***Abstract*—abstract...**

Automated Conversion of Sketches into Source Game Engine Maps

Cameron Stevenson

I. Introduction

II. Proposed Method

A. Segmentation

We aim to find the segments divided by pencil lines.

The sketch is grayed, inverted, and blurred. An adaptive threshold highlights both sides of each pencil mark as black, with the background in white. This is inverted so pencil edges are white against a black background. A closing operation with 2 dilations and 1 erosion of a 5x5 kernel joins the two sides of each pencil mark, giving thick white pencil against a black background. These white edges may contain black pixels within them. This is acceptable as long as the edge has no black path between segments.

Now that black segments are divided by solid white boundaries, flood fill is used to identify internal segments and the external background. Each pixel in the image which is still black is used as a seed point for flood fill with intensity value 128. A segment is accepted as internal if it is large enough (more than a few pixels) and does not touch the image border. These segments are given intensity value 192 to indicate they are discovered. Rejected segments are given intensity value 255, so that they are treated as equivalent to pencil. The resulting image has background and pencil in white, and internal segments in gray.

B. Text Recognition

Short handwritten text is used to label segments based on what they will be in a map (floors, walls etc.). We aim to locate text precisely enough for labelling, and recognise the handwritten letters.

The common method of detecting text with EAST and recognising text with Tesseract was tried. It was found to work best with longer words and a background without other pencil. Our application requires short labels (up to 2 letters at most) and text near other pencil, so this approach was unsuitable.

Instead convolution methods were tried. Inverted samples of handwritten letters were turned into kernels of various scales and convolved over the inverted image, with the idea being that highlights in the output showed presence of the letter. This detected well, but structural features in the sketches were also detected as text. Convolution by its definition rewards white overlap but does not reward black overlap, and does not explicitly punish opposites. For example a solid block of pencil would be detected as any sampled letter. A more specific sort of convolution is needed.

Template matching gives us the specificity needed by rewarding pixel similarity and punishing dissimilarity. Squared difference template matching was used with the sampled letters at various scales over the image, to produce a mostly whitish image with black spots where letters are found. This is inverted and thresholded so that only confident highlights are retained. These highlights from all scales are merged into one image (as a weighted average where every scale has the same weight). It was found that each true highlight tends to appear on multiple scales, so thresholding this merged image retains these and discards some random highlights. Suzuki et al. [1] contour finding algorithm is used to find the location of remaining highlights. As a further step of narrowing down true highlights, on the sketch we use double letters as labels (e.g. ”XX”). So we look for pairs of highlights within a certain distance and angle range. The midpoint of each pair and the letter is recorded. This gives us the text label letters and locations.

C. Labelling and Gap Filling

From the segments and text labels we want to produce an image where each label has a certain colour, and each segment with that label is coloured that way.

Prior to segmentation, for each text label we fill a white rectangle over it so it isn't interpreted as a structural segment. This does not need to perfectly cover the text, but does need to reach the top and bottom of the text, so that flood filling and pencil gap filling work later.

After segmentation we colourise segments. For each text label, we use a user-defined lookup table to find its colour, then flood fill that colour from the label's location. This gives an image where labelled segments are coloured, background and pencil is white, and unlabelled segments are gray. From this point on we consider background and unlabelled segments to be ”wall”. We replace white and gray with black as they can be treated as the same from now on. To improve efficiency of the gap filling step we crop down the image to remove extraneous background, and scale the image down (the loss in precision is irrelevant when compared to the inaccuracy in segmentation).

We want to fill in the black gaps left by pencil marks, both between segments (intended structural marks) and within segments (typically produced by uncovered text). For this we make PENCIL_THICKNESS passes over the image, where PENCIL_THICKNESS is half the width of the widest pencil mark we need to cover. In each pass if a black pixel has a coloured neighbour, it becomes that colour. This slowly grows the coloured segments until they meet each other, removing inter-segment gaps. It also fills in internal holes. After this step any remaining black segments are considered to be wall, and filled in with the ”wall” colour. This gives a simplified image with colourised segments for the rest of the program to operate on.

D. Segment Bordering and Triangulation

We aim to find the border of each coloured segment, simplify it, and triangulate it to make brushes for the map file. This section operates independently of the prior sketch processing so that computer drawn images can be used.

Flood fill is used to find the pixels in each segment. A simple border walk is used to find the exact pixel by pixel border. Borders are simplified with a tolerance as a proportion of the area of the segment. Some preprocessing passes are made which operate locally, to reduce complexity. Firstly for any 3 collinear points the middle point is removed. Secondly when a corner of 1 pixel in width or height is encountered, the corner point is removed. After this tolerance based removals are applied globally, each time looking for the point which when removed changes the segment area the least. Only concavities are removed, which is suitable in this context as it ensures no gaps between the final map objects.

The colour of segments are used as labels to generate map objects. Map objects have borders and a type (wall, floor etc.). For each object its border is triangulated. This uses a fairly simple approach of cycling around the border looking for valid triangles (counter-clockwise, not intersecting the border's edges, and not containing another border point). A valid triangle is added to the result. The polygons left over from removing this triangle are themselves triangulated recursively to find the remaining triangles.

III. Results

A. Setup

OS: macOS 10.13.4
Processor: 2.4 GHz dual core I5-4258U
IDE: Visual Studio Code
Language: Python 3.7.2
Device: PC
Camera: 300dpi printer scanner (Deskjet 3070 B611)
OpenCV Version: 4.5.1

References

[1] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.