**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/m50848**

**October 2019, Geneva, CH**

|  |  |
| --- | --- |
| **Source** | **InterDigital** |
| **Status** | **Input Contribution** |
| **Title** | **[VPCC][CE] CE2.17 Patch Border Filtering Report** |
| **Author** | Julien Ricard, Céline Guede, Pierre Andrivon |

# Introduction

This document presents the results of CE2.17 on Patch Border Filtering (PBF) proposal [m47479].

The CE mandates are as follows:

1. Anchor: TMC2v7 Anchor, including tool B (**m43501**)
2. PBF: TMC2v7 with tool A (**m47479**) enabled and tool B disabled
3. WS: TMC2v7 Anchor without geometry smoothing (i.e., tool B disabled)

The PBF process is described in the following contribution: m47479 [VPCC][New proposal] Patch border filtering, Geneva, Switzerland, March 2019, ISO/IEC JTC1/SC29 WG11.

# Results

BD-Rate is obtained with the implementation of PBF on top of V-PCC V7.0 (see[w18666] ) using CTC (see [w18665]).

This proposition is applicable to the category 2 test model of the MPEG 3DG Ad Hoc on Point Cloud Compression for lossy compression.

Results for ALL frames are provided, comparing:

1. PBF [2] versus the TMC2 as anchor [1].
2. TMC2 without grid smoothing [3] versus the TMC2 as anchor [1].

## PBF [2] versus the TMC2 as anchor [1]

Anchor: Tm2 r7.0

Tested: Tm2 r7.0 with PBF

### Metrics



PBF has similar gains for all intra and random access:

* AI 🡪 1.5% in D1, 6.4% in D2. For attributes, the gain is 0.9% in luma.
* RA 🡪 2% in D1, 6.4% in D2. For attributes, the gain is 0.8% in luma

### Times



In term of self-run time, PBF is similar on encoder side and 24% less complex on the decoder side.

### Memory usage



In term of memory consumption, PBF takes more memory on encoder side (8%) and less on decoder side (10%).

### Hardware Performance

The Papi (Performance API) version 5.7.0 tool has been used to get hardware performance counters on Anchor and PBF.

The code has been instrumented with the Papi tool to track the smoothing process only. This code has been delivered to cross-checkers.



The following counters have been tracked:

* Instruction counting:

1. PAPI\_TOT\_INS Instructions completed
2. PAPI\_TOT\_CYC Total cycles

* Cache access

1. PAPI\_L2\_TCA L2 total cache accesses
2. PAPI\_L3\_TCA L3 total cache accesses
3. PAPI\_L1\_DCM L1 data cache misses
4. PAPI\_L2\_DCM L2 data cache misses

For all counters, PBF performs better than the anchor.

## TMC2 without grid smoothing [3] versus the TMC2 as anchor [1]

Anchor : Tm2 r7.0

Tested : Tm2 r7.0 without smoothing

### Metrics



For both all intra and random access, TMC2 without smoothing has poor performance.

### Times



TMC2 without grid smoothing is 22% less time consuming.

### Memory usage



TMC2 without grid smoothing takes less memory than TMC2.

# Proposed specification

Here is a suggested specification, starting from DIS V-PCC\_SCD\_d40.docx with the description of the PBF process.

## Boundary points and neighboring patches identification

This process is invoked before the patch border filtering process.

Inputs to this process are:

* Atlas coordinates (x, y).
* Patch coordinates (u, v).
* Index of the patch, p.
* gFrame[ 0 ][ y ][ x ][0], first component of the decoded geometry frames at nominal resolution, where  y = 0..asps\_frame\_height – 1, x = 0.. asps\_frame\_width – 1.
* oFrame[ y ][x][0], first component of the occupancy map corresponding to the current point cloud frame at nominal resolution, where y = 0..asps\_frame\_height – 1, x = 0..asps\_frame\_width – 1.

Outputs to this process are:

* dilatedGeoFrame[p][x][y][0], first component of the dilated geometry frames at nominal resolution for a patch of index p, where  y = 0.. Patch2dSizeX[ p ] – 1, x = 0.. Patch2dSizeY[ p ]– 1.
* borderPoints[p], a 3D border points list for a patch of index p.
* borderBoundingBox[p], a border bounding box list for a patch of index p.
* neighbouringPatches[p], a list for all patches of neighbouring patches for a patch of index p.

