Edgebreaker Based Mesh-Coding Method

Yue Tang

B.Sc., Beijing University of Posts and Telecommunications, 2013
B.Sc., Queen Mary University of London, 2013

July 08, 2016

University of Victoria

Outline

Triangle Mesh

Triangle Mesh

- Triangle mesh: A polygon mesh with all its facets being triangles
- Two types of information in the mesh:
  - Connectivity (topological relationship)
  - Geometry (vertex positions)

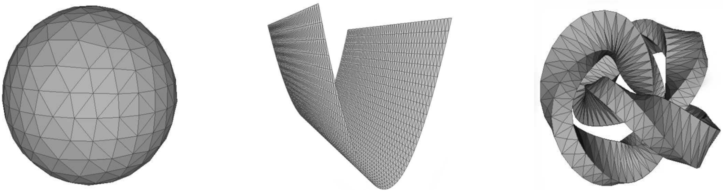


Figure 1: Examples of triangle meshes.

Introduce to Mesh Coding

Motivation

- Precise representation of the models
- High-speed data transmission
- Remote access of the datasets

The Edgebreaker Method [1]

- Developed by Rossignac in 1999
- Connectivity-driven mesh-coding method
- Main idea: triangle traversal

Halfedge Data Structure I

Halfedge Data Structure

- Each edge is represented as a pair of directed edges (halfedges)
- Each halfedge stores five pointers

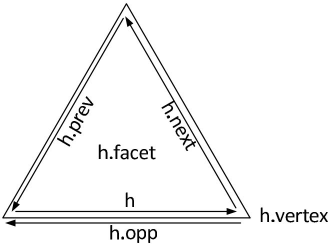


Figure 2: Pictorial view of the halfedge data structure.

Halfedge Data Structure II

Extended Halfedge Data Structure

- Used in the Edgebreaker method
- Two extra pointers for the adjacent halfedges around the border

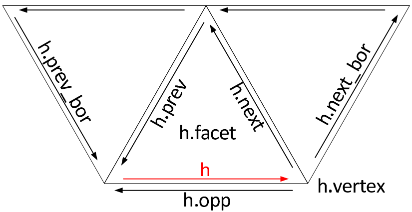


Figure 3: Pictorial view of the extended halfedge data structure.

Edgebreaker Based Mesh-Coding Method

Mesh-Coding Method

- Connectivity Coding: The Edgebreaker connectivity method
- Geometry coding: The parallelogram-prediction scheme

Main Idea

- Encoder (Triangle traversal):
  - Generate op-code sequence, encode vertices
- Decoder (Op-codes sequence traversal):
  - Reconstruct the triangle mesh

Encoding Method

Algorithm 1 Mesh encoding method

Input: Uncompressed triangle mesh
Output: Coded triangle mesh

```
1: Quantize all mesh vertices to produce integer quantization indices
2: while Not all the triangles are processed by the encoding method do
3:     Find the current triangle's type and add it to the op-code sequence
4:     Predict the position of newly encountered vertex in the triangle
5:     Encode the prediction residual
6:     Move to a particular adjacent triangle
7: end while
```

Terminology

Terminology

- Active gate `g`
- Opposite-gate vertex `g.v`
- Bounding loop
  - Active bounding loop `B`
  - Inactive bounding loop

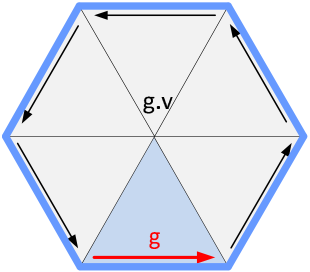


Figure 4: Pictorial view of `g`, `g.v` and `B`.

Triangle Types I

Triangle Type (Op-code)

- Describe the topological relationship between the current triangle and the remaining mesh's boundary
- Seven triangle types: C, L, R, S, E, M, and M'

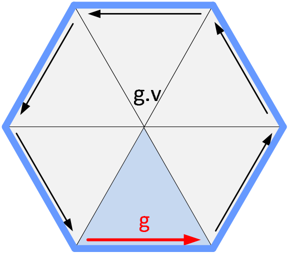


Figure 5: Triangle type distinction.

