FINAL REPORT

THE PUZZLER

April 30, 2017

CprE 575 - Computational Perception

Alex Luehm & David Wehr

Contents

Introduction 2
- Use Case 2
- Applications 4

Previous Work 4

Approach 4
- Development Process 4
- Equipment 5
- Piece Matching 6

Implementation 10
- Design 10
- Preprocessing 11
- Detection 11
- Characterization 12
- Matching 14

Results 17

Future Work 18

References 19

INTRODUCTION

Jigsaw puzzles have been a favorite table-top pastime for generations - allowing friends and family to constructively work together and enjoy one another's company. In addition, puzzles have become a popular way for individuals to unwind after a stressful day at the office.

Individuals typically start out with simple, smaller puzzles - usually around 100 - 300 pieces - and gradually progress to larger puzzle ranges. As the size of puzzles increase, so does the complexity in shape and color of the pieces. It soon becomes more of a burden to bear than an activity to enjoy, causing frustration when progress seems to have come to a standstill.

The puzzle that once brought relaxation and enjoyment now becomes a sore in the room - taking up space and constantly demanding attention. The feeling of guilt begins to feel overpowering when walking into the room and seeing the pieces still needing to be sorted and slotted into their respective places. The puzzle needs to be completed, but the friction has become too great.

The Puzzler is designed to alleviate this sense of dread and frustration. Designed for both avid and novice puzzlers alike, The Puzzler will aid its users in finding potential candidate pieces given a selected region of a partially completed puzzle. Through edge detection and piece profiling of detected curves and angles, The Puzzler suggests potential next-pieces from the pool of yet-unused pieces.

Example Use Case

As initially imagined, the user would interact with The Puzzler through a smartphone application. The user will indicate to the app which region it should find pieces for by first taking a picture of the desired region followed by highlighting the specific edge, as seen in Figure 1. The app will then allow the user to take a picture of the remaining puzzle pieces. In the current implementation, the application runs on the desktop, but has the same desired user interface features.

After characterizing the curve of the desired region and processing the remaining pieces, the application determines which pieces are most likely to fit the specified region and indicates to the user with a visual overlay which pieces have the highest probability of matching, as shown in Figure 2.

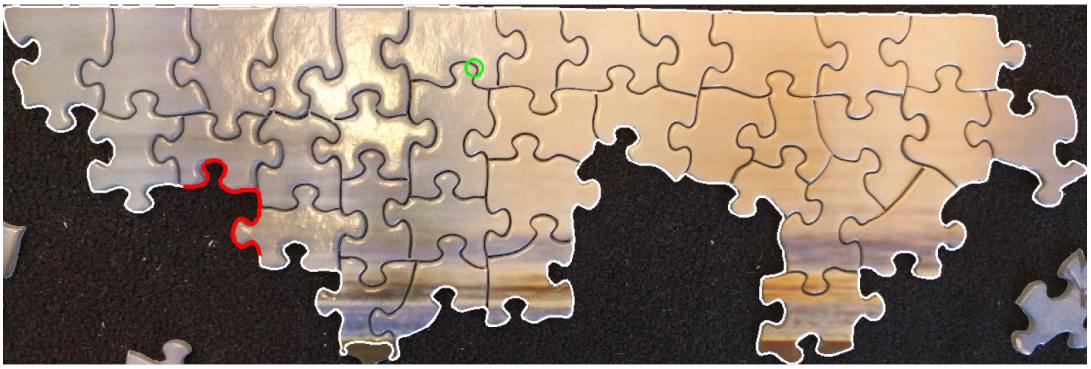

Figure 1: The desired location is highlighted by the user in the application

Figure 2: The application identifies one or more potential fitting pieces

Potential Applications

Outside of puzzle solving, there are other applications for algorithms related to jigsaw puzzle solving. In addition to solving entire puzzles, the identification of pieces and their correct global position can be extended to reconstructing artifacts from archaeological digs, recovery of shredded documents, and pairing proteins with amino acids [1].

PREVIOUS WORK

Work on similar problems has been conducted by various groups in the past, mostly in full puzzle reconstruction. In each case either a color-matching or a shape-matching technique was developed and used, although no case of the two methods being used together has been found.

Successful attempts by Goldberg, Wolfson, Hoff, and Olver have been made in reconstructing jigsaw puzzles based entirely upon puzzle piece shape parametrization. Various algorithms have been developed which focus on different aspects of the pieces' shapes and their characterization, all of which will be discussed later on in this paper. With these methods, complete reconstruction of up to 200 piece puzzles based upon scanned images have been possible [1, 2].

