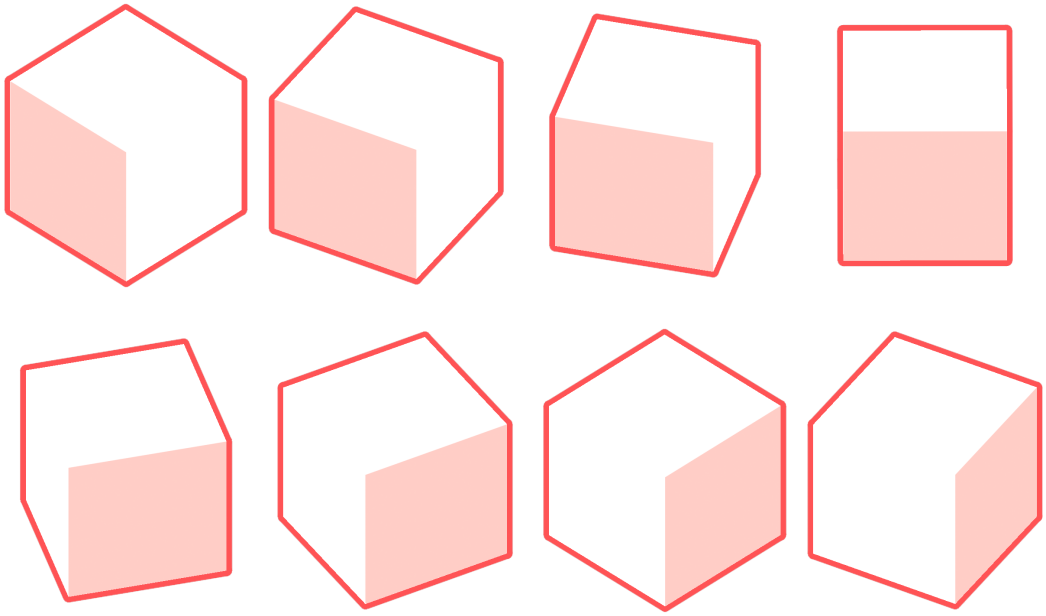


Figure 55: *Image Sequence of MeasureIt-ARCH Silhouette Linework, on a Rotating Cube*

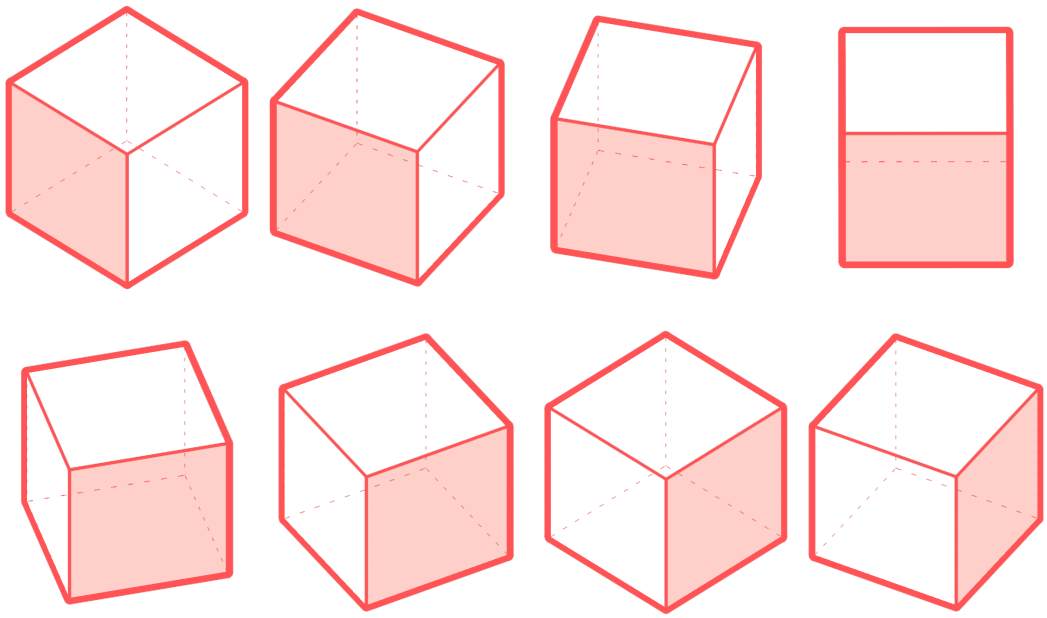


Figure 56: *Image Sequence Showing all Three MeasureIt-ARCH Line Behaviours on a Rotating Cube*

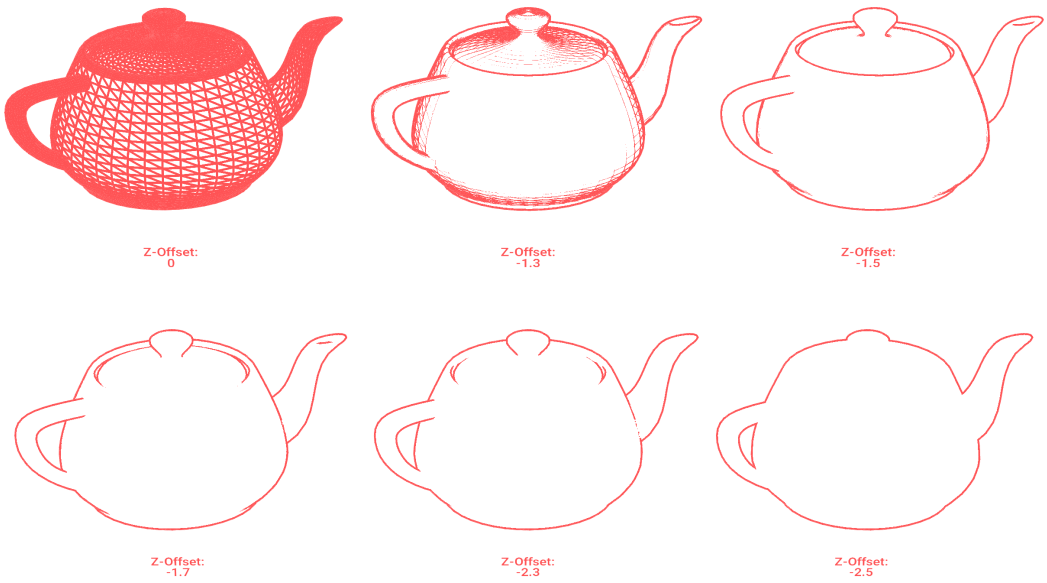


Figure 57: *MeasureIt-ARCH Silhouette Lines*
A progression showing how the display of Variable Silhouette lines change as the Z-Offset value is adjusted

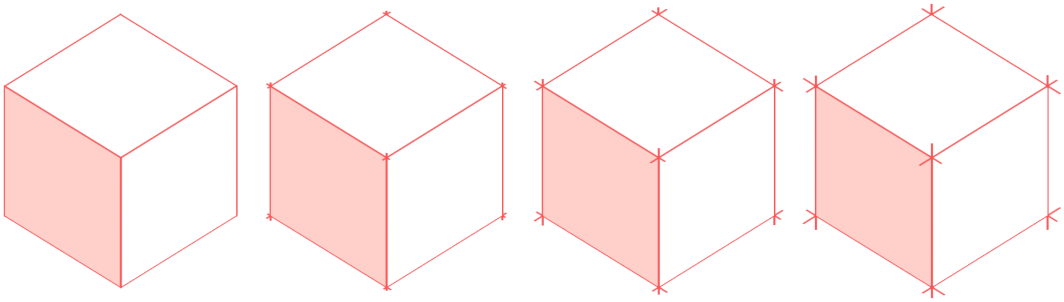


Figure 58: *MeasureIt-ARCH Line Overextension.*
From left to right, the extension values shown are 0.0, 0.05, 0.1, 0.15

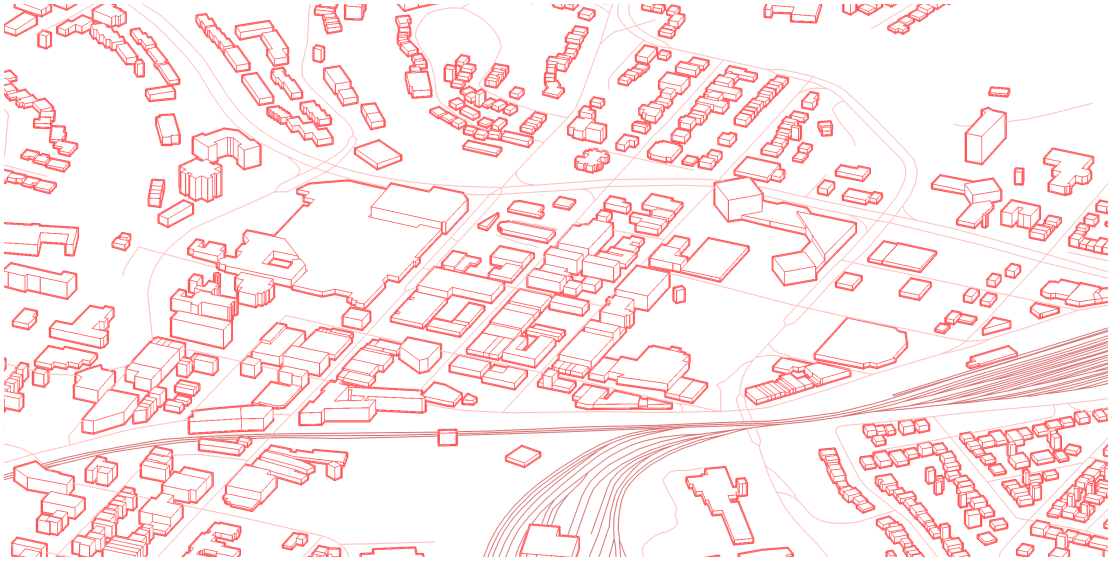


Figure 59: *MeasureIt-ARCH Linework and Silhouettes Applied to a GIS Model*
The city shown is Sudbury Ontario. Massing models provided by Open Street Map

Automated Line Group Creation

Priority: Necessary

Status: Partially Implemented

Current Features:

- An automated system is provided to automatically initialize MeasureIt-ARCH line groups based on creases in mesh geometry that are greater than a user specified threshold angle.
- This automated line group creation system can also identify and create linework on non-manifold edges.

Remaining Features:

- Line Groups can not yet be automatically generated based on:
  - Material boundaries

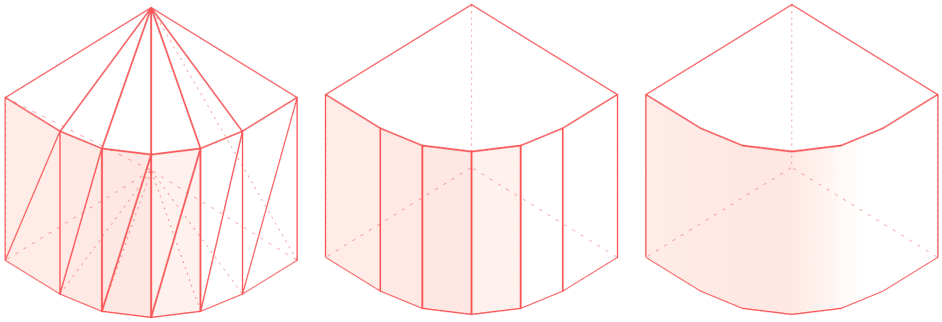


Figure 60: *Line Group by Crease Operator Behaviour.*
A cube with one rounded corner drawn with (left) MeasureIt-ARCH linework on all mesh edges. (center) MeasureIt-ARCH linework generated by the Line Group by Crease Operator using a crease threshold angle of 10 degrees. (right) MeasureIt-ARCH linework generated by Line Group by Crease Operator using a crease threshold angle of 30 degrees

Annotations

Priority: Necessary

Status: Substantially Implemented

Current Features:

- Annotations can display multiple lines of user-entered text
- Annotations are anchored to a user specified mesh vertex, object origin, or an empty object
- Annotation leader lines conform to ISO 129-1 Section 5.5
- Annotations have user-defined:
  - Font
  - Font Size
  - Color
  - Alignment (Left, Right, Top, Bottom)
  - Leader Line Weight
  - End Cap
- Annotations can display user defined metadata about the object they are anchored to.

Remaining:

- While Annotations can display user-defined Metadata, they cannot yet display the assigned material of their anchor object without manual entry.

Beyond the Spec:

Future development on MeasureIt-ARCH annotations could focus on:

- Support for the display of common architectural symbols

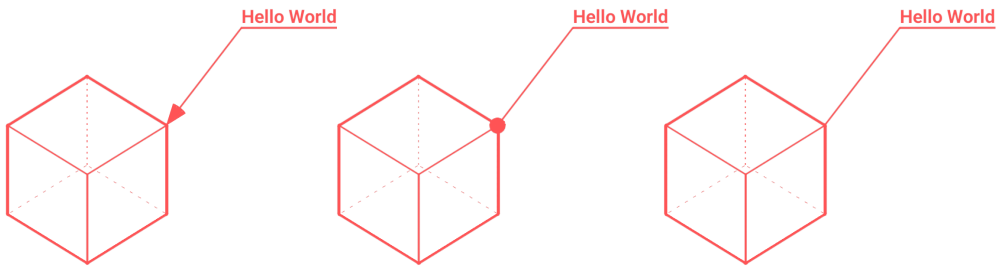


Figure 61: *MeasureIt-ARCH Annotations.*
(left) arrow endcap, (center) dot endcap, (right) no endcap

Aligned Dimensions

Priority: Necessary

Status: Fully Implemented

Current Features:

- Aligned Dimensions measure the distance between two points in 3D space. These two points can be;
  - Object Origins
  - Vertices
  - Empty Objects
  - Light Objects
  - Camera Objects
- Aligned Dimension extension lines are always placed perpendicular to the line they measure (ISO-129-1 Section 5.4)
- Aligned Dimensions have user-defined:
  - Font
  - Font Size
  - Color
  - Rotation (taking the line being measured as the axis of rotation)
  - Leader Line Weight
  - Terminations (According to ISO 129-1 Section 5.3.2)

Figure 62: *MeasureIt-ARCH Dimension Terminations. From Left to Right: None, Arrow, Filled Arrow, Dashed*

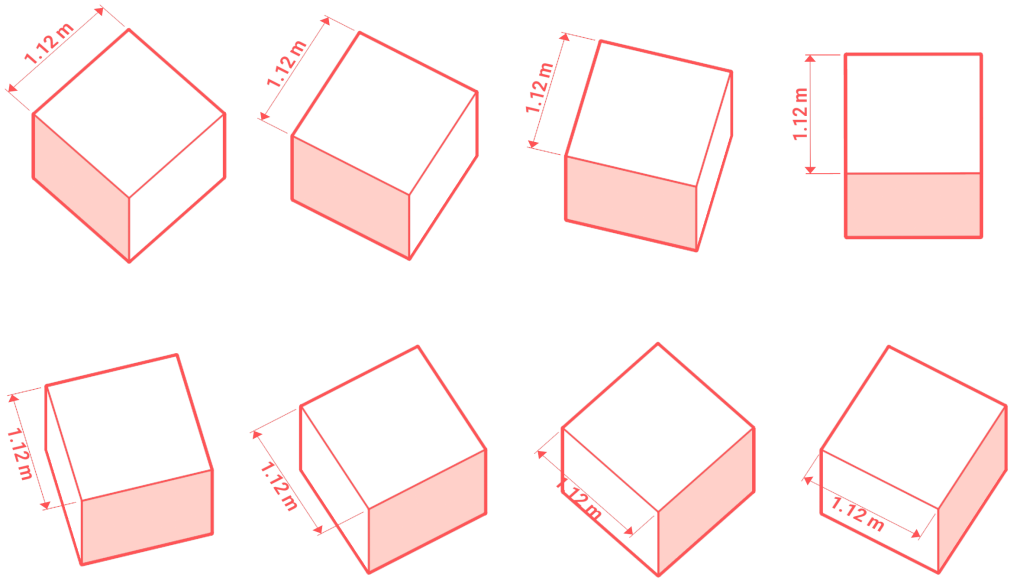


Figure 63: *Image Sequence Showing a MeasureIt-ARCH Aligned Dimension's Behaviour when Attached to a Rotating Cube*

Single Axis Dimensions

Priority: Necessary

Status: Substantially Implemented

Current Features:

- Single Axis Dimensions measure the distance between two points in 3D space, but only along a specified Axis. These two points can be;
  - Object Origins
  - Vertices
  - Empty Objects
  - Light Objects
  - Camera Objects
- Single Axis Dimensions are always placed perpendicular to the axis along which they measure
- Single Axis Dimensions have user-defined:
  - Font
  - Font Size
  - Color
  - Line Weight (for the leader lines)
  - End Cap (arrows or dashes at the ends of the dimension line)
- The axis of measure for a Single Axis Dimension can be:
  - Any cardinal Axis (X, Y, Z)

Remaining Features: + Currently, Axis Dimensions only support the measurement along a cardinal Axis (X, Y, or Z). Future development could focus on adding support to allow for measurements to be taken along any arbitrary user-defined axis.

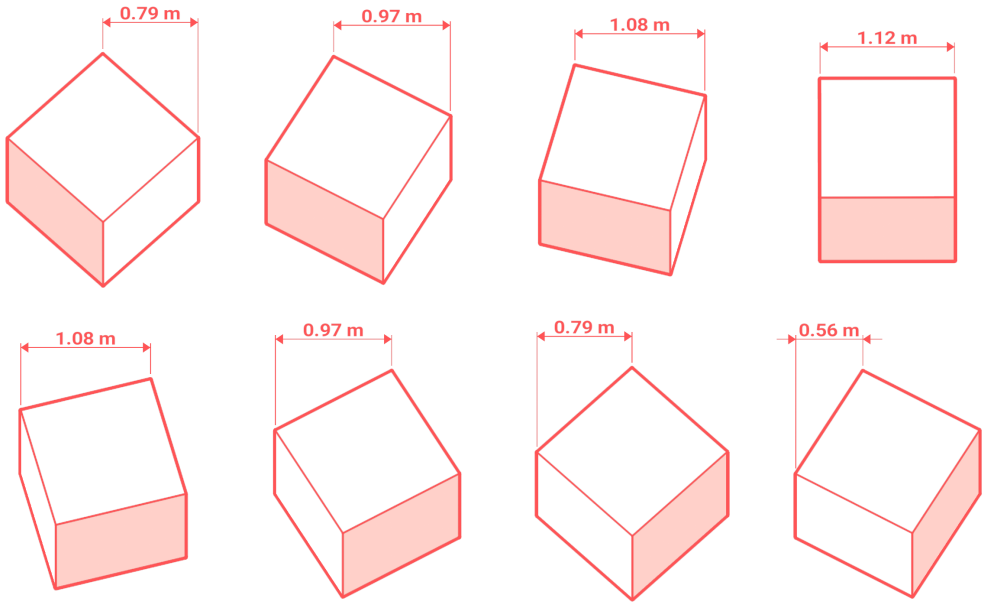


Figure 64: *Image Sequence Showing a MeasureIt-ARCH Axis Dimension's Behaviour when Attached to a Rotating Cube*

Angle Dimensions

Priority: High

Status: Substantially Implemented

Current Features:

- Angel Dimensions measure the angle between two lines, defined by three points.
- Angle Dimensions are aligned in-plane with the 3 points that define them.
- Angle Dimensions can have user-defined:
  - Font
  - Line Weight
  - Color
  - Font Size
  - Radius
- Angle Dimensions can display their measurement in degrees or radians depending on Blender's unit settings.
- Angle Dimensions can display either the non-reflex (between 0 and 180 degrees 65) or reflex (greater than 180 degrees 66) angles defined by the 3 points.

