QuickDraw: Improving Drawing Experience for Geometric Diagrams

Salman Cheema
University of Central Florida
Orlando, FL
salmanc@cs.ucf.edu

Sumit Gulwani
Microsoft Research
Redmond, WA
sumitg@microsoft.com

Joseph J. LaViola Jr.
University of Central Florida
Orlando, FL
jjl@eecs.ucf.edu

ABSTRACT

We present QuickDraw, a prototype sketch-based drawing tool, that facilitates drawing of precise geometry diagrams that are often drawn by students and academics in several scientific disciplines. Quickdraw can recognize sketched diagrams containing components such as line segments and circles, infer geometric constraints relating recognized components, and use this information to beautify the sketched diagram. Beautification is based on a novel algorithm that iteratively computes various sub-components of the components using an extensible set of deductive rules. We conducted a user study comparing QuickDraw with four state-of-the-art diagramming tools: Microsoft PowerPoint, Cabri II Plus, Geometry Expressions and Geometer's SketchPad. Our study demonstrates a strong interest among participants for the use of sketch-based software for drawing geometric diagrams. We also found that QuickDraw enables users to draw precise diagrams faster than the majority of existing tools in some cases, while having them make fewer corrections.

Author Keywords

Sketch-based Interfaces; Sketch Recognition; Sketch Beautification; Geometry Constraint Solving

ACM Classification Keywords

G.4 Mathematical Software: User Interfaces; H.5.2. User Interfaces: Interaction Styles

General Terms

Algorithms, Experimentation, Human Factors, Measurement

INTRODUCTION

Students and teachers in scientific disciplines often have to draw very precise diagrams. Drawing such diagrams is very time consuming and cumbersome, motivating the development of software to aid diagramming. For geometric drawings, popular tools include Geometer's Sketchpad [6], Cabri II Plus [3], and Geometry Expressions [7]. Additionally, a growing number of people use Microsoft Office products such as PowerPoint [16] to draw diagrams, due to the availability of several drawing templates for common shapes within the Microsoft Office suite.

CHI'12, May 5–10, 2012, Austin, Texas, USA.

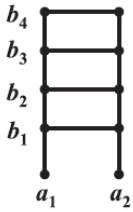


Figure 1. Source : NCERT Mathematics Book, Grade 11, Chapter 2

Figure 1 shows an example of a geometry diagram. Here, a user wants to draw a ladder with four rungs. Notice that a_1 and a_2 are both vertical and have the same length. Both a_1 and a_2 begin and end at the same horizontal level. Additionally, line segments b_1, b_2, b_3, b_4 are horizontal, equidistant, of same length and are all perpendicular to both a_1 and a_2. Also, b_1, b_2, b_3, b_4 begin and end at the same vertical level. It is clear that this deceptively simple diagram would require a fair amount of effort if one was to do the construction precisely, even with existing state-of-the-art tools.

Our research goal is to investigate the use of sketch-based interfaces for drawing precise diagrams since they can provide significant benefits over existing tools. Sketch-based interfaces closely mimic pen and paper and let a user draw very naturally. They can also lower the input complexity of a diagram construction tool. We believe that by leveraging techniques from areas of sketch recognition, machine learning, and geometry constraint solving, it is possible to build a sketch-based drawing tool that can reduce the difficulty of precisely drawing a given diagram and thus enhance the productivity of users in various scientific fields. To this end, we have developed QuickDraw, a prototype sketch-based diagram drawing tool, that allows a user to sketch and beautify a given diagram. QuickDraw can process the input sketch to recognize components of the diagram and the geometric relationships between them. We examined various elementary mathematics and science text books to come up with a set of geometric constraints that can be used to beautify a large subset of diagrams found in the appropriate books. The set of inferred constraints is used to beautify the input sketch. Figure 2 provides an example of such a beautification, where QuickDraw converts the user's sketch on the left into the precise diagram on the right.

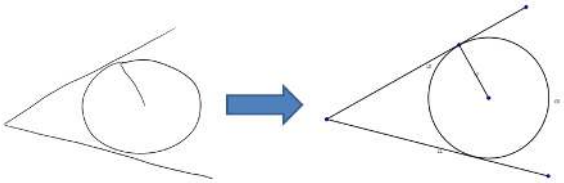


Figure 2. A diagram beautified by QuickDraw. The input sketch is shown on the left, while the beautified diagram is shown on the right. If a user was to construct this diagram in a traditional diagram construction tool, it would require a great amount of precision to ensure the tangent constraints.

We conducted a usability study to compare drawing performance in QuickDraw with four state-of-the-art diagram construction tools. It should be noted that QuickDraw is fundamentally different from the other tools in our study. The others tools rely on the use of pre-defined templates to let a user construct a diagram that is a close approximation of what the user has in mind. QuickDraw allows a user to express intent by sketching the diagram itself. It then tries to understand the user's intent through sketch and constraint recognition. Using its unique beautification algorithm, it can generate a mathematically precise diagram as opposed to a close approximation with other tools.

RELATED WORK

In general, sketch recognition is a hard problem due to the complexity and variation in notation in different disciplines. User perception of sketch recognition has been explored by Wais, Wolin, and Alvarado [19], yielding useful insights for designers of sketch-based interfaces. CogSketch [5] is an open domain sketch analysis system which enables conceptual labeling of sketches. Hammond et al., have developed LADDER [9] which lets users specify recognition rules for sketch primitives and some high level semantics in text form. LADDER can generate domain-specific recognizers from such text descriptions. MathPad^2 [14] is a domain independent system that allows users to associate written mathematics with a sketched diagram to perform animation. None of these systems focus on enabling a user to easily sketch a precise diagram.