Attempts by Sholomon et al. [3] have been able to successfully reconstruct puzzles based solely on content and color information. Images would be taken in and enhanced, then broken down into a set of identically-sized squares. Pieces would be translated in a series of iterations, bringing pieces closer and closer to their correct global position. This method operated entirely without outside noise but was able to operate successfully in a reasonable amount of time.

In the end, characterization based upon shape was chosen as our primary method of piece identification, as reasonable results were able to be obtained with scanned and processed images. The method based upon color was reliant on similarly-sized pieces and isolation from outside noise. If shape identification alone can not yield satisfying results, additional heuristics may be developed based upon basic color and pattern matching.

APPROACH

Development Process

Our development progressed in four major stages:

1. Capturing scans and pre-processing
2. Identification and characterization of pieces and curves
3. Curve matching
4. Tuning algorithm and parameters for best results on a variety of images, including camera-captured images

The first step was to capture scans and perform preprocessing to ensure that we can obtain representative images of our puzzle pieces. This step embodied the feasibility portion of our project. We were uncertain if the uniqueness of pieces could be determined from typical flatbed scanners and digital cameras. At first, we used standard photo editing software to perform preprocessing to verify that the images were of high enough quality to be compared digitally. Later we developed self-contained routines within OpenCV to perform any desired preprocessing.

To detect pieces, we needed the ability to segment pieces and characterize them. The characterization of pieces allows us to describe them by their physical shapes and contours. During this phase, we worked to produce an algorithm that, when given a similar input image, will consistently produce a corresponding output parameterization and have the ability to compare different parameterizations. We focused on the identification of and location of indents and outdents, as well as size-based heuristics for determining the difference between pieces and partially completed puzzles.

Curve matching takes the found curves and compares them to each other. We use invariant points such as the centers of tabs for initial placement, then calculate how well the pieces fit by comparing the raw point sets using Iterative Closest Point (ICP).

All of our development progressed on a desktop computer using C++; we additionally created a user-friendly interface for selecting the desired edge to match to and highlighting of the potential piece matches.

Equipment

Android Device

The Puzzler was initially planned to be implemented as a hand-held device in the form of an Android app. Unfortunately time constraints limited the final product to a desktop application with an external webcamera. However, the application could easily be ported to an Android application in the future. By using a mobile device,the user would be able to

easily scan regions and pieces of puzzles and allow for easy interaction with the application through the existing touch screen. In addition, the camera contained within most Android devices would provide ample resolution for adequate edge detection and piece profiling.

Puzzles

We started our development using smaller puzzles, consisting of 24 pieces. These were two "Finding Nemo" puzzles that have large, standard shaped (rectangular) pieces, as shown in Figure 3. These puzzles allowed us to refine our algorithm in an ideal environment, since they have a consistent shape and are large enough to provide high resolution captures with low noise. Later we tried our application on smaller, more "adult" puzzles. These smaller puzzles pushed the limits of our approach and exposed the major weakness, most notably, the sensitivity to noise.

Figure 3: Close-up of puzzle we began testing with

Piece Matching and Parametrization

As mentioned in Previous Work, there have been several different approaches to the problem of finding correspondences between puzzle pieces based upon shape for the purpose of automated assembly. Here we give an overview of the different approaches and indicate their strengths and weaknesses.

Goldberg et al.

Goldberg et al. [1] described a robust method of solving entire puzzles based entirely upon piece shape. While we only intend on matching pieces to a region selected by the user, we found the methods used to obtain accurate representations of the pieces and to match pieces based upon shape closely paralleled various challenges that we too would have to overcome.

Goldberg first describes how they were able to obtain crisp and and accurate representations of puzzle pieces through the use of a copier and flatbed scanner. After copying the pieces against a high-contrast background, the images were then scanned and stored on disk. These scans were then converted into binary images via an adaptive threshold and divided into individual pieces through the use of connected components. The borders of these pieces were then smoothed via Gaussian blurring and digitized into approximately 600 vertices.

Indents and outdents were then identified through the use of inflection points. Care was taken to filter out false inflection points by looking for points where curvature differed by at least 10 degrees. After locating inflection points, lines tangent to these points were drawn which would identify indents if these lines crossed outside of the piece. Straight edges would then be identified, leaving the remaining regions to be outdents.

