BIDIRECTIONAL SCRIPTING AND GEOMETRIC MANIPULATION

by

James William Rodgers Jr.

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Architecture III and
Master of Science in Information Technology

Charlotte

2016

Approved by:

_____
Dr. John Gero

_____
Professor Chris Beorkrem

_____
Professor Jefferson Ellinger

ABSTRACT

JAMES WILLIAM RODGERS JR. Bidirectional scripting and geometric manipulation. Under the direction of (DR. JOHN GERO, PROFESSOR CHRIS BEORKREM, and PROFESSOR JEFFERSON ELLINGER)

Working within the bounds of two-dimensional drawing programs and algorithmic design software, this project seeks to improve the accessibility to hidden information that is often obscured in conventional CAD software. Building off of visual programming tools like Grasshopper and Dynamo, a series of video prototypes have been constructed. This project proposes that a feasible and usable platform can be devised to take advantage of a bidirectional link between scripting and direct manipulation. This is demonstrated in the video prototypes by combining conventions from modern CAD software, and algorithmic scripting software. A heuristic evaluation is used to analyze the usability of the proposed software seen in the video prototypes.

TABLE OF CONTENTS

INTRODUCTION

In the early phases of schematic design, architects explore many ideas at a fast pace in order to arrive at a suitable solution, without immediately committing themselves to a particular design. Because of this, CAD is rarely used at this stage due to its level of precision being inconducive to sketching and iterative drawing. As stated by Gross et al. (2009) in *Computational Support for Sketching in Design: A Review* , sketching can be simply defined as "quickly made depictions that facilitate visual thinking," as they represent an opportunity to think through drawing and generate new ideas (p. 3). "Sketching is the traditional method for early-phase design when both problems and solutions are unclear." (p. 10)

Architecture students are often encouraged to sketch freehand rather than using CAD at early stages of design, usually because of the restrictions and influences that CAD programs place on designers. These influences arise from the distinct difference between the way in which a designer views a sketch, and the way the sketch data is stored in the computer. A designer may be able to interpret their sketch in infinite ways, while the computer has stored either a fixed or basic interpretation of the drawing. In conventional CAD software, the designer is presented with a straightforward representation of their drawing, as this medium is most similar to the way in which architects work with pencil and paper. This representation can be considered an abstraction of the underlying data structure, in order to make this data legible and easily manipulated. However, there is an opportunity to alter the designer's understanding of the drawing by presenting them with a less abstracted view of their work. This task has been

most notably addressed by programs like Grasshopper for Rhinoceros, a visual programming language that produces geometry from scripts of pre-made components.

Grasshopper is an effective method for facilitating visual thinking in designers, as it produces an entirely new way of interpreting data that would normally be viewed as straightforward geometry. However, Grasshopper fails to be an effective method for sketching, as creating Grasshopper scripts can be a cumbersome and time-consuming task. Also, creating geometry in Grasshopper is done entirely with scripting, limiting a designer to this singular (albeit novel) understanding of the drawing. Gross et al. also state that "proficiency in sketching goes beyond applying marks to the page - one must also interpret and re-interpret drawings in order to use them effectively."

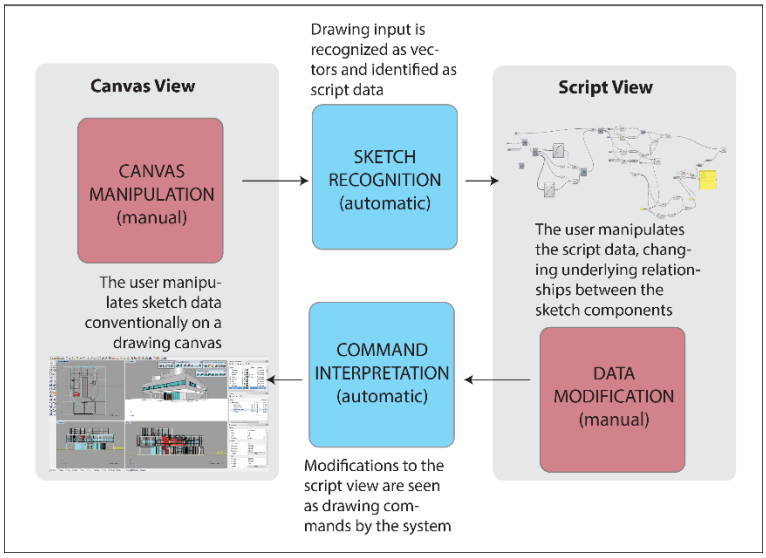

Figure 1: Bidirectional software diagram

If the process of scripting in grasshopper were reversed, it would allow a designer to sketch and manipulate a drawing, while their actions were recorded in script as the computer records them. By offering a simultaneous view of the script and canvas, a designer could achieve a better understanding of the hidden information, which is influencing their drawing. Finally, an opportunity to edit the automatically generated script could be offered, meaning that the designer could move seamlessly between one interpretation of the drawing to another.

This concept, combined with the natural advantages of iterative drawing on a computer (making copies quickly), could be a compelling way of offering visual thinking through “sketching” in a CAD program. Video prototypes for this software will be built to demonstrate the phenomenon that occurs when a designer sees and manipulates these two interpretations of their work. These videos will also show that a feasible and usable platform can be devised to take advantage of this bidirectional link between scripting and direct manipulation. These prototypes are made clear and understandable through combining and adapting affordances from modern CAD software and algorithmic scripting software. A heuristic evaluation will then be used to analyze the usability of the proposed software seen in the video prototypes.

LITERATURE REVIEW

Drawings and the design process

Drawings are used by architects throughout the design process, generally increasing in specificity and detail as the process moves forward. Early in the process and before many decisions are made, freehand sketches are often the tool of choice for many architects. Donald Schon (1983) speaks about how an architect in this phase "seeks to discover the particular features of his problematic situation, and from their gradual discovery, design an intervention." Schon also points out that there is "a problem in finding the problem" at these early stages of design. (p. 129) As Schon goes on to describe, this means that the designer must have a process to set problems and attempt to solve them, with failed attempts leading to reflection and a revised definition of the problem. By iterating on these attempts to solve the problem, designers learn from the series of moves they have completed, and this leads them to reappreciate, reinvent, and redraw. Describing the work of an example design student and her instructor, Schon states, "the graphic world of the sketchpad is the medium of reflection-in-action. Here they can draw and take their moves in spatial-action language, leaving traces which represent the forms of buildings on the site. Because the drawing reveals qualities and relations unimagined beforehand, moves can function as experiments." (p.157) Here it is made clear how Schon envisions drawings as a tool for exploring design and arriving at solutions for problems that are often difficult to define.

With the prototype software *Constraint Explorer*, Mark Gross (1985) sought to aid designers in defining problems through establishing a collection of constraints. In this project, constraints are defined as rules, requirements, relations, conventions, and

principles that define the context of a design problem. Gross also assumes that a design will begin with general specifications, and gradual reflection and iteration will lead to a set of more specific solutions. The proposed prototype allows a user to input constraints through command line input. These constraints are then used to establish a set of rules and relationships, which grow in complexity with each command from the user. Figure 2 shows an example of work produced with this prototype, with a simple drawing of a lintel and two columns becoming a complex system of relationships and constraints. Sometimes further input is not allowed by the system, as the new commands may conflict with those set previously by the user. This prototype represents a design tool which could aid designers in defining a problem through iterative additions, producing a network of constraints where originally there was little for the designer to work with.

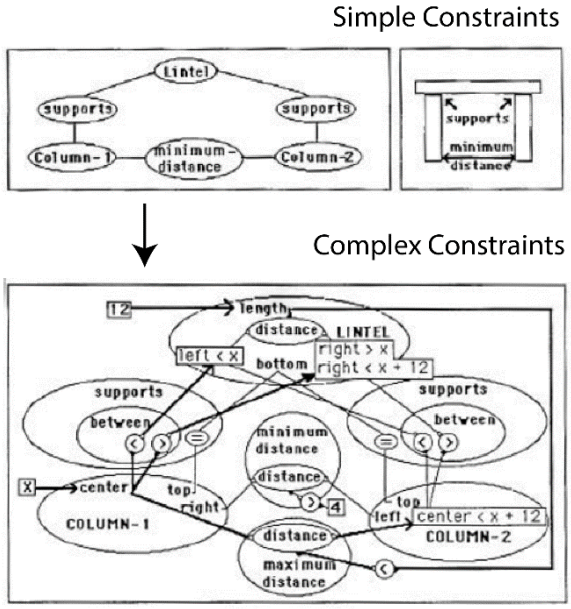

Figure 2: *Constraint Explorer* diagrams

Relating sketching to the activity of problem definition, Mark Gross et al. (2009) stated "Sketching is the traditional method for early-phase design when both problems and solutions are unclear." (p.10) Gross also offers a concise definition of sketching as "quickly made depictions that facilitate visual thinking" (p.3). These depictions may be rough in appearance, but play an important role in the thought process of a designer as problems are being defined. Belardi (2014) offers a similar definition: "sketching is a

quick, readily available, dense, self-generative, and, above all, extraordinarily communicative notational system." (p. 32) Belardi notes that part of the reason sketches are described as "quickly made" is that the materials for their creation are readily available, as pens, pencils and scraps of paper are common items used for sketching. Seeking to expand their definition of sketching, Gross et al. describe sketching as a tool for thinking and working through a design at its early stages, rather than a representational tool for clients or contractors. Robbins (1994) posited that sketches and drawings may be viewed as either instruments for design or representations of complete work, but on their own neither of these understandings is enough to define drawing comprehensively. Rather, he states that "drawing both produces architectural knowledge and is a production of that knowledge." (p. 5) This suggests that the process of completing a drawing includes simultaneous actions of learning and representation. Subsequently, the repetition of these two actions could produce and present a collection of new knowledge gained from the process.

Introducing another aspect of the sketching process, Gross (2009) stated that designers who are proficient with sketching must not only apply marks to a page, but iteratively interpret and re-interpret their sketches in order to arrive at new conclusions and to generate ideation. This means that through repeated sketching and reflection, gradually accumulated discoveries can lead to the design of a successful solution to the defined problem. Trace paper is a common and traditional tool for completing this task, as drawings can be layered on top of one another to achieve gradual changes through iteration. Belardi (1994) describes how the sensory input that occurs when a designer draws or traces creates knowledge or meaning where previously there was none. Belardi

cites a 1970s study by Gerald Edelman, which proposed that before the brain assigns meaning to a new form of stimulation or sensory input, it is assigned "a certain 'category,' a coherent response in the brain that is the antecedent of a 'meaning." (p. 20) This idea is supported by the work of Benjamin Libet, who showed that when a person traces something, their brain can take 500 milliseconds to process the mark consciously, while only 150 milliseconds pass before an unconscious sensory reception occurs. This was found to be particularly common when the drawing act was done impulsively, as would be expected when tracing over lines from a previous drawing. (p.24) The mysterious gap in between these two events is likened to a "black hole," and could be key for defining exactly how creative ideation occurs. Belardi also proposes how repeating this process of interpretation could improve a designer's creativity, stating "successive 'explorations'— i.e., sensations of the same subject in different times and contexts— are never the same, each category is determined and then reclassified an infinite number of times. It is through this endless process of structuring/ destructuring/ restructuring that each person creates his or her creative aptitude." (pp. 20-21) Speaking of the way various tools could effect consciousness during the drawing process, Peter Cook (2008) stated: "Is it not a spontaneous means of summarizing immediate attention? A form of jotting down. Shouting, murmuring, kicking or the wandering of the mind are less impeded by the necessary use of an implement, such as a pencil. The many effects on our consciousness of such implements has led to ceaseless pondering, whether it involves the impact of a lead pencil or the use of a particular computer program" (p. 8). The definitions of sketching put forth by Gross et al. (2009) were proposed in order to assist human-computer interaction practitioners in creating opportunities for this natural

sketching and drawing in a digital environment. The following literature reviews will seek to outline the challenges as well as the opportunities in completing this task.

