A Practical Framework for Constructing Structured Drawings

Salman Cheema[1], **Sarah Buchanan**[1], **Sumit Gulwani**[2], **Joseph J. LaViola Jr.**[1]
[1]University of Central Florida, Orlando, FL , [2]Microsoft Research, Redmond, WA
[1]{salmanc,sarahb,jjl}@cs.ucf.edu, [2]sumitg@microsoft.com

ABSTRACT

We describe a novel theoretical framework for modeling structured drawings which contain one or more patterns of repetition in their constituent elements. We then present PatternSketch, a sketch-based drawing tool built using our framework to allow quick construction of structured drawings. PatternSketch can recognize and beautify drawings containing line segments, polylines, arcs, and circles. Users can employ a series of gestures to identify repetitive elements and create new elements based on automatically inferred patterns. PatternSketch leverages the programming-by-example (PBE) paradigm, enabling it to infer non-trivial patterns from a few examples. We show that PatternSketch, with its sketch-based user interface and a unique pattern inference algorithm, enables efficient and natural construction of structured drawings.

Author Keywords
Sketch-based Interfaces, Programming by Example, Pattern Inference, Structured Drawing

ACM Classification Keywords
H.5.2. User Interfaces: Interaction Styles

General Terms
Algorithms, Human Factors

INTRODUCTION

Images and drawings containing structured repetition are common in real-life (e.g., brick patterns, tiling patterns, and architectural drawings). We define 'Structured Drawings' as drawings that can be drawn using a CAD tool: either requiring a user to write a script using CAD APIs or perform repetitive Copy-Paste operations with some underlying mathematical logic. Figure 1 shows examples of structured drawings. Making structured drawings is hard and time consuming without software support. Although no software tools exist solely for constructing structured drawings, some commercial applications such as Microsoft PowerPoint, Ink Scape, Adobe Illustrator and AutoDesk Inventor can be adapted for this purpose. These tools let users specify rudimentary patterns via a WIMP interface but rely mostly on variations of Copy-Paste (translation and scaling) to replicate repetitive elements.

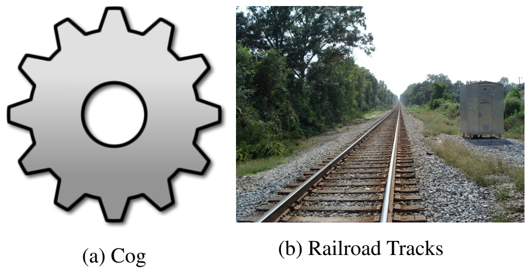

(a) Cog (b) Railroad Tracks

Figure 1: Examples of structured drawings (Source: Google Images)

In this paper, we describe a novel framework for modeling structure and repetition in drawings. Our framework uses a small set of abstract constructions, while giving an application developer the freedom to support different types of drawing elements and interaction methods. We leverage ideas from the domain of programming by example (PBE) [5, 9, 17] and set-theoretic constructions to create a novel algorithm that can infer patterns from a few examples highlighted by the user, and use them to complete structured drawings. We also describe PatternSketch, a sketch-based drawing tool built using our framework, that can recognize and beautify hand-drawn sketches, which users can manipulate with a series of gestures to identify and extend repetitive elements.

RELATED WORK

Programming by example (PBE) [5, 9, 17] is a popular paradigm for automating end-user programming tasks and has been used in a wide variety of domains including text-editing programs [16], spreadsheet data manipulation [10] and algebra problem generation [19]. We leverage ideas from PBE to construct a framework for modeling structured drawings, where the user provides a few examples of repetitive drawing elements, and our tool predicts the next elements in the sequence. The key technical contributions include a novel framework for representing sequences of drawing elements as well as a synthesis algorithm that infers example elements in the intended sequence from a collection of selected objects using a majority voting scheme.

