Spirale Reversi: Reverse decoding of the Edgebreaker encoding

Martin Isenburg Jack Snoeyink

Department of Computer Science
University of British Columbia
{isenburg | snoeyink}@cs.ubc.ca

October 4, 1999

Technical Report: TR-99-08

Abstract

We present a simple linear time algorithm for decoding Edgebreaker encoded triangle meshes in a single traversal. The Edgebreaker compression technique, introduced in [7], encodes the topology of meshes homeomorphic to a sphere with a guaranteed 2 bits per triangle or less. The encoding algorithm visits every triangle of the mesh in a depth-first order. The original decoding algorithm [7] recreates the triangles in the same order they have been visited by the encoding algorithm and exhibits a worst case time complexity of $O(n^2)$. More recent work [8] uses the same traversal order and improves the worst case to $O(n)$. However, for meshes with handles multiple traversals are needed during both encoding and decoding. We introduce here a simpler decompression technique that performs a single traversal and recreates the triangles in reverse order.

Key words: Triangle mesh compression, Edgebreaker, topology encoding, linear decoding.

1 Introduction

Efficiently encoding the topology of triangular meshes has recently been the subject of intense study [5, 7, 9, 10, 2, 1, 3] and many representations have been proposed. The sudden interest in this area is fueled by the emerging demand for transmitting 3D data sets over the Internet (e.g. VRML). Since transmission bandwidth is a scarce resource, compact encodings for 3D models are of great advantage.

The Edgebreaker compression technique, introduced in [7], encodes the topology of meshes homeomorphic to a sphere with a guaranteed 2 bits per triangle or less. The encoding algorithm visits each triangle of the mesh in a depth-first order by using five different operations called C, L, E, R, and S. Each triangle is labeled according to the operation that processes it. The traversal order induces the same vertex-spanning tree as in [9, 10, 2, 4]. The resulting CLERS string describes the shape of this vertex-spanning tree and the arrangement of edges that complete the mesh. This captures the connectivity of the mesh.

The decoding algorithm recreates the triangles in the same order as they have been visited by the encoding algorithm. The original decoding algorithm [7] has an asymptotic worst case complexity of $O(n^2)$. These costs are a result of the look-ahead procedure that is necessary for decoding subsequences in the CLERS sequence. These subsequences, which are encapsulated by a S operation and a corresponding E operation, reflect recursions in the Edgebreaker encoding scheme. More recent work [8] eliminates the need for this look-ahead procedure and improves the worst case to $O(n)$. However, for meshes with handles this algorithm requires multiple traversals of the mesh triangles.

We introduce here a simple decompression technique which recreates the triangles in reverse order. The CLERS sequence is processed backwards starting at the last label. This completely eliminates the look-ahead procedure of [7] or the zipping procedure of [8]. Following a suggestion by Jarek Rossignac we call this decompression scheme Spirale Reversi.

In the next section we briefly summarize the Edgebreaker encoding scheme. A detailed description of the algorithm can be found in [7]. The Edgebreaker decoding scheme is covered in Section 3 and the Wrap&zip decoding scheme is covered in Section 4. We introduce our Spirale Reversi decoding scheme in Section 5. In these sections we assume that the input mesh has no boundary, no holes, and no handles. Later we explain how encoding and decoding generalizes to meshes with boundary in Section 6, with holes in Section 7 and with handles in Section 8.

2 Edgebreaker encoding

Before we describe the Edgebreaker encoding scheme, we want to define what properties the input mesh is expected to have:

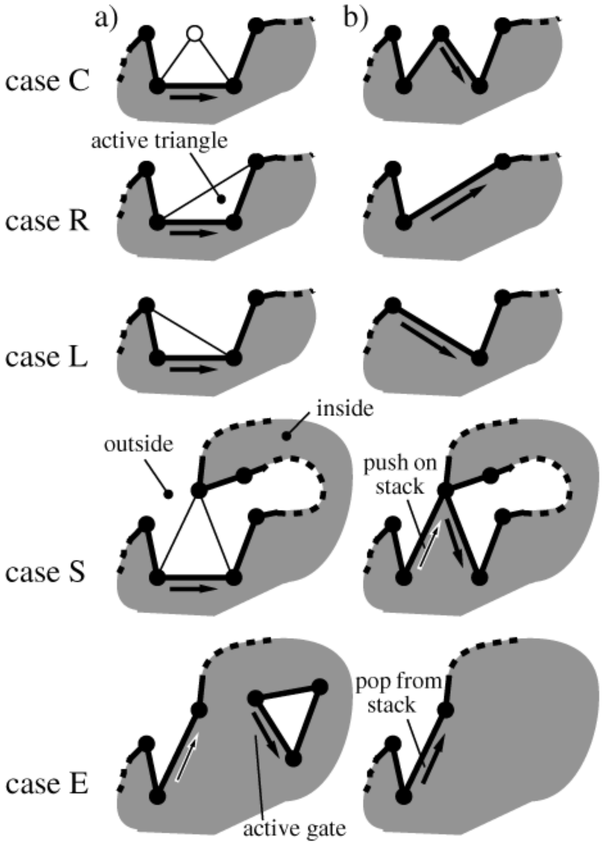


Figure 1: The Edgebreaker encoding operations C, L, E, R, and S.

1. The mesh is a surface composed of topological triangles (e.g. every face is bound by three edges).
2. The mesh has no boundary and no holes (e.g. every edge is bound by two faces).
3. The mesh has no handles (e.g. the mesh is topologically equivalent to a sphere).

Later we will describe how the Edgebreaker encoding scheme deals with meshes that have a boundary, have holes, or have handles.