Fiducial points, or the centers of ellipses fitted to the contours of indents and outdents, were then created, which aid in comparison of indents/outdents. The puzzle pieces that composed the border were then found and fitted together to create the finished border. We will not discuss border piece identification and assembly in great detail, as it does not directly pertain to our project.

After the successful creation of the border, the interior pieces were identified and slotted into place. This was done in a specific manner working only with "pockets", consisting of two or more pieces making "L" or "U" shapes. This is important, as we eventually wish to allow the user the option of specifying straight-edges for completion, instead of only pockets.

Pockets were evaluated one at a time, finding potential candidates for each pocket. Instead of fitting the piece with the highest chance of fitting, pockets that have the greatest ratio of "fit" between the first choice and second choice were fitted. This produced a greater rate of success, as this chose the option with the smallest ambiguity.

Pieces were tested for "fit" in the following manner: a candidate was placed such that the distances between the fiducial points located on the candidate's indents/outdents and the points located on the pocket's indents/outdents were minimized. A score was then calculated by walking along the edge of the candidate piece, finding the shortest distance between every point on the candidate piece and an edge on the pocket's boundary. The score was then calculated to be the average of the cubed distances. Pieces with the highest chance of fitting

had the smallest score.

Some general assumptions were made with respect to general piece structure. It was assumed that pieces would follow an overall rectangular shape, with four distinct edges - each edge containing either an "indent" or an "outdent".

Hoff, Olver

In "Automatic Solution of Jigsaw Puzzles", Hoff and Olver [4] describe a method of assembling a jigsaw puzzle by comparing pieces via a so-called "extended Euclidean signature". Their approach to characterizing pieces and comparing them is relevant to our project, so we studied their paper and have described the method below.

After obtaining an outline of a puzzle piece, they break the piece down into "bivertex arcs". A bivertex arc is a segmentation of the puzzle curve by "generalized vertices", defined as points in the outline curve where

$$\kappa_s(s) = \frac{d\kappa(s)}{ds} = 0 \tag{1}$$

where (s) is the curvature of the outline, defined in the standard way for plane curves. A generalized vertex can be viewed simply as an inflection point, which may extend for more than one vertex in the case of straight lines or circular arcs.

Every curve has a unique bivertex decomposition, and pieces can be compared by computing a similarity score between every bivertex arc pair. The parameterization of an arc, also known as its signature, is defined by

$$\{(\kappa(s), \kappa_s(s)) | 0 \leq s \leq L\} \tag{2}$$

which is simply a set of tuples of (curvature, derivative of curvature) along the entire length L. To compare arcs, Hoff and Olver calculate the electrostatic repulsion between two sets of arc parameterizations. This is done by treating every point in the parameter space of the arc as a positively charged particle, and having each pair of particles contribute a repulsion force proportional to

$$\frac{1}{d^\lambda} \tag{3}$$

where d is the distance between the points, and λ is some constant. The larger the repulsion force, the closer the similarity of the arcs.

Hoff and Olver had success in assembling puzzles with irregularly-shaped pieces (pieces that don't fit the standard square-with-tabs style), which is a strength of their approach

over the other methods we reviewed, since the other methods expect either separable edges and/or clearly defined tabs.

Unfortunately, there are two main weaknesses in this approach, which are sensitivity to noise and high computation costs. $\kappa_s(s)$ is the third derivative of the curve. Therefore, it is highly sensitive to small variations. Because of the sensitivity to noise, they had to run a spline fitting and vertex respacing algorithm for 1500 iterations on each piece before running the main curve separation. This was even with pieces captured via a flatbed scanner, which results in high-quality images.

Additionally, the arc comparison is an $O(N^2)$ operation, as it requires computing the repulsion between every pair of points. Hoff and Olver found that they needed 80 $\frac{pixels}{cm}$ to get an accurate enough representation, so matching a single arc (roughly 3 cm) would require 60,000 repulsion calculations. This would need to be done for every piece in the frame, and for every bivertex arc in the piece. This number of calculations does not exclude the algorithm, but it raises some concerns about the feasibility of computing it in a reasonable time, especially considering the additional computations to perform smoothing.

Wolfson

Much of the work produced by Wolfson was used by Goldberg et al. in the development of their algorithms - namely the method for identifying piece sides through inflection points, reconstructing the border of the puzzle, and the use of both a local and global matching algorithm.