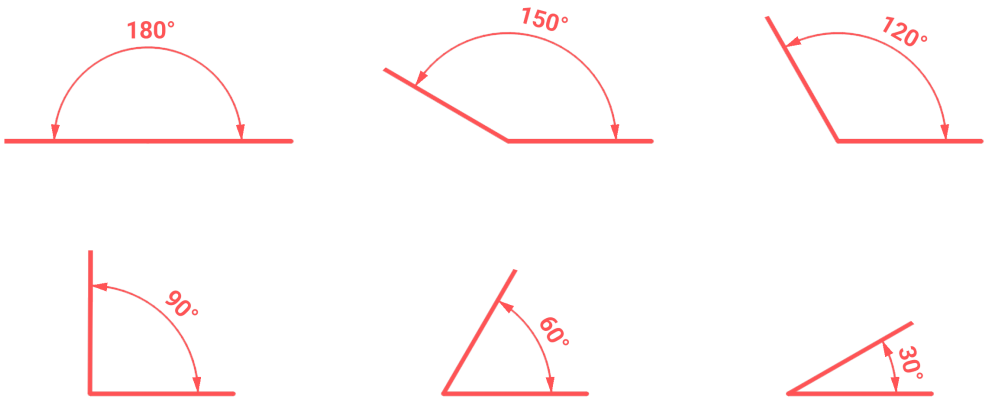


Figure 65: *Angle Dimensions at 30 Degree Increments*

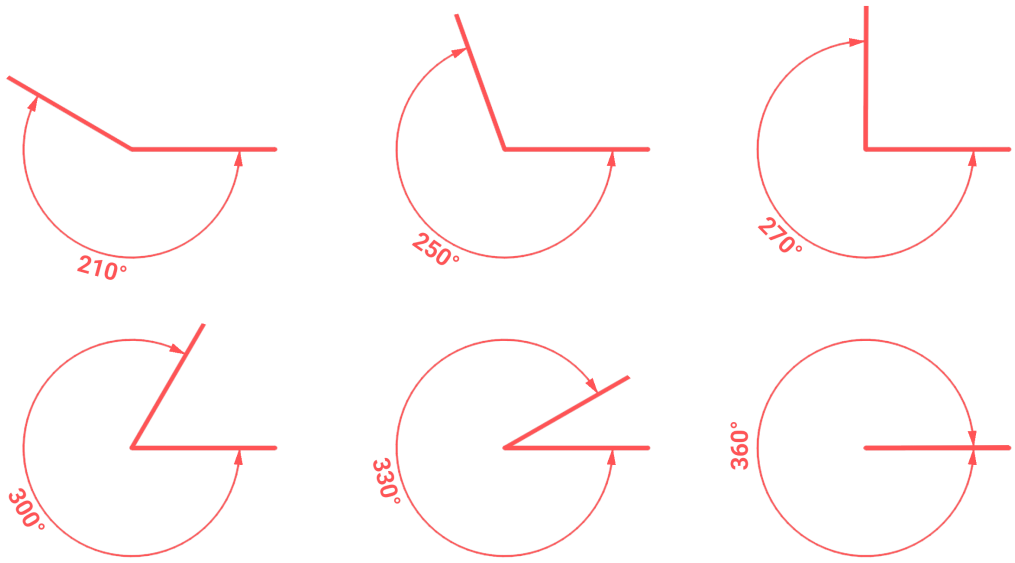


Figure 66: *Reflex Angle Dimensions at 30 Degree Increments*

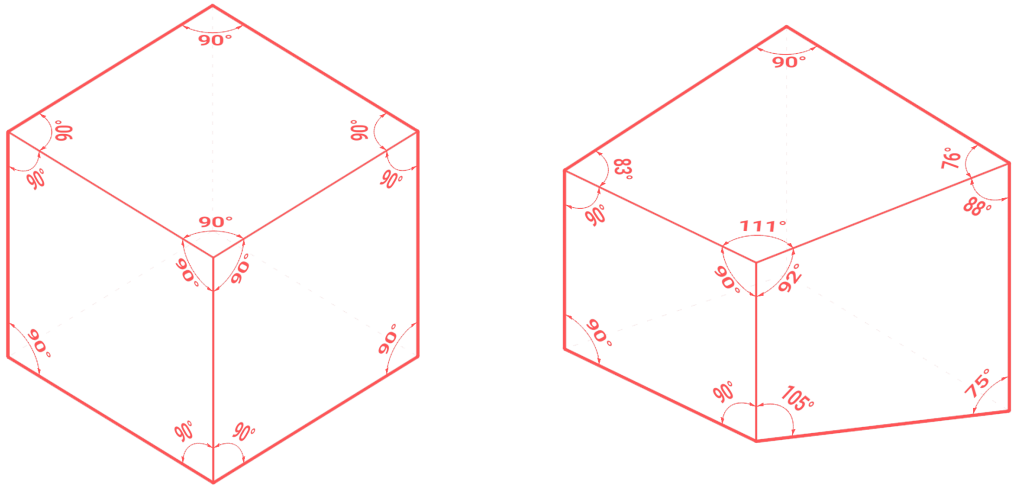


Figure 67: *Angle Dimensions on a Cube (left) and Deformed Cube (right)*

Arc Dimensions

Priority: High

Status Substantially Implemented

Current Features:

- Arc Dimensions measure the Radius (ISO 129-1 Section 7.3) and Arc Length (ISO 129-1 Section 7.6) of an Arc, defined by 3 points, where points 1 and 3 are the extremes of the arc.
- Arc Dimensions are aligned in-plane with the 3 points that define them.
- Arc Dimensions can have user-defined:
  - Font
  - Line Weight
  - Color
  - Font Size
  - Radius
  - Endcaps

Remaining Features:

- Add a toggle to allow for the display of only the radius, or only the arc length if desired.
- Add an option to display the arc length in degrees or radians

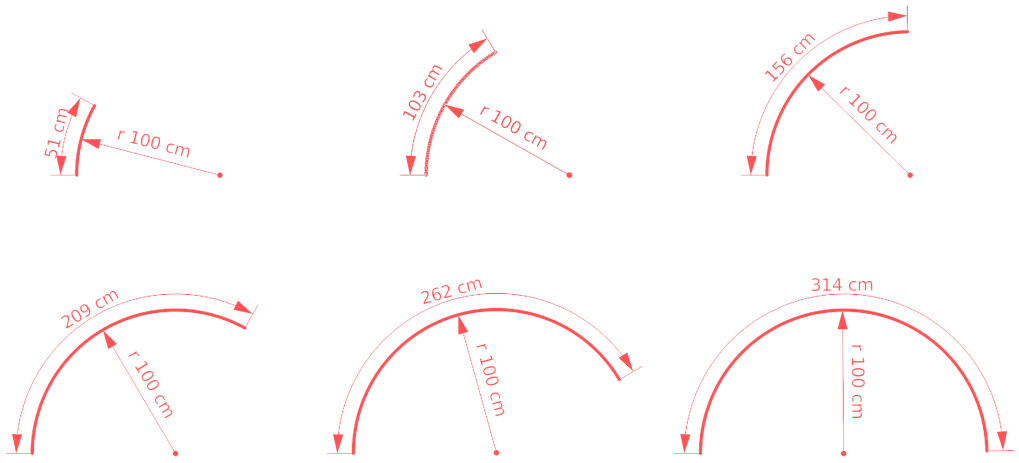


Figure 68: *Arc Dimensions at 30 Degree Increments with a Radius of 100cm (0 - 180 degrees)*

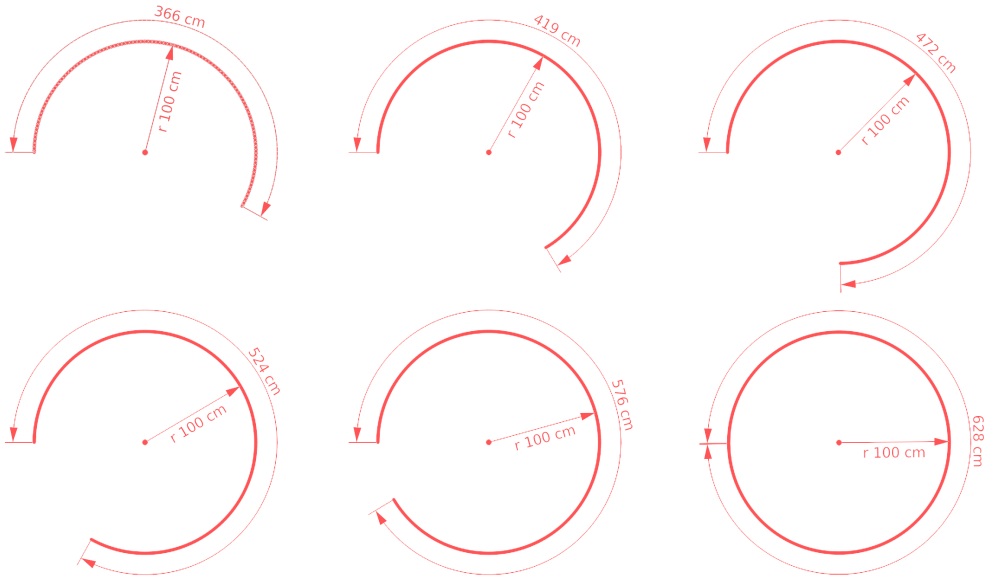


Figure 69: *Arc Dimensions at 30 Degree Increments with a Radius of 100cm (180 - 360 degrees)*

Room Area Tags

Priority: Medium

Status: Not Implemented

Schedules & Reporting

Priority: Medium

Status: Not Implemented

Title Blocks

Priority: Medium

Status: Not Implemented

General Requirements

Style System

Priority: Necessary

Status: Substantially Implemented

Current Behaviour:

- Styles are available for all MeasureIt-ARCH elements.
- Styles define the visual properties of that element type. Typically properties such as:
  - Color
  - Font
  - Line Weight
- Style use is user-defined on a per-element basis.
- The Styles UI is visually similar to its corresponding element type's UI panel
- A default style, that is applied to all new elements on creation, can be defined by the user.

Remaining Features: Not all styled properties can be overridden on a per-element basis, currently only a dimension's preferred view plane and camera visibility can be overridden per element. Some inspiration for the future development of this override system could be taken from the library override system currently being developed for linked data planned for a future release of Blender.

3D Integration

Priority: High

Status: Fully Implemented

Current Behaviour:

- All MeasureIt-ARCH elements are placed directly in 3D space, and do not require conventional 2D reference systems such as a 'work plane' or 'paper space'
- All MeasureIt-ARCH elements have an 'In Front' drawing option, which if selected allows the element to ignore occlusion and draw in front of all geometry in the scene.

Gizmo Implementation

Priority: Medium

Status: Partially Implemented

Current Behaviour: Accomplishing Gizmo Implementation for MeasureIt-ARCH has been challenging, as access to Blender's gizmo system through the Python API is mostly undocumented. Currently, gizmos for manipulating a dimension's offset, and annotation's offsets, and an annotation's rotation have been implemented by examining Blender's source code, and the few existing Gizmo template examples, in an attempt to understand the API implementation. However, many other properties could still benefit from gizmo support.

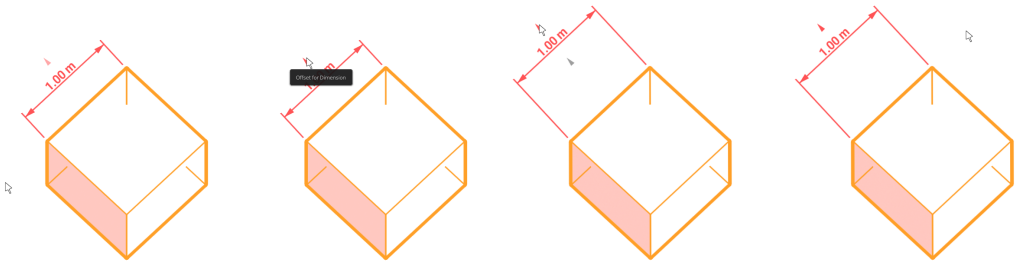


Figure 70: *Dimension Offset Gizmo Behaviour.*
(from left to right) Gizmo enabled. Gizmo displays tool tip on mouse hover. Click and drag to adjust gizmo property. Gizmo Released to confirm change.

Adaptive Behaviour

Priority: High

Status: Substantially Implemented

Current Behaviour:

- MeasureIt-ARCH lines adjust their appearance depending on their occlusion and silhouette state.
- MeasureIt-ARCH text automatically adjusts its orientation to remain legible to the user. (**Figure 74**) (See ISO-129-1 Section 5.6.2 Figure 23 for comparison)
- Aligned and axis dimensions adjust their placement to remain in the users view if possible. (**Figure 71**)
- MeasureIt-ARCH dimension text and termination symbols adjust their placement to avoid overlapping. (**Figure 75**)

Remaining: While MeasureIt-ARCH's dimensions and text both exhibit adaptive behaviour, annotations are not yet capable of the kind of automatic positioning that MeasureIt-ARCH's dimensions display. Future development could explore having annotation text rotate to stay parallel to the desired camera or viewport to improve annotation workflows.

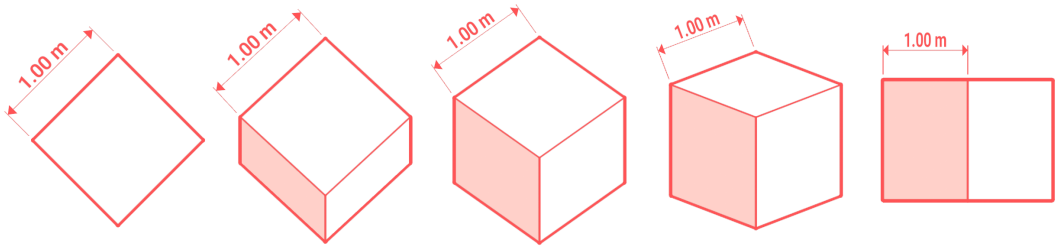


Figure 71: *Aligned Dimension Placement Behaviour.*
Image sequence showing MeasureIt-ARCH's Aligned Dimension placement behaviour. Note that the dimension position interpolates from being in plane with the cube's top face, to being in plane with the cubes side face, as the cube's orientation relative to the viewport changes

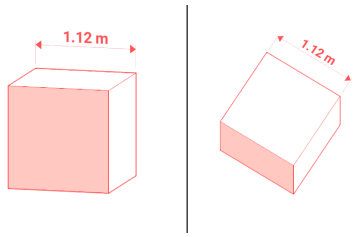


Figure 72: *A Cube with an Aligned Dimension, Viewed from Two Different Viewports Simultaneously.*
Note that the dimension can draw in two different positions in two different viewports, simultaneously, depending on the observer's position in that viewport.

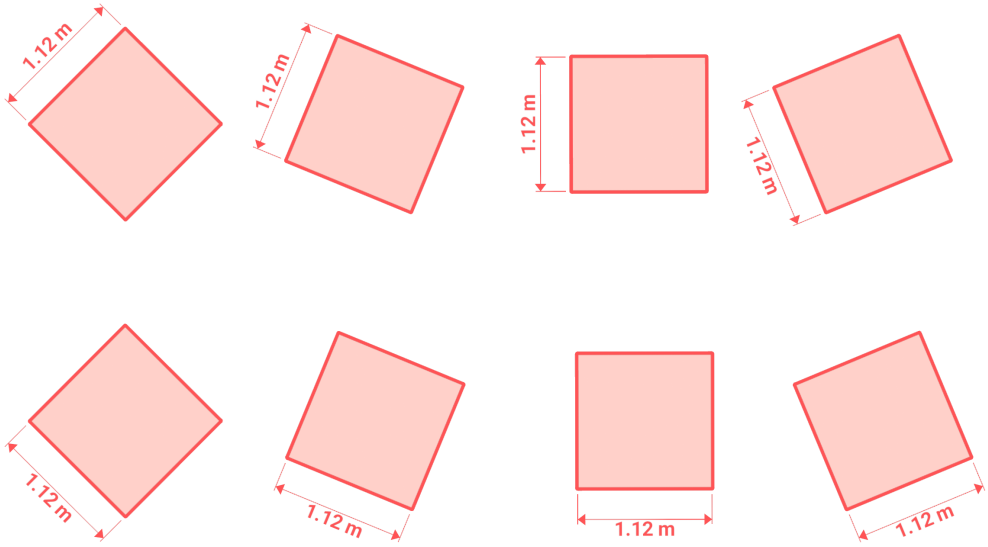


Figure 73: *Image Sequence Showing MeasureIt-ARCH's Text Orientation Behaviour.*
Note that the text flips between the third and fourth image in the sequence once it rotates past the acceptable threshold.

