DrawCAD: Mouse-sketch-based engineering drawing

Abhiram Ranade and Shripad Sarade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
ranade@cse.iitb.ac.in

ABSTRACT

While there has been a lot of work on freehand drawing programs, robust programs which are easy to learn are still not available. We feel our program DrawCAD takes significant steps in this direction. Two dimensional drawings are sketched by the user using the mouse; DrawCAD analyzes the sketches and infers the intent of the user and produces beautified drawings. DrawCAD also infers and maintains constraints between various elements of the drawing. Supported constraints include horizontality or verticality of lines, tangency between arcs and lines, parallelism/perpendicularity between lines, equality of angles or line segment lengths. Some of these constraints are inferred as the user draws, and some can be explicitly added using well known conventions of Euclidean Geometry (e.g. putting a wedge to indicate perpendicularity). The key idea in this is to treat the strokes drawn by the user on one hand as actual drawing elements, but at the same time as gestures. We feel that this makes it easy for the user to declare his/her intent, and also easy for DrawCAD to recognize the intent.

User studies are presented in which some benchmark drawings are created using DrawCAD as well as standard programs. Some of our benchmarks are simple informal drawings, and others are detailed engineering drawings with dimensions etc. Our general conclusion is that DrawCAD is very robust, easy to learn, and fast to use.

Author Keywords

sketching; engineering drawing; constraints

ACM Classification Keywords

H.5.2 Information interfaces and presentation: User Interfaces. - Graphical user interfaces.

INTRODUCTION

Conventional wisdom has it that mouse drawn sketches are great for rough drawing, i.e. putting down ideas on paper just as you might doodle on backs of envelopes. There is work on *beautifying* these sketches, but it seems to be generally accepted that such sketching is not useful for making accurate drawings, e.g. engineering drawings. Indeed, standard engineering drawing programs such as AutoCAD encourage the use of the WIMP (windows, icons, menus, pointer) approach, i.e. basically a *tool* is provided for every primitive object to be drawn, and the tool is invoked through a series of menus/icons/dialogue boxes. For example, we might want to draw a circle that passes through a given point, is tangential to a given line, and has a certain radius. To draw this requires invoking the circle drawing tool, and somehow filling in the different attributes, either explicitly through dialogue boxes, or through a sequence of menu choices and icon clicks. Selecting menus and items requires substantial mouse movement and time. How to fill the dialogue boxes and which specific menus/icons to use presents a substantial learning curve.

DrawCAD explores a more natural approach to engineering drawing. The user is asked to *sketch* the required drawing using the mouse (or touchscreen etc.), and the program infers the user intent, and produces the required exact (much more stringent than just "beautified") drawing. So in the circle example mentioned above, the user would simply draw a freehand curve through the given point, and such that it appears tangential to the given line. At the end of drawing this curve, the radius of the circle is typed in. If the recognition process worked well (and it *does* work well as you will see shortly), you would have your circle. The benefits of this approach are two-fold: (a) the time to make the drawing can be much less, and indeed, our experiments indicate that DrawCAD is upto 2-3 times faster than conventional drafting programs such as AutoCAD, and (b) the time required by the user to learn DrawCAD is miniscule in comparison to standard drafting programs.

Note that DrawCAD does not expect users to be artists and draw accurate sketches. This would be completely unacceptable! Indeed, most of us artistically challenged people find it hard to draw a perfect circle using a pencil, let alone a mouse. On top of that if we need to make the circle pass through certain points etc. then the task becomes very daunting. The key insight in DrawCAD is to treat the stroke drawn using the mouse not as a precise geometric shape, but as a *caricature* of the actual shape. As we discuss later, while it is difficult to draw shapes accurately using the mouse, we believe that most users can control the mouse well enough to indicate features such as co-incidence, tangency, horizontality and so on. Hence DrawCAD pays more attention to inferring and managing such features than to the precise shape. This is not to say that the shape is completely ignored. Only that feature cues, if present, are considered more important.

India HCI 2013 and APCHI, 2013, Bangalore, India.