Local matching was done with the curve-matching algorithm produced by Schwartz and Sharir, which was fed into the global curve matching algorithm. This global algorithm would work to keep track of accumulated error and would periodically "refresh" this error [2].

The Schwartz and Sharir curve-matching method [5] also plays a large role in the Goldberg approach. They outline a process where the distances between two curves is accumulated and then used to create a metric of how well the two curves match. The curves are represented by a series of points which can be iterated through, finding the minimum distance between each point on one curve and the edge of another curve.

Summary

Of the different algorithms that we reviewed, the algorithm described by Goldberg et al. is the most useful to our application. It has lower sensitivity to noise, so only requires simple Gaussian smoothing as a preprocessing step, and compares sides using least-squares regressions,

which is $O(N)$ complexity. Because of the robustness to noise and lower complexity costs, it lends itself well to a mobile application with lower quality images and limited processing power. Therefore, we borrowed several ideas from their approach in our own application.

IMPLEMENTATION

The final implementation of the project took place in C++ so that the OpenCV library could be easily utilized, while allowing for a structured and object-oriented approach throughout the detection, characterization, and matching stages of our program. A brief summary of the data structures that were developed used is provided to help the reader understand the steps described in later sections.

Design

We decided to use a combination of various C++ classes to allow for efficient handling of information and distinct containers for storing and processing various bits of information. The Piece class was developed to store information about each piece detected by the program. A Piece stores an ordered list of 2-D points that describe the outer shape of each piece. In addition, lists of indicies that reference the points of a piece are kept to identify inflection points, points that lie along the convex hull, points of describing the local maxima in convex hull defects, and a unique identifier are stored for each piece. An ordered list of Curve type objects is also maintained for each piece.

A Curve class is used to store information for each curve found on a given piece, where curve here refers to either an indent or an outdent. A Curve is made up of a starting index, a stopping index, an origin, and a type. The starting and stopping index refer to the indicies stored within a piece and describe where each indent our outdent begins and ends. The origin is a 2-D point that is created during Curve creation and is stored solely within the Curve itself. It describes the approximate center, or fiducial point, of each indent or outdent. The type field is used to determine whether a Curve describes an indent or outdent, and is used later to efficiently determine a subset of potential Edge candidates.

The Edge class allows a series of Curves to be combined to describe a physical edge along a puzzle piece and is ultimately what allows regions on two separate pieces to be compared against one another. An Edge contains an ordered list of points (copied from the original pieces instead of referenced like in Curves), an origin, a handle, and an ordered list of Curve types. We create copies of the original points so that edges can be translated and rotated

without modifying the original pieces. The origin describes the center of the first Curve contained within the Edge and is used in edge translation, while the handle of an Edge describes the center of the last Curve of and Edge, and is used during Edge rotation. The ordered list of Curve types is used for efficient Piece sorting and Edge creation.

Pre-processing

We have so far restricted our implementation to only work with images of pieces placed against a solid background that provide enough contrast to allow thresholding the image to separate the pieces. Therefore, our preprocessing involves the following steps:

1. Convert image to grayscale.
2. Threshold image such that the puzzle pieces are clearly separated from the background.
3. Perform morphological operations to smooth out the boundaries and eliminate extraneous noise.
4. Extract the contours from the image, creating a set of 2D points representing the boundary.
5. Discard components which have an area too small to be considered a puzzle piece.
6. Identify contours in the image which are large enough to be multiple pieces, and consider those as partially assembled puzzles.
7. Store the boundary of each piece separately and keep track of its original location in the source image.

Performing these steps results in clean outlines of the puzzle pieces that allow us to differentiate between the different pieces. After doing these steps, we may find that step (5) does not provide enough smoothing, or removes important artifacts from the pieces. If so, we will need to use another smoothing method, such as the spline-respace method used by Hoff and Olver or Gaussian blurring used by Goldberg et al (see Future Work).

Detection

To detect the individual puzzle pieces contained within a source image, OpenCV's "Structural Analysis and Shape Descriptors" module was used. Images are first converted into binary images by applying a threshold value of 100/255. Dilation and erosion with 5x5 structuring

elements is then applied to fill in any regions along edges falsely omitted by the threshold. A median blur filter with an aperture of 25 is then used to smooth over any residual jagged edges.