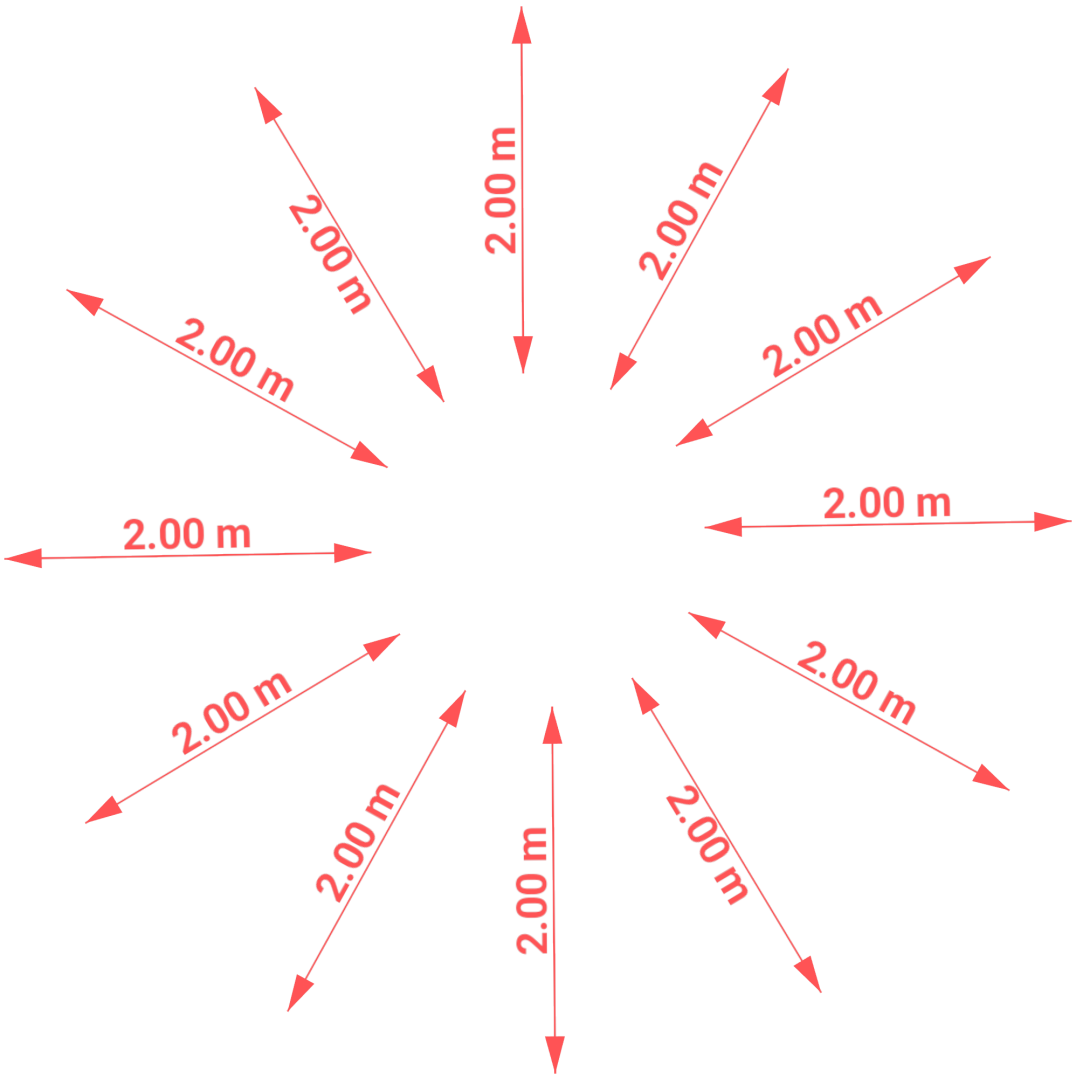


Figure 74: *Dimension Text Orientation Shown at 30 Degree Intervals*

Figure 75: *MeasureIt-ARCH Dimension Text Placement.*
A dimension's text and termination arrows automatically adjust their placement to avoid overlapping.

'Real Time' Responsive Performance

Priority: High

Status: Substantially Implemented

Current Behaviour:

- MeasureIt-ARCH's line drawing system averages between 50 to 70 ms per frame to draw 525,312 line segments (59.6 ms / frame measured using Python's time.time() function averaged over 100 frames. 53.2 ms / frame measured with Blender's built in Redraw Time, averaged over 10 frames).

Remaining:

- Draw call batching and caching should be implemented for MeasureIt-ARCH dimensions to help improve their draw speed.

Animated Properties

Priority: High

Status: Fully Implemented

Description: Building MeasureIt-ARCH within Blender exposes all of its properties to Blender's animation system. Any MeasureIt-ARCH property can be animated. These animated properties can be used for simple tasks, like toggling the visibility of drawing elements, or more complex animations can be produced, like the one shown below.

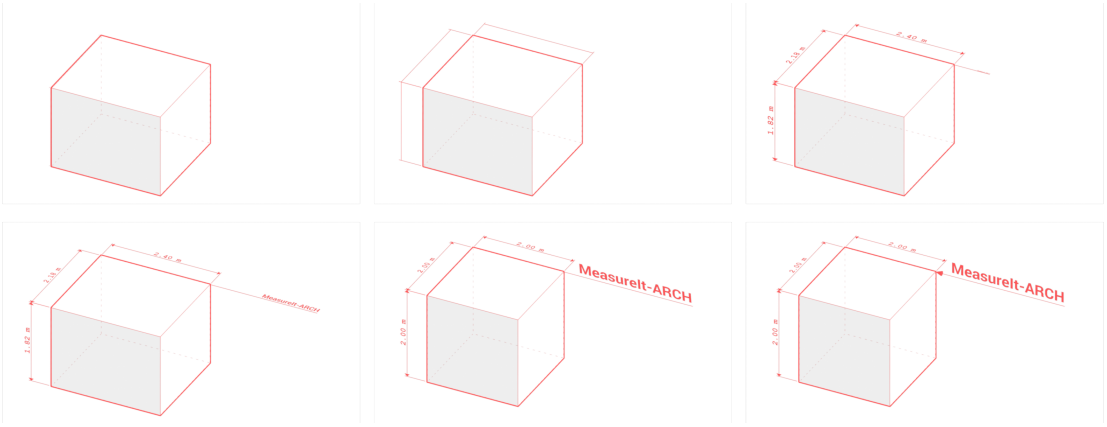


Figure 76: *Animated Sequence of MeasureIt-ARCH Elements From 'MeasureIt-ARCH Version 0.3 Update'*

Hybrid Rendering

Priority: High

Status: Substantially Implemented

Current Behaviour: All elements created by MeasureIt-ARCH are drawn as an overlay overtop of any of Blender's existing render engines in the viewport. When rendered MeasureIt-ARCH elements are drawn to a .png image file, that can be composited over Blender's render passes. As an overlay, MeasureIt-ARCH can facilitate hybrid forms of representation that make use of Blender's diverse options for stylistic or realistic rendering.

Remaining: MeasureIt-ARCH data is currently rendered as it's own image layer, which can be composited over the output of any of Blender's render engines. While this MeasureIt-ARCH layer is automatically composited over Blender viewport when working in the 3D scene, exporting a final rendered image requires manual set up in Blender's compositor (**Figure 79**). Further work could be done on integrating MeasureIt-ARCH's output directly with Blender's existing render passes, however work on this front is currently limited by the functions available in Blender API for manipulating the data in Blender's render passes. A possible solution would be to automatically generate the compositing setup required for the final render before rendering. Further development could also look at allowing each of MeasureIt-ARCH's element types to render to a unique render pass (or compositing pass) as well.

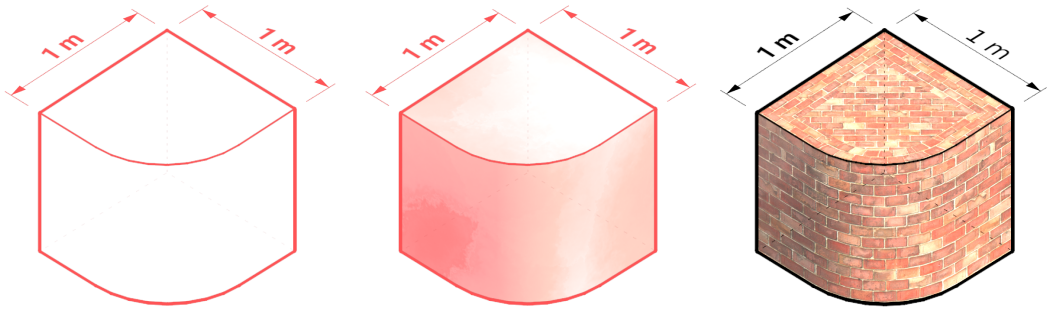


Figure 77: *MeasureIt-ARCH Linework and Dimensions Overlaid on Three Distinct Rendering Styles.*
(left) Flat shaded, (center) Stylized water color shading (right) Realistic Brick

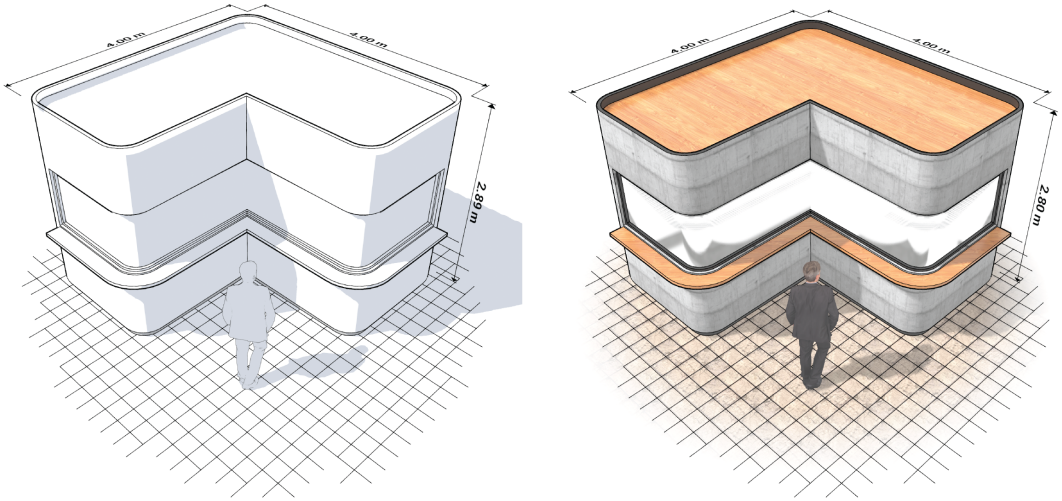


Figure 78: *A Small Pavilion with MeasureIt-ARCH Linework and Dimensions, Rendered in Two Styles*
Design inspired by the works presented in 'KIOSK' by Owen D. Pomery

76

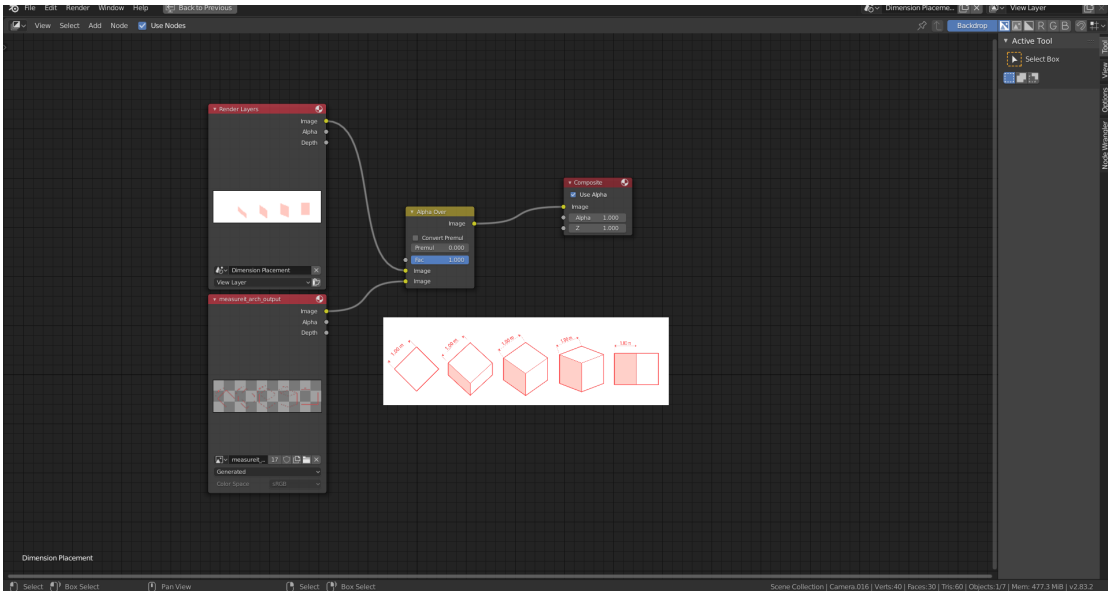


Figure 79: *MeasureIt-ARCH Render Compositing Setup*

[76] Owen D. Pomery, *KIOSK* (Architecture Research Press, 2018), https://owenpomery.com/kiosk

Instancing Support

Priority: Medium

Status: Partially Implemented

Description: Taking advantage of Blender's linked data capabilities, MeasureIt-ARCH annotations and line groups are able to be instanced, within the same Blender file or across multiple files, along with their host object. However, the current implementation does not support the instancing of dimensions. Dimension instancing is possible; however, the current system for drawing dimension text is not able to account for local changes in the scale or rotation of an instanced object when updating the dimension's text. To avoid the display of incorrect information, dimension instancing has been disabled until these errors can be fixed.

Full Instancing support, though outside of the primary objective of this thesis, is a significant priority for post-thesis improvements to MeasureIt-ARCH. A proper instancing system could begin to facilitate workflows closer to that of a BIM Level 2 capable software, allowing users to create linked libraries of objects that would function similarly to Revit's families.

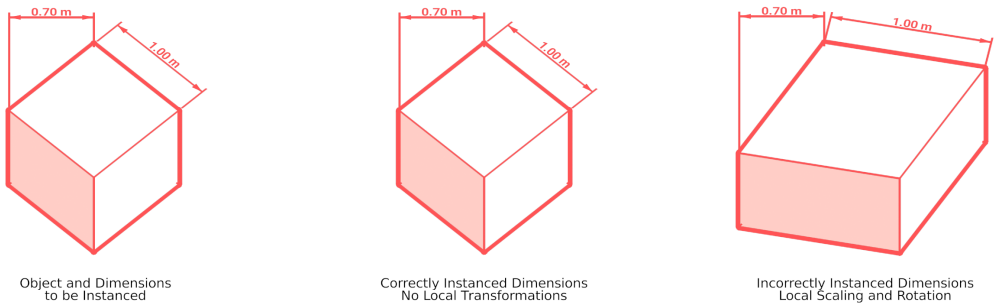


Figure 80: *MeasureIt-ARCH Linework and Dimension Instancing.*
(Left) Original object, (Center) Correct instance, no local rotation or scaling (Right) incorrect instance, local rotation and scaling are not accounted for in the dimension text.

Interoperability

Priority: Medium

Status: In Active External Development

Description: Although import and export functionality was not a direct focus of MeasureIt-ARCH's development during this thesis, it is a high priority for ongoing development. Fortunately, Blender's existing Import capabilities are quite strong. Blender currently supports the import of .dxf AutoCAD data, and .obj , .fbx , .3ds , and .dae files for the import of general-purpose 3D models. Blender also has rudimentary support for the importing of .ifc files thanks to the IfcOpenShell importer,[77] although the imported model does not currently maintain the class structures of the IFC standard.

Blender's weakness in this area is its export capacity. While it supports the export of many general purpose 3D model formats, like those mentioned above, there has been no capacity for the export of BIM data through IFC. However, Dion Moult, an architect from Sydney, Australia is currently in the process of creating an .ifc exporter,[78] and as of October 1st has been in touch about the possibility of integrating its support with MeasureIt-ARCH[79]

In addition to the IFC support, some future development focus should be given to the conversion of MeasureIt-ARCH elements to a viable vector format to allow for editing in external packages. Currently MeasureIt-ARCH elements can only be rendered out as raster graphics. The .svg file format seems like the best candidate here.

[77] IfcOpenShell, "IfcOpenShell," accessed October 7, 2019, http://ifcopenshell.org/.

[78] Dion Moult, "Blender IFC Exporter," GitHub, accessed October 7, 2019, https://github.com/IfcOpenShell/IfcOpenShell.

[79] Dion Moult, *MeasureIt-ARCH Features & Suggestions - Comment, Dion Moult*, 2019, https://github.com/kevancress/MeasureIt-ARCH/issues/4#issuecomment-536822024.

Scaled Camera Space

Priority: High

Status: In Active External Development

Description: Scaled Camera Space functionality has not been added directly to MeasureIt-ARCH at this point. However, I have previously implemented support for these features, as outlined in the specification, in Blender as part of the 'Per Camera Resolution' add-on created in the summer of 2018.[80] Merging the two add-ons in future development could allow for further quality of life improvements to MeasureIt-ARCH, allowing font sizes and line weights to be aware of, and impacted by, the intended paper size and scale.

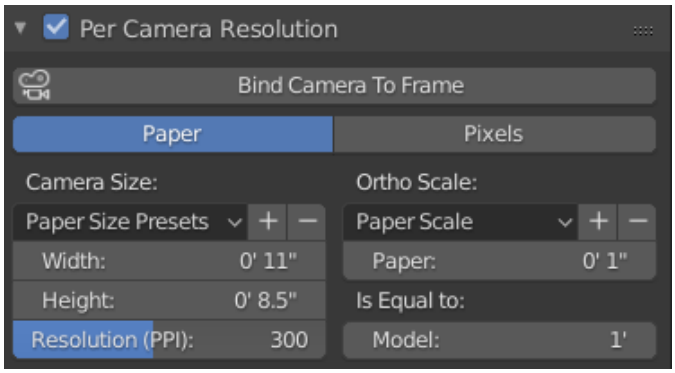


Figure 81: *The Current UI for the 'Per Camera Resolution' Add-on*

[80] Kevan Cress, *Per Camera Resolution*, 2018, https://github.com/kevancress/per_camera_resolution.

Summary

In its current state, MeasureIt-ARCH has reached Substantial Implementation or more on the majority of its necessary and high priority targets. As we will see in the drawings produced for the Lodge at Pine Cove, presented in the following section, the current state of MeasureIt-ARCH is capable of producing simple working drawings for small scale architectural projects. This, in itself, is a significant step forward for Blender's capacity for use as an Open Source architectural platform. That being said, our evaluation of the proposed specification shows that there is still significant room for improvement. Yet, the recent communication with other interested parties, such as Dion Moult, and the feedback from the Blender community through MeasureIt-ARCH's GitHub, shows a potential way forward that might begin to address the future high priority issues like Interoperability.

SECTION 6:

Testing and Implications

Testing and Implications Overview

Now that we've discussed what Blender and MeasureIt-ARCH are, what features they have, and how MeasureIt-ARCH was developed, the Testing and Implications section steps through three examples of how Blender and MeasureIt-ARCH might begin to be applied in architectural work.

First, we'll look at a direct application of Blender & MeasureIt-ARCH's current state. Showing how MeasureIt-ARCH's tools were used to produce several renovations drawings for the Lodge at Pine Cove.

Second, we'll take a look at how Blender and MeasureIt-ARCH can be augmented with other Add-ons to interface and communicate with other computational systems. To illustrate this possible future potential, a simple but dynamic simulation of the Noosphere at the Futurium in Berlin by Philip Beesley Architects Inc. has been developed as a proof of concept.

Finally, we'll revisit the software challenges facing today's Open Source Architecture (OSArc) projects, that we touched on in the motivations section of this thesis. We'll look at the WikiHouse project, and use their relation to software as a case study for how an Open Source design platform might help OSArc projects meet their goal of genuinely Open Source design.

These three provocations are intended to pique the reader's curiosity, and prompt discussion about directions for MeasureIt-ARCH's continued development after the completion of this thesis.

Continuing Work with the Lodge at Pine Cove

As mentioned in the motivations section of this thesis, my earlier work with the Lodge at Pine Cove in the summers of 2016 and 2017 helped me to identify Blender's potential strengths and weakness when used in a small scale architectural project. The design of two 1,000 sq. ft. Modular cottages during that time period showed Blender's capability for modeling and rendering, but also it's lack of available toolset for the creation of technical drawings, as we have discussed.

Thanks to the continued support of Alex Strachan and Daniel Viirtella of Pine Cove, I have had the opportunity to continue to work on design projects for the Lodge and utilize these projects as a practical testing ground for MeasureIt-ARCH during its development. To date, Blender and MeasureIt-ARCH have been used to design and produce preliminary plans and visualizations for eight upcoming renovations to existing Pine Cove Cottages, as well as preliminary construction drawing for the six of these additions. In addition to these renovation drawings, MeasureIt-ARCH and Blender have been used to produce the preliminary designs for a new Sauna for the Lodge (although whether the design presented in this thesis or an alternative floating sauna design will be chosen for actual construction remains to be seen).

