HSF: Image Compression From Segmentation using Duality and Mesh Compression

Emily Jones Robert Pieh Bjorn Wastvedt
advised by Professor Olaf Hall-Holt
MSCS Department
St. Olaf College
Northfield, MN 55057
{jonesek, pieh, wastvedb}@stolaf.edu

Abstract

We propose and implement a new image compression algorithm. Beginning with a planar straight-line graph created via image segmentation software, we create a modified dual graph consisting of triangles, with a triangle face corresponding to every polygon vertex. Using the Edgebreaker algorithm, we compress the connectivity of the triangle mesh (at a rate of about two bits per polygonal vertex) and encode the vertex and color data in the order of the connectivity of the triangle mesh. From this connectivity data, we can reconstruct the triangle mesh from which the polygonal dual graph (the original segmentation) is rebuilt. For high compression rates, we expect that our algorithm's compression ratios will be competitive with those of the JPEG and TIFF file formats, while still retaining the essential features of the image.

1. Introduction

Image compression is becoming increasingly important as image resolutions continue to grow and demand increases for transferring and storing the images. Although the standard compression methods such as JPEG and TIFF have proven themselves highly effective in many cases, it is important to continue to examine new and unexplored avenues of image compression.

Traditional image compression techniques make use of a wide variety of strategies.[2] Most simply, general lossless compression techniques such as run-length encoding or various kinds of entropy encoding are common practice. The latter may take the form of Huffman encoding, which assigns variable-length binary strings to each distinct element in a data-set based on its relative frequency, or arithmetic encoding, which does not split up the data into pieces but compresses multiple elements at once.

Quantization is frequently employed to provide lossy compression. After quantization, a near-continuous data-set (such as the color spectrum) is broken into distinct sections, which can be represented much more efficiently. Lossy image compression usually takes advantage of information unique to images: which data is valuable to the viewer, and which data can be omitted. For example, JPEG compression splits the image up into blocks, then encodes each block with varying levels of detail depending on what it contains. A complex operation blurs the data inside each block slightly, capitalizing on the fact that images (especially natural ones) often lack sharp edges. In certain situations, removing sharp edges is not desirable. Our technique indirectly deals with this problem because it is approaches the issue in a different way.

We base our compression on segmentation. Segmentation software (we utilized eriol, developed by Professor Olaf Hall-Holt[10]) returns one or several planar straight-line graphs (PSLGs) which are segmented representations of parts of the original image. Each PSLG is a polygonal mesh, which we then compress.

2. Duality

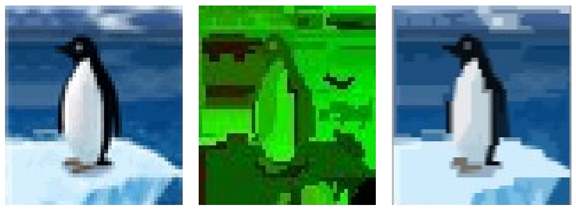


Figure 1: Original, segmentation, and .fim file

Beginning with a standard format image, we convert the image into a .ppm file and then obtain a segmentation using the eriol segmentation program. From this segmentation, we can use another function of eriol to obtain the .fim file. This file contains the vertices from the segmentation listed counter-clockwise for each polygon, followed in the same line by color data for the polygon. Thus the .fim file contains a polygonal mesh of the image.

By using the graphical property of duality, we can reduce the problem of compressing the segmentation's graph significantly. Duality refers to the fact that one mesh can be translated into another: for every vertex in the primal (original) mesh, there is a corresponding face in the dual graph, and likewise every face in the primal matches a vertex in its dual. Thus the number of polygons which touch each primal vertex is the number of sides of the dual polygon. In Figure 2, the black lines represent a portion of a primal mesh, and the blue triangles make up a portion of its dual. This duality maintains connectivity data from the original mesh, and the color and vertex data from the original mesh correspond to the appropriate vertices and polygons in the dual mesh.