The Edgebreaker encoding process starts with a triangulated mesh and produces a *CLERS string*. It visits every triangle of the mesh by including it into an *active boundary*. Initially the active boundary is an arbitrary triangle of the mesh. The encoding uses five different operations called C, L, E, R, and S to include a triangle into the active boundary. Which operation is chosen depends on how the respective triangle is attached to the active boundary. This expands (operation C), shrinks (operation R and L), splits (operation S), or terminates (operation E) the active boundary. The sequence of C, L, E, R, and S operations describes the traversal of the triangles of the mesh. The corresponding CLERS string is a compact encoding of the topology of the mesh. Now the details:

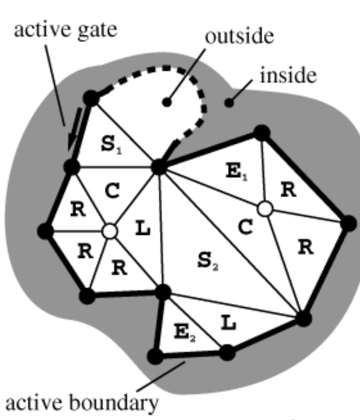
The encoding process starts off with defining an arbitrary triangle in the mesh to be initial active boundary. The three vertices of the triangle become *boundary vertices* and the three edges of the triangle become *boundary edges*. The boundary edges are directed clockwise around the triangle. The triangle itself is declared to be *inside* of the boundary; the remaining mesh is declared to be *outside* of the boundary. Initially this boundary is the only element in a stack of boundaries. The active boundary is always the top element of this stack.

One of the three initial boundary edges is defined to be the *gate* of the boundary. The gate is directed in the same way as the boundary edges. The adjacent triangle right of the gate is inside, the adjacent triangle left of the gate is outside of the boundary. The *active gate* is the gate of the active boundary. The *active triangle* is the adjacent triangle left of the active gate.

The essential element of the Edgebreaker encoding scheme is: With every operation the active triangle moves from outside to inside of the active boundary. The invariant of the Edgebreaker encoding scheme is: A triangle that lies outside of some boundary is not yet encoded. A triangle that lies inside of all boundaries is already encoded. The Edgebreaker encoding terminates after exactly $t - 1$ operations, with t being the number of triangles of the mesh. Every triangle is processed by one operation with exception of the one that defines the initial active boundary.

The active triangle is included into the active boundary with one of the five operations C, L, E, R, or S. Which operation is chosen depends on how the active triangle is attached to the active boundary. If its third vertex is not on the active boundary then operation C is used. If its third vertex is the next boundary vertex on the active boundary then operation R is used. (Remember that the boundary edges are directed clockwise around the inside.) If its third vertex is the previous boundary vertex on the active boundary then operation L is used. If its third vertex is some other boundary vertex on the active boundary then operation S is used. If its third vertex is the previous *and* the next boundary vertex on the active boundary then operation E is used. This can only happen for an active boundary of length three. See also the illustration in Figure 1a.

Each operation requires an update of the active boundary, since the active triangle moves from the outside to the inside of the boundary. The active boundary is expanded (operation C), is shrunk (operation R and L), is split (operation S), or is terminated (operation E). Each operation also requires an update of the active gate, since it moves



code	boundary length change	for s_1	for s_2
S_1	+1	*0	
C	+1	1	
R	-1	0	
R	-1	-1	
R	-1	-2	
L	-1	-3	
S_2	+1	-2	*0
L	-1	-3	-1
E_2	-3	-6	-4
C	+1	-5	
R	-1	-6	
R	-1	-7	
E_1	-3	-10	

*start of offset computation

Figure 2: Computing the offsets of the S operation for the Edgebreaker decoding.

with the active triangle to the inside of the boundary. Figure 1b illustrates the necessary updates. They are as follows:

- The **C operation** inserts one new boundary vertex, inserts two new boundary edges, and removes one boundary edge. The old gate is the removed boundary edge, the new gate is one of the inserted boundary edges. It is on the left as seen from the old gate.
- The **R and L operation** both remove one boundary vertex, remove two boundary edges, and insert one new boundary edge. The new gate is the inserted boundary edge, the old gate is one of the deleted boundary edges. The two operations differ by whether the old gate is on the right (R) or on the left (L) as seen from the new gate.
- The **S operation** splits the active boundary into two boundaries that share one boundary vertex. It inserts two new boundary edges and removes one boundary edge. The total count of boundary vertices increases by one because the shared boundary vertex is counted twice. Both inserted boundary edges become a gate for the respective boundary. The current top element of the boundary stack is popped and the two boundaries are pushed onto the stack. The new top element becomes the active boundary.
- The **E operation** removes the last three boundary vertices and the last three boundary edges. The current top element of the boundary stack is popped. If the stack is empty the encoding ends. Otherwise the new top element becomes the active boundary.

The Edgebreaker encoding scheme as presented so far captures the topology of an unlabeled mesh. Together with the right permutation of the vertex data it captures the topology of a labeled mesh. The vertex data, such as coordinates, texture information, or surface normal, are stored in the order in which the vertices are encountered during the encoding process. Vertices are encountered in the moment they are inserted into the active boundary. For the first three vertices this happens at the start of the Edgebreaker encoding when the initial boundary is defined. For all other vertices this happens during a C operation.

For triangle meshes with v vertices and t triangles that are homeomorphic to a sphere t equals $2v-4$. The traversal of the mesh triangles reaches new vertices only with the C operation. Since there are two times more triangles than vertices, half of all operations will be of type C. A straight-forward encoding that encodes a C operation with one bit and the remaining four operations with three bits is guaranteed to use no more than $2t$ or $4v$ bits. A more elaborate encoding of the CLERS sequence guarantees an even lower bound of $3.67v$ bits [6].

The detailed example in Figure 11 leads step by step through the final twelve operations of Edgebreaker encoding a mesh.

3 Edgebreaker decoding

The Edgebreaker decoding process starts with a CLERS string and produces a triangulated mesh. Two traversals of the CLERS string are needed: A preprocessing phase that computes an offset value for every S operation. A generation phase that creates the triangles in the order in which they were encoded by the Edgebreaker encoding process.

The preprocessing phase computes an offset value for every S operation. The Edgebreaker encoding uses the S operation whenever the third vertex of the active triangle is a vertex on the active boundary other than the previous or the next. In this case, the active boundary is split into two boundaries with this third vertex appearing in both. When the Edgebreaker decoding creates this triangle, it needs to know which vertex on the active boundary to use as the triangle's third vertex. The offset value that is computed during the preprocessing phase is the distance between the active gate and this vertex along the active boundary.

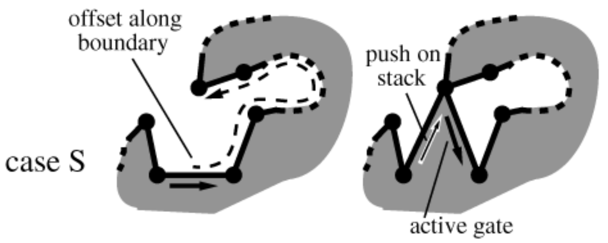


Figure 3: Using the offset of the S operation during the Edgebreaker decoding.