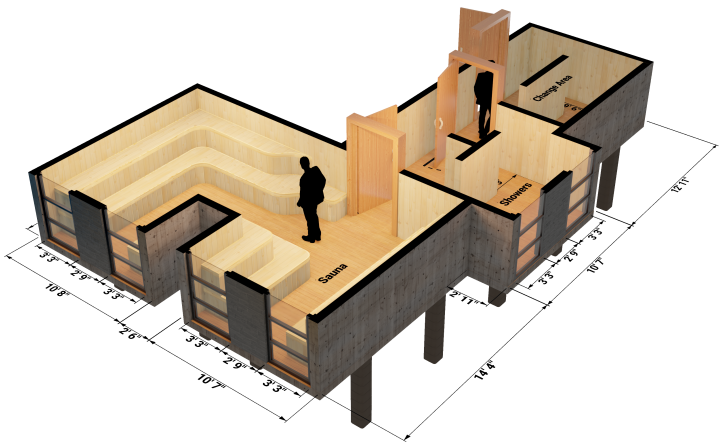


Figure 82: *Pine Cove Sauna - Cut Perspective, Overlaid with MeasureIt-ARCH Dimensions and Annotations*

Production Driven Improvements to MeasureIt-ARCH

The production of these technical drawings for the Lodge at Pine Cove has helped greatly to improve MeasureIt-ARCH throughout it's development. Using MeasureIt-ARCH in a production settings helped to challenge my own assumptions about how MeasureIt-ARCH's tools should be developed and how they should function. Some of the key improvements to MeasureIt-ARCH inspired through it's testing in the creation of the Pine Cove cottage drawings include;

The Line Group by Crease Operator

Attempting to use an early version of MeasureIt-ARCH to produce Linework for one the Pine Cove renovation drawings helped motivate the creation of the 'Line Group by Crease' operator, as it quickly became apparent when working to a deadline that MeasureIt-ARCH's direct selection method for the creation of Line Groups was far too time consuming and tedious to be the only method for Line Group creation. Instead it became apparent that a workflow that made use of the automated 'Line Group by Crease' operator to initially identify major feature lines, which could then be tweaked with the direct selection operators was a much more efficient workflow.

Empty Object Support

A frequent task during the creation of the Pine Cove cottage drawings was the need to create annotations that were not attached to any object's geometry. These 'floating annotations' were useful for adding general notes to a drawing, and for labeling room areas (as a dedicated room area tag has not yet been implemented). With early versions of MeasureIt-ARCH floating annotations were achieved by anchoring an annotation to a mesh object that contained only a single vertex, but this was somewhat cumbersome and time consuming to set up.

To solve this, support was added to allow MeasureIt-ARCH elements to anchor to Blender's 'Empty' object type. Empties are objects that have no mesh data, but appear in Blender's 3D view as a simple user defined shape. Empties are simple for the user to select, move, and rotate in the 3D scene, but they don't appear in any of Blender's output renders, which makes them an ideal anchor for floating annotations.

To make the creation of these floating annotations as simple as possible, an option was added to the 'Add Annotation' operator that would facilitate their creation. If the 'Add Annotation' operator is run when no object in the scene

is selected, the operator creates an Empty object at the users cursor location and anchors a new annotation to it. This new annotation's text is centered on the Empty with no leader line, making it ready to use as a floating annotation.

Support to allow MeasureIt-ARCH's dimensions to anchor to Blender's Empty objects was also added. This allows for dimensions that are not tied directly to an objects mesh geometry if the user desires. This can be used to make 'floating dimensions' that can be used as a sort of quick ruler in the 3D scene, or for representing datum lines.

Multi Line Annotations

Using annotations to add notes to drawings also required annotations to support multiple lines of text. This required a refactoring of MeasureIt-ARCH's text drawing system to support multiple text fields per MeasureIt-ARCH element. In the annotation settings, an operator is provided that gives users the option to add or remove text fields from MeasureIt-ARCH annotations, allowing for multiple lines of text to be entered and displayed per annotation. Due to MeasureIt-ARCH's inheritance structure, this support for multiple text fields is now also available for dimensions as well, however no UI options have been added yet to let users access this functionality. In the future this could be expanded to allow users to add custom text to dimension objects.

Blender and MeasureIt-ARCH; Hybrid Workflows

Work on the Pine Cove cottage drawings also helped to develop workflows for integrating MeasureIt-ARCH elements with Blender's rendering capabilities, informing practices that allow Blender's internal system to work in concert with MeasureIt-ARCH to allow for quick transitions between rendered visualization and dimensioned drawing. The two main techniques used to create the Pine Cove cottage drawings are;

Style Specific Render Engine Setups

A tremendously useful feature of Blender's material system allows each material to have a customized unique output for each of Blender's three render engines. What this allows us to achieve is a workflow where we tailor our materials to achieve a unique style in each render engine. Once our materials have been set up, we can switch between visual styles with the change of a single setting. In the workflow developed for the Pine Cove cottage drawings, this set up was used to create a single model that could quickly transition from a rendered presentation model, to a flat shaded technical drawing. To achieve this, materials were set up to output a textured and shaded appearance when Blender's EEVEE render engine was enabled, and to output a plain white flat shaded appearance when Blender's Workbench render engine was enabled. MeasureIt-ARCH elements can be composited over both of these styles if desired.

Boolean Cutting

Once MeasureIt-ARCH elements had been updated to function in a stable manner with Blender's Modifier system, it was possible to use Blender's boolean modifier to create plan and section cuts. To set this up, we create a mesh object that will act as our cutting volume, and assign all of the objects we want to be cut a boolean modifier that will procedurally conduct a boolean difference operation with our cutting volume. With this set up we can move our cutting volume however we like and see the resulting plan or section cut in real time, also since Blender's modifiers are 'non-destructive' we can simply disable our boolean modifier, or move the cutting volume away from our objects to restore our model to it's un-cut state. We can also take advantage of another of Blender's material properties to further improve our plan and section cuts.

The cut faces of a Blender object that has been effected by a Boolean modifier will inherit the material assigned to the cutting volume, so if we assign a black flat shaded material to our cutting volume, Blender will

procedurally apply this material to any cut face. This technique can be used to quickly and procedurally adjust the location and appearance of plan and section cuts in our Blender model, over which our MeasureIt-ARCH drawing elements can be composited.

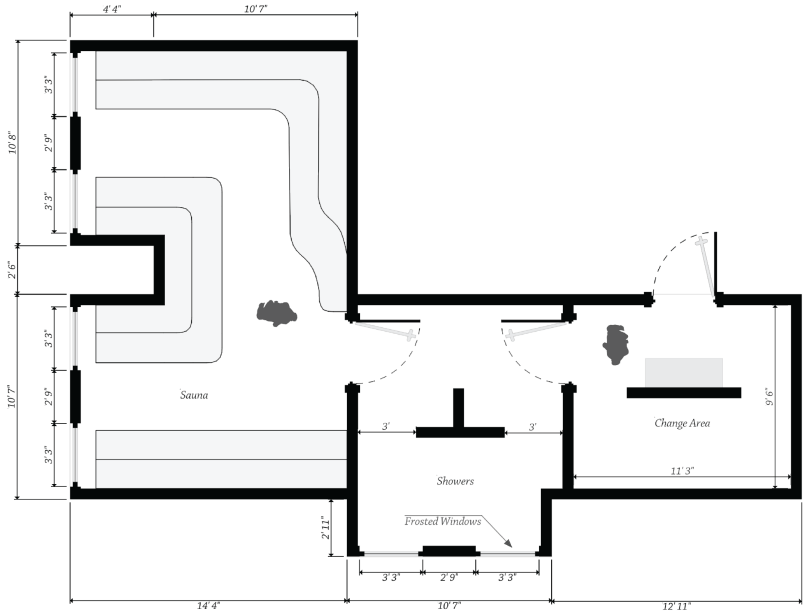


Figure 83: *Pine Cove Sauna - Conceptual Plan*
Linework and dimensions drawn by MeasureIt-ARCH

Figure 84: *Pine Cove Sauna - Conceptual Render*
Blender EEVEE render overlaid with a MeasureIt-ARCH plan and section.

The Example Project

The Blender scene file for the version of the Sauna project used to produce the images presented in this thesis is publicly available in the Demo Files folder of MeasureIt-ARCH's GitHub repository. This demo serves as an illustration of how dimensioned plan, section and elevation drawings, as well as presentation renders, can all be generated in the same scene, from the same 3D model using Blender and MeasureIt-ARCH.

Drawing Examples

The drawings presented on the following pages provide a sampling of the work done for Pine Cove with Blender and MeasureIt-ARCH

Figure 85: *Paul Kane Cottage - Proposed Renovation Plan, with MeasureIt-ARCH Drawing Elements.*

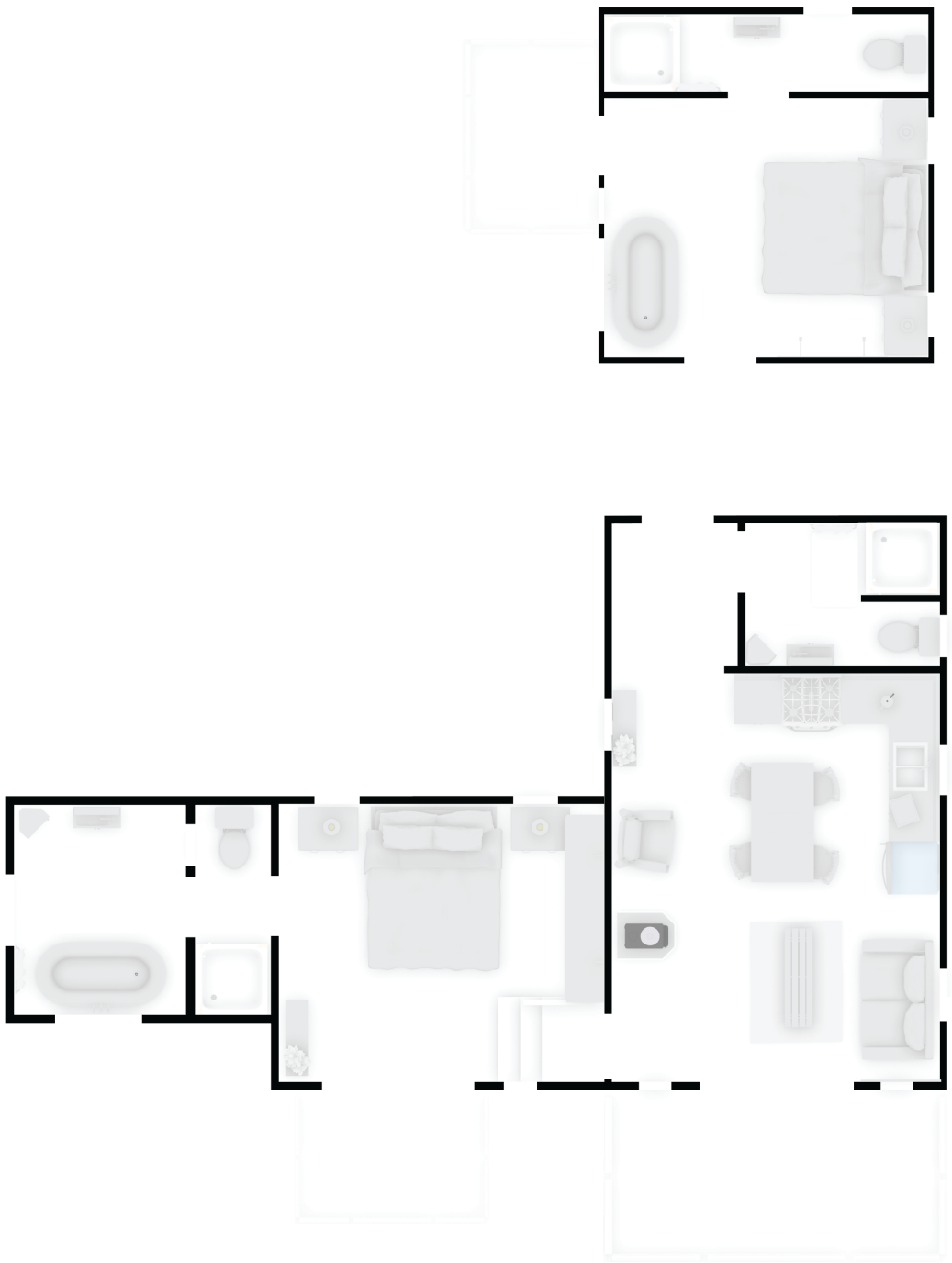


Figure 86: *Paul Kane Cottage - Proposed Renovation Plan, without MeasureIt-ARCH Drawing Elements.*

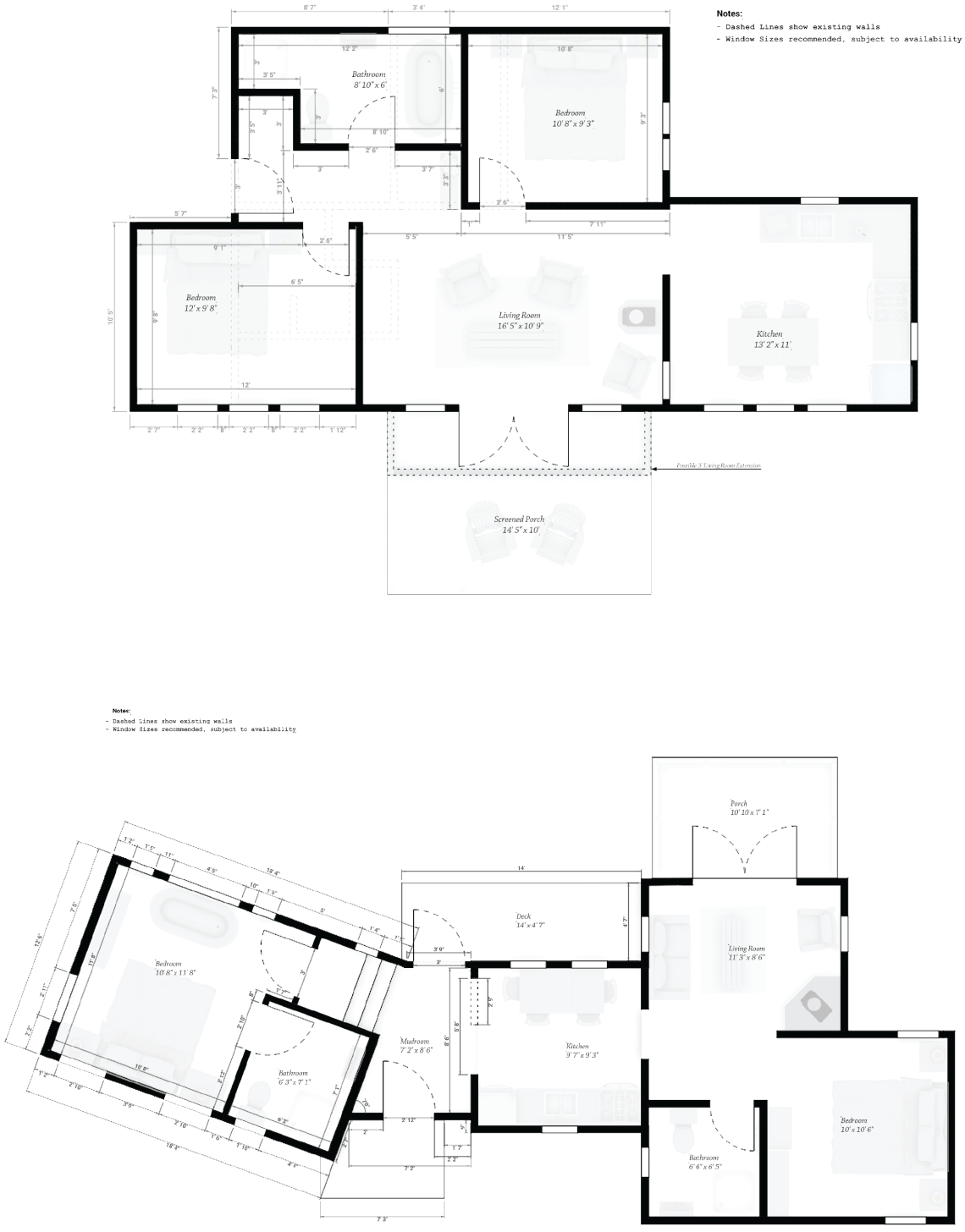


Figure 87: *Two Additional Proposed Pine Cove Cottage Renovation Plans with MeasureIt-ARCH Drawing Elements.*

Dynamic Data and Linkages

Our second test with Blender and MeasureIt-ARCH is a simple dynamic visualization of the Noosphere installation at the Futurium center in Berlin, by, the Living Architecture Systems Group (LASG) and Philip Beesley Architect Inc. (PBAI) Thanks to support by Philip Beesley, Matt Gorbet, Michael Lancaster, and others from the LASG and PBAI, a system was developed to allow the 'brain' of the Futurium Noosphere, written in the Processing programming language, to communicate directly with Blender over the Open Sound Control (OSC) Network protocol to drive various properties of a Blender visualization. This visualization helps to highlight MeasureIt-ARCH's annotation system, namely its ability to draw from custom metadata and update to changes in that metadata in real-time, even when that data is being driven from external sources. It also illustrates the potentials that arise from chaining together multiple Open Source Add-ons and external sources using Blender as a common platform.

The Futurium Noosphere and the Processing Simulator

The Futurium Noosphere is a mass of dynamic sensors, actuators and microprocessors embedded in a body of stretched spars of steel and acrylic. Mylar fronds and Rebel Star LED's articulate and pulse in response to information from the sculpture's sensors and underlying behaviour algorithms. Controlling the Noosphere's behaviour is the Processing Simulator developed by the team at PBAI. The Processing Simulator drives the actuators of the sculpture, communicating with the various microprocessors to receive sensor input, which it interprets based on its behaviour algorithms before returning an action for the actuators. However, this dual function piece of software doesn't only drive the physical sculpture. It also provides a digital symbolic representation of the sculpture's sensors and actuators. The Processing Simulator tries to mimic its physical counterpart as closely as possible, communicating with its virtual microprocessors using the OSC network protocol, just as it does with the physical sculpture. This virtual double of the Futurium Noosphere allows the team at PBAI to explore and simulate adjustments to the sculpture's behaviour before the sculpture's physical construction, and see its behaviour represented symbolically.

The Processing Simulator is, as its name suggests, coded in the processing programming language. It is a purpose-built, from the ground up, 3D

environment. The simulator gives the team at PBAI very fine-grained control of the sculpture's behaviour code but results in a somewhat symbolic, abstracted representation of sculpture. This level of abstraction is ideal for a control interface, but a visualization of higher visual fidelity, presented in tandem with the Processing Simulator could help simulate and give the team at PBAI some feedback on the more qualitative aspects of the sculpture's nature. As a proof of concept to show how Blender & MeasureIt-ARCH might be able to create this kind of high fidelity simulation, the following exercise was conducted.

