**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/m51010**

**October 2019, Geneva, CH**

|  |  |
| --- | --- |
| **Source** | **Apple Inc.** |
| **Status** | **Input document** |
| **Title** | **[G-PCC][New proposal] Improved implementation of the Prediction and Lifting schemes.** |
| **Author** | **Zhenzhen Gao, David Flynn, Alexis Tourapis, and Khaled Mammou** |

# Abstract

The lifting and prediction scheme extensively use nearest neighbor searches during the Level of Detail (LoD) generation and predictors building stages. Poor implementation of these two processes significantly impact the encoding and decoding complexity of the G-PCC reference software. In this contribution we highlight a few issues in the current implementation and propose an improved implementation. The proposed implementation offers similar RD performance when compared to TMC13v7, with 4%-17% reduction in terms of attributes encoding/decoding time, number of instructions and memory loads/stores.

# Existing Implementation

The current TMC13v7 implementation suffers from the following limitations:

* The PCCPredictor:: insertNeighbor() does not handle insertion of neighbors within the same distance of the current point correctly.

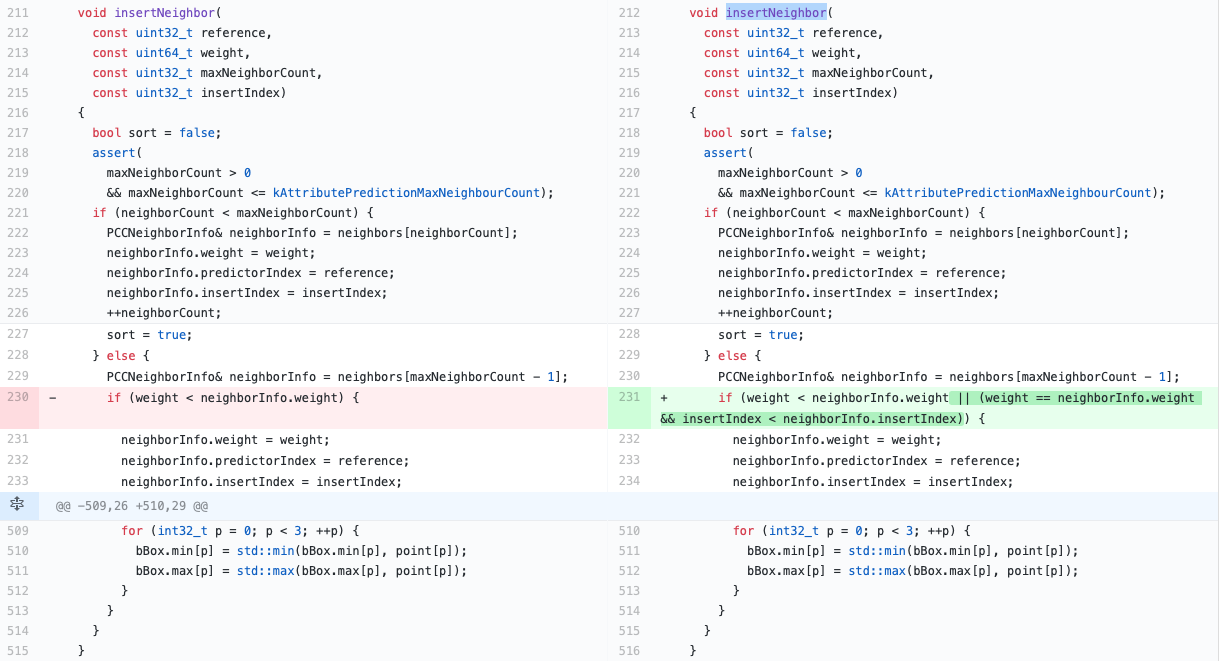


Figure 1. PCCPredictor:: insertNeighbor() has issues with neighbor’s insertion.

* The function computeNearestNeighbors() does not handle neighbors within the same distance of the current point correctly.



Figure 2. computeNearestNeighbors() has issues with neighbor’s insertion.

* In the function buildPredictorsFast() and its subroutines, unnecessary cast operations from Vec3<double> to Vec3<int32\_t> are significantly impacting the overall codec performance.

# Proposed Implementation

We propose the following improvements:

* Simplify how neighbors at the same distance are handled
  + Neighbors in subsequent LODs with the same distance from the current point are handled according to the priority described in Figure 2.

Figure 2. Neighbors in subsequent LoDs: Handling neighbors with the same distance from the current point.