The computation of these offset values is very simple. The resulting change in boundary length is added up for all operations following an S operation until and including its corresponding E operation. Since pairs of S and E operations are always nested, the offset values for all S operations can be computed in a single traversal. See also the illustration in Figure 2.

The generation phase starts with creating the initial triangle. The active boundary and the gate are identified and the CLERS string is processed. What follows is an almost exact replay of the encoding algorithm. With every operation a new triangle is created and included into the active boundary. The triangle is always attached to the left of the active gate. Which vertex is used as the triangle's third vertex depends on the current operation. Only for the C operation a new vertex is created. For all other operations a vertex from the active boundary is used. For the R operation this is the next and for the L operation this is the previous vertex on the active boundary. For the S operation it is some other boundary vertex. The precomputed offset value specifies its distance from the active gate along the boundary. When the E operation occurs, the active boundary consists of only three boundary vertices. This leaves no choice for the third vertex.

The five operations of the Edgebreaker decoding perform the same updates on boundary and gate as those of the Edgebreaker encoding (see Figure 1). Only the S operation is more complex. It uses the precomputed offset to locate the third vertex for the newly created triangle as illustrated in Figure 3.

The Edgebreaker decoding scheme as presented so far reconstructs the topology of the unlabeled mesh. Using the vertex permutation that is produced by the Edgebreaker encoding, the mesh labeling is reconstructed. The vertex data is assigned to unlabeled vertices in the order in which they are encountered. Vertices are encountered in the moment they are inserted into the active boundary. For the first three vertices this happens at the start of the Edgebreaker decoding when the initial boundary is defined. For all other vertices this happens during a C operation.

Although in practice only a small fraction of operations are of type S, they imply an asymptotic worst case time complexity of $O(n^2)$ for the Edgebreaker decoding, if the active boundary is maintained in a linear data structure. Each S operation requires a linear search for the vertex specified by the offset. This cost may be reduced to $O(n \log n)$ if the active boundary is maintained in a data structure with a logarithmic instead of a linear search time. However, the more complex update operations of a data structure with logarithmic search time (such as a balanced binary tree) would increase the expected complexity from $O(n)$ to $O(n \log n)$.

The detailed example in Figure 12 leads step by step through the final twelve operations of Edgebreaker decoding a mesh.

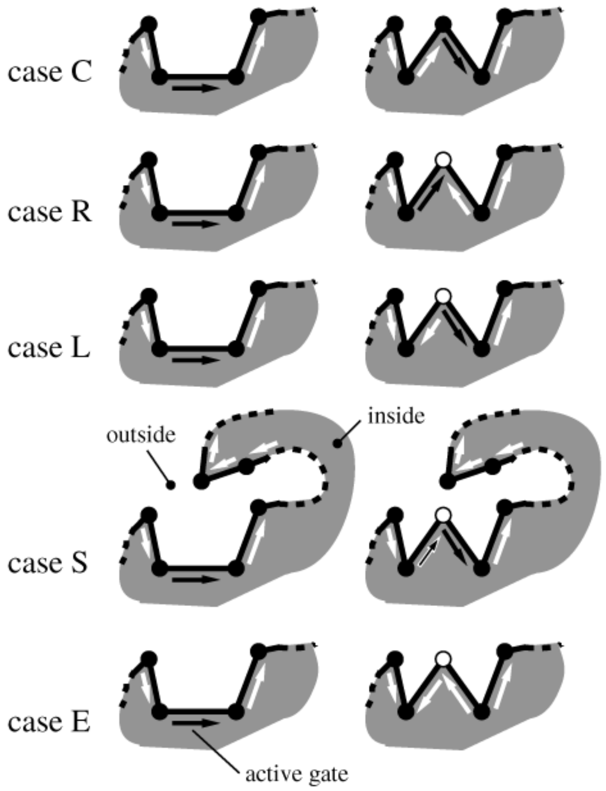


Figure 4: The Wrap&zip decoding operations C, L, E, R, and S.

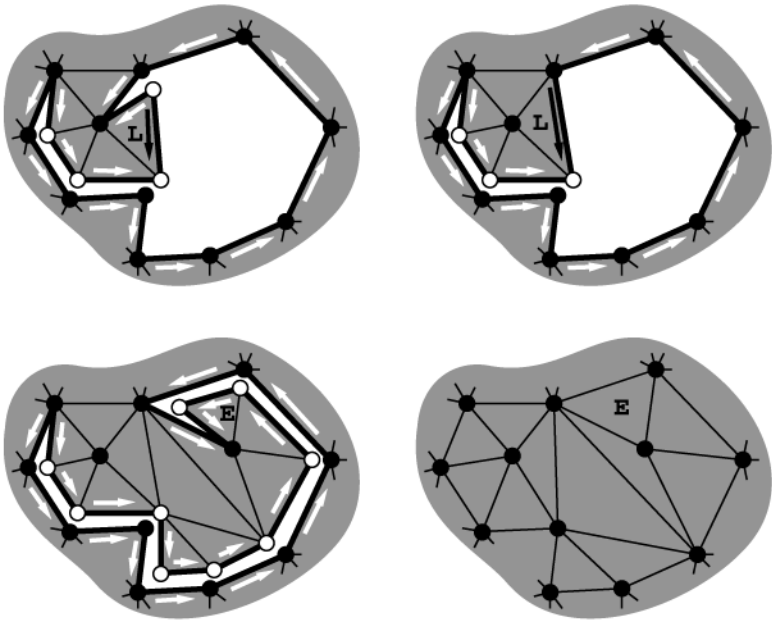


Figure 5: Single zipping after an L operation (top) and recursive zipping after an E operation (bottom).

4 Wrap&zip decoding

The Wrap&zip decoding process starts with a CLERS string and produces a triangulated mesh. Only one traversal of the CLERS string is needed. It starts with creating the initial triangle. The active boundary and the gate are identified and the CLERS string is processed. What follows is a modified replay of the encoding algorithm. With every operation a new triangle is created. The triangle is always attached to the left of the active gate. The decision which vertex is the triangle's third vertex is postponed for all operations but the C operation. Instead of some boundary vertex from the active boundary a dummy vertex is used for operations of type L, E, R, and S. This is the wrapping part of the Wrap&zip decoding. For the C operation nothing changes. Like before a newly created vertex is used.

