Resolving Matched Stereoscopic Surfaces into a Three-Dimensional Model

Stephen Lee, Guanlun Mu, Sam Bedell
Computer Science
St. Olaf College
1520 St. Olaf Avenue
Northfield, MN 55057
leesn@stolaf.edu, mu@stolaf.edu, bedell@stolaf.edu

Abstract

We present a twofold method for creating a cohesive 3D model from processed stereographic images. Given outlined surfaces matched between stereo images, we create a simplified, colored 3D surface with texture and depth occlusions solved. Once these surfaces are created for a particular model, imperfections between them are minimized by stitching their edges together and filling these gaps with new surfaces. We demonstrate the effectiveness of these techniques on manufactured and real data.

Introduction

While development in 3D modelling has become more and more advanced, the approach of building a cohesive 3D model from stereo images is still an active area of research. In our research, the general idea of 3D modelling was to create the model by merging 2D polygons which were referred to as tiles. The detailed procedure of implementing this will be introduced in section “Tile Creation”. However, one particular problem presented when merging stereoscopic images is occlusion and inconsistency between the images. When creating the model, these inconsistencies result in numerous problems that cause incorrect texture mapping, inaccurate depth information, and general model incoherence. Given distinct surface outlines and depth information obtained by matching identical points on the objects between the images, we propose a method of solving this problem.

Our solution is twofold. First, we create polygons from the outlined and mapped surfaces. When these polygons are created, they are simplified to reduce the number of corners and regions that need to be included from each of the different images. Second, we attempt to fix remaining problems with inconsistency in the polygons by filling the gaps between them with additional triangulated surfaces. Vertices of nearby polygons are stitched together with new segments, which form the foundation of new surfaces that finalize the model. The ideal result is a visually consistent model.

Background and Prior Research

Much of the previous work in the field of stereoscopic imaging solves this problem in one step. Texture and depth inpainting are common ways to solve the issue of occlusion, using neighboring areas on the images or other pairs of images, as in [1] and [2]. While effective, these methods are more suited to creating a depth and texture map of a single perspective (for example, when creating 3D video) and less suited to creating entire navigable scenes.

Procedure and Results

Tile Creation

Each object appears in multiple stereoscopic image pairs. We call a contour the region of a stereoscopic image that represents a particular object. Each contour belongs to an image taken by exactly one camera and can be traced on the image plane of that camera. From these contours, we seek to generate a single polygon in three dimensions, called a tile, to which color data can be mapped from the appropriate images. This tile should both have relatively few edges and accurately represent the contour data.

The number of edges of the tile is decreased using the Douglas-Peucker algorithm for line simplification. Based on the work of Shi and Cheung, this algorithm seems the best for maintaining the faithfulness of the simplified line.

Often, the contours are not simply connected regions. They may have holes in the middle of them which do not represent the object. These holes may in turn have regions inside of them which do represent the object. In addition, a contour may be composed of multiple, disconnected components. While some of this variation stems from errors in the data collection, much of it simply stems from the two-dimensional appearance of three-dimensional space: a wreath, for example, will have a hole in it.

Thus the first step of this algorithm is simply finding which boundaries of each contour are interior edges (forming holes) or exterior edges. This algorithm presumes an even/odd approach: it determines the nesting order of the boundaries and judges every other boundary going in as exterior.

Once these boundaries have been determined, the algorithm transforms all the contours from the coordinates of their individual image planes to the coordinates of one of the image planes using a homography, generated using projective geometry from corresponding feature points found between the images. These projected contours allow the algorithm, from this point on, to consider everything relative to a single coordinate system.

The first step of the algorithm is to find the regions of this one image plane that are within more than one projected contour and combine them into a single shared region. In practice, this is a union of intersections, using polygon operations. With three contours A, B, and C, this region is $(A \cap B) \cup (A \cap C) \cup (B \cap C)$; for two contours it is simply $A \cap B$. Any part of this shared region is automatically included in the final tile. All boundaries of this shared region are simplified using the Douglas-Peucker algorithm.