The sketching gap in CAD software

Ivan Sutherland's introduction of *Sketchpad* (1963) represented one of the first examples of a digital implement which would have an impact on the way architects interact with their drawings. As the first example of a graphical user interface, the concepts demonstrated via the use of this software were entirely novel, needing to be explained thoroughly to viewers who had no previous conception of what a CAD program could or should resemble. A particularly important aspect of this software was in allowing the user to directly manipulate their drawings using a light-pen, and to visualize the drawing using an x-y point plotter display. In comparison to the command-line interfaces present in many early CAD systems, Sketchpad's GUI and direct manipulation offered a drawing tool that was much closer to traditional drawing methods.

While Sutherland's development of Sketchpad represented a leap forward in utilizing traditional drawing methods in a digital environment, modern CAD systems have been thoroughly developed and integrated in all phases of the design process except for the areas of sketching and early schematic design. In these early phases of schematic design, architects explore many ideas at a fast pace in order to arrive at a suitable solution, without immediately committing themselves to a particular design. Because of this, CAD is often avoided at this stage due to its level of precision being unconducive to sketching and iterative drawing. Architecture students are often encouraged to sketch freehand rather than using CAD at early stages of design. Some of the reasoning behind

this is seen in this statement from Sutherland, as he points out the inherent differences between digital and manual drawing techniques:

> An ordinary draftsman is unconcerned with the structure of his drawing material. Pen and ink or pencil and paper have no inherent structure. They only make dirty marks on paper. The draftsman is concerned principally with the drawings as a representation of the evolving design. The behavior of the computer-produced drawing, on the other hand, is critically dependent upon the topological and geometric structure built up in the computer memory as a result of drawing operations. The drawing itself has properties quite independent of the properties the object it is describing. (Sutherland, 1975 as cited in Stiny, 2006, p. 61)

Since a digital drawing has an underlying structure that is independent from its visual representation, there are many aspects of the drawing that a novice user may not understand or be in control of. For architecture students at early phases of design, this means that those topological and geometric structures from the computer's memory are not accessible, but will have great influence on the way the drawing behaves and translates. While these structures offer advantages to those who are in control of them, they can represent a hazard for the creativity of those who are less aware.

Gross and Do (1996) have developed a prototype that sought to combine the strengths of manual and digital drawing to fill the sketching gap in CAD tool usage. *The Electronic Cocktail Napkin* offers users many features for digital sketching, one being a simulation of the way designers use trace paper to copy and combine drawings. As layers are created they can be stored, hidden, and brought back for use at a later time. An

important feature of these layers is their semi-transparency, meaning that the layer underneath is not totally abandoned once new layers are added. This allows an opportunity for re-interpretation, as new iterations of the drawing can be created after thoughtful reflection and the addition of a new semi-transparent layer. In addition to this layering, the program is capable of recognizing simple shapes as the user draws them. Groups of these simple shapes can be stored as configurations of elements, which the system is also capable of recognizing. Through recognizing these items, the system can then make assumptions about the drawing and relationships between its parts. The data behind these relationships can be relayed to the designer, which may offer them knowledge that they would not have gained from a system without "intelligent paper" as *The Electronic Cocktail Napkin* describes its interface. This prototype also offers tools that imitate the results of applying varying levels of pressure with a drawing or painting implement in analog methods. For example, applying more pressure with the given drawing instrument in this prototype will result in a darker, thicker line, as would occur while using a paintbrush. Despite the strengths of this project, architects have not adopted similar software in a widespread manner. It seems that attempts to directly replicate pen and paper sketching in digital form are not likely to become preferable to analog method in the eyes of architects. Features such as the semi-transparent trace paper overlays that are inspired by sketching, but do not strictly adhere to the analog methods may be more promising for future development. By taking advantage of the strengths of the computer, such as duplication, these features can enhance aspects of sketching instead of trying to imitate them.

Limitations in digital sketching

Rahinah Ibrahim (2010) displayed how CAD programs can hinder novice designers in the early stages of design by being too restrictive and limiting intuitive ideation. This was completed through a long-term ethnographic study of a design studio and its designers. Some of the benefits of analog drawing techniques were defined as, flexibility in ideation due to the tangible interface, ease of use, ease of learning, and ease in maintaining a design idea during the design process, because of the ability to see and compare all of the documents together. On the other hand, challenges with the current CAD tools included difficulty in learning and gaining the ability to use the tools, as well as losing consistency in the design due to a lack of control over the CAD software. However, the data collected in the study led the authors of this paper to propose that when used alone, neither traditional analog sketching tools nor the current CAD tools were adequate for handling the entire design process in the setting of modern architectural education. The proposed solution in this case was to devise a new form of design medium, a virtual reality (VR) device that would ease the transition between analog and digital drawing techniques.

Zafer Bilda and Halime Demirkan (2002) showed that traditional media was preferable to digital means in the categories of perception of visual–spatial features, production of alternative solutions and better conception of a design problem. This study involved the careful counting and classification of "cognitive actions," or CA, that a designer made while completing a design task. It was found that in general, more CA were carried out during the use of analog sketching and drawing tools than with the given CAD tools. Bilda and Demirkan proposed that this difference was caused by the

designers' experience with traditional sketching and their habitual activities while using traditional sketching media, which were not adequately facilitated by the given CAD tools. The authors cited that the inflexibility of the CAD tools, and the lack of a "doodling activity" kept the designers from exploring more diagrammatic representations of their work. To fix this, they suggested that CAD tools could offer semi-transparent copying activities, modeled after the way tracing paper is used in traditional media. After an initial sketch is created, a layer of trace paper is placed on top of the drawing, and another drawing is made on this new surface. Creating this second layer involves reading or "interpreting" the first drawing, while simultaneously sketching a new drawing. As more layers of trace paper are applied and sketched upon, the "re-interpretation" described by Gross et al. (2009) occurs in traditional media. Implementing a similar process in a digital environment would potentially offer more opportunities for re-interpretation and diagrammatic representation.

Despite these documented conflicts between digital and manual drawing methods, Peter Cook (2008) referenced an observation from Perry Kulper, an architecture professor who strictly works in analog format but is not averse to digital production: 'I wonder what about the 'construction lines' of the digital world. Might there be some architectural knowledge embedded there as I find in the construction lines of a drawing when working manually - constructed latent knowledge, for example?' (pp. 220-221) The term "construction lines" is synonymous with the act of drafting manually, as the skilled use of a T-square, parallel bar and triangle map out the framework of an orthographic drawing. In *Elements of Parametric Design*, Robert Woodbury (2010) pointed out the ways in which parametric modeling programs increase the number of drawing tools and

subsequently increase drawing complexity: “Parametric modeling, also known as constraint modeling, introduces a fundamental change: ‘marks’, that is, parts of a design, relate and change together in a coordinated way. No longer must designers simply add and erase. They now add, erase, relate, and repair. Relating and repairing impose fundamental changes on systems and the work that is done in them.” (p. 8) Since adding and erasing result in directly visible changes, they are easily recognized by the user. However, relating and repairing drawing items may result in changes that are not directly visible, and the user is asked to recall, rather than recognize this criteria.

Algorithmic CAD software

Woodbury (2010) described how digitally produced drawings can often be described with “directed graphs” which are made up of nodes and links. These nodes can represent the ‘marks’ in a design, as well as the actions of adding, erasing, relating and repairing. Each node is connected via links, and the resulting graph can be seen as a diagrammatic representation of the drawing and all of its parts. Schleich (1994) discussed how this can lead to a large amount of “hidden information” which is often not conveniently accessible to the user. This is distinctly different from a manual drawing, where all of the drawing’s information is visible to the designer. Schleich cited this is a problem with WYSIWYG (What You See Is What You Get) software, where a drawing, text, or other graphics can be seen and edited in a form that is close or exactly like what the final product will be. The “final product” in the case of graphics editors refers to the image that will be produced by a printer. However, the computer’s memory will usually store much more information than the printer can relay visually. Schleich refers to this as

hidden information, which is part of the topological and geometric structure that is compiled as the user creates a drawing. He offered an array of potential hidden information, including exact dimensions of an object, object grouping, invisible attributes, constraints, and design history. The solution in this case was a prototype that displayed all of this hidden information in the form of a "structure browser", so that it would be directly accessible to the user. These structure browsers consist of a series of nodes and links referring to all of the elements and attributes the drawing contains. By editing these structure browsers, a user may change grouping characteristics among items, as well as the application of attributes to specific items. When this representation of the drawing grows larger, the user has an opportunity to expand or collapse branches of the structure browser to hide or reveal the nodes as they wish. While an advanced user of the WYSIWYG graphics editor may have full knowledge of the underlying information that they have produced in the process of completing a drawing, this prototype makes the hidden information more accessible to the casual user.

Common algorithmic CAD tools such as Grasshopper and Dynamo utilize similar structures to represent drawing objects and tools, allowing designers to build their drawings through the medium of nodes and links. In a conceptual project by Daniel Belcher's *Sketching Dynamic Geometry,* (2013) a designer can sketch on a canvas and see his drawing steps automatically mapped onto a Grasshopper-like node and link interface. Whereas Grasshopper allows users to build and manipulate a network of nodes and links that produce a drawing, Belcher's project suggests that direct input in a drawing canvas could be interpreted by the system as a network of nodes and links. When the user draws as they would in a conventional graphics editor, script components appear

automatically in a separate window. In turn, these script components can be edited and manipulated to better reflect the mental model the designer holds of the drawing. The dual view of a sketch offers a designer an insight into the way that sketching data is stored, and fosters a meaningful relationship between the designer's mental model of their design and the contrasting model stored in the computer. In this way, some or all of the hidden information behind the drawing's WYSIWIG representation become more transparent and accessible to the user. Actions in one screen are mapped to the other, so that the designer moves *bidirectionally,* back and forth constantly from a literal version of their drawing to its graph representation. In this way, the gap between the user's creative, imaginative understanding of the drawing and the computer's concrete, constrained storage of the drawing begin to be bridged. These constraints can also become useful to the designer as their relationship to one another becomes clear, and design options can be narrowed down to find a solution that fits within the set constraints.

However, the bidirectional actions seen in Belcher's conceptual video are quite simple, e.g. a single line is drawn, and the two endpoints are stored as data in a directed graph format. A more well-established program that also exhibits this type of bidirectionality is Adobe Dreamweaver, a commonly used WYSIWYG aid for producing web pages quickly and without the need for detailed programming skill. HTML code is generated automatically as the designer draws out a webpage in a more conventional vector drawing format. However, this is still not very useful as a manipulable programming language, according to Paul Coates' definition in *Programming Architecture* (2010). Here he explained the distinctions between HTML and more complete programming languages: "A list of coordinates is similarly just data. For the

text to read itself we need a full set of conditional expressions, control structures and arithmetic operators. HTML and lists of numbers are essentially collections of nouns, a programming text in the sense I mean here also has verbs, adjectives and adverbs." (p. 3) Here, Coates distinction between 'nouns' and 'verbs, adjectives and adverbs' can easily be compared to Robert Woodbury's description of 'adding and erasing' versus 'relating and repairing.' Coates also mentioned the difference between simple data storage and a true parametric algorithm: "Many people casually refer to any slightly 'techy' text as a computer code, but, for instance, HTML is not a language in the sense I am trying to develop here, but a piece of data. It has a syntax and a lexicon all right, but in order to produce a web page it has to be read by an algorithm written in a programming language, which, using the data renders the page by painting the dots on the screen different colors to make words and pictures" (p. 3). By this definition, HTML code is only capable of storing the adding and erasing actions spoken of in Woodbury's text. The actions of relating and repairing are better suited for a more articulate programming language that can describe the relationships between each element within the data.

In addition to lacking the complexity and modifiers necessary for an encompassing description of an architectural drawing and its hidden information, the HTML code produced by WYSIWYG programs such as Dreamweaver is often unnecessarily bulky and untidy. Spiesser (2004) outlined the pitfalls of this bloated code, which can cause high storage costs, transmission costs, and increased download times. Because the user's actions are automatically translated into HTML code, the inefficient repetition of elements and attributes often occurs. In many cases, an attribute that could be generally applied to a large number of elements at once is instead applied repetitively

to every single element. Also, unnecessary formatting often results in dead code that serves no computational purpose, nor is it useful or intuitive for a user to interpret visually. Spiesser's work focused on finding new optimization techniques for this code, seeking out algorithms and processes that would run through the code automatically and result in a more efficient product. This often involved the elimination of unnecessary dead code, as well as reducing the repetition of attributes. While these optimization techniques may be useful and replicable, it is clear that any time a WYSIWYG software attempts to represent direct graphic manipulation through code or graphs, inefficiency and unnecessary repetition is likely to occur. This represents a challenge when attempting to improve upon the bidirectional functionality seen in Daniel Belcher's prototype software.