Triangle Types II

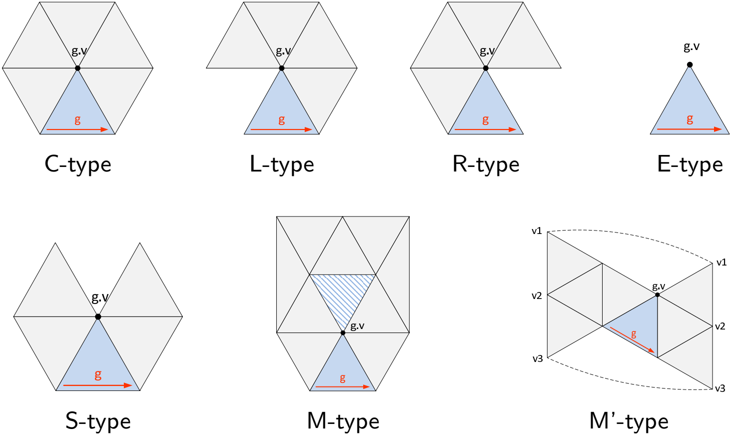


Figure 6: Pictorial view of seven triangle types.

Parallelogram-Prediction Scheme

Parallelogram-Prediction Scheme

- Predict vertex from zero or more vertices in the parallelogram
- The prediction residual Δ is obtained as

$$\Delta = r - \hat{r} \tag{1}$$

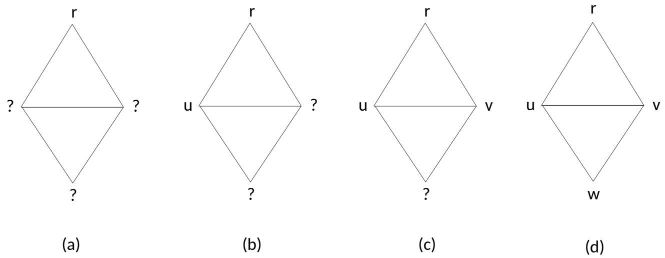


Figure 7: Pictorial view of the parallelogram-prediction scheme.

Decoding Method

Two Traversals of The Op-code Sequence

- Initialization phase:
  - Calculate quantities for the mesh reconstruction
- Generation phase:
  - Create triangles and reconstruct the vertex positions
  - The created triangle is represented by the indices of its three vertices
  - Data structure: circular doubly-linked list

Decoding Method

Algorithm 2 Mesh decoding method

Input: Coded triangle mesh
Output: Decompressed triangle mesh

```
Initialization phase: Compute required quantities
{Generation phase}
while Not all the op-codes are processed by the decoding method do
    Read the op-code and compute the three indices of the triangle
    Decode the prediction residual of newly encountered vertex
    Reconstruct the vertex position
    Move to the next op-code in the op-code sequence
end while
```

Software Implementation

Software Overview

- Programs: `encode_mesh` and `decode_mesh`
- Input mesh: OFF format, coded mesh: EB format
- Libraries: C++ standard library, CGAL, and SPL

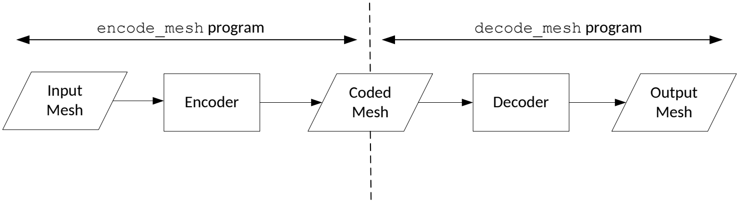


Figure 8: The overall structure of the Edgebreaker mesh-coding software.

EB File Format

EB File Format

- For the coded mesh data
- Contain six main parts

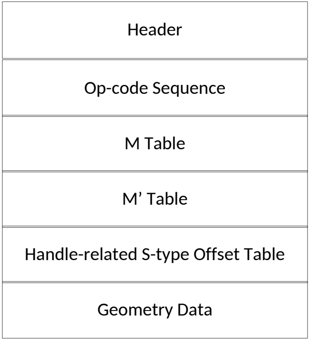


Figure 9: Structure of the EB file format.