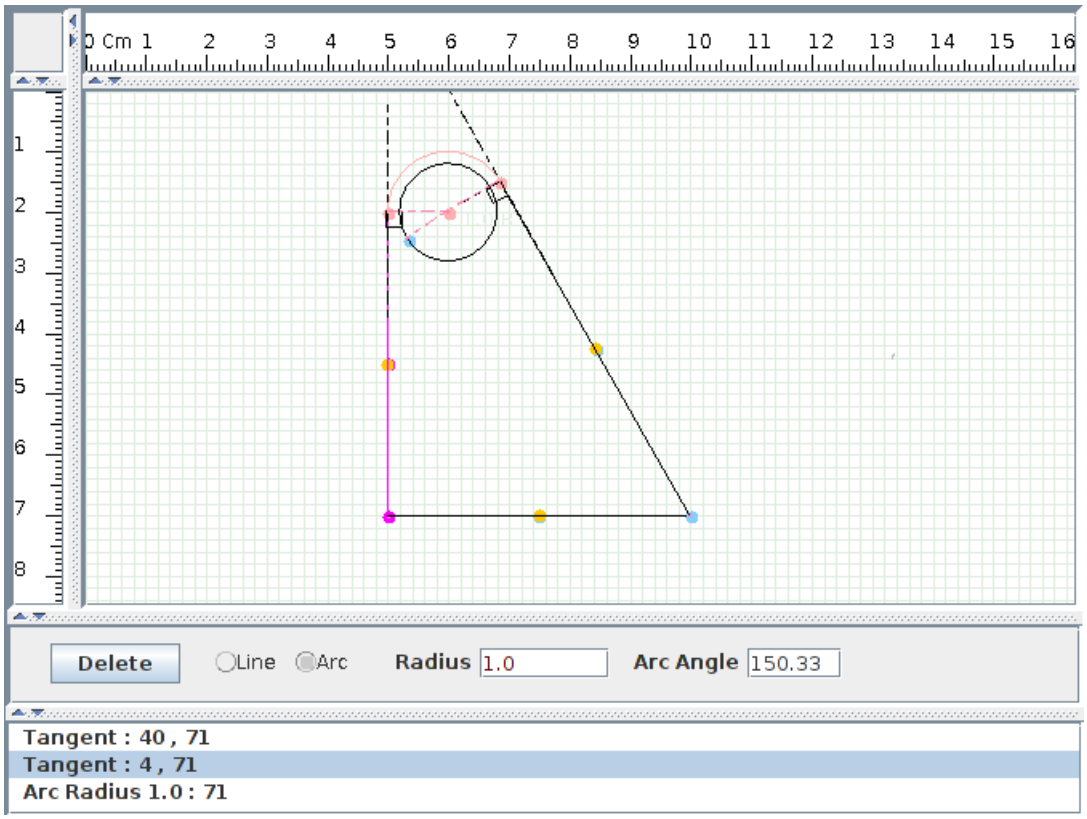


Figure 1. Screenshot of DrawCAD

Section (Constraints) gives the features/constraints currently recognized in DrawCAD.

Features/constraints are thus very important and intrinsic to DrawCAD. DrawCAD provides another important mechanism by which the user can indicate geometric constraints, which we call *Nudging*. In this, the mouse is used to drag a point/line/arc in the drawing. The geometry/constraint engine of DrawCAD lets the element be dragged while simultaneously adapting the rest of the drawing so that all constraints inferred so far continue to be obeyed. The user stops dragging at a convenient juncture: say when the dragged arc looks (within a certain preset tolerance) tangential to the required line. On release, the constraint inference kicks in again, and the arc is indeed declared to be tangential and the constraint is installed into DrawCAD. Nudging is very useful in many cases, e.g. when two points need to be merged. We also implement the conventions used in Euclidean geometry to indicate parallelism (arrows on the relevant lines), or equality of line segments or angles (similar tick marks on them), or perpendicularity (wedge between the two lines). These symbols: arrows, ticks and wedges are collectively referred to as markers in what follows.

Contributions

Many of the ideas we have discussed above, except perhaps the notion of caricatures, are natural and obvious, and have appeared in previous papers.

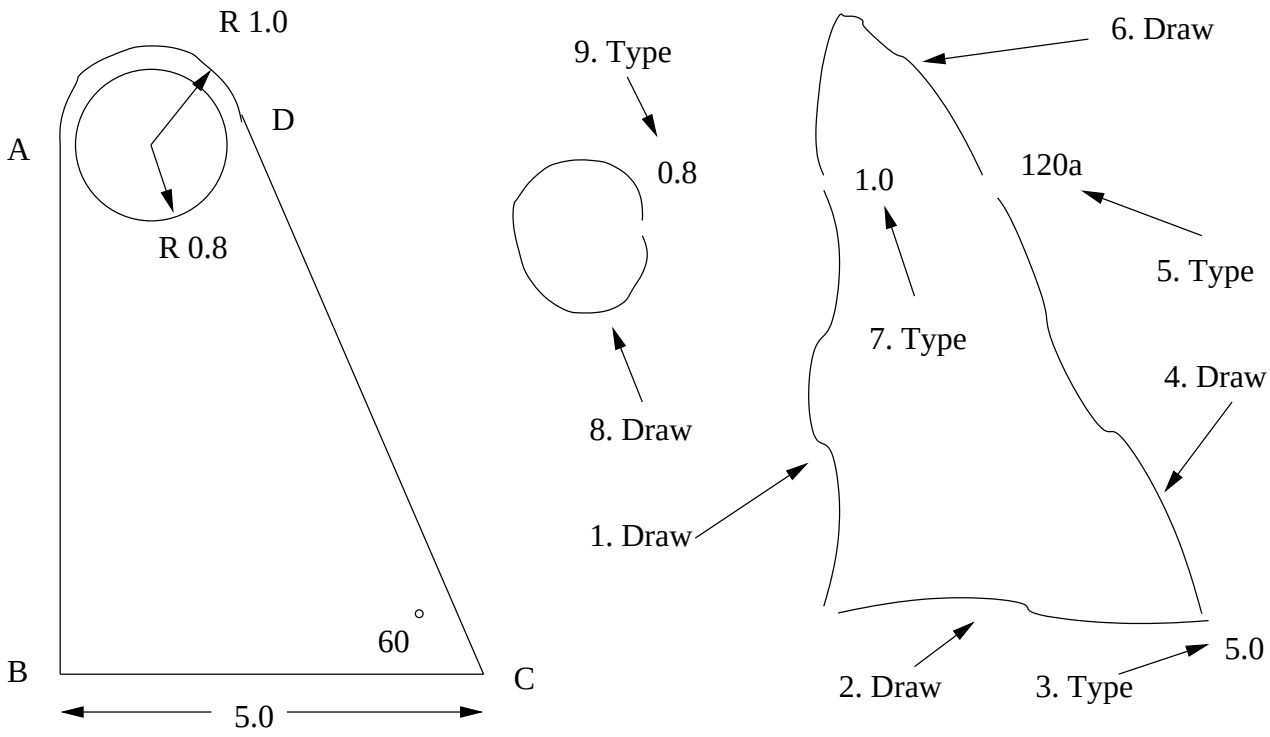
Our main contribution is in identifying the correct combination of ideas needed to make sketch-based drawing viable: the notion of caricatures, managing constraints, and nudging. Our evaluation criteria for this are (a) time to make the drawing, (b) time to become familiar with the program. It is customary in the literature to also present *error analysis*. We have not done so because our users make very few errors, and they correct themselves quite easily. Also, our concern is more with the overall performance.

We evaluate DrawCAD using a benchmark set of drawings (Section (USER TRIALS)). We give a 30 minute introduction to DrawCAD and then ask the users to make the drawings. We report the time taken by users. Likewise we also ask users to make the drawings using their favourite programs, e.g. CATIA or AutoCAD. In this case we also report the expertise level of the users. We find that it takes 2 to 3 times longer to draw using these programs than using DrawCAD.

It is also worth comparing DrawCAD to programs such as Parsketch[7] and Quickdraw [9]. We discuss this later.

