**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC1/SC29/WG11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG2020/m54628**

**June 2020, Online**

|  |  |
| --- | --- |
| **Source** | **Panasonic corporation** |
| **Status** | **Input document** |
| **Title** | **[G-PCC] CE13.39 report on Bit count coding for predictive coding** |
| **Author** | Loi Keng Liang, Toshiyasu Sugio |

1. **Abstract**

In CE13.39, the improved bit count coding for predictive geometry coding is evaluated. The result shows significant improvement for coding performance under C2 conditions, as well as some improvement for predictive geometry compression ratio under CW conditions when slice partitioning is applied with number of points per slice.

1. **Evaluated method**

## m53538: [G-PCC] [EE13.8 Related] Predictive tree encoding modifications [1]

In current residual coding for predictive geometry coding, there is a bit count process that first computes numBits, the number of bits need to encode the absolute value of the 32 bits residual for each axis. The numBits for each axis is then arithmetic encoded with 5 bits in Figure 1. It assumes that the maximum absolute value of residual could be represented with 32 bits.

 \_aec->encode(numBits & 1, ctxs[0]);

 \_aec->encode((numBits >> 1) & 1, ctxs[1 + (numBits & 1)]);

 \_aec->encode((numBits >> 2) & 1, ctxs[3 + (numBits & 3)]);

 \_aec->encode((numBits >> 3) & 1, ctxs[7 + (numBits & 7)]);

 \_aec->encode((numBits >> 4) & 1, ctxs[15 + (numBits & 15)]);

*Figure 1*

However, the maximum absolute value of residual depends on the bit depth of the point cloud in each axis and it would change. By considering this feature, the improvement modification was made as Figure 2, in which the value of numBits is arithmetic encoded with variable bits based on the bit depth of the point cloud in each axis.

 int resi\_bit\_cnt = ceillog2(uint32\_t(geom\_bit\_depth[k] + 1));

 for (int n = 0; n < resi\_bit\_cnt; n++) {

 \_aec->encode((numBits >> n) & 1, ctxs[(1<<n) - 1 + (numBits & ((1 << n) - 1))]);

 }

*Figure 2*

The bit depth of the point cloud is stored as geom\_bit\_depth variable in GeometryBrickHeader and encoded into the bitstream, so that it can be decoded for use to determine the residual bit count that controls the amount of context switching for arithmetic decoding at the decoder side.

During the conduct of the experiment, it was found that applying improved bit count coding for prediction geometry coding alone is insufficient to pass CTC test. Currently prediction value, where residual value is calculated from, is unbounded as shown in Figure 3.

auto residual = point - pred;

*Figure 3*

As prediction value is unbounded, there are cases where the number of bits required to encode the absolute residual value exceeds the geometry bit depth. This leads to both encoding and decoding error.

Clipping on the calculated prediction value as shown in Figure 4 is needed to bound it within geometry bit depth limits of the point cloud. This is to ensure that residual value is also bounded within geometry bit depth limits (point is within geometry bit depth limits).

 for (int n = 0; n < 3; n++) {

 pred[n] = std::max(std::min(pred[n], (1 << geom\_bit\_depth[n]) - 1), 0);

 }

*Figure 4*

1. **Experimental results**

CE13.39 method was implemented on TMC13v10.0 software [2] with predictive geometry coder enabled and evaluated under C2 and CW conditions in CTC. The computing platform is Linux 64bits and the executables were compiled on 64-bit Linux with gcc 5.4.2.

The baseline is set to TMC13v10.0 software with predictive geometry coder enabled, Azimuth pre-sorting and slice-partition-framework divided into slices using 3 different maximum points per slice limits.

Anchor vs CE13.39 simulation results run under C2 and CW conditions in CTC with slice partitioning maximum points limit set at 512 are as shown in Table 1-1.





Table 1-1: Summarized result of CE13.39 method with slice=512
and azimuth pre-sorting

As expected for lower dimension slices, there is significant improvement for coding performance under C2 conditions with overall average of -4.4%. In addition, there is some improvement observed for geometry compression ratio under CW conditions with overall average of 97.7%.

Anchor vs CE13.39 simulation running under C2 and CW conditions in CTC with slice partitioning maximum points limit increased to 1024 are tested next with results as shown in Table 1-2.





Table 1-2: Summarized result of CE13.39 method with slice=1024
and azimuth pre-sorting

For maximum number of points per slice doubled to 1024, there is still improvement for coding performance under C2 conditions with overall average of -3.2%. There is also some improvement observed for geometry compression ratio under CW conditions with overall average of 98.4%.

Lastly, anchor vs CE13.39 simulation running under C2 and CW conditions in CTC with slice partitioning maximum points limit increased to default 1100000 are tested next with results as shown in Table 1-3.





Table 1-3: Summarized result of CE13.39 method with slice=0
and azimuth pre-sorting

Evening with no slice partitioning applied, there is still some improvement for coding performance under C2 conditions with overall average of -0.9%. There is still a small improvement observed for geometry compression ratio under CW conditions with overall average of 99.7%.

1. **Conclusion**

In this contribution, we evaluated the improved bit count coding for predictive geometry coding. The result showed significant improvement for coding performance under C2 conditions, as well as some improvement for predictive geometry compression ratio under CW conditions when slice partitioning is applied with number of points per slice. Based on this result, it is recommended that CE13.39 method be adopted in next G-PCC specification and TMC13 software.

# References

1. "[G-PCC] [EE13.8 Related] Predictive tree encoding modifications", ISO/IEC JTC1/SC29/WG11 Doc. m53538, Alpbach, AT, April 2020
2. TMC13v10.0 software

http://mpegx.int-evry.fr/software/MPEG/PCC/TM/mpeg-pcc-tmc13