* + Neighbors in the same LOD Neighbors with the same distance from the current point are handled according to the priority described in Figure 3.

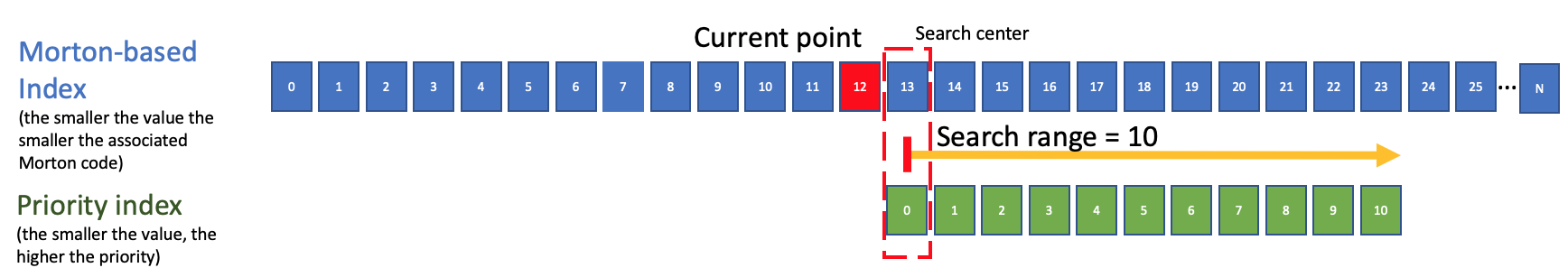


Figure 3. Neighbors in the same LoD: Handling neighbors with the same distance from the current point.

* + Neighbors in the subsequent LoDs have a higher priority than neighbors in the same LoD.
* Avoid cast operations by storing the coordinates as integers instead of doubles.

# Experimental results

The proposed implementation was compared to the TMC13v7 anchor under the CTC conditions. Figure 4 shows that the proposed implementation offers similar RD performance, while reducing the total encoding/decoding times by 4-17% on average, depending on the test condition. When considering only the attributes encoding/decoding times, the reduction is 25%-40% on average. Figure 5 provides a more detailed statistics in terms of number of instructions and memory loads/stores. According to all the reported metrics, the proposed implementation outperforms TMC13v7.