Outline

We begin by showing a small session using DrawCAD. Then we discuss the philosophy and design of DrawCAD. Then we describe the some of the drawings we have done using DrawCAD. Then we present our benchmark drawings and report our user trials. After that we present the previous work in the area. We have deferred the discussion of the previous work



(a) Target figure (b) As sketched in DrawCAD

Figure 2. Using DrawCAD

so that it can be better compared with DrawCAD. Finally we summarize and conclude.

A SAMPLE SESSION

Figure 1 gives a screenshot of DrawCAD in action. As you can see, there are 3 main panels: the Drawing panel at the top, the Edit panel below it, and the Constraints panel at the bottom. Drawing is done in the Drawing panel, and the figure shows a drawing in progress. Information about the last drawn (or last selected) element appears in the Edit panel. In the figure this is the panel in which some buttons and text boxes are visible. These can be used to delete the element or change the element parameters. The constraints related to the last drawn/selected element appear in the Constraints panel. The figure shows the constraints related to the top arc in the drawing, which was the selected element. The selected element is highlighted in a different colour in the Drawing panel. By hovering the mouse on a constraint in the Constraints panel, a constraint can be highlighted, as shown. As a result, the elements involved in the constraint get highlighted in the Drawing panel as shown.

We will use DrawCAD to draw the figure of Figure 2(a). Drawing is accomplished in DrawCAD using the left mouse button. Move the mouse to the position you want to start drawing. Press the left button and drag, a line appears on the screen. When you want to stop drawing, release the button. This is what is meant when the directions below ask you to “draw”. Typing in dimensions is easy: you merely type, without worrying about where the cursor is. What you type will appear at your cursor position, and vanish when you hit ENTER.

Figure 2(b) pictorially gives the sequence of actions you need to perform. Here is a brief textual explanation of each action.

1. Draw a vertical looking line. This is meant to become line AB.
2. Continue and draw a horizontal looking line. This is meant to become BC.
3. Type “5.0” followed by enter. This specifies the length of the last drawn line, AB. Note that typing does not require the mouse to be moved.
4. Continue and draw a line at an incline. This is meant to become line CD.
5. Type “120a”. This specifies the angle made by the last drawn line, CD, with the positive *x* axis.
6. In this step we draw the arc DA. For this, draw a stroke from D to A. It will be good if the stroke appears to be tangential to CD, and also to AB. Other than this, nothing is needed. As you can see, we have drawn it as an inverted ’V’ shape, with a sharp angle. It can even knots, i.e. it can self intersect if you find that while drawing the stroke your mouse has drifted too far.
7. Type “1.0”. This specifies the radius of the arc.
8. Draw the circle.

9. Type “0.8”. This specifies the radius.
10. Right click on the center of the circle, and drag it to the center of the arc DA.

The last step, step 10, is not shown in Figure 2.

Most likely, at this point you would have correctly drawn the figure. However, if you did not, you can easily correct yourself as you go along. Here are some of the possibilities.

1. You meant to draw a straight line, but it was recognized as an arc by DrawCAD. In this case you simply need to select the radio button labelled Line in the Edit panel. Vice versa if your stroke was recognized as a line when you meant an arc.
2. If the line you drew was not recognized as vertical (or horizontal), drag an endpoint with the mouse right button, and release when it becomes nearly vertical (or horizontal).
3. You forgot to make your arc tangential to a line, or you tried but DrawCAD did not catch it. In this case you have two choices:

   Nudging: Drag the point where the line and arc meet using the right mouse button. If you jiggle it around enough, you will see that the arc and line do become nearly tangential somewhere. Release the mouse at that point. On release, DrawCAD will detect that the line and arc are nearly tangential, and so will assume that you want them tangential. So the constraint will be recognized.

   Markers: Draw a small arrowhead on the line, and another one on the arc. This is how parallel lines are indicated in Euclidean Geometry. In DrawCAD, this can be used to indicate parallelism as well as tangency. If DrawCAD did not recognize your arrowhead, then you can correct it by selecting the appropriate radio button in the Edit panel.
4. You mistyped a number. All you need to do is click on the element and retype. If you realize your mistake before hitting enter, you can just use the backspace and correct, of course.
5. If some constraint you did not want is falsely recognized, then place your mouse on that constraint in the Constraints panel. Then press the delete key.

Our constraint recognition is fairly good, so by and large corrective actions are not needed.

PREMISES

We begin by stating our premises regarding what (typical) users cannot and can be expected to sketch using the mouse. Then we consider how engineering drawing is different from other kinds of informal sketching.

On precision sketching:

We have remarked earlier that sketching with reasonable accuracy may not be easy for most users, even using pencil and paper. The situation is even more difficult with mice, or even touch screens. It may be noted that mice were invented originally not for drawing, but as a pointing device. If we wish to merely draw a straight line segment connecting two points, it can usually be drawn fairly reliably. However, suppose we wish to draw an incircle of a triangle, then maneuvering the mouse is usually very difficult for most users.

In contrast, here is what we think can be done well by (not necessarily artistic) users using available (noisy) mice. We believe that users can reliably start the drawing at the point they want to. Furthermore, the direction of the movement can be controlled reasonably well at the start. As the stroke progresses, however, the noisiness/lack of skill appears to become evident. As a result, the mouse drifts from the path the user might have wanted. However, if the user is assured that he/she need not be worried about the drifting, and may simply try to bring back the mouse on course, we feel that most users can end the stroke at the designated spot. Furthermore, usually it is possible to approach the designated ending spot in the appropriate direction. Another point is that the user might wish to pass the stroke through a certain point. This also can be managed by most users; again provided they are told not to get flustered if the mouse seems to drift, but just bring it back to the target point somehow and carry on.

The nature of engineering drawings:

Rough, informal sketches drawn using a mouse have been substantially explored. However, these are fundamentally different from engineering drawings. Typically, in engineering drawing, all the drawing elements will have specific dimensions, either specified explicitly for the element, or implied by the dimensions of other elements. This implies that the activity of assigning dimensions will be almost as frequent as sketching. Second, elements in engineering drawings are also often associated with interesting constraints. For example, a line may be horizontal, or be tangent to an arc and so on. We would like DrawCAD to infer these constraints, display them to the user, and get immediate confirmation about the correctness of the inference.

