Constraint Based Drawing Tools

Tim Judkins
Auckland University
Auckland, New Zealand
Tjud007@ec.auckland.ac.nz

ABSTRACT

Constraint based drawings are useful in many applications and have been an ongoing area of research since the early 60s. The specifics of six constraint based drawing tools were examined; Spetchpad, CoDraw, Juno, Brair, EREP 2D Sketcher and Juno 2. Common problems the different tools faced were looked at, as well as the methods each used in tackling these different problems. Overall it was found that Juno 2 was the best of the tools examined, being a highly powerful and extensible system. However, it still was not completely successful at solving all problems constraint based drawing tools face, having problems with the obvious displaying of constraints on a diagram and the high level of learning required to use some of its features. There is still room for some improvements in these areas.

INTRODUCTION

Drawings with underlying constraints can be very useful things in certain applications. Ideas of constraint based drawing have been around for a long time and the concepts surrounding them are actually useful in many other applications, such as automatically generated user interfaces and interactive diagramming tools. The main work in this area has mostly been in two particular areas, creating an overall tool to create constraint based drawings with and the more underlying mathematical techniques and processed to actually recalculate values of a drawing. This report looks at some of the different specific tools that have been created, rather than the pure mathematical research. It then moves onto look at some of the common problems associated with constraint based drawing tools and how the different tools tried to overcome these problems when they were encountered.

There are 6 different tools looked at in this report, they are each introduced in the chronological order in which they were created. Firstly Sketchpad (Sutherland, 1963), considered the first constraint based drawing tool and graphical user interface for that matter, then Juno (Nelson, 1985), a tool that brought in an underlying semantic language to describe constraints. Next is CoDraw (Gross 1992) that looked mostly at the user interface and visual representation of structures of constraints. Then Brair (Gleicher and Witkin, 1993) which tried to make the whole process of creating constraint based drawings more visual and separated the establishing and maintaining of constraints. Then it looks at the EREP 2D Sketcher (Fudos, 1993) which used some similar techniques to Juno in the underlying semantic language for a drawing, and finally Juno 2 (Heydon and Nelson 1994), which built upon much of the work done in Juno, improving on it in a few particular areas.

It is also of note to mention that the work in this area has mainly been very linear, with little concurrent work. This report highlights the specific areas of problems in creating a good constraint based drawing tool and the things different tools tried in solving these common problems.

SUMMARY OF TOOLS

Sketchpad

Sutherland (1963) presented the Sketchpad system which allowed a user to interact with the computer screen with a light pen. It was a huge step in human computer interaction, being the first real graphical user interface and actually allowing direct manipulation of an image on the screen. But besides the advancements this brought in the interaction space, the entire program it centered around was in fact a constraint based drawing system. The editor itself was fairly limited in that it only allowed the drawing of points, straight lines, circles and arcs, but the real power came from the constraints that could be made between these components.

Figure 1 - User interface of the Sketchpad system

The basic constraints and relations Sketchpad included were "to make lines vertical, horizontal, parallel, or

perpendicular; to make points lie on lines or circles; to make symbols appear upright, vertically above one another or be of equal size; and to relate symbols to other drawing parts such as points and lines have been included in the system". It also had an option to see graphical icons that represented the constraints attached to particular points and lines. Over all, Sketchpad was a big stepping stone in both the area of constraint based drawings as well as the general areas of human computer interaction as a whole.

Juno

Nelson (1985) produced the constraint based solving program Juno, which presented some new ideas on the common problems already existing in the field. The key feature it had was as well as having a 'what you see is what you get' type editor, it also had an underlying semantic language to describe pictures and their constraints. The two parts were connected, so that any changes you made in the image editor would be made in the code and any changes to the code would reflect in the image. This made it so the constraints were very precisely described and added another very powerful ability of user defined procedures. Having procedures just described as a programmatic method, it was simple to create your own in the same way. This meant that a user was able to create their own procedures and apply them to a drawing if they wished.

The interface tried to give a visual representation of constraints when adding them to a drawing by using somewhat intuitive icons, such as a pencil to draw lines and a showman to freeze points. The underlying constraint solver of Juno also attempted to allow for dragging of parts of the image while keeping the constraints active, but there were problems with this which will be mentioned later.