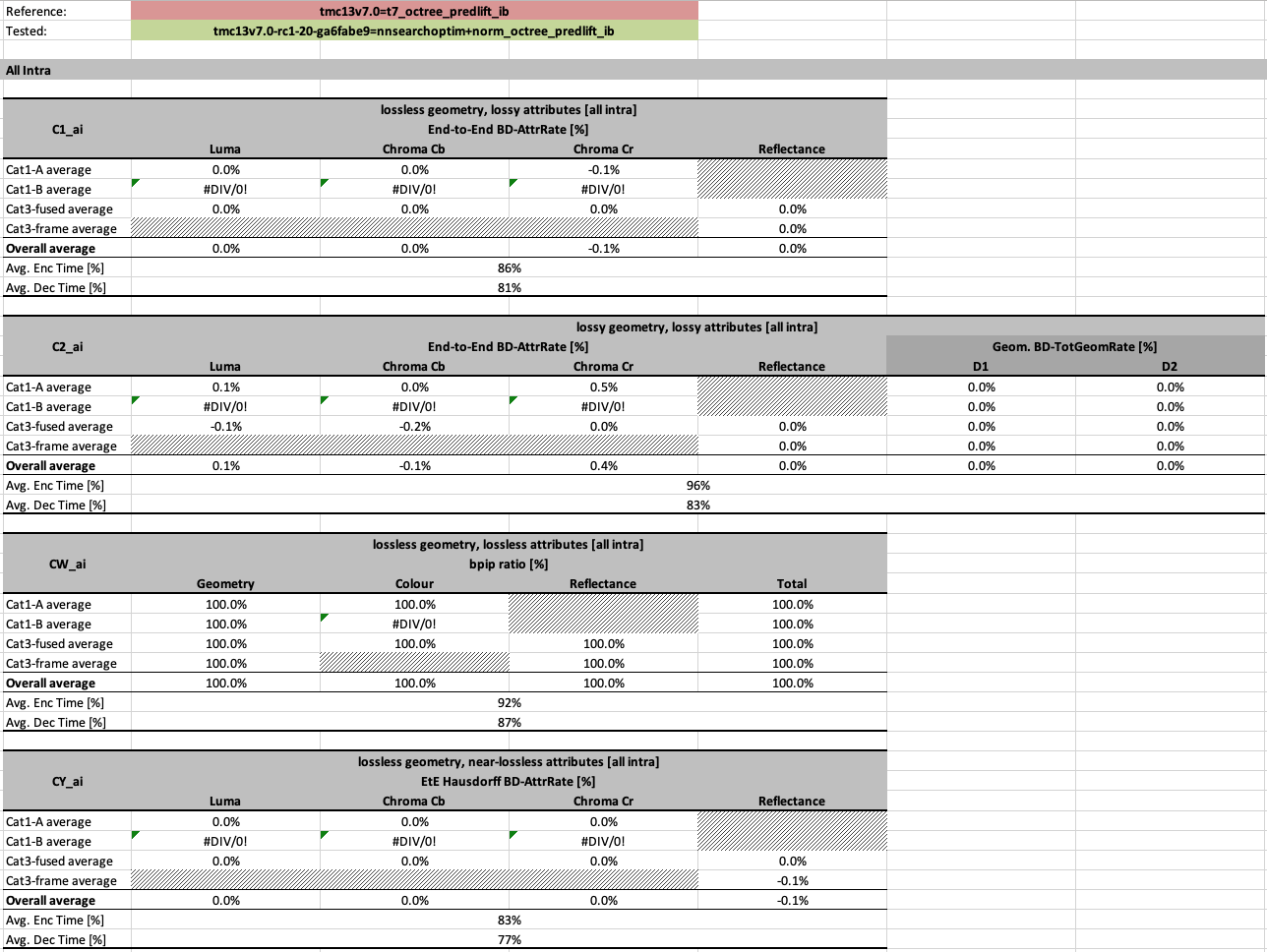


Figure 4. Proposed vs. TMC13v7 anchor: similar RD performance with 4%-17% gain in terms of total encoding/decoding time.