The process is divided into two steps:

* Dilatation of geometry map
* Identification of boundary points and neighbouring patches

### Dilatation of geometry map

Issued from the push-pull algorithm, the dilation process fills the background pixels of gFrame[ 0 ][ y ][ x ][0] with pixel values from lower levels:

For all patch, p, the following applies:

N= Ceil(Log2(max (Patch2dSizeX[ p ] , Patch2dSizeY[ p ] )))

width [N] = Patch2dSizeX[ p ]

height[N] =Patch2dSizeY[ p ]

For u = 0..Patch2dSizeX[ p ] − 1, v = 0..Patch2dSizeY[ p ] – 1, the following applies:

geo [p][N][u][v][0] = gFrame[0][u][v][0]

ocm[N][u][v][0] = oFrame[u][v][0]

for each level, l = N..1

width [l-1] = ( width [l] + 1 ) /2

height[l-1] =( height[l] + 1 ) /2

For u = 0..width[ l-1 ] , v = 0..height[ l-1 ]

w1 = ocm[l][u\*2][v\*2][0] : 255 : 0

w2 = ocm[l][u\*2+1][v\*2][0] : 255 : 0

w3 = ocm[l][u\*2][v\*2+1][0] : 255 : 0

w4 = ocm[l][u\*2+1][v\*2+1][0] : 255 : 0

if (w1+w2+w3+w4 > 0)

geo [p][l-1][u][v][0] = (w1\* geo[p][l][u\*2][v\*2][0] + w2 \* geo[l][u\*2+1][v\*2][0] +

w3\* geo[p][l][u\*2][v\*2+1][0]  + w4 \* geo[p][l][u\*1][v\*2+1][0]) / (w1+w2+w3+w4)   
 ocm[l-1][u][v][0] = 1

for each level, l = 1 to N

for u = 0..width[ l ], v = 0.. height[ l ]

if (!ocm[l][u][v][0])

geo [p][l-1][u][v][0] = ( 144 \* geo[p][l-1][u>>1][v>>1][0]

+ 48 \* geo[p][l-1][u>>1 + (u%2== 0 ? -1 :1 ) ][v>>1][0]

+ 48 \* geo[p][l-1][u>>1  ][v>>1 + (v%2== 0 ? -1 :1 )][0]

+16 \* geo[p][l-1][u>>1  + (u%2== 0 ? -1 :1 ) ][v>>1 + (v%2== 0 ? -1 :1 )][0] ) / 255

dilatedGeoFrame[p][x][y][0] = geo [p][N][u][v][0];

### Identification of boundary points and neighbouring patches

for u = 0..Patch2dSizeX[ p ] − 1, v = 0..Patch2dSizeY[ p ] – 1, the following applies:

if ( oFrame[u][v][0] & ( !oFrame[u-1][v][0] | !oFrame[u+1][v][0] ||

!oFrame[u][v-1][0] | !oFrame[u][v+1][0] | !oFrame[u-2][v] [0] ) ||

!oFrame[u+2][v][0] | !oFrame[u][v-2][0] | !oFrame[u][v+2][0] ) ||

 !oFrame[u-1][v-1][0] |!oFrame[u-1][v+1][0] | !oFrame[u+1][v-1][0] || !oFrame[u+1][v+1][0] )

reconstruct a 3D point, point, as described in 9.4.6 “reconstruct 3D point position” [V-PCC\_SCD\_d40]

add point to borderPoints[p] and borderBoundingBox[p]

for each patch, p

for each patch, candidatePatch, with candidatePatchIndex != p

if boundingBox of patch intersects the boundingBox of candidatePatch

add candidatePatch to neighbouringPatches[p]

## Patch Border Filtering

Inputs to this process are:

* Atlas coordinates (x, y).
* Patch coordinates (u, v).
* Index of the patch, p.
* oFrame[ y ][x][0], first component of the occupancy map corresponding to the current point cloud frame at nominal resolution, where y = 0..asps\_frame\_height – 1, x = 0..asps\_frame\_width – 1.
* dilatedGeoFrame[p][x][y][0], first component of the dilated geometry frames at nominal resolution for a patch of index p, where  y = 0.. Patch2dSizeX[ p ] – 1, x = 0.. Patch2dSizeY[ p ]– 1.
* borderPoints[p], a 3D border points list for a patch of index p.
* borderBoundingBox[p], a border bounding box list for a patch of index p.
* neighbouringPatches[p], a list for all patches of neighbouring patches for a patch of index p.