Dataset I

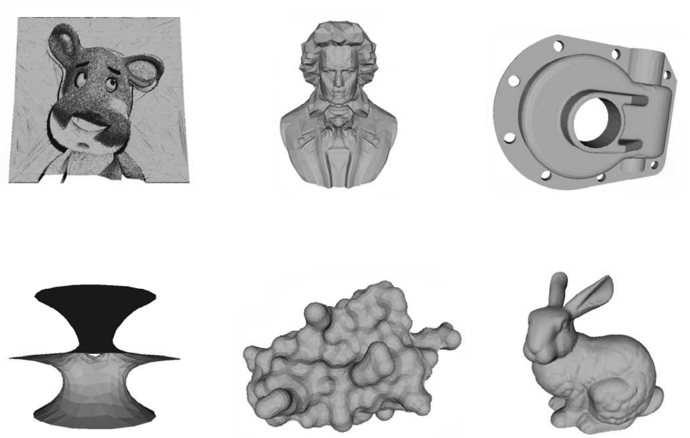


Figure 10: Dataset I.

Dataset II

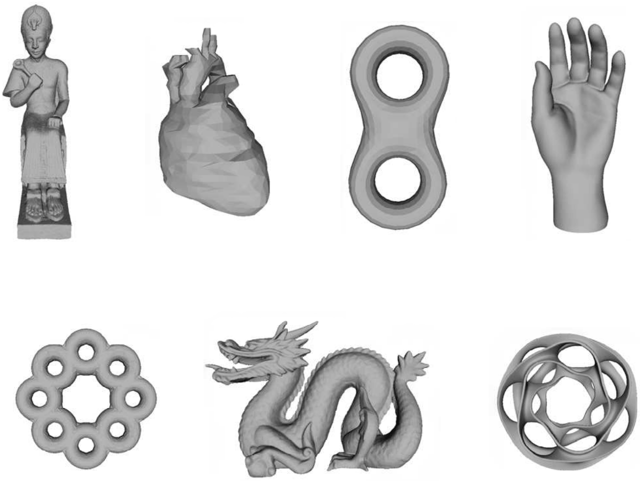


Figure 11: Dataset II.

Coding Efficiency I

Average Coding Rate

- Original OFF file: 489.22 bits per vertex (bpv)
- The Edgebreaker method: 38.58 bpv
  - Compression ratio: 12.68
- The gzip compression technique: 161.71 bpv
  - Compression ratio: 3.03
- Edgebreaker method is roughly 4.19 times better than gzip

Coding Efficiency II

Coding Rate Comparison

- Compare to the Touma and Gotsman (TG) method in [2] and the topological-surgery (TS) method in [4]

Table 1: Coding efficiency comparison

Name	Vertices	Size: bits/vertex					
		Edgebreaker		TS method		TG method	
		Geometry	Connectivity	Geometry	Connectivity	Geometry	Connectivity
beethoven	2258	10.4	3.4	15.0	4.8	10.8	2.4
blob	8036	7.7	3.3	10.3	3.4	7.9	1.7
eight	766	9.2	3.7	12.0	3.8	7.1	0.6
shape	2562	9.1	3.1	14.3	2.2	9.3	0.2
triceratops	2832	9.8	3.4	10.3	4.3	8.3	2.2

Time Complexity

Time Complexity

- Measured by execution time
- If two meshes contain the similar number of vertices, the mesh with more handles or holes takes a longer encoding time (20% to 30%)
- Code profiling:
  - Boundary traversal is the most time consuming operation

Memory Complexity

Memory Complexity

- Memory analysis based on data structures. For N vertices mesh:
  - Encoder:

$$28 \cdot (V + E + F) + 12 \cdot F + (12 + 32) \cdot 3.2\% \cdot F \approx 195N \quad (2)$$

  - Decoder:

$$(12 + 8) \cdot V + 12 \cdot (V + 2 \cdot F) + (12 + 32) \cdot 3.2\% \cdot F \approx 83N \quad (3)$$

- Actual peak memory usage:
  - encoder: 0.78 MB to 154.35 MB bytes, decoder: 0.69 MB to 22.06 MB bytes

Conclusion

Summary

- The Edgebreaker mesh-coding method produces reasonable results
- The software has implemented the Edgebreaker method effectively

Future Work

- Reduce the time required for the mesh's boundary traversal
- Potential research on the data structure improvement

References

[1] J. Rossignac, *Edgebreaker: Connectivity compression for triangle meshes*, IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, Jan. 1999.