All boundary edges except for the gate have an additional direction assigned that depends on the operation that created them. This *zip direction* is used for the zipping part of the Wrap&zip decoding. Which operation assigns which zip direction is shown in Figure 4.

Each time the zip directions of two adjacent boundary edges point to a common vertex, they are zipped together by identifying their other ends. This zipping continues recursively if the resulting vertex exhibits the same property. Whether a zip is necessary needs only to be checked after L and E operations. No immediate zipping is possible after C, R, and S operations. A zip after an L operation never starts recursive zipping, whereas a zip after an E operation always starts recursive zipping. A small example in Figure 5 illustrates single zipping after an L and recursive zipping after an E operation.

The Wrap&zip decoding scheme as presented so far reconstructs the topology of the unlabeled mesh. The mesh labeling is reconstructed in the same way as in the Edgebreaker decoding.

The wrapping and zipping technique of this decoding scheme improves on the asymptotic worst case time complexity $O(n^2)$ of the original Edgebreaker decoding. It can be shown that the number of zip operations equals the number of edges in the vertex-spanning tree. Therefore the decoding algorithm has linear time complexity.

The detailed example in Figure 13 leads step by step through the final twelve operations of Wrap&zip decoding a mesh.

5 Spirale Reversi decoding

The Spirale Reversi decoding process starts with a CLERS string and produces a triangulated mesh. Only one *reverse* traversal of the CLERS string is needed. This completely eliminates the overhead for the S and E operation pairs that is necessary for the Edgebreaker and the Wrap&zip decoding. It can be seen as a step by step reversal of the Edgebreaker encoding.

The Spirale Reversi decoding scheme uses the same boundary definitions as the Edgebreaker encoding scheme. It starts with creating an unlabeled triangle as the initial boundary. It is unlabeled in the sense

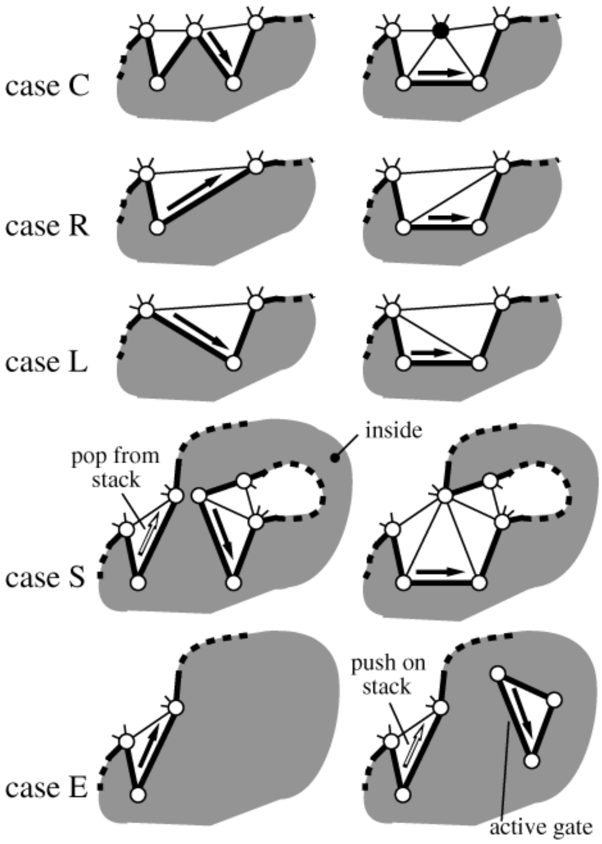


Figure 6: The Spirale Reversi decoding operations C, L, E, R, and S.

that no physical vertex is yet associated with the three boundary vertices. The triangle itself is declared to be outside of the boundary. The boundary edges are directed counterclockwise around this triangle.

One of the three boundary edges is defined as the initial active gate. Inside of the boundary is right of the gate, outside of the boundary is left of the gate. The Edgebreaker encoding was growing the inside until there was no triangle left outside. The Spirale Reversi decoding however is growing the outside until there is no triangle left inside. This reflects the *reverseness* of the Spirale Reversi decoding.

The essential element of the Spirale Reversi decoding scheme is: After every operation the triangle left of the active gate has moved from inside to outside of the active boundary. The invariant of the Spirale Reversi decoding scheme is: A triangle that lies outside of some boundary is already decoded. A triangle that lies inside of all boundaries is not yet decoded. The CLERS sequence is processed in the reverse order. Depending on the processed operation the active boundary is shrunk (operation C), is expanded (operation R and L), is merged with the next boundary on the stack (operation S), or is created new (operation E).

Reversing the encoding algorithm works as follows: With every operation a new triangle is created. This triangle is always attached to the right of the active gate. Which vertex is the triangle's third vertex depends on the type of the operation. For the C operation it is the previous boundary vertex on the active boundary. For the R and the L operation a new but unlabeled vertex is created. For the S operation it is a vertex from the boundary that is in the boundary stack directly below the active boundary. More exactly it is the vertex at the origin of this boundary's gate. The vertex at the destination of this boundary's gate and the vertex at the origin of the active gate need to be identified. For the E operation three new unlabeled vertices are created that form a new active boundary in the same way as during initialization.

The required updates of the boundary and of the gate are as follows:

- The **C operation** removes one boundary vertex, removes two boundary edges, and inserts one new boundary edge. The new gate is the inserted boundary edge, the old gate is one of the removed boundary edges. It is on the left as seen from the new gate.

- The **R and L operation** each insert one new boundary vertex, insert two new boundary edges, and remove one boundary edge. The old gate is the removed boundary edge, the new gate is one of the inserted boundary edges. The two operations differ by whether the new gate is on the right (R) or on the left (L) as seen from the old gate.

- The **S operation** merges the active boundary with the boundary that is directly below in the boundary stack. Thereby one boundary vertex from each boundary are identified into one boundary vertex. It removes two boundary edges and inserts one boundary edge. The total count of boundary vertices decreases by one because the two identified boundary vertices are only one count. Both removed boundary edges are old gates of the respective boundary. The new gate is the inserted boundary edge. The two top elements of the boundary stack are popped and the merged boundaries is pushed onto the stack.