We have chosen a sketch-based interface for PatternSketch so that it mimics the natural input method of pen and paper. Early work in sketch recognition focused on incorporating gesture recognition with direct manipulation [15] and on user guided recognition and beautification [2, 13]. Paulson et al. [18] have developed techniques for recognition and beautification of low-level sketch primitives. Interactive beautification [11, 22] and guided beautification [7, 8] of hand-drawn sketches have been explored in different application settings. More recently, beautification of sketched drawings involving line segments and circles by using inferred geometric constraints has been examined by [4]. PatternSketch employs a small set of drawing elements (line segments, polylines, arcs, and circles), in order to minimize interface confusion and to lower the likelihood of recognition errors. We adapted our recognition and beautification techniques from QuickDraw [4] and extended them to incorporate polylines and arcs. QuickDraw [4] was chosen as the beautification engine because it incorporates a rich set of constraints to ensure robustness.

Sketch-based interfaces have also been applied to several modeling tasks such as 3D Modeling [3, 12, 21], 3D curve sketching [1] and for animation tools for novice users [6]. Recently, Kazi et al. [14] have developed Vignette that utilizes texture synthesis and preserves individual style during sketching to generate artistic sketches. Vignette and PatternSketch both require users to specify initial examples and guide the predicted drawing but both tools differ in important ways. In Vignette, users generate texture patterns from a few examples, repeating this process a number of times for each drawing. This behavior can be modeled using our proposed framework (with its abstract representation of drawing elements and patterns) but is not entirely supported by our current prototype (PatternSketch), which instead focuses on ensuring precision. Comparing the two is difficult as both tools can probably demonstrate superior performance in different contexts, depending on whether an artistic look or precision is required. PatternSketch does not preserve individual style but uses constraint-based beautification and pattern generation to ensure precision in generated drawings.

THEORETICAL FRAMEWORK

Structured drawings contain one or more repetitive patterns in their constituent elements. Some patterns are simple and can be abstracted as linear Copy-Paste (i.e., a single element replicated many times by applying a translation and/or scaling). More often, repetition lies beyond the capability of linear Copy-Paste. Figure 1b shows a picture of railroad tracks where the planks on the inside of the rail tracks form a pattern incorporating both translation and scaling. However, the translation relating the copied planks is not constant. Additionally, an alignment operation is necessary to ensure that new planks always maintain the geometric relationship with the track lines. Figure 1b also demonstrates that repetition may not always occur in a straight line, thus requiring either clever inference or user intervention. Linear Copy-Paste can only extend patterns in a single direction.

We consider repetition as a generative operation starting from an initial sequence of drawing elements sketched by a user. From a theoretical perspective, the choice of how to define the generative operation is inconsequential and we leave this choice open to application developers. It can either be defined explicitly by the user or it may be inferred from a sufficient number of examples by using programming-by-example techniques. Additionally, the operation encodes information about how new elements are to be aligned with respect to existing elements. It can also be manipulated by a user to create new elements in one or more directions. We now define a few key ideas related to structured drawings:

Drawing Element (e) is a basic component of a sketched drawing. Examples are points, line segments, circles and composite shapes.

Collection ($\overline{E}$) is a set of drawing elements. A single drawing may contain zero or more collections.

Filter An operation used to select a subset of a collection's elements in a particular order.

Pattern (ϕ) is a spatial relationship inherent in an ordered sequence of drawing elements. A pattern is a generative operation that can extend the sequence by creating new drawing elements. The relationship can either be inferred from the entire initial sequence, a filtered subset, or can be explicitly defined by the user.

Frame of Reference A geometrical construct that serves as a frame of reference for a pattern within a structured drawing. It can potentially be used to denote boundaries within which a pattern can be extended. Additionally, new drawing elements generated by a pattern may have to be aligned or positioned relative to the frame of reference.

Copy-Paste (ϕ_{copy}) A special generative operation that creates a copy of a selected drawing element at a specified location. Any required alignment must be performed manually by the user.

These abstract entities enable us to model complicated structured drawings. From our perspective, the choice of which drawing elements and patterns to support, as well as interaction metaphors for creating collections, performing filtering, and extending patterns are implementation details that can vary from system to system, depending on context. Our framework permits the use of any type of drawing element. Potentially, even collections of elements can act as building blocks in a larger pattern. Similarly, we place no restrictions on interaction metaphors for creating collections, performing filtering, inferring/defining, or extending patterns. User interaction may be enabled via WIMP interfaces, sketch-based interaction, 3D gestures or even voice input. The notion of a frame-of-reference is also abstract. It can either be a drawing element or a path drawn by a user or even some virtual geometrical construct. Our framework also supports Copy-Paste functionality in its traditional form. The notion of collections that combine a sequence of drawing elements with a generative operation is very powerful and affords users the freedom to extend the pattern as they see fit or till some condition is met. Collections combined into hierarchical relationships can be used to create chains of generative patterns which can enable interesting effects.