[2] C. Touma and C. Gotsman, *Triangle Mesh Compression*, Proceedings Graphics Interface 98, pp. 26-34, 1998

[3] J. Rossignac, "Estimate function", `http://www.cc.gatech.edu/~jarek/edgebreaker/eb/PredictionScheme.htm`, 2016

[4] G. Taubin and J. Rossignac, *Geometric compression through topological surgery*, ACM Trans. Graph., vol. 17, pp. 84-15, Apr. 1998.

[5] M. D. Adams, "Signal Processing Library", `http://www.ece.uvic.ca/~mdadams/SPL`, 2015.

[6] "CGAL, Computational Geometry Algorithms Library", `http://www.cgal.org`, 2015.

THANK YOU!

Q&A

Quantization

Midtread Uniform Scalar Quantizer

- The classification rule maps a real number x to the integer quantization index k

$$k = (\operatorname{sgn} x) \left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor \tag{4}$$

where Δ denotes the quantization step size

- The reconstruction rule for this quantizer is simply

$$y = Q(x) = k \cdot \Delta \tag{5}$$

Euler's Formula

Euler's Formula for Triangle Meshes

$$V - E + F = 2 - 2g - b \tag{6}$$

V, E, F, g, and b are the numbers of vertices, edges, facets, genus, and bounding loops of the mesh, respectively.

Genus

Genus

- Number of handles in the mesh

$$g = 1 - \frac{1}{2}b - \frac{1}{2}(V - E + F), \tag{7}$$

where V, E, F, and b are the numbers of vertices, edges, facets, and bounding loops of the mesh, respectively.

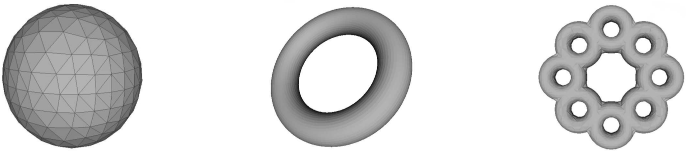


Figure 12: Triangle meshes with different genus.

OFF File Format

OFF File Format

- Header
- Vertices information
- Facets information
- Edges information (optional)

```
OFF
7    6    0
 0     0    0
 2     0    0
 1     1    0
-1     1    0
-2     0    0
-1    -1    0
 1    -1    0
3    0    1    2
3    0    2    3
3    0    3    4
3    0    4    5
3    0    5    6
3    0    6    1
```

Figure 13: The OFF data for a hexagon mesh.

Handle

Remark on Handle

- Find the split point for the S-type triangle
-

$$O_V = R_V - 2, \tag{8}$$

R_V is the No. of vertices located on the right submesh's boundary.

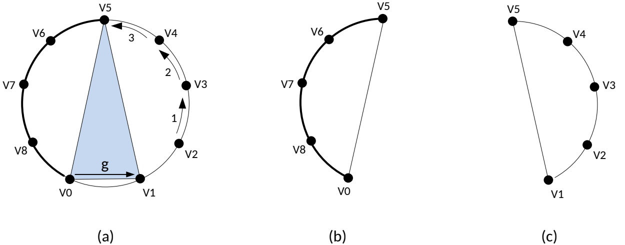
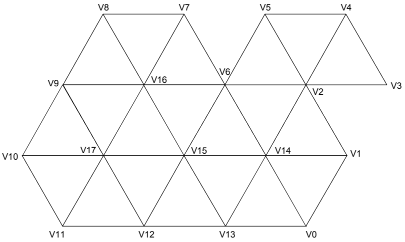
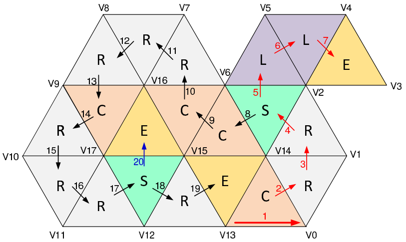


Figure 14: An example illustrating the split operation.