Figure 4: Progression from thresholding to morphological operators to median filter

OpenCV's `findContours` method is then used to detect and extract the individual puzzle pieces. From this, we obtain a 2-D vector of points, containing ordered lists of points for each individual point. We use a simple approximation, where line segments are compressed into as few points as possible.. Although `findContours` has the ability to find internal contours as well, we are only interested in the external edge description. Pieces are then filtered by area, removing any contours that are too small to be true pieces. This removes many false artifacts detected in the background material. The size of any remaining pieces are then averaged and used to discard any additional outliers. The remaining elements are simplified using the `approxPolyDP`, which uses the Douglas-Peucker method, resulting in smoother edges and fewer points while still preserving information pertaining to a piece's curve structure. The resulting information is stored within Piece objects, one per puzzle piece.

Partially assembled pieces are treated the same as individual pieces, with the exception that they are labeled as "compound" and receive additional processing, which is described later, before being characterized.

Characterization

After Piece objects have been created from puzzle pieces contained within an image, the are characterized via the following steps:

1. Inflection point identification.
2. Discovery of the convex hull.
3. Identification of indents.
4. Identification of outdents

Inflection points

Inflection points are locations on the piece's boundary where curvature has switched from convex to concave, or vice-versa, and are vital to the location of indents and outdents. For every indent and outdent, a pair of inflection points exist that can be used to determine whether the region between them is convex or concave. These points, with additional information, can be used to characterize the piece in question.

Due to the noise observed in the provided images and the inability to use floating point coordinates with all OpenCV functions, many minor inflection points are detected when analyzing a piece, even along straight edges. In order to combat this, a threshold is used to limit the detection of inflection points such that only regions where the curvature has changed by a minimum of 40 degrees since the last inflection point are considered for future inflection point detection. This allows insignificant inflection points to be discarded and improve general accuracy in detection.

Convex hull

After locating inflection points, the convex hull of each piece is computed using OpenCV's `convexHull` method. This returns a set of ordered 2-D points. Using these, we further determined local maxima of each defect of the hull. This provided us with the location of points that maximize the distance between the convex hull and the piece boundary. This was accomplished via the `convexityDefects` method, with the results being stored within each piece for later use. Again, a simple thresholding was implemented to discard irrelevant local maxima based on distance from the hull.

Finding the points that both minimize and maximize distance from the convex hull allow for more robust indent and outdent detection. In addition to a pair of inflection points being found on each indent and outdent, the two points must have either a local maxima (indents) or minima (outdents) from the conex hull. These points represent the "peaks" of each curve. By finding these and considering the inflection points on either side, we can offer a more robust and less computationally intensive method of indent and outdent detection.

Indent detection

Indents are located before outdents as they are slightly easier to determine and help reduce the probability of discovering false outdents. To find indents, the previously found points that maximize the distance between the convex hull and the contour of the piece are iterated through one at a time. For each point, the nearest inflection points to either side are located.

From these two inflection points, tangent lines to the piece boundary are computed. If these two tangent lines cross outside of the piece's boundary, the region is considered a potential indent.

To filter out any false positives, an ellipse of best fit and a minimum enclosing circle is fit to the region between the two inflection points. The areas of the two shapes are compared, and if found to be within 30% and no other pre-existing Curve shares the same starting/stopping points, the region is confirmed to be an inflection point. The index values of the points between the two inflection points (inclusive) are stored in a new Curve object, along with the type (indent) and the center of the ellipse.

Outdent detection

Outdents are located by iterating through pairs of inflection points and discovering whether any points along the convex hull lie between the two. If so, the same procedure as above, in that tangent lines are calculated from the two inflection points and their crossing point located. If the crossing point lies within the piece boundaries, then the region is labeled as a potential outdent. As with indents, an ellipse of best-fit is found, tested, and then a new Curve object created if no previous indents or outdents share the same inflection points.