- The **E operation** creates a new active boundary. It inserts three new boundary vertices and three new boundary edges. The new gate is any of the three boundary edges. The new active boundary is pushed on the boundary stack.

The Spirale Reversi decoding scheme as presented so far reconstructs the topology of the unlabeled mesh. Using the reverse of the vertex permutation that is produced by the Edgebreaker encoding, the mesh labeling is reconstructed. The vertex data is assigned to unlabeled vertices in the order in which they are abandoned. Vertices are abandoned in the moment they are removed from the active boundary. For the last three vertices this happens at the end of the Spirale Reversi decoding. For all other vertices this happens during a C operation.

The detailed example in Figure 14 leads step by step through the first twelve operations of Spirale Reversi decoding a mesh.

6 Handling boundaries

The Edgebreaker approach is capable of encoding the connectivity of any simple triangle mesh without holes. The scheme can easily be made capable of handling a single hole. A triangle mesh with boundary is a triangle mesh with a single hole.

Instead of selecting the loop of edges oriented clockwise around an arbitrary mesh triangle as the initial active boundary, we select the loop of edges oriented clockwise around the hole. Like before an arbitrary edge from this boundary is declared to be the initial active gate. The vertex data of all boundary vertices is stored in counterclockwise order around the hole starting at the active gate. From there Edgebreaker encoding proceeds like before.

Both the Edgebreaker decoding and the Wrap&zip decoding need additional information to decode the boundary case. They need to know the length of the initial boundary loop (e.g. the length of the hole). This can be precomputed during an initial traversal of the CLERS string. The Spirale Reversi decoding nesds no additional information. After decoding the last label of the reversed CLERS string, the active boundary loops around the hole. Then the boundary vertices are simply assigned their data in the opposite order as it was stored during encoding.

7 Handling holes

For every additional hole the Edgebreaker encoding runs into a situation in which the third vertex of the active triangle lies on the boundary of a hole. For this scenario the M operation is introduced. The active boundary is merged with the boundary of the hole by opening both at their common vertex and reconnecting them as depicted in Figure 7. The vertex data of all boundary vertices is stored in counterclockwise order around the hole starting at the common vertex. In addition to the label M of the operation the following information needs to be recorded:

- A **length** that specifies the number of vertices on the boundary of the hole.

The decoding of a hole is straightforward for all three decoding algorithms. When a label M is processed the as-

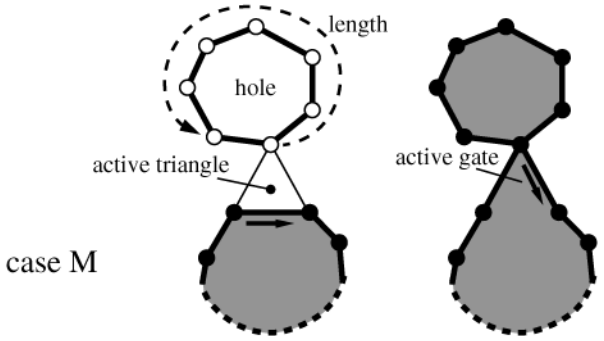


Figure 7: The Edgebreaker encoding operation M.

sociated length value is used to update the boundary accordingly. This involves assigning the data to all vertices around the hole. In Figure 7 the Spirale Reversi decoding of a hole is illustrated.

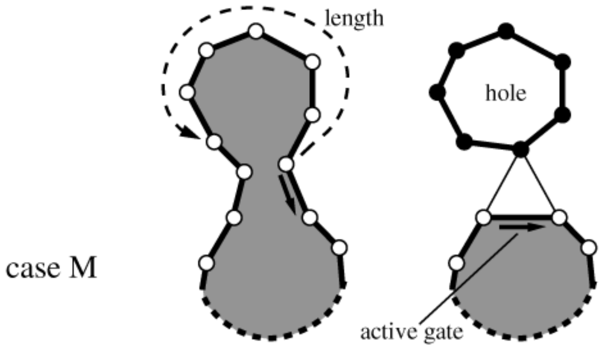


Figure 8: The Spirale Reversi decoding operation M.

8 Handling handles

For every handle the Edgebreaker encoding runs into a situation in which the third vertex of the active triangle is not on the active boundary, but on some other boundary in the stack. For this scenario the M' operation is introduced. This operation merges these two boundaries into one by opening both at their common vertex and reconnecting them as depicted in Figure 9. The respective boundary is then removed from the stack.

In addition to the label M' of the operation three integers are recorded. We modified the original Edgebreaker encoding by the last integer. This will allow to decode a mesh with handles using just a single reverse traversal of the CLERS string. The three integers are as follows:

- An **index** that specifies the respective boundary within the stack of boundaries.
- An **offset1** that specifies the counterclockwise distance between the common vertex and the gate of the

boundary from the stack.

- An **offset2** that specifies the counterclockwise distance between the gate of the boundary from the stack and the common vertex.

The original Edgebreaker decoding uses the three integers it associates with the M' operation to replay the situation encountered during the encoding. The decoding cost per M' operation is $O(n)$. However, the number of M operations is bound by the genus of the mesh and generally very small. Neither Wrap&zip nor Spirale Reversi decoding aim at improving the worst case time complexity for the M' operation.

For the Wrap&zip decoding of meshes with handles the authors had to modify the Edgebreaker encoding. The modified approach is more complicated and requires three instead of one traversal of the mesh triangles. For details we refer to the original reference [8].

The Spirale Reversi decoding of the M' operation follows the concept of reversing the encoding process. The two offsets specify the split of the active boundary and the position of the gate in the boundary that is inserted into the stack. The index specifies the position at which this boundary is inserted into the stack. This is illustrated in Figure 10

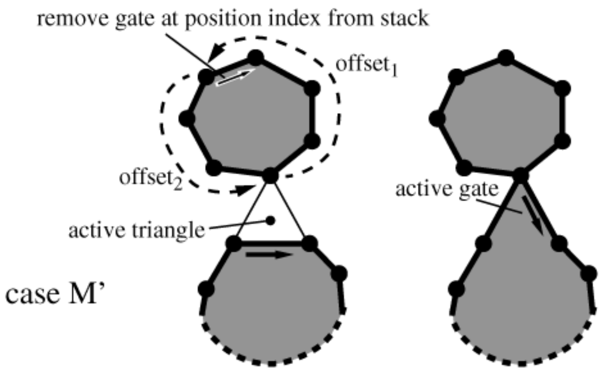


Figure 9: The Edgebreaker encoding operation M'.