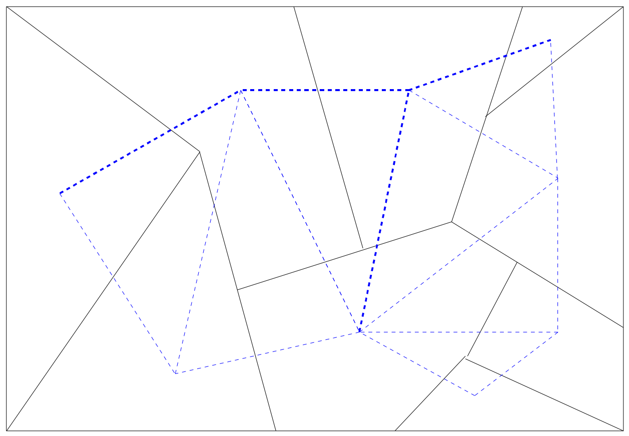


Figure 2: Polygonal Mesh Duality

There are many ways to compress a mesh, but the most efficient algorithms work with triangle meshes.[3,6,7,9] Since the vertices and faces of the original graph switch in the dual graph, if the original polygonal mesh contains only degree three vertices, the corresponding dual mesh will contain only triangles. We can store the color and vertex data in the image by associating every vertex in the triangle mesh with a color and every face with a vertex location (from the original polygonal mesh).

There are many special cases to deal with before a triangle mesh can be created through duality from a .fim polygonal mesh. The Edgebreaker algorithm was designed for compressing triangle meshes which are 2D manifolds. This means that it must be possible to traverse the entire mesh without running into holes or edges. Once we have handled the cases of vertices that are not degree three, we must add a polygon which connects all the outer edges in order to make the mesh homeomorphic to a sphere. This “infinity polygon,” which is only conceptual when represented in 3-space, is achieved by using a “point at infinity” in the dual graph, to which multiple “triangles at infinity” connect. Thus, after creating the dual graph, we arrive at a 2D manifold whose connectivity can be compressed utilizing the Edgebreaker algorithm.

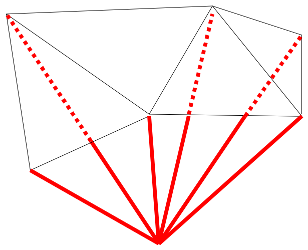


Figure 3: Triangle mesh with infinity point

In addition to adding the infinity polygon, we must also deal with vertices not of degree three. At this point in the compression, vertices of degree two are ignored and polygons with these vertices are treated as though the degree two vertices do not exist. However, we retain the vertex and color data associated with these degree two vertices and encode them later in the algorithm.

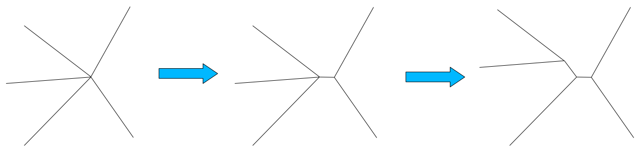


Figure 4: Greater than degree three splitting

Degree four or higher vertices are split into multiple vertices of degree three. Although visually is it easier to see the multiple vertices as existing in separate locations, in reality they coexist at the same location as separate entities. We use a counter-clockwise sort around the vertex to determine which polygons can be split into which vertices.

3. Degree Two Vertices

Dealing with the degree two vertices in the polygonal mesh before constructing our dual graph presents many more problems than do polygonal vertices of degree greater than three. A vertex which occurs in only two polygons cannot be changed into a degree three vertex without adding an extra edge to the polygonal mesh, increasing the number of polygons by splitting one polygon into two. However, the resulting polygons would hold redundant color data and result in an unnecessary increase in compressed file size. Instead, we follow a different procedure.

Refer to Figure 5 below. A sequence of consecutive degree two vertices in the polygonal mesh (g and h) will always divide two polygons since it is composed entirely of degree two vertices (2 and 4). Moreover, it will always lie between two degree three vertices (d and e), for if it did not then it would describe an open, disconnected polygon, an impossibility because of the PSLG assumption above. We encode each degree two sequence by listing the relevant information: first degree three vertex, degree two vertices, second degree three vertex, two polygon numbers. The vertices are listed in the order they appear counterclockwise around the border of the lowest numbered polygon. In this case, we have