Research through design

In order to explore this concept of bidirectional scripting and direct geometric manipulation, this project will consist of a series of video prototypes demonstrating the capabilities of a proposed bidirectional interface. These prototypes will be heuristically evaluated to determine both their usability and usefulness to architects and HCI designers. Zimmerman et al. (2007) have put forth the values of using design as a vehicle for research, and as the topic of research based papers. They have presented a model of research that involves repetitive design and reflection, allowing research and design to occur simultaneously. In order to bring greater validity and standardization to this type of research, Zimmerman has proposed four types of criteria for judging these design research contributions. The first of these is process, judging the rigor used in selecting a

methodology for the design activities. The iterative production of video prototypes and their individual heuristic evaluations will represent the process in this case. Zimmerman's next criteria is the invention, stating that the designers must show that they have produced a novel integration of different subject matters, and how this product advances the particular field of technology. The third criteria for judging design research contributions is relevance. The term relevance is used here as opposed to "validity," as validity may suggest there would be a particular right answer that designers should arrive at, given a particular problem. Relevance is a preferable term in this situation because it suggests that the product of a design research project is useful for solving the given task, and preferable to other tools or methods. The last criteria is extensibility, meaning that the given work documented and described in a way that will allow future practitioners to leverage the knowledge gained from the design research. Using this model, video prototypes will be constructed to demonstrate the feasibility and usage of bidirectional scripting and geometric manipulation at the early schematic design level.

METHODOLOGY

The experiment design for this project consists of a single condition study: a video prototype usability study. In order to test the video prototypes, the videos have been completed and hosted on a collaborative video commenting platform developed in the UNCC Human-Computer Interaction department, called the Video Collaboratory. Human-Computer Interaction and User Experience design experts, including UNCC Software and Information Systems department faculty and students, were asked to view the video and provide comments based on the Nielsen Usability Heuristics. Architecture professors and advanced students with extensive experience with current CAD software have also participated and added comments within this forum. The Nielsen Heuristics include a list of 10 standards which are meant to be universally applicable for evaluating the usability and quality of user experience that a given interface possesses. These comments were taken into consideration during the design of subsequent video prototypes.

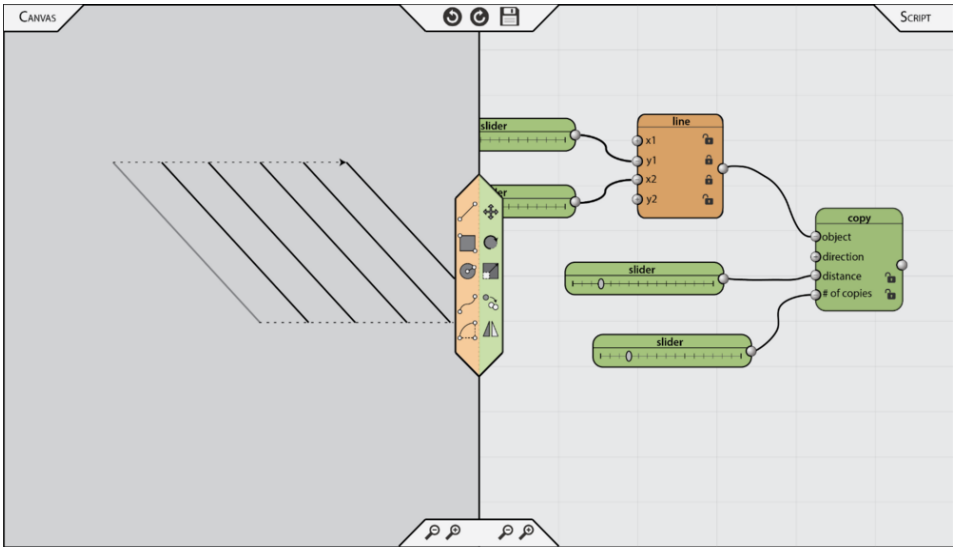

Figure 3: Video prototype screenshot

In figure 3, a screenshot from the proposed interface can be seen. The screen on the left is a canvas allowing for direct manipulation as would be seen in any modern 2D vector drawing platform. On the right, a script panel hosts the drawing data in algorithmic form, borrowing language and affordances from programs such as Grasshopper and Dynamo. In figure 4, the interface is labeled to display all of the available tools for drawing and scripting. A central menu has been purposefully placed between the two screens, as each tool is meant to be useful on both sides of the screen. For example, the user may select the line tool and draw as they would using a conventional drawing program on the canvas side. They may also select the line tool and use this to create a new line component by clicking within the scripting window.

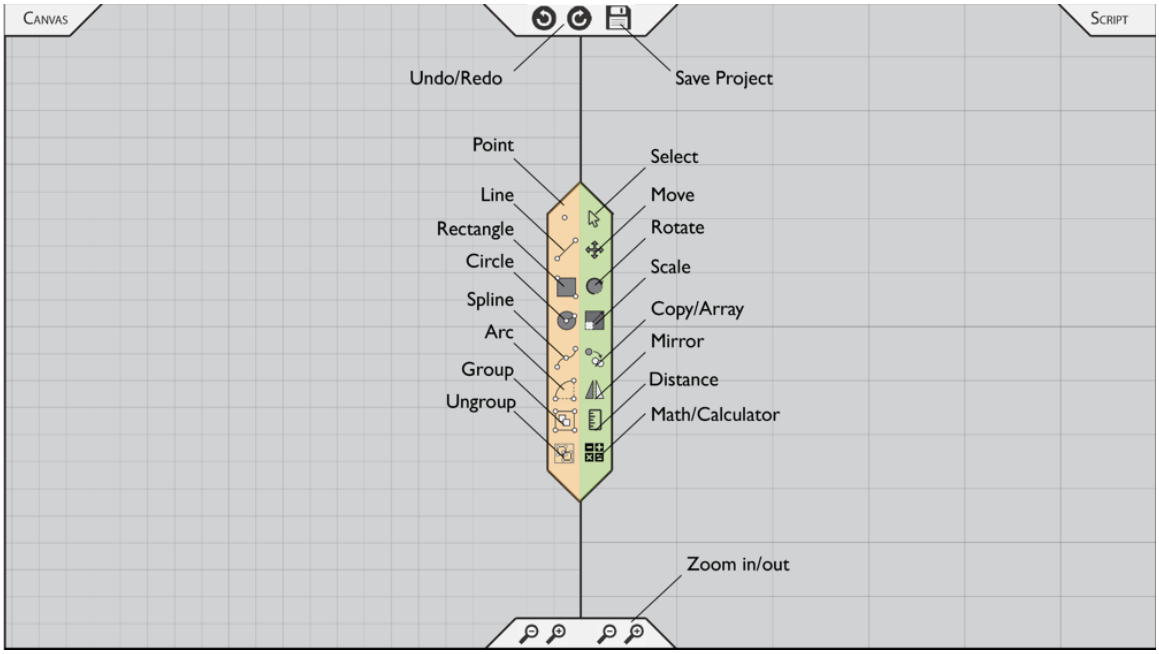

Figure 4: Labeled interface diagram

Two video prototypes have been hosted on the Video Collaboratory App created at UNCC, and shared with 10 HCI and architecture faculty and student experts in the field of UX design and architecture. They were asked to provide comments on specific

moments of the videos, with positive or negative points based on the Nielsen Usability Heuristics listed below:

1. Visibility of system status

2. Match between system and the real world

3. User control and freedom

4. Consistency and standards

5. Error prevention

6. Recognition rather than recall

7. Flexibility and efficiency of use

8. Aesthetic and minimalist design

9. Help users recognize, diagnose, and recover from errors

The HCI, UX experts were given several example comments (positive and negative) to help guide them through the review process:

"Visibility of System Status- When a user creates a line, the parameters for its coordinates are not visible by default. This means it may be difficult for the designer to understand the exact coordinates being used by the line."

"Minimalist Design - When a user creates a line on either the canvas or the script side of the screen, the same line button is used for each option. This is a good use of minimalist design."

The results gathered from this study consist of all of the comments posted by the participants on the online Video Collaboratory forum. They are sorted by Heuristic, and counted. This is done to aid in finding areas of strength and weakness in the interface. This sorting and the comments themselves have influenced the design of a final video prototype incorporating the advice and lessons learned from the heuristic evaluations.

VIDEO PROTOTYPE: LINE MANIPULATION

In this first video prototype, *Line Manipulation,* a user can be seen drawing and manipulating two lines using the bidirectional interface. This simple interaction is meant to demonstrate how the user is allowed to use both the canvas and script windows to create the drawing. In figure 5, a single line has been drawn, and the script components referencing the line can be seen appearing automatically after the line is drawn.

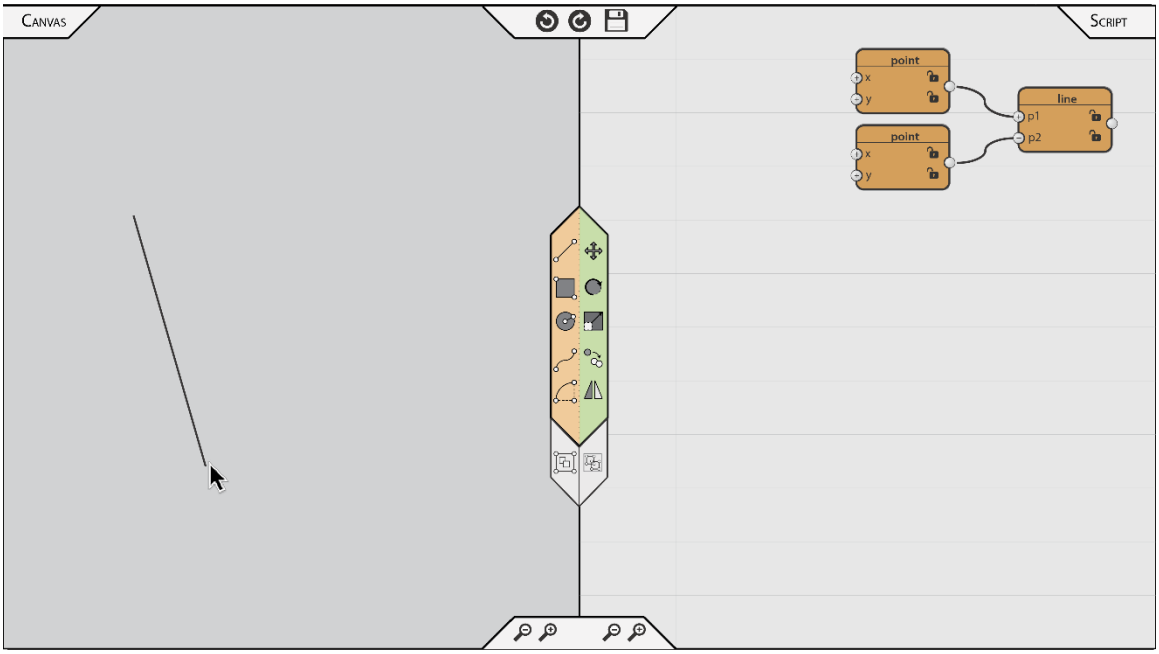

Figure 5: A line is recorded

In figure 6, another line has been drawn using the canvas window. Also in this screenshot, we can see that the user has expanded the slider components for the second point from the first line. This is done using the plus and minus buttons at the end of each connector in the scripting window. This expanding and collapsing feature has been added in order to give the user control over the amount of information seen in the scripting window.

