Visualizing Lists in CAD: A Step Towards Programming in the Model

Maryam M. Maleki, Lyn Bartram

IAT 814: Knowledge, Visualization, and Communication
Spring 2009

Abstract

Lists of points are used in some Computer Aided Design systems to create simple or complex shapes such as lines, curves, and surfaces. During the task of making these lists, users can benefit from visual feedback that shows them different attributes of the lists including membership, order, and hierarchy. Visualizing these attributes presents a challenge that requires exploration and investigation of several different visualization design principles. A few of those principles are presented in this project in the form of four brushing techniques. In addition, a prototype was implemented for this project that allowed the simulation of different situations and comparison between these brushing techniques.

1 Introduction

Architects and engineers use Computer Aided Design (CAD) systems to design and develop 2D and 3D models of their final design, whether it is an engine, a house, or an airport. In more complex projects, designers sometimes have to use computer programming in CAD to do the tasks that can only be done or can only be done efficiently by using programming. In those cases, designers become end-user programmers: non-professional programmers who need to write code to get a task done in the domain of their own expertise. As it is known in the literature, there are many issues in end-user programming in general and also issues that are specific to CAD.

For my master's thesis, I am working on end-user programming in CAD. More specifically, I am working on *programming in the model* in CAD as a means of using designers' visual and spatial abilities in end-user programming. In other words, we are working on bringing computer programming closer to the model in CAD to minimize the mental shift between the two tasks of design and programming.

In a parallel course, IAT 812 Cognition, Learning, and Collaboration, I explored this idea by choosing *lists* as a programming element and implementing a prototype in Generative Components (a CAD system from Bentley Systems) to study the effects of bringing the task of making lists to the model in CAD. In this course, IAT 814 Knowledge, Visualization, and Communication, I looked at the same problem from a visualization perspective, in order to find out some of the challenges in visualizing lists in the model and explore some of the solutions.

For this purpose, a prototype was created in Processing that enabled us to explore different brushing techniques in several different situations. In this paper, we explain the project and some of the visualization challenges specific to this project. We then introduce some of the visualization principles that can be used to address those issues. Then a brief description of the prototype is included, along with thorough explanation of the brushing techniques used in this project, followed by conclusions and ideas for the future works.

2 Visualization challenges

In some CAD systems (Generative Components in particular), lists of points are used to make shapes such as line strings and curves. Nested lists are used to create surfaces. The difference between a list and a set is that order is important in the list. A curve created on points 1, 2, and 3 looks different from the curve created on points 2, 1, and 3.

In this project we bring the task of making lists to the model in order to investigate *programming in the model* as an approach to end user programming in CAD. In other words, users make the lists in the model by simply selecting the points and adding them to the lists, without having to worry about the syntax (brackets and commas) in the script editor. However, without any visual feedback, users cannot keep track of the members of the lists and their order and it is equally difficult to modify lists, as users have to use the textual representation of the lists and the names of the points to connect them together.

There is important information about the lists that needs to be displayed to the user. Here are the three most important ones:

- *membership*: Users need to know which points belong to a list and also what lists a point belongs to.
- *order*: As mentioned above, order is important in the lists, as it affects the shape of the final model. Therefore we have to show the order (or the history of selection) of the members of each list.
- *hierarchy*: Nested lists (or two dimensional arrays) are commonly used to make more complex geometries. So there may be lists that contain other lists and this hierarchy among lists has to be visually represented.

In addition to membership, order, and hierarchy, there are other issues that we need to consider. For instance:

- How does this visual feedback combine with the CAD model? Does it interfere with the model? Which technique works better with wireframe models and which one works better with a rendered or shaded model?
- How do these lists work with each other in the model when they share points and overlap?
- How does the visualization of the lists connect with other types of representation, such as list icons or list text?
- How does it scale? Does it work for large lists? Does it work for large or complex models?

In order to find the answer to some of these questions, we explored several brushing and linking techniques to visually represent lists in the model view. Different information visualization design principles were considered for this purpose. Some worked and some didn't. Here is a brief survey of those design elements, mostly cited from Ware (2005) and Bartram (2009):

- *Proximity*

  Proximity cannot be used in this project simply because the position of the points in the model are important to the geometry and cannot be changed to group the points together.

- *Similarity*

  Similarity in shape, color, and size can be used to group the points. We explore similarity in color in this project. However, similarity in shape was not used for the points due to issues with overlapping lists when one point has to have more than one shape. Shape was used in combination with connectedness for arrowheads.

