Sketch Recognition Mini-project 2 Report

Team Members: Aniket Bonde, Mayank Sharma, Niloofar Zarei

In this project, we implement a sketch recognition system in JavaScript to recognize trusses. We first describe previous work in this area, then move on elaborate on our methodology. Results are given based on the system's achieved accuracy, recall, and more importantly, f-measure which will be our main performance metric.

1 - Introduction:

Trusses are basic anatomic formations that appear in various structures to ensure stability, such as bridges and airplane wings. They can appear frequently in domains like Mechanical engineering, so recognizing truss sketches can be a crucial feature for sketch-based systems that operate in these domains. One example of such systems is Mechanix, which is a tool for Mechanical engineering students to receive quick feedback on their submitted homework, and for teachers to use this feedback to evaluate class performance [1].

The challenge that is faced in recognition of truss sketches stems from two factors: 1- the complex nature of these structures, which makes it somewhat impossible to fully and comprehensively formalize in terms of a geometric description similar to that of a basic shape such as a circle, and 2- the vast difference in drawing style of each individual, which requires the developer to use a combination of different methods of recognition. In our work, we have based our recognizer on a set of geometric "rules" as per discussed in class, and then extended these rules through in-group brainstorming sessions.

The rest of this report is organized as follows: in section 2, we discuss the previously published work with sketch-based systems that recognize trusses. In section 3 and 4, we discuss our data structure, pre-processing steps, and debugging method. Section 5 discusses our truss recognition algorithm in detail, and in sections 6 and 7 we present and analyze the system performance results.

2 - Previous Work:

In the previous work in order to identify a truss out of a complete diagram, Fields et. al.[2] and Valentine et. al. [3] utilize the shared-edge property of trusses and the additional property of the domain that no other shapes have such shared edges. The forces, supports, and other shapes that are drawn attached to a truss all connect at a single point, instead of sharing an edge. Thus they conclude the existence of truss, if algorithm can find two polygons that share an edge. Additionally, any polygon that shares an edge with a truss must also be part of that truss. The paper uses the PaleoSketch algorithm [4] to preprocess each stroke and identify all the line segments. PaleoSketch will segment a 'polyline' stroke into each individual line segment, and additionally segment line segments whenever two of them meet, hence all line segments intersect at their endpoints. They use all the line segments in a diagram to build a graph with each segment as a node, and edges between segments that intersect. Once the connection graph has been built, they consider each edge as a potentially shared edge. Then this each considered shared edge is removed to check if there is another oath between those points, if not, it is not even a part of polygon, and so can't be a shared edge. If yes, then the two paths that were found with BFS each represent a polygon sharing that edge. However, to characterize the structure of a truss, we are more interested in the square and the triangle, and less interested in the right trapezoid. The BFS approach helps the algorithm to identify and separate the basic building blocks from which the truss is built. The other method uses a simple variant of the mark-sweep algorithm. They first mark all of the edges in the two paths found as 'truss'. Then, any polygons that have an edge inside the marked set can be added to the marked set. Any extraneous symbols, like forces or supports attached the the truss, will have edges adjacent to the marked set but never in the marked set. Once we have considered all other polygons (cycles in the graph), the marked set represents a truss.

3 - Data Structure and Pre-processing Steps:

The testing data provided for this project is in the form of JSON sketch objects with the following properties:

PROPERTY	DESCRIPTION
`id`	The sketch's automatically-generated UUID.
`time`	The sketch's creation time.
`domain`	The sketch's user-defined collection of domains.
`canvasWidth`	The width of the canvas that the sketch was drawn in.
`canvasHeight`	The height of the canvas that the sketch was drawn in.
`strokes`	The sketch's collection of strokes.
`shapes`	The sketch's collection of interpreted shapes.

Even though the data was generally much cleaner than what we had for the previous project, we observed an issue with the 'shapes' , where it was empty in many sketch objects.

The pre-processing steps that are performed in this project are:

1. **Resampling:** We are using iStraw algorithm for corner recognition, which requires a previously resampled sketch as input. Resampling by distance has been performed, to obtain equidistant points suitable for iStraw.
2. **Corner-finding:** Trusses are polyline shapes consisting of polygons , so accurate corner recognition can be a huge help in identifying correct structures. In our work we have used the iStraw [5] method to recognize corners.