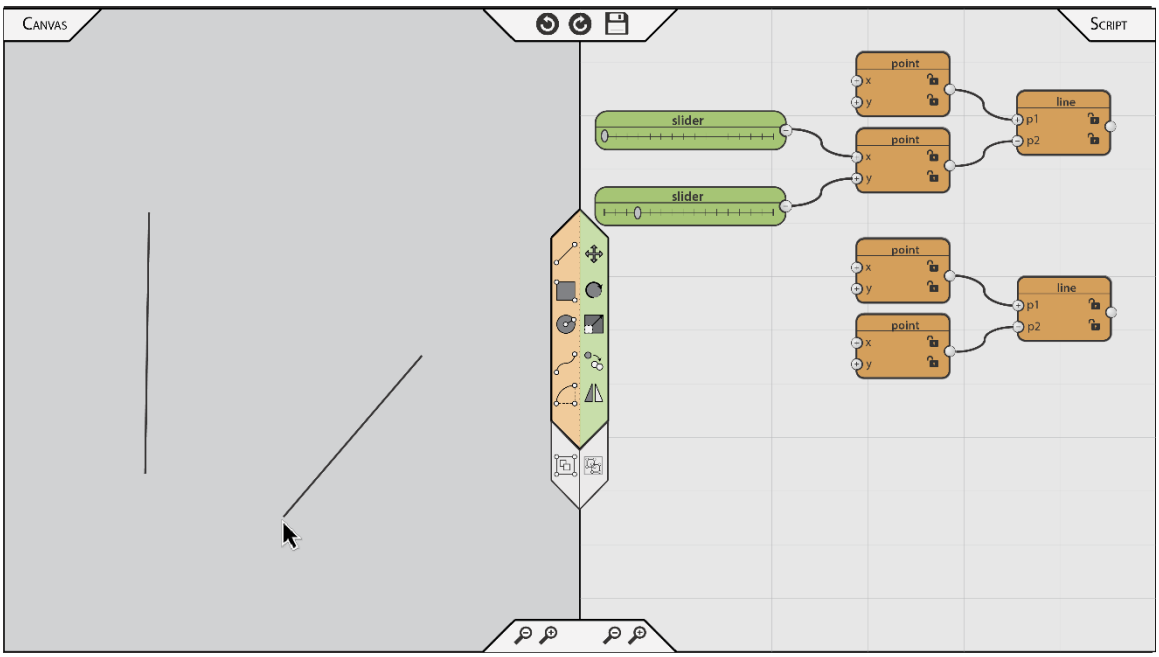

Figure 6: A second line is added

Finally in figure 7, the user has used the scripting window to create a relationship between the two lines they have drawn. The second point from the first line is now shared as a point for the second line. The video prototype displays how this change effects the way the drawing behaves in the canvas window, as the user drags one line but both are affected because of the shared point.

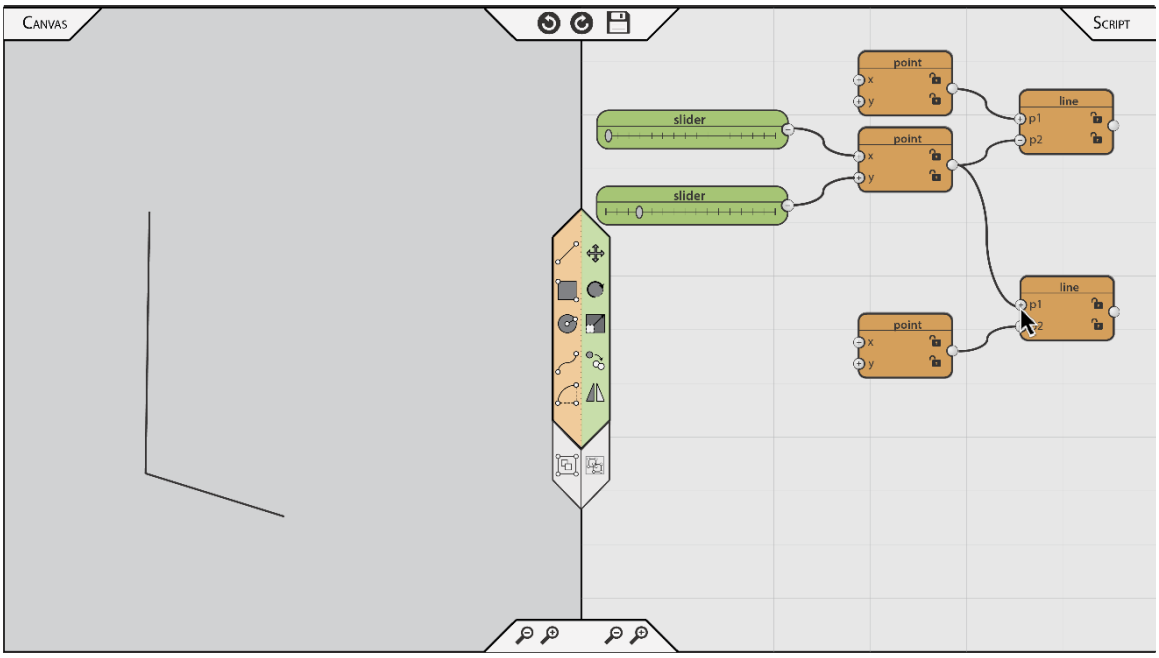

Figure 7: The two lines share a point

This video prototype is meant to be a simple introduction to how this bidirectional interface could be usable and understandable. In the next section, the results of a heuristic evaluation of this video prototype will be displayed.

TEST RESULTS: LINE MANIPULATION

In this heuristic evaluation, 5 reviewers offered 22 total comments on the first video prototype, each relating to a single Nielsen Heuristic. The results, including the relevant moment from each video, are sorted by Heuristic and displayed below. The reviewers have been de-identified and their names replaced with ID numbers.

Table 1: Consistency & Standards

Reviewer ID	Video Time	Comment
3	00:09	When the user draw a line, the line already has X, Y value but it needs a slider to change the value. If the user can change the X, Y value on the canvas window, why do we need the slider? it is not natural. We can think about another way since the user should explore a model on the canvas screen more freely. This problem is related with following similar interface with Grasshopper. It might need automatically generated slider or text box in the point component, not slider.
3	00:20	What is the small black box in the second line components?
5	00:24	The use of color to code the two sets of tools is reasonable, but it would help to have the color of the tools match the color of either scripting or canvas. Also it would help to have the tool that is being used or activated highlighter on the screen
3	00:25	Some usage is different with my modeling and scripting experience. It should consider users' mental model.
5	00:33	I think the conventions of using a grid in the canvas would really be helpful. It is now well established that it will be difficult to float the elements in empty space without disorienting the users.

1	00:40	Save buttons normally save projects, this sounds like you are exporting the project to an image.
4	00:40	Too picky, but I'll add another snap point on the middle of the line on 'spline' icon.

These comments relate to consistency and standards, and one of the most interesting is seen on top, suggesting that sliders are an affordance that is only necessary when direct manipulation of the canvas is not available. In Grasshopper, sliders are commonly used and scrubbing the cursor back and forth using a slider often results in movement or changes in the Rhinoceros 3D modeling window. This means that the affordance of moving a slider directly relates to the movement in the modeling window, creating a clearer mental model for the user. In this bidirectional interface where both direct manipulation and scripting have been made available, it may be more useful to have a text box for entering exact dimensions. This would coincide with Schleich and Durst's (1994) emphasis on exact dimensions being critical hidden information in WYSIWYG programs. Many of the other comments relate to consistency and standards as compared to modern graphics editors, which can be easily applied to increase clarity.

Table 2: Visibility of system status

Reviewer ID	Video Time	Comment
1	00:03	When a control is selected, there is no feedback that the control is actually selected.
2	00:09	When a user creates a line, the sliders for its parameters are not visible by default, meaning that it may be difficult to understand the exact coordinates being used by the line's

		endpoints.
3	00:20	How does the user know which point(canvas window) is which component(script window)? It just automatically generate points
4	00:21	It is really hard to follow the order of the points from the script to the geometry specially at this moment when it gets more complicated. Perhaps, it is better to mark the points on the screen (LEFT), all in red, and once you click on the right side the associated point becomes green on left.
1	00:30	There no indication that these items will be selected. Basically it's unclear if I need to envelop the items for them to be selected or if it might include the items crossed by the bounding box borders.

The comments in Table 2 highlight many drawing conventions that are common in modern drawing programs, but missing from the video prototype. The first comment about knowing which tool is selected is usually solved by having distinct cursors for each tool. There are also comments unique to this category of software involving multiple windows displaying the same information through different means. When the user selects an item or component in one window there should be an indication, such as highlighting, of what corresponds to the currently selected item in the other window.

Table 3: Recognition rather than recall

Reviewer ID	Video Time	Comment
5	00:27	It would be really useful to have some way to highlight the links between the canvas and the script. Perhaps some form of highlighting would really help, particularly as both side become more and more complex
5	00:31	The two shape tools at the bottom of the toolbar are different in kind than the others, but the graphics are not very clear about that
1	00:34	Rotational tools normally indicate the central point of rotation. This helps the user recognize where the originally set the point (e.g., clicked on line) especially as they drag their cursor away to rotate the object.

Again, highlighting between two windows is referenced in the heuristic "recognition rather than recall." This is understandable since without any indication of what script components control which drawing items, a user would have to remember all of the connections they have made. In this drawing where only two lines are drawn, it is not terribly difficult to recall which script component refers to which line. However, these scripts will grow much larger, and with each added script component the task of recalling the connections between drawing and script will become more difficult.

Table 4: User control and freedom

Reviewer ID	Video Time	Comment
4	00:11	The number sliders need to show numeric values with the option to be flexible from both sides.

4	00:33	It's nice you can rotate intuitively but what if you what to give specific rotation angles... this may be fixed by adding a number slider to the right side once you click on rotation icon.

Table 5: Aesthetic and minimalist design

Reviewer ID	Video Time	Comment
3	00:25	Most modeling or drawing apps use top menu bar and left menu bar since it is useful for utilization of space. Moreover center menu prevent users' attention to modeling.
2	00:42	When the user creates a line on the canvas or script side, the same line button is selected from the central menu. this is a good example of minimalist design

In table 5 it is seen that the choice to introduce a central menu may be controversial, as it could be distracting to designers in the center of their view. However, for demonstration purposes the central menu may be an effective way to remind users that the menu tools can be used on either side of the screen.

Table 6: Match between system & real world

Reviewer ID	Video Time	Comment
4	00:11	The number sliders need to show numeric values with the option to be flexible from both sides.
4	00:33	It's nice you can rotate intuitively but what if you want to give specific rotation angles... this may be fixed by adding a number slider to the right side once you click on rotation icon.

Table 7: Error prevention

Reviewer ID	Video Time	Comment
1	00:21	There's no indication (like it turns red) when making this connection that the previously connected module will be removed.

Error prevention, as referenced in table 7, is something that has not been addressed in this prototype, but would likely be key to the success of such a bidirectional interface. Error Prevention and management of errors is a very important topic for this software which will be prone to mistakes, but the user in the video does not cause any errors. Often there will be limitations because of the need for communication between the dual windows, and the user must be well informed of these limitations. The comments relating to visibility of system status reflect that When a user makes changes on one side of the screen, it needs to be more clear what changes are happening on the other side, and why. Also, some affordances from grasshopper, such as sliders, may not be appropriate since direct manipulation has been enabled in this proposed interface. Lastly, Consistent standards for drawing software are widely known and used, and the next video needs to follow these more closely. Overall, the collection of comments on this first video prototype consisted of constructive criticism and were useful in producing the next video prototype, which grows in complexity and will be shown in the next section.

VIDEO PROTOTYPE: ADJACENCY DIAGRAMS

In this second video prototype, *Adjacency Diagrams*, a user can be seen constructing an adjacency or "bubble" diagram and manipulating the relationships between the items, which are represented by circles. Bubble diagrams, or adjacency diagrams, are a tool used by architects in certain settings for space planning at an early phase in the design process. Each "bubble" represents a space, room, or other programmatic requirement in a building. In figure 8, the user has used the canvas to draw two circles as the first two items in the diagram. Then, using the script window, the user creates a relationship between the two items – in this case the radius of the second circle is set to be equal to one third of the distance between the two circles. Moving back to the drawing canvas, the user then moves one of the circles and the radius of the second circle is impacted, as the distance between the two circles has changed. This is an example of the user seeing and modifying the hidden information in their drawing, which then has an effect on the way the drawing reacts to direct manipulation.