PATTERNSKETCH: AN OVERVIEW

PatternSketch can recognize and beautify sketched drawings containing line segments, polylines, arcs and circles. A series of gestures can be used to interact with the drawing. The 'Lasso' gesture is used to group drawing elements into a collection. It can also be used to select existing collections. Once a collection is selected, the user can filter it by selecting a subset of its elements and assigning them an explicit ordering. For inferring patterns, users can either have the system consider the filtered collection or all possible ordered subsets. Once a pattern is inferred, the 'Drag' gesture can be used to generate new drawing elements.

Recognition and Beautification

Recognition is triggered by hitting the 'Recognize' button, after which the sketched drawing is parsed into its component elements (line segments, polylines, arcs, and circles). We use the IStraw [20] cusp finding algorithm to enumerate cusps in each ink stroke within the sketch, followed by a series of heuristics to classify each ink stroke as either a line segment, a circle, an arc, or a polyline. Our recognition heuristics and beautification system are based on ideas presented in [4], but have been extended to include polylines and arcs.

Constraints used for Beautification	
Applicable To	Constraint
Line Segments	Vertical line segment Horizontal line segment Parallel line segments Perpendicular line segments Touching line segments Line segments with same length
Circles	Circles with same radius Concentric circles Circles touching at their circumference Circle passing through the center of another circle
PolyLines	Similar structure (interior angles) Regular convex polygon Polygon
Circles, Line Segments & PolyLines	Line segment tangent to circle Line segment passing through center of circle Line segment touching circumference with an endpoint Line segment touching circle center with an endpoint PolyLine point touching a line segment PolyLine point touching a circle

Table 1: List of constraints that are inferred for beautifying drawings

After a drawing is recognized, our beautification subsystem infers geometric constraints between its recognized elements, which are used to beautify the drawing. In comparison to [4], we utilize a smaller set of constraints relating line segments and circles and have incorporated new constraints relating polylines with line segments and circles (See Table 1). After beautification, ink strokes are replaced by beautified drawing elements on the screen. For details of the beautification algorithm, please refer to [4].

Collections

Users can enable a special mode called 'Lasso' via the system menu. In 'Lasso' mode, a user can create a collection, select a element/collection, do Copy-Paste, or filter a subsequence from a collection for pattern inference. The user initially draws an ink stroke that encloses one or more drawing elements. If the encircled elements are part of an existing collection, the collection is selected. If selected elements belong to different collections, a new collection is created. Newly created collections are automatically selected (Selected collections are displayed with a colored bounding box). If a single element is lassoed, it is considered only for Copy-Paste. After selection, the user can either trigger pattern inference via the 'Infer' button on the menu or use the 'Tap' gesture to paste the selection at a new location. New elements created by Copy-Paste can be manually aligned by selecting 'Edit' mode via the menu and manipulating the element. PatternSketch allows the construction of the following collections of drawing elements:

Collection of Line Segments Homogeneous collection containing only line segments
Collection of Circles Homogeneous collection containing only circles
Collection of Polylines Homogeneous collection containing only polylines
Super Collection A collection containing other collections
Mixed Collection A collection containing several different types of drawing elements

With a selected collection, users can draw a line from the boundary of one element to the boundary of another element within the collection to assign an explicit ordering to the two elements thus creating a sequence within a collection. Sequences of drawing elements thus created can be extended by drawing another line from one of the elements within the sequence to another element outside the sequence but within the same collection. User defined orderings are rendered as dotted arrows. In this manner, PatternSketch merges the two steps (selection and ordering) of the filter operation into a single step which is intuitive and makes it easy for the user to indicate the intended ordering of elements within a collection.

Pattern Inference

Pattern inference is triggered by hitting the 'Infer' button from the menu. If the selected collection is filtered, the inference system tries to infer a pattern from the filtered elements. For an unfiltered collection, our inference system considers all possible ordered subsets of its elements. Within each subset, our inference algorithm considers ordered pairs (forming an input-output example) of drawing elements in isolation and uses a simple voting mechanism to determine the dominant pattern (See Algorithm 1).

