**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG2019/m51012**

**October 2019, Geneva, CH**

|  |  |
| --- | --- |
| **Source** | **Apple Inc.** |
| **Status** | **Input document** |
| **Title** | **[G-PCC][New proposal] Predictive Geometry Coding.** |
| **Author** | **Zhenzhen Gao, David Flynn, Alexis Tourapis, and Khaled Mammou** |

# Abstract

Geometry-based Point Cloud Compression (G-PCC) uses an octree-based approach to compress the point cloud geometry. While octree-based geometry coding provides excellent compression performances for dense point cloud, the gains for sparse Lidar point cloud such as Category 3 content are limited. For this type of content, the computational complexity required to build the octree and to compute the neighborhood occupancy information is too high for the obtained coding benefits. Furthermore, the octree-based coding is inherently a high latency scheme since it requires a high number of points before the compression/decompression process could start. Otherwise, poor compression results may be obtained.

In this contribution, we propose a predictive geometry coding scheme as an alternative to the octree-based approach. The predictive scheme offers the following advantages:

• Supports low latency applications/streaming, and

• Low complexity decoding.

# Proposed approach

**The proposed solution targets only Category 3 content.** It starts by defining a prediction structure on the point cloud. Such structure could be described by a prediction tree (see Figure 1), where each point in the point cloud is associated with a vertex of the tree. Each vertex could predict only from its ancestors in the tree. Various prediction strategies are possible. We considered the following four predictors:

* No prediction
* Delta prediction (i.e., p0)
* Linear prediction (i.e., 2p0-p1),
* Parallelogram predictor (i.e., 2p+p1-p2),

where p0, p1, p2 are the positions of the parent, grandparent and grand-grandparent of the current vertex.



Figure 1: Example of prediction tree.

The tree structure is encoded be traversing the tree in a depth order and encoding for each vertex the number of its children. The positions of the vertices are encoded by storing the chosen prediction mode and the obtained prediction residuals. Arithmetic coding is used to further compress the generated values.

Building the optimal prediction tree is an NP-hard problem. The encoder could use different heuristics to build sub-optimal prediction trees that offer various compromises in terms of:

* Computational complexity
* Memory requirements
* Latency
* Robustness to errors, and
* Energy consumption among others.

In the reminder of this section we discuss two strategies to build prediction trees adapted for three use cases:

1. high latency use-case slow mode,
2. high latency use-case fast mode, and
3. Low latency use-case.

## High latency strategy slow mode

The points are re-ordered based on their Morton codes. A kdtree data structure is used to keep track of potential predictors. At the beginning, the kdtree is empty. The points are visited iteratively in the selected order. The k-nearest neighbors in the kdtree of the current point are determined and one of them is chosen as the predictor based on various criteria such as

* the magnitude of the prediction residuals,
* the number of the node children, and
* the frequency of the chosen prediction mode among others.

Once a predictor is chosen, the current vertex is added to the children of the vertex associated with the predictor. Next, new predictors are created based on the current vertex (see the previous section for examples of predictors) and the predicted positions are added to the kdtree. This process is repeated until all the points are traversed. At the end of this process the prediction tree structure is determined.

## High latency strategy fast mode

Uses the same algorithm as the “high latency slow mode”, while replacing the kd-tree search by an approximate nearest neighbor search according to the Morton order. Here, a search range of 128 was used.

## Low latency strategy (window-based prediction)

The encoder processes the points in the same order as they are received. A buffer of limited size is used to limit the system latency. When looking for the best predictor for each vertex, the encoder will consider only the points that are in the buffer.

# Experimental results

## High latency strategy slow mode vs. TMC13v7

This uses cases targets applications were encoding is not constrained. However, decoding complexity and decoding time are highly constrained due to the end device capabilities and power budget. Figure 2, compares the compression performance, the encode/decode time and the memory loads/stores of the proposed predictive geometry compression scheme compared to the octree-based approach in TMC13v7. The proposed approach offers 3.5%-15% gain in terms geometry compression.

The decoding process is 2.3x-5.4x faster, with:

* 23-33% lower number of instructions,
* 31%-44% lower number of memory loads, and
* 13%-18% lower number of memory stores.

The encoding process is 2.7x-5.5x slower, with:

* 84-104% higher number of instructions,
* 47%-77% higher number of memory loads, and
* 79%-84% higher number of memory stores.