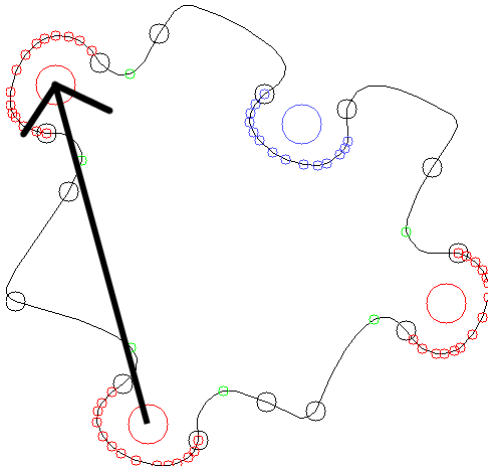

Figure 5: A piece extracted from a picture and marked with key characteristics

Matching

Once the pieces have been characterized, they are compared against each other. There are three main steps in this process:

1. Target edge creation.

2. Find potential pieces.
3. Align each potential piece to its best possible fit.
4. Evaluating the fit

Target Edge Creation

Before pieces can be matched, a target curve must be established against which to perform our comparison. After all pieces have been detected and characterized, The user is prompted with an image of the partially-completed puzzle piece that was found during the detection phase. The user is then able to select two points along the edge of the piece to specify what curve we should be matching against. As of now, the user is limited to "pockets", or to regions along the edge where two pieces form a concave corner.

After the user selects two points, we an ordered list of 2-D points located within the selected region and create a "fake" piece. We invert the region, so that the concave area specified by the user is transformed into a convex region. This is done by inserting a point into the ordered list such that an approximate 90°is formed between the start and end points. This allows the same indent and outdent detection to be performed as used with regular pieces. After identifying the fake Piece's indents and outdents, its list of Curves is populated and an Edge is created. From this Edge object, a sequence of indents and outdents can be discovered, allowing us to quickly filter out Pieces that do not match the required pattern.

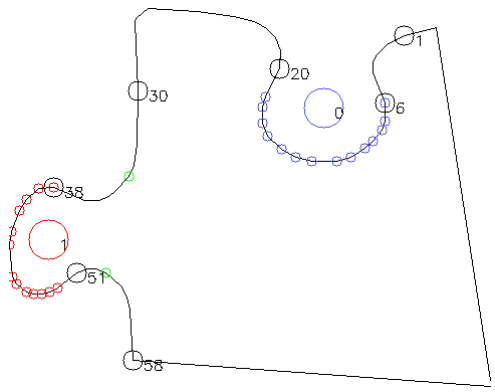

Figure 6: A "fake" piece created from user selection

Finding Potential Pieces

After creating the Edge that will be compared against, corresponding Edges must be created for each Piece. Because an Edge only describes a small subset of the Curves found on each

Piece, many Edges could be generated from each Piece. For smaller puzzles, excessive Edges being generated might not pose an issue, but for a larger number of Pieces being proceed, 100s of Edges might be generated and require processing.

In order to cut down on the number of Edges to process, we can filter out many based entirely on the sequence of indents/outdents that the target Piece possesses. After generating the Edge based on the user's selection, a target sequence of indent/outdents is extracted and used to selectively generate potential Edges to match with.

For each Piece, we iterate through its list of Curves, looking for the desired pattern. When the required pattern is located, an Edge object is created from the respective Curves and set aside for further processing. This prevents many excess Edges from being created. With the small puzzle consisting of 24 pieces that was used during testing, we were able to limit the list of potential Edges from 104 (four per piece when matching against two Curves) to just over 20 Edges.

Aligning Pieces

To test a potential piece for alignment, we first find a rough alignment, then refine the fit with the Iterative Closest Point algorithm.

Because we are comparing pockets, each piece has exactly two tabs that must interlock. The vector between the centers of the two tabs provides a unique orientation and origin for the piece (Fig. 5). To find the center of the curve, we fit an ellipse to the points associated with each tab as in [1]. Using this origin and orientation, the piece to compare against is translated and rotated so the origins and vectors align.

The Iterative Closest Point (ICP) algorithm is a point set registration algorithm that finds a transformation matrix T that, when applied to the set of points, locally minimizes the distance between the two sets of points. Because it finds a local minimum, a close initial transformation, as described above, is necessary. We used the implementation provided by the OpenCV Contrib modules. It is a variant on the standard ICP algorithm which uses the point-to-plane distance, rather than the point-to-point distance, allowing different point samplings to still match closely along the edges. Running this refinement step on the fitting allows us to find a match that more closely represents how well the pieces would physically fit, despite differences in the centers of the fitted ellipses.

Evaluating the Fit

As ICP attempts to minimize the sum of squared errors, we can use the resulting error as a metric for how well the pieces fit. The piece with the minimum error is most likely, but because of noise and imperfections in the input images, it is possible that the true piece has a slightly higher error. We tried two approaches for determining a likely candidate. Our first approach used the standard deviation and considered a fit likely if the error was less than 0.5 standard deviations above the minimum error. This approach had poor results when there were two likely candidates, and was too generous when there was a wide variance in errors. Our second approach simply considered a piece likely if it had an error less than 0.01 above the minimum. This was an empirically determined amount, but resulted in the most practical results.