Techniques for sketch beautification have also been explored by several researchers. PaleoSketch [18] explores techniques for recognition of low-level sketch primitives and beautification. It can recognize a large set of sketch primitives, but the beautification aspect of this system is very low level and applies only to individual primitives. Lineogrammer [21] is a modeless diagramming tool that enables users to construct diagrams by interactively beautifying ink strokes. However, it does not leverage the power of geometric constraint reasoning and performs very limited beautification. In contrast to both of these, QuickDraw infers a user's intent by recognizing constraints relating sketched primitive components and uses this information to beautify the entire sketch, instead of just individual components. Igarashi et al., [11] have investigated the use of inferred geometric constraints for interactively beautifying a sketched diagram. QuickDraw offers several benefits

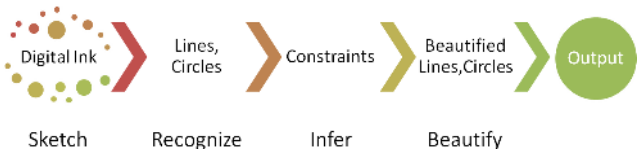


Figure 3. The flow of information within QuickDraw. A user's sketch is recognized and geometric constraints are inferred, which are then used for beautification, resulting in a precisely rendered diagram.

in relation to their approach. First, we support an extensible set of diagram components including lines and circles, and an extensible set of inferred constraints between components (earlier work by Igarahi et.al is limited to line segments). Second, we employ a novel beautification algorithm to beautify diagrams that are either very hard to draw with the earlier approach or cannot be drawn altogether.

Our beautification algorithm uses an extensible set of deductive rules to solve geometric constraints for model construction. There are several existing approaches to constraint solving. The goal here is to find a configuration for a set of geometric objects that satisfy a given set of constraints between the geometric elements [2]. A variety of techniques have been proposed including logical inference and term rewriting [1], numerical methods [17], algebraic methods [13], and graph based constraint solving [2]. These techniques either require some symbolic reasoning or some form of search. QuickDraw's beautification algorithm is also a form of geometry constraint solving. However, it is much simpler because it deals with saturated constraints, which obviate the need for any sophisticated symbolic reasoning or search. This makes our beautification technique more efficient and hence real-time.

With regards to geometry education, Jiang et al., have developed PenProof [12] which is a sketch-based geometry theorem proving system. Its focus is on correlating steps in a hand-written proof with aspects of the sketch. Another recent system is IIPW [15], which is a framework of tools for interactive geometry editing on a whiteboard. While this tool can beautify diagrams using the Line Intersection Method, it relies heavily on pre-defined templates to specify geometric constraints. Gulwani et.al. have developed a method for automatically synthesizing ruler/compass based geometry constructions starting from declarative specifications [8]. Their method starts out by translating the declarative specification into a model using numerical search. In principle, declarative specification can be replaced by an easier sketch-based specification, while model construction can be performed using our beautification algorithm.

QUICKDRAW : AN OVERVIEW

In its current form, QuickDraw can recognize and beautify diagrams containing line segments and circles. Figure 3 shows the flow of information among the subsystems of QuickDraw. A user sketches a diagram using a stylus on a tablet computer. QuickDraw recognizes the components of the sketch and infers the geometric relationships between them. Using this information, it beautifies the sketch, resulting in a precisely rendered diagram from approximate sketch input.

Constraints recognized by QuickDraw	
Applicable To	Constraint
Line Segments	Vertical line segment Horizontal line segment Collinear line segments Parallel line segments Perpendicular line segments Equidistant line segments Touching line segments Intersecting line segments Line segments with same length Line segments with endpoint(s) at same horizontal level Line segments with endpoint(s) at same vertical level
Circles	Circles with same radius Concentric circles Circles touching at their circumference Intersecting Circles Circle passing through the center of another circle
Circles & Line Segments	Line segment tangent to circle Line segment intersecting circle Line segment passing through center of circle Line segment touching circumference with an endpoint Line segment touching circle center with an endpoint

Table 1. Table showing all the constraints that can be recognized by QuickDraw's inference subsystem. The constraints can be grouped into three categories depending on the type of sketch components that are related by them.

Recognition

A sketched diagram forms the input to QuickDraw. Recognition in QuickDraw is a two step process. First, the sketch is parsed into diagram (basic) components. QuickDraw can recognize line segments and circles as components. Our recognition heuristics rely on the IStraw cusp finding algorithm [20]. A cusp is a region of high curvature in a digital ink stroke. An ink stroke corresponding to a circle has two cusps, its starting and ending points, both of which should be close together, resulting in a closed shape. A circle also has approximately uniform curvature about its centroid. To measure curvature in an ink stroke, QuickDraw applies a threshold to the standard deviation in the radius of an ink stroke. Once an ink stroke is classified as a circle, its centroid is assigned as the center of the circle. The diameter of the circle is assigned as the average of the width and height of the ink stroke's bounding box. Similarly, an ink stroke corresponding to a line segment has two cusps, and is approximately straight. To check if an ink stroke forms an approximately straight line, we use a measure similar to the approach in IStraw [20]:

$$Linearity = \|1.0 - \frac{\sum_{i=1}^{n-1} \|p_i, p_{i+1}\|}{\|p_1, p_n\|}\|$$

where p_i is the i^{th} point in the ink stroke. We use a threshold of $Linearity < 0.1$ to detect line segments. Ink strokes containing more than two cusps are broken down into components containing two cusps each, which are then examined using the heuristics described above. Recognized sketch components are assigned a canonical ordering $\mathcal{O}$, from left to right followed by top to bottom. This is done to ensure that a deterministic view of each diagram emerges, independent of the order in which its components were drawn.

Algorithm 1 Beautification Algorithm

$\{\tilde{C}$ = Set of Components; α = Set of Constraints; $\}$
Require: $\tilde{C}$, α
  $\mathtt{A}$:= Set of all sub-components of components in $\tilde{C}$;
  $\mathtt{B} := \emptyset$;
  while $\mathtt{B} \neq \mathtt{A}$ **do**
    if $\mathtt{A} - \mathtt{B}$ contains a sub-component s that is computable from sub-components in $\mathtt{B}$ because of α (using V) **then**
      Compute s;
    else
      s := sub-component from $\mathtt{A}-\mathtt{B}$ with the highest rank.
      Read value of s from sketch.
    end if
    $\mathtt{B} := \mathtt{B} \cup \{s\}$;
    C := parent component of s.
    if C is determined from its sub-components in $\mathtt{B}$ (using U) **then**
      Beautify C; $\mathtt{B} := \mathtt{B} \cup$ Sub-components of C;
    end if
  end while

Beautification

The recognition engine recognizes basic components C (lines and circles). After this, the inference subsystem infers the intended constraints between recognized components (see Table 1 for supported constraints). We now describe a generic and extensible beautification algorithm (Algorithm 1) that takes as input the recognized components and inferred constraints between them, and translates them into a precise geometric drawing whose components satisfy the intended constraints.