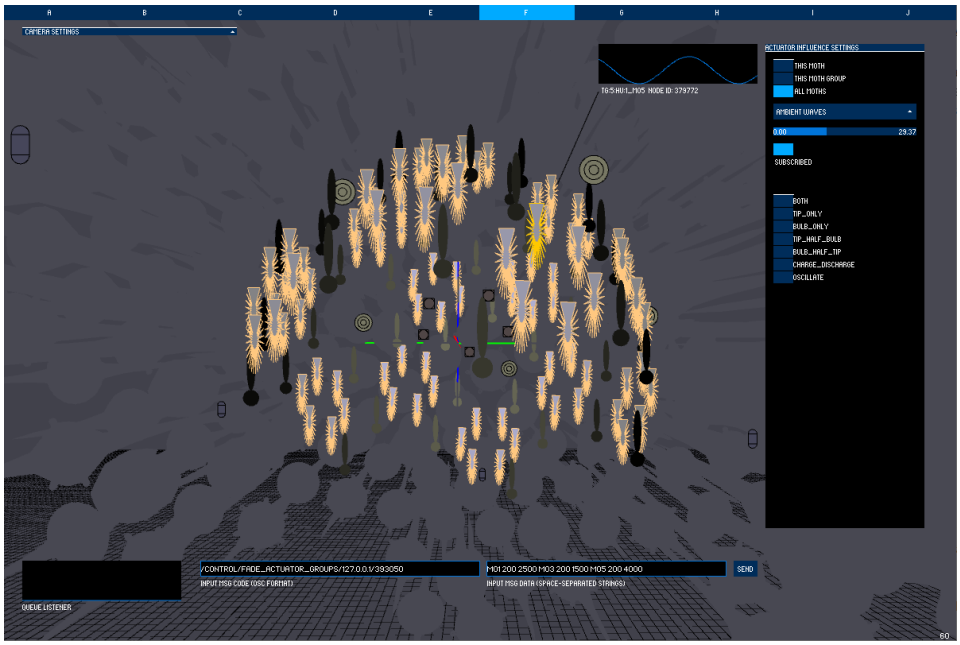


Figure 88: *The Futurium Noosphere as Represented in the Processing Simulator*

The OSC Parser

To allow Blender and MeasureIt-ARCH to receive data from the Processing Simulator, Blender needed to be able to speak the same language as the Noosphere sculpture, that is, It needed to be able to communicate using the OSC (Open Sound Control) network protocol. Implementing basic OSC support was accomplished using JPfeP's Blender OSC Communication add-on.[81] This add-on provides essential support for the receiving of OSC messages. However, its implementation didn't support the style of OSC messaging necessary to communicate with the Processing Simulator.

An OSC message consists of 3 major parts;

1. A **client** that will receive the message
2. An OSC **message address**,
3. The **data** that the message contains

The challenge of implementing OSC in Blender is defining how these last two parts of the message, the message address and data, get interpreted and mapped onto the Blender properties that you want to control. In JPfeP's original add-on, every Blender property that you wanted to control had to be allocated to its own OSC message address. This one-to-one mapping works fine for simple implementations but does not scale well when dealing with the volume and speed of messaging necessary for a complicated installation like the Futurium Noosphere.

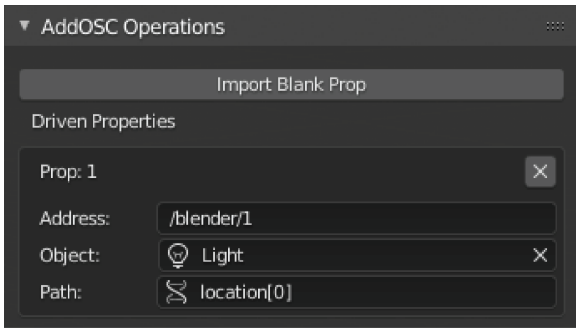


Figure 89: *Blender UI for a Simple OSC Mapping.*
The OSC address "/blender/1" maps to the location[0] property of the object "Light"

[81] JPfep, "Blender AddOSC Addon," accessed August 17, 2019, http://www.jpfep.net/pages/addosc/

Figure 90: *A Simple OSC Test; Virtual Coffee and a Distance Sensor*

Rather than sending a unique address and message for every actuator, the Processing Simulator concatenates OSC data into a single string that is sent to the microprocessor responsible for setting the properties of the desired actuators. This helps reduce the number of messages being sent each second, helps to improve the performance of the simulator and keeps the sculptures running smoothly, but this messaging structure does require some extra interpretation on the receiving end. An address from the Processing Simulator takes this format;

"/CONTROL/FADE_ACTUATOR_GROUPS/393072"_

Where "393072" refers to the particular microprocessor being addressed. The data sent to that message address might look something like this;

"MO1 36 10 MO2 39 10 MO3 30 10 MO4 20 10 MO5 18 10 MO6 25 10 DR1 65 10 DR2 48 10 DR3 13 10"

This data contains the values to be used by nine actuators. Each actuator

value starts with a Prefix that identifies it, for example, "*MO1*" refers to the first "Moth" actuator (The vibrating Mylar fronds) while "*DR3*" would refer to the third pair of Rebel Star LEDs. The value immediately following the identifying prefix is the data meant for that particular actuator (MO1 gets set to 36, and DR3 gets set to 13), and the following value is the expected time between messages in milliseconds, in this case, 10 ms for all actuators.

To allow Blender to interpret these messages, a Message Parser was added to JPfeP's OSC add-on. This parser acts as a sort of stand-in on Blender's side for the simulator's virtual microprocessors. Each parser has a unique message address, splits the incoming string of data for that address into a list, and provides a UI to allow the user to specify a unique mapping for each of the incoming messages prefixes.

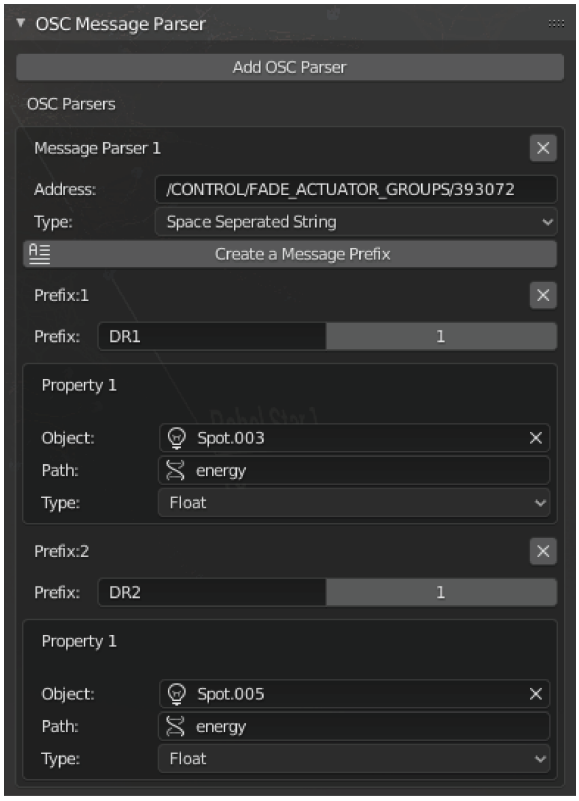


Figure 91: *Our UI to Parse a Message from the Processing Simulator*

Modeling the Futurium Noosphere for Performance.

Now that we are able to parse and receive messages from the Processing Simulator, we need a model of the Futurium Noosphere that we can map our message data on to. The Noosphere is incredibly dense as a digital model, being made up of hundreds of components each with complex semi-organic forms. Thankfully by taking advantage of Blender's object instancing we can optimize our digital model of the Noosphere to be suitable for real time rendering.

Running _testmaxspeed in Rhino on the model of the Futurium Noosphere provided by PBAI returns an average viewport performance of 2.04 FPS, which is 489.2 milliseconds per viewport update. To use this model for real-time rendered data visualization well need to improve on this significantly. To accomplish this, our Blender model of the Futurium Noosphere can be broken down to it's component parts and re-assembled using Blender's linked data instancing. First we create a library file containing the five main components that make up the Noosphere. From this library, the components are linked into the our main Blender file where the simulation will occur, and each component is attached to an empty placeholder object, generated from the original Rhino file of the Noosphere. Each of these Empty objects contains the components correct location and rotation when in position in the assembled sphere, and instancing the correct component at each empty leaves us with a fully populated model of the sphere.

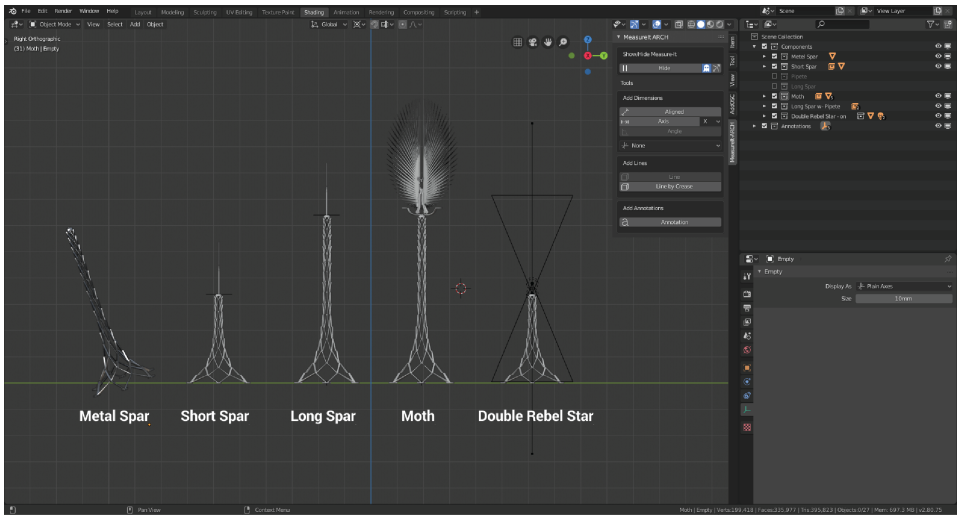


Figure 92: *The Futurium Noosphere Component Library*

Once the sphere is assembled from the linked components, we can create proxy objects of the elements whose properties we would like to drive from the OSC parser, and who's values we would like to annotate with MeasureIt-ARCH. Proxies allow for local changes to an instance of a Blender object that has been linked from a library file. Once we've made Proxies of the elements we'd like to be able to manipulate, we can assign their properties to a message prefix in the OSC parser, and add a MeasureIt-ARCH annotation to display that properties current value. Currently, MeasureIt-ARCH annotations can only display data from an object's custom properties, and not any of the object's standard properties. This limitation means that, for now, we must use Blender's drivers to synchronize our annotation-displayed custom property with the object property being controlled by the OSC input.

Rendered with the EEVEE render engine, with MeasureIt-ARCH overlays, and OSC input from the Processing Simulator driving four lights, Blender's debug timer reports viewport update times that average around 142 milliseconds per viewport update, roughly 3.5 times faster than Rhino's basic shaded viewport. Of course, 7 FPS is still not fast enough to be considered truly 'real-time'. However, there are still significant optimizations that could be made, to the geometry of the Noosphere components, the draw code of MeasureIt-ARCH, and the property assignment code of the OSC parser, that could all improve the performance of this visualization.

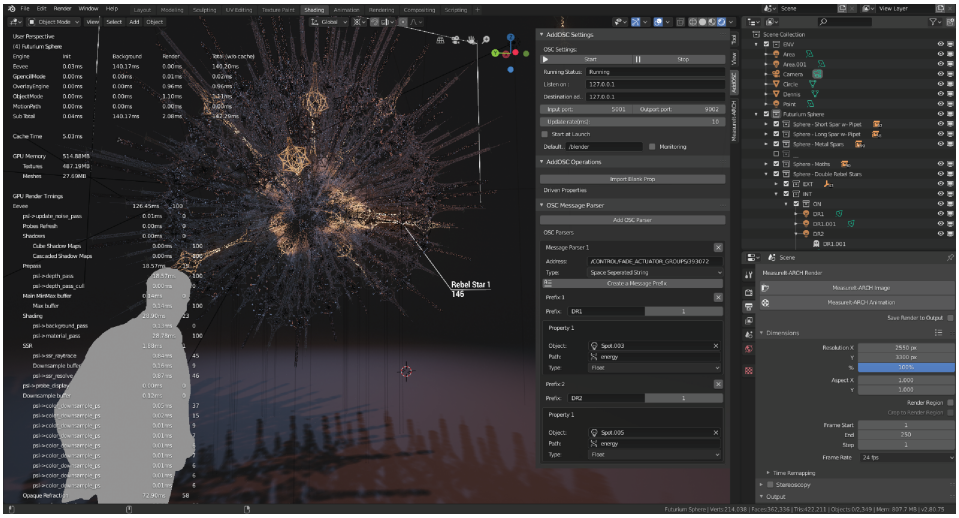


Figure 93: *Rendered Noosphere Visualization with Debug Timings*

The WikiHouse Project and Open Systems Labs

WikiHouse is an 'Open Source' architectural project founded by Alastair Parvin and Nick Lerodiaconou in 2011. The WikiHouse project is a contemporary attempt to leverage the design talents of the masses to produce an innovative system for affordable housing.[82] Their unique structural system WREN consists of an assembly of pieces that can be cut from standard plywood sheets on a CNC router and used to construct a home. All plans and models developed by the project are released for free on the organizations GitHub and are released under the MPL2.0 license, which allows collaborators to download and modify their contents as they see fit.

WikiHouse has seen a good deal of media coverage, over a dozen completed projects,[83] and general praise as a project that directly advances local manufacturing and low-income housing and construction. Whether the project meets its mandate as an 'Open Source' architectural project though is up for some debate.

The last commit to any of the repositories in the WikiHouse's GitHub page was, at the time of my writing, in November of 2018. The projects Slack channel on the other hand, called WikiHouse-Contributors, sees regular daily discussion about the project and its development. Sharing the latest information about the project through slack is somewhat problematic for an Open Source project however, as slack only allows partial access to the discussion history of a channel to those without a premium subscription, although slack is the least of WikiHouse's problems when it comes to proprietary software.

[82] *Architecture for the People by the People*, 2013, https://www.ted.com/talks/alastair_parvin_architecture_for_the_people_by_the_people

[83] Open Systems Lab, "WikiHouse," accessed August 21, 2019, https://www.wikihouse.cc/projects

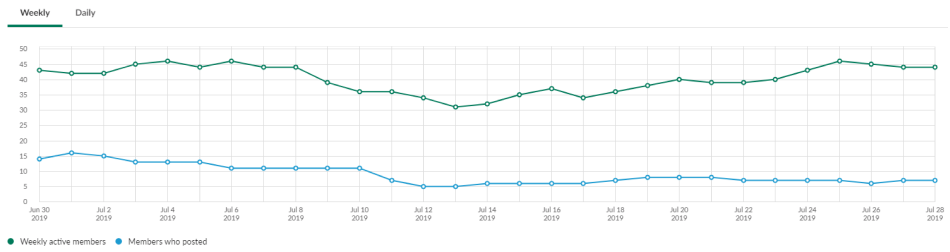


Figure 94: *WikiHouse Contributors Slack Engagement - July 2019*

Older WikiHouse designs like the Microhouse were based initially in SketchUp. SketchUp, previously the gold standard of intuitive, publicly approachable, 3D drawing packages, used to be an excellent option for the production and sharing of OSArc projects. Originally developed by '@LastSoftware' in 2000, SketchUp was purchased by Google in 2006. Google proceeded to release both a free and pro version of the software. The $0 price tag, intuitive interface, basic dimensioning tools, plus its integration with google earth, made SketchUp a popular choice for many. However, in 2012 Trimble Inc. purchased Sketchup from Google. In the seven years since Trimble's acquisition, they have redirected SketchUp's development. Trimble has taken SketchUp from an introductory but capable 3D modelling tool aimed at beginners and hobbyists to a reasonably well-featured BIM software package with the addition of tools like SketchUp Layout. While this push has been a positive development for small to mid-size Architectural firms that make use of SketchUp, the refocused development has left the free version of the software, behind as a casualty.

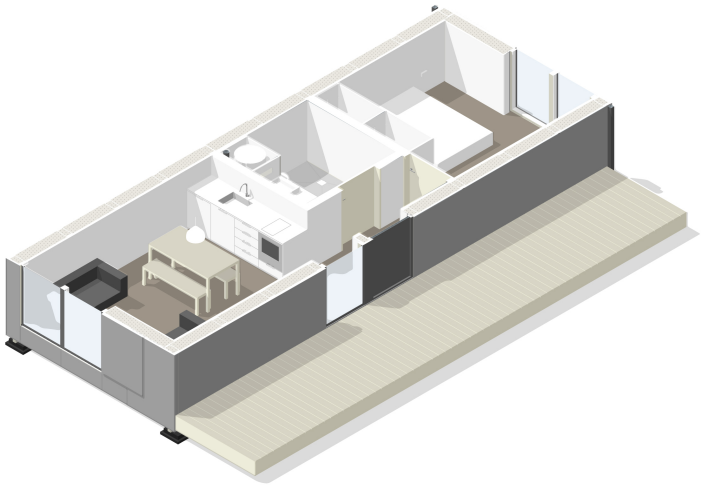


Figure 95: *Wikihouse Microhouse Iso.*
Copyright Open Systems Lab, Licenced CC BY-SA 3.0

In early 2017 on a GitHub Issue discussing improving the open availability of WikiHouse project data, Clayton Prest, the hardware and R&D lead of WikiHouse and its sister project Open Systems Lab noted that "*We have used SketchUp as the default format to date as that's where much of the earlier WikiHouse contributions were modelled. But I agree that this no longer makes sense, "*.[84]

In November of 2017 the free version of SketchUp, SketchUp Make, was discontinued and replaced with a severely limited online-only web application.

The loss of SketchUp as a capable, publicly approachable 3D modelling tool has left OSArc projects like WikiHouse in a tough situation with limited options. They're left to either, ask individual contributors to invest their personal funds into purchasing a commercial software package in order to engage with the project, or, invest a portion of the projects time and capital into the production of a bespoke design tool that lets the public engage with their system.

[84]Clayton Prest, "Wikihouseproject / Microhouse Issue #6," January 25, 2017, https://github.com/wikihouseproject/Microhouse/issues/6#issuecomment-275063114

Currently the 'source code' that WikiHouse has shared for the WREN system is based in Rhino's Grasshopper, which means a minimum buy-in of $995 for an individual to engage with the project, and McNeil's Rhino is one of the cheaper options when it comes to architectural design software. To an established architecture firm, $995 may seem like a nearly insignificant cost. But it's important to remember that the lifeblood of an Open Source project is individual, volunteer participation,[85] which means that for a project to grow and flourish the barrier to entry must be as low as possible. Lowering the financial barrier to entry for OSArc projects is a difficult problem to overcome when looking at the landscape of architectural software available today, which has become more costly and proprietary year by year.

Even WikiHouse itself seems to be moving away from Rhino and Grasshopper, and towards the creation of their own bespoke design tool, Buildx.