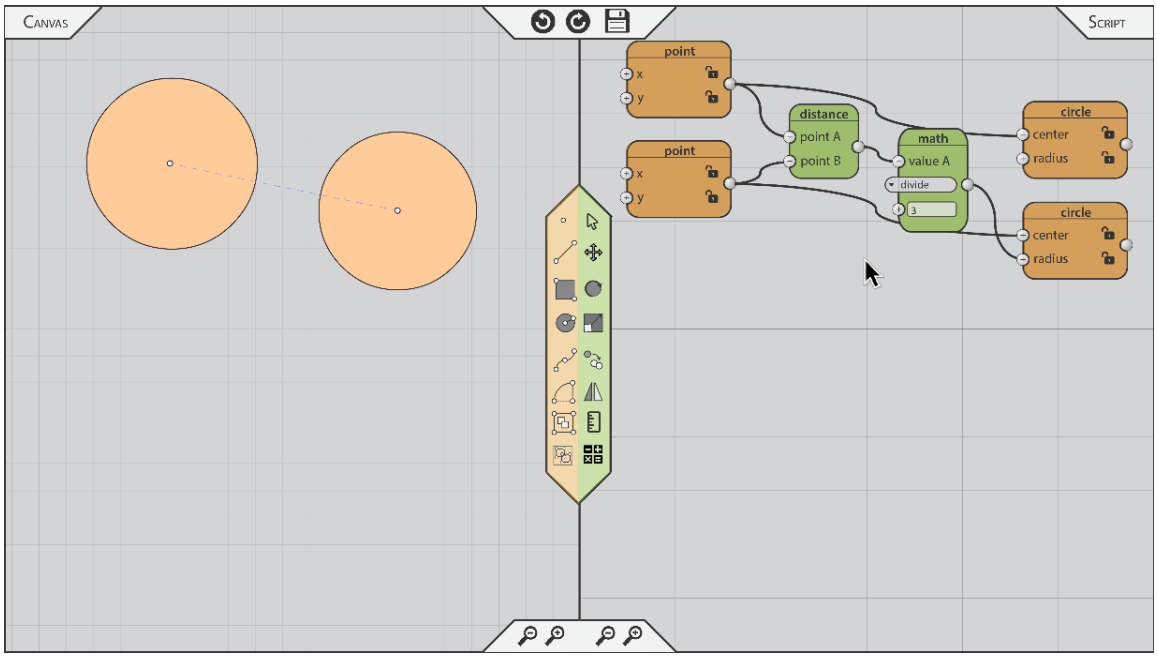

Figure 8: Relationship between two diagram items

In figure 9, the diagram has been completed with two more circles, and each radius is related to the distances between the circles. These important dimensions are displayed as dashed lines in the video prototype. The user constructs these relationships in two ways. First, the distance tool is selected, a component is created in the script window and then the desired circles are related to one another by manipulating the script connectors. Then, the user again selects the distance tool and uses it by directly clicking on the two circles in the drawing window which they wish to relate. When a circle is selected for use, the corresponding circle script component is highlighted to aid the user in understanding the automatically generated distance component which follows.

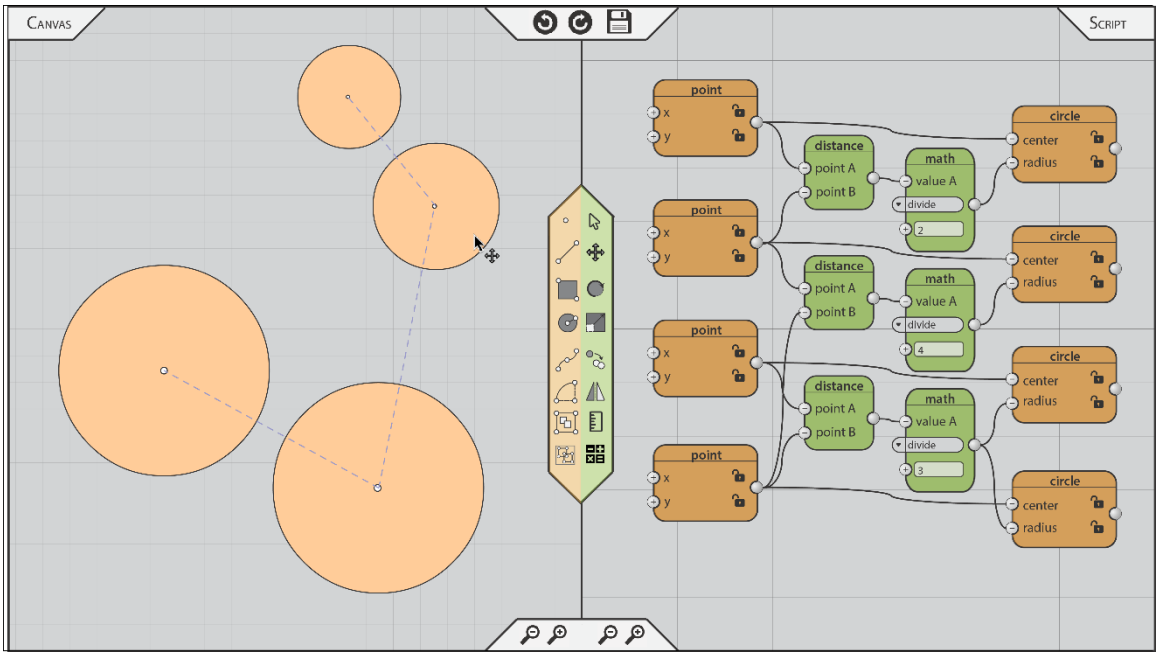

Figure 9: Adjacency diagram w/ complexity added

TEST RESULTS: ADJACENCY DIAGRAMS

In this heuristic evaluation, 6 reviewers offered 26 total comments on the second video prototype, each relating to a single Nielsen Heuristic. The results, including the relevant moment from each video, are sorted by Heuristic and displayed below. The reviewers have been de-identified and their names replaced with ID numbers.

Table 8: Visibility of system status

Reviewer ID	Video Time	Comment
3	00:00 - 00:14	When the user create a circle in Canvas, the circle box in the Script screen does not shows X, Y, and radios value.
8	00:04	It is visible that the system automatically generating scripts from geometric manipulation, which keep users informed about what is going on.
3	00:14	What does the black box of second circle in the Script Screen mean?
7	00:34 - 00:39	Wished I could see the numbers for radiuses and distance as playing with the circles position. It would be helpful to see a numeric representation of the calculations in addition to the visuals.
9	00:43	When you select a circle and move the script module is also highlighted. It shows "system status".
8	00:59	There are 4 circles in the script, two of them includes black squares in the box, but it couldn't tell what is that in the circles that are in the canvas.
9	01:04	When you move the circles around and change the distance, the distance is not reflected in the script. And it does not show that the distance is changing. so system status here is not visible.
7	01:04	When user selects an object in the canvas the related code block changes color to clarify the relation. But the opposite

		does not happen when the user starts by selecting from the script area.

As with the first video prototype, visibility of system status was one of the most referenced heuristics in this evaluation. In this second round of evaluation, it seems that the comments have grown more specific, and there may be less confusion about the connections between the canvas and script windows. Unfortunately something that has been referenced in both evaluations was a black box that resulted from a rendering error in the video, and it is imperative to fix this for the final video prototype.

Table 9: Flexibility and efficiency of use

Reviewer ID	Video Time	Comment
3	00:03	Most icons in the menu support only either Canvas screen or Script screen. It means that each screen has own functions. If all menu can support both screens, it will be better for expert user. In addition, providing more functions in Canvas screen can encourage to design more freely. (Current video shows that Canvas screen looks controlling an established script.
8	00:04	With the feature of automatically generating scripts from geometric manipulation, users can see the model results directly and also control the variable value in script accurately. The way that system generates scripts automatically will really be flexible and efficient for user to build the model.
3	00:57	The circles have same rules. If the user can copy a set of rule in the both screen (Canvas and Script screen), it will be easier to design a repetitive pattern.
5	0:00:23	Distinction between novice and expert users may be critical. It would seem that assuming that users will already know scripting could be a given, but if you are designing for users with limited understanding of scripting, you may need a more suggestive interface for it to be useful. Or the interface could have shortcuts for experts.

A common area of confusion in these evaluations seems to be raised around the central menu. Each tool in the central menu is meant to be usable in both the canvas and script windows, allowing the user to choose between interacting directly with drawing, or indirectly through the script they have created. However, many of the reviewers seem to have interpreted some of the tools as being useful only in one window.

Table 10: Recognition rather than recall

Reviewer ID	Video Time	Comment
3	00:57	Current system depends on Script to establish a rule for modeling. For example, if I want to replace the circles as other shapes such as rectangle and star keeping current script, I should have same process again going back and forth between Canvas and Script screen. However, I want to simply sketch a modeling having automatically creating script.
5	00:58	The layout of the scripting size does a good job of expressing the relationships of the different element (better that with the single pair where the connecting lines sometimes are hard to read. How will be handled automatically with some sort of "cleanup" function, or are you thinking that you will use the user input to keep it organized and legible?
8	00:59	Once user creates several circles in the canvas, it is hard to remember which circle is related to specific circle box in the script. sometimes the system highlights the circle box when user clicks the circle but sometimes not, and it didn't highlight the circle when user clicks the circle box. It will be great if user can understand easily which circle and circle box that they are exactly using.

Here it is pointed out that the highlighting has been implemented from script to canvas, but vice versa is still difficult to recognize. These comments do seem to confirm

that using highlighting as a way of making these connections between the two windows clear. However, the highlighting will need to occur consistently across both screens if this feature is going to be considered truly bidirectional.

Table 11: User control and freedom

Reviewer ID	Video Time	Comment
8	00:32	The two buttons in the top are helpful for users to undo/redo, they are easy to understand and provided the options for users.
6	0:00	User control and freedom: the locks on the buttons could be placed to the left near the things that they are actually locking, it isn't obvious when looking at them that there shouldn't be two outs from the node where they are sitting currently.

The padlock options on each script component allow for constraints to be applied to the drawing. If a padlock is “locked” then the unit which it refers to will not be allowed to change during direct manipulation of the drawing. The second comment in table 11 critiques the readability of this feature, and there is no precedent for a feature like this to be taken from Grasshopper. As suggested in the comment, moving the padlocks to the left side of the script components may make them more readable, but this may interfere with the “+ and -” icons which allow for collapsing and expanding of the script.

Table 12: Match between system and real world

Reviewer ID	Video Time	Comment
9	00:22	Some controls are intended for Script only, and some controls are intended for both and it's not obvious which is which. I

		think the menu icon placement is peculiar, which relates to the "real word" heuristic.
7	00:33-00:36	I guess the black box in the second circle script means the radius is a dependent variable. It was not clear how it was created that way and the box is not a familiar way to represent that.
6	00:25	The Math operation could be more cleanly organized its not obvious how it works at first glance
5	00:47	The fact that the bubbles change size can be confusion for users of these diagrams who expect that the size of the bubbles represents the area of each activity, and that it does not change because of adjacency.

In table 12, the first comment references some confusion about the central menu that seems to be shared by many of the reviewers. Each tool in the central menu is meant to be usable in both the canvas and script windows, allowing the user to choose between interacting directly with drawing, or indirectly through the script they have created. However, many of the reviewers seem to have interpreted some of the tools as being useful only in one window. This would likely be quickly made clear in a working prototype, but this confusion may be resulting from the menu's two colors divided between the two windows.

Table 13: Aesthetic and minimalist design

Reviewer ID	Video Time	Comment
8	00:04	The design of interface is clear and contains relevant information of main function.
6	00:25	The swoopy curves could be more like grasshoppers, I think the longer ones need to straighten out as they get longer.
5	00:00	The gridding on the drawing end seems appropriate to its spatial character.

Table 14: Error prevention

Reviewer ID	Video Time	Comment
7	01:10	We cannot judge your design in term of error prevention as we are not trying the design ourselves and no evidence of error prevention was shown in this video.