Each component C has constituent sub-components s such that it can be uniquely determined after some appropriate subset of its sub-components have been determined. For example, the sub-components of a line segment are its slope, intercept (y-Intercept if slope is not vertical; otherwise x-Intercept), length, and x/y coordinates of the two end points. A line segment can be uniquely determined from the x/y coordinates of the two end points, or alternatively from its slope, intercept and y coordinates of its two end points (if the slope is not vertical). Similarly, the sub-components of a circle are its radius and the coordinates of its center. A circle is uniquely determined if all of its sub-components are known. This knowledge is captured as an extensible set of rules U.

The idea behind defining constituent sub-components for each component is that the constraints between components uniquely identify some of their sub-components. This knowledge is captured as an extensible set of rules V, each of which specifies how to determine the value of some sub-component

from values of some other sub-components under some appropriate constraints. For example, under the constraint that a line L is tangent to a circle C, C.radius can be computed from C.center, L.slope, L.intercept. In particular, C.radius can be computed as the perpendicular distance between C.center and the line determined by L.slope and L.intercept. As another example, under the same constraint that a line L is tangent to a circle C, L.intercept can be computed from L.slope, C.center, and C.radius.

Our beautification algorithm maintains a worklist B that holds the set of all sub-components whose values have been computed. B is initialized to the empty set. The main loop of the beautification algorithm is repeated until B is equal to set A that holds all sub-components of all recognized components. Each iteration of the main loop attempts to identify a sub-component $s \in (\mathtt{A} - \mathtt{B})$ (of some component C) whose value can be computed from the sub-components in B using any of the rules $V : s$ is then added to B. If component C is uniquely determined from its sub-components in B using any of the rules U, then C is beautified and all of its sub-components are added to B.

If no such sub-component s exists, then to maintain progress, the algorithm identifies a sub-component $s \in (\mathtt{A} - \mathtt{B})$ with the highest rank. The rank of a sub-component s of a component C is given by lexicographic ordering on the following tuple:

$$\mathtt{Rank}(s) \;=\; \left(\mathtt{Max}_S \left\{ \frac{\sum_{s' \in S \cap \mathtt{B}} W(s')}{\sum_{s' \in S} W(s')} \right\}, \mathcal{O}(C), \frac{1}{W(s)} \right)$$

An interesting element of the above rank tuple is a weight function W that maps each sub-component type to some score between 0 and 1 and is used to assert the relative importance of knowing some sub-component over another. More specifically, the relative weight ordering reflects the order of observing any visual discrepancies (thereby avoiding the need to edit components unless really required), and also the order of ease of editing components if QuickDraw didn't get it right. For our experiments, we used the following relative weights: (a) Sub-components of a line-segment: x-y coordinates of the two end-points and length (0.5 each), intercept (0.75), slope (1). (b) Sub-components of a circle: x-y coordinates of the two end-points (0.5 each), radius (1).

The first element of the rank tuple identifies a component C that is closest to being determined. This is estimated by computing the maximum of the weighted ratio of the sub-components that are known from among some minimal set of sub-components S of C that uniquely determine the component C. The second element of the rank tuple breaks any ties among C by using the canonical ordering $\mathcal{O}(C)$ assigned to the component at recognition time. The third element of the rank tuple identifies a sub-component s of component C that has the lowest weight. In the future, we plan to refine the rank computation by taking into account the probability of the constraint recognized by the recognition engine and using these probabilities to associate a confidence score with each computed sub-component. The value of the sub-component s with the highest rank is then read off from the sketch. C is then beautified and all of its sub-components are added to B.

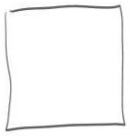


Figure 4. A approximate square sketched in QuickDraw.

The beautification algorithm has two interesting characteristics: robustness and interactive support. The algorithm is robust due to the powerful deductive reasoning enabled by an extensible set of rules U and V over a saturated set of constraints returned by the inference engine. This allows it to correctly beautify diagrams when the recognition engine misses out on some constraints. The algorithm also allows for interactive drawing. The main loop of the algorithm can be run in an incremental fashion after adding sub-components of any new component sketched by the user to A and updating the set of constraints.

Example: A Sketched Square

Given the sketched square in Figure 4, QuickDraw should ideally infer the following constraints between four recognized line segments:

1. Two line segments are horizontal and two are vertical.
2. The horizontal line segments are parallel, and are both perpendicular to the vertical line segments.
3. The vertical line segments are parallel, and are both perpendicular to the horizontal line segments.
4. All the line segments in the sketch form a connected path, and are all equal in length.
5. The perpendicular distance between horizontal line segments is the same as that between vertical line segments.

Beautification now proceeds as follows. (i) After computing the slope of all the line-segments, the algorithm reads off the x-y coordinates of the top-left corner and the y-coordinate of the bottom-left corner from the sketch and then beautifies the left line-segment. (ii) Next, the algorithm computes the y coordinate of the top-right corner from the y coordinate of the top-left corner (because of the top line-segment having horizontal slope constraint), and then the x coordinate of the top-right corner from the two left corners (because of the equal length constraint between the top and left line-segments), and then beautifies the top line-segment. (iii) In a manner similar to the previous case, the algorithm computes the y coordinate of the bottom-right corner from the y coordinate of the bottom-left corner (because of the bottom line-segment having horizontal slope constraint), and then the x coordinate of the bottom-right corner from the two left corners (because of the equal length constraint between the bottom and left line-segments), and then beautifies the bottom line-segment. However, suppose that the inference engine failed to infer any equal length constraint involving the bottom line-