Push to Develop Proprietary Software

> "*for rules-based building systems like WREN we think there are limitations to common CAD software. The WikiHouse Foundation is currently developing a web-based platform for open digital construction, to make the design and development of WikiHouse technologies much much easier. It will have a front end for architectural design and back end for system R&D, which will be closer to coding than CAD, but still intuitive for visually-minded designers to use*" - Clayton Prest[86]

The motivation to design a custom piece of design software seems to be motivated by two significant concerns;

1. According to Prest, the WREN building system is pushing the limits of what Grasshopper can achieve as a parametric modelling system.

   > "*We are definitely not looking at using Grasshopper long-term, or even medium-term in the development of the Buildx platform. We're already pushing the performance limits of Grasshopper/Rhino because its graphics processing can only run on a single thread, and WikiHouse models are rather large!*" - Clayton Prest[87]

[85] Raymond, *The Cathedral & the Bazaar*

[86] Prest, "Wikihouseproject / Microhouse Issue #6."

[87] Clayton Prest, "WikiHouse-Contributors / Software Dev," WikiHouse-Contributors, May 16, 2018, https : / / wikihousecontributors . slack . com / archives / C15DW4XBK /

2. The WikiHouse team wants to make designing with the WikiHouse system more accessible by

   "*developing web infrastructure powered by game-changing automation technology to make development simple, transparent and low risk.*"[88]

Figure 96: *Buildx Prototype*
Designed by Open Systems Lab

While this system would certainly open up the use of the WikiHouse WREN system to a greater number of individuals, it simultaneously codifies and obfuscates the rules and logic that drive the system. It simplifies the end-user experience while making it more difficult for users to make changes to, and iterate on, the underlying parametric rules that drive the WREN system. The tendency for OSArc projects to move towards these kinds of hyper-specific bespoke design tools is another complication with the OSArc movement highlighted by Vardouli and Buechley.

> "*in all these prototypes, implementations and visions, the user acquires access to design via a black box—a design software environment— that embodies its author's assumptions, knowledge and expertise. Users do not have the ability to access or modify this black box. Although pragmatically, these platforms for participatory design can be argued to "open" design, to non-expert users, they are themselves closed and inaccessible. "* - Theodora Vardouli and Leah Buechley[89]

Bespoke design tools like Buildx are inherently siloed. They serve a

p1526490467000617

[88] ibid.

[89] Vardouli and Buechley, "Open Source Architecture."

single task and enhancing or augmenting their capabilities requires direct investment by the group managing their development. Of course, whether or not this is the desired outcome of the project is entirely up to the WikiHouse team itself to decide. Even as a bespoke tool the Buildx System has the potential to be a design tool that could give more people than ever before the opportunity to design there own affordable housing. It also has the potential to help make distributed manufacturing available to the masses, which could be a dramatic step forward for sustainable social housing. All of this can be achieved by the project regardless of its 'Open Sourced-ness' and we should celebrate the WikiHouse project and its team for these feats.

However, the question does remains; if there were a freely available tool that could encode the parametric logics of the WREN system and provide them to users through a simple accessible interface, while still maintaining the capacity for deeper editing and connection with other tools available to those users who desired to dive deeper, could it better meet the needs of the WikiHouse Project and future OSArc projects?

Blender and MeasureIt-ARCH's Implications.

Although the complexity of the WikiHouse system is beyond what Blender and MeasureIt-ARCH are *currently* capable of representing, it would seem there is some space in the gap between commercial complex design tools like Rhino, and free simple bespoke design systems like Buildx. This might be an ideal niche to explore for future development of MeasureIt-ARCH and Blender. If, for example, a parametric system like Buildx could be added into Blender, either explicitly as an add-on or through the proposed parametric node system being developed by Jaques Lucke, could it be used as a simple interactive design tool, as Buildx is currently, while still allowing interested users to take advantage of Blender's general-purpose modelling tools? Could a hybridization of bespoke parametric systems with general-purpose modelling tools allow for greater customization and more individual expression and refinement, depending on the user's interest and ability? Further still could Interfacing with other add-ons, Like MeasureIt-ARCH, allow for documentation or even further hybridization by connecting with other OSArch systems?

Although MeasureIt-ARCH and Blender do not yet have the capability to answer these questions, I hope that our discussion of MeasureIt-ARCH's development, and its current capacities, paired with these questions help to paint a picture of what might be achieved with Open Source software in architecture in the years to come.

If MeasureIt-ARCH is the first small step towards encouraging Blender's use as a general Open Source platform for architectural design, then Open Source Architecture likely represents the ideal niche of the architectural profession to focus on for future development. If that is to be the case, then improving the ability to interface and add support for OSArc design projects like WikiHouse and its WREN structural system will be an essential next step.

Conclusion

Looking Beyond this Thesis

> "*When you start community-building, what you need to be able to present is a plausible promise. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete […]. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future.*" -Eric S. Raymond[90]

MeasureIt-ARCH, as it stands at the end of this thesis work, is a toolset that is integrated closely with Blender's established design and is capable of creating simple responsive drawings that embrace and engage the inherent complexity and opportunity of three dimensional space. MeasureIt-ARCH's dimensions respond to the user's viewpoint and make use of their relationship to an objects geometry to determine a reasonable placement. MeasureIt-ARCH's linework provides a fast and simple toolset to create line drawings comprised of plain lines, hidden dashed lines and silhouette lines that draw and update in real time. MeasureIt-ARCH's annotations allow users to display their own text, or information from an objects custom properties as notes in the 3D space.

MeasureIt-ARCH is a functioning piece of free and Open Source software that allows for the creation of simple architectural drawings within the Open Source 3D creation tool Blender.

MeasureIt-ARCH is, of course, not without its faults and limitations, and still has a long way to go before it can be considered a useful tool for medium or large-scale architectural projects. The addition of tools that make better use of Blender's object metadata and linked data workflows, to produce schedules, interrogate a models information, and allow for the integration and layering of models from different disciplines are all improvements that could be achieved in the not so distant future. In the shorter term, MeasureIt-ARCH's user experience could benefit from the proper implementation of manipulator Gizmos that allows the user to adjust the properties of a dimension without leaving the 3D viewport, and optimization of the drawing system to allow for better performance when dealing with a large number of dimension elements on screen, to allow for the drawing of more complex objects.

Most importantly though, I hope MeasureIt-ARCH represents what Eric S. Raymond refers to as 'A Plausible Promise'. A small spark that might inspire other to think about the sorts of tools and improvements they might be able

[90] Raymond, *The Cathedral & the Bazaar*.

to make and share with others.

Its worth a moment to here in the conclusion to reflect on the three areas that motivated this thesis which we presented in the first section of this work. Now that we've begun to augment Blender to a level where it can tackle small scale architectural projects, could it start to provide a cost-effective option for small firms and individual designers? Could it start to allow Open Source Architecture projects to truly meet their Open Source mandate, allowing them to produce their drawings and models in a format that is available to be read and modified by anyone? Could it start to serve as an Open Source platform upon which the AEC industry might start to build a culture of distributed development, providing opportunities for the passionate community of architectural technology innovators to fold their advancements back into a common platform for all the industry to use?

Only time and more development will tell, but I hope that the potential of each of these questions helps to inspire others to engage in future developments that move us, small step by small step, towards a more open future.

MeasureIt-ARCH's development does not end with this thesis. Thanks to the skills I have learned through this exercise in the craft of digital tool making, I can continue to work on and improve MeasureIt-ARCH, and I hope that through sharing MeasureIt-ARCH's development process, as well as the larger issues that motivate it, others may become interested in participating in MeasureIt-ARCH, Blender's development, or the craft of digital tool making in general.

MeasureIt-ARCH and Blender have the potential to take further steps towards becoming a useful Open Source design platform for architects, that provides a free and open option for the creation of architectural drawings that can be shared, read, and modified by anyone.

The most promising of these future developments will be those that draw heavily on cross-collaborative work. Whether that collaboration focuses on practical application like the Pine Cove project, dynamic linkages with other systems like the Futurium Noosphere project or OSArc activism like the WikiHouse project, each of these avenues has the potential for unique developments that could add to the virtual palimpsest of ideas and implementations that make up Blender as an Open Source platform.

If we wish to foster modes of working that take real advantage of the skills and design thinking that our industry has to offer, then we need to ensure that our software offers the potential for bottom up, architect-driven modification and enhancement. As a profession, we have a responsibility to take an active role in shaping the tools that increasingly define our work. MeasureIt-

ARCH is hopefully only one of many small steps that we as a profession can take together towards making a viable Open Source architectural Software, that is free to be reshaped, redesigned, redeveloped by the best ideas our profession has to offer.

References

History & Conventions of Dimensioning

Brothers, Cammy. "What Drawings Did in Renaissance Italy." In *Companion to the History of Architecture*, edited by Harry Francis Mallgrave. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016. https://doi.org/10.1002/9781118887226.

Canadian Construction Documents Committee. "CCDC 2: 2008 - Stipulated Price Contract." Canadian Construction Documents Committee, 2008.

Carpo, Mario. "Building with Geometry, Drawing with Numbers." In *When Is the Digital in Architecture?*, edited by Andrew Goodhouse and Canadian Centre for Architecture, 35–44. Montréal: Canadian Center for Architecture, 2017.

———. "Drawing with Numbers: Geometry and Numeracy in Early Modern Architectural Design." *Journal of the Society of Architectural Historians* 62, no. 4 (2003): 448–69. https://doi.org/10.2307/3592497.

International Standards Organization. "ISO 129-1:2004 Technical Drawings — Indication of Dimensions and Tolerances." International Standards Organization, n.d.

Pérez Gómez, Alberto, and Louise Pelletier. *Architectural Representation and the Perspective Hinge*. Cambridge, Mass: MIT Press, 1997.

Sarton, George. "The First Explanation of Decimal Fractions and Measures (1585). Together with a History of the Decimal Idea and a Facsimile (No. XVII) of Stevin's Disme." *Isis* 23, no. 1 (1935): 153–244. http://www.jstor.org/stable/225223.

Relevant Software

Associates, McNeel &. *RhinoCommon Is the .NET SDK for Rhino5 / Grasshopper: Mcneel/Rhinocommon*. 2010. Reprint, Robert McNeel & Associates, 2019. https://github.com/mcneel/rhinocommon.

Autodesk. "About Element Behavior in Revit | Revit Products 2018 | Autodesk Knowledge Network." Accessed April 1, 2019. https://knowledge.autodesk.com/support/revit-products/getting-started/caas/CloudHelp/cloudhelp/2018/ENU/Revit-GetStarted/files/GUID-5BFA499A-5ACA-4069-852C-9B60C9DE6708-htm.html.

———. "About Family Visibility and Detail Level | Revit Products 2018 | Autodesk Knowledge Network." Accessed April 1, 2019. https://knowledge.autodesk.com/support/revit-products/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Revit-Customize/files/GUID-B28F98C2-D3DA-486E-B198-96B8C862F28B-htm.html.

———. "Help: What Can You Do with the Revit Platform API?" Accessed April 4, 2019. http://help.autodesk.com/view/RVT/2019/ENU/?guid=Revit_API_Revit_API_Developers_Guide_Introduction_Getting_Started_Welcome_to_the_Revit_Platform_API_What_Can_You_Do_with_the_Revit_Platform_API_html.

———. "Understanding Revit Terms | Revit Products | Autodesk Knowledge Network." Accessed April 1, 2019. https://knowledge.autodesk.com/support/revit-products/getting-started/caas/CloudHelp/cloudhelp/2014/ENU/Revit/files/GUID-2480CA33-C0B9-46FD-9BDD-FDE75B513727-htm.html.

CAD, BRL. "Mged - BRL-CAD." Accessed July 26, 2019. https://brlcad.org/wiki/Mged.

McNeel. "Cycles| Rhino 3-D Modeling." Accessed April 4, 2019. https://docs.mcneel.com/rhino/6/help/en-us/options/cycles.htm.

Trimble. "SketchUp Ruby API — SketchUp Ruby API Documentation." Accessed April 4, 2019. http://ruby.sketchup.com/.

Blender

5agado. “Blender 2.8 Grease Pencil Scripting and Generative Art.” Medium. Accessed July 31, 2019. https://towardsdatascience.com/blender-2-8-grease-pencil-scripting-and-generative-art-cbbfd3967590.

“AZR Studio – Visualisaties, Walkthrougs & 3D Modellen.” AZR Studio. Accessed August 13, 2019. https://azrstudio.nl/?lang=en/.

“Deckor | International 3D Rendering Studio.” Accessed August 13, 2019. https://www.deckor.co/.

Dounas, Theodoros, and Alexandros Sigalas. “Blender, an Open Source Design Tool: Advances and Integration in the Architectural Production Pipeline,” 2009, 8.

Dupont de Dinechin, Matthieu. “Blender for Architecture: It’s Much More Than Viz.” Confrence Presentation presented at the Blender Conference, November 7, 2017. https://www.youtube.com/watch?v=cSiKSrQ9eOI.

Dupont de Dinechin: Matthieu. “Blender as a Design Tool for Architecture – LGM 2014.” Confrence Presentation presented at the Libre Graphics Meeting, August 11, 2014. https://www.youtube.com/watch?v=1kxViKJ-ny4.

Foundation, Blender. “Asset Manager.” Blender Developers Blog. Accessed April 13, 2020. https://code.blender.org/2020/03/asset-manager/.

———. “Blender 2.8 Design Document.” Blender Developers Blog. Accessed August 9, 2019. https://code.blender.org/2017/10/blender-2-8-design-document/.

———. “Blender Development Fund.” Accessed July 31, 2019. https://fund.blender.org/.

———. “Blenders History.” blender.org. Accessed February 27, 2019. https://www.blender.org/foundation/history/.

———. “Eevee Roadmap.” Blender Developers Blog. Accessed August 9, 2019. https://code.blender.org/2017/03/eevee-roadmap/.

———. “Viewport Project – Plan of Action.” Blender Developers Blog. Accessed August 9, 2019. https://code.blender.org/2016/12/viewport-project-plan-of-action/.

Giachino, Alberto. “Parametric Panels with Sverchok (1).” CodePlastic, April 24, 2018. http://www.codeplastic.com/2018/04/24/parametric-panels-with-sverchok-1/.

JPfep. “Blender AddOSC Addon.” Accessed August 17, 2019. http://www.jpfep.net/pages/addosc/.

O’Riordan, Fin, and Pablo Vazquez. “Fin. On Twitter: ”@PabloVazquez_ Soooooo..... More Like This Then? https://t.co/WcHnuNplBd” / Twitter.” Twitter. Accessed July 29, 2019. https://twitter.com/FinEskimo/status/1097551783327645699.

Reynish, William. “The Evolution of Blenders User Interface,” n.d., 25.

Roosendaal, Ton, and Blender Foundation. “Blender Development Fund, Projects for First Half 2019.” Blender Developers Blog, December 24, 2018. https://code.blender.org/2018/12/blender-development-fund-projects-for-first-half-2019/.

———. “Development Fund Report, July 2019.” Blender Developers Blog, July 20, 2019. https://code.blender.org/2019/07/development-fund-report-july-2019/.

Open Source & Craft

Bradley, Dale A. "The Divergent Anarcho-Utopian Discourses of the Open Source Software Movement." *Canadian Journal of Communication* 30, no. 4 (January 10, 2006). https://doi.org/10.22230/cjc.2005v30n4a1642.

Burley, Brent, and Walt Disney Animation Studios. "Physically-Based Shading at Disney," n.d., 26. https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf.

DeRose, Tony, Michael Kass, and Tien Truong. "Subdivision Surfaces in Character Animation." In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '98*, 85–94. Not Known: ACM Press, 1998. https://doi.org/10.1145/280814.280826.

Raymond, Eric S. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. 1st ed. Beijing ; Cambridge, Mass: O'Reilly, 1999.

Sennett, Richard. *The Craftsman*. New Haven: Yale Univ. Press, 2008.

Werner, Stefan. "Developer.Blender.Org Stefan Werner's Account History." Accessed October 3, 2019. https://developer.blender.org/p/swerner/.

Participatory Design

Alexander, Christopher, ed. *The Oregon Experiment*. New York: Oxford University Press, 1975.

Bryant, Greg. "The Oregon Experiment After Twenty Years." *RAIN*, 1991. http://www.rainmagazine.com/archive/1991-1/the-oregon-experiment-revisited.

De Carlo, Giancarlo. "Architecture's Public." In *Architecture and Participation*, Digit. print., 3–22. London: Taylor & Francis, 2009.

PARTISANS. *Rise and Sprawl: The Condominiumization of Toronto*. Edited by Hans Ibelings and Nicola Spunt. Montreal Amsterdam: The Architecture Observer, 2016.

Rittel, Horst W. J., and Melvin M. Webber. "Dilemmas in a General Theory of Planning." *Policy Sciences* 4, no. 2 (June 1, 1973): 155–69. https://doi.org/10.1007/BF01405730.

Open Source Architecture

Ratti, Carlo. “Open Source Architecture (OSArc).” Domus, June 15, 2011. https://www.domusweb.it/en/opinion/2011/06/15/open-source-architecture-osarc-.html.

Ratti, Carlo, and Matthew Claudel. *Open Source Architecture*. New York, New York: Thames & Hudson, 2015.

Terzidis, Kostas, and Jan Jungclaus. “Predicting the Future: Open Source CAAD?” n.d., 6.

Vardouli, Theodora, and Leah Buechley. “Open Source Architecture: An Exploration of Source Code and Access in Architectural Design.” *Leonardo* 47, no. 1 (August 3, 2012): 51–55. https://doi.org/10.1162/LEON_a_00470.

WikiHouse

Architecture for the People by the People, 2013. https://www.ted.com/talks/alastair_parvin_architecture_for_the_people_by_the_people.

Edward, David F. A., and Marialena Nikolopoulou. "Building Open-Source to What Extent Does WikiHouse Apply the Open-Source Model to Architecture?" University of Kent, 2018. https://static1.squarespace.com/static/5392f715e4b032d797fc94ed/t/5b00a0ce6d2a734d9cb89662/1526767832775/BuildingOpenSource_DEdward.pdf.

Lab, Open Systems. "WikiHouse." Accessed August 21, 2019. https://www.wikihouse.cc/projects.

Parvin, Alastair. "Scaling the Citizen Sector," 28. United Kingdom: Community Land Trust Network. Accessed September 30, 2018. https://medium.com/@AlastairParvin/scaling-the-citizen-sector-20a20dbb7a4c.

Prest, Clayton. "WikiHouse-Contributors / Software Dev." WikiHouse-Contributors, May 16, 2018. https://wikihousecontributors.slack.com/archives/C15DW4XBK/p1526490467000617.

———. "Wikihouseproject / Microhouse Issue #6," January 25, 2017. https://github.com/wikihouseproject/Microhouse/issues/6#issuecomment-275063114.

IFC & BIM Standards

buildingSMART. "IFC Overview Summary — Welcome to buildingSMART-Tech.Org." Accessed February 1, 2019. http://www.buildingsmart-tech.org/specifications/ifc-overview.

IfcOpenShell. "IfcOpenShell." Accessed October 7, 2019. http://ifcopenshell.org/.

