October 2023

Mesh Compression

Anonymous

Mesh Compression

Embodiments described herein improve the lossless compression ratio of the Corto codec. An example mesh used for 3D graphics rendering may have many vertices, where each vertex coordinate is composed of X,Y integer pixel position coordinates, and Z is a floating point depth coordinate whose value ranges from 0 (close) to 1 (far). The depth value is coded as a “half float” 16 bit value.

In order to improve lossless compression, the following improvements were considered:

- Using uncorrelated coding for X,Y,Z coordinates. Using this option, the amount of bits required to code each predicted vertex is computed independently, instead of one value for a triplet of X,Y,Z values.
- Adding Zstd as an entropy codec instead of the default one (Tunstall)

The two methods above contribute around 20% each to the compression ratio, bringing together above **9X** lossless compression for the tested meshes. Details on these changes can be found below.

Comparison to Draco codec
The Draco codec uses “Asymmetric Numeral Systems” based entropy coding. Inclusion of Zstd helps to neutralize this advantage. From an algorithmic point of view, Draco’s Edgebreaker based index compression algorithm yields better index compression however the compressed indexes contribute only around 15% to the total compressed stream, so this advantage becomes minimal. Using the tested meshes we found the Corto has around **4X** runtime performance advantage over Draco, which makes it important especially for low power target devices.

General description of the Corto codec and improvements thereto

The main stages of the Corto are described below, together with the suggested improvements. Data structures such vertex “Logs” table and Index “Clers” tables are coded using entropy coding (marked in blue), whereas headers and vertex bitstream are not.

Topology coding

The index data is sorted based on the indices, and a table with codes describing the relationship between the faces is created (“Clers” table) using a predefined set of codes, belonging to a distinct set (Left, Right, End, Boundary, Delay, Split).

The below picture explains the codes used to create the “Clers” table (taken from the following https://zenodo.org/record/2565935/files/paper.pdf)

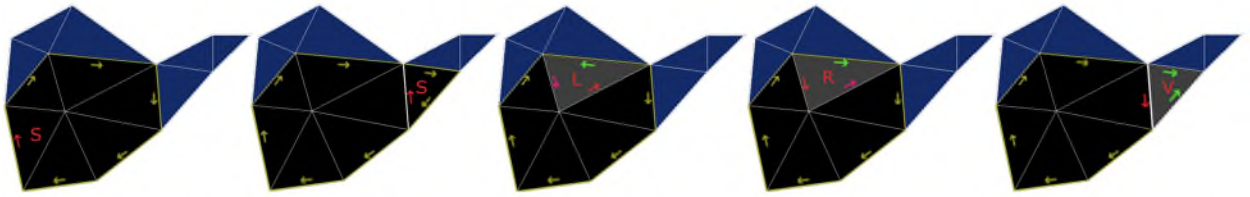

Fig. 1. Enumerating Corto various cases. From left to right: (S)kip, (S)kip, (L)eft, (R)ight, (V)ertex. Already processed triangles in black. Not-yet-processed triangles in blue. Currently encoded triangle in gray. Current active edge in red. Boundary edges in yellow and removed boundary edges in magenta. Newly added edges in green.

CLERS code table

"Clers" code table
Code0
Code1
..
CodeN

The size of the table depends on the topology. For instance a mesh with 14K indexes yields a 16K table. The table is coded using an entropy codec, which results in a high compression ratio (typically **20x-40x,** depending on the entropy codec) because of the limited set of values. In addition to this table there is an additional bitstream which includes bits for events such as "Splits", but its size is usually small, unless the topology is composed of disconnected vertices.

Vertex coding

The vertex coordinates are reordered based on the order of encoding the indexes. Each vertex coordinate, except for the first one, is predicted (subtracted) from neighboring coordinates determined by the topology coding step, using a "Parallelogram Prediction" method, where each value is subtracted from the average of its neighbors on the parallelogram (composed of two neighboring polygons)
Following is an example of the predicted coordinates:

Predicted Vertex Table

X0	Y0	Z0
DX1	DY1	DZ1
..	..	..

DXn	DYn	DZn

Correlated vs Uncorrelated Vertex coding

Using the existing “correlated” method, the number of bits required to encode each triplet of vertex coordinates is determined as the **maximum** number of bits for **any** of the coordinates.

The number of bits pushed to the bitstream buffer is different for each triplet, for example:

Vertex Bitstream Buffer

Vertex data	X0	Y0	Z0	DX1	DX2	DZ1		DXn	DYn	DZn
Number of bits	13	13	13	8	8	8	..	5	5	5

In order to enable decoding the bitstream, the number of bits for each coordinate triplet is written to a “Logs” table, which is later compressed using entropy codec.

Vertex Log Table

Number of maximum of bits required to encode any of DX,DY,DZ
13
8
..
5

The entropy codec compression ratio of the Log table is usually high, since the number of distinct bit count values is in a small range, so using separate Log tables for X,Y, and D, given the fact the D is uncorrelated in values to X,Y, yields better results, because most of the data is spent on the vertex bitstream buffer.

The uncorrelated method can be viewed as the following.

Vertex Bitstream Buffer

Vertex data	X0	DX0	DX1		Y0	DY1		Z0	DZ0	
Number of bits	11	3	2	…	11	3	..	13	8	..

Vertex Log Tables

Log table for X	Log table for Y	Log table for Z
11	11	13
2	2	8
..	2	8
3	3	5

Zstandard entropy coder

The default entropy coder used by Corto is “Tunstall”. The code references additional codecs such as LZ4, HUFFMAN, and ZLIB.

LZ4 performs better than Tunslall for “Clers” table, but worse for the “Logs” table.

Adding code that uses Zstandard entropy codec shows that it performs better than Tunstall in all cases. Zstandard can be configured with compression levels, where the lower levels are faster but yield worse compression. Level 9 was found to be fast, with close to optimum compression ratios. In addition, the codec can be trained with a training set in order to create dictionaries that can be loaded at runtime (by both encoder and decoder). Also, due to the better compression ratio of Zstandard, the bitstream buffer itself can be entropy coded, although for most bitstreams further compression is not possible.