$$e\ h\ g\ d\ 2\ 4 \tag{1}$$

Decompression can recover the degree two data given in the list above. To find where the list of degree two vertices (h g) should be inserted (and in what order), a polygon ordering convention is followed. Duality equates this ordering with a triangle vertex ordering which the triangle mesh traversal described below accomplishes. Thus the polygon numbers are encoded according to the original mesh traversal's numbering of them, and decoded in the same manner. By looking at the reconstructed vertices (ordered counterclockwise) of the relevant polygons, we can find the sequence (e d) or (d e) exactly once in each of the two bordering polygons. The degree two vertices will occur in the encoded order for the lower-numbered polygon and in reverse order for the other polygon. Thus decompression makes the following changes after the triangle mesh is traversed and the polygonal mesh (sans degree two
data) is reconstructed:

	original	encoded	final
2:	e h g d	d e	d e h g
4:	f j i c d g h e	f j i c d e	f j i c d g h e

Even though the sequence of a polygon's vertices may not be reconstructed exactly as before, the vertices will still be in counterclockwise order, preserving the polygon. Encoding the raw data in (1) is relatively expensive; however, using delta encoding, we expect to utilize the neighboring degree three vertices (d, e) to arrive at a much more efficient compression method for degree two data.

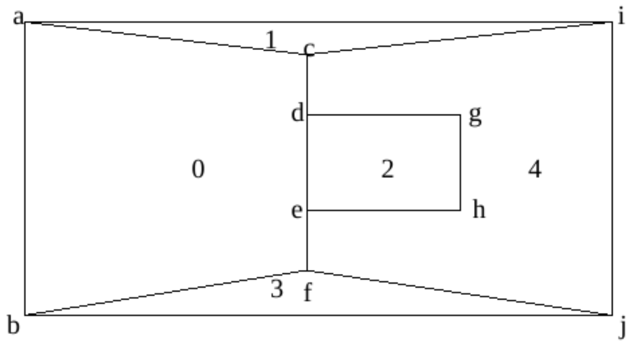


Figure 5: Degree two reconstruction

4. Triangle Mesh Compression

After a triangle mesh is created via duality applied to the segmented polygonal mesh, we compress its connectivity using the Edgebreaker algorithm, designed by Rossignac et al.[6] This algorithm walks through the 2D manifold triangle mesh, assigning a letter from the set {C, L, E, R, S} to each consecutive triangle. Each letter in this set describes one of the five distinct ways in which a triangle can relate to its adjacent triangles as the mesh is traversed. These relations have to do with whether the neighboring triangles and the corners opposite the corners in the current triangle have already been visited by the traversal. See [6] for an explanation of the algorithm, and Figure 6 for an example.

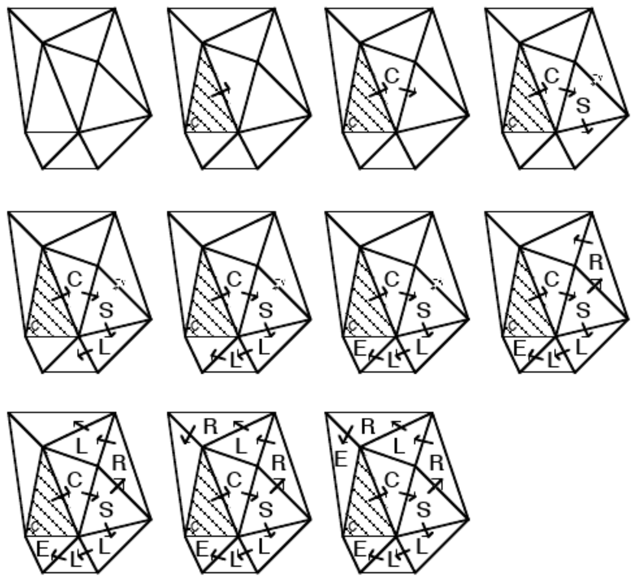


Figure 6: Edgebreaker applied to part of a triangle mesh

Due to this encoding strategy, only five distinct binary representations of maximum length three bits are needed for encoding. After applying Huffman encoding, Rossignac et al. arrived at the following representations:

C – 0   L – 110   E – 111   R – 101   S – 100

After compression, the combinatorial structure of the entire triangle mesh is represented by a CLERS sequence. Construction of the CLERS string depends on two arrays of booleans and two vectors of integers. The arrays designate which vertices and triangles have already been marked with a letter. One of the vectors represents the set of vertices in the dual mesh, numbered in the order in which they are encountered, and the other lists the triangles in order. The vertex ordering is used to store the color data for the polygonal array later, as it is also an ordering on the polygons in the original mesh. Likewise, the ordering on the dual mesh's triangles will be used to encode the vertices of the polygonal mesh.