Limited, BSI Standards. "PAS 1192-2:2013," n.d. http://www.hfms.org.hu/web/images/stories/PAS/PAS1192-2-BIM.pdf.

Moult, Dion. "Blender IFC Exporter." GitHub. Accessed October 7, 2019. https://github.com/IfcOpenShell/IfcOpenShell.

Succar, Bilal. "Building Information Modelling Maturity Matrix." *Concepts and Technologies*, 65–103. Accessed August 18, 2019. https://www.academia.edu/186259/Building_Information_Modelling_Maturity_Matrix.

MeasureIt-ARCH Resources and Publications

Cress, Kevan. “MeasureIt-ARCH Architectural Dimensions for Blender 2.8 (in Development) — Blender.Community,” March 24, 2019. https://blender.community/c/today/3Mcbbc/.

———. “MeasureIt-ARCH Commit Logs,” n.d. https://github.com/kevancress/MeasureIt-ARCH/commits/master.

———. “MeasureIt-ARCH Issues,” n.d. https://github.com/kevancress/MeasureIt-ARCH/issues?utf8=%E2%9C%93&q=.

———. “MeasureIt-Arch Project Board,” n.d. https://github.com/kevancress/MeasureIt-ARCH/projects/2.

———. *Per Camera Resolution*, 2018. https://github.com/kevancress/per_camera_resolution.

Cress, Kevan and Beesley. “Architectural Design in Open-Source Software - Developing MeasureIt-ARCH, an Open Source Tool to Create Dimensioned and Annotated Architectural Drawings Within the Blender 3D Creation Suite.” In *Sousa, JP, Xavier, JP and Castro Henriques, G (Eds.), Architecture in the Age of the 4th Industrial Revolution - Proceedings of the 37th eCAADe and 23rd SIGraDi Conference - Volume 1, University of Porto, Porto, Portugal, 11-13 September 2019, Pp. 621-630*. CUMINCAD, 2019. http://papers.cumincad.org/cgi-bin/works/paper/ecaadesigradi2019_561.

MeasureIt-ARCH Introduction - YouTube, 2019. https://www.youtube.com/watch?v=QL_ArANpsVU&t=90s.

MeasureIt-ARCH Rough Demo Timelapse - YouTube, 2019. https://www.youtube.com/watch?v=lHI78SDB8bs.

MeasureIt-ARCH Version 0.3 Update - YouTube, 2019. https://www.youtube.com/watch?v=MWo87QvcEPk&t=2s.

Moult, Dion. *MeasureIt-ARCH Features & Suggestions - Comment, Dion Moult*, 2019. https://github.com/kevancress/MeasureIt-ARCH/issues/4#issuecomment-536822024.

Vazquez, Antonio. *Blender MeasureIt Addon*. Accessed January 31, 2019. https://github.com/Antonioya/blender.

Vazquez, Antonio, and Kevan Cress. *MeasureIt-ARCH*, 2018. https://github.com/kevancress/MeasureIt-ARCH.

Line Drawing & Dimension Placement

Appel, Arthur. "The Notion of Quantitative Invisibility and the Machine Rendering of Solids." In *Proceedings of the 1967 22nd National Conference on* -, 387–93. Washington, D.C., United States: ACM Press, 1967. https://doi.org/10.1145/800196.806007.

Bénard, Pierre, and Aaron Hertzmann. "Line Drawings from 3D Models: A Tutorial." *Foundations and Trends® in Computer Graphics and Vision* 11, nos. 1-2 (2019): 1–159. https://doi.org/10.1561/0600000075.

Chan, Eric, and Frédo Durand. "Chapter 22. Fast Prefiltered Lines." In *GPU Gems*. NVIDIA. Accessed December 15, 2018. https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter22.html.

Cole, Forrester, and Adam Finkelstein. "Two Fast Methods for High-Quality Line Visibility." *IEEE Transactions on Visualization and Computer Graphics* 16, no. 5 (September 2010): 707–17. https://doi.org/10.1109/TVCG.2009.102.

DesLauriers, Matt. "Drawing Lines Is Hard." Matt DesLauriers, March 9, 2015. https://mattdesl.svbtle.com/drawing-lines-is-hard.

Eiserloh, Squirrel. "Math for Game Programmers: Understanding Homogeneous Coordinates - YouTube." Confrence Presentation presented at the Game Developers Conference, Moscone Center - San Francisco, 2015. https://www.youtube.com/watch?v=o1n02xKP138.

Gooch, Bruce, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. "Interactive Technical Illustration." In *Proceedings of the 1999 Symposium on Interactive 3D Graphics - SI3D '99*, 31–38. Atlanta, Georgia, United States: ACM Press, 1999. https://doi.org/10.1145/300523.300526.

Grabli, Stéphane, Emmanuel Turquin, Frédo Durand, and François X. Sillion. "Programmable Rendering of Line Drawing from 3D Scenes." *ACM Transactions on Graphics* 29, no. 2 (March 1, 2010): 1–20. https://doi.org/10.1145/1731047.1731056.

Junya Christopher Motomura. "GuiltyGear Xrd's Art Style: The X Factor Between 2D and 3D," 24. Moscone Center - San Francisco, 2015. http://www.ggxrd.com/Motomura_Junya_GuiltyGearXrd.pdf.

McGuire, Morgan, and John F. Hughes. "Hardware-Determined Feature Edges." In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering - NPAR '04*, 35. Annecy,

France: ACM Press, 2004. https://doi.org/10.1145/987657.987663.

Thalmann, Nadia Magnenat, and Daniel Thalmann. *Computer Animation: Theory and Practice*. Tokyo: Springer Japan, 1990. http://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=3102178.

Vollick, Ian, Daniel Vogel, Maneesh Agrawala, and Aaron Hertzmann. "Specifying Label Layout Styles by Example," n.d., 10.

Appendices

MeasureIt - ARCH Code Overview

The following is a brief outline of the file structure of MeasureIt-ARCH's code base, describing where key elements of MeasureIt-ARCH's functionality can be found within it;

init.py

The init file contains all code relevant to the initialization of the add-on and handles the registering and unregistering of all classes. The init file also defines some of MeasureIt-ARCH's scene level properties.

auto_load.py

A set of utilities written by Jaques Lucke to automatically register and unregister an add-ons classes without having to manually add them to a list in the init file. These methods are called from the init file.

measureit_arch_baseclass.py

measureit_arch_baseclass.py contains classes which define the base Property Groups that all MeasureIt-ARCH elements inherit from. This file also contains the common operators for the deletion of MeasureIt-ARCH elements.

measureit_arch_main.py

measureit_arch_main.py contains code relating to the core operations of MeasureIt-ARCH. Notable sections include:

- The save and load handlers responsible for cleaning MeasureIt-ARCH data
- MeasureIt-ARCH's main operator UI panel.
- MeasureIt-ARCH's settings UI panel
- The Operator to enable and disable MeasureIt-ARCH's draw handlers (The Show/Hide Button)
- The main draw loops for both 2D (text) and 3D elements
- The Post View and Post Pixel draw callback's
- Several utility methods that assist with interpreting user selection.

measureit_arch_geometry.py

measureit_arch_geometry.py contains the draw methods for each element type, as well as the select_normal() method which handles the calculation of MeasureIt-ARCH dimension's placement. Each of the draw methods defined in measureit_arch_geometry.py are called from the main draw loops in measureit_arch_main.py.

measureit_arch_*element type*.py

measureit_arch_dimensions.py, measureit_arch_annotations.py, and measureit_arch_lines.py each contain all of the unique Property Definitions, Operators, and UI panels for their respective MeasureIt-ARCH element type.

measureit_arch_styles.py

Contains all code related to the creation of MeasureIt-ARCH styles and their UI.

measureit_arch_render.py

Contains all code to necessary to draw MeasureIt-ARCH elements to a rendered frame, or series of rendered frames in the case of an animation. This involves pre-drawing the scene to the rendered images depth buffer to be available for depth testing, setting flags to ensure that the draw methods use the camera position rather than the users viewpoint for operations like view segmentation, and writing the rendered image to an image ID block within Blender so it can be accessed by the user.

measureit_arch_gizmos.py

Contains the work in progress code for Measureit-ARCH's Gizmo System.

shaders.py

Contains all the GLSL shaders used by MeasureIt-ARCH's drawing system, wrapped in python classes so they can be accessed by Blender's Python API.

MeasureIt - ARCH v0.4 Documentation

INSTALLATION

Add-on Installation

1. Download “MeasureIt-ARCH_v03.zip”
2. Open Blender 2.8 (you can get the latest build here)
3. Open the Add-on Preferences (Edit -> Preferences -> Add-ons) and click install.

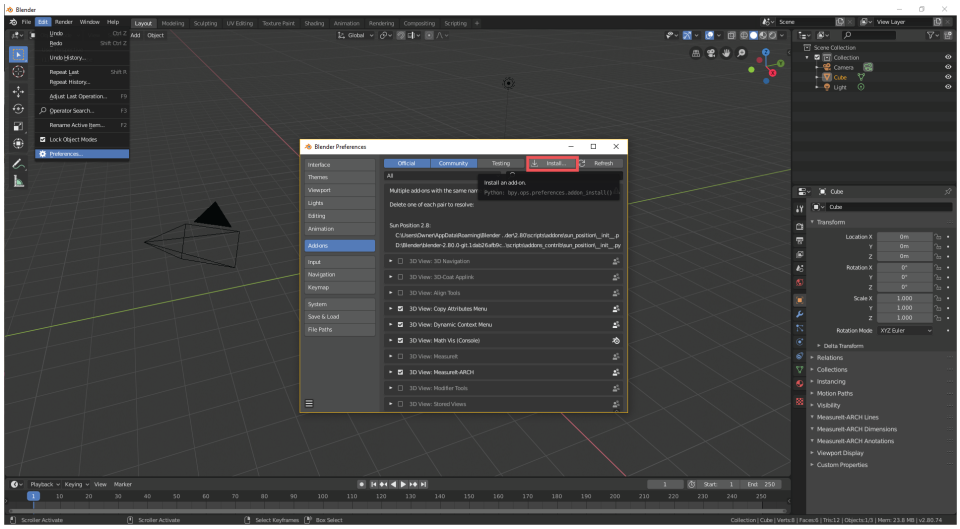


Figure 97: *Installing MeasureIt-ARCH*

4. Navigate to and double click on “MeasureIt-ARCH_v03.zip”
5. Click the Checkbox to enable the Add-on

FEATURES & USER INTERFACE

Main Tool Panel

The main tool panel is where you can add MeasureIt-ARCH Items to your 3D scene. This toolbar is located in the **3D Viewport Editor** and can be accessed by **pressing the "n" key**

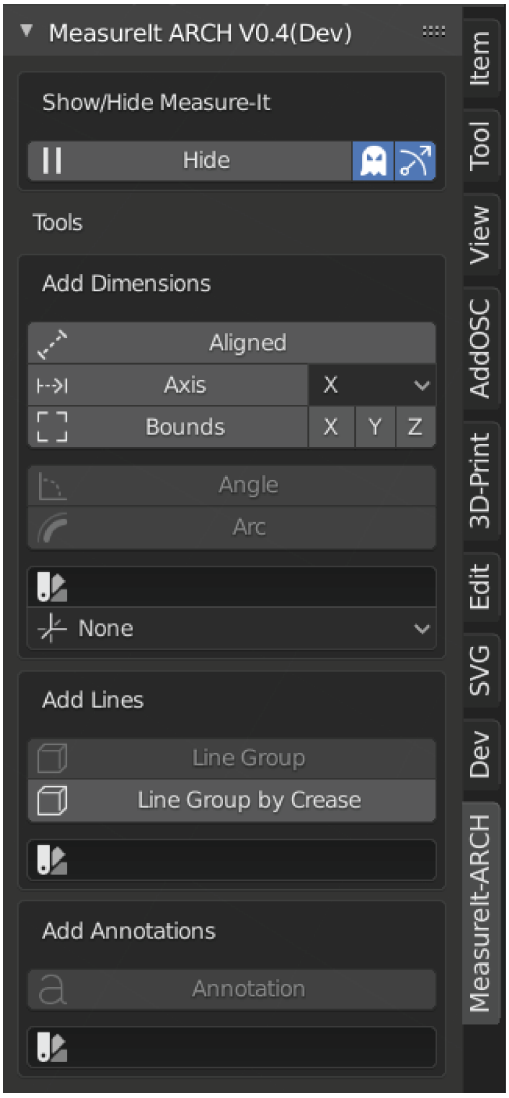


Figure 98: *MeasureIt-ARCH Main Tool Panel*

Show / Hide MeasureIt Toggle

- Shows and hides all items created by MeasureIt-ARCH

Selected Object Only Toggle (Ghost Icon)

- When disabled, MeasureIt-ARCH will only show items belonging to currently selected objects

Show Gizmo's (Arrow Icon)

- When enabled MeasureIt-ARCH will show gizmo's for the selected object.

Add Dimensions

Aligned

- Adds an Aligned Dimension between 2 Objects or Vertices
  - Object Mode: Select two objects and then press the Aligned Button
  - Edit Mode: Select two or more Vertices and press the Aligned Button

Axis

- Adds a Dimension that measures along a single Axis between 2 Objects or Vertices
  - Object Mode: Select two objects and then press the Aligned Button
  - Edit Mode: Select two or more Vertices and press the Aligned Button
- **Axis Selection** Lets you pick the axis to be measured on creation

Bounds (Object Mode Only)

- Adds a set of dimensions that measure the Bounding Box of the selected object
- **Axis Selection** Lets you pick the axes to be displayed on creation

Angle (Edit Mode Only)

- Adds an angle dimension for 3 selected vertices.
  - The 2nd vertex selected defines the corner of the angle.

Arc (Edit Mode Only)

- Adds an arc dimension circumscribing the 3 selected vertices.

Dimension Style (Color Swatch Icon)

- Lets you select a style to be assigned to new dimension on creation.

View Plane (Axis Icon)

- Lets you select the preferred view plain for new dimensions (used to automatically place dimensions on creation)
  - **XY Plane (Plan View):** Dimensions placed to be viewed from the top or bottom
  - **YZ Plane (Section/ Elevation View):** Dimensions placed to be viewed from the left or right
  - **XZ Plane (Section/ Elevation View):** Dimensions placed to be viewed from the front or back
  - **None:** Dimensions placement will be based on the angles of the adjacent surfaces.

Add Lines

Line Group (Edit Mode Only)

- Creates a Line Group from selected edges. Select the desired edges in edit mode and press the Line button.

Line Group by Crease (Object Mode Only)

- Creates a Line Group from any edges sharper than the specified crease angle.

Line Style (Color Swatch Icon)

- Lets you select a style to be assigned to new Line Group on creation.
  - **Note** This option will not be visible if you have not created any styles in your scene.

Add Annotations

Annotation

- Adds an Annotation to the selected Object or Vertex.

Annotation Style (Color Swatch Icon)

- Lets you select a style to be assigned to new Annotation on creation.
  - **Note** This option will not be visible if you have not created any styles in your scene.

Scene Settings

MeasureIt-ARCH Styles & Settings can be found in the Scene Tab of the Properties Editor.

MeasureIt-ARCH Styles

Styles have a nearly identical user interface to their corresponding items. Style-able properties can be found in the item's settings. Some settings, like an Annotations Offset, or a Dimensions Distance are still set per item, even when using a style.

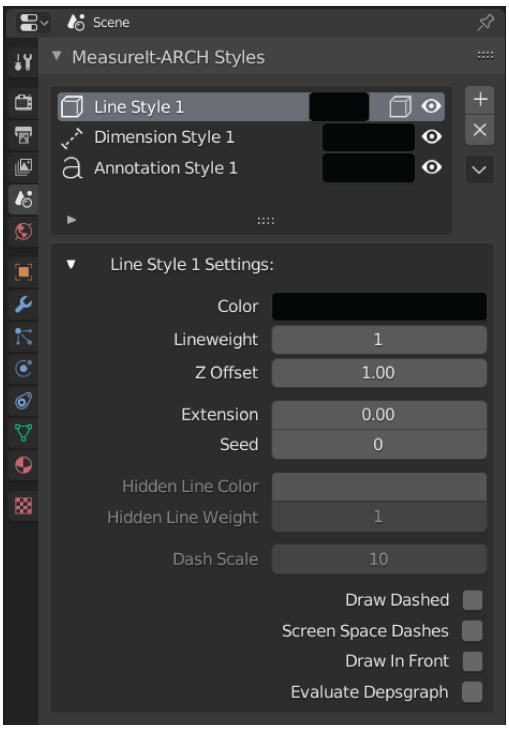


Figure 99: *Style Settings*

MeasureIt-ARCH Unit Settings

MeasureIt-ARCH unit setting can be found in Blender's Scene Settings under the Units panel. MeasureIt-ARCH makes use of Blender's Scene Unit System.

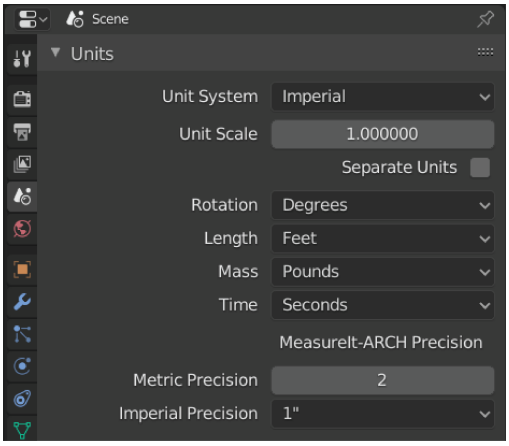


Figure 100: *Unit Settings*

Metric Precision

- Defines the number of decimal places included in your dimensions when using the Metric Unit System

Imperial Precision

- Fractional Precision to be used when using the Imperial Unit System

MeasureIt-ARCH Settings

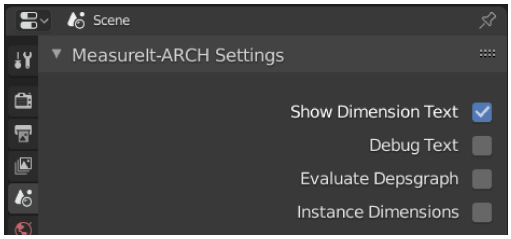


Figure 101: *Scene Settings*

Show Dimension Text

- Show or hide the text on dimension and annotation elements

Debug Text

- Writes Dimension Text to an image for Debug purposes

Evaluate Depsgraph

- Evaluate Blender's Dependency Graph before drawing MeasureIt-ARCH elements. By default, MeasureIt-ARCH does not evaluate the Dependency Graph as some generative modifiers can give unpredictable results. Enabling this setting will make MeasureIt-ARCH attempt to evaluate these modifiers, in its calculations.

Instance Dimensions

- Will Enable Dimension Instancing.
- **Note:** Text on instanced Dimensions will not adjust to the changes in the instances scale or rotation.