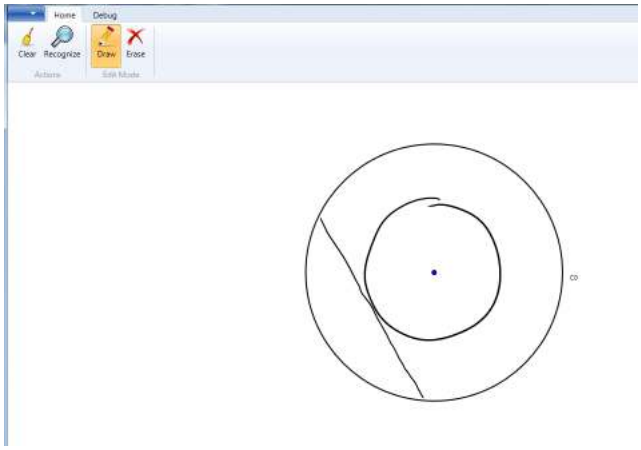


Figure 5. A diagram sketched in QuickDraw. In this case, the user has sketch the diagram incrementally. The outer circle has already been drawn, recognized and beautified. The inner circle and its tangent have just been sketched and await recognition.

segment. The algorithm can still compute the x coordinate of the bottom-right corner from the x coordinate of the top-right corner (because of the right line-segment having vertical slope constraint). However, suppose that the inference engine also failed to infer the vertical slope constraint for the right line-segment. The algorithm can still compute the slope of the right line-segment from the slope of the top line-segment (because of the perpendicular constraint between the top and right line-segments) followed by computing the intercept of the right line-segment from the x coordinate of the top-right corner. The algorithm can then compute the x coordinate of the bottom-right corner from the two top corners (because of the equal length constraint between the right and top line-segments). These instances of missing constraints highlight the robustness of our beautification algorithm, which is able to make up for the missing constraints by making effective use of other (logically equivalent) constraints.

User Interaction

Figure 5 shows the user interface for QuickDraw. Users can sketch a drawing in two ways. They may sketch the entire drawing and then trigger recognition by hitting the 'Recognize' button. However this method does not provide instant recognition feedback. Users can also sketch incrementally (i.e., they sketch one component at a time), hitting 'Recognize' after each step. QuickDraw supports both of these sketching methodologies. For a future version, we are planning to add support for automatically rendering a component as soon as it gets uniquely determined.

It must be kept in mind that the goal of QuickDraw is to come up with a precise diagram, given a user's sketch as input. However, it is possible for the recognition and constraint inference engine in QuickDraw to make a mistake. Some are minor and can be rectified by manipulating the positions of beautified sketch components. Each sketch component has one or more 'edit' points which can be used to manipulate its position. These edit points are highlighted in blue on the screen. For a line segment, these are its endpoints. For a circle, its center can be used to move it. A user can hold the stylus over an edit point for a few seconds to select it. Once selected, the point can be moved with the stylus to wherever the user requires.

Figure 6. A photograph showing the tablet computer used for our experiment.

Using QuickDraw, a user can draw any diagram containing lines, polygons and circles. However, we cannot beautify all such drawings since precise reasoning of geometric constraints reduces to reasoning about non-linear arithmetic, which is a hard problem. Specifically, we cannot beautify drawings that require construction of intermediate objects or drawing of arbitrary angles (e.g., drawing a regular pentagon requires either drawing an angle of 108 degrees, or drawing several temporary objects). The current version of QuickDraw also has limited editing capabilities. Entire line segments cannot be moved. A user must move both endpoints separately. QuickDraw also does not allow users to resize circles. For more major inference errors, a user can remove the offending sketch component by going into 'Erase' mode and then redrawing it. Note that QuickDraw in its current form is a prototype and was engineered to focus primarily on the recognition, inference and beautification aspects of the system. As of yet, we have not invested fully into adding enhanced editing capabilities. However, one of the goals of our usability study was also to investigate diagram drawing strategies which would yield insights into the type of editing functionality required by QuickDraw.

USABILITY STUDY

We conducted a user study to compare drawing performance in QuickDraw with four state-of-the-art diagramming tools: Cabri II Plus, Geometry Expressions, Geometer's SketchPad, and Microsoft PowerPoint. The first three tools were chosen because they are very popular and for their capabilities in aiding a user in geometrical drawings. Microsoft PowerPoint was chosen due to its widespread use and convenience for drawing diagrams. Our primary objective in this study was to compare QuickDraw with traditional WIMP (Windows, Icons, Menus, Pointers) based drawing tools. A related question is how does the drawing performance scale with diagram difficulty for each tool. Lastly, we wanted to get user feedback concerning the use of sketch-based interfaces for diagram drawing.

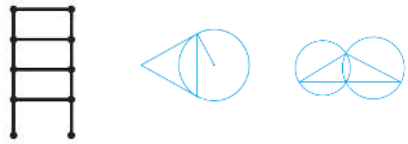
Easy Difficulty	Medium Difficulty	Hard Difficulty
		

Table 2. This figure shows the diagrams that were drawn by participants in our study. They have been grouped by difficulty. Difficulty for a diagram was determined by (i) number of components in the diagram and (ii) number of constraints that a person would have to ensure while drawing it. We conducted a pilot study with two volunteers that allowed us to verify the difficulty for these diagrams.

Subjects and Apparatus

We recruited 19 participants (17 male and 2 female) from the University of Central Florida for participation in the study. The ages of participants were between 18 and 37 years. They were asked to draw nine diagrams with each of the diagramming tools. Each participant took 90-120 minutes to complete the experiment and was paid $10 for their time. Only one participant was left-handed. Eleven participants had used a computer with stylus input before. Nine participants reported that they often used a computer for drawing diagrams. None of the participants had used Cabri II Plus, Geometer's Sketchpad or Geometry Expressions before the experiment. On the other hand, only one person was not familiar with Microsoft PowerPoint. Figure 6 shows the experiment setup for our user study.

The experiment was conducted on a Lenovo ThinkPad X220 multi-touch tablet computer equipped with an Intel Core-i7-2620M processor and four gigabytes of memory. The screen resolution was set at 1366x768 pixels. We disabled multi-touch interaction on the tablet which was placed on a table for the experiment. Participants sat down and drew each diagram either with the mouse or the stylus, depending on the tool being used. Each participant's drawing session was recorded by screen capture via the Fraps utility. For analysis purposes, we divided the diagrams used in the experiment into three categories, based on drawing difficulty (see Table 2). The difficulty of a diagram was initially determined based on the number of constraints that a user would have to ensure while drawing that diagram. The assigned order of difficulty was tested and confirmed during a pilot study with two volunteers.