- *Connectedness*

  Connectedness is argued to be a fundamental Gestalt principle and more powerful in grouping objects than color, size, or shape. It is used here in the form of lines that connect members of a list. However, connectedness alone is not enough for distinguishing the lists from each other. When a point belongs to more than

one list, there is no way to know which line to follow from that point. Color, shape or other elements should be used with connectedness to make it effective.

- *Closure*

  Closed contours divide the space into inside and outside. The objects inside or outside the contour are perceived as a group. We use a contour for each list that wraps around the points in the list. Contours combine with each other very well and are still distinguishable. Color and texture can be used for more complex models with multiple overlapping contours.

- *Transparency*

  If contours are filled regions, transparency is necessary to prevent them from occluding each other. It can also be used with shapes, when several shapes (arrows) are in the same place.

- *Color*

  Color can be a powerful visual cue in grouping objects, if used correctly. It can also be used to link the lists in the model view and other representations of them by simply giving them the same colors. In this project we only use three colors of red, green, and blue for the lists. More investigation is required to determine the effect of color and to find out what colors work better for this visualization problem. Color deficiency options must be considered as well.

- *Size*

  Size can be used to display order in each list. It can be in the form of point size or line thickness.

- *Direction*

  Direction is used here in combination with connectedness to show the order in the lists.

3 The prototype

To further investigate the challenges of visualizing lists in the model view in CAD systems, we implemented an interactive prototype in Processing.

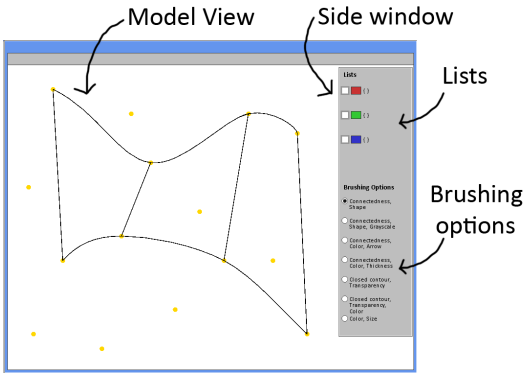


Figure 1: The prototype

The interface and functions are similar to Generative Components, so that we can compare them in future studies. The model (points, curves, ...) is displayed in the model view. The side window is where list options and brushing options are located (Figure 1).

In order to interact with a list, it must be activated by clicking on its icon in the side window (Figure 2). A thick dark border appears around a list icon that is activated. Only one list can be activated at any time. So clicking on another list will deactivate the first list. A list can also be deactivated by clicking on its own icon.

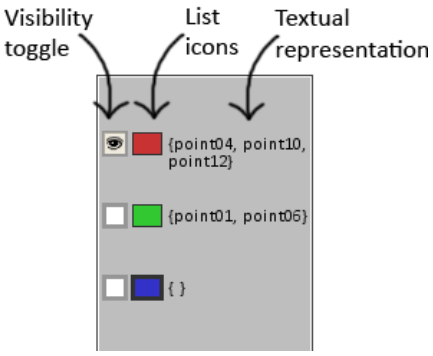


Figure 2: List options in the prototype

When a list is activated, points in the model view

can be added to it by holding down the control key and clicking on the points. Right clicking on the points removes them from the active list. The textual representation of each list is displayed next to the list icon.

Lists can also contain other lists and make two dimensional lists. Control click on a list icon adds it to the active list and right click removes it from that list.

Brushing techniques are used to visually represent the lists in the model view. The visibility toggle icon next to the list icon can be used to turn the list on and off. Hovering the mouse over the list icon turns the list on temporarily. Additionally, hovering the mouse over a point in the model view shows the lists it belongs to in addition to its name.

Different brushing techniques are provided in the prototype for comparison. These techniques can be explored by using the radio buttons in the Brushing Option section (Figure 2).

The prototype can be accessed at http://www.sfu.ca/~mmaleki/iat814/FinalProject/. Unfortunately, the prototype is not bug free. A list of known bugs is provided at the end of this paper.

4 Brushing techniques

The brushing techniques that we explored in this project can be categorized into three major groups, based on the visualization principle used to link the members of the lists. The three principles are color, connectedness, and closed contour. However, connectedness alone does not suffice to separate the lists from each other, because when two lines go out of a point, there is no visual clue which one to follow for each list. Therefore, we used shape and color as additional visual cues with connectedness. The final categories are:

- Color
- Connectedness and shape
- Connectedness and color
- Closed contour

In this section, we describe each technique and how it does or does not address the visual problems of the lists and also the specific issues in each technique.