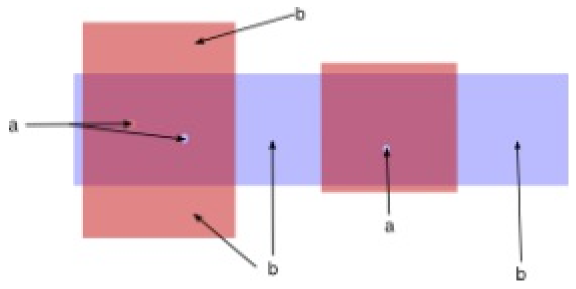
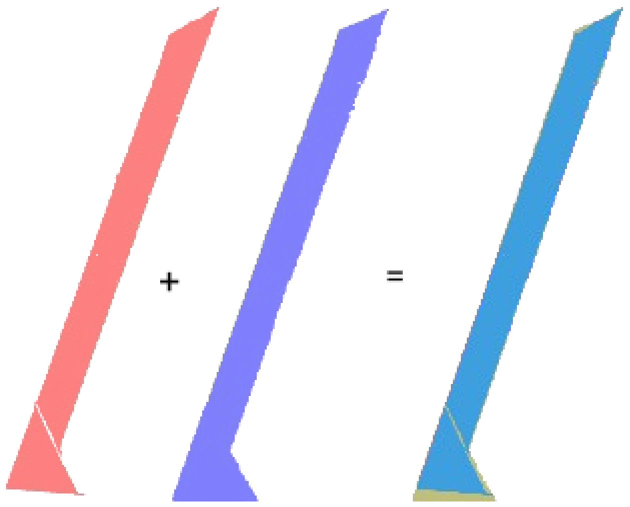


Figure 1: This drawing depicts a the contours, one light blue and the other dark red, of two different cameras. The darkest regions where the contours intersect are automatically included in the final polygon. That will also include the regions labelled (a) because they are completely surrounded by intersecting regions, and the regions labeled (b) since they are large enough.

After this shared region is found, the regions of each contour outside of this shared region are found using polygon difference. These are also simplified using the Douglas-Peucker algorithm. Every one of the unique regions of each contour is classified into one of three distinct types, depending on the extent of its common boundary with the shared region (see Figure 1). If the unique region is completely bounded by the shared region, it is automatically included in the final polygon. If not, it is included based on its size. Currently, we use the same threshold both for unique regions that share no boundary with the shared region and for unique regions that share part of their boundary with the unique region. This may be improved in the future.

Once all the regions of the image that will compose the final polygon have been

Figure 2: In this image, the red and the blue contours combine to form the rightmost section. In that, the blue represents the intersection and the unique regions of the first image and tan represents the unique portions of the second image.



determined, the algorithm decides from which image to color. For this purpose, we assume that the cameras are ordered by preference. The obvious criterion, if it can be determined, is the proximity of the camera to the tile being generated. Since each contour is associated with one camera, the ordering of the cameras also provides an ordering of the contours. Thus, each contour is taken in this order and its portion of the shared region, together, with its unique regions judged worthy of inclusion, are added to the final tile with the instruction to color them based on its image. Note that second or third contours only color their portion of the shared region not colored by any higher-ranked contour.

Results

As shown in the image to the left, the results are encouraging. In this image, the blue represents the intersection and the unique regions of the first image and tan represents the unique portions of the second image. Portions of the image not included in the final polygon are red. Clearly, most appropriate regions were included and attributed to the correct image. However, a few problems remain. For one thing, these polygons are unsimplified. When line simplification was run, the bottom tan section was not included. Also there are a few isolated regions which should not have been included but were (too small to see in this image).