Experiment Procedure

Each participant was asked to fill out a pre-questionnaire, which collected demographic information. The participant was given a tutorial of each of the diagramming tools to be used in the experiment. They were shown how to draw basic diagram components such as line segments, circles, triangles, and polygons. They were also shown how to edit each component's position and size. Lastly, they were shown how to erase diagram components in each tool. For QuickDraw, participants were explicitly told they could either sketch the entire diagram and then trigger recognition, or they could do recognition incrementally by sketching and recognizing parts of it. After the tutorial, the participant was given a set of three practice diagrams (see Table 3). They were asked to use each tool to draw all three practice diagrams. This helped familiarize the participants with each tool. Participants were also asked to be very precise in their drawing.

Table 3. Practice diagrams used to familiarize participants with each diagramming tool.

After the practice session, participants were asked to draw nine diagrams with each tool. The drawing difficulty of each diagram was not communicated to participants. Each diagram had written instructions about relationships that participants needed to ensure while drawing. Participants were told that they should complete each diagram to ensure that the written instructions were satisfied as closely as possible. Participants were then told they had three minutes to complete each diagram. This time limit was decided during our pilot study during which all volunteers were able to complete all diagrams in under three minutes. The order in which diagramming tools were used was randomized for each participant. For each diagramming tool, the order in which participants drew diagrams was also randomized. Table 2 shows the diagrams used in the experiment. The entire session was recorded via screen capture. Furthermore, a proctor monitored each participant throughout the session and noted the time taken to complete each diagram with each tool. The number of editing operations performed by participants was also noted down for each diagram and tool combination. Editing operations included copy-paste, manipulating positions of diagram elements, and erasing diagram elements. At the end of the experiment, participants were given a post-questionnaire, allowing them to give feedback about their experience with each tool.

METRICS

We recorded two quantitative metrics for each diagram drawn by a participant:

- *Completion Time* : Time taken to complete a diagram in a particular software tool.
- *Edits* : Number of editing operations performed in a software tool to complete a diagram.

These metrics allowed us to compare the tools in terms of how easily a user can complete a given diagram. Additionally, we also wanted to measure QuickDraw's failure rate. As mentioned before, QuickDraw can fail due to incorrect recognition of constraints. In such cases, users had to make alterations to the beautified diagram. Minor errors were fixed by editing the positions of beautified diagram components. Se-

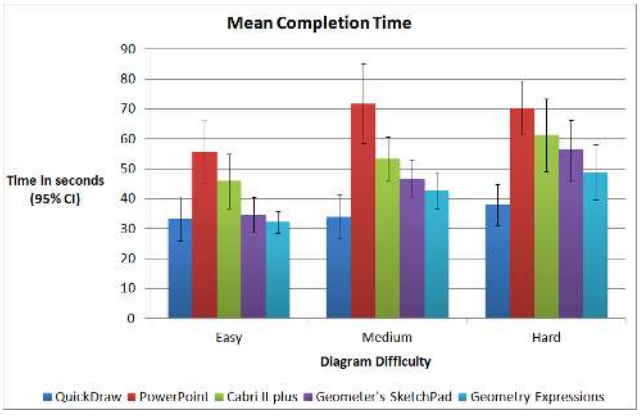


Figure 7. Mean time to complete a given diagram in each of the different software tools examined in our study.

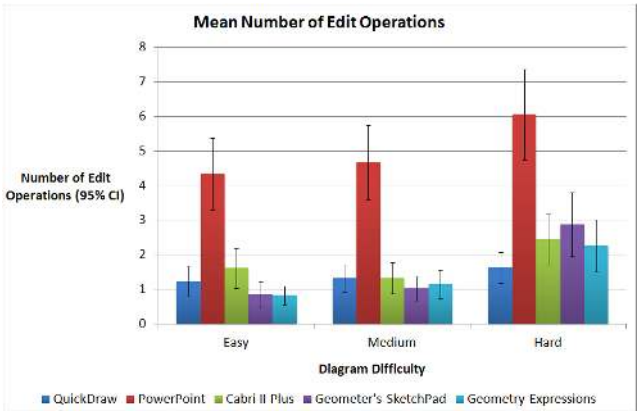


Figure 8. Mean number of editing operations to complete a given diagram in each of the different software tools examined in our study.

vere errors required a user to erase and redraw part or all of the diagram. Specifically, we were interested in measuring the probability that a user would not be able to complete a diagram in the allotted time due to an error by QuickDraw's constraint inference subsystem.

Difficulty	Easy	Medium	Hard
Failure Rate	0%	10.526%	12.281%
Overall Failure Rate 7.602%			

Table 4. Failure rate for diagrams in each category of drawing difficulty.

For this purpose, whenever a participant was not able to complete a diagram with QuickDraw in the allotted time due to a severe error, the proctor made a note of it. By analyzing the session videos recorded for each participant, we conclude that these instances of failure were primarily the result of incorrect constraint inference. They become severe when the inference system infers a false constraint (as opposed to missing a constraint). Table 4 shows the rate of occurrence of failure instances for each category of diagram difficulty.

It should be noted that the other tools in our study do not utilize any sort of diagram recognition or beautification. Instead, they require a user to construct a precise diagram through use of predefined templates. QuickDraw on the other hand relies on correct recognition of a sketched diagram and the constraints contained therein to produce a precise diagram. As participants were given 3 minutes to complete a diagram, it is possible for them to fail to complete a diagram in QuickDraw due to inference errors. Including such instances of failure would inflate some data points for QuickDraw to a constant 180 seconds. As these cannot be directly compared to the completion times in the other tools in our study, we chose to disregard such instances of failure where participants were unable to complete a given diagram in QuickDraw in the allotted time.