4 - Debugging Setup:

In order to debug our algorithm step-by-step, we used the sketch preview setup available in [6], integrated it in our `index.html`, and used it to view step by step elimination of wrong strokes as we developed our algorithm. This method helped us hugely in identifying if our expectations were true in the sketch data provided, also we found out in many cases that we needed to consider more careful conditions or thresholds to avoid including wrong strokes or elimination of correct strokes in the output. The parts of code related to this section have been commented out in our final submission file since they are not directly related to our goal and are just meant for convenience in debugging.

5 - Truss Recognizer Algorithm:

For the purposes of this project, similar to [2,3], we consider a truss to be a collection of convex polygon. Each of which shares at least one side with another polygon in the truss. Based on the discussion in class, these polygons can include at most 4 edges, and at most one rectangle, the rest should be triangles. After pre-processing the sketch is now evaluated for having a Truss structure. We apply a multi-layered approach and process each stroke to gain additional information which helps in eliminating those strokes which are not likely to be a part of a Truss structure. We apply the following procedures to recognize the strokes which can form a Truss.

1. Single Line Check (with Arrow Elimination)
2. Poly Line Check
3. Proximity Matrix and Corner Degree calculation
4. Remove strokes with Degree One corners
5. Golden Rule 1: Shapes cannot have more than 4 sides
6. Golden Rule 2: Every Node must be connected to a Triangle
7. Golden Rule 3: Beams cannot pass through a node OR No nodes in the middle of beam.
8. Platinum Rule 1: Numerical Relation between edges and corners of a Truss.
9. Platinum Rule 2: Most Trusses have at most two corners with a degree of 2

Detailed Explanation of each step is given below.

1. Single Line Check (with Arrow Elimination)

For each stroke we first check whether the stroke is a simple line or not and then proceed to the Arrow eliminator procedure which removes most of the basic arrows drawn in usual manner. Note that the following Arrow Recognition procedure do not eliminate axes drawn in the sketch and is handled separately.

Recognizing an Arrow:

- *Single Stroke Arrows*

For recognizing single stroke arrows, we check the proximality of the last point of the stroke to all other points and if we find at least 2 other points in the vicinity of 7 times the Sampling Distance from the last point, we classify the stroke as a part of Arrow and remove the stroke from further consideration. This step removes all single stroke arrows as well as head part of the Two stroke arrows.

- *Two Stroke Arrows*

The head part of a Two Stroke gets removed in the precious check for Single Stroke Arrows. Now to remove the shaft of a two stroke arrow, we have to check both the previous and next stroke. This involves checking the whether any of those stroke is a single line and if yes, we check if any of the line segment endpoints is in the proximity of 3 times the sampling distance from the second point of the Arrow Head. We use the second point so that the algorithm works well for both 3 corners and 2 corners Arrowheads.

- *Three Stroke Arrows*

Removal of Three stroke arrow is not taken as a separate case, yet they are deleted effectively via the two Stroke arrow check. The reason being that all the strokes constituting it get classified into two 2 stroke arrows and are thus eliminated.

- *Other simple stroke elimination*

Apart from removing arrows, we also eliminate lines smaller than a particular length to remove the unintended ink drawn by the user. We employ a separate check to remove axes and the extended unconnected beams of trusses. This is done after the calculation of Proximity Matrix on the reduced Stroke set in Step 4.

2. Polyline Check

The rest of the strokes are then evaluated for polylines, eliminating the curved strokes in this step. This section is inspired from Polyline check employed in Paulson's PaleoSketch [4] which is based on 3 basic calculations:

Pseudo Code:

- Each sub-stroke must pass the line test.
- The average least squares error of each sub-stroke must be less than some threshold.
- The stroke has a high DCR value.

We saw that employing just the first and the third works fine in evaluating polyline strokes for the given data and thus we did not implement the second criteria. The DCR value was also relaxed slightly from the one suggested in paper given that all the sub-strokes pass the line test.