We define the fundamental unit of each pattern as a 'PointSet' (ζ), which is a sequence of points that encodes one of the following relationships:

PointSet on Circle (ζ_C) Sequence of points along a circle's circumference separated by a constant non-zero arc length.

PointSet on Line Segment (ζ_L) Sequence of points along a line segment separated by a constant non-zero translation.

PointSet on Polyline (ζ_P) Sequence of points along a polyline separated by a constant non-zero distance.

PointSets form the basis for the following high-level patterns in PatternSketch:

1. Concentric Circles: A sequence of circles with a constant difference in radii whose centers are the same point.
2. Moving Lines: A sequence of line segments whose respective endpoints either form a PointSet or are the same point.
3. Moving Polylines: A sequence of polylines whose respective endpoints either form a PointSet or are the same point.
4. Moving Circles: A sequence of circles whose centers form a PointSet.
5. Moving Collections: A sequence of composite drawing elements or collections whose centroids form a PointSet.

PointSets are extremely useful because they capture low-level relationships between drawing elements and also encapsulate a frame of reference, as described in our theoretical framework. They enables us to model patterns as sets of low-level geometric transformations that are easy to visualize and can have one or more frames of reference.

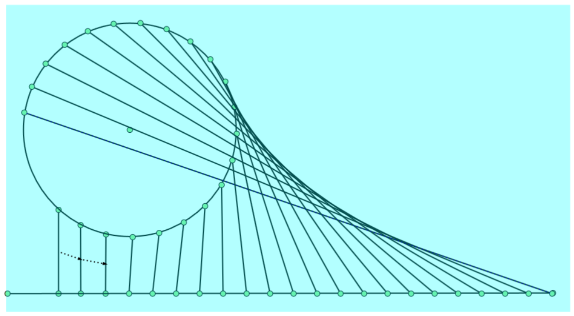


Figure 2: Example of a pattern of line segments that has two different frames of reference encoded by two different PointSets.

With patterns that leverage PointSets, we can enable powerful construction mechanisms. Figure 2 depicts a collection of line segments whose endpoints form PointSets along two different surfaces with different increments. As each of these PointSets encodes the relevant surface as a frame of reference, alignment is an implicit part of the repetition. It is also possible to use the frame of reference itself as a way to replicate a given pattern. For example, a collection of circles that form a pattern along the circumference of a larger circle can be selected for replication via the system menu. By selecting a target circle or even a collection of circles, the entire pattern can be duplicated onto the target. This is a non-trivial context-sensitive replication operation whose rules are encoded in our patterns and PointSets, enabling a very modular and powerful architecture.

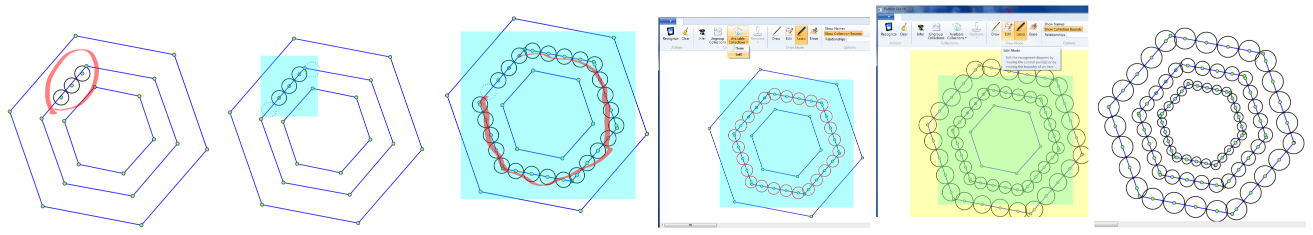
Algorithm 1 gives the outline of our inference algorithm, which looks at each ordered pair (an input-output example pair) of elements and tries to determine the geometric relationship between them. The $InferRelationShip$ procedure looks at the ordered pair (e_i, e_j), and determines if they constitute an example for one of the five patterns supported by the system. This is tested by applying the following rules:

Algorithm 1 Algorithm to infer a pattern from an unordered set of drawing elements

```
 1: function INFER($\overline{E}$)    ▷ $\overline{E}$ are ordered drawing elements
 2:     $\Phi := \emptyset$    ▷ $\Phi$ is the list of inferred patterns
 3:     for $i = 1 \to \|\overline{E}\|$ do
 4:         for $j = 1 \to \|\overline{E}\|$ do
 5:             $\phi_i \leftarrow InferRelationShip(e_i, e_j)$    ▷ $i \neq j$
 6:             if $\Phi$ Contains $\phi_i$ then
 7:                 Get $\phi_k$ from $\Phi$    ▷ $\phi_k$ is similar to $\phi_i$
 8:                 $\phi_k \leftarrow Merge(\phi_k, \phi_i)$
 9:                 $Score(\phi_k)$ += 1
10:             else
11:                 $Score(\phi_i) = 1$
12:                 $\Phi \leftarrow \Phi \cup \phi_i$
13:             end if
14:         end for
15:     end for
16:     Select $\phi_{best} \in \Phi$ with highest score
17:     return $\phi_{best}$
18: end function
```

- If e_i and e_j are circles and have the same center, they are considered an example of 'Concentric Circles'.
- If e_i and e_j are circles and their centers lie on the same line segment, polyline or circle, they are considered an example of 'Moving Circles'.
- if e_i and e_j are line segments, they are considered an example of 'Moving Lines', if two PointSets $(\zeta_C, \zeta_L, or\ \zeta_P)$ can be formulated involving both endpoints of both segments.
- if e_i and e_j are polylines, they are considered an example of 'Moving Polylines', if two PointSets $(\zeta_C, \zeta_L, or\ \zeta_P)$ can be formulated involving the first and last points of both polylines.
- if e_i and e_j are collections, they are considered an example of 'Moving Collections', if we can formulate a PointSet (ζ_L only with a virtual line as frame of reference) involving the centroids of both collections.

If the newly inferred pattern ϕ_i is similar to an existing pattern ϕ_k, then the score of ϕ_k is incremented and ϕ_i is merged with ϕ_k along with its example (e_i, e_j). If ϕ_i is not similar to any existing pattern, it is added to the set of possible patterns Φ with a score of one. Determining if two example pairs are part of the same pattern is difficult because users may not draw the elements with perfect spacing. Hence we introduce a similarity metric to infer patterns from noisy user input. We leverage PointSets and their unifying structure for this purpose. A pattern (ϕ_i) in our system contains one or more PointSets $(\zeta_1(\phi_i) \ldots \zeta_n(\phi_i))$ and a list of numeric values $\delta(\phi_i)$ (to denote pattern-specific information such as scaling coefficients, rotation values, alignment hints, etc). Each PointSet $\zeta_1(\phi_i)$ also has a list of interesting numerical values such as distances, arc length, vector offset, etc. Consequently, each pattern in our system can be represented as an



(a) Create Collection (b) Infer Pattern (c) Extend along Polyline (d) Copy Pattern (e) Select Target Collection (f) Replicate

Figure 3: Example scenario showing how mathematical art can be created using PatternSketch. The user draws three hexagons and creates a collection of circles on the surface of one of them. The 'Lasso' and 'Drag' gestures are used to extend the circles along the entire hexagon. Afterwards, the user can quickly replicate this pattern onto the other hexagons.

n-dimensional numeric vector:

$$\langle \phi_i \rangle = [\zeta_1(\phi_i) \ldots \zeta_n(\phi_i), \delta(\phi_i)]$$

Testing similarity among patterns simply constitutes comparing their vector representations using context-specific threshold values:

$$\text{Similarity}(\phi_1, \phi_2) = \langle \phi_1 \rangle - \langle \phi_2 \rangle \leq \varepsilon(\phi_1, \phi_2)$$

Where $\varepsilon(\phi_1, \phi_2)$ is a function to determine context-specific threshold values for comparing ϕ_1 and ϕ_2. Context-specific information can include things like touch constraints, distance thresholds, etc. Once all ordered pairs have been examined, the highest ranked pattern ϕ_{best} is picked as the most-likely candidate. It should be noted that, ideally, with programming-by-example, three examples are needed to determine if a set of drawing elements forms a pattern. As our system only considers pairs of elements in isolation, it can yield false positives, e.g., if two interesting points in a selection lie on a drawing element, they will always be considered a PointSet (ζ_C, ζ_L or ζ_P). Such erroneous cases are mitigated automatically because our algorithm picks the highest voted pattern. Erroneous PointSets will get few votes and be automatically suppressed. Also, as our algorithm only considers pairs of points, it will never consider a set of points in unison, rendering it unable to infer movement along a virtual circle. The workaround for this is to draw the virtual circle initially and erase it after creating the pattern. Virtual lines are deducible because each equidistant set of points on a virtual line will contribute higher votes to the same translation offset, making it a candidate for highest ranked pattern ϕ_{best}.

