INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 7
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 7 m 56635
Online – April 2021

Source KETI and Sungkyunkwan University
Title [V-PCC] Fast Grid-based Refining Segmentation Method in V-PCC
Author Jieon Kim, Yong-Hwan Kim, Eun-Seok Ryu

1. Abstract

This contribution proposes in the first place a fast grid-based refining segmentation (G-RS) method and provides a software optimization report on TMC2 v12.0 in the second part. The average saved time only considering the self-time of G-RS is 41%, and considering the self-time of V-PCC, the saved time is 28%.

2. Proposal

2.1. Encoding time-complexity in the patch generation

The average time consumption ratio of each procedure in the patch generation is presented in Table 1. Since TMC2 v12.0 includes the latest G-RS optimization method [1], we experimented with TMC2 v12.0. G-RS [2] accounts for the most execution time in the patch generation process: an average of 62.90% for the Class A test sequences, 32.3% on average for the Class B sequence, and 18.0% or more on average for the Class C sequences. There is room for further complexity reduction in G-RS, in particular, for the Class A sequences.

Table 1. Time consumption ratio of each procedure in the patch generation process (TMC2 12.0, 32frames, RA configuration)

Class	Test point cloud	Estimate Normals	Grid-based Refinement	Segment Patches
A	loot	24.95	**63.08**	11.92
	redandblack	23.44	**63.34**	13.17
	soldier	24.06	**62.13**	13.76
	queen	22.63	**63.37**	13.95
B	longdress	43.12	32.33	24.46
C	basketball_player	48.43	17.56	33.92
	dancer_player	50.77	18.58	30.56

2.1 FGM: Fast G-RS Method

As observed earlier, G-RS is one of the complex processes in the patch generation process. Each point is associated with one of the projection planes based on its normal vector. Then, the initial projection plane index (PPI) is refined iteratively. The conventional refining steps in V-PCC [3] are applied concerning all 3D points in the point cloud despite that most 3D points are likely to have accurate initial PPI. We propose a fast G-RS that selects voxels that need G-RS in V-PCC.

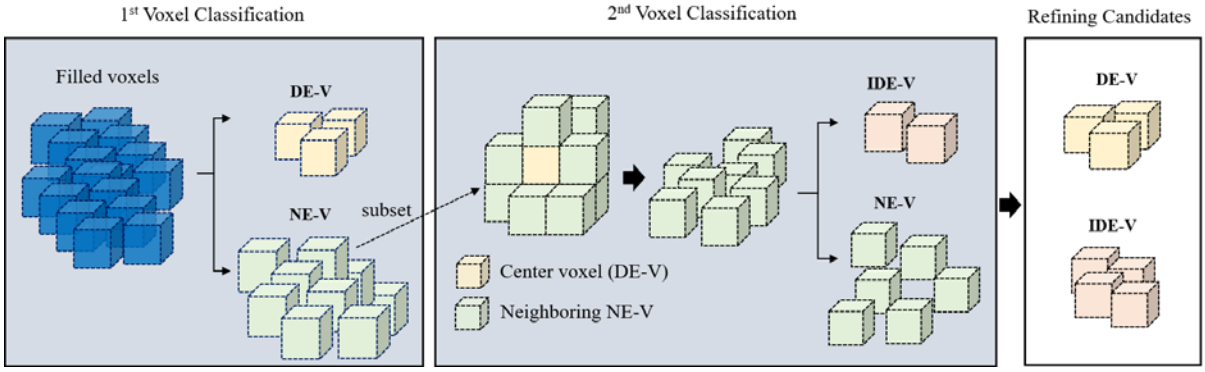

Figure 1. Proposed voxel classification scheme.

2.1.1 Overall Proposed G-RS Scheme

Figure 1 shows the proposed voxel classification scheme. The proposed method first classifies a voxel as either by direct edge-voxel (DE-V) or no edge-voxel (NE-V) according to *uniform index*. Later, the neighboring voxels labeled with NE-V of the DE-V are classified as either NE-V or indirect edge-voxel (IDE-V) according to *extended uniformity index*. Based on the voxel classification, the refining steps are only limited to DE-V and IDE-V.

2.1.2 Voxel Classification Methods

G-RS is post-processing of initial segmentation to smooth out wrongly associated PPI to obtain uniform regions for more accurate patch segmentation. Because the PPI accuracy of a 3D point is measured based on its neighbors, voxels with non-uniform PPI distribution are the candidates to be considered by G-RS.