Table 15: Consistency and standards

Reviewer ID	Video Time	Comment
5	0:00	The tool palette would be clearer if it was separated in sets of tools that are a. available to both drawing and scripting sides (selection tools, etc.) b. Available only to the drawing side c. Available only to the scripting side. Use of color background and or location on the screen could help with this

Again in Table 15, confusion surrounding the central menu is brought up. Regarding consistency and standards, this confusion may be related to one inconsistent feature that has been seen in the first two video prototypes. When the user uses the move tool in the canvas view, no move script component is reflected in the script window. In Grasshopper, a move script component applied to a shape would result in a copy of that shape, after the translation implemented by the move component has been applied. If this were directly implemented in the bidirectional interface, countless translation script components would be automatically generated whenever the user manipulates the drawing directly. It seems that the user needs options to make simple translations without recording inconsequential movements in script form, but also an option to make more deliberate translations recorded as script components. A resolution here could involve offering a "resize" tool when the user hovers over or clicks on shapes in the canvas, as seen in programs like Adobe Illustrator. Accompanied by changes in the cursor reflecting opportunities to stretch, move, or rotate the object, this convention could allow the user to make changes without creating new script components. Rather, the changes would be reflected in the data which has been stored for that particular shape. This can be seen in the video prototype, *Space Planning*. In this video the user stretches rectangles and the sliders which refer to them can be seen changing to reflect the new values for their width and height. In future prototypes, selecting the move tool from the central menu should result in new script components being created, while translations done without selecting a tool from the menu will not result in new components.

HEURISTIC EVALUATION ANALYSIS

A grand total of 48 comments were gathered during the heuristic evaluations, from nine different reviewers. Visibility of System Status was the most consistently cited heuristic, and this is likely due to the need for better affordances to highlight the connections between the script and canvas windows.

Table 16: Comment totals per heuristic

Nielsen Heuristic	Number of Comments	
	Line Manipulation	Adjacency Diagrams
Consistency & Standards	7	1
Visibility of System Status	5	8
Flexibility and Efficiency of Use	0	4
Recognition Rather than Recall	3	3
User Control and Freedom	2	2
Aesthetic and Minimalist Design	2	3
Match Between System and Real World	2	4
Error Prevention	1	1
TOTAL	22	26

It is difficult to gather significant observations from Table 16, as the comment totals are different and not all of the reviewers were present for both evaluations. There is also the factor of the Video Collaboratory being a public forum where the reviewers can see each other's comments, and this may have some influence on the popularity of a

particular heuristic over another. Two of the Nielsen Heuristics were never mentioned in the review: Help users recognize, diagnose and recover from errors, and help and documentation. This means that these two heuristics were either not relevant or not applicable with the content given in the video prototypes. Error Prevention was only mentioned once in both evaluations, since this was not a topic covered in the prototypes. It would be ideal to showcase some ideas for error prevention and handling in the future. The most significant change from one evaluation to the next was in consistency and standards, from 7 comments in the first to one comment in the second evaluation. Perhaps this can be attributed to an increase in consistency from the first prototype to the second, but it may be more likely this was due to the public nature of the comments amongst the reviewers.

VIDEO PROTOTYPE: PERSPECTIVAL DRAWING

In this final video prototype, *Perspectival Drawing*, the user can be seen drawing a basic two-point perspective drawing using the bidirectional interface. The choice of a perspective drawing for the subject matter in this video may be seen as a departure from the environment of early design sketching, as perspective drawings are not a universally utilized tool at this stage. However, perspective drawings are sometimes used at these early phases to make design decision a about a three dimensional space. Also, a perspective drawing offers opportunities for visually interesting adjustments in the canvas window, and constraints set in the script window that will affect those canvas view adjustments. A new feature seen in this video is the use of an "object snap" feature that many drawing software users are familiar with from programs such as AutoCAD. When the user hovers over a point or intersection of two lines in the canvas view, a yellow symbol appears to allow them to select an endpoint or intersection point. In turn, this point is referenced in the script for any new shapes that are drawn. In figure 9, the user has established one of the points as a vanishing point, editing its title to reflect this designation. The vanishing point is then used to draw four perspective lines, with each new line component utilizing the vanishing point as its first endpoint. In this way, another level of automatic generation occurs, since the interface can interpret which point the user would like to reference automatically. Also, figure 10 demonstrates how the vanishing point component is highlighted when the user references it, improving the visibility of system status. This was a major area of concern raised in the heurist evaluations.

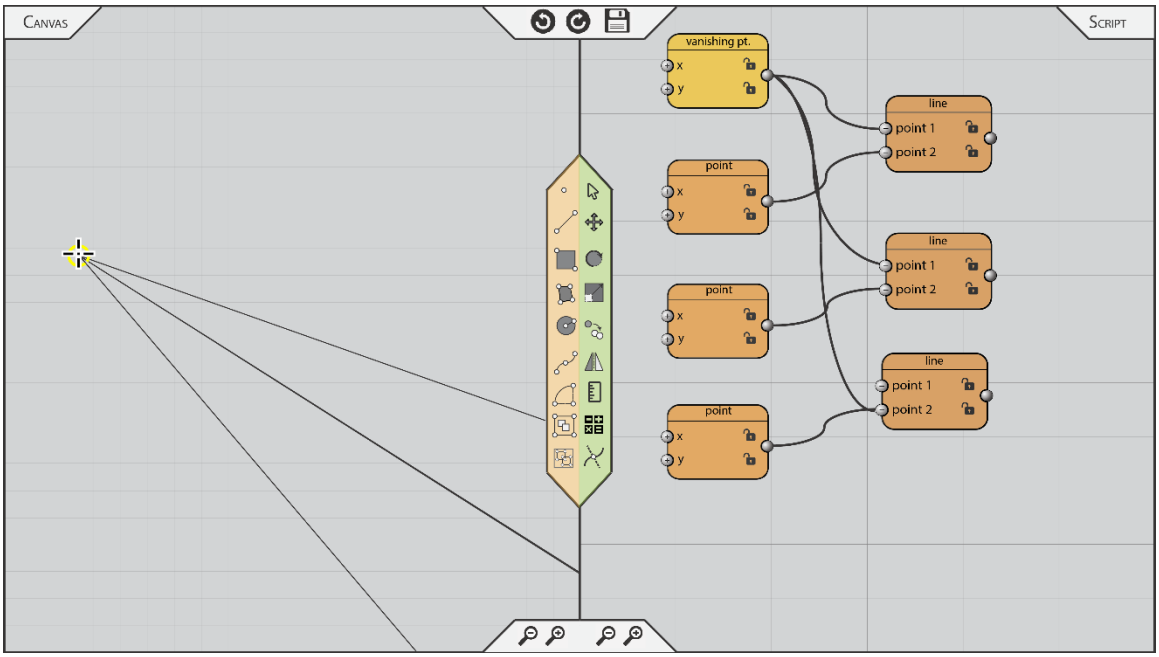

Figure 10: Establishing a vanishing point

In figure 11, the user has created all of the parts necessary for their perspective drawing. Using the intersection object snap, points were created which represent each corner of the 3D volume drawn on the 2D canvas. Then, these points are referenced by three quadrilateral shapes, so that they are dynamically attached to the perspective lines. Now the user will be able to manipulate their perspective drawing directly in the canvas, while setting constraints for the drawing by editing the script components.

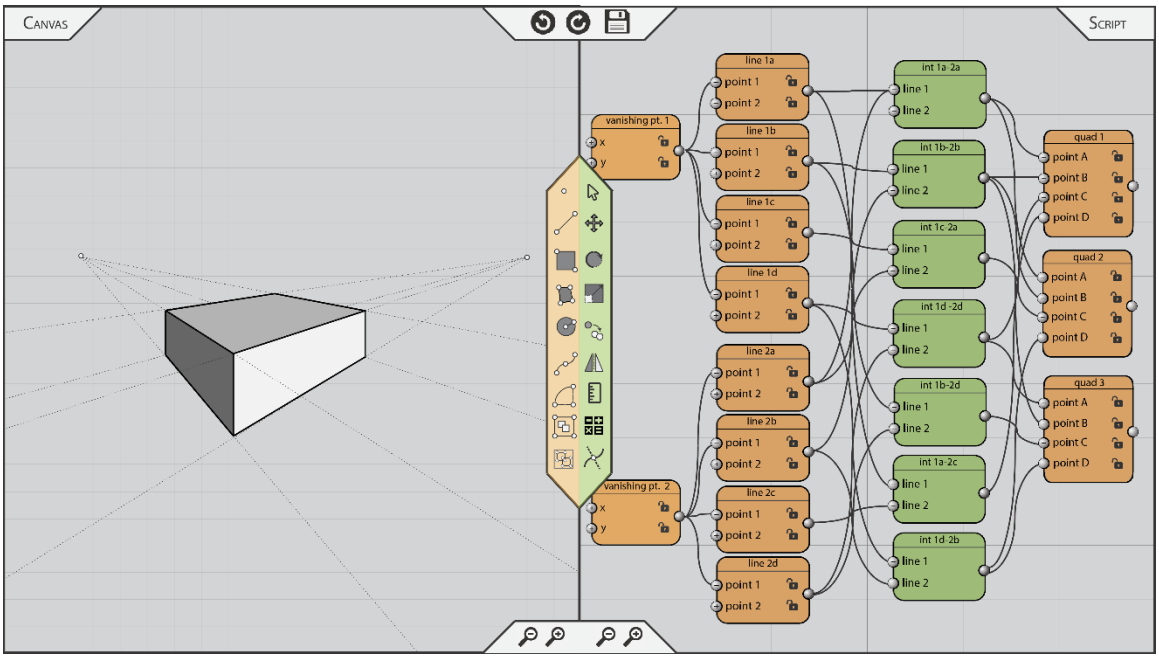

Figure 11: Two-point perspective is established

Finally in figure 12, the user has used the zoom tool on both the canvas and script views. In the script view, this zoom was carried out to isolate the script components which the user wishes to edit. By utilizing the padlock icon, the x values of the two vanishing points have been constrained to their original values. Consequently, when the user drags elements of the drawing, these points will be immovable in the x-axis. This represents a constraint that the user wishes to impose on their drawing. Another modification of the script that has occurred is the grouping of the two vanishing points, so that when one point is moved, the other will receive the same translation. In this way, the user has applied a constraint that will effectively act as a movable horizon line for the perspective drawing. After these constraints are applied, the user may skew the entire drawing by dragging just one of the vanishing points. As seen in figure 11, this results in a new view of the 3D volume that can perhaps be seen as an eye-level view of an architectural volume, as opposed to the “helicopter” view that was seen beforehand.

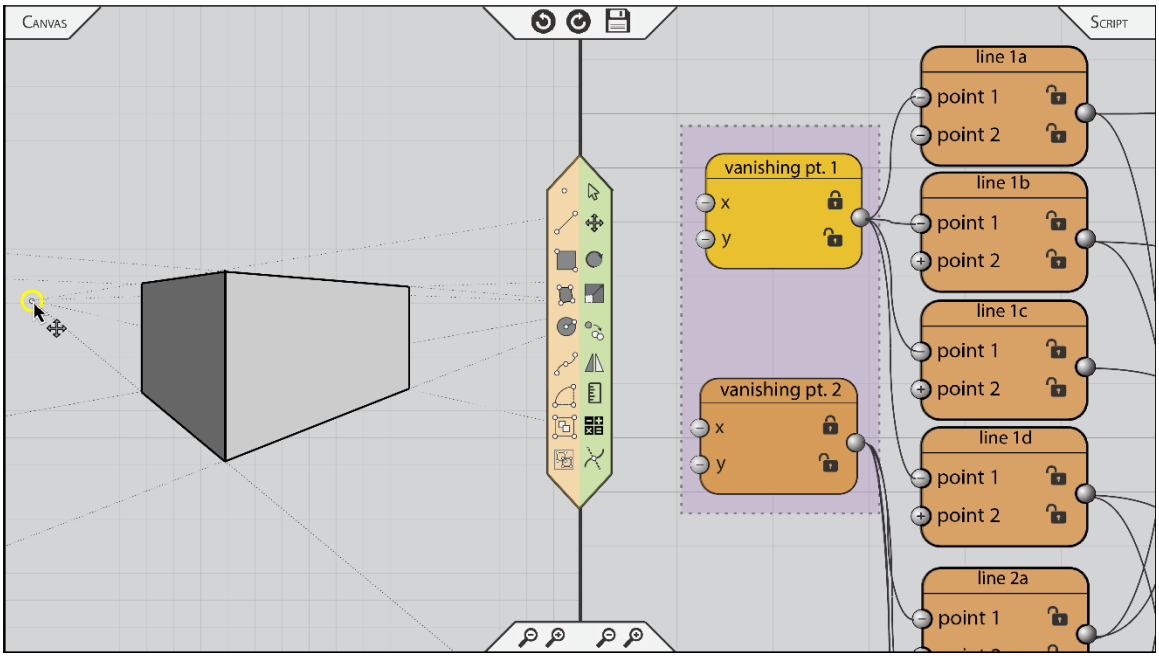

Figure 12: Perspective manipulation

Any number of constraints could be applied to this drawing according to the user's intent. The first constraint applied effectively allows for adjusting the field of view in the perspective, while the second collection of constraints results in a manipulable horizon line. If the design problem in this case were to create an appealing perspective of the given volume, the user has further defined the problem by restricting which items can be changed. By manipulating what would normally be hidden information in a conventional 2D drawing interface, the user has created a dynamic and parametric perspective drawing. Without the script view and the established relationships between each shape and point in the drawing, the user would have to shift all of these elements every time they wished to make an adjustment. The borrowed conventions of object snap allow this proposed interface to feasibly interpret the users' intent, resulting in a drawing that can be manipulated efficiently.