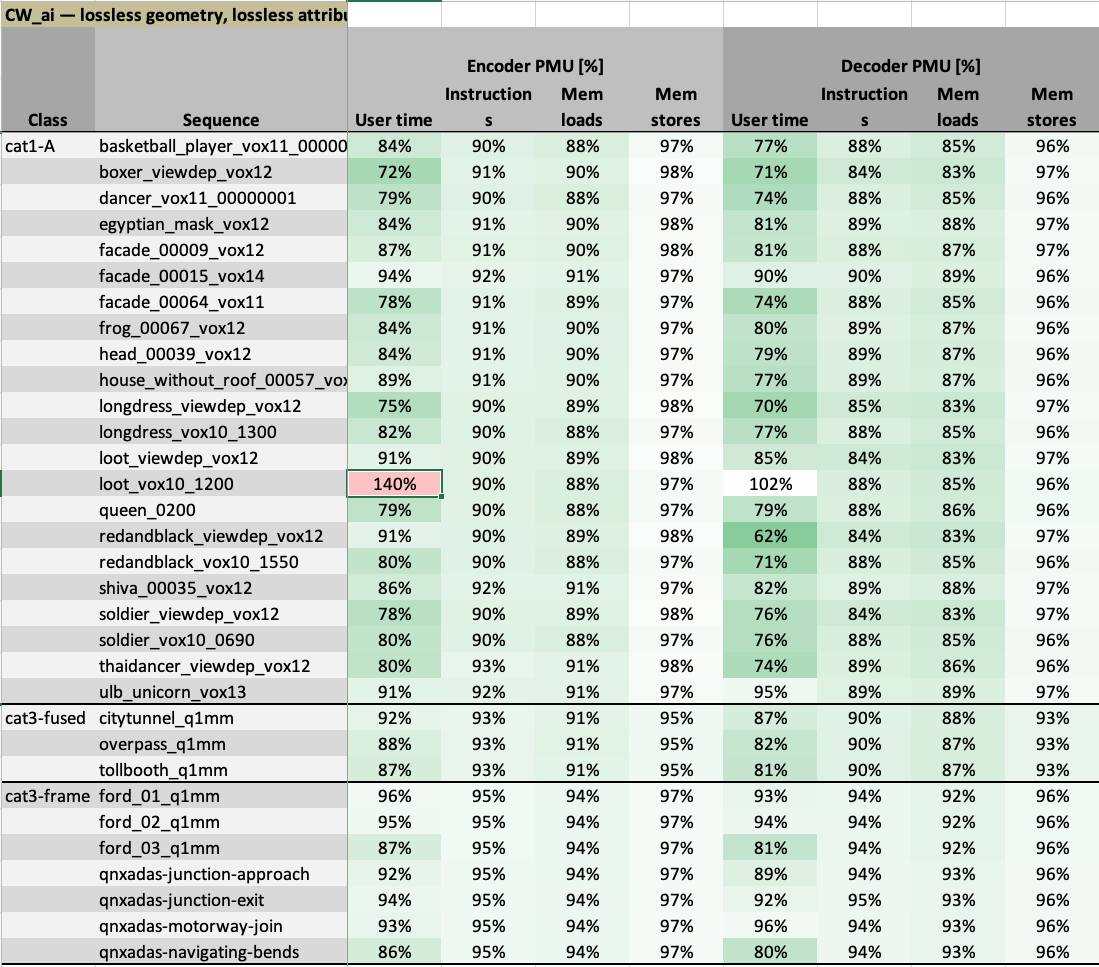


Figure 5. Proposed vs. TMC13v7 anchor: 3%-38% average reduction in terms of number of instructions and memory loads/stores.

# Specification change

In Section 8.3.22, replace the following

|  |
| --- |
| for (i = startIndex; i < endIndex; i++) {  currentIndex = assignedPointIndexes[i];  currentMortonCode = McodeUnsorted [currentIndex];  currentPos = PointPos[currentIndex];  while (j < nonAssignedPointCount &&  currentMortonCode >= McodeUnsorted[nonAssignedPointIndexes[j]) j++;  j = min(nonAssignedPointCount − 1, j);  j0 = max(0, j − searchRange);  j1 = min(nonAssignedPointCount, j + searchRange + 1);  neighboursCount[ currentIndex ] = 0;  for (k = j0; k < j1 ; k++) {  neighbourIndex = nonAssignedPointIndex[k];  neighbourPos = PointPos[neighbourIndex];  d2 = (currentPos[0] – neighbourPos[ 0 ])2 + (currentPos[1] – neighbourPos[1]) 2 +  (currentPos[2] – neighbourPos[2]) 2;  insertIndex = ( k − j > 0 ) ? ( ( ( k – j ) << 1 ) – 1 ) : ( ( j – k ) << 1 );  If (neighboursCount[ currentIndex ] < numPredNearestNeighbours) {  p = neighboursCount[ currentIndex ];  neighbours[ currentIndex ][ p ] = neighbourIndex;  neighboursDistance2[ currentIndex ][ p ] = d2;  neighboursInsertIndex[ currentIndex ][ p ] = insertIndex;  neighboursCount[ currentIndex ]++;  sortNeighbours(neighboursCount[ currentIndex ], neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ] ,  neighboursInsertIndex[ currentIndex ]);  } else if (d2 < neighboursDistance2[ currentIndex ] [ numPredNearestNeighbours−1) {  neighbours[ currentIndex ] [ numPredNearestNeighbours−1 = neighbourIndex;  neighboursDistance2[ currentIndex ] [ numPredNearestNeighbours−1 = d2;  neighboursInsertIndex[ currentIndex ][ numPredNearestNeighbours − 1] = insertIndex;  sortNeighbours(numPredNearestNeighbours, neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ] ,  neighboursInsertIndex[ currentIndex ]);  }  }  if (currentLayer < liftingIntraLodPredictionNumLayers) {  j1 = min(endIndex, k + searchRange);  for ( k = i + 1; k < j1; k++) {  neighbourIndex = assignedPointIndex[k];  neighbourPos = PointPos[neighbourIndex];  d2 = (currentPos[0] – neighbourPos[ 0 ])2 + (currentPos[1] – neighbourPos[1]) 2 +  (currentPos[2] – neighbourPos[2]) 2;  insertIndex = ((k – i) << 1) – 1;  If (neighboursCount[ currentIndex ] < numPredNearestNeighbours) {  p = neighboursCount[ currentIndex ];  neighbours[ currentIndex ][ p ] = neighbourIndex;  neighboursDistance2[ currentIndex ][ p ] = d2;  neighboursInsertIndex[ currentIndex ][ p ] = insertIndex;  neighboursCount[ currentIndex ]++;  sortNeighbours(neighboursCount[ currentIndex ],  neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ] ,  neighboursInsertIndex[ currentIndex ]);  } else if (d2 < neighboursDistance2[ currentIndex ][ numPredNearestNeighbours – 1]) {  neighbours[ currentIndex ][ numPredNearestNeighbours – 1 ] = neighbourIndex;  neighboursDistance2[ currentIndex ][ numPredNearestNeighbours – 1 ] = d2;  neighboursInsertIndex[ currentIndex ][ numPredNearestNeighbours − 1] = insertIndex;  sortNeighbours(numPredNearestNeighbours, neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ],  neighboursInsertIndex[ currentIndex ]);  }  }  }  } |