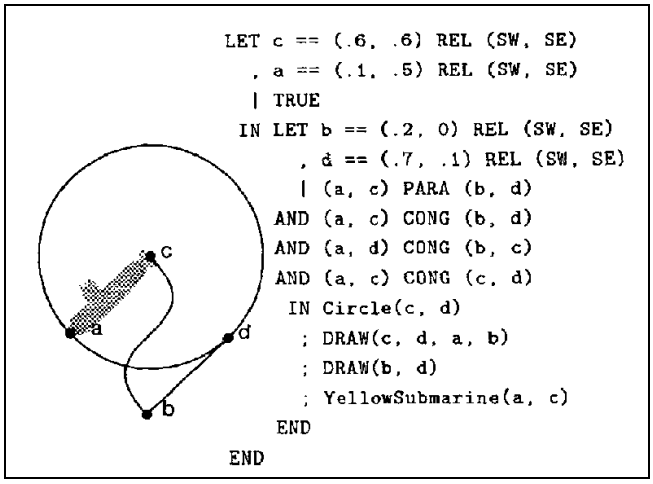

Figure 2 - An example of a Juno picture and its textual representation

CoDraw

CoDraw by Gross (1992) was a constraint based drawing tool that did not provide too much in the way of different constraints itself, but was more of a exploration into better ways to represent things and build an underlying system. Although there were not many constraints available to use, CoDraw actually allowed users to define new constraints in the program itself using linear algebra.

CoDraw's interface was also an aspect of importance in the design. The drawing of diagrams was wanted to be a simple point and click method, with constraints displayed graphically so they can be seen on a diagram. The tool had different icons for easy selection when drawing shapes, as did the different constraints which were also added to a diagram by simple pointing and clicking. The constraints were also visually displayed on the diagram with different representations so the user could see how they applied to a drawing and predict how it would behave when modified.

There were also many different pallets that could be opened to display and edit different areas of information, such as a constraint graph displaying how constraints linked together and a part graph showing assembly structures. These gave the user access to see much of the information about a drawing, without cluttering up the screen by trying to show them all at once by default.

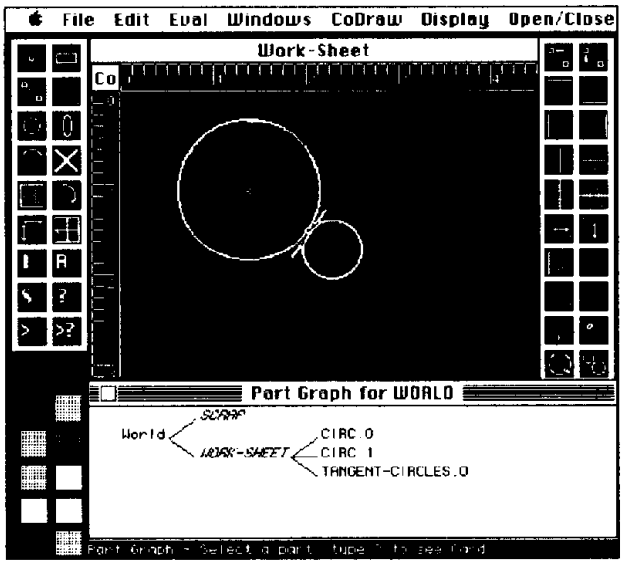

Figure 3 - CoDraw's worksheet, part graph and tool pallets

Brair

Further down the track of work on constraint based drawing editors came Brair, created by Gleicher and Witkin (1993). The main constraint based drawing problems Brair focused on improving upon was making the creation of constraints much easier and their visualization more obvious. Different constraints were represented on the image with different icons, making them easily identifiable by just looking at them. The interface also used many horizontal and vertical lines to display locations of lines on the drawing and parallel constraints between them.

In their constraint solver, Brair also managed to keep constraints across the drawing during dragging parts

updated to create smooth transitions that showed how other parts are affected, though this did stumble when drawings became too complex. They also tried to separate the declaration of constraints from the modification process, meaning that having been already solved on declaration, when being modified they just had to be kept in a solved state, which was a simpler process.