After a candidate pattern ϕ_{best} is chosen, the system first uses it to align the initial sequence. This involves aligning the elements with respect to the inferred frame(s) of reference. After alignment, our system uses the inferred pattern ϕ_{best} to predict the next element in the sequence and renders it on the screen with a dotted blue line. The 'Drag' gesture can then be used to extend the pattern. The user can drag the stylus to intersect the predicted item, causing the system to add it to the currently selected collection and predict a new item. This lets a user extend a collection as needed. Figure 3 shows a scenario where a sequence of circles is being extended along the edge of a polyline by continually dragging the stylus across the predicted elements.

Interactive Editing

Our beautification engine can make mistakes due to incorrectly inferred constraints, which can be corrected by entering 'Edit' mode, and manipulating the positions/sizes of beautified elements. For a circle, moving its center changes its position and moving its circumference changes its radius. For a line segment, moving the segment itself changes its position while moving either endpoint changes its length. For polylines, users can change the positions of its points by moving them as needed. Moving the boundary of a polyline moves the entire polyline to a new position. For severe beautification errors, the user can erase and redraw part or all of the drawing. We support the 'Scribble-Erase' gesture for erasing elements from the drawing. We also provide an intelligent 'Erase' mode which lets users delete parts of drawing elements. In intelligent 'Erase' mode, a user draws a region and any portions of recognized line segments falling within the region are clipped. PatternSketch also provides a method to break all groupings of drawing elements within a sketch by hitting the 'Ungroup Collections' button from the menu.

DISCUSSION AND CONCLUSION

Figure 4 shows a variety of structured drawings constructed with PatternSketch. Our framework provides a general way to describe structured drawings with a small set of abstract entities yet it leaves several important questions unanswered from an application developer's perspective. These questions include:

- What set of drawing elements should be supported?
- What types of patterns to support?
- What is a good method to instantiate supported drawing elements?
- What interaction metaphors are suitable for creating collections and filtering them?
- What is a good interaction metaphor for describing repetitive patterns explicitly?
- What algorithms may be used to infer patterns automatically?
- How can a user be empowered to extend and manipulate patterns?

We have addressed these practical questions with our prototype system, PatternSketch. It should be noted that PatternSketch is the first step toward the realization of our theoretical

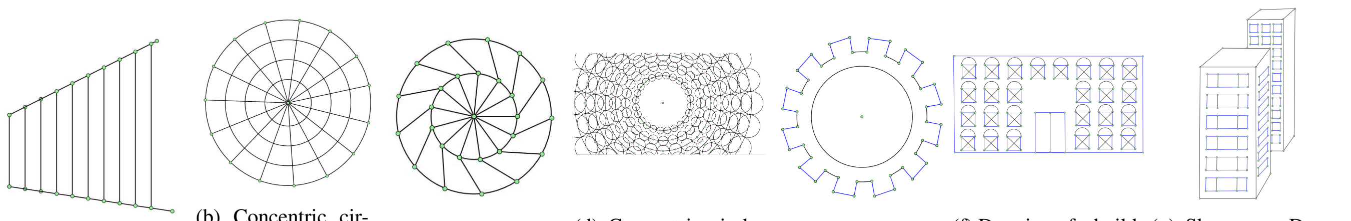

(a) Linear pattern. (b) Concentric circles and radial line patterns. (c) Lines moving on a circle pattern. (d) Concentric circles and radial circle patterns. (e) Polylines moving on circle pattern. (f) Drawing of a building. Collections moving on a line pattern. (g) Skyscraper Drawing. Collections moving on a line pattern.