With

|  |
| --- |
| for (i = startIndex; i < endIndex; i++) {  currentIndex = assignedPointIndexes[i];  currentMortonCode = McodeUnsorted [currentIndex];  currentPos = PointPos[currentIndex];  while (j < nonAssignedPointCount &&  currentMortonCode >= McodeUnsorted[nonAssignedPointIndexes[j]) j++;  j = min(nonAssignedPointCount − 1, j);  j0 = max(0, j − searchRange);  j1 = min(nonAssignedPointCount, j + searchRange + 1);  neighboursCount[ currentIndex ] = 0;  for ( k = j0; k < j1 ; k++) {  neighbourIndex = nonAssignedPointIndex[k];  neighbourPos = PointPos[neighbourIndex];  d2 = abs(currentPos[0] – neighbourPos[ 0 ]) + abs(currentPos[1] – neighbourPos[1]) +  abs(currentPos[2] – neighbourPos[2]);  if (abs(k-j) <= 3) {  insertIndex = ( k − j > 0 ) ? ( ( ( k – j ) << 1 ) – 1 ) : ( ( j – k ) << 1 );  } else if (k > j) {  insertIndex = 7 + k – j;  } else {  insertIndex = searchRange + 4 + j - k;  }  If (neighboursCount[ currentIndex ] < numPredNearestNeighbours) {  p = neighboursCount[ currentIndex ];  neighbours[ currentIndex ][ p ] = neighbourIndex;  neighboursDistance2[ currentIndex ][ p ] = d2;  neighboursInsertIndex[ currentIndex ][ p ] = insertIndex;  neighboursCount[ currentIndex ]++;  sortNeighbours(neighboursCount[ currentIndex ], neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ] ,  neighboursInsertIndex[ currentIndex ]);  } else if (d2 < neighboursDistance2[ currentIndex ] [ numPredNearestNeighbours−1) {  neighbours[ currentIndex ] [ numPredNearestNeighbours−1 = neighbourIndex;  neighboursDistance2[ currentIndex ] [ numPredNearestNeighbours−1 = d2;  neighboursInsertIndex[ currentIndex ][ numPredNearestNeighbours − 1] = insertIndex;  sortNeighbours(numPredNearestNeighbours, neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ] ,  neighboursInsertIndex[ currentIndex ]);  }  }  if (currentLayer < liftingIntraLodPredictionNumLayers) {  j1 = min(endIndex, k + searchRange);  for ( k = i + 1; k < j1; k++) {  neighbourIndex = assignedPointIndex[k];  neighbourPos = PointPos[neighbourIndex];  d2 = abs(currentPos[0] – neighbourPos[ 0 ]) + abs(currentPos[1] – neighbourPos[1]) +  abs(currentPos[2] – neighbourPos[2]);  insertIndex = 2 \* searchRange + (k – i);  If (neighboursCount[ currentIndex ] < numPredNearestNeighbours) {  p = neighboursCount[ currentIndex ];  neighbours[ currentIndex ][ p ] = neighbourIndex;  neighboursDistance2[ currentIndex ][ p ] = d2;  neighboursInsertIndex[ currentIndex ][ p ] = insertIndex;  neighboursCount[ currentIndex ]++;  sortNeighbours(neighboursCount[ currentIndex ],  neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ] ,  neighboursInsertIndex[ currentIndex ]);  } else if (d2 < neighboursDistance2[ currentIndex ][ numPredNearestNeighbours – 1]) {  neighbours[ currentIndex ][ numPredNearestNeighbours – 1 ] = neighbourIndex;  neighboursDistance2[ currentIndex ][ numPredNearestNeighbours – 1 ] = d2;  neighboursInsertIndex[ currentIndex ][ numPredNearestNeighbours − 1] = insertIndex;  sortNeighbours(numPredNearestNeighbours, neighbours[ currentIndex ],  neighboursDistance2[ currentIndex ],  neighboursInsertIndex[ currentIndex ]);  }  }  }  } |

# Conclusion

In this contribution, we proposed improved implementation of the prediction/lifting scheme. The proposed implementation offers similar RD performance when compared to TMC13v7, with 3%-38% reduction in terms of total encoding/decoding time, number of instructions and memory loads/stores.

# References

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