4.1 Color

- **Membership**

  In this technique, the color of the points is defined by the list they belong to (Figure 3).

- **Order**

  Size of the points is used to show order in the list. The position (index) of a point in the list determines its size. The smallest point in a list is the first member of that list (Figure 3).

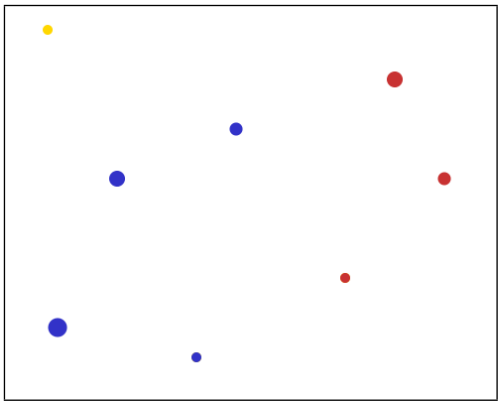


Figure 3: Color: Membership and order

- **Hierarchy**

  At this point, we do not have any way of showing hierarchy of the lists in this technique. A possible solution can be an additional larger point under the point (it looks like a ring) that represents the nested list. However, the size of that point has to be adjusted to the size of the first point, base on the order of the list.

- **Connecting with the textual lists**

  The color of each group of points matches the color of the list they belong to. So it is easy to make a connection between the list in the

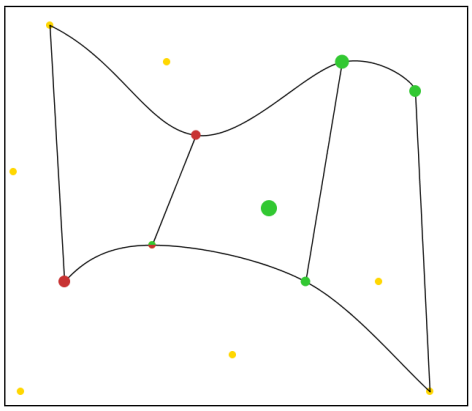


Figure 4: Color: wireframe model

model and the list in the side window and its textual representation.

- **On a wireframe model**

  The smaller points may get lost in a complex model and larger points may interfere with the model itself or get confused with the elements of the model, such as the columns. They work in less complex models with colors that are different from the colors used in the model (Figure 4).

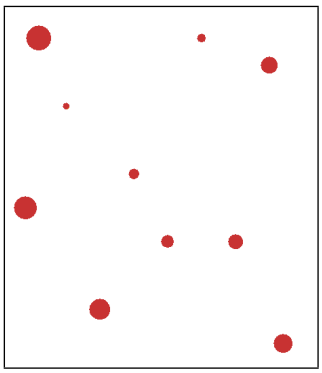
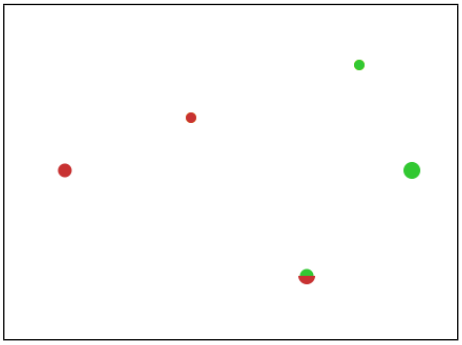


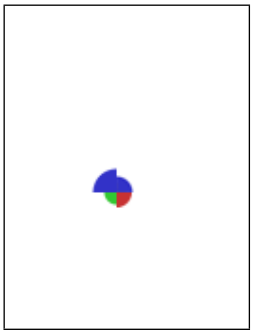
Figure 5: Color: scaling

- **Scaling**

  This technique works for lists of up to 10 points. It does not scale well for larger lists, as the size of the points gets too big and make the model unreadable (Figure 5).