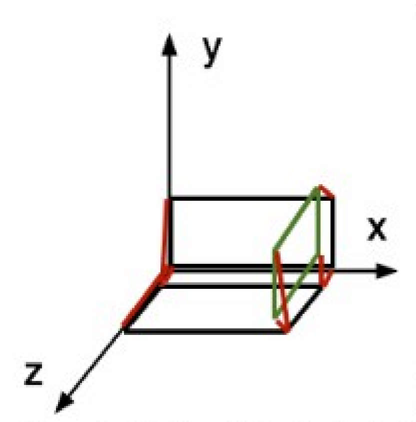


Figure 3: Vertices joined to their nearest different-tile neighbors by segments. The three long segments are undesirable. One tile is colored for visual clarity.

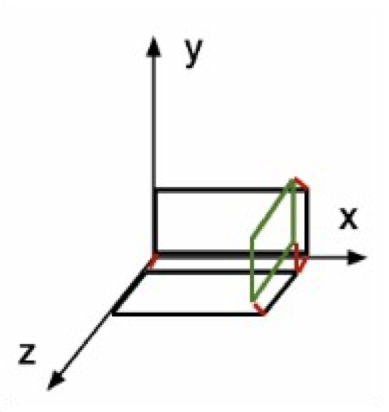


Figure 4: Segments of length greater than one standard deviation more than the mean removed.

Gap filling

Our method for filling spaces between tiles depends on the idea that tiles near each other should be connected. To accomplish this filling, we used two basic steps. First, tile faces are stitched together with three dimensional segments. These new segments provide the foundation for the boundaries that eventually define the gaps. Second, we find the simplest path from one terminus of a newly created segment to the other to create the boundary of a gap; here "simplest" means the fewest number of vertices touched. Once these boundaries are established, they can be tessellated to complete the surface between the tiles.

Segments are created between two vertices that are close to each other. Specifically, we find each vertex's closest neighbor; this neighbor must not lie in the same tile. We create a segment to the closest vertex outside of the current tile, avoiding the creation of duplicates. Once all segments are created, we remove those whose lengths are longer than some threshold; we used the average length plus one standard deviation. This threshold exists to account for inconsistencies in the data: if a vertex exists far from any others, it is probable that it should not be included in any gap boundary. The segments remaining provide the structure for the creation of the new surfaces.

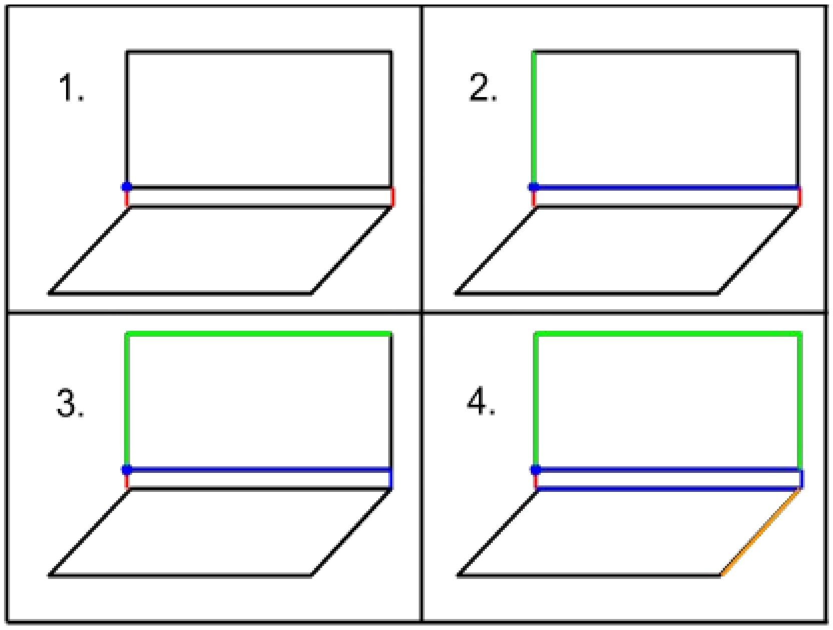


Figure 5: Progression of the gap-bounding search. Red lines are newly added segments; the left segment begins this search. Each new branch of the tree is colored differently. The blue branch successfully meets the vertex adjacent to the root, thus completely bounding the gap.