FINDINGS

Analysis of Metrics

Figure 7 shows the mean time to complete a given diagram in each of the software tools tested in our study. Similarly, Figure 8 shows the mean number of editing operations required to complete a given diagram. The data in both figures are grouped by drawing difficulty. For each category of difficulty, we used a 5-way repeated measures ANOVA analysis to test for significant differences in completion time and edits between QuickDraw and other tools. If we found significant differences with respect to either mean completion time or mean number of edits, we conducted a post-hoc analysis with pairwise t-tests to find the interesting differences. In our post-hoc analysis, we controlled the chance of Type I errors by using a Holm's Sequential Bonferroni adjustment with four comparisons at $\alpha = 0.05$ [10].

Diagrams with Easy Difficulty

We found significant differences between QuickDraw and the other tools for both completion time ($F_{4,14} = 7.692, p < 0.001$) and number of editing operations ($F_{4,14} = 22.826, p < 0.001$). Using pairwise t-tests, we found that the mean completion time for an easy diagram in QuickDraw is less than Microsoft PowerPoint ($t_{18} = -3.325, p < 0.0125$). Due to the Bonferroni adjustment, the difference in completion time between QuickDraw and Cabri II Plus ($t_{18} = -2.570, p = 0.019$) was not found to be significant. However, the small p-value suggests that this difference may be significant with more participants. We found no significant differences for completion time between QuickDraw and Geometer's SketchPad or Geometry Expressions. We also found that participants performed fewer edits in QuickDraw than in Microsoft PowerPoint ($t_{18} = -5.41, p < 0.0125$). There were no significant differences in the number of edits performed in QuickDraw and in either Cabri II Plus , Geometer's SketchPad or Geometry Expressions.

Diagrams with Medium Difficulty

Once again, we found significant differences between QuickDraw and the other tools for both completion time ($F_{4,14} = 15.553, p < 0.001$) and number of edits ($F_{4,14} = 29.003, p < 0.001$). The results of pairwise t-tests indicate that completion time for a medium difficulty diagram in QuickDraw is less than in Microsoft PowerPoint ($t_{18} = -6.667, p < 0.0125$), Cabri II Plus ($t_{18} = -4.586, p < 0.0167$) and Geometer's SketchPad ($t_{18} = -3.002, p < 0.025$). There was no significant difference between QuickDraw and Geometry Expressions in terms of the time taken to

complete a medium difficulty diagram. Additionally, participants performed fewer edits in QuickDraw than in Microsoft PowerPoint ($t_{18} = -5.691, p < 0.0125$). There were no significant differences in the number of edits performed in QuickDraw and in either Cabri II Plus, Geometer's SketchPad or Geometry Expressions.

Diagrams With Hard Difficulty

Within the hard difficulty category, we found significant differences between QuickDraw and other tools for both completion time ($F_{4,14} = 8.665, p < 0.001$) and number of edits ($F_{4,14} = 21.208, p < 0.001$). We found participants took less time to complete hard diagrams in QuickDraw than in Microsoft PowerPoint ($t_{18} = -8.781, p < 0.0125$), Cabri II Plus ($t_{18} = -3.489, p < 0.0167$) or Geometer's SketchPad ($t_{18} = -3.222, p < 0.025$). There was no significant difference in completion time between QuickDraw and Geometry Expressions for hard to draw diagrams. Participants also performed fewer edits in QuickDraw than in Microsoft PowerPoint ($t_{18} = -6.788, p < 0.0167$). Due to the Bonferroni adjustment, the difference in edit operations between QuickDraw and Cabri II Plus ($t_{18} = -2.378, p = 0.029$) and between QuickDraw and Geometer's SketchPad ($t_{18} = -2.541, p = 0.020$) was insignificant. There was no significant difference in the number of editing operations performed in QuickDraw and Geometry Expressions.

Factors Affecting Completion Time

Completion time, as defined earlier, is a broad metric, which glosses over a lot of detail. Several factors influence the completion time for a given diagram in a particular tool. First, completion time is affected by the order in which a tool is used. Cabri II Plus, Geometer's SketchPad and Geometry Expressions all have similar usage scenarios. It is possible for users to learn aspects of one of them while using another. This may have happened in our study despite our best efforts to randomize the order of tools and diagrams for each tool. We also noticed variation in drawing strategy among people. Some participants took extreme care to ensure the constraints indicated for each diagram while others did not. Some people drew quickly, whereas others took their time and tried to draw carefully. In QuickDraw, it is possible to complete a diagram in two ways. A user can sketch the entire diagram and then trigger recognition. Alternatively, the diagram can be drawn incrementally (i.e., sketching individual components one at a time and triggering recognition after each step). Depending on the strategy used, completion time can vary greatly within QuickDraw. We found that a large majority of participants drew the diagrams incrementally. While it is critical to support both modes of drawing, it should be noted that QuickDraw's best performance is achieved when the diagram is drawn in a non-incremental manner. Lastly, incorrectly inferred constraints can increase the completion time for diagrams drawn in QuickDraw. As none of the other tools does any recognition, inference errors inflate the completion time for QuickDraw by forcing users to perform extra editing operations.

Questionnaire : All Tools		
No.	Question	Response Type
1.	I was given a sufficient introduction to each tool.	7-point Likert
2.	Please rate your overall reaction to each tool.	7-point Likert
3.	Please rate your drawing performance with each tool.	7-point Likert
4.	I was able to draw a given diagram in a relatively straightforward manner.	7-point Likert
5.	Was it difficult to correct your mistakes?	7-point Likert
6.	I was confused by the interface.	7-point Likert
7.	The system had adequate capabilities to allow me to complete a given diagram in an easy manner.	7-point Likert
8.	Please explain your answer to Q7.	free response
9.	Please rate the different tools from best to worst.	ranking
Questionnaire : QuickDraw Specific		
1.	QuickDraw was able to correctly understand the sketched diagram most of the time.	7-point Likert
2.	Using QuickDraw would enable me to draw diagrams more quickly.	7-point Likert
3.	I would find QuickDraw useful in my work.	7-point Likert
4.	Would you like to use a sketch-based interface for drawing diagrams in the future?	Yes/No
5.	List the most positive aspects of QuickDraw's interface.	free-response
6.	List the most negative aspects of QuickDraw's interface.	free-response
7.	Please suggest any new features that would enhance the usefulness of QuickDraw.	free response

Table 5. Table showing the post-questionnaire used in our study. For the 7-point Likert scale responses, one is the most negative response while seven is the most positive response. Questions in the 'All Tools' section were asked for each of the five diagramming tools tested in the experiment, while questions in the 'QuickDraw specific' section pertained only to QuickDraw

ANALYSIS OF QUESTIONNAIRE DATA

We asked participants to fill out a questionnaire at the end of the experiment. The questionnaire was based on [4] and was split into two sections. In the first section, participants were asked to rate each diagramming tool on a number of qualitative metrics. The second section contained questions related only to QuickDraw. Table 5 shows the questions asked in each section.