We distinguish three types of voxels as below:

a) Voxels with PPI variations within their voxel. They are denoted by DE-V and can be either a single-point (S-DE) or multi-points (M-DE) in a voxel. The former is usually isolated points, and the latter indicates the presence of a point cloud surface.
b) Voxels with uniform PPI distribution within their voxel but non-uniform compared to the nearest-neighbors. They are clustered with discontinuity PPI from their neighbors and denoted by IDE-V.
c) Voxels with uniform PPI distribution within their voxel and the nearest-neighbors. These voxels are considered as uniform regions and they are denoted by NE-V.

***Uniformity index* in the 1st voxel classification:**

Uniformity index U_i of a filled voxel V_i is defined using the PPI score S_i as a histogram as follows:

$$U_i = \frac{S_i(p_i^*)}{|S_i|}, \text{where } p_i^* = \underset{p \in K}{\operatorname{argmax}} S_i(p),$$
$$p \text{ and } K \text{ denotes PPI and the number of projection planes, respectively}$$

, and each voxel is classified as below:

$$V_i \in \begin{cases} \text{DE-V}, & U_i \neq 1 \\ \text{NE-V}, & U_i = 1 \end{cases}.$$

***Extended uniformity index* in the 2nd voxel classification:**

This applies to NE-V. The reference range of a histogram for *extended uniformity index* is extended from within a voxel to its neighbors. The smooth score $\bar{S}_i$ is used. Similar to uniform index, we can estimate *extended uniformity index* $\widehat{U}_i$ within the selected reference range as follows:

$$\widehat{U}_i = \frac{S_i(\bar{p}_i^*)}{|S_i|}, \text{where } \bar{p}_i^* = \underset{p \in K}{\operatorname{argmax}} \bar{S}_i(p)$$

Because NE-V constitutes most of point cloud, the amount of necessary computation and memory access for $\overline{S_i}$ of every NE-V is considerable. Instead, the neighboring (V_j) of V_i is examined in the refining steps of V_i. $\widehat{U}_j$ is defined as below:

$$\widehat{U}_j = \frac{S_j(\bar{p}_i^*)}{|S_j|}, \text{where } \bar{p}_i^* = \underset{p \in K}{\operatorname{argmax}} \bar{S}_i(p)$$

, and $\widehat{U}_j$ can be further simplified by just comparing the dominant PPI in $\bar{S}_i$ $(\bar{p}_i^*)$ and S_j (p_j^*).

NE-V that is close neighbor of V_i is classified as below:

$$V_j \in \begin{cases} \text{NE-V}, & U_j = 1, \widehat{U}_j = 1 \\ \text{IDE-V}, & U_j = 1, \widehat{U}_j \neq 1 \end{cases}.$$