3. Proximity Matrix and Corner Degree Calculation

The set of stroke IDs remaining after the polyline check is termed as reduced Stroke set. Assuming all the corners in the reduced Stroke set as Graph vertices and the constituent polylines as edges, we calculate the Proximity Matrix for each corner. Proximity Matrix is an 'n x n' upper triangular sparse matrix and is valued 'True' if two corners are in a proximity of 3 times the Sampling Distance, where n denotes the number of corners in the reduced Stroke set.

Based on this proximity matrix we calculate the degree of each corner. The degree of a corner is the number of edges incident on that corner which is same as the number of proximal corners. Apart from the Proximity Matrix, each corner is attributed two additional degrees if it is in proximity of the central part of an edge. It is done so that **T** shaped corners are recognized which will otherwise get eliminated in the degree checks employed further. The steps involved in this process are:

Pseudo Code:

- For each corner with degree 1 in reduced Stroke set:
- Take every sub-stroke (which would be a line) from the reduced Stroke set:
- Remove the initial and final 20% of the sub-stroke as tails (just for calculation)

- Calculate the perpendicular distance between the corner and sub-stroke.
- If the distance is less than the threshold of 4 times sampling distance, consider it as node on the sub-stroke and increase the degree count of that corner by 2.

After this step, we further reduce our stroke set by removing axes and extended Truss strokes in next step.

4. Remove strokes with Degree One corners

In this step, we remove all the sub-strokes whose corners have a degree of 1. This is used to remove the extended unconnected strokes in the Truss and is based on the simple fact that the one of the endpoint of the extended segment would not have any other corner in its proximity. This also removes axes drawn in the sketch. The endpoints of the polyline part of the axes also do not have any proximal corners as the arrow heads and the labels are already eliminated before preparing the reduced stroke set in the first step.

5. Golden Rule 1: Shapes cannot have more than 4 sides

Now that we have all the polyline stroke segments which can form a probable Truss, we apply several rules to predict the presence of Truss in the given sketch. The first rule that we deploy checks for the constituent shapes formed by the reduced stroke set. We flag the sketch as a Non-Truss if it contains shapes having more than 4 sides i.e. we can at most include a square in our Truss definition. We deploy separate checks to identify whether the corners in the reduced stroke set form a square or higher degree shape. Detecting a square in this stage does not declassifies the sketch to be a Truss, but it just helps in our further calculations. We just perform the basic check depending on the number of corners in single stroke, like for a stroke having 6 or more corners we check whether the start and end point are in close proximity or not. For shapes drawn in multiple strokes, we rely on further degree checks for classifying them as non-trusses.

6. Golden Rule 2: Every Node must be connected to a Triangle

For any node that is not connected to a triangular shape, will have two adjacent corners of degree two. The simplest example involves a connected triangle and square. The two nodes of the square which are not connected to a triangular shape are of degree two and are adjacent. The similar pattern can be seen on all other structures. Thus we consider all two degree corners from the reduced stroke set and check whether any of its adjacent corner is also of degree two. If any such pattern appears, we classify the sketch as Non-Truss.

7. Golden Rule 3: Beams cannot pass through a node OR No nodes in the middle of beam

Although less probable, but if there is a node in the middle of a beam we should not consider it as a Truss. But while going through the training data we saw too many cases countering this assumption where the Truss is completed by passing a single beam through multiple nodes and thus we do not consider this rule into our prediction of interpretation of the sketch.

8. Platinum Rule 1: Numerical Relation between edges and corners of a Truss

We observed several trusses and derived a numerical relation among the edges and corners of a valid truss structure. The relation can be expressed as, "For a given number of corners C, a well-drawn Truss would have at least a minimum of 2C – 4 edges." We relaxed this relation by 1 edge as we found few corner cases as well. Thus, this rule predicts a given sketch to be a Truss if there exists at least 2C – 5 number of edges for C number of corners in the reduces stroke set.

Equation : **E >= 2C - 5**

9. Platinum Rule 2: Most Trusses have at most two corners with a degree of 2

We also observed that normal trusses usually have at most two corners having a degree of 2 unless it has a triangular structure. Thus we simply count the number of two degree corners and we already know if there is a triangular shape that exists in the reduced sketch set from calculations of Step 5. We take an intersection of both the conditions and use it to predict the Truss interpretation of the sketch.