9 Conclusion and Acknowledgments

We presented a simple linear time algorithm for decoding Edgebreaker encoded triangle meshes. The concept of reversing the encoding process allows to decode a mesh with a single traversal of the CLERS string. For meshes without handles our scheme eliminates the need for the look-ahead procedure used by the original Edgebreaker decoding [7] and the need for the zipping procedure used by the Wrap&zip decoding [8]. Furthermore for meshes with handles our scheme eliminates the need for multiple traversals of the CLERS string and/or the mesh triangles during both, encoding and decoding.

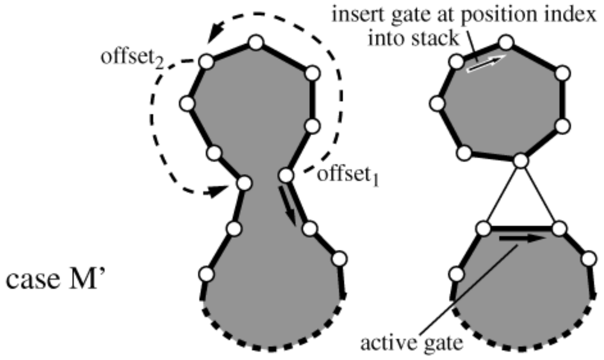


Figure 10: The Spirale Reversi decoding operation M'.

The first author thanks and Davis King for explaining the details of Edgebreaker during our *pool session* at SCG 99, Miami Beach, Florida and Jarek Rossignac for suggesting the name Spirale Reversi. This work has been supported by NSERC, IRIS, and a UBC Graduate Fellowship.

10 References

[1] L. de Floriani, P. Magillo, and E. Puppo. A simple and efficient sequential encoding for triangle meshes. In *Proceedings of 15th European Workshop on Computational Geometry*, pages 129–133, 1999.

[2] S. Gumbold and W. Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH'98 Conference Proceedings*, pages 133–140, 1998.

[3] M. Isenburg and J. Snoeyink. Mesh collapse compression. In *Proceedings of SIBGRAPI'99 - 12th Brazilian Symposium on Computer Graphics and Image Processing*, pages 27–28, 1999.

[4] M. Isenburg and J. Snoeyink. Mesh collapse compression video. In *Proceedings of SCG'99 - 15th ACM Symposium on Computational Geometry*, pages 419–420, 1999.

[5] K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. In *Discrete Applied Mathematics*, pages 239–252, 1995.

[6] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proceedings of 11th Canadian Conference on Computational Geometry*, pages 146–149, 1999.

[7] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[8] J. Rossignac and A. Szymczak. Wrap-zip: Linear decoding of planar triangle graphs. *The Journal of Computational Geometry, Theory and Applications*, 1999.

[9] G. Taubin and J. Rossignac. Geometric compression through topological surgery. In *ACM Transactions on Graphics*, pages 17(2):84–115, 1998.

[10] C. Touma and C. Gotsman. Triangle mesh compression. In *GI'98 Conference Proceedings*, pages 26–24, 1998.

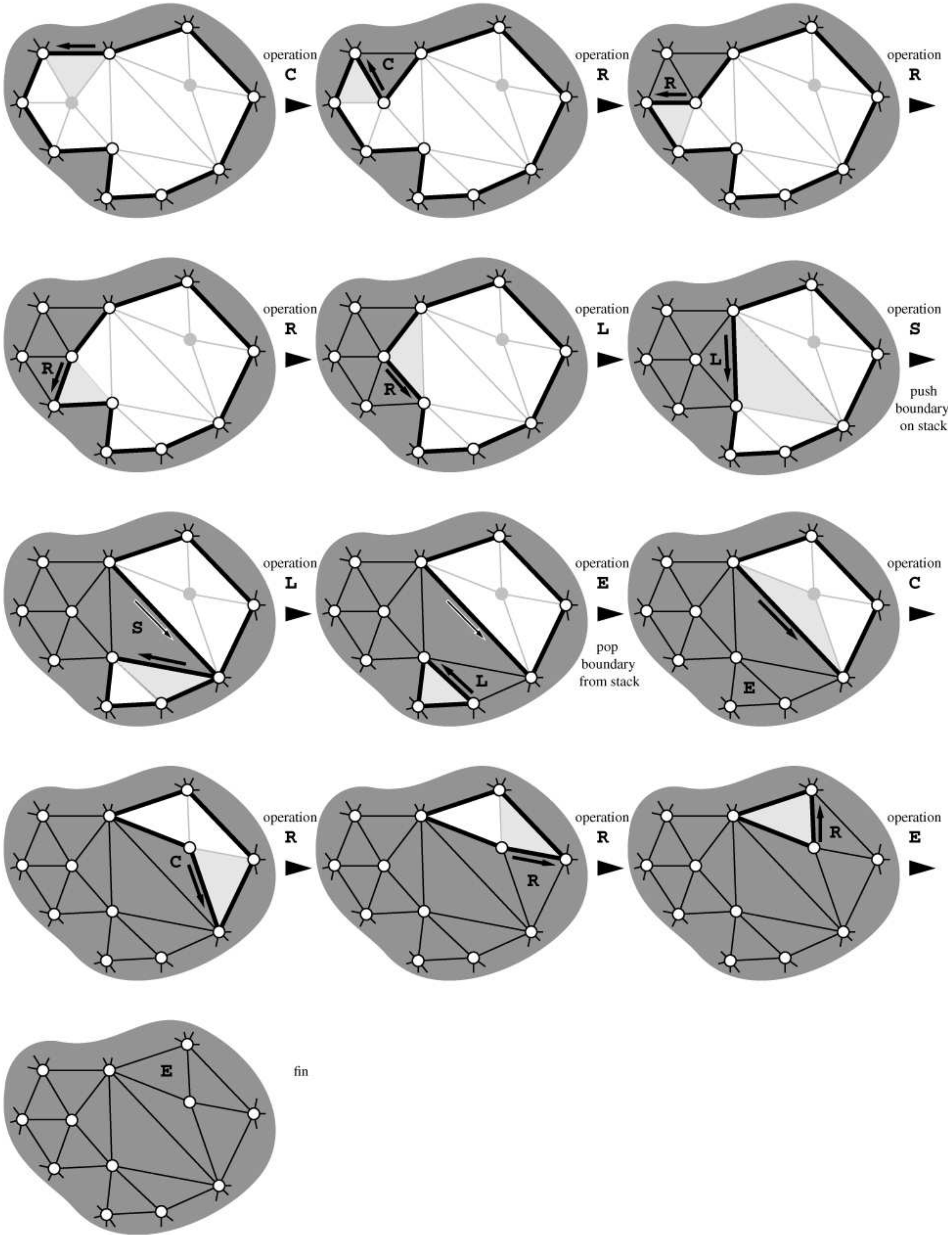


Figure 11: An example of the final twelve operations of *Edgebreaker encoding* a mesh.