(a) Two lists share a point



(b) Several lists share a point

Figure 6: Color: Combining lists

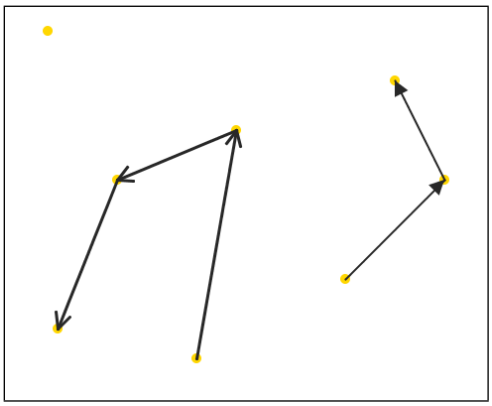
- **Combining with other lists**

  In this technique, when a point belongs to more than one list, the area of the circle is divided into several sections depending on the number of lists that share that point. Each section follows the color and size of the point in one of the lists. However, this method does not scale well for more than 3 or 4 lists. It gets harder and harder to follow the membership and the order when the number of lists goes up (Figure 6).

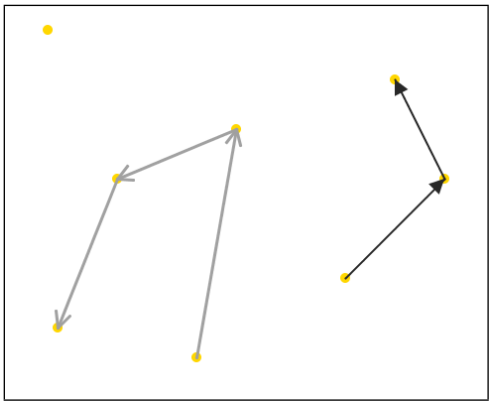
4.2 Connectedness and shape

- **Membership**

Members of a list are connected by a line. In order to distinguish between multiple lists, each list has a different shape associated to it that works like an arrow. Both lines and shapes define the membership of the points. Furthermore, we added a third element of color (only shades of grey) to this technique to see its effect on this brushing technique (Figure 7).



(a) No color



(b) With shades of grey

Figure 7: Connectedness and shape: Membership and order

- **Order**

  The shapes in this technique also carry the role of showing order in the lists. They are like arrows that show the history of user's selection in each list (Figure 7).

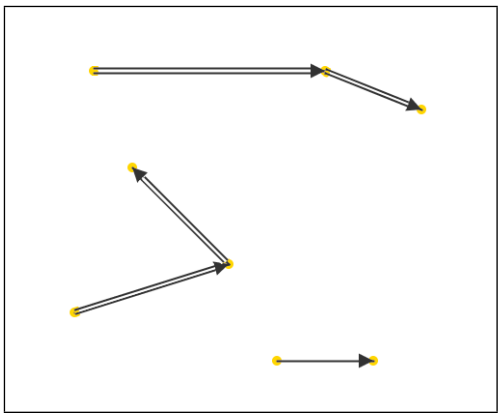


Figure 8: Connectedness and shape: Hierarchy

- **Hierarchy**

  One way of showing hierarchy with connectedness is to use double lines for inside lists. However, this method does not scale well for lists with more depth (three dimensional lists and more) (Figure 8). This technique is not included in the prototype.

- **Connecting with the textual lists**

  There is no good way of connecting the lists in the model with the lists in the side window and the textual lists, except for using the shapes to make some kind of legend. We don't know how clear it will be for the users without implementing and testing it.

- **On a wireframe model**

  Wireframe architectural and engineering models usually contain lines and curves that create forms. The lines used in this technique for brushing are very similar to those of the model and can easily be confused with them. It is hard to both follow the lines to find the lists and comprehend the model when the lists are visible (Figure 9).

- **Scaling**

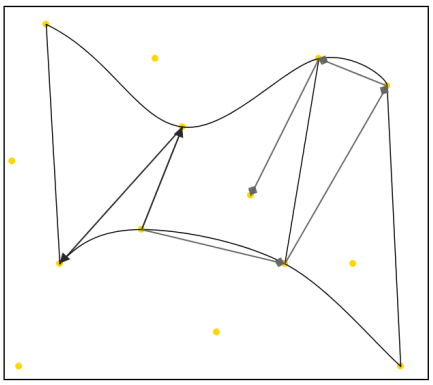


Figure 9: Connectedness and shape: Wireframe model

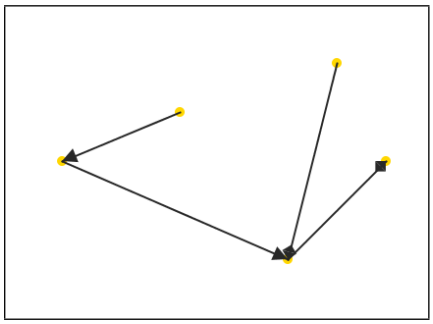
The technique scales well for large lists. The only problem happens when there are many lists to brush, as we may run out of shapes to distinguish them.