The Final decision for the Truss interpretation of the sketch is made using the Golden & Platinum Rule 1 and 2. We classify the given sketch as Truss if both Golden Rule 1 & 2 and either of the Platinum Rules recognize the sketch as Truss. The stroke IDs returned is the reduced stroke set IDs remaining at the end. We only return stroke IDs if we have a Truss prediction.

6 - Results:

The results of our recognizer is given below. We have tested our system with 3 different input: (1) truss data only, (2) non-truss data only, and (3) All data. For each of these cases, the performance measures (accuracy and f-measure) have been given in two separate conditions: considering interpretation only, and considering final truss stroke IDs.

1. **Trusses_training.json**
   a. **Interpretation**

	Truss	non-truss
Predicted Truss	**975**	**365**
Predicted non-truss	**108**	**32**

Accuracy: 68.04 %

F-measure: 80.47 %

   b. **Stroke IDs**

	Match	No match
Have Predicted IDs	**819**	**365**
No Predicted IDs	**264**	**32**

Accuracy: 57.50 %

F-measure: 72.25 %

2. **Other_training.json**
   a. **Interpretation**

	Truss	non-truss
Predicted Truss	**0**	**203**
Predicted non-truss	**0**	**784**

Accuracy: 79.43 %

F-measure: 88.53 %

b. Stroke IDs

	Match	No match
Have Predicted IDs	**0**	**203**
No Predicted IDs	**0**	**784**

Accuracy: 79.43 %

F-measure: 88.53 %

3. Sketches_training.json

a. Interpretation

	Truss	non-truss
Predicted Truss	**1209**	**582**
Predicted non-truss	**114**	**562**

Accuracy: 71.78 %

F-measure: 77.64 %

b. Stroke IDs

	Match	No match
Have Predicted IDs	**1025**	**582**
No Predicted IDs	**298**	**562**

Accuracy: 64.32 %

F-measure: 74.65 %

7 - Discussion:

After analyzing our algorithm's step by step performance on v1 data, we saw a much better performance in recognition of truss objects and strokes, but after migrating our system to use data v2, we saw a reduction in f-measure. This is due to the fact that v2 data is much more similar to 'real life' data, in that it is more noisy and complex.

The fact that most of our training sketches contained simple vector analysis diagrams was helpful in that it limited our recognition to a specific domain. For example, we observed that most sketches contained arrows, and since we had done arrow recognition in the previous homework, we used that knowledge to recognize and remove arrows in our algorithm.

8 - Individual Contributions

In this project, we generally worked as a team through brainstorming, writing code and debugging. We divided the code into tasks of roughly same value, and assigned them to each member. After each teammate finished with their initial implementation, we went over the code in a meeting, and tried to identify what was right or wrong and needed changing. The report and submission files were also prepared collaboratively.

For the purpose of this section, we will list each team member's initial tasks below:

Aniket: Proximity check, Degree evaluation, Detecting corners that are connected to triangles.

Mayank: Elimination of arrows and single lines, Using edge to corner relation to eliminate squares, Polyline check.

Niloofar: Corner recognition, Sketch preview visualization, Performance measures.

9 - References

[1] Brooks, Randy H. "Score Improvement Distribution When Using Sketch Recognition Software (Mechanix) as a Tutor: Assessment of a High School Classroom Pilot."

[2] Field, Martin, et al. "Sketch recognition algorithms for comparing complex and unpredictable shapes." *IJCAI*. Vol. 11. 2011.

[3] Valentine, Stephanie, et al. "Mechanix: A Sketch-Based Tutoring System for Statics Courses." *IAAI*. 2012.

[4] Paulson, Brandon, and Tracy Hammond. "Paleosketch: accurate primitive sketch recognition and beautification." *Proceedings of the 13th international conference on Intelligent user interfaces*. ACM, 2008.

[5] Xiong, Yiyan, and Joseph J. LaViola Jr. "Revisiting ShortStraw: improving corner finding in sketch-based interfaces." *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*. ACM, 2009.

[6] http://people.tamu.edu/~ptaele/csce624/basicSketchDataViewer/index.html