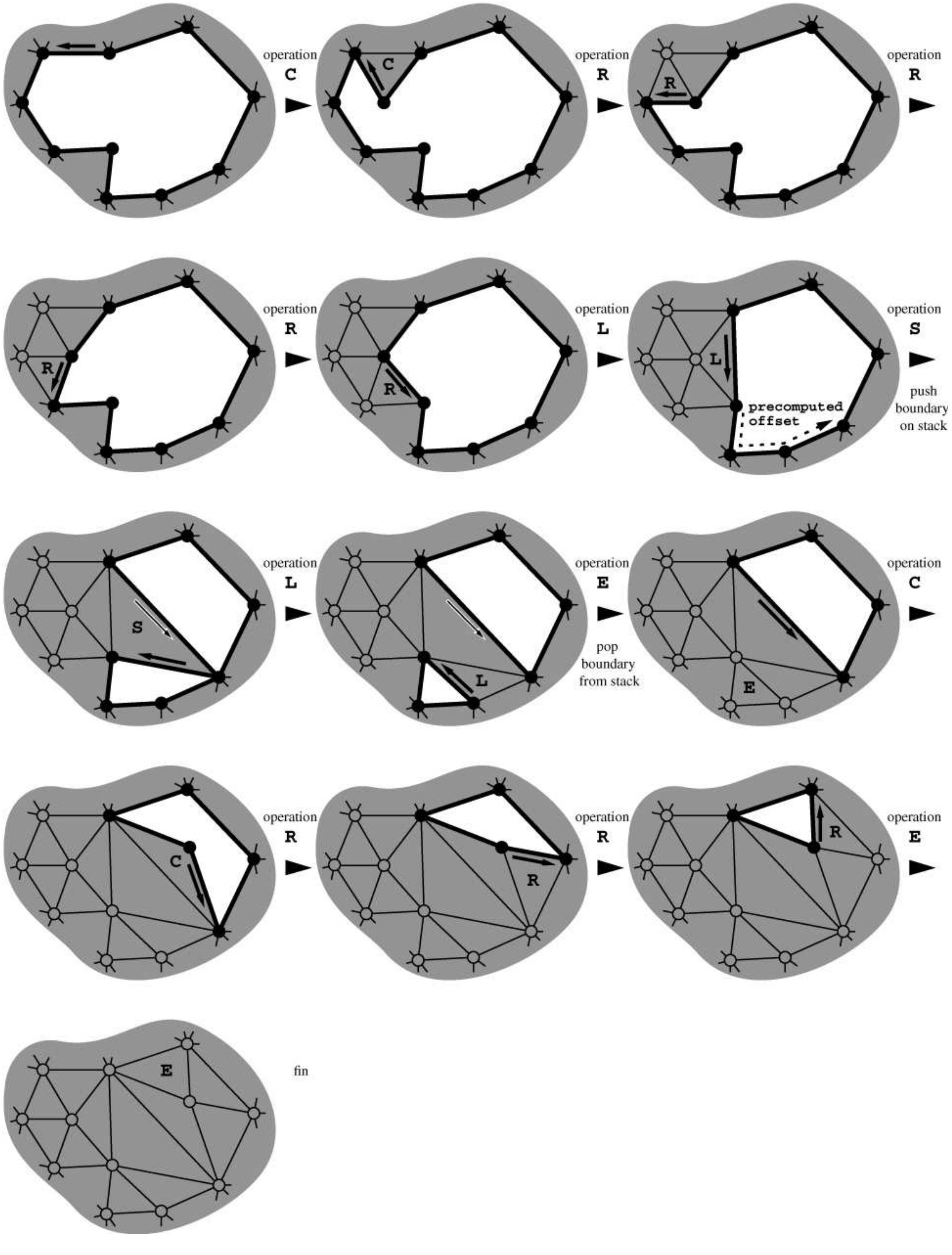


Figure 12: An example of the final twelve operations of *Edgebreaker decoding* a mesh.