- **Combining with other lists**

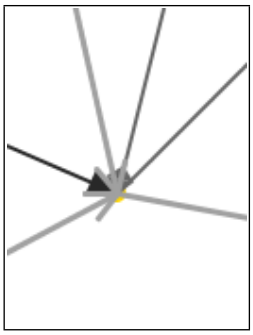
  When more than two or three lists share a point, the shapes get cluttered over each other and occlude each other, to a degree that they can be completely useless and unreadable, especially in the first method that everything has the same color (Figure 10).

- **Other issues**

  A line is only meaningful when there are two points for it to connect. So when there is only one point in the list, there can be no line. In order to solve this issue we can put a single shape on the point (like an arrow without its line) to show which list the point belongs to. As soon as the second point is added to the list, this single shape disappears and the points are connected and the shape follows the order of selection (Figure 11).



(a) Two lists share a point



(b) Several lists share a point

Figure 10: Connectedness and shape: Combining lists

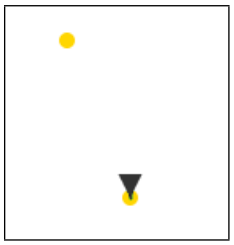
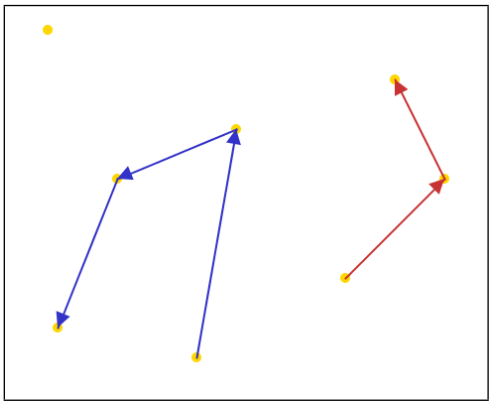


Figure 11: Connectedness and shape: Single member lists

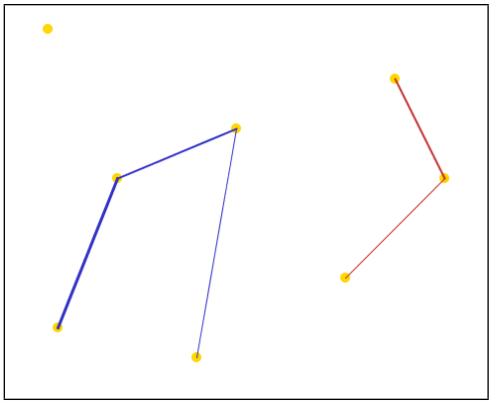
4.3 Connectedness and color

- **Membership**

  Membership is shown by a line that connects the members of a list. However, connectedness alone is not enough to separate the lists in the model. Color is added to the connectedness to assist in visualizing membership (Figure 12).



(a) Arrows show order



(b) Line thickness shows order

Figure 12: Connectedness and color: Membership and order

- **Order**

  Two different techniques are used here to show order. In the first technique, arrows are used to show the history of selection and consequently the order of the lists. In the second technique, line thickness is used for that purpose. The thickness of the lines depend on their position (index) in the lists. The thinnest line is always the first member of the list (Figure 12).

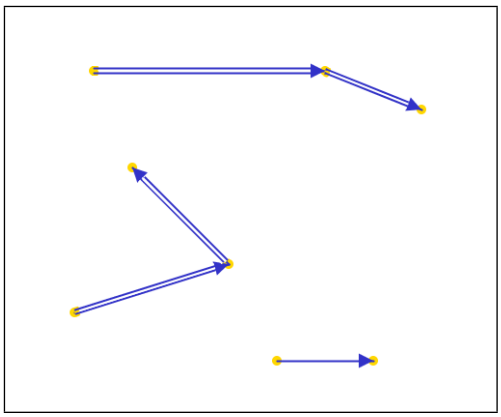


Figure 13: Connectedness and color: Hierarchy

- **Hierarchy**

  Like the previous method, hierarchy can be shown by double and triple lines. The problem occurs for lists with more than two levels of hierarchy (Figure 13).