Coding Example I



(a)



(b)

Figure 15: Coding example. (a) The triangle mesh with one boundary and no handles. (b) The Edgebreaker encoding sequence.

Coding Example II

Offset Value

- First S-type triangle (i.e., S[1])

$$\texttt{S[1]} = 5 - 0 - 2 = 3 \tag{9}$$

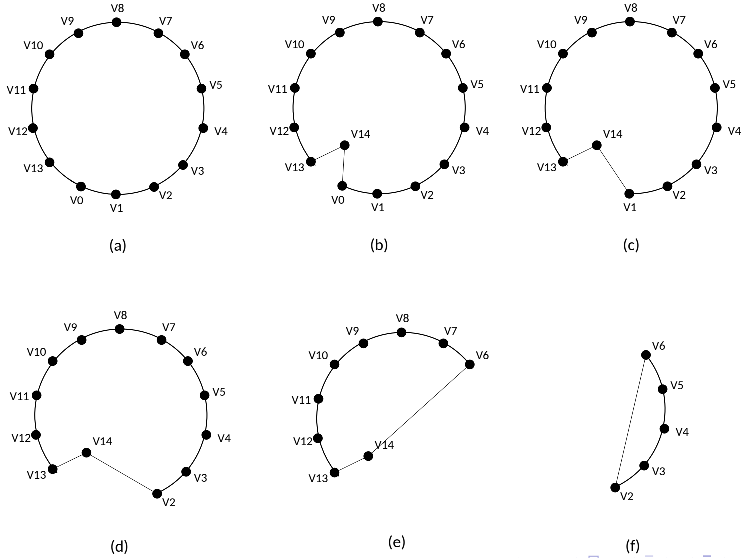
- Second S-type triangle (i.e., S[2])

$$\texttt{S[2]} = 11 - 7 - 2 = 2 \tag{10}$$

Table 2: Calculation of the e, s, and S parameters in the initialization phase

	C	R	R	S	L	L	E	C	C	R	R	R	C	R	R	R	S	R	E	E
e	-1	0	1	0	1	2	5	4	3	4	5	6	5	6	7	8	7	8	11	14
s	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2
S[s]							3												2	

Coding Example III



Parallelogram-Prediction Scheme I

Parallelogram-Prediction Scheme

- Case 1: $\hat{r} = (0, 0, 0)$, Case 2: $\hat{r} = u$
- Case 3: $\hat{r} = \frac{1}{2}(u + v)$, Case 4: $\hat{r} = u + v - w$

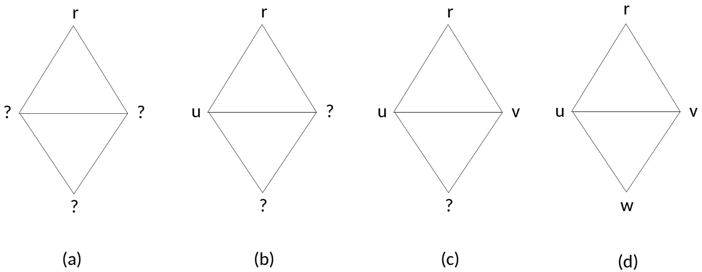


Figure 17: Pictorial view of the parallelogram-prediction scheme.

Parallelogram-Prediction Scheme II

Encode and Decode Vertex

- Predicted vertex $\hat{r}$:

$$\hat{r} = u + v - w \tag{11}$$

- Prediction residual Δ:

$$\Delta = r - \hat{r} \tag{12}$$

- Vertex reconstruction r':

$$r' = \Delta + \hat{r} \tag{13}$$

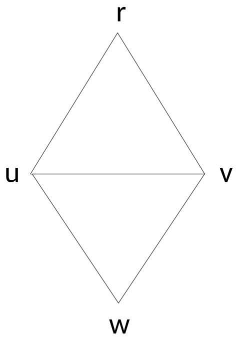


Figure 18: Geometry coding.