Participant Reaction to All Tools

Our post-questionnaire (see Table 5) asked participants to rate each tool at the end of the experiment. We used a non-parametric Friedman test to check for significant differences in qualitative metrics reported by participants for each tool. If significant differences were found, we performed as post-hoc analysis with Wilcoxon signed rank tests to uncover interesting patterns. In our post-hoc analysis, we controlled the

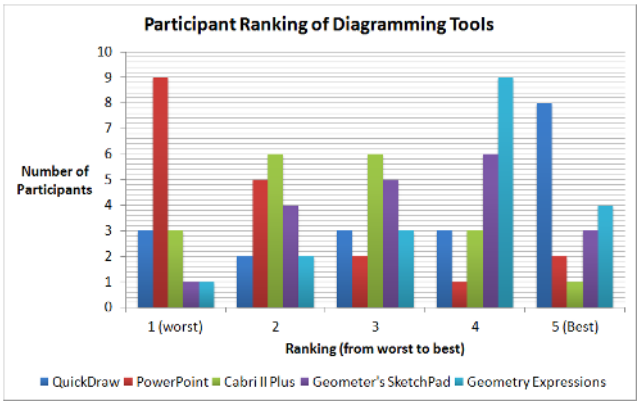


Figure 9. Software tools ranked from worst to best by participants.

chance of Type I errors by using a Holm's Sequential Bonferroni adjustment with four comparisons at $\alpha = 0.05$ [10].

We found significant differences for overall reaction ($\chi^2 = 11.810, p < 0.05$), perceived drawing performance ($\chi^2 = 15.93, p < 0.005$), and the ability of each tool to enable easy diagram drawing ($\chi^2 = 17.675, p < 0.01$). Using Wilcoxon signed rank tests, we found no significant difference in participants' overall reaction to any of the software tools. Participants rated their drawing performance in QuickDraw as better compared to Microsoft PowerPoint ($z = -2.431, p < 0.0125$), but not significantly different from Cabri II Plus, Geometer's SketchPad or Geometry Expressions. There was no significant difference in drawing capabilities among the software tools. QuickDraw's interface was no less confusing than that of any other tool in the experiment. It was no less difficult to correct mistakes in QuickDraw than in any of the other tools. It was also possible to draw in a straightforward manner in each tool. These are interesting findings because, in light of our non-parametric Friedman tests, we expected to find significant differences between QuickDraw and at least one of the other tools in terms of overall reaction and ability to enable easy drawing. However, the only differences approaching significance in both these aspects were with respect to Microsoft PowerPoint and were culled due to the Bonferroni adjustment. This leads us to suspect that given a larger number of participants, these differences may become significant. We also asked participants to rank all the software tools from worst to best. Figure 9 shows the rankings assigned by participants. It shows that more participants assigned the highest ranking to QuickDraw than to any of the other tools used in our experiment.

Feedback and Suggestions

The post-questionnaire asked participants to rate QuickDraw in terms of recognition accuracy and usefulness. It also asked them to list positive and negative aspects and sought feature suggestions (see Table 5). When asked if QuickDraw was able to recognize the sketched diagram correctly most of the time, the majority of the participants responded positively (Median = 6, Mean = 5.22). Similarly, when asked if QuickDraw enabled them to draw faster, the participants responded positively (Median = 6, Mean = 5.44). Lastly, when asked if they would find QuickDraw useful in their work, the overall response was a little above neutral (Median = 5, Mean = 4.94). 16 out of 19 participants indicated that they would like to use a sketch-based interface in the future.

Participants liked QuickDraw's simple interface. A large number remarked that the interface enabled very fast drawing, while freeing them from worrying about drawing correctly by doing beautification itself. When asked to list the negative aspects, the major complaint was about having limited editing capabilities. For some users, incorrectly beautified diagrams due to inference errors negatively impacted their experience, while others were happy to erase and redraw. A few participants lamented the lack of keyboard shortcuts, and redo/undo functionality. The most commonly suggested feature was the inclusion of better editing capabilities (manipulation, snapping, grid, etc). Some people suggested the inclusion of a mathematics recognition engine in order to be able to write down angles/dimensions on the sketched diagram. Some people wanted to be able to sketch out constraints on the diagram.

DISCUSSION

Our usability study highlights the usefulness of sketch-based interaction for building a diagramming tool. Participants' responses indicate a strong desire to use sketch-based software in the future. An analysis of data collected in our study provides several useful insights. For easy diagrams, participants were able to finish them more quickly using QuickDraw than with Microsoft PowerPoint. At this level of difficulty, QuickDraw was no worse than Cabri II Plus, Geometer's SketchPad or Geometry Expressions. However, as the difficulty of diagrams increased, QuickDraw overtook Cabri II Plus and Geometer's SketchPad in terms of completion time, while still being no worse than Geometry Expressions. It seems that for a sufficiently complicated diagram, QuickDraw would enable a user to achieve the minimum completion time among all the tools we have studied.

QuickDraw's performance is hampered by three major factors. First is a high failure rate ($\sim 13\%$) for difficult diagrams, which degrades performance and causes frustration among users. We adjusted for this by disregarding instances where participants could not complete diagrams in QuickDraw. This is fair because the completion time in such instances is inflated to a constant 180 seconds for QuickDraw which cannot be directly compared with the completion time in other tools. The second detriment to performance is the lack of adequate editing capabilities. Lastly, the majority of participants in our study chose to draw diagrams incrementally in QuickDraw. This slightly inflates the completion time, as QuickDraw achieves the best time when the entire diagram is recognized in one go. Yet it must be noted, that despite these detriments, QuickDraw was, in some cases, able to outperform three out of four state-of-the-art geometry construction tools. Indeed, we believe that if QuickDraw was to incorporate better editing capabilities and a further improved constraint inference engine, it would outperform the state-of-the-art in diagramming tools at all levels of difficulty.