Object Settings

Dimension, Annotation, and Line Group settings can be found in Object Tab of the Properties Editor. (**Note:** These panels will only appear if the active object has a dimension, annotation or line group associated with it. Too add dimensions, annotations or line groups use the main tool panel)

Dimensions

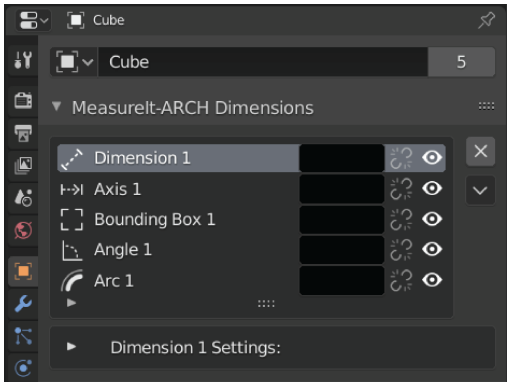


Figure 102: *Dimension List*

Color

- Sets Dimension Color

Link Style (Link or Broken Link Icon)

- Toggles if this Dimension uses a Style

Visibility (Eye Icon)

- Toggles the Dimensions visibility

Delete (x Icon)

- Deletes the Dimension

Dimension Settings

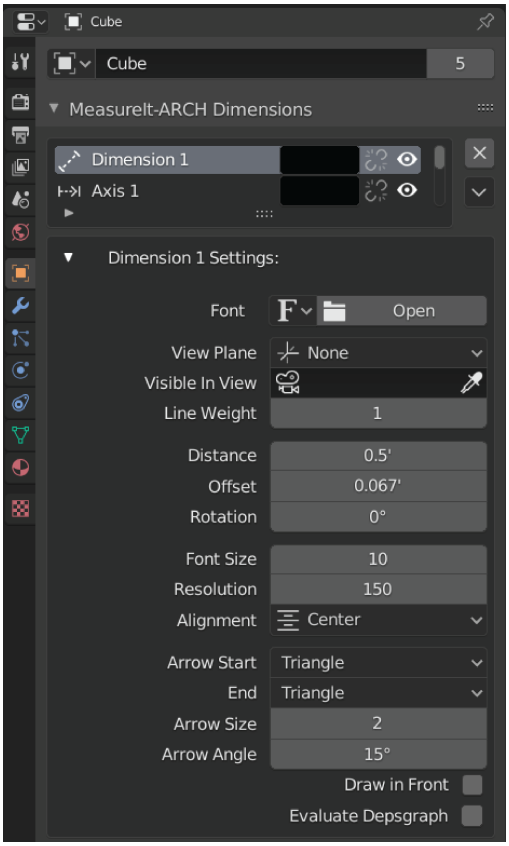


Figure 103: *Dimension Settings*

Font

- Lets you select a custom font for the Dimension.

View Plane

- Set the preferred view plain for this Dimension
  - **XY Plane (Plan View):** Dimensions placed to be viewed from the top or bottom
  - **YZ Plane (Section/ Elevation View):** Dimensions placed to be viewed from the left or right
  - **XZ Plane (Section/ Elevation View):** Dimensions placed to be viewed from the front or back
  - **None:** Dimensions placement will be based on the angles of the adjacent surfaces.

Measurement Axis (Axis & Bounds Dimensions Only)

- Select the Axis to Measure

Visible In View

- Limit the Dimensions visibility to a specific Camera in your scene.
  - If no Camera is selected the Dimension will be visible in all Cameras
  - If a Camera is selected the Dimension will only be visible when that Camera is the Active Camera

Line Weight

- The Dimensions Line Weight

Distance

- The Distance of the Dimension Text from the Objects or Vertices it's attached to.

Radius (Arc and Angle Dimensions Only)

- The Distance of the Dimension Text from the center of the Arc or Angle.

Offset

- The offset distance from the ends of the Dimension line to the Vertex or Object it's attached to

Rotation

- Rotates the Dimension around the axis of its measurement

Font Size

- The Dimension font size.

Resolution

- The Dimension font resolution

Alignment

- The Dimension text alignment relative to the dimension line (Left, Center, Right)

Arrow Start & End

- Set the style of the dimension terminations

Arrow Size

- Set the size of the dimension terminations

Draw In Front

- Makes this element Ignore Visibility tests.

Evaluate Depsgraph

- Evaluate Blender's Dependency Graph before drawing this MeasureIt-ARCH element.

Line Groups

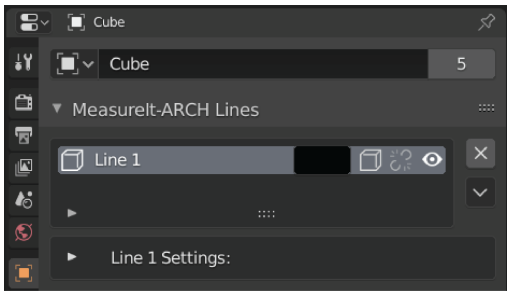


Figure 104: *Line Group List*

Color

- Sets Line Group Color

Draw Hidden Lines (Cube with Dashed Lines Icon)

- This Line Group will draw hidden lines as dashed lines.

Link Style (Link or Broken Link Icon)

- Toggles if this Line Group uses a Style

Visibility (Eye Icon)

- Toggles the Line Groups visibility

Delete (x Icon)

- Deletes the Line Group

Line Group Menu (Chevron Icon)

Add to Line Group (Edit Mode Only)

- Adds selected Edges to this Line Group

Remove from Line Group (Edit Mode Only)

- Removes selected Edges from this Line Group

Line Group Settings

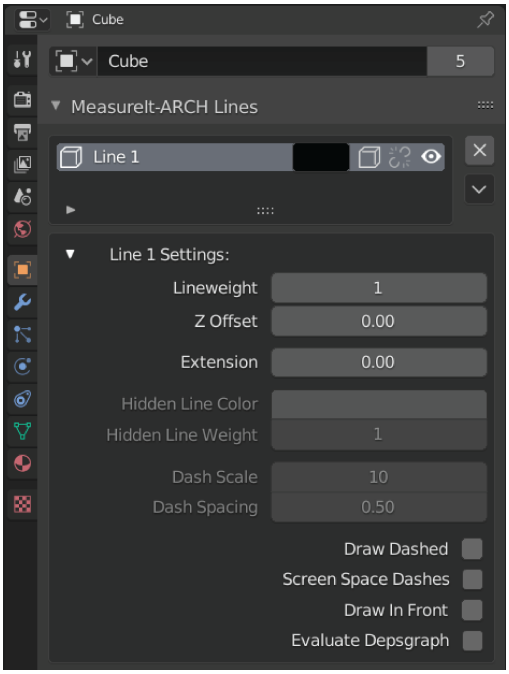


Figure 105: *Line Group Settings*

Line Weight

- Set the Line Groups line weight

Z Offset

- Tweaks the Line Groups Distance from the screen in Clip Space. Higher values move the Lines closer to the screen.
  - This is useful for adjusting Line Groups that don't appear to be drawing correctly (Jagged Edges etc.)
  - Making this value negative allows for the drawing of silhouettes. Higher values will move lines further backwards

Extension

- Adds a slight over-extension to each line segment in this Line Group

Hidden Line Color (Only Available if Draw Hidden Lines is Enabled)

- Sets the color of hidden lines

Hidden Line Weight (Only Available if Draw Hidden Lines is Enabled)

- Sets the line weight of hidden lines

Dash Scale (Only Available if Draw Hidden Lines or Draw Dashed is Enabled)

- Changes the dash size of dashed lines. Larger values make smaller dashes

Dash Spacing (Only Available if Draw Hidden Lines or Draw Dashed is Enabled)

- Changes the dash spacing for dashed lines. 0.5 gives even spacing.

Draw Dashed

- Draws all lines in this Line Group as dashed lines, regardless of visibility

Screen Space Dashes

- Calculates Dash Spacing in Screen Space. Useful to achieve more even

dashes in still renders when some lines are nearly parallel to the view. Can cause dashes to appear to 'slide' along edges when used in animations.

Draw In Front

- Makes this element Ignore Visibility tests.

Evaluate Depsgraph

- Evaluate Blender's Dependency Graph before drawing this MeasureIt-ARCH element.

Annotations

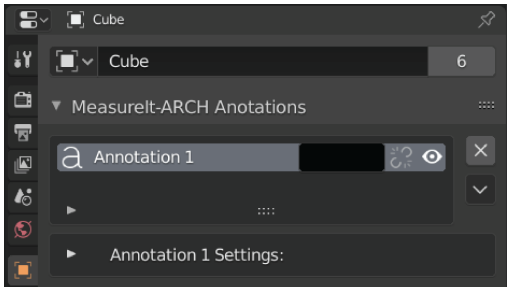


Figure 106: *Annotation List*

Color

- Sets Annotation Color

Link Style (Link or Broken Link Icon)

- Toggles if this Annotation uses a Style

Visibility (Eye Icon)

- Toggles the Annotations visibility

Delete (x Icon)

- Deletes the Annotation

Annotation Settings

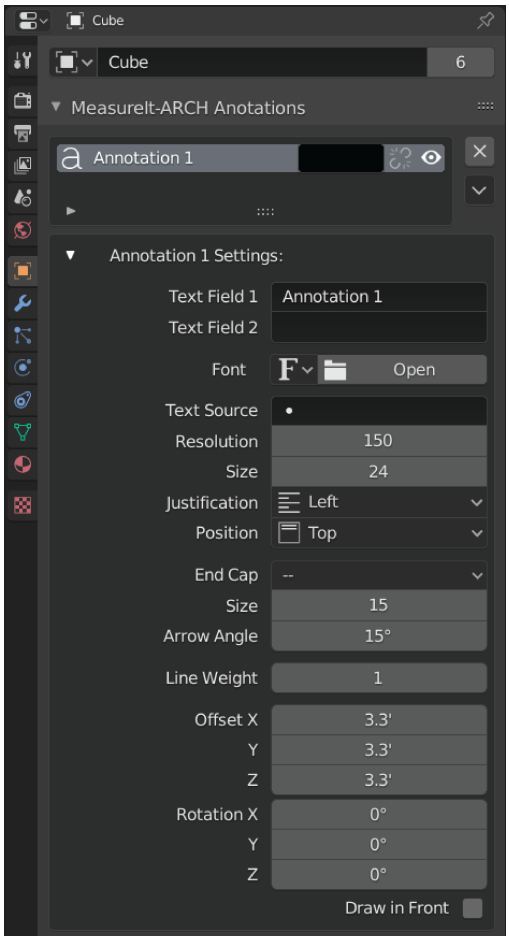


Figure 107: *Annotation Settings*

Text Field

- Sets the text for the annotation
- Annotations can have multiple text fields, each new text field will display as a new line in the Annotation Text

Font

- Lets you select a custom font for the Annotation from your system.

Text Source

- MeasureIt-ARCH can pull annotation text from an objects Custom Properties metadata. This field defines the source custom property.
- If two text fields are available, MeasureIt-ARCH will use the first to display the custom properties name, and the second to display the value. If only one text field is available, only the value will be displayed.

Size

- The Annotation font size

Resolution

- The Annotation font resolution

Justification

- Text Justification relative to the end of the Annotation leader line (Left, Center, Right)

Position

- Text Position relative to the end of the Annotation leader line (Top, Middle, Bottom)

Endcap

- **Dot** Adds a Circle to the end of the Annotation Leader
- **Triangle** Adds an Arrow to the end of the Annotation Leader

Endcap Size

- Sets the size of the Dimension Leader Endcap

Line Weight

- Line Weight of the Annotation leader

Offset

- The XYZ offset from the object or vertex that the annotation is attached to.

Rotation

- The XYZ rotation of the annotation text

Draw In Front

- Makes this element Ignore Visibility tests.

Rendering

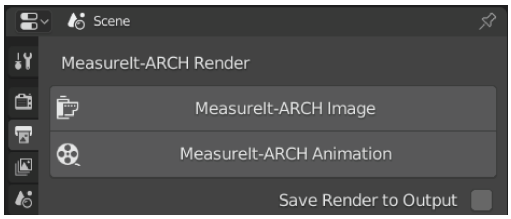


Figure 108: *Render Buttons*

MeasureIt-ARCH Render Settings can be found in the Render Panel of the Properties Editor. Currently this renders all MeasureIt-ARCH items to an image file which can be layered over Blender's render in the compositor.

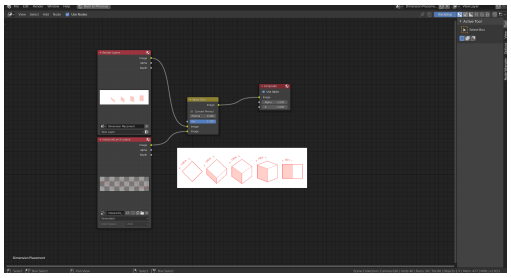


Figure 109: *MeasureIt-ARCH Compositing Setup*

MeasureIt-ARCH Image -Renders a Still Image - *Note:* If 'Save Render to Output' is not enabled the rendered image will only be stored in an image data-block within Blender.

MeasureIt-ARCH Animation

- Renders the full frame range of the current scene
- Animation Renders can be Cancelled with the Esc key, or by Right Clicking in the 3D View
- *Note:* A 3D Viewport window must be open for MeasureIt-ARCH to render animations
- *Note:* Animation frames will be saved to the Output path defined in the Render Panel

Save Render to Output

- Saves Still Image renders to the Output path defined in the Render Panel after rendering

Glossary of Terms

General Terms

Add-on A supplementary piece of software that adds functionality to an existing software.

AEC Architecture Engineering and Construction. The acronym commonly used to refer to the multi-disciplinary industry responsible for the creation of our built environment.

API Application Programming Interface. An API is a library of functions that allow for external access to the functions of a piece of software. A software's API allows for the programming of external add-ons that interface with the software's core functionality.

BIM Building Information Modeling

CAD Computer Aided Design

Git A system for managing version control of a software's development.

Git Commit In the context of this thesis, a Commit refers to a particular state in the software's git version control system. Each commit represents the set of changes made to the software since the last commit.

GitHub An online platform for the hosting of projects managed with the Git version control system.

GPU The Graphics Processing Unit, a specialized piece of computer hardware optimized for the types of calculations necessary to draw images and graphics.

Manifold Geometry In the context of digital 3D models, manifold geometry can be thought of as objects that are continuous and closed. Cubes, Torus's and Spheres are all examples of the 3D manifold geometries. More precisely every edge of a manifold geometry in 3D is adjacent to exactly 2 faces.

Method A method refers to a named block of code within a piece of software that runs a particular procedure. Methods can be called multiple times from various locations within the code, and help to reduce the need for code duplication.

MeasureIt The add-on developed by Antonio Vazquez to add basic dimensioning functionality to Blender.

MeasureIt-ARCH The add-on developed for this Thesis which expands and improves on the functionality of MeasureIt

Normal A direction orthogonal (perpendicular) to a given plane.

OpenGL Open Graphics Library. An API originally developed by Silicon Graphics now managed by the Khronos Group. OpenGL facilitates cross-platform, programming language-independent, communication with the computers GPU for the drawing and rendering of 2D and 3D content.

Rhino A NURB's based 3D modelling and drafting application developed by Robert McNeel & Associates.

Revit A Building Information Modeling application common in the North American Architecture Engineering and Construction Industry. Developed by Autodesk

SketchUp A mesh-based 3D modelling application owned by Trimble Inc.

String A data type used in computer programming which stores a sequence of characters (letters).

Work Plane (C-Plane) A construct used in many Computer-Aided Design software packages. The Work Plane, or Construction Plane (C-plane), is a user-defined 2D plane used to assist in the placement of dimensions and other drawing elements.

Co-ordinate Spaces

Our discussion of MeasureIt-ARCH's development makes use of several unique Co-ordinate Spaces when describing two dimensional, three dimensional and in one case, four dimensional space. Each of these Co-ordinate Spaces is described briefly below.

Camera Space Camera Space is a three dimensional cartesian co-ordinate system where the Z axis is pointing directly away from the 'camera' or the observer of the 3D scene in their viewing direction. The X axis and Y axis in Camera Space can be thought of as the horizontal and vertical axes of the picture plane.

Clip Space Clip Space is a four dimensional homogeneous co-ordinate system. 4D homogeneous co-ordinates are used frequently in computer graphics applications as they allow common three dimensional geometric operations to be calculated through four dimensional matrix multiplication. For our purposes in this thesis, Clip Space is used to calculate and manipulate the perspective projection of objects in 3D space. In this case Clip Space is similar to Camera Space, with the Z axis pointing away from the observer in their viewing direction and the X axis and Y axis representing the picture plane. The fourth axis of Clip Space, W, can be thought of roughly as a measure of how much any point in Clip Space is distorted by a perspective projection when drawn to the screen.

Image Space Image Space is a two dimensional cartesian co-ordinate system where the X axis and Y axis represent the horizontal and vertical axis respectively, of a rendered image.

Screen Space Screen Space is a two dimensional cartesian co-ordinate system identical to image space, we use the term screen space when discussing an image that is rendered temporarily to the screen as part of a 3D application, rather than a rendered image that is created to be saved by the user.

Object Space Object Space is a three dimensional cartesian co-ordinate system where the X, Y and Z axes are aligned with the transformation of the object in question. If the object has not undergone any transformations (scaling, rotation, or translation) then its Object Space is identical to the World Space.

World Space World Space is a three dimensional cartesian co-ordinate system where the X, Y and Z axes are the 3 perpendicular basis vectors of the 3D space.

Dimension Terminology

The following terms are used to refer to the various components that make up Annotations and Dimensions

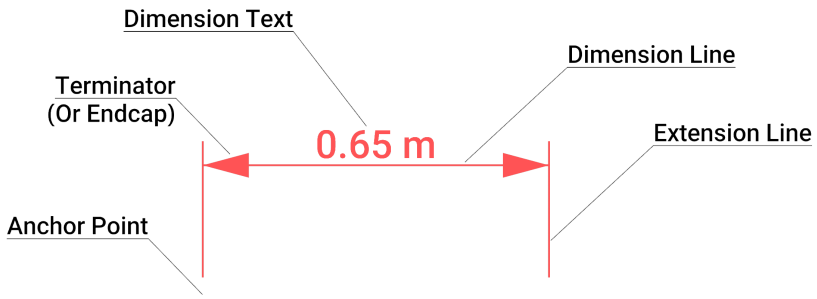


Figure 110: *Components of a Dimension*
Based on the terminology outlined in 'ISO 129-1:2004 Technical drawings — Indication of Dimensions and Tolerances'

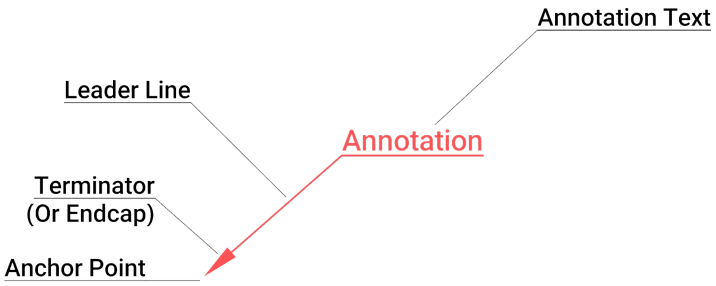


Figure 111: *Components of an Annotation*
Based on the terminology outline in 'ISO 129-1:2004 Technical drawings — Indication of Dimensions and Tolerances'