Parallelogram-Prediction Scheme Example

An Example

- Four vertices:

$$u = (6, 5, 0), v = (8, 1, 0), w = (4, 1, 0), \text{and } r = (10, 5, 0) \tag{14}$$

- Vertex $\hat{r}$:

$$\hat{r} = u + v - w = (6, 5, 0) + (8, 1, 0) - (4, 1, 0) = (10, 5, 0) \tag{15}$$

- Prediction residual Δ:

$$\Delta = r - \hat{r} = (10, 5, 0) - (10, 5, 0) = (0, 0, 0) \tag{16}$$

Hausdorff Distance

Methodology

- Quantize the `bunny_hole` mesh vertex by different bits
- Measure the Hausdorff distance for each reconstructed mesh

Table 3: Hausdorff distance and coding rate for `bunny_hole` mesh

Quantization (bit/coordinate)	Minimum (meter)	Maximum (meter)	Mean (meter)	Total (bits/vertex)	Geometry (bits/vertex)	Connectivity (bits/vertex)
10	0.000000	0.000057	0.000017	14.33	11.07	3.24
12	0.000000	0.000014	0.000004	18.91	15.66	3.24
14	0.000000	0.000004	0.000001	24.38	21.13	3.24
16	0.000000	**0.000001**	**0.000000**	30.19	26.93	3.24

Memory Complexity for Different Quantization Size

Methodology

- Quantize the `bunny_hole` mesh vertex by different bits
- Measure the memory complexity for each program

Table 4: Memory complexity for `bunny_hole` mesh with different quantization size

Quantization (bit/coordinate)	Encode peak memory (bytes)	Encode peak memory (bytes)
10	7389696	1671168
12	7389696	1671168
14	7389696	1671168
16	7389696	1671168

Test Dataset

Table 5: Basic information of the test meshes

Name	Vertices	Edges	Facets	Boundaries	Genus
9handle_torus	9392	28224	18816	0	9
animal	44382	132971	88590	1	0
bethoven	2258	6686	4429	1	0
blob	8036	24102	16068	0	0
bunny_hole	34835	104310	69473	4	0
casting	5096	15336	10224	0	9
dragon	50000	150000	100000	0	1
eight	766	2304	1536	0	2
fandisk	6475	19419	12946	0	0
globe_west	199065	597189	398126	0	0
hand	36616	109554	72937	3	0
heart	1280	3782	2494	8	1
heptoroid	286678	860160	573440	0	22
horse	112642	337920	225280	0	0
hypersheet	487	1407	917	3	1
lena	7864	23432	15569	1	0
ramesses	826266	2478792	1652528	0	0
shape	2562	7680	5120	0	0
tre_twist	800	2400	1600	0	1
triceratops	2832	8490	5660	0	0

Coding Efficiency

Table 6: Individual coding efficiency results

Name	Vertices	Geometry (bits/vertex)	Connectivity (bits/vertex)	Total (bits/vertex)	Gzipped (bits/vertex)	Gzipped/Edgebreaker Ratio
9handle_torus	9392	39.20	3.81	43.06	141.60	3.29
animal	44382	35.25	3.52	38.78	163.04	4.20
beethoven	2258	39.10	3.36	42.69	145.80	3.42
blob	8036	35.28	3.33	38.68	142.34	3.68
bunny_hole	34835	26.93	3.24	30.19	199.70	6.61
casting	5096	30.63	3.58	34.31	178.43	5.20
dragon	50000	33.38	3.52	36.91	220.12	5.96
eight	766	36.22	3.69	40.57	128.60	3.17
fandisk	6475	26.64	3.23	29.95	153.27	5.12
globe_west	199065	29.58	3.40	32.98	191.06	5.79
hand	36616	29.54	3.17	32.72	162.83	4.98
heart	1280	35.11	3.95	39.46	118.01	2.99
heptoroid	286678	20.41	3.06	23.47	158.59	6.76
horse	112642	20.39	3.02	23.41	169.80	7.25
hypersheet	487	41.68	4.16	46.88	169.28	3.61
lena	7864	40.94	3.51	44.51	149.78	3.37
ramesses	826266	27.18	3.43	30.61	188.96	6.17
shape	2562	38.53	3.08	41.81	115.18	2.75
tre_twist	800	43.89	3.70	48.23	157.67	3.27
triceratops	2832	34.92	3.38	38.49	174.22	4.53
median value	—	—	—	38.58	161.71	4.19

Time Complexity

Table 7: `ramesses` mesh encoding time complexity analysis with profiling. Results showing in table are the accumulated time from eleven runs.