Along with the previously explained assumptions of Edgebreaker, the Edgebreaker traversal requires the ability to find the corner lying opposite of a given corner of the triangle mesh. In order to find opposite corners, we construct an array before Edgebreaker is executed holding pointers to the corners opposite each corner in the triangle mesh. To construct this array, our algorithm begins with the corner at the infinity point of one of the infinity triangles and finds its opposite edge within the triangle. Once this edge is found, the algorithm examines the vertex which is opposite to this edge in the adjacent triangle (the triangle which shares the edge). Though the algorithm normally searches for triangles which share the edge in question with the original triangle, when there are more than two triangles which share one edge there are several possible opposite corners.

The following figure is one illustration of this problem. Four “infinity triangles” all share the same edge. Thus there are two possible vertices which could be opposite the current corner, only one of which is correct. To resolve this issue we check to see if a polygon edge (in the primal graph) connects the possible opposite corner's triangle with the current vertex. If this is so, then this is the correct opposite vertex. See Figure 7.

Once we have the proper opposite corners, we can produce the CLERS string which dictates how we store the accompanying polygon color and vertex data and convert all this to a binary file.

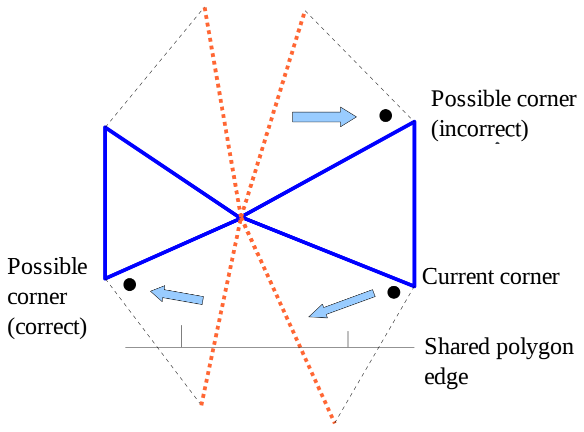


Figure 7: Opposite Problem.
Blue - Triangle mesh
Dashed – Triangles containing infinity point
Orange dashed lines - Shared edge between the four infinity triangles

5. Compression and Decompression

Using the Edgebreaker method described above, along with the detailed opposite finding and degree two encoding, we compress the data for the polygonal mesh. We start the Edgebreaker algorithm at the infinity point of the triangle mesh (representing the infinity polygon in the polygonal mesh), thus ensuring that we will know which polygon is the infinity polygon (so that we can remove it from the final polygonal mesh product) as we decompress the image.

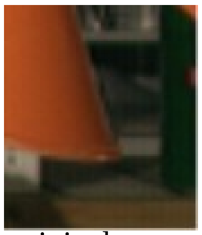
Once we have the triangle mesh compressed via Edgebreaker, we must include information about the location of each polygon vertex and the color of each polygon. As described above, we use two vectors of values, one for vertices and one for colors. Since every triangle corresponds with a vertex in the polygonal mesh, the vertices can simply be printed in the order of the CLERS sequence and no other identifying information is needed. Since every vertex of the triangle mesh represents a face in the polygonal mesh, the color data can be stored as a list of numbers. Every time we visit a vertex for the first time, it is represented in the CLERS string as a C. Therefore, we can print the colors into the compressed file in the order of the Cs in the CLERS string, thus eliminating the need for identifying information. Once the connectivity, vertex, and color data has been written to the compressed file, all that remains is the encoding of the degree two vertices, described above.

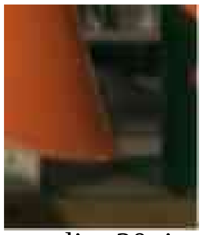
When decompressing the image, we first decode the CLERS string as the Edgebreaker algorithm describes. We then use the property of duality to convert our triangle mesh back to the corresponding polygonal mesh. Using the order of the Cs found in the CLERS string, we reassign the colors to each polygon. From the order of the triangles given in the CLERS string, we assign locations to the vertices in the polygonal mesh. Due to the way we split the degree four vertices, we have to keep only the first occurrence of any repeated vertex. Finally, we can output a .fim file identical to that which was compressed.