There is also a deeper point. In informal sketching, the *shape* of the stroke drawn by the user has great importance. Most drawing programs, e.g. [13] and even Parsketch[7] as far as we can tell, will pay great attention to the general shape of the stroke drawn, and try to *fit* the recognized element to the stroke shape. Except for very simple strokes, say straight lines, the fit is unlikely to be exact, because of the noisiness inherent in the mouse. But in case of engineering drawings, an approximate fit is useless: if I intended an arc to have radius 50, it does not make a difference to me if it is recognized to have radius 51 or 100. This is because in both cases I must correct the radius somehow: correcting 51 to 50 is really no different than correcting 100 to 50. But once we realize that inferring the shape is not too useful, we might as well tell the user that and ask him/her to concentrate on other aspects such as co-incidence and tangency.

THE DESIGN OF DrawCAD

The design of DrawCAD follows naturally from the above discussion.

Before we go into it, we discuss some basic terms. The main elements in a DrawCAD drawing are **line**s, i.e. straight line segments, **arc**s, **markers**, **point**s, and **constraint**s. Points can be primary, by this we mean endpoints of lines and arcs. Points can also be secondary, by this we mean centers of arcs and midpoints of lines.

A **stroke** is the curve drawn on the screen using the mouse. As mentioned earlier, we do this using the left mouse button: to start drawing the left button is pressed, then the mouse is moved as needed, and finally the left buttion is released to end the stroke.

One stroke = one element

In our prototype implementation we have chosen to interpret every stroke drawn as a single drawing element. For example, if we wish to draw a rectangle, we must raise the mouse after finishing one side, and lower it again to begin the next side. This is clearly inconvenient, and we indeed will remove the restriction later. However, for now, treating one stroke to be a single element allows us to experiment with some interesting ideas.

In Section (On precision sketching) we discussed why the stroke data we get should be considered somewhat unreliable. But if stroke data is unreliable, it is best to keep the stroke size small, and also tell the user how the program interprets the last stroke immediately after it is drawn.[1] By asking the user to only draw one element per stroke, we are indirectly encouraging small strokes! Further, after the program interprets the stroke, the interpretation is displayed in the Edit panel. As may be seen in Figure 1 the Edit panel has radio buttons using which the user can immediately choose the correct element type if the progrm made a mistake in interpretation. Clearly this would be very clumsy if every stroke was deemed to consist of several elements: there would be too much information to show in the panel.

The second reason comes from our characterization of engineering drawings. Most elements need to be dimensioned. If a stroke is deemed to be a single element, the user can immediately give the dimensions for the element. We have already remarked how this is done: the length(radius) of lines(arcs) is typed immediately, and/or so is the inclination(angle measure). These changes can also be made in the Edit panel if the user so chooses. If the stroke was deemed to consist of several elements, then the dimensions given by the user would have to be matched with the elements derived from the stroke. This would be quite messy.

The third reason also comes from our characterization of engineering drawings. We remarked that each drawing element usually has some important features (e.g. a line could be vertical) or is in some geometric relationship with other elements (e.g. the line is tangential to a previously drawn arc). We feel it is appropriate to show these features/relationships immediately to the user and get them confirmed. This we do in the Constraint panel. Occasionally the several constraints are inferred for newly drawn element. In such cases, the user may hover the mouse over each constraint, which causes the involved elements to get highlighted in the Drawing panel. If a certain constraint has been inferred incorrectly by DrawCAD then the user can just type DELETE which will cause that constraint to be disabled (Section (Constraint removal)). Needless to say, if a single stroke was segmented into several elements, too many constraints might have to be shown in the Constraints panel. The clutter might confuse more than help.

[1] Analogous to why short packets are preferred in noisy channels.

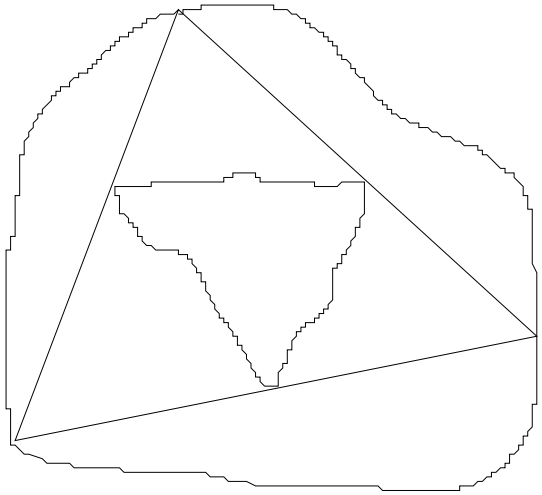


Figure 3. Acceptable ways of drawing an incircle and a circumcircle

Caricature

There is also a deeper, somewhat subtle fallout of the single element rule. It allows us to provide a more convenient drawing protocol to the user.

We explain this with an example. Suppose the user wants to draw an incircle of a given triangle. As mentioned, this is a difficult undertaking for most users. What most users draw will invariably have several “corners”, with many straight patches, and inevitably the curvature will change many times in the stroke. Without the single element rule, our stroke interpreter would have to work very hard to figure out that what the user intends is in fact a circumcircle. With the single element rule, the process is very easy. For an incircle, the user may inscribe any closed figure inside the triangle. For example, the user may just inscribe another triangle, as in Figure 3. It is easy for our stroke classifier to see that this stroke closes on itself and hence must be a circle, rather than a straight line. Further, it is also easy for the classifier to see that the stroke is touching the sides of the triangle but never crosses any side. Thus the stroke is immediately classified as an incircle. Similar remarks apply to drawing a circumcircle. In general, it is easy for our classifier to confidently infer properties such as: (a) the given stroke touches a line but remains on one side, which implies it is tangential, (b) a given stroke passes through a given vertex. Even a clumsy user can draw strokes in this manner – note that the line drawn by the user may veer off from its course, but the user can bring it back without worrying about “how it looks” to the classifier. Basically the simplicity makes the task easier for the user as well as for our program. The reader might realize that our directions in step

6 of Section (A SAMPLE SESSION) were precisely because of this protocol.