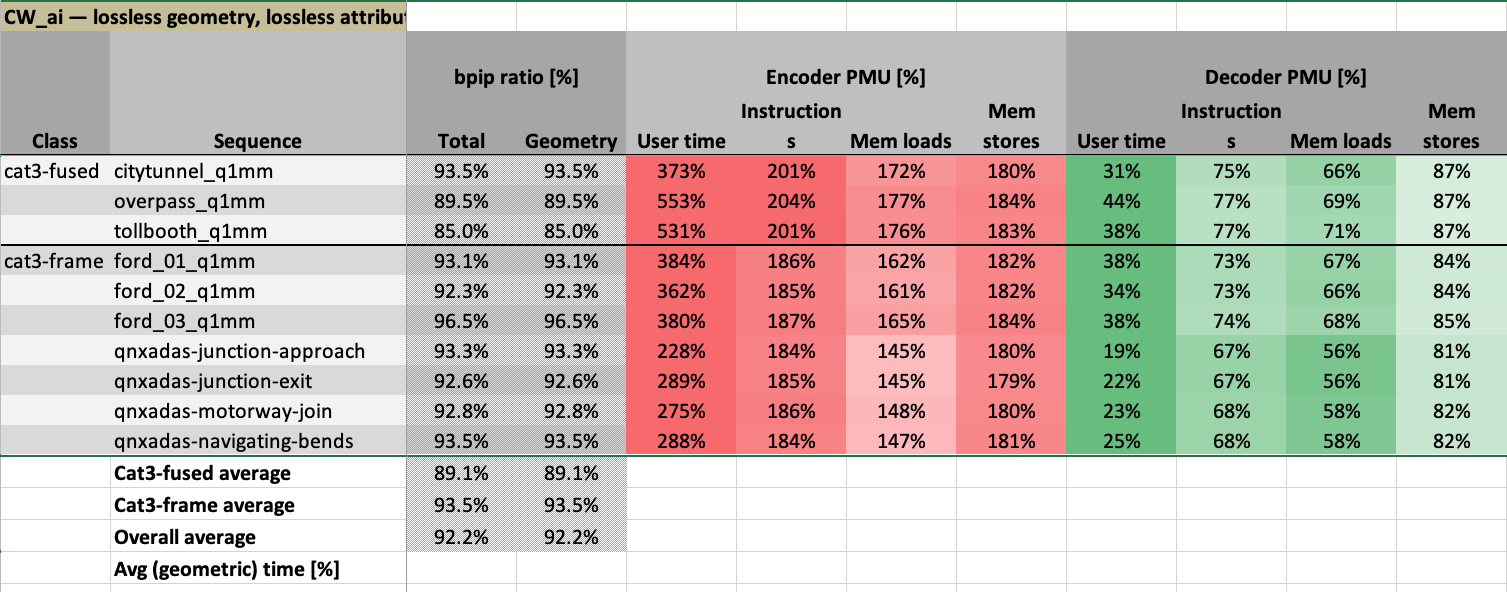


Figure 2: High latency predictive geometry coding slow mode vs. TMC13v7.

## High latency strategy fast mode vs. TMC13v7

In this section, we report the performance of the high latency strategy fast mode described in Section 1.2. Here, a search range of 128 was used, when looking for the best predictor. Figure 3 compares the compression performance, the encode/decode time and the memory loads/stores of the proposed predictive geometry compression scheme compared to the octree-based approach in TMC13v7. The proposed approach offers -2.5%-5.8% gain in terms geometry compression.

The decoding process is 2.4x-5.9x faster, with:

* 23-33% lower number of instructions,
* 29%-44% lower number of memory loads, and
* 13%-19% lower number of memory stores.

The encoding process is 1.1x-2.2x faster, with:

* 3-17% lower number of instructions,
* 14%-32% lower number of memory loads, and
* 11%-18% lower number of memory stores.