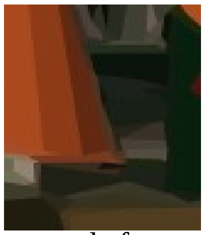
6. Results

Our algorithm has the capability to compete with standard lossy image compression algorithms such as JPEG, especially at high compression rates. It works well with images containing relatively large areas of the same or similar colors, as these area produce large polygons during segmentation, each of which can be represented by just one triangle vertex in the dual graph. Geometric patterns and distinct edges appear sharper after HSF compression than after JPEG compression because the edges of the image are already similar in nature to a segmentation's edges. JPEG tends to blur these important details.

The test image shown below comes from the well-known Tsukuba image used to test image segmentation algorithms. The piece of this image that we used is shown below: first the original, then a low quality JPEG, and finally an HSF (which has a roughly equivalent file size). Table 1 illustrates the compression ratios achieved by each compression method. The gains reported by our compression method only reflect compression of the connectivity. As the vertex and color data are raw encoded in the example, we can expect much higher compression ratios once quantization and delta encoding are implemented. The addition of gradient data will help to reduce the flat appearance of the HSF compressed image.


original .ppm


quality 30 .jpg


.hsf

tsukubaSection.ppm: 80px x 100px

format	**size**	**ratio**
.ppm	25853 B	
.jpg @ quality 70	1823 B	07.06%
.seg	5774 B	22.33%
.hsf	869 B	03.36%
.jpg @ quality 30	906 B	03.50%

Table 1: Results for test image

7. Future Work

At this time, we are able to compress the connectivity data using Huffman encoding at approximately two bits per triangle. We do not yet have the vertex and color data compressed, and in the future we plan to use a combination of run-length, delta, and Huffman encoding to compress this data. We may also employ quantization on the color data, limiting the number of colors slightly to decrease the file size.

The limiting factor in the compression process is the segmentation, as our own compression and decompression algorithms run quite quickly. Most sections of our program run in O(*n log n*) time; however a few remain which run in O(n^2) time. These will all be changed to O(*n log n*), though they do not affect the overall runtime significantly.

One key element in functionality which we have yet to implement is the processing of islands. An island is any piece of a polygonal mesh which is not directly connected to the main graph. To implement the encoding of islands, we will treat each as its own mesh and simply run the compression and decompression on each piece, combining them together at the end of decompression into a single .fim file.

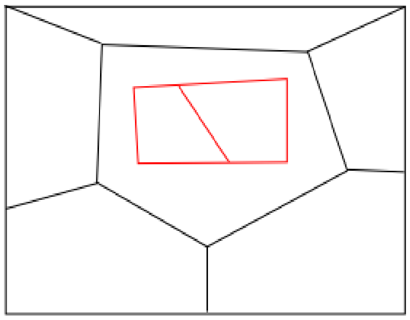


Figure 8: An island polygonal mesh within a segment of a larger polygonal mesh

A more intricate problem involves islands containing two or fewer degree three vertices. Because degree two vertices are essentially ignored when making the triangle mesh, there is no information from which the order of the degree two vertices can be determined in such a small polygonal mesh. Two simple examples of this problem are an empty square or an envelope-shaped mesh, which could be found as islands within a larger image. Figures 9 and 11 demonstrate these degenerate cases. The square is the original island, but after removal of degree two vertices no data remains for encoding. After removing degree two vertices, the envelope becomes three lines connecting the same two points in the polygonal mesh, which results in only one edge in the triangle mesh.

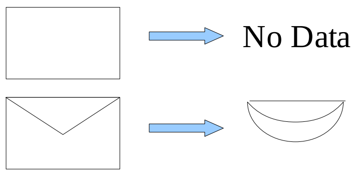


Figure 9: Degree two problems: before and after removing degree two vertices

One way to solve this problem is by adding a border to every island, creating vertices of degree at least three on the edge.