The key idea here is that we are inviting the user to draw a *caricature* of the element of interest, rather than worrying about rendering it realistically. The caricature distorts the stroke to emphasize the important feature (tangency in our case), just as in a caricature or a cartoon. A caricature could be considered to be half way between a realistic drawing and a gesture.

Constraints

DrawCAD supports a wide range of constraints, which include (we think!) the constraints needed most commonly. In this section we discuss how these sections can be explicitly added. Several of these constraints do not need to be added explicitly, but will be inferred by the program, when the stroke is drawn, or during nudging. We discuss inference later.

1. Horizontality or verticality of lines: This can be usually inferred at the time of drawing. However, there are many ways to add it later. One possibility is nudging: one of the endpoints of the line can be dragged to make the line more vertical and horizontal. If the endpoint is then released, the constraint will be inferred. Alternatively, immediately after drawing or after explicit selection the angle may be supplied as "0a" or "90a" to indicate horizontality or verticality.

2. Co-incidence: Whether an arc/line passes through primary/secondary points outside the arc/line. Some of this will be inferred when the arc/line is first drawn. But otherwise, nudging can be used to enforce this. The point which is to be on the line must be dragged to the line and released.

3. Tangency: Whether an arc/line is tangential to other arcs. This is also most conveniently inferred at the time the arc/line is drawn. But if not, nudging can be used. Drag the point of intersection between the first arc/line and the other arc. There will be a dragging direction, for which you will see the involved elements become tangential. This direction will be fairly easily found.

   Another way is to use markers: draw little arrows on the elements you want tangential. Then DrawCAD will enforce tangentiality.

4. Whether two lines have equal length: At present, DrawCAD does not try to infer this. The user must indicate equality by drawing a small tick (very small line segment) on the two lines that are to be declared to have equal length. This is also exactly like what high school students do in Euclidean geometry diagrams.

5. Whether two angles have equal measure: At present, DrawCAD does not try to infer this. The user must indicate equality by drawing a small tick (very small line segment) in the angles, near the vertex. This is also like what students do in Euclidean geometry diagrams.

6. Whether two lines are parallel: At present, DrawCAD does not try to infer this. The user must indicate parallelism by drawing small arrows on the lines. This is also like what students do in Euclidean geometry diagrams.

7. Whether two lines are perpendicular: At present, DrawCAD does try to infer this. Inference appears good, but more experience is needed. To add explicitly, the user must draw a wedge connecting the sides of the angle that is to be made a right angle. This is also like what students do in Euclidean geometry diagrams.

8. Merging two points: DrawCAD does not merge points by default even if they come very close. If you want points merged, you can use nudging, i.e. drag one to the other.

9. Whether two arcs are concentric: At present, DrawCAD does not try to infer this. This must be done by nudging: drag the center of one arc to the center of the other.

There is a simple way to remove unwanted constraints, as will be discussed in Section (Constraint removal).

Note that dimensions of lines or arcs are also treated as constraints.

Inference

In principle, it is possible to infer all the above constraints. However, an inference can be incorrect. If an incorrect inference is made, the user must take corrective action which is an overhead. On the other hand, when a correct inference is made, the user is spared the effort of indicating the constraint explicitly. Clearly, whether we should attempt to infer a certain constraint depends upon (a) P_{fp}, the probability of false positives, (b) E_a, the effort (required to add the constraint) saved by the user in case of a correct inference, (c) E_r, the effort (for constraint removal) wasted by the user in case we made a wrong inference. Clearly, we should be infering a constraint provided $(1 - P_{fp})E_a > P_{fp}E_r$. From Section (Constraints) we can see that the effort of adding or removing constraints explicitly is comparable. But often, there is an irrational factor at play: people get annoyed if a wrong inference is made. So we really only make inferences if we can have very low false positive probability.

Our experience is that we can correctly infer horizontality/verticality, with extremely high probability (low false positive as well as negative). So we do infer this. Our experience is that horizontal and vertical lines are extremely common, and making this inference saves us a considerable amount of time while drawing.

It would seem that a user should be able to indicate fairly unambiguously whether an arc passes through previously defined points. The problem arises here when the drawing becomes very dense, i.e. there are many elements close together on the screen. Remember that for each line we highlight its endpoints as well as its midpoint, and for each arc its center as well as its endpoints. We have found that it is somewhat difficult for the user to navigate his/her way through the many points on the screen. So we adopt a via media. We consider the midpoints and arc centers to be secondary, and the other points primary. If a primary point appears anywhere on the stroke we report that as a constraint. However, we require that

a secondary point must appear only as a stroke endpoint in order to get reported as a constraint. If we want a secondary point to be on a line, it must be nudged in, as discussed in Section (Constraints).

As for tangency: our experience has been that tangency inferences have very low false positive rate (this of course depends upon the thresholds we set in the algorithm). So we have decided to infer tangency. This seems to work well.

We have found that equal length constraints are somewhat tricky to infer. In general there appear to be more false positives, somewhat independent of the threshold in our algorithm, because users often do not pay attention to the length when they draw. Second, in engineering drawing, most lines need to be given a dimension. The cost of adding a dimension is very little – so the saving from inferring the equal lengths constraints is somewhat small. Hence our current decision is not to attempt to infer the equal length constraint. The situation for equal angles is similar.[2]

The rules for inference after nudging are similar to those described above.

Selecting elements

You can select a line/arc by clicking on it. Selecting an element causes data regarding the element to be shown in the Edit panel. Also all constraints associated with the element are shown in the Constraints panel.

You can also edit the data shown: e.g. change the length/radius or orientation/angle measure, either by typing the values in the Drawing panel (as you would after drawing the element), or in the Edit panel. Also, you may disable constraints by hovering over them and clicking DELETE.

Note that by default the last drawn element is considered to be "selected".

It is also possible to select points. In this case, the related constraints are shown in the Constraints panel.