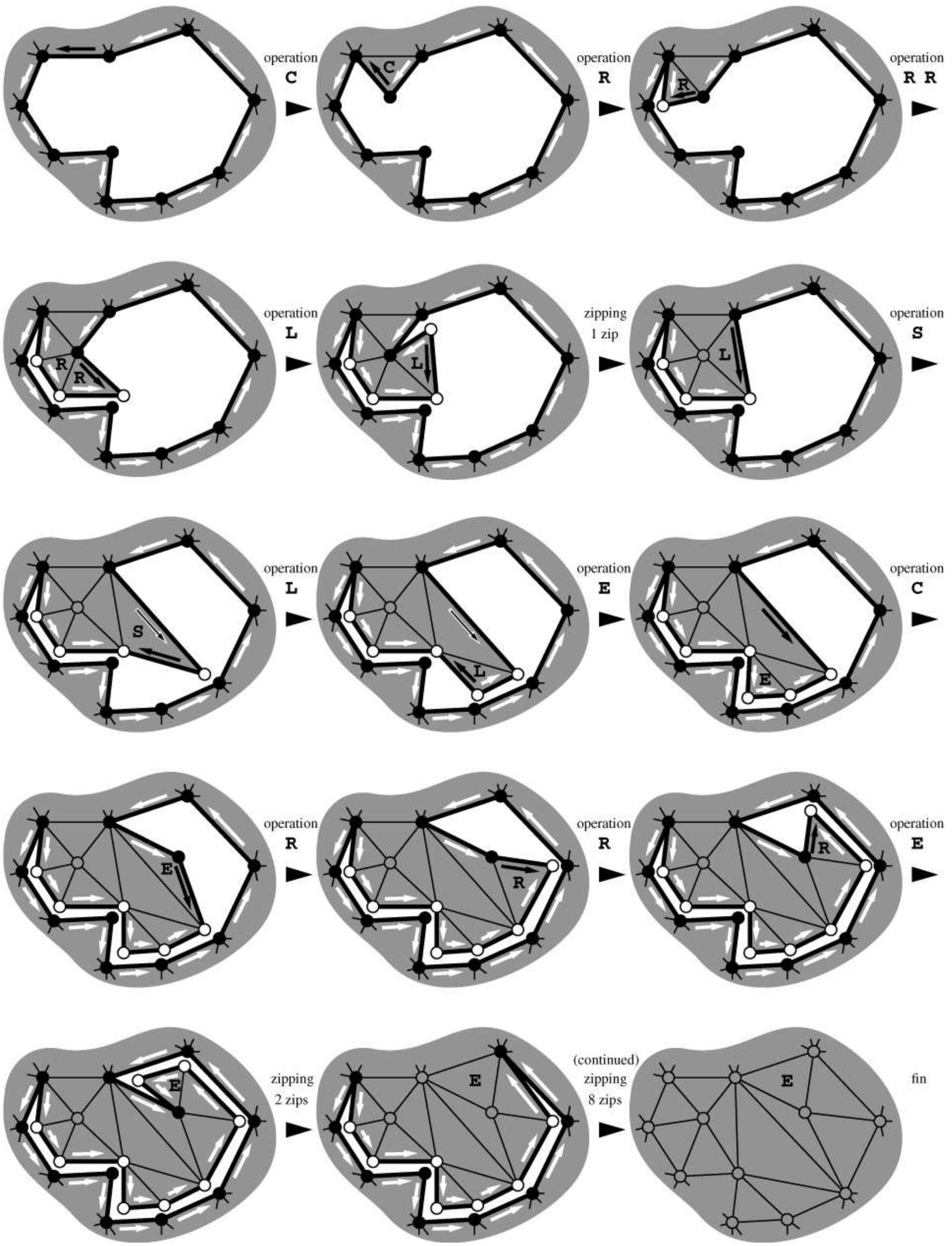


Figure 13: An example of the final twelve operations of *Wrap&zip decoding* a mesh.

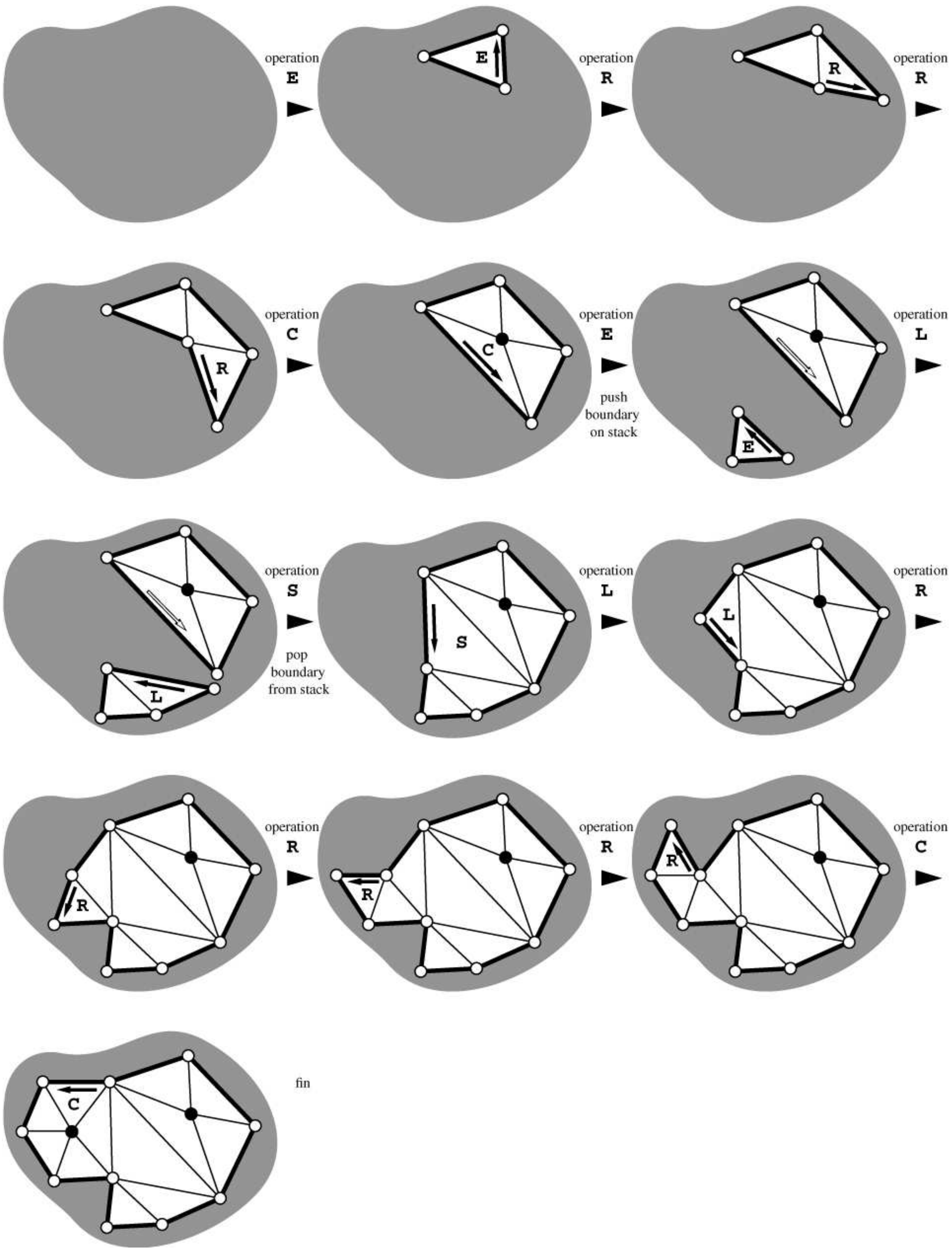


Figure 14: An example of the first twelve operations of *Spirale Reversi decoding* a mesh.