Nio 2D-Mesh Reconstruction Hackathon

Adrian Vecina Tercero ʃ Amber Swarbrick ♡ Toby Benjamin Clark λ Zac Garby
School of Computer Science, University of Nottingham, United Kingdom

Introduction. This project takes an image of a hole-pattern mesh and converts it into a `.nas` mesh file. This report outlines and evaluates our methodology. We divide the problem into two parts:

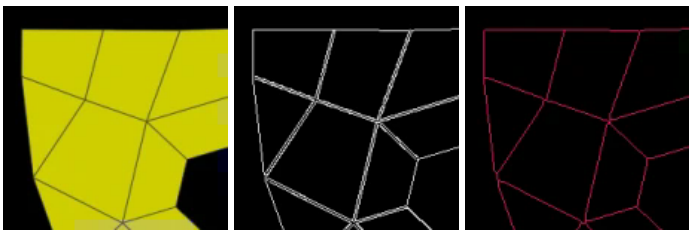
- Segmenting individual polygons from the source image.
- Finding precise node coordinates, in terms of the provided axis.

Topological Thinning. In order to refine exact node positions, we use information from the mesh *skeleton*. *Skeletonisation* is an image processing technique that reduces a binary image to a one-pixel wide *skeleton* whilst preserving its overall topology.

Firstly, the input image **(1)** is converted to a binary image. We convert it to greyscale, and apply canny edge detection to extract mesh edges; this generates an initial, but noisy, binary image **(2)**.

We then dilate and erode the image, filling small gaps and removing any noise. Sometimes, mesh nodes are circled in the input, which could be interpreted as an edge. To resolve this, such circles are detected and filled to provide a clean binary mask of the mesh.

Lastly, the binary image is skeletonised to reduce the mesh structure to a one-pixel-wide representation **(3)**. The generated skeleton is retained, and later used for refining exact node positions.



(1) *Raw Input Image (Crop)* **(2)** *Canny Edge & Greyscale* **(3)** *Skeletonised Image*

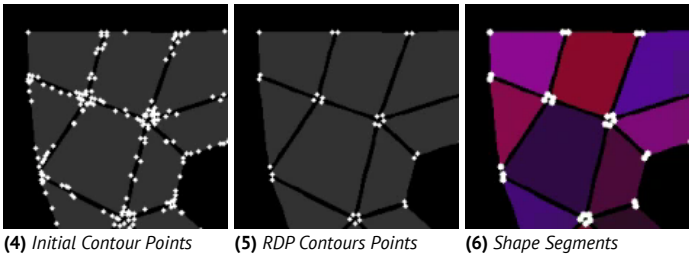
Segmenting Shapes. Meshes consist of collections of polygons (quadrilaterals and triangles). To extract these shapes from the raw image, we apply several image processing techniques.

To begin, we use our skeleton image **(3)**. Gaussian blurring and thresholding is then applied to create a mask that separates the foreground (shapes) from the background. This mask is used to detect the basic *contours* (outlines) of the shapes.

We then filter contours by a minimum size (calculated using image moments). The remaining contours describe the shapes as a set of points **(4)**, but due to the nature of pixel images – they contain far more vertices than are necessary to describe the shapes. Since BDF classifies shapes as either quadrilateral or triangular, we must describe shapes using only either 3 or 4 node positions.

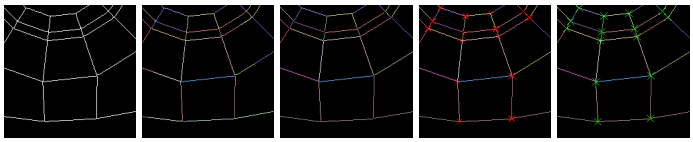
Therefore, we use the Ramer-Douglar-Peucker (RDP) algorithm to find the minimal number of vertices required to describe the shape **(5)** by iteratively removing points that lie within a specified distance of a straight line between neighbouring vertices.

Whilst this provides us with the structural connections, and approximate node positions of each shape **(6)**, the exact corner positions can be inaccurate due to the nature of RDP. In examples 122 and 115, RDP alone achieves IoU scores of 83.51%/84.85% respectively. To improve this score, we refine node positions using line intersects.



(4) *Initial Contour Points* **(5)** *RDP Contours Points* **(6)** *Shape Segments*

Node Refinement. We use the image skeleton to identify the precise locations of the edges and nodes in the mesh.



(7) *Skeleton* **(8)** *Flooded* **(9)** *Merged* **(10)** *Intersections* **(11)** *Points*

We start with the skeleton, and identify each of the straight edges which make it up. A first approximation is found by flood-filling from each point in turn. We flood segments of the skeleton whilst they a) have exactly two neighbouring "on" pixels, and b) do not deviate from the segment's direction by too much. See **(8)**: depending on where each flood begun, an edge may consist of several segments.

These need cleaning up. We use singular value decomposition over the point coordinates of each segment to vector line equations describing them. At this point we are working in real space rather than pixel space. Any two edges which touch and which are approximately colinear may be merged into one, yielding **(9)**. Two edges which meet at a junction with a third may not be merged.

To find the nodes, we find all intersections between pairs of edges. As shown in **(10)**, this locates multiple nodes at each junction, due to slight differences in the lines' vectors. We replace each cluster with its midpoint, and **(11)** illustrates the final set of identified nodes.

Finally, we must find the nearest, new, pixel-correct node, for each approximate node from the earlier segmentation **(6)** The first . To do this efficiently, we construct a *k*-d tree from our new node coordinates, and use this to perform nearest neighbour checks. This leaves us with the correct pixel-corners for each shape, which are then converted according to the provided axes.

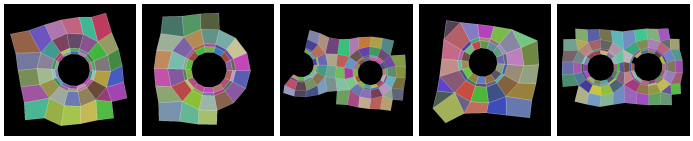
File Output & Validation. Generated meshes are validated for basic BDF criterion using the `pyNastran` library. Individual mesh elements (quads, triangles, and nodes) are translated into atomic cards, and written into a `.nas` output file.

Evaluation & Results Without node position refinement, the initial approach, using only RDP and thresholding, achieves IoU scores of 83.51% on example 122 and 84.85% on example 115. After adding node position refinement, the IoU scores increase to 99.28% for example 122 and 98.59% for example 115. These results provide a clear ablation demonstrating that node refinement significantly improves IoU accuracy on seen examples.

Our method is fast (sub-two-second), making it tractable for batch image processing/dataset generation. Thresholding and RDP parameters can be adjusted depending on the colours present in the input image and the pixel size of the polygonal regions.

Conclusion We presented how image processing techniques, and the Ramer-Douglar-Peucker algorithm can be used for generating hole-pattern `.nas` mesh files from simple images. We also demonstrated, through ablation, how line intersections from the source image can be used to refine node positions in the generated mesh file, leading to a significant increase in accuracy.

In their combination, our approaches form an fast, accurate method for extracting geometric meshes from simple images (see below) .



AI Declaration: *Large Language Models (GLM5.1 and Opus 4.6) were used to create visualisers, debuggers and some in-code prose. All ideation and methodology is our own brains.*