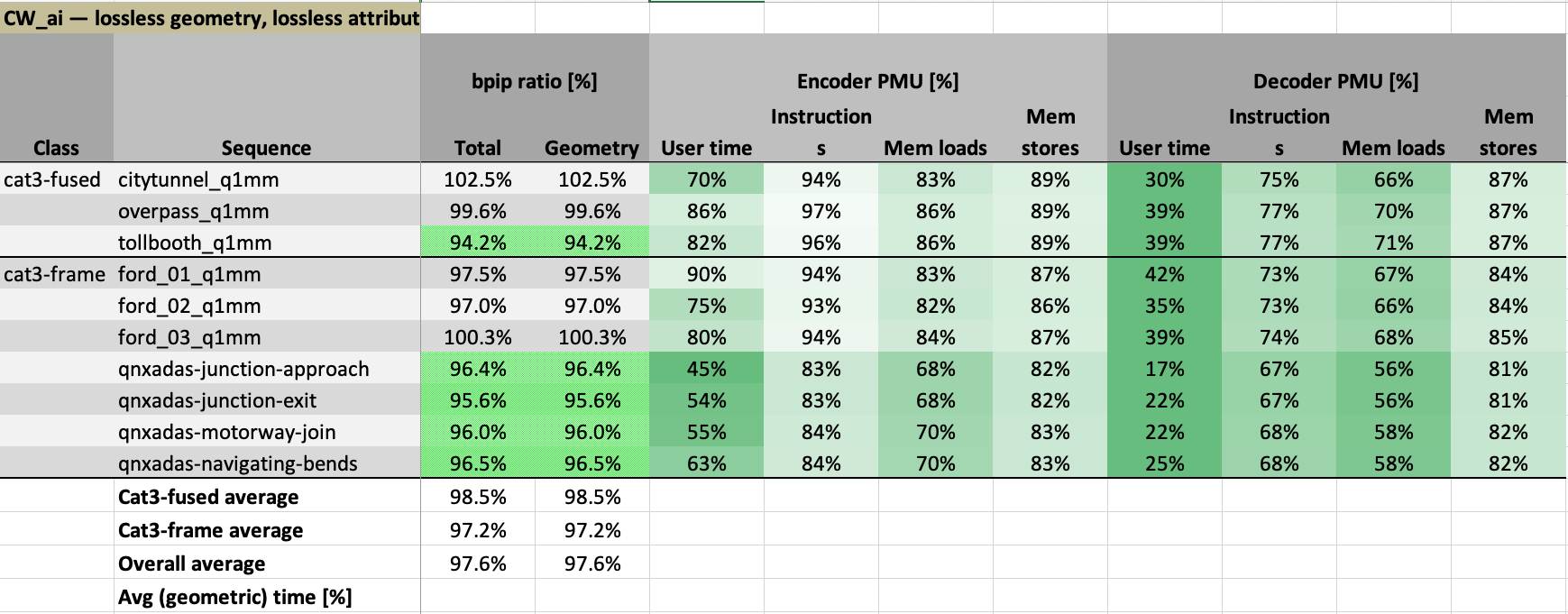


Figure 3: High latency predictive geometry coding fast mode vs. TMC13v7.

## Low latency strategy slow mode vs. TMC13v7

In this section, we compare the low latency predictive geometry to TMC13v7. The reader should note TMC13v7 does not support low latency uses cases and requires buffering the entire geometry brick before starting the decode process. Furthermore, the TMC13v7 decoding requires all the octree layers to be decoded to output the first point. On the other hand, the proposed low latency predictive geometry coding has maximum latency of 128 points on the encoder side and starts outputting points on the decoder side without extra delays.

Figure 4 compares the compression performance, the encode/decode time and the memory loads/stores of the proposed predictive geometry compression scheme compared to the octree-based approach in TMC13v7. The proposed approach offers -2.6%-5.1% gain in terms geometry compression.

The decoding process is 2.5x-5.5x faster, with:

* 23-33% lower number of instructions,
* 29%-43% lower number of memory loads, and
* 13%-18% lower number of memory stores.

The encoding process is 1.1x-1.5x faster, with:

* -3-7% lower number of instructions,
* 14%-20% lower number of memory loads, and
* 11%-17% lower number of memory stores.