Constraint removal

To remove a constraint, it must first get displayed in the Constraints panel. Constraints associated with an element get displayed in the Constraints panel automatically when the element is first drawn. Alternatively, selecting an element also causes the associated constraints to be shown.

If the user moves the mouse over a constraint shown in the Constraints panel, then the relevant elements get highlighted in the drawing panel. To delete a constraint, simply press the DELETE key when the mouse is over the constraint. This marks the constraint as disabled and retains it in DrawCAD's database. The reason for retaining the constraint in a disabled form is the following. Since the constraint was inferred once, very likely it will be inferred again. However, the user has just told us that the constraint should not be added. So keeping it around in the disabled form will tell us not to add it again. What if the user decides that he/she has made a mistake and wants the constraint anyway? He/she simply DELETEs it again. This time it will be completely removed from DrawCAD. Now if it is inferred again it will get installed.

[2]There could of course be situations with more compelling arguments: e.g. consistent symmetry. We will explore these in the future.

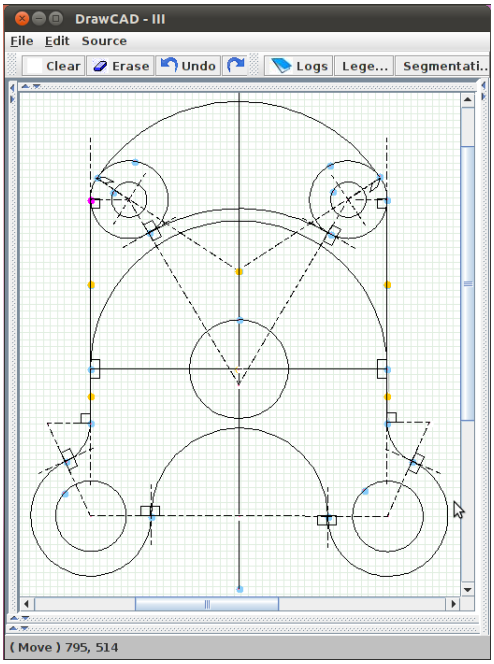


Figure 4. Test diagram 3

IMPLEMENTATION

We have implemented DrawCAD in Java. The idea was not to build a full fledged drawing program, but build a test bed for our ideas about how mouse based sketching can be used for engineering drawings. As a result, our code does not have several standard features such as copy/paste, or zooming in and out, saving drawings in different formats. These will of course be needed in a production version. In a production vesion we will also allow several elements to be drawn in a single stroke. We are optimistic that our program will be able to tell the difference between a caricature (which denotes a single element) and a stroke which truly denotes multiple elements.

A note is in order regarding our constraint solver. We just use a simple Newton-Raphson method. This requires SVD computation which is done using [2]. It is suggested in the literature that constraints be introduced gradually, e.g. if we want to make two lines have equal length, then look at the current ratio, and make it go to one in a series of steps. The rationale for this is that Newton-Raphson is guaranteed to work well only if the target solution is "close" to the starting point. Our experience has been that basic solver itself converges satisfactorily when constraints get added one at a time. It would be nice if this experience can be backed by rigorous proof.

OUR DRAWING EXPERIENCE

We have exercised DrawCAD substantially and used it to draw many kinds of simple and complex engineering drawings, two of which we will discuss in detail later. We have also tested DrawCAD on an example taken from [7]. Another

Diagram 1			
User id	Time taken	Program used	Expertise level
1	364 sec	CATIA	3 years
1	105 sec	DrawCAD	30 minutes training
2	243 sec	CATIA	5 years
2	102 sec	DrawCAD	30 minutes training
3	120 sec	AutoCAD	4 years
3	98 sec	DrawCAD	30 minutes training
4	70 sec	AutoCAD	Expert
5	34 sec	DrawCAD	1 year (developer)
Diagram 2			
User id	Time taken	Program used	Expertise level
1	363 sec	CATIA	3 years
1	225 sec	DrawCAD	30 minutes training
2	326 sec	CATIA	5 years
2	291 sec	DrawCAD	30 minutes training
3	300 sec	AutoCAD	4 years
3	218 sec	DrawCAD	30 minutes training
5	100 sec	DrawCAD	1 year (developer)

Table 1. Data from user trials

interesting example we tried out was a regular 5 pointed star (often called a pentagram). There are of course various ways in which this can be drawn. We choose one which shows off the use of DrawCAD's constraint solver. We draw the star as a sequence of 5 lines, and then put a circumcircle around it. Then we put an incircle for the inner pentagon. Finally, we merge the centers of the two circles. If the incircle and circumcircle have the same center, then it can be easily shown that the star is regular. Another interesting example we have tried out is an animation of a piston-cylinder system. For this we place the midpoints of the horizontal sides of a fixed size rectangle ("the piston") on a fixed vertical line, which constrains the piston to move only along the line if nudged. Next, a circle is drawn below the piston but centered on the same vertical line. Then, a line ("crankshaft") connects the bottom center of the piston to a point on the circle. Finally, we fix the length of the crankshaft. Now nudging the piston will cause the figure to be animated as in a piston-cylinder system.

Figure 4 shows the most complex diagram so far we have drawn using DrawCAD. We took this from [5] This took us 182 seconds.

USER TRIALS

We conducted user trials on 2 diagrams, shown in Figure 5. These diagrams were adapted from [11] and [8]

Table 1 gives the results of the trials. The AutoCAD program was version 2007, and CATIA was version V5R15. Our users 1,2,3 (identified in the table by the column "User id") were students who have been trained in the programs mentioned and have been using these for the period mentioned for their academic work. User 4 is a CAD professional.

A number of points are to be noted. The most important is that for both diagrams, DrawCAD took the least time. The best DrawCAD time beat the nearest competing time by factors of 2 and 3 on diagrams 1 and 2 respectively.

A further interesting feature is that three of our users who normally use other programs were able to quickly learn DrawCAD, and they were able to draw the diagrams faster using DrawCAD than their more familiar programs! We feel that this is indicative of the ease with which DrawCAD can be learnt.