Our analysis of the questionnaire data also raises some interesting points. Participants rated their drawing performance

in QuickDraw as higher compared to Microsoft PowerPoint. However, the overall reaction to QuickDraw was no worse than any of the other software tools. Interestingly, several participants complained about inadequate editing capabilities in QuickDraw. Yet there were no significant differences in their responses when asked to rate each tool on its ability to allow correction of mistakes. This seems to indicate that participants thought all tools including QuickDraw had a similar level of editing capability.

FUTURE WORK

The results of our user study suggest two avenues of future work. First, we would like to improve the editing abilities in QuickDraw. These include the ability to move/resize components, as well as the ability to zoom in/out of a diagram. We would also like to see if allowing users to explicitly sketch constraints on a diagram improves performance. Second, we want to improve inference of constraints by using classifiers over a variety of features (as inferred by machine learning techniques) as opposed to our existing hard-coded thresholds over a small number of features. We plan to examine the session videos recorded during the course of our study to identify instances that can cause incorrect constraints to be inferred and then work to rectify as many of them as possible. Additionally, we also want to extend the constraint language to refer to virtual components. These components are not explicitly sketched out by the user, but are required to express constraints between diagram components such as reference line-segments.

CONCLUSION

We have presented QuickDraw, a prototype sketch-based diagramming tool that lets a user easily express her intent in the form of a sketched diagram. QuickDraw can recognize individual components in a sketch, infer constraints between them, and then beautify them using a deductive geometry constraint solving algorithm. In this manner, it enables the drawing of precise diagrams. We conducted a usability study to compare QuickDraw with four state-of-the-art diagramming tools : Microsoft PowerPoint, Cabri II Plus, Geometer's SketchPad, and Geometry Expressions. Analysis of quantitative metrics shows that QuickDraw performed better than Microsoft PowerPoint for all difficulty levels, and better than Cabri II Plus and Geometer's SketchPad for medium and hard difficulty diagrams. There was no significant difference in drawing performance between QuickDraw and Geometry Expressions. However, deeper analysis revealed that the majority of participants did not utilize QuickDraw to its full potential. We believe that by adding better editing capabilities and improving the beautification algorithm further, QuickDraw has the potential to perform better than any WIMP based diagramming tool.

ACKNOWLEDGMENTS

This research was conducted at Microsoft Research during the course of a summer internship. Microsoft Research also supported this research by a Consulting Grant. We would also like to thank Bo Kang and Sarah Buchanan for helping to pilot test QuickDraw. This work is supported in part by NSF CAREER award IIS-0845921 and NSF awards IIS-0856045 and CCF-1012056.

REFERENCES

1. Aldefeld, B. Variation of geometries based on a geometric-reasoning method. *Computer Aided Design 20*, 3 (April 1988), 117–126.
2. Bouma, W., Fudos, I., Hoffmann, C. M., Cai, J., and Paige, R. Geometric constraint solver. *Computer-Aided Design 27*, 6 (1995), 487–501.
3. Cabri ii plus, 2011. `http://www.cabri.com`.
4. Chin, J. P., Diehl, V. A., and Norman, K. L. Development of an instrument measuring user satisfaction of the human-computer interface. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (1988), 213–218.
5. Forbus, K., Usher, J., Lovett, A., Lockwood, K., and Wetzel, J. Cogsketch: Sketch understanding for cognitive science research and for education. *Topics in Cognitive Science* (2011).
6. Geometer's sketchpad, 2011. `http://dynamicgeometry.com/`.
7. Geometry expressions, 2011. `http://www.geometryexpressions.com/`.
8. Gulwani, S., Korthikanti, V. A., and Tiwari, A. Synthesizing geometry constructions. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, PLDI '11, ACM (New York, NY, USA, 2011), 50–61.
9. Hammond, T., and Davis, R. Ladder, a sketching language for user interface developers. *Computers and Graphics 29*, 4 (2005), 518 – 532.
10. Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics 6*, 2 (1979), 65–70.
11. Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H. Interactive beautification: a technique for rapid geometric design. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97 (1997), 105–114.
12. Jiang, Y., Tian, F., Wang, H., Zhang, X., Wang, X., and Dai, G. Intelligent understanding of handwritten geometry theorem proving. In *Proceedings of the 15th international conference on Intelligent user interfaces*, IUI '10, ACM (New York, NY, USA, 2010), 119–128.
13. Kondo, K. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design 24*, 3 (1992), 141–147.
14. LaViola, Jr., J. J., and Zeleznik, R. C. Mathpad2: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph. 23* (August 2004), 432–440.
15. Li, Q., Liu, Y., Xu, H., Ren, L., and Ma, C. An intelligent interactive pen-based whiteboard for dynamic geometry teaching. In *Information Technologies and Applications in Education, 2007. ISITAE '07. First IEEE International Symposium on* (nov. 2007), 396 –401.
16. Microsoft powerpoint, 2011. `http://office.microsoft.com/en-us/powerpoint/`.
17. Nelson, G. Juno, a constraint-based graphics system. In *SIGGRAPH* (1985), 235–243.
18. Paulson, B., and Hammond, T. Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '08, ACM (New York, NY, USA, 2008), 1–10.
19. Wais, P., Wolin, A., and Alvarado, C. Designing a sketch recognition front-end: user perception of interface elements. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, SBIM '07, ACM (New York, NY, USA, 2007), 99–106.
20. Xiong, Y., and LaViola Jr., J. J. Technical section: A shortstraw-based algorithm for corner finding in sketch-based interfaces. *Comput. Graph. 34* (October 2010), 513–527.
21. Zeleznik, R. C., Bragdon, A., Liu, C.-C., and Forsberg, A. Lineogrammer: creating diagrams by drawing. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, ACM (New York, NY, USA, 2008), 161–170.