RESULTS

To evaluate our work, we analyzed the ability of our program to uniquely identify and parameterize pieces with the addition of outside noise. Our goal was to produce a working product that was capable of providing the user a selection of possible pieces for a specified region within a useful amount of time. We had initially set out with the intention of creating a functioning Android application.

In many aspects of our application, we were successful. We were able to successfully analyze and identify individual pieces contained within a provided image. With proper consideration to the background, clean and unique parameterizations of each piece were able to be created and stored when using scanned images. In addition, pieces were able to be differentiated from partially completed regions of the puzzle based on size. Users are able to select a target region, from which useful information is able to be extracted. Finally, given two sets of parameters, our program is able to produce a meaningful measure of error.

In our proposal, we stated that we hoped to reduce the number of pieces that the user must manually search through by 90%. In more exact terms, if provided a set of images of puzzle pieces U and a single piece to match with, we had hoped to be able to identify a subset of pieces S such that the matching piece is in S, and $|S| \leq 0.1 \times |U|$. Towards this end, we achieved moderate success given particular conditions. With our development puzzle, our application determined the correct piece 100% of the time.

FUTURE WORK

The largest challenge with our current implementation is the ability to segment pieces from the background. This is the first step in the pipeline, and if distortions are introduced during this stage, the rest of the processing fails. Our current approach for segmentation and pre-processing is simple, and has much room for improvement.

One potential approach to improve piece detection is via the Generalized Hough Transform. This would allow us to use less aggressive pre-processing to retain detail, then discard objects that do not fit one of the common shapes. Another approach to improve the quality of shapes would be to introduce a spline or curve fitting algorithm into our process. Fitting a curve to the outlines of the piece would allow us to eliminate noise while still preserving the main shapes. This has an advantage over morphological operations and blurring by being more selective in the removal of noise.

Another difficulty we ran into was a robust method of identifying indents and outdents. While we followed the methods outlined by goldberg as closely as possible, we were still required to implement many of the methods according to our own interpretations and, as a result, led to various rates of success. As of now, we are filtering false indents/outdents based on area, where a more desirable method would be using a metric of best-fit for when fitting our ellipses. By reinforcing this method, we could produce higher accuracy when characterizing pieces.

In addition, our approach assumes that indents/outdents will have a high correlation to the shape of an ellipse or circle. For many "adult" puzzles, indents and outdents can carry a wide variety of shapes and sizes. To provide more robust detection we recommend the use of a calibration routine, various pre-defined curve profiles, or learning curve profiles.

Pieces detected in images are assumed to be the same size. Currently we have no mechanism in place to rescale images during edge comparison. This implies that the pieces being sorted through and the edge being matched against must either appear within the same image, or that images input into the program must be taken from roughly similar distances and angles. However, through the nature of the mechanisms currently in use, future implementation of scaling on matching should require relatively little refactoring and redesigning. By adding this functionality, less work will be required from the user when capturing images.

Our application currently runs on a desktop PC, which is not convenient for potential users, as it requires the user to take a picture using an external camera and upload it to the computer. To improve the usability, the application could be ported over to run as a smartphone application. OpenCV can be compiled to run on iOS and Android, and our work

has been implemented using C++, so it could be easily packaged as a library and utilized in a platform-specific application.

References

[1] D. Goldberg, C. Malon, and M. Bern, "A global approach to automatic solution of jigsaw puzzles," *Computational Geometry*, vol. 28, no. 2-3, pp. 165–174, 2004.

[2] H. Wolfson and A. Kalvin, "Solving jigsaw puzzles by computer," *Annals of Operations Research*, vol. 12, no. 12, pp. 51–64, 1988.

[3] D. Sholomon, O. David, and N. Netanyahu, "A genetic algorithm-based solver for very large jigsaw puzzles," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1767–1774.

[4] D. J. Hoff and P. J. Olver, "Automatic solution of jigsaw puzzles," *J. Math. Imaging Vis.*, vol. 49, no. 1, pp. 234–250, May 2014. [Online]. Available: http://dx.doi.org/10.1007/s10851-013-0454-3

[5] J. Schwartz and M. Sharir, "Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 29–44, 1987.