FUTURE WORK

Future work stemming from this project would of course lend itself to producing a working prototype based on the video prototypes and their heuristic evaluations. A user study involving design students completing a simple design task could potentially allow for a bolder hypothesis regarding the capabilities this software could lend to its users. If it can be shown that this bidirectional platform could have a positive effect on the insight a designer gains from the act of completing a sketch, this would be very encouraging for further research in this area. If a working prototype were created, a user study could involve an analysis of design work done with bidirectional scripting and geometric manipulation. The product of this analysis could then be compared to work done by designers completing similar tasks within the type of single direction interaction that has already been seen in scripting programs such as Grasshopper. Think-aloud protocol could be utilized to gather more intangible observations from the users of each software, and differences in the design process of the two groups of users. Another primary feature that Grasshopper's single direction interaction does not offer is highlighting connections to the script as a user manipulates the drawing directly. By offering this feature, the bidirectional prototype could help users in learning how each script component affects the drawing in a fluid manner. Hypothetically, users and their design work would benefit from this new ability to fully manipulate and understand two separate representations of their drawing in script and canvas windows.

As script components are automatically generated from direct input, an interesting phenomenon could occur in reference to Belardi's (1994) analysis of cognitive studies completed by Benjamin Libet. In these studies, Libet found that after an impulsive

drawing act, the brain takes 500 milliseconds to process the act consciously, while the sensory reception of the act occurs in 150 milliseconds. It is possible that the appearance of newly generated script components would occur during this "black hole" gap in cognition as Belardi has described it. As the user assigns a conscious meaning to the act of drawing they just completed, they may also become aware of the script definition that the system has assigned to their action. The influence that this could have on the user's creativity and the quality of their drawing could potentially be revealed through a user study, given that a working prototype is completed. Since the generated script definition of the user's actions could conflict with their personal assigned meaning of the action, the influence this has on the user could certainly be negative as well as positive. However, any substantial influence would be encouraging for further research. Negative results regarding the system's influence on the user and drawing could also reflect and validate the discrepancies between hidden information and the mental models that CAD users hold of their drawings.

In terms of improving the development of this type of software, such a user study could also reveal critical scenarios where error handling and management would need to take place, as the user misunderstands the interface or completes an action that is not possible within the bounds of the bidirectional platform. In the current video prototypes, the "user" moves along a predetermined script without making mistakes, getting confused, or changing their mind at any point during the process. Designers using a working prototype of this bidirectional interface would likely go through all of these actions, and their difficulties with the software would be helpful for designing further error handling.

If this platform were also accessible as a plug-in for more robust software such as Rhinoceros or any Autodesk product, a user could potentially switch back and forth between the bidirectional software for an environment conducive to ideation, and the more technical environment offered by the established software. Autodesk has shown its interest in a platform such as this with their development of FormIt (2012) which is essentially a Sketchup competitor that would be directly compatible with their other lines of software. Currently, products such as Revit are a necessity for architectural offices to use due to its efficiency and the competitive nature of the architectural practice. However, these tools do not currently offer dedicated sketching or preliminary design features. Belardi (1994) notes that two primary qualifying features of sketching is that it must be quick and readily available (p.32). If designers using Revit or similar products had tools dedicated to early design that could be described as "readily available" without exiting their CAD software, this may allow for the advantages of sketching to be implemented. The current lack of such tools may lead to a use of premade, detailed components at early phases of design. Consequently, the user would lose control and awareness of an abundance of hidden information that is held within these components. Donald Schon (1983) pointed out the advantages of manual drawing in this regard, "drawing functions as a context for experiment precisely because it enables the designer to eliminate features of the real-world situation which might confound or disrupt his experiments." (p.159) Where analog sketching can be beneficial by forcing abstraction, the use of digital drawing tools which infer information may introduce unnecessary and unintentional variables to the creative "experiment" which Schon discusses. A platform that is honest and accessible in regard to the automatically generated hidden information

could be helpful in this instance. While experienced users may not miss this hidden information, it could undoubtedly become useful to designers who would like to have greater access to the geometric and topological structures that are created as they complete their drawing.

The video prototype *Perspectival Drawing* introduces the capability for users of this proposed software to apply and edit constraints in their drawing. One of these opportunities is seen in the padlock feature that has been proposed in this interface. This feature is used to demonstrate methods for setting constraints in the script window of the prototype, which add definition and complexity to the drawing. The "group" tool is also used as a constraint, as two grouped points cannot be moved independently from one another, and the distance between them is made static. A collection of constraints can make up the definition of a design problem. As seen in Donald Schon's *The Reflective Practitioner,* (1983) identifying and developing design problems is key to procuring an effective solution. Currently, the padlock feature allows the user to lock specific values, which will result in elements in the drawing window no longer being open to direct manipulation. The concept of constraints could be expanded to other areas of the presented drawings, such as allowing the definition of maximum and minimum dimensions or point values for each element. These are features seen in Mark Gross' (1985) prototype *Constraint Explorer,* and future work on this bidirectional interface would benefit from the addition of these options.

Another less intentional constraint for users of this bidirectional system will be the limitations of the software. When live users are introduced in a working prototype, they will often attempt actions that are beyond the capabilities of the software. Error

prevention represents an area that has not been thoroughly addressed in this project, and this would likely be critical to the user experience in a working prototype of this software. As Spiesser (2004) has established, a bidirectional platform such as Dreamweaver often produces code that is far from perfect, and proper error prevention could be a solution to this. If the software could guide a user toward making decisions that made efficient scripts, this burden would be partially lifted from the software's automatic script generation algorithms. The Nielsen heuristic "help users diagnose and recover from errors" could potentially even become more important than error prevention in this case. Inevitably, users of this bidirectional interface will complete actions that are either impossible or are inadvisable because of the inefficient script that they would produce.

Scalability is another area of concern that has not been fully addressed by the video prototypes produced in this project. While the script in *Perspectival Drawing* grows to contain 20 components, complex scripts produced in Grasshopper may contain hundreds of components. Given that components are automatically generated in this bidirectional interface, the size and computability of the scripts could quickly become unmanageable if the user draws in the canvas without concern for the resulting consequences in the script window. One fundamental feature in Grasshopper is the ability to use lists of data, and this feature is not seen in any of the video prototypes. These lists can be used as a collection or container, in order to group some variable number of data items that have a shared significance to the problem being solved and need to be operated upon together in some controlled fashion. An example where this could be utilized is seen in the video prototype *Adjacency Diagrams* when the user draws two circles, and

this results in 4 components; two circle nodes and two point nodes related to their centers. In Grasshopper, a single circle node could be given a list containing the data for those two center-point nodes, and the drawing would remain the same. More coordinate data could be added to the collection, resulting in more circles in the drawing, but still just one circle component. This is an issue that other WYSIWIG interfaces such as dreamweaver often fail to complete efficiently, sometimes creating a great number of separate elements that could easily be condensed into a collection or list. If the user in this bidirectional interface could group data into a list, or if it could be done automatically, this would reduce the size and the "spaghetti-like" nature of scripts. One feature that was seen in the prototypes for this issue is the "+ and -" icons, which are meant to allow for collapsing and expanding of branches of the script. While this would not actually reduce the size of the overall script, a user could hide components that are not important or necessary for their intentions. This would result in a script that is more usable and understandable for the user, without actually compressing or losing any complexity of script data.

Just as Dreamweaver is a tool often used by those without the coding experience to directly script a webpage in HTML, this proposed bidirectional interface could be an entry point or teaching tool for architecture students and professionals who wish to move toward a more technically complex algorithmic software like Grasshopper or Dynamo. Since direct manipulation of the canvas is available, anyone familiar with modern vector drawing programs could begin a drawing in the canvas view of this program. As the script updates in a live format, the user would potentially be able to learn quickly about the links and nodes that are generated by their actions. As the video prototypes for this project were produced, improving the visibility of system status was a major focus from

one video to the next. If this platform were to be viewed as a teaching tool, the highlighting of nodes in the script and drawing items in the canvas would be absolutely critical to new users who are trying to learn what effect their drawing inputs have caused. While seeing script components automatically generate during the drawing process will be helpful for the comprehension of these new users, things may happen too quickly and abruptly for full understanding. In this case, the ability to select each item and see its corresponding script or drawing element in the opposite window become highlighted could be a highly effective method for gaining understanding of the script.

Overall, the future work for this project would consist of a working prototype based on the heuristic evaluation of the given series of video prototypes. This working prototype would carry forward the goal of improving visibility of system status that have been addressed through the series of videos. As live users are introduced to this working prototype, they may be asked to find solutions for simple design problems. This would allow for an evaluation of the results of their activity against similar tasks performed in scripting interfaces that do not offer bidirectionality. The benefits of this proposed bidirectional software at early phases of design could then be measured, through the quality and creativity of the design solutions the users in each system produce. Think-aloud protocol could also be used during these user studies, to compare the understanding that users of the bidirectional software gain to those of the single direction interface. Error handling and prevention issues would also arise as the user pushes the bidirectional system to limits that it cannot reach. Managing these errors would be part of a constraint defining system that would also offer opportunities to lock numeric values, group objects, and set maximum and minimum dimensions of shape items. As a designer establishes a

collection of constraints, the design problem will become more properly defined, and the corresponding solution could benefit from this clarity.

CONCLUSION

In reference to a lack of effective sketching and preliminary design tools in modern CAD software, Gross et al. (2009) established a set of guidelines for future software development in this field. To describe sketching, Gross stated that designers who are proficient with sketching must not only apply marks to a page, but iteratively interpret and re-interpret their sketches in order to arrive at new conclusions and to generate ideation. Belardi (1994) reflects this statement with his proposal that repeated exploration of a single drawing in different times and contexts leads to restructuring of the drawing, and that the repetition of this process is key for the creativity and ideation (p.20) New types of CAD software could utilize this description of sketching by offering opportunities for re-visualizing a drawing across different forms of diagrammatic representation. In order to achieve this goal, this project has sought to find ways to combine the strengths of direct digital drawing and algorithmic scripting software. This project has also pursued methods for improving the accessibility to information that is often obscured in conventional CAD software. Schleich (1994) discussed how a large amount of "hidden information" which is often not conveniently accessible to the user could be found in the drawing software of that time period. This is distinctly different from a manual drawing, where all of the drawing's information is visible to the designer. Schleich offered an array of potential hidden information, including exact dimensions of an object, object grouping, invisible attributes, constraints, and design history.