A note is in order to explain the difference in the times taken for DrawCAD and the other programs. The first major advantage of DrawCAD is in drawing arcs. Arcs need to be drawn with various properties such as tangency, or that they pass through a point or have a certain radius. The test diagrams as well as the Figure 2 have many examples of this. Drawing these arcs is easy in DrawCAD, you merely draw, or better, just caricature them! In standard programs, selecting the specific properties requires work and also experience. The second advantage concerns *trimming*. In standard tools, often you must draw a complete circle (or even line) and then remove unwanted portions from it. In DrawCAD it suffices to just draw what you actually want. Another important advantage of DrawCAD is in indicating constraints: we feel that the combination of standard Euclidean geometry markers and nudging works nicely. A simple but important example for constraints is recognizing verticality and horizontality: this works near perfectly in DrawCAD. It is instructive to compare this to an AutoCAD feature called *ortho*, which makes it easier to draw horizontal/vertical lines. This feature must be invoked explicitly and disabled explicitly when we want to go back to drawing slanting lines. These mode changes, e.g. shifting from different lines to arcs and within arcs to different kinds of arcs and within lines to different kinds of lines slows down drawing in our opinion. And of course, how to make the mode changes and what modes exist needs to be learned. We believe that in all these cases DrawCAD is simpler and more natural to use and learn.

We examined in depth the techniques used by User 4 whose timings came closest to the best DrawCAD timings. The following important point was revealed. The "polygon drawing tool" was used to draw the hexagonal slot in Diagram 1. This is a dedicated tool in AutoCAD. Doubtless similar tools exist in other programs. However, it is to be noted that our other users did not make use of such tools, indicating that they either did not learn about the tool, or had forgotten its use. Clearly, specialized tools require a learning curve. In contrast, users who have barely seen DrawCAD can quickly acheive decent drawing times. Thus we feel that our general purpose features are as good as specialized features of standard programs, in addition to being more intuitive and easier to learn.

Finally, we will also mention some anecdotal feedback we received from these users. By and large, the users were very happy with DrawCAD. There was some discomfort, however, that DrawCAD did not have a zoom-in feature, something that is very helpful when the drawing becomes cluttered. It is true that this is a standard feature that should be added. And we will do so, along with other features discussed in Section (IMPLEMENTATION). It is possible that this may improve our timings further.

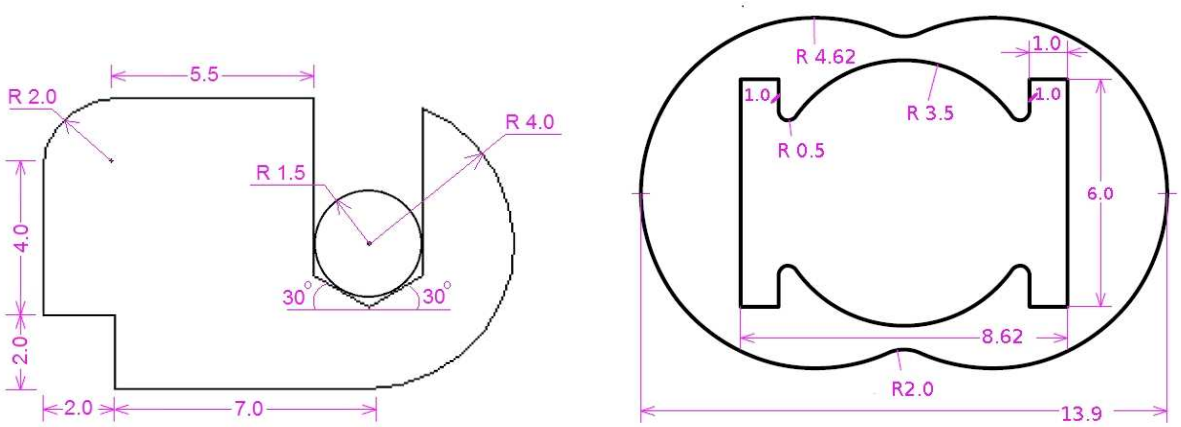


Figure 5. Test diagrams 1 and 2

PREVIOUS WORK

It can be said that sketch based interaction has come of age[1, 4]. A lot of work has been done for use of sketching in a variety of areas, mathematical formulae, class diagrams, mechanical engineering systems and also general purpose drawing programs[13].

Some of the early geometric design systems such as Pegasus[3] did use a "one stroke one segment" principle. Some of the motivations in Pegasus are similar to ours, e.g. for each recognized stroke Pegasus tried to suggest alternates. However, we feel our system is much richer than Pegasus: Pegasus doesnt appear to support arcs, nor does it support as rich a set of constraints as ours. It also does not appear to allow dimensioning. The idea of suggesting alternatives has been explored by others too, for example Murugappan et al [6]. Murugappan et al discuss an example in which an arc is drawn connected to a line. Their system graphically shows the possible relationships between the arc and the line (mainly tangency or lack of it) and the user can choose. We feel that such relationships can be more conveniently indicated in DrawCAD at the time of drawing itself. And as a fall back it can be indicated by nudging or by using markers. There is also some concern that explicitly showing too many alternatives might confuse the user[13].

The work closest to ours is *Parsketch*[7]. Parsketch is also a program for making engineering drawings using mouse-sketches as input. Parsketch also attempts infer user intent by considering constraints between geometric elements. Parsketch also has a vocabulary of gestures (similar to Euclidean geometry conventions) to explicitly specify constraints. However, it appears to us from the presentation in the paper that the program is somewhat preliminary. For one thing, the benchmark figures used are extremely simple, and even on these, the authors' comment is as follows[7]:

> "The recognition rate for gesture recognition was 90 percent. Rates for geometry recognition were very variable, depending upon the complexity of the generated stroke and the ability of the user creating the sketch."

No direct comparison is given for the time required to draw the same figure in Parsketch and in commercial softwares, though a comparison of the number of strokes/clicks is given. Also, it does not appear that Parsketch supports nudging or caricatures.

We feel that the performance of DrawCAD is clearly superior to that of Parsketch. We also feel that we understand the reason behind this: our performance derives from our premises about sketching and engineering drawing (Section (PREMISES)).