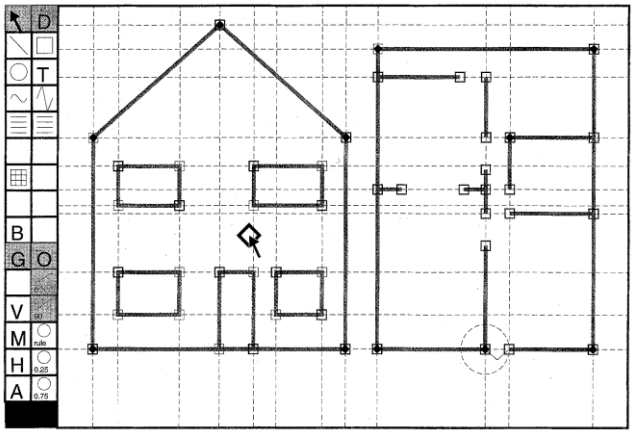

Figure 4 - Brair editing a constrained drawing

EREP 2D Sketcher

Around the same time Brair was created, Fudos (1993) made the Editable Representations 2D Sketcher, or EREP. It used a similar idea to the Juno editor where the user makes an initial sketch and adds constraints to it in the drawing interface, then this is transformed into a high level textual description which contains all the information needed to represent the image. The constraint solver would then take this high level description, find the solution and return the result in the same high level description.

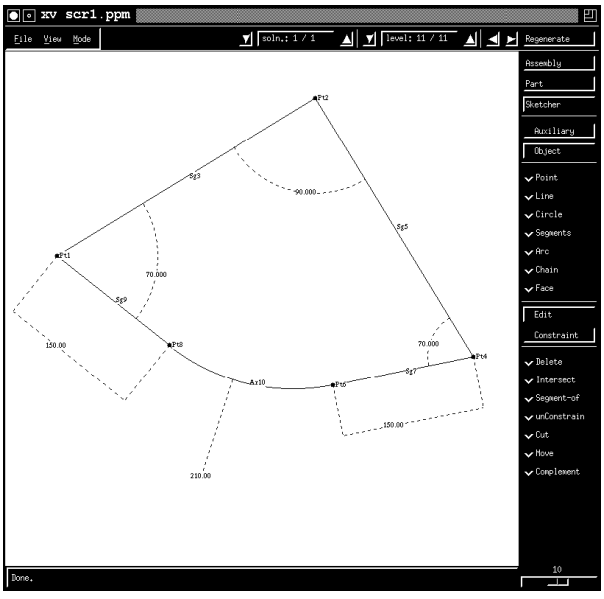

Figure 5 - The EREP user interface panel

One particlaurly nice feature the EREP had was that if a constrained drawing that had been solved found multiple possible solutions, it let the user choose which one was correct. This was much better than a solver finding what it thought was the correct solution and changing the drawing to an incorrect state. The constraint solver was also able to identify clusters of constraints that affected each other, to better work out what needed to be changed on an edit to the diagram.

Juno 2

The work in the original Juno was furthered by Heydon and Nelson (1994) to create Juno 2. It used the same system of a double view editor, with the graphical representation and textual language describing the image and its constraints. By having both these views visible at the same time, someone who knew the meaning behind the text could quite easily see what constraints existed in a diagram without having to clutter it up with iconic representations. The underlying language was also very extensible, allowing for user defined constraints to be written and then used in the editor.