Percentage time (%)	Cumulative time (seconds)	Self time (seconds)	Function description
37.59	79.34	79.34	Process S-type triangle
7.23	94.59	15.25	Find halfedge's incident vertex
6.09†	107.45	12.86	Encode coordinates in regular mode*
5.47	118.99	11.54	Find number of connected components in the mesh
5.00†	129.55	10.56	Entropy coding*
4.26	138.54	8.99	Updates adjacent halfedges information
3.87†	146.70	8.16	Arithmetic encoding function*
2.83	152.68	5.98	Output encoded bits*
2.76	158.50	5.82	Lookup halfedges in the mesh
2.48	163.74	5.24	Predict the vertex position

*Arithmetic coding related function from SPL library.

†Items used to calculate the time consumption in the arithmetic coder.

Time Complexity

Table 8: `heptoroid` mesh encoding time complexity analysis with profiling. Results showing in table are the accumulated time from eleven runs.

Percentage time (%)	Cumulative time (seconds)	Self time (seconds)	Function description
14.51†	5.84	5.84	Encode coordinates in regular mode*
8.84†	9.40	3.56	Entropy coding*
8.49	12.82	3.42	Find halfedge's incident vertex
7.45†	15.82	3.00	Arithmetic encoding function*
7.10	18.68	2.86	Updates adjacent halfedges information
5.14	20.75	2.07	Find number of connected components in the mesh
4.07	22.39	1.64	Output encoded bits*
3.97	23.99	1.60	Lookup halfedges in the mesh
2.86	25.14	1.15	Read mesh from OFF file
2.86	26.29	1.15	Update the probability distribution for the arithmetic coder*

*Arithmetic coding related function from SPL library.

†Items used to calculate the time consumption in the arithmetic coder.

Time Complexity

Table 9: `ramesses` mesh decoding time complexity analysis with profiling. Results showing in table are the accumulated time from eleven runs.

Percentage time (%)	Cumulative time (seconds)	Self time (seconds)	Function description
28.93†	15.81	15.81	Decode coordinates in regular mode*
18.99†	26.19	10.38	Entropy coding*
13.98	33.83	7.64	Arithmetic coder probability adjustment*
7.73	38.06	4.23	Update the probability distribution for the arithmetic coder*
7.17†	41.98	3.92	Arithmetic decoding function*
5.10	44.77	2.79	Read encoded bits*
4.61	47.29	2.52	Arithmetic coder interval check*
3.93	49.44	2.15	Read EB file from input stream
2.56	50.84	1.40	Decode vertex coordinates in bypass mode*
1.57	51.70	0.86	Predict the vertex position

*Arithmetic coding related function from SPL library.

†Items used to calculate the time consumption in the arithmetic coder.

Time Complexity

Table 10: `heptoroid` mesh decoding time complexity analysis with profiling. Results showing in table are the accumulated time from eleven runs.

Percentage time (%)	Cumulative time (seconds)	Self time (seconds)	Function description
32.32†	6.46	6.46	Decode coordinates in regular mode*
17.08†	9.88	3.42	Entropy coding*
14.21	12.72	2.84	Arithmetic coder probability adjustment*
9.08	14.53	1.82	Update the probability distribution for the arithmetic coder*
7.15†	15.96	1.43	Arithmetic decoding function*
5.50	17.06	1.10	Arithmetic coder interval check*
3.85	17.83	0.77	Read EB file from input stream
3.20	18.47	0.64	Read encoded bits*
2.10	18.89	0.42	Predict the vertex position
1.25	19.14	0.25	Decode vertex coordinates in bypass mode*

*Arithmetic coding related function from SPL library.

†Items used to calculate the time consumption in the arithmetic coder.