Figure 4: Examples of drawings created using PatternSketch

framework and does not use the framework to its full potential. Our prototype system highlights the expressive power of our framework for modeling structured drawings. However, design trade-offs such as the use of sketch-based interaction can introduce errors in input (due to recognition and beautification), which can degrade drawing performance by requiring editing and redrawing. These errors can also cause problems with our pattern inference algorithm. Additionally, our choice of supporting a small set of drawing elements limits our system's capability to support all possible structured drawings that can be modeled using our framework.

PatternSketch uses a sketch-based interaction method to enable users to draw drawing elements in a natural manner. It supports four different types of drawing elements, and also enables composite elements by using collections. For creating and selecting collections, users can use the 'Lasso' gesture and can draw lines between drawing elements to perform filtering. PatternSketch supports five different types of patterns to cater to supported drawing elements. The patterns are built using the notion of PointSets that describe each pattern as a collection of low-level geometric transformations and implicitly encapsulate the notion of frame(s) of reference. We also provide the details of a novel inference algorithm that leverages concepts from the programming-by-example paradigm and exploits geometric constraints with a simple voting mechanism to identify dominant patterns. Our inference algorithm can work with both filtered and unfiltered collections.

ACKNOWLEDGEMENTS
This work is supported in part by NSF CAREER award IIS-0845921 and NSF awards IIS-0856045 and CCF-1012056.

REFERENCES
1. Bae, S.-H., Balakrishnan, R., and Singh, K. Everybodylovessketch: 3d sketching for a broader audience. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, ACM (2009), 59–68.
2. Baudel, T. A mark-based interaction paradigm for free-hand drawing. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, ACM (1994), 185–192.
3. Bernhardt, A., Pihuit, A., Cani, M.-P., and Barthe, L. Matisse: Painting 2d regions for modeling free-form shapes. In *SBIM '08* (2008), 57–64.
4. Cheema, S., Gulwani, S., and LaViola, J. Quickdraw: improving drawing experience for geometric diagrams. In *CHI '12* (2012), 1037–1046.
5. Cypher, A., Ed. *Watch What I Do – Programming by Demonstration*. MIT Press, 1993.
6. Davis, R. C., Colwell, B., and Landay, J. A. K-sketch: a 'kinetic' sketch pad for novice animators. In *CHI '08* (2008), 413–422.
7. Fernquist, J., Grossman, T., and Fitzmaurice, G. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 373–382.
8. Fung, R., Lank, E., Terry, M., and Latulipe, C. Kinematic templates: end-user tools for content-relative cursor manipulations. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, ACM (2008), 47–56.
9. Gulwani, S. Synthesis from examples: Interaction models and algorithms. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (2012). Invited talk paper.
10. Gulwani, S., Harris, W. R., and Singh, R. Spreadsheet data manipulation using examples. *Commun. ACM 55*, 8 (2012), 97–105.
11. Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H. Interactive beautification: a technique for rapid geometric design. In *UIST '97* (1997), 105–114.
12. Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99* (1999), 409–416.
13. Julia, L., and Faure, C. Pattern recognition and beautification for a pen based interface. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1, IEEE (1995), 58–63.
14. Kazi, R. H., Igarashi, T., Zhao, S., and Davis, R. Vignette: interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *CHI '12* (2012), 1727–1736.
15. Kurtenbach, G., and Buxton, W. Issues in combining marking and direct manipulation techniques. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*, ACM (1991), 137–144.
16. Lau, T. A., Domingos, P., and Weld, D. S. Version space algebra and its application to programming by demonstration. In *Machine Learning (ICML)* (2000), 527–534.
17. Lieberman, H. *Your Wish Is My Command: Programming by Example*. Morgan Kaufmann, 2001.
18. Paulson, B., and Hammond, T. Paleosketch: accurate primitive sketch recognition and beautification. In *IUI '08* (2008), 1–10.
19. Singh, R., Gulwani, S., and Rajamani, S. Automatically generating algebra problems. In *AAAI* (2012).
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. Yang, C., Sharon, D., and van de Panne, M. Sketch-based modeling of parameterized objects. In *SIGGRAPH 2005* (2005).
22. Zeleznik, R. C., Bragdon, A., Liu, C.-C., and Forsberg, A. Lineogrammer: creating diagrams by drawing. In *UIST '08* (2008), 161–170.