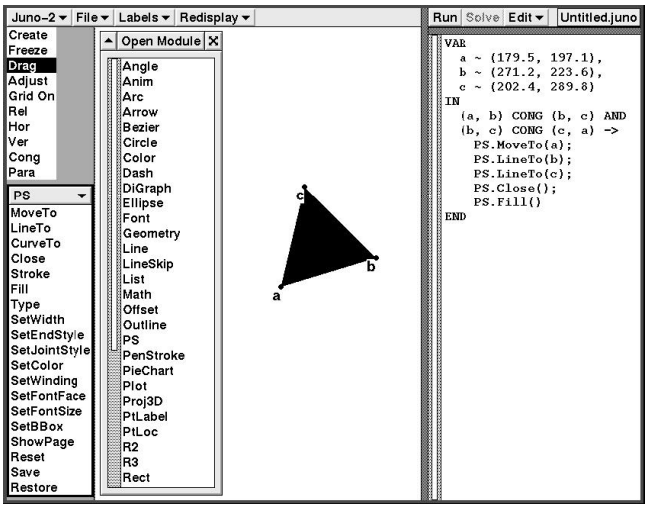

Figure 6 - Double view of the Juno 2 editor

The constraint solver for Juno 2 was also very powerful, in that it could deal with many of the harder problems such as cyclic constraints. It would also preprocess to reduce the number of variables the solver had to deal with as well as a number of other techniques to speed up the whole solving process. This meant that the solver worked very fast and complex diagrams could be changed and dragged with near real time changes. Another interesting factor of the constraint solver was the concept of hints. Hints helped the common problem many constraint solvers hit upon when there were multiple solutions to a problem, of the solver finding a correct solution that was not the one the user was intending. Juno 2's hints used user supplied points or the existing locations of points for an approximate area to look for a numeric solution around. This was surprisingly good

at speeding things up and finding the solution a user was intending the solver to find.

COMMON PROBLEMS

Displaying Constraints

One of the hardest areas in a constraint based drawing tool is the visual representation of already created constraints in a diagram to a user. The two main methods groups have tried to solve this problem have been to annotate the diagram as constraints are added, or to separate the constraints out to a separate view.

Sketchpad largely did not show the constraints on the interface and although it was possible to have it display constraints relating to points and lines the graphical representation was very unclear as to how they related to the diagram. Juno, EREP, Juno 2 and CoDraw all separated out the constraints to separate views, although CoDraw did also give some representation on the diagram too. CoDraw also allowed many other bits of information to be viewed visually about constraints and how they interrelated. The problem with the separation of constraints from the diagram they relate to was that it often takes hard mental operations to work out how they apply to the drawing. It also creates hidden dependencies between the two, with no visual information as to how they were connected but still having the two inexorably tied together.

Brair took the other option of trying to display all constraints on a diagram using icons and lines. There was however problems with this too as drawings became larger and more complex. As all constraints were displayed on the screen, for larger drawings the display would get very messy and cluttered. It also suffered from not being as clear as to how some of the constraints related across a diagram.

Constraint Declaration

Another area that different tools have tried to improve upon is the process of adding constraints to a diagram. Often constraints are simple for you to imagine them as you want them to be but hard to add to a tool as intended. Also many of the tools suffered from having only a few constraints built in.

Generally all the tools tried to work the declaration of constraints into the visual editor. Juno, EREP and Juno 2 also allowed for the declaring of constraints to happen in their underlying language of the diagrams, which while initially unintuitive was powerful once you got the hang of it. CoDraw and Juno 2 allowed for user defined constraints to be added, giving a major added power to them as tools, making the systems more extensible and giving larger uses.

Initial Constraint Solving

The initial solving of constraints was an issue for most of the earlier tools, where their solving system would find unexpected solutions and the whole diagram could jump unexpectedly. Sketchpad, Juno, Brair and CoDraw all suffered from this problem.

EREP presented one solution to the problem, of detecting when a constrained system had roots involved, meaning that there were multiple solutions possible. It would then tell the user this and give them the option of choosing between the different possible solutions. The problem was that this could not work for numeric solving methods needed for some constraints to be solved.

Juno 2's method of hints was a different approach to the problem that worked rather well. Having a solver that could essentially be told 'the solution should be somewhere around here' made it so solving almost always lead to the intended result the user wanted.

Constraint Maintenance

An issue all constraint based drawing tools have had to deal with as well is the problem of how they maintain constraints. It is generally preferred for the user to be able to grab points or lines of the diagram and drag them, with the constraints of the diagram remaining consistent throughout. But as this essentially means solving all the constraints for every single step while moving, for a complex diagram which takes a significant time for the solver to compute a solution, this is simply not possible.

All of Sketchpad, CoDraw, Juno, Brair and EREP suffered from this problem, being able to deal with smaller diagrams but having major slowdown as the diagrams became more and more complex. Juno 2's solver however managed to significantly improve its solving through the preprocessing and variable packing and unpacking to be able to handle reasonably complex diagram movement in real time. Its hint system was also good for this, being able to supply the previous positions of points as the hints for finding their new locations as they moved.

General Limitations

Many of the drawing tools were limited in what they could actually use in a drawing. Like Sketchpad, many of the programs were limited to circles, arcs, lines and points, with curved lines being much more difficult to create. The learning curve for all the different tools was also quite high, with none of them succeeding in making the drawing process intuitive and easy, as well as keeping it powerful. This is especially evident in some of the more complex operations such as the declaration of new constraints in Juno 2 being recommended to be left only to experts to carry out.