Another relevant program is Quickdraw [9]. This is meant more for casual sketching rather than engineering drawing. For this reason perhaps they do not talk about providing dimensions. The authors are more agressive than us about inferring constraints. For example, they attempt to infer the equal lengths constraints which we do not for reasons discussed in Section (Inference). And it appears that their error rates are correspondingly higher. They test their system using a set of 9 diagrams, each consisting of 3-4 elements (arcs or lines). It appears these were completed by their test participants in 90-120 minutes. To us these times seem enormous! We expect that their simple figures would be drawn much faster using DrawCAD. In one figure, DrawCAD users would have to explicitly specify the equal length constraint; in all the other figures only drawing will suffice.

We have compared DrawCAD with AutoCAD/CATIA, however it should be noted that this comparison is of course very limited. DrawCAD lacks many features, e.g. more complex curves, facilities such as cut and paste, transformations such as reflection and so on. Also, AutoCAD/CATIA have extensive special purpose support for a variety of domains from mechanical engineering parts to circuits and even flow diagrams. DrawCAD is tiny. Our program, as well as our testing is meant to only evaluate our ideas about how sketches can be turned into accurate drawings.

CONCLUDING REMARKS

Our most important contribution, of course, is to show that engineering drawings can be created much faster through

sketching than standard programs, and that sketching is much easier to learn. Our claims are based on significant size benchmarks diagrams, and we do head-to-head comparison with standard programs.

We feel the success of our program derives from our premises about the nature of engineering drawing and about the precision possible while sketching (Section (PREMISES)). Metaphorically it could be said that we model the mouse as a noisy channel. We have proposed what signals this channel can convey well (features such as co-incidence and tangency) and what signals the channel cannot (the precise shape to be drawn). Our drawing protocol tries to match the strengths of the channel and the requirements of engineering drawing: our advice to users is not to worry about the precise shape but concentrate on other features and draw a caricature. Because of this, we can set our tolerances in a relaxed manner and improve our recognition performance. As far as we can tell, our notion of nudging is also new. Yes, dynamic geometry programs all the way from Sutherland's Sketchpad[12] to Geometer's Sketchpad[10] allow geometric elements to be dragged around while the other elements adjust to maintain the constraints. However, we believe we are the first to combine this with a subsequent inference as happens in nudging.

We believe that our software architecture has room for routine and non-routine extensions. A simple example is that the edit panel could offer choices of attributes such as color and line type (whether the line is a construction line or dotted line and so on). More ambitiously, in an ongoing project, we have expanded the Edit panel to include a "spline" type element, which if selected by the user would (re)interpret the currently drawn stroke as a spline. Of course, we continue to rely on inferring whether the new stroke is tangential to other elements etc.

It is possible to think of DrawCAD as a *parametric editor*. By this we mean that a drawing constructed using DrawCAD consists of drawing elements, constraints between the elements, and element dimensions. It is possible to change the dimensions, e.g. change the length of a certain line, and get a new drawing with the given set of drawing elements satisfying the given constraints, but with the dimensions changed. A fairly simple extension that will further these ideas is to allow entering symbolic expressions when line lengths/radii or angles are entered. Entering x for one line length and $2x$ for another will effectively assert that the second line should be twice as long as the first. We believe our constraint solver will easily handle such input.

We can use DrawCAD for building interesting animations too. We have discussed how we used nudging to make an animation of a piston moving inside a cylinder and coupled to a shaft. In general, the main idea in building animations is to fix some elements, and relate other elements using constraints. A user can interact with such diagrams by nudging; the constraint management system responds to produce the animation as a byproduct.

ACKNOWLEDGMENTS

DrawCAD has evolved over time starting form the work Vishal Khandelwal, Chintan Shah, Sunil Kumar, and Prateek Sharma.

REFERENCES

1. Davis, R. Magic Paper: Sketch-Understanding Research. *Computer* (Sept. 2007), 34–41.
2. Hicklin, J., Moler, C., and Webb, P. Java Matrix Package. http://math.nist.gov/javanumerics/jama/. Last accessed on: 12/4/2012.
3. Igarashi, T., Kawachiya, S., Tanaka, H., and Matsuoka, S. Pegasus: A drawing system for rapid geometric design. In *Proceedings of CHI 98* (1998), 24–25.
4. Igarashi, T., and Zeleznik, B. Guest editors' introduction: Sketch-based interaction. *IEEE Computer Graphics and Applications 27* (2007), 26–27.
5. Mechanical Engineering, A Complete Online Guide for Every Mechanical Engineer. CAD 3D Drawing 2. http://www.mechanicalengineeringblog.com/tag/autocad-drawing/. Last accessed on: 12/4/2012.
6. Murugappan, S., Sellamani, S., and Ramani, K. Towards beautification of freehand sketches using suggestions. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modelling* (2009), 69–76.
7. Naya, F., Contero, M., Aleixos, N., and Company, P. Parsketch: A Sketch-Based Interface for a 2D Parametric Geometry Editor. In *Human-Computer Interaction, Part II, HCI 2007*, J. Jacko, Ed., vol. 4551 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, 2007, 115–124.
8. NC state university. Autocad tutorial. http://www.ncsu.edu/project/ graphicscourse/gc/ acadtut/ tutor2Anew.html. Last accessed on: 12/4/2012.
9. S. Cheema, S. Gulwani, J. L. J. QuickDraw: Improving Drawing Experience for Geometric Diagrams. In *CHI 2012* (2012).
10. Scher, D. Lifting the curtain: the evolution of the Geometer's Sketchpad. *Mathematics Educator 10*, 1 (Winter 2000), 42–48.
11. SolveSpace. Parametric 2D-3D CAD. http://solvespace.com/2d.pl. Last accessed on: 12/4/2012.
12. Sutherland, I. Sketchpad: A man-machine graphical communication system. In *AFIPS Sprint Joint Computer Conference* (1963), 329–346.
13. Zeleznik, R., Bragdon, A., Liu, C., and Forsberg, A. Lineogrammer: creating diagrams by drawing. In *Proceedings of ACM UIST 08* (2008), 161–170.