Borrowing affordances from visual programming tools like Grasshopper and drawing software such as AutoCAD, a series of video prototypes have been constructed

to evaluate the usability of a software that focuses on this hidden information. After a heuristic evaluation to analyze the usability of the proposed software seen in the video prototypes, the collected comments and advice have been used to produce a more complex example of a user manipulating this type of bidirectional interface. From this evaluation it has become clear that visibility of system status may be the most commonly relevant heuristic for a dual window interface like this. Each window contains a different representation of the same drawing data, and a user must be able to see clearly which elements of the drawing relate to each script component, and vice versa. Improving the prototypes in this area through a highlighting feature has made them more consistently readable. Recognizing the connections between the parallel representations of script and canvas also offers the user an opportunity to re-interpret their drawing as they move back and forth from manipulation in one window to the other. As automatically generated script components appear due to direct manipulation of the canvas, the user's understanding of their drawing will either coincide or conflict with these script components and their relationships to the drawn elements. Belardi (1994) has stated: "successive "explorations"— i.e., sensations of the same subject in different times and contexts— are never the same, each category is determined and then reclassified an infinite number of times. It is through this endless process of structuring/ destructuring/ restructuring that each person creates his or her creative aptitude."(pp. 20-21). Any conflict between the user's understanding and the automatically generated script could potentially prompt the user to restructure – either altering the script or even their own understanding, in order to reach a better union between the two models. The clarity of the connections between the canvas drawing and its script representation has also opened a

conversation about this interface as a teaching tool for novice users of algorithmic design software, as they may be able to learn from the connections between script and canvas after creating a simple drawing using direct manipulation.

This movement between two representations of a drawing may be a tool for re-interpretation in the same way that Schon (1983) states that drawing and talking are parallel means of designing. (p.80) In his discussion of reflective design, Schon described a design review conversation between a studio master (named "Quist") and an architectural student who is presenting a set of drawings (named "Petra"): "The verbal and non-verbal dimensions are closely connected. His [Quist's] words are obscure except insofar as Petra can connect them with the lines of the drawing. His talk is full of dychtic utterances- "here," "this," "that" – which Petra can interpret only by observing his movements… We must reconstruct Quist's pointing and drawing, referring to the sketches which accompany the transcript." (p. 81) When separated, Quist's verbal critique of Petra's drawings and the visual representation of the drawings themselves are less powerful than when they are connected by Quist's pointing and gestures. This is indirectly similar to the way a software that offers bidirectionality between dual representations of a drawing would need to make clear connections between the two models. In turn, these connections could push the design forward by encouraging a dialogue of reflection and re-interpretation. Another comparison between dialogue and the sketching process comes from Robbins (1997): "The drawing is used by architects in their own thinking and design process as a kind of internal conversation and as a way to record, test and reflect on a design." (p. 32) Information that is hidden or obscured in WYSIWIG drawing programs could be seen as a hindrance to this dialogue, as some of

the hidden information could be important and useful in the designer's creative process. The video prototypes that have been produced seek to show how this bidirectional interface could improve the accessibility of this hidden information.

The bidirectional interface that has been proposed may offer seamless and fluid movement between two representations of the drawings in script and canvas, but it seems that the process of producing drawings in this interface would not be fast or particularly efficient without an expert user. However, this does not invalidate the usefulness of the interface, as it should be viewed as a design tool that seeks to aid users in gaining new insights or ideas by re-interpreting their original drawing. The final video prototype involves a perspectival drawing, which is a context that strays somewhat from the original intent of producing quick sketches or diagrams early in an architectural design. Perspectival sketching can be useful for architects, but is less commonly recognized as a widely used device at this stage of design. The motivation for this content was to showcase a user manipulating their script to apply a series of constraints to their drawing, and to see these constraints cause the drawing to react differently to direct manipulation in the canvas. As the user modifies the constraints applied to their drawing, they are exercising a process of problem definition. As Donald Schon (1985) makes clear, there is "a problem in finding the problem" at these early stages of design. (p. 129) As this exercise takes place, the actions in *Perspectival Drawing* can also be viewed as a form of *re-interpretation*. As the user changes the drawing, they make adjustments to its script definition without abandoning its original format. This is comparable to the way trace paper can be used in traditional media, creating semi-transparent overlays that allow for new drawings that are different and further developed, but are still inspired by the

previous layer underneath. Schon also describes how an effective designer “seeks to discover the particular features of his problematic situation, and from their gradual discovery, designs an intervention.” (p.129) This bidirectional software could allow for this process as the user repetitively adjusts the definition of their drawing in script form, and tests the effects of this change in the canvas. In *Perspectival Drawing*, this type of repeated adjustment ends with the user establishing and manipulating a set horizon line for their drawing. By also taking advantage of conventions from parametric modeling software, these adjustments are done in *Perspectival Drawing* with much less effort than what could be required using pen and paper.

Future work in this area would involve applying the lessons learned from heuristic evaluation to the development of a working software prototype. This prototype could be useful for user studies, to discover the benefits that this bidirectional tool may offer to users over the single direction tools that have been seen in programs such as Grasshopper and Dynamo. These studies would also be useful to find and manage the real problems that will occur due to errors, scalability, and readability as scripts grow to include great numbers of components. Most importantly, these user studies could be used to show how encouraging constant re-interpretation of a preliminary design sketch in this bidirectional platform could have a positive effect on the insight a designer gains from the act of completing the sketch.

REFERENCES

Belardi, P. (2014). *Why Architects Still Draw*. Cambridge, MA: The MIT Press.

Belcher, D. (2013). *Sketching Dynamic Geometry* [Video file]. Retrieved June 21, 2016, from vimeo.com/53042541

Bilda, Z., & Demirkan, H. (2002). An insight on designers' sketching activities in traditional versus digital media. Retrieved June 21, 2016, from yoksis.bilkent.edu.tr/pdf/files/10.1016-S0142-694X(02)00032-7.pdf

Coates, P. (2010). *Programming Architecture*. New York: Routledge.

Cook, P. (2008). *Drawing: The Motive Force of Architecture*. Chichester, England: Wiley.

Gross, M. (1985). Design as Exploring Constraints. Retrieved August 3, 2016, from depts.washington.edu/dmgftp/publications/pdfs/gross_thesis.pdf.

Gross, M., & Do, E. Y. (1996). Demonstrating the electronic cocktail napkin. *Conference Companion on Human Factors in Computing Systems Common Ground - CHI '96*. doi:10.1145/257089.257092

Gross, M., Do, E. Y., Johnson, G., & Hong, J. (2009). Computational Support for Sketching in Design: A Review. *Foundations and Trends in Human–Computer Interaction*. Retrieved June 30, 2016, from code.arc.cmu.edu/archive/upload/1100000013_Johnson.0.pdf

Ibrahim, R. *Comparison of CAD and Manual Sketching Tools for Teaching Architectural Design*. Universiti Putra Malaysia, 2014, from researchgate.net/publication/259841007_Comparison_of_CAD_and_manual_sketching_tools_for_teaching_architectural_design

Introducing Autodesk FormIt [Video file]. (2012, November 25). Retrieved July 23, 2016, from youtube.com/watch?v=M-4MH64pnWw

Robbins, E., & Cullinan, E. (1994). *Why architects draw*. Cambridge, MA: MIT Press.

Schleich, R., & Durst, M. (1994). Beyond WYSIWYG: Display of Hidden Information in Graphics Editors. *Computer Graphics Forum, 13*(3), 185-194. doi:10.1111/1467-8659.1330185

Schön, D. (1983) *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic.

Spiesser, J., & Kitchen, L. (2004). Optimization of html automatically generated by wysiwyg programs. *Proceedings of the 13th Conference on World Wide Web - WWW '04*. doi:10.1145/988672.988720

Stiny, G. (2006). *Shape: Talking about seeing and doing*. Cambridge, Mass: MIT Press.

Sutherland, I. (2007, November 17). *Ivan Sutherland : Sketchpad Demo* [Video file]. Retrieved June 30, 2016, from youtube.com/watch?v=USyoT_Ha_bA

Woodbury, R. (2010). *Elements of parametric design*. London: Routledge.

Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). Research Through Design as a Method for Interaction Design Research in HCI. *Carnegie Mellon University Research Showcase @ CMU*. Retrieved June 21, 2016, from repository.cmu.edu/cgi/viewcontent.cgi?article=1041&context=hcii

VIDEO PROTOTYPES

Rodgers, J. (2016, July 27). Line Manipulation. Retrieved August 03, 2016, from vimeo.com/175910406

Rodgers, J. (2016, July 27). Adjacency Diagrams. Retrieved August 03, 2016, from vimeo.com/175910041

Rodgers, J. (2016, July 27). Space Planning. Retrieved August 03, 2016, from vimeo.com/176980841

Rodgers, J. (2016, July 27). Perspectival Drawing. Retrieved August 03, 2016, from vimeo.com/175910501

BIBLIOGRAPHY

Banks, M., & Cohen, E. (1990). Realtime Spline Curves from interactively Sketched Data. *ACM SIGGRAPH Computer Graphics*. Retrieved June 30, 2016, from dl.acm.org/citation.cfm?id=91425

Carpo, M. (2011). *The Alphabet and the Algorithm*. Cambridge, MA: The MIT Press.

Carpo, M. (2013). *The Digital Turn in Architecture*. West Sussex, UK: Wiley.

Eggli, L., Hsu, C., Bruderlin, B., & Elber, G. (1997). Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*. Retrieved June 30, 2016, from pages.cpsc.ucalgary.ca/~samavati/cpsc789/paper_list/hsu-eggli-cad.pdf

Fish, J. (1990)*Amplifying the Mind's Eye: Sketching and Visual Cognition*. 1st ed. Vol. 23. Boston: MIT, 117-126. jstor.org/stable/1578475?seq=1#page_scan_tab_contents

Gero, J. S., & Purcell, A. T. (2006). Drawings and the design process. *DST: Design Studies*. Retrieved June 30, 2016, from mason.gmu.edu/~jgero/publications/1998/98PurcellGeroDesignStudies.pdf

Hammond, T. and Davis, R. (2002). Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. rationale.csail.mit.edu/pubs/hammond/SSS902Hammond.pdf

Igarashi, T., Kawachiya, S., Tanaka, H., & Matsuoka, S. (1998). Pegasus. *CHI 98 Conference Summary on Human Factors in Computing Systems - CHI '98*. doi:10.1145/286498.286511

Kay, A. (2009, January 18). *Alan Kay Demos GRaIL* [Video file]. Retrieved June 30, 2016, from youtube.com/watch?v=QQhVQ1UG6aM&feature=related

Lin, J., Newman, M., Hong, J., & Landay, J. (2002). DENIM: An Informal Sketch-based Tool for Early Stage Web Design. Retrieved June 30, 2016, from cs.cmu.edu/~jasonh/publications/aaai2002-denim-final.pdf

Lissitzky, El. *Prounenraum (Proun Room)*. 1923. Yale University Art Gallery, New Haven, CT.

March, L., & Steadman, P. (1971). *The Geometry of Environment*. London: RIBA.

McConnell, C. *Designing with Parametric Sketches*. Calgary: Mechanix Design Solutions. Retrieved June 30, 2016, from cadlinecommunity.co.uk/hc/en-us/articl es/201699182-Autodesk-Inventor-D esigning-with-Parametric-Sketches

McCullough, M., & Mitchell, W. J. (1990). *The Electronic Design Studio*. Cambridge, MA: The MIT Press.

McGrath, B., & Gardner, J. (2007). *Cinemetrics: Architectural Drawing Today*. West Sussex, UK: Wiley.

Negroponte, N. (1975). *Soft Architecture Machines*. Cambridge, MA: The MIT Press.

Rowe, P. *Design Thinking*. (1987) Cambridge, Mass: MIT Press.

Schutze, M., Sachse, P., & Romer, A. (2003, May). Support value of sketching in the design process. *Research in Engineering Design*. Retrieved June 30, 2016 from link.springer.com/article/10.1007/s00163-002-0028-7

Sezgin, M. Sketch Based Interfaces: Early Processing for Sketch Understanding. Cambridge, Mass: MIT Press, Retrieved June 30, 2016, from iui.ku.edu.tr/sezgin_publications/2007/Sezgin-SIGGRAPH-2007.pdf

Stevens, P. (1980) *Handbook of Regular Patterns: An Introduction to Symmetry in Two Dimensions*. Cambridge, Mass: MIT Press.

Szalapaj, P. (2001). *CAD Principles for Architectural Design*. Woburn, MA: Architectural Press.

Zeleznik, R., Herndon, K., & John, H. (1996). SKETCH: An Interface for Sketching 3D Scenes. *Computer Graphics Proceedings*. Retrieved June 30, 2016, from cs.brown.edu/~jfh/papers/Zeleznik-SAI-1996/paper.pdf