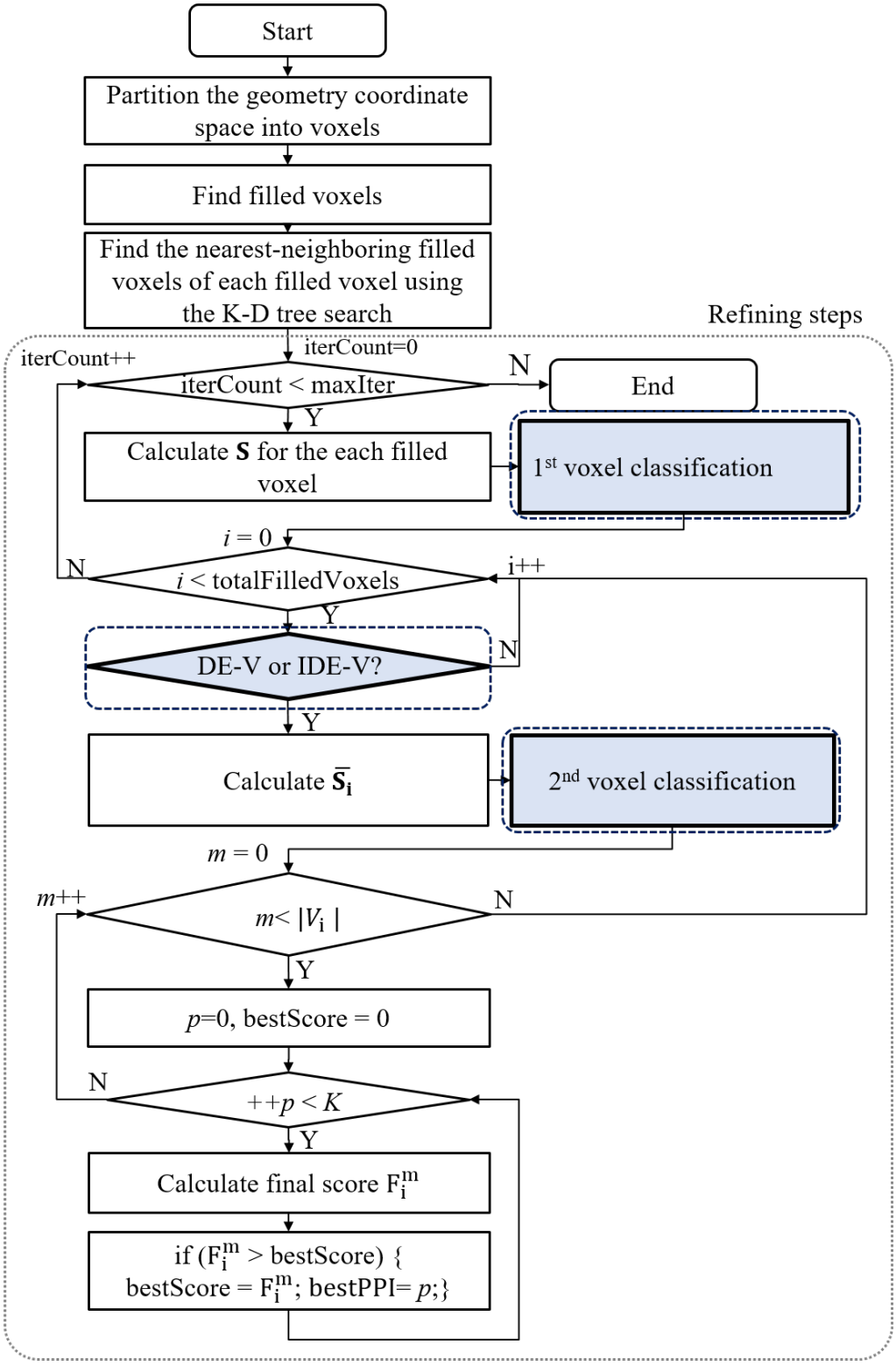

Figure 2. Flowchart of the proposed grid-based refining segmentation procedure in TMC2.

Figure 2 shows a flowchart of the proposed algorithm, and the additional steps by the proposed algorithm are represented with a dotted line.

2.2 Method2: Software Optimizations to TMC2 v12.0

The unordered map uses a hash function to map its elements, which in itself is expensive. Here are places where there is a call to the unordered map in G-RS:

Original code in TMC2 v12.0

```
std::unordered_map<size_t,PointIndicesOfGridCell> gridWithPointIndices;
std::unordered_map<size_t, ScoreSmoothOfPointsInGridCell> gridWithScores;

for (size_t i = 0; i < gridCenters.getPointCount(); ++i) {
    size_t nnPointCount = 0;
    auto&  currentAdjOfI = adj[i];
    auto   iter = currentAdjOfI.begin();

    for (; iter != currentAdjOfI.end(); ++iter) {
      auto& pos = gridCenters[*iter];
      nnPointCount += gridWithPointIndices[subToInd(pos[0], pos[1], pos[2])].pointIndices->size();
      if (nnPointCount >= maxNNCount) { break; }
    }

    weights.push_back(lambda / nnPointCount);

    if (iter != currentAdjOfI.end()) { currentAdjOfI.erase(iter + 1, currentAdjOfI.end()); }

    std::sort(currentAdjOfI.begin(), currentAdjOfI.end());

    std::vector<size_t*> tempScoreSmoothFromAdjsOfI;
    tempScoreSmoothFromAdjsOfI.reserve(currentAdjOfI.size());
    for (const auto& j : currentAdjOfI) {
      const auto& pos = gridCenters[j];
      auto&       vox = *(gridWithScores[subToInd(pos[0], pos[1], pos[2])].scoreSmooth);
      tempScoreSmoothFromAdjsOfI.push_back(vox.data());
    }

   scoreSmoothFromAdjsOf.emplace_back(std::move(tempScoreSmoothFromAdjsOfI);

    const auto& pos = gridCenters[i];
    auto&           uniq  =  gridWithPointIndices[subToInd(pos[0],
pos[1], pos[2])].pointIndices;
    pointIndicesOfI.push_back(uniq.get());
  }
```

The number of hash map access = **totalNumOfFilledVoxels* numOfNeighborVoxls**

The number of hash map access = **totalNumOfFilledVoxels* numOfNeighborVoxls**

The number of hash map access = **totalNumOfFilledVoxels**

We have added a pre-processing step to the source code to avoid the remaining hash map accesses. The idea is simple and straightforward. The located addresses of two Class objects, *i.e.*, their objects pointers, in the grid is constant. For this reason, we have related the two hash maps with two vectors that pointer objects that having data type of each Class object as follows:

Proposed pre-processing code

```
const uint64_t uiTotalNumOfVoxs = gridCenters.getPointCount();

std::vector<PointIndicesOfGridCell*>                pointIndicesOfVox;
pointIndicesOfVox.reserve(uiTotalNumOfVoxs);

std::vector<ScoreSmoothOfPointsInGridCell*>    scoreSmoothOfVox;
scoreSmoothOfVox.reserve(uiTotalNumOfVoxs);

for (size_t i = 0; i < uiTotalNumOfVoxs; ++i) {
  auto& p = gridCenters[i];

  PointIndicesOfGridCell* pPI = gridWithPointIndices[subToInd(p[0], p[1], p[2])].get();
  ScoreSmoothOfPointsInGridCell* pAttr = gridWithScores[subToInd(p[0], p[1], p[2])].get();

  pointIndicesOfVox.push_back(pPI);
  scoreSmoothOfVox.push_back(pAttr);
}
```

The number of hash map access = **totalNumOfFilledVoxels**

Optimized code in TMC2 v12.0

```
for (size_t i = 0; i < uiTotalNumOfVoxs; ++i) {
    size_t nnPointCount = 0;
    auto&  currentAdjOfI = adj[i];
    auto   iter = currentAdjOfI.begin();
    for (; iter != currentAdjOfI.end(); ++iter) {
      nnPointCount += pointIndicesOfVox[*iter]->getPointCount();
      if (nnPointCount >= maxNNCount) { break; }
   }

  weights.push_back(lambda / nnPointCount);

  if (iter != currentAdjOfI.end()) {
    currentAdjOfI.erase(iter + 1, currentAdjOfI.end()); }
  }
```

No hash map access!

Table 2. A comparison of the total number of hash map access in G-RS in which A represents the total number of filled voxels and B denotes the number of neighboring voxels of each filled voxel

	Original TMC2 v12.0	**Proposed Method 2**
Total number of hash map access	A × (2B+1)	2A

3. Experimental Results

The proposed method was implemented in the V-PCC reference software TMC2 12.0. We evaluated the performance of the proposed method under the lossy-AI and lossy-RA configurations. The self-time of V-PCC was measured by subtracting HEVC encoding time and color conversion time from the total encoding time. A summary of the BDBR performance and time-complexity reductions is shown below.

There is one parameter in our implementation:

- r_β: the reference range for IDE-V

, and r_β for IDE-V is set to 2 for the Class A test point cloud and 1 for the rest of test point cloud sequences. The experimental results with different r_β are included in the attached spreadsheets.

AI

A. BDBR performance

Point cloud	Geom. BD TotGeomRate		End-to-End BD AttrRate			- Geom. BD TotalRate		End-to-End BD TotalRate		
	D1	D2	Luma	Cb	Cr	D1	D2	Luma	Cb	Cr
loot	0.17%	0.18%	-0.07%	-0.49%	-0.08%	0.21%	0.22%	-0.02%	-0.32%	-0.02%
redandblack	0.01%	0.00%	0.01%	-0.06%	0.07%	-0.05%	-0.05%	0.03%	-0.03%	0.07%
soldier	0.12%	0.14%	0.03%	0.01%	0.38%	0.04%	0.09%	0.06%	0.05%	0.32%
queen	-0.01%	-0.03%	0.14%	0.04%	0.20%	0.00%	-0.02%	0.08%	0.03%	0.15%
longdress	-0.02%	0.00%	-0.01%	-0.05%	0.00%	-0.04%	-0.01%	0.00%	-0.04%	0.00%
basketball_player	0.08%	0.10%	-0.03%	0.09%	-0.06%	0.09%	0.14%	-0.01%	0.08%	-0.03%
dancer_player	-0.01%	0.00%	0.01%	0.32%	-0.46%	-0.03%	-0.01%	0.01%	0.24%	-0.35%
Overall average	0.05%	0.06%	0.01%	-0.02%	0.01%	0.03%	0.05%	0.02%	0.00%	0.02%