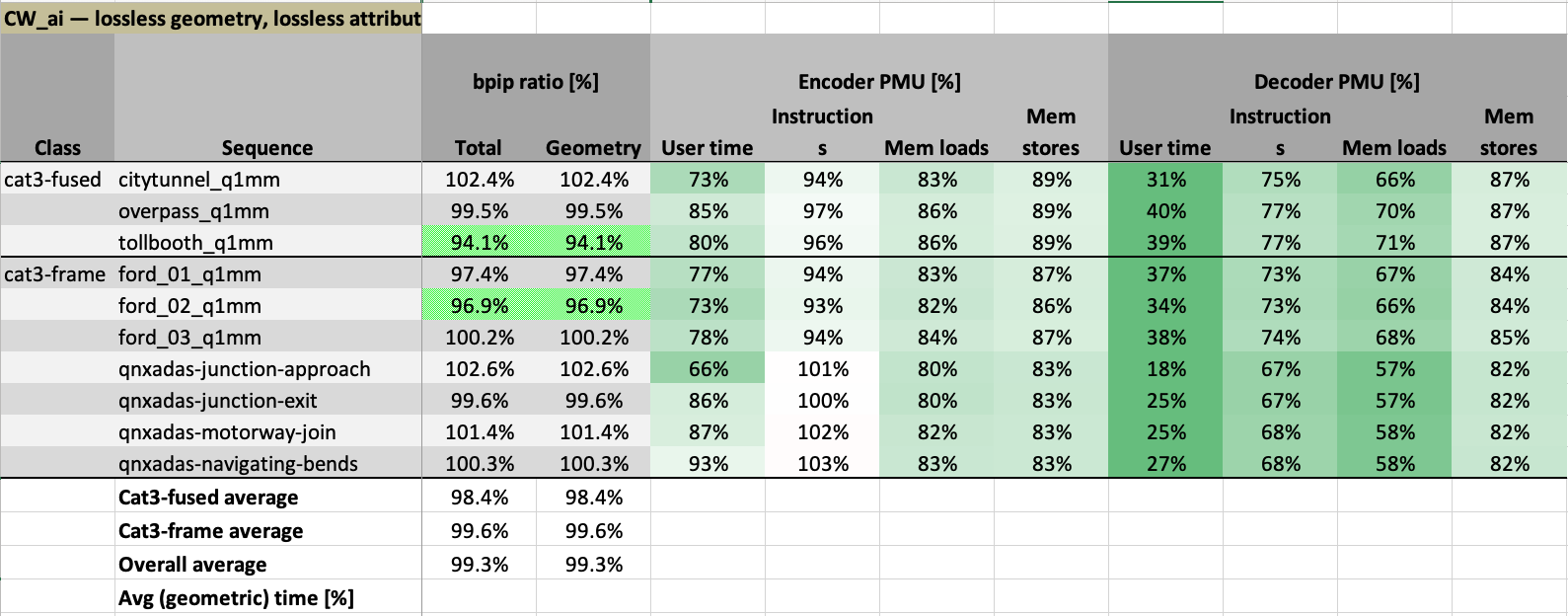


Figure 3: Low latency predictive geometry coding mode vs. TMC13v7.

# Specification change

|  |  |
| --- | --- |
| geometry\_parameter\_set( ) { | Descriptor |
| **gps\_geom\_parameter\_set\_id** | ue(v) |
| **gps\_seq\_parameter\_set\_id** | ue(v) |
| **gps\_box\_present\_flag** | u(1) |
| if( gps\_box\_present\_flag ){ |  |
| **gps\_gsh\_box\_log2\_scale\_present\_flag** | u(1) |
| if( gps\_gsh\_box\_log2\_scale\_present\_flag = = 0 ) |  |
| **gps\_gsh\_box\_log2\_scale** | ue(v) |
| } |  |
| **gps\_predictive\_mode\_enabled\_flag** | u(1) |
| if (!gps\_predictive\_mode\_enabled\_flag) { |  |
| **unique\_geometry\_points\_flag** | u(1) |
| **neighbour\_context\_restriction\_flag** | u(1) |
| **inferred\_direct\_coding\_mode\_enabled\_flag** | u(1) |
| **bitwise\_occupancy\_coding\_flag** | u(1) |
| **adjacent\_child\_contextualization\_enabled\_flag** | u(1) |
| **log2\_neighbour\_avail\_boundary** | ue(v) |
| **log2\_intra\_pred\_max\_node\_size** | ue(v) |
| **log2\_trisoup\_node\_size** | ue(v) |
| } else { |  |
| **init\_qp\_minus26** | se(v) |
| **slice\_qp\_delta\_enabled\_flag** | u(1) |
| } |  |
| **gps\_extension\_present\_flag** | u(1) |
| if( gps\_extension\_present\_flag ) |  |
| while( more\_data\_in\_byte\_stream( ) ) |  |
| **gps\_extension\_data\_flag** | u(1) |
| byte\_alignment( ) |  |
| } |  |