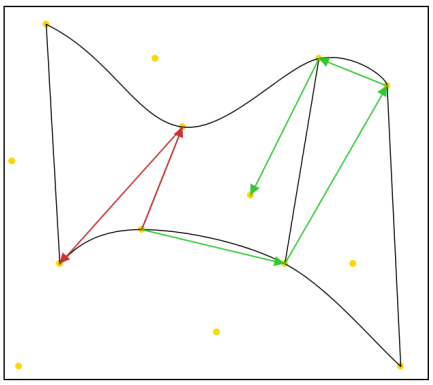
- **Connecting with the textual lists**

  Color can be a good visual cue to connect the lists in the model with their corresponding lists in the side window and their textual representations.

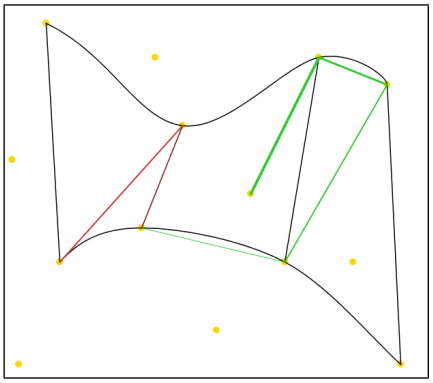
- **On a wireframe model**

  Again, lines in this brushing technique can interfere with lines and curves in the model. I find it specifically harder to follow the line thickness in the second technique when combined with a wireframe model. However, if the wireframe model is in greyscale and no other color is used in it, the color of the brushing lines can separate them from the model better than the previous technique (Figure 14).

- **Scaling**



(a) With arrows



(b) With line thickness

Figure 14: Connectedness and color: Wireframe model

There is scaling issue for the line thickness when the lists contain too many points and the lines get too thick (Figure 15).

- **Combining with other lists**

  The arrows cover each other and become hard to read when more than one or two lists share a point. But the color helps make it a little better than the previous method (connectedness and shape) to follow the lists. Thickness seems to be much better in scaling in this situation as there are no arrows and lines are easier to follow (Figure 18).

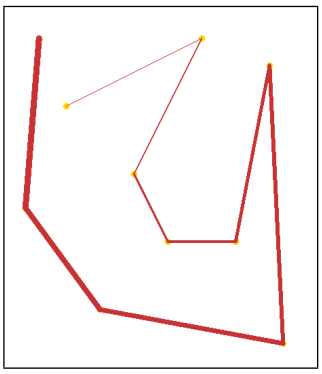


Figure 15: Connectedness and color: Scaling

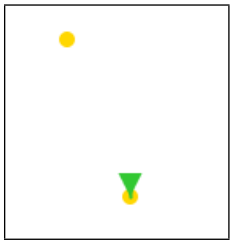


Figure 16: Connectedness and color: Single member lists

- **Other issues**

  The same problem applies here for the lists with just one member, as there is no way to produce a line with only one point. A single arrow can be used to show the membership for that point until the second member is added to the list (Figure 16).

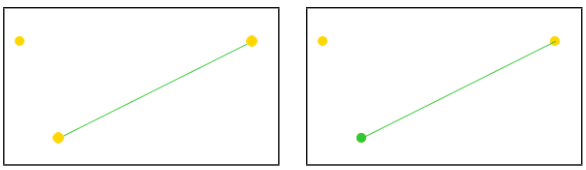
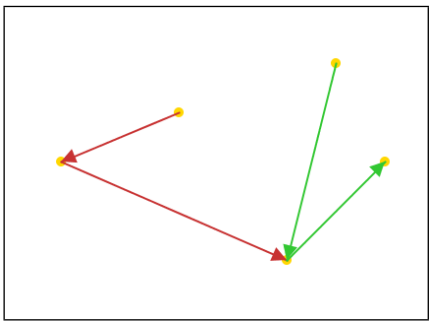
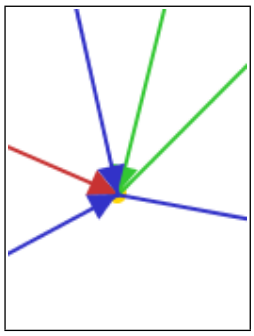


Figure 17: Connectedness and color: Two-member lists and thickness

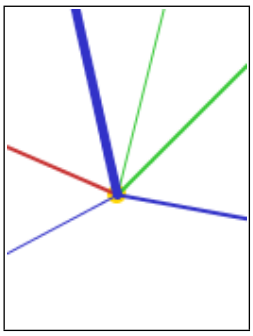
However, the thickness method still does not