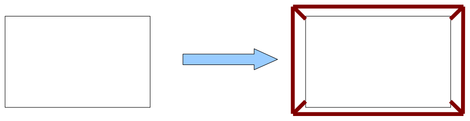


Figure 10: Adding a border to a degenerate case

Adding a border adds polygons to the mesh and thus information to the compressed file, but this is our best solution to the problem at the moment. Another problem occurs if the above examples and similar cases are the entire polygonal mesh rather than just islands found within the mesh. In this case, adding a border to the entire mesh resolves the issue as well. Admittedly, the probability of an image consisting entirely of these degenerate cases is low or nonexistent.

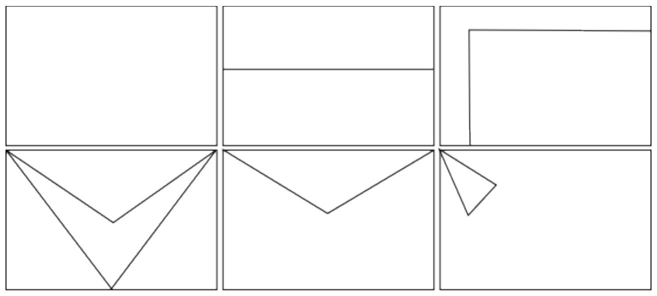


Figure 11: Possible Degenerate Cases

While the Edgebreaker algorithm which we use is highly effective at compressing the connectivity of a polygonal mesh, the color and vertex coordinate data must be compressed using other methods. There are several traditional methods for compressing vertex data in polygonal meshes, given a traversal: exactly the situation that we are in after Edgebreaker.[1,4,5,8] One possibility for compressing vertex data is to use delta encoding to store the difference vector between two consecutive polygonal mesh vertices, rather than storing each vertex location individually. Color data could also be compressed using delta encoding. Another technique for compressing these data is quantization, which results in lossy but more efficient compression. Along with these data specific compression methods, we could also apply general data compression techniques including run-length and entropy encoding.

We have made substantial progress regarding image compression based on segmentation thus far and expect even greater compression ratios in the foreseeable future with the implementation of above techniques.

References

This paper is associated with the Palantir project of St. Olaf College and developed using Olaf Hall-Holt's segmentation software (eriol).

[1] Deering, Michael. “Geometry Compression.” Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (1995): 13-20. CSA Illumina. Web. 9 Jan. 2010.

[2] Michael, Hoffman, and Khalid Sayood. Multimedia communications: directions and innovations. N.p.: Academic Press, 2001. 61-81. Print.

[3] Isenburg, Martin. “Compressing Polygon Mesh Connectivity with Degree Duality Prediction.” Graphics Interface (2002). CSA Illumina. Web. 29 Jan. 2010.

[4] Isenburg, Martin, Ioannis Ivrissimtzis, Stefan Gumhold, and Hans-Peter Seidal. “Geometry prediction for high degree polygons.” Spring Conference on Computer Graphics: Proceedings of the 21st spring conference on Computer graphics (2005). CSA Illumina. Web. 9 Jan. 2010.

[5] Isenburg, Martin, and Pierre Alliez. “Compressing Polygon Mesh Geometry with Parallelogram Prediction.” VISUALIZATION: Proceedings of the conference on Visualization 2002 (2002). CSA Illumina. Web. 9 Jan. 2010.

[6] J. Rossignac, A. Safonova, A. Szymczak, “3D Compression Made Simple: Edgebreaker on a Corner-Table,” Invited lecture at the Shape Modeling International Conference, Gemoa, Italy. May 7-11, 2001.

[7] King, D., J. Rossignac, and A. Szymczak. “Connectivity compression for irregular

quadrilateral meshes.” Technical Report TR-99-36 (1999). CSA Illumina. Web. 10 Jan. 2010.

[8] Taubin, G., and J. Rossignac. “Geometric Compression Through Topological Surgery” ACM TRANS GRAPHICS 17.2 (1998): 84-115. CSA Illumina. Web. 9 Jan. 2010.

[9] Touma, Costa, and Craig Gotsman. “Triangle Mesh Compression.” Proceedings: Graphics Interface (1998): 26-34. CSA Illumina. Web. 9 Jan. 2010.

[10] Hall-Holt, Olaf. Eriol segmentation software.