Once the segments are created, we use a breadth-first search to traverse the model, with vertices representing nodes and line segments that connect adjacent vertices representing edges. Edges may exist between adjacent vertices on a tile or between the termini of newly created segments; a node's children are those vertices either before or after the node on a tile's boundary or the opposite terminus of a newly created segment. The root node is one of the termini of a newly created segment, with the other terminus being the goal of the search. When the goal is found, we define the path used as the boundary of a gap. This search runs once for each newly created segment.

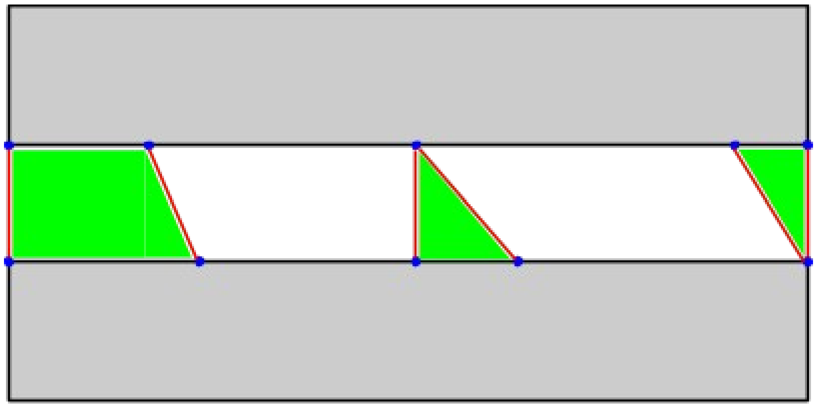


Figure 6: Illustrating the problem caused by extraneous vertices. Black lines and gray faces represent tiles. Red lines are newly added segments between close vertices and green faces represent grout polygons. Following the algorithm for gap bounding, these are the resulting surfaces.

Results

On test data mimicking the figures, the algorithm works as intended. When presented with real data, the success of the algorithm depends on the simplicity and coherence of the tiles: the simpler the edges and the closer the tiles, the more effective the solution. In areas where edges are reasonably close, new polygons, dubbed “grout” polygons, fill somewhere between one and two thirds of the gaps.

The most significant area for future improvement is the discerning selection of segments, since gap finding is a product of this choice. When nearby edges are complex (i.e. have many vertices), many unhelpful segments are drawn in the middle of these edges. For maximum effectiveness, segments should be drawn between significant corners of tiles. To implement this, segment choice could take into account the angle defined by a vertex and its neighbors on a tile: vertices defining extreme angles may be ignored. To deal with further edge complexity, it may be helpful for a segment to prefer to join vertices of similar (or opposite) angles.

Conclusion

We have presented a method for creating and refining a 3D model from processed stereoscopic images. To produce 3D surfaces, stereoscopic images showing that surface are convolved with priority relating to the visibility of the surface. Once the surfaces are generated, imperfections between them are removed by stitching the their edges together. Testing of this solution was done on a very specific set of real data. In consequence, its effectiveness when used in a similar process is proven. More testing is needed on a varying dataset to improve the robustness of the solution.

References

[1] Wang, L., Jin, H., Yang, R., and Gong, M. "Stereoscopic Inpainting: Joint Color and Depth Completion from Stereo Images." Accessed online at < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.139.171&rep=rep1&type=pdf>

[2] Jung, J. and Ho, Y. "A Hole Filling Technique in the Temporal Domain for Stereoscopic Video Generation." Accessed online at <http://www.apsipa.org/proceedings_2011/pdf/APSIPA019.pdf>

[3] Shi, W. and Cheung, C. "Performance Evaluation of Line Simplification Algorithms for Vector Generalization," The Cartographic Journal, March 2006, Vol. 43 No. 1, pp. 27-44. Accessed online at <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.5882&rep=rep1&type=pdf>