(a) Two lists share a point

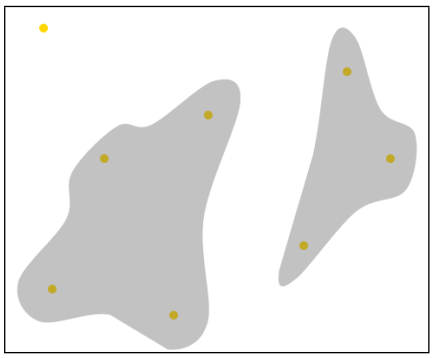


(b) Arrows: Several lists share a point

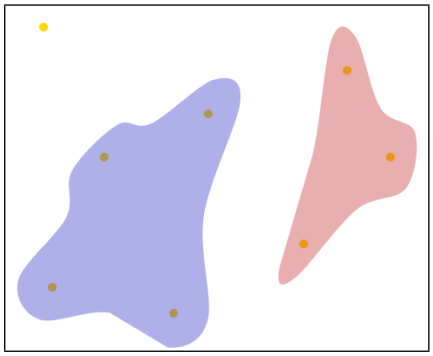


(c) Thickness: Several lists share a point

Figure 18: Connectedness and color: Combining lists



(a)



(b)

Figure 19: Closed contour: Membership

work with two points. Line thickness is meaningful only when there are more than one line to compare. Point color can be used here to show the membership in single member lists and to show order in two-member lists (Figure 17).

4.4 Closed contour

- **Membership**

  A closed contour wraps around the members of a list. The contours can have the same color or have different colors that reflect different lists. Both methods are capable of showing membership. Transparency is also used to allow the contours to overlap without occluding each other (Figure 19).

- **Order**

  At this point, there is no way of showing order in this technique. A possible solution is adding numbers besides points that reflect their position (index) in the list.

- **Hierarchy**

  When a list contains another list, there is a contour for the inside list and another one that wraps around that list and any other member of the bigger list (Figure 20).

- **Connecting with the textual lists**

  The monocolor contours do a good job in linking the points in the lists and showing hierarchy as well as the colored ones do. But they do not

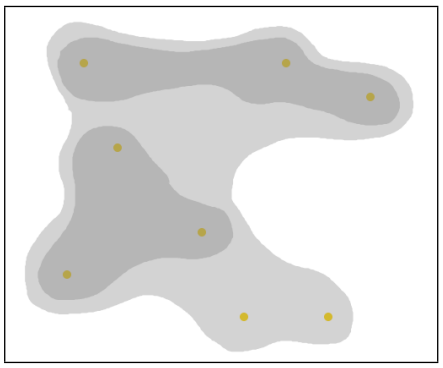


Figure 20: Closed contour: Hierarchy

connect the lists in the model window with the lists in the side window and the textual representations of them. Color works better in doing that.

- **On a wireframe model**

  Since wireframe models only contain curves, lines, and points, filled closed contours seem to work well with those models. they do not interfere with the model and are easy to follow (Figure 22). More investigation is necessary for other types of models, such as shaded and rendered models.

- **Scaling**

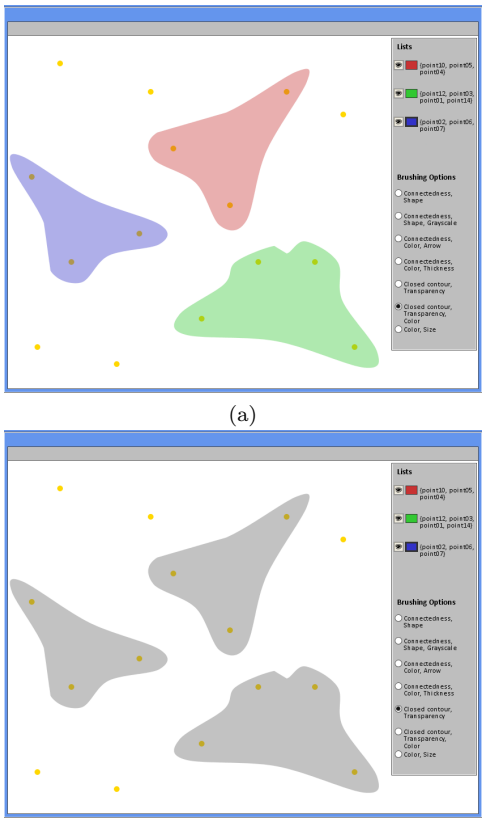
  Scaling issues may apply to very large lists, due to the large area of contours that covers large parts of the model.