B. Time-complexity reduction ratio

Point cloud	Self-time of VPCC			Self-time of G-RS		
	Overall	Method 2	FGM	Overall	Method 2	FGM
loot	37.9%	15.5%	22.4%	41.3%	22.5%	18.8%
redandblack	34.8%	16.5%	18.2%	40.7%	23.4%	17.3%
soldier	35.7%	15.6%	20.1%	39.6%	22.7%	16.9%
queen	34.4%	8.9%	25.5%	35.4%	13.9%	21.5%
longdress	23.8%	2.6%	21.2%	38.3%	9.0%	29.2%
basketball_player	17.6%	2.6%	15.0%	39.8%	16.5%	23.3%
dancer_player	16.9%	2.5%	14.4%	37.5%	15.4%	22.2%
Overall average	28.7%	9.2%	19.5%	38.9%	17.6%	21.3%

Encoding time reduction of FGM to the original G-RS: 21%

Encoding time reduction of Method 2 to the original G-RS: 18%

Encoding time reduction of the proposed method (FGM+ Method 2) to the original G-RS: **39%**

Encoding time reduction of the proposed method to the original self-time of V-PCC: **28%**

RA

A. BDBR performance

Point cloud	Geom. BD TotGeomRate		End-to-End BD AttrRate			Geom. BD TotalRate		End-to-End BD TotalRate		
	D1	D2	Luma	Cb	Cr	D1	D2	Luma	Cb	Cr
loot	0.08%	0.03%	0.06%	-0.47%	0.50%	0.08%	0.01%	0.06%	-0.28%	0.13%
redandblack	0.03%	0.11%	0.24%	0.55%	0.27%	0.10%	0.20%	0.13%	0.33%	0.12%
soldier	0.29%	0.33%	0.35%	0.43%	-1.07%	0.30%	0.36%	0.31%	0.29%	-0.47%
queen	0.13%	0.13%	-0.48%	1.11%	-0.03%	0.14%	0.16%	-0.25%	0.61%	0.02%
longdress	0.05%	0.07%	0.08%	-0.16%	0.07%	0.11%	0.12%	0.06%	-0.12%	0.05%
basketball_player	0.08%	0.09%	-0.07%	-2.16%	-2.48%	0.06%	0.06%	0.00%	-1.21%	-1.48%
dancer_player	-0.05%	-0.05%	0.32%	0.59%	-0.36%	0.04%	0.05%	0.12%	0.17%	-0.28%
Overall average	0.09%	0.10%	0.07%	-0.01%	-0.44%	0.12%	0.14%	0.06%	-0.03%	-0.27%

B. Time-complexity reduction ratio

Point cloud	Self-time of VPCC			Self-time of G-RS		
	Overall	Method 2	FGM	Overall	Method 2	FGM
loot	31.2%	14.6%	16.5%	37.5%	21.5%	16.0%
redandblack	36.3%	15.3%	20.9%	38.7%	22.1%	16.6%
soldier	31.3%	15.1%	16.2%	36.7%	22.1%	14.6%
queen	33.7%	9.6%	24.1%	35.5%	14.8%	20.7%
longdress	30.1%	4.2%	25.9%	55.0%	14.0%	41.1%
basketball_player	16.4%	2.9%	13.5%	45.1%	17.9%	27.2%
dancer_player	16.2%	2.8%	13.4%	43.5%	16.9%	26.6%
Overall average	27.9%	9.2%	18.7%	41.7%	18.5%	23.3%

Encoding time reduction of FGM to the original G-RS: 23%

Encoding time reduction of Method 2 to the original G-RS: 19%

Encoding time reduction of the proposed method (FGM+ Method 2) to the original G-RS: **42%**

Encoding time reduction of the proposed method to the original self-time of V-PCC: **28%**

4. Conclusion

This document presents a fast G-RS method (FGM) in V-PCC. Voxels with the uniform PPI distribution are unlikely to be affected by G-RS. For this reason, we have introduced the *uniformity index* of a voxel in the proposed method. The uniformity is examined within a voxel or predefined search range. The proposed method skips the refining steps for voxels of which PPI distribution is uniform. Together with FGM, a new implementation approach (Method 2) to G-RS was presented. Method 2 provides bit-exact matches compared to the original TMC2 v12.0. Experimental results demonstrate that the proposed fast algorithms (FGM + Method 2) can reduce on average 42% and 39%% of the self-time of G-RS in the TMC2 12.0 reference software without RD coding loss under the RA and AI conditions, respectively.

References

[1] “Cache-friendly Refine Segmentation for Tmc2 Speed-up”, ISO/IEC JTC1/SC29/WG7 Doc. m55143, Online, October 2020.
[2] “Gridbased partitioning”, ISO/IEC JTC1/SC29 WG11 Doc. m47600, Geneva, March 2019.
[3] “V-PCC Codec Description”, ISO/IEC JTC 1/SC29/WG11 N19092, Online, October 2020.