SUMMARY

Across all six tools examined, all faced many common problems they had to deal with. Often the tools chose to deal with these problems in different ways and most tended to focus more on trying to solve a particular subset of problems. The problems that tools have had the least success in solving seem to be the intuitive display of

existing constraints in a drawing, and the problem of actual methods of solving constraints, which is still ongoing research today.

Of the tools examined Juno 2 was the best overall, with good ability to tackle complex constraints and a high extensibility. Although there were hidden dependencies between the code view and the pictorial view of a drawing, this helped to avoid much of the clutter that tools such as Brair faced when there were a large number of constraints. However what it did not manage to achieve, along with none of the other tools, was to give a nice way to link the constraints to the actual drawing. Some areas in Juno 2, such as the creation of entirely new constraints, are also very difficult and require large amounts of learning to do. Although, once a user learns how to use it fully, Juno 2 becomes a very powerful constraint based drawing tool that much can be done with.

DISCUSSION AND FUTURE WORK

Although there has been more recent work on the solving methods in constraint based systems, since the creation of Juno 2 there has not been a lot of work on creating more drawing tools. Research has more seemed to focus on other application of the constraint systems, such as user interfaces.

There has been similar appliance of the ideas into computer aided design tools for areas such as making precise models of objects for engineering applications. A few of the tools examined actually mentioned the ideas of moving their constraint solving from two dimensions into three for their future work, which is likely what lead to the creation of some of these computer aided design tools. Juno 2's solver could in fact do this already and is a nice example of being able to scale a well made project.

But it seems that there has been no real demand for a new tool to specifically make two dimensional constraint based drawings. Generally, other than a nicer more intuitive interface, Juno 2 did a very good job in creating a constraint based drawing tool. It was highly extensible and was very powerful in what it could do. It seemed to cause a lull in the research field of making an actual editor and moved research more into use in other applications and to the pure solvers.

REFERENCES

1. Sutherland, I. E. (1963, May 21-23). *SketchPad: A man-machine graphical communication.* Paper presented at the AFIPS Spring Joint Computer Conference (pp. 329-346) Detroit, Michigan. Online reproduction retrieved March 28, 2008, from http://www.guidebookgallery.org/articles/sketchpadamanmachinegraphicalcommunicationsystem
2. Nelson, G. (1985) *Juno, a constraint-based graphics system***.** International Conference on Computer Graphics and Interactive Techniques (pp. 235 - 243). New York, USA: ACM. http://delivery.acm.org.ezproxy.auckland.ac.nz/10.1145/330000/325241/p235-nelson.pdf?key1=325241&key2=7849396021&coll=GUIDE&dl=GUIDE&CFID=22224249&CFTOKEN=47937893
3. Gross, M. D. (1992) *Graphical constraints in CoDraw.* 1992 IEEE Workshop on Visual Languages (pp.81 - 87). Seattle, USA http://ieeexplore.ieee.org/iel2/895/6833/00275780.pdf?tp=&isnumber=&arnumber=275780
4. Fudos, I. (1993) *Editable Representations for 2d Geometric Design*. Master's thesis, Purdue University http://citeseer.ist.psu.edu/cache/papers/cs/1432/ftp:zSzzSzftp.cs.purdue.eduzSzpubzSzfudoszSzMsczSzMsc.pdf/fudos93editable.pdf
5. Gleicher, M., & Witkin, A. (1993) *Drawing With Constraints.* The Visual Computer 11(1), 39-51. http://citeseer.comp.nus.edu.sg/cache/papers/cs/1814/http:zSzzSzwww.gleicher.comzSzmikezSzPaperszSzbriar.pdf/drawing-with-constraints.pdf
6. Heydon, A., & Nelson, G. (1994) *The Juno-2 Constraint-Based Drawing Editor.* Digital Systems Research Center. California, USA http://coblitz.codeen.org:3125/citeseer.ist.psu.edu/cache/papers/cs/133/ftp:zSzzSzgatekeeper.dec.comzSzpubzSzDECzSzSRCzSzresearch-reportszSzSRC-131a.pdf/heydon94juno.pdf