- **Combining with other lists**

  Transparent contours seem to be more effective than connectedness and size when lists overlap in the model. Even without color separation, contours are easy to follow in such situations, as we are good at following curves (Figure 23).

- **Other issues**

  When there are many objects in the model and lists contain points that are located far apart from each other in the model, contours may have to get strange and more complex shapes



(a)

(b)

Figure 21: Closed contour: Connecting with other types of data

to avoid covering the points that don't belong to their lists. It may also become hard to follow them when they get too big.

5 Conclusions and future work

Our interaction with the prototype and observation of the brushing techniques show that they all have their strengths and weaknesses. Color works well in grouping the points and has the least interference with the CAD model, but does not scale well and also does not work for the combination of lists. Connectedness in general is a good way of showing

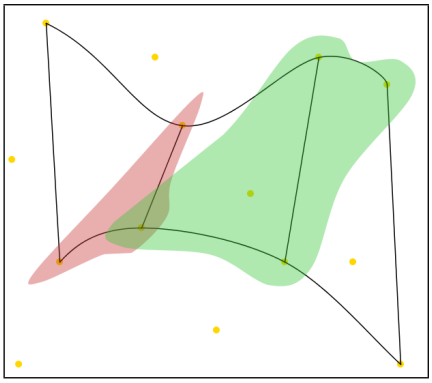


Figure 22: Closed contour: Wireframe model

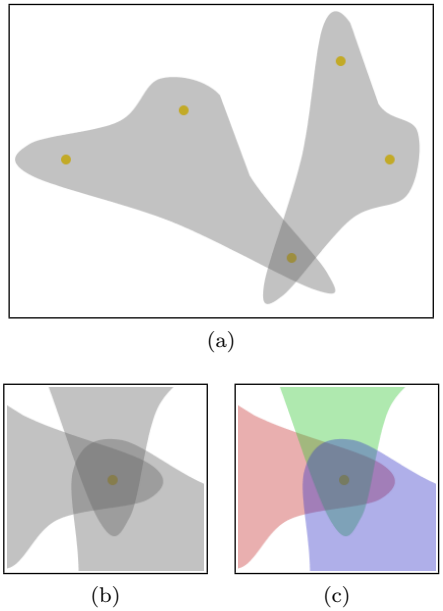


Figure 23: Closed contour: Combining lists

membership, and order, but has the most interference with wireframe models. closed contour on the other hand works well with the wireframe model and also in combination with other contours, but does not scale and we still don't have a way of showing order in this technique.

In addition to our exploration of these techniques, we need to set up experiments in the future to test the techniques. In the experiment, we may ask participants to answer questions about membership, order and hierarchy in different situations. The prototype can be used to prepare the situations for these tests. It can also be used for interactive experiments in which participants make the lists themselves.

However, points are not the only members of the lists in CAD. Lines, curves, and other shapes can also be used in lists to create more complex objects such as solid bodies. Brushing these shapes may be even more challenging and needs to be investigated.

Another area for future research is visualizing lists in 3-dimensional models, as most CAD models are 3D objects and it is common for CAD users to interact with them in 3D views. Some of these techniques may not be feasible in 3D and some may be hard to use.

At the end, we can conclude that each technique has its own cons and pros and we haven't yet found one that answers all our questions and solves all of the visualization issues in this project. Using a combination of these techniques may be the answer, in a way that each technique is used for what it is good at.

6 Known bugs and issues in the prototype

- Color
  - When points are shared between the blue and the red lists, the red list doesn't show up in the model.
  - Hierarchy is not implemented in this technique.
- Connectedness

  - Lines cover each other when they share both the end point and the start point. Blue is alway the dominant color. Green is the next dominant color.
  - Hierarchy is not implemented in this technique.

- Closed contour
  - Hierarchy does not function as it is supposed to. It wraps around each of the lists, but doesn't wrap around them all.
  - The contour changes shape depending on the order of selection. Clockwise and counter clockwise selection of three points result in different shapes in the contour.
  - Contours sometimes twist.
  - Contours cover points that are in the middle of the list members, even if they do not belong to the list.

References

Bartram, L. (2009). IAT 814: Knowledge, visualization, and communication. graduate course at school of interactive arts and technology, simon fraser university.

Ware, C. (2005). *Information visualization : perception for design*. Morgan Kaufmann, San Francisco Calif., 2. ed., [Nachdr.] edition.