|  |  |
| --- | --- |
| geometry\_slice\_header( ) { | Descriptor |
| **gsh\_geometry\_parameter\_set\_id** | ue(v) |
| **gsh\_tile\_id** | ue(v) |
| **gsh\_slice\_id** | ue(v) |
| if( gps\_box\_present\_flag ) { |  |
| if( gps\_gsh\_box\_log2\_scale\_present\_flag ) |  |
| **gsh\_box\_log2\_scale** | ue(v) |
| **gsh\_box\_origin\_x** | ue(v) |
| **gsh\_box\_origin\_y** | ue(v) |
| **gsh\_box\_origin\_z** | ue(v) |
| } |  |
| if (!gps\_predictive\_mode\_enabled\_flag) { |  |
| if (!slice\_qp\_delta\_enabled\_flag) |  |
| **slice\_qp\_delta\_abs** | ue(v) |
| if (slice\_qp\_delta\_abs) |  |
| **slice\_qp\_delta\_sign** | u(1) |
| } else { |  |
| **gsh\_log2\_max\_nodesize** | ue(v) |
| } |  |
| **gsh\_num\_points** | ue(v) |
| byte\_alignment( ) |  |
| } |  |

|  |  |
| --- | --- |
| geometry\_slice\_data( ) { | Descriptor |
| If (!gps\_geometry\_prediction\_flag) { |  |
| for( i = 0; i < gsh\_num\_points; ++i) { |  |
| **children\_count** | ae(v) |
| **prediction\_mode** | ae(v) |
| for( j = 0; j < 3; ++j) { |  |
| **residual\_is\_zero** | ae(v) |
| if (!residual\_is\_zero) { |  |
| **residual\_sign** | ae(v) |
| **residual\_bitcount** | ae(v) |
| for( k = 0; k < residual\_bitcount-1; ++j) { |  |
| **residual\_bit** | ae(v) |
| } |  |
| } |  |
| } |  |
| } else { |  |
| for( depth = 0; depth < MaxGeometryOctreeDepth; depth++ ) { |  |
| for( nodeIdx = 0; nodeIdx < NumNodesAtDepth[ depth ]; nodeIdx++ ) { |  |
| xN = NodeX[ depth ][ nodeIdx ] |  |
| yN = NodeY[ depth ][ nodeIdx ] |  |
| zN = NodeZ[ depth ][ nodeIdx ] |  |
| geometry\_node( depth, nodeIdx, xN, yN, zN ) |  |
| } |  |
| } |  |
| if ( log2\_trisoup\_node\_size > 0 ) |  |
| geometry\_trisoup\_data( ) |  |
| } |  |
| } |  |

**gps\_predictive\_mode\_enabled\_flag** equal to 1 indicates that the preditive geometry mode is enabled. gps\_predictive\_mode\_enabled\_flag equal to 0 indicates that octree-based geometry coding is enabled.

**init\_qp\_minus26** plus 26 specifies the initial value of slice QP for each geometry slice referring to the GPS. The initial value of slice QP is modified at the slice layer when a non-zero value of slice\_qp\_delta is decoded. The value of init\_qp\_minus26 shall be in the range of XXX to YYYY, inclusive.

**slice\_qp\_delta\_enabled\_flag** equal to 1 specifies that slice\_qp\_delta may be present in the geometry slice header unit syntax. **slice\_qp\_delta\_enabled\_flag** equal to 0 specifies that slice\_qp\_delta is not present in the slice header unit syntax.

**slice\_qp\_delta\_abs** specifies the absolute value of the difference between the slice quantization parameter and init\_qp\_minus26.

**slice\_qp\_delta\_sign** specifies the sign of the difference between the slice quantization parameter and init\_qp\_minus26.

**children\_count** specifies the number of children the current node of the prediction tree has.

**prediction\_mode** specifies the prediction mode of the current node of the prediction tree.

**residual\_is\_zero** equal to 1 specifies that the residual is zero. residual\_is\_zero equal to 1 indicates that the residual is different from 1.

**residual\_bitcount** specifies the number of bits needed to represent the absolute value of the residual with a fixed-length representation.

**residual\_bit** specifies the k-thbit of the fixed length representation of the residual.

# Conclusion

In this contribution, we propose a predictive geometry compression scheme adapted for low latency streaming of Category 3 content. Such important functionality is currently not supported by the TMC3v7 and is crucial for several applications such as autonomous driving. Furthermore, the proposed scheme offers a very low complexity with decoding times 2.5x-5.9x times faster than those of TMC3v7, which makes well adapted for constrained environments (e.g., constrained power budget) and low-end devices. The encoder complexity could be adapted to the considered use case and could vary from 5.5x slower to 2.2x faster encoding depending on the desired compression performance. For the slow configuration, 3.5%-15% compression gain compared to TMC3v7 were observed. The fast configuration offered -2.5%-5.8% compared to TMC3v7.

# References

1. “Common Test Conditions for PCC” ISO/IEC JTC1/SC29 WG11 MPEG2019 Doc. N18474, Geneva, CH, March 2019.