Outputs to this process are:

* oPbfFrame[ y ][x][0], first component of the updated occupancy map corresponding to the current point cloud frame at nominal resolution, where y = 0..asps\_frame\_height – 1, x = 0..asps\_frame\_width – 1.

The process is the following:

Build neighborhood

for each patch, p

for each neighbouringPatches[p], q

    for all 3D border points of q, b

        if b in borderBoundingBox[p]

(ur,vr,dr) = generate2D( p, r )

if ( | dr – dilatedGeoFrame[p][ur][vr][0] | < 10 )

neighborDepth[ur][vr]= dr

Erode occupancy frame

for each patch, p, iter = 0..1, u = 0.. sizeU, v =0..sizeV

if( oFrame[u][v][0] )

countNeighbor = oFrame[u-1][v][0]+oFrame[u+1][v][0]+oFrame[u][v-1][0]+oFrame[u][v+1][0]

if ( !countNeighbor )

oPbfFrame[u][v][0] = 0

else if( countNeighbor < 4)

orientation = orientation[ (oFrame[u-1][v-1][0]<<7)|(oFrame[u][v-1][0]<<6)|

(oFrame[u+1][v1][0]<<5)|(oFrame[u1][v][0]<<4)|(oFrame[u+1][v][0]<<3)|

(oFrame[u-1][v+1][0]<<2)|(oFrame[u][v+1][0]<<1)|( oFrame[u+1][v+1][0]) ]

vX = vectorOrientation[orientation]

vY = vectorOrientation[(orientation+2)%8]

for xW = -sizeWindowX..sizeWindowsX,  yW = -sizeWindowY..sizeWindowsY

if ( neighborDepth[p][u+vX[0]\*xW+vY[0]\*yW][v+vX[1]\*xW+vY[1]\*yW])

sumP += sqrt( (xW\*vX[0]+yW\*vY[0])^2+pow(xW\*vX[1]+yW\*vY[1])^2 +

+(neighborDepth[p][u+vX[0]\*xW+vY[0]\*yW][v+vX[1]\*xW+vY[1]\*yW]

-dilatedGeoFrame[p][u][v][0])^2 );

sumE += sqrt( ((xW+1)\*vX[0]+yW\*vY[0])^2+ ((xW+1)\*vX[1]+yW\*vY[1])^2

+ (neighborDepth[p][u+vX[0]\*xW+vY[0]\*yW][v+vX[1]\*xW+vY[1]\*yW] –

-dilatedGeoFrame[p][u-vX[0]][ v-vX[1]][0])^2 );

count++;

oPbfFrame[u][v][0] = ((count == 0) || (sumE >= sumP))

else

oPbfFrame[u][v][0] = 1

with:

vectorOrientation[8][2] ={{1,0},{1,1},{0,1},{-1,1},{-1,0},{-1,-1},{0,-1},{1,-1}}

orientation = { 0,0,6,0,0,0,0,6,4,0,0,5,0,0,0,5,0,0,0,0,0,0,7,7,0,0,0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,4,0,5,0,0,0,5,

0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,5,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,3,3,0,4,3,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,

0,0,0,0,0,0,7,7,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,7,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,3,3,0,3,0,0,0,4,

1,0,0,0,1,0,1,0,2,3,0,0,1,2,0,0};

Stuffing: create new 3D border points

for each patch, p, u=0..sizeU, v = 0..sizeV0

if (oPbfFrame[u][v][0] & ( !oPbfFrame[u-1][v][0] |!oPbfFrame[u+1][v][0] | !oPbfFrame[u][v-1][0] |

!oPbfFrame[u][v+1][0] | !oPbfFrame[u-2][v][0] | !oPbfFrame[u+2][v][0] |

!oPbfFrame[u][v-2][0] | !oPbfFrame[u][v+2][0] | !oPbfFrame[u-1][v-1][0] |

!oPbfFrame[u-1][v+1][0] | !oPbfFrame[u+1][v-1][0] | !oPbfFrame[u+1][v+1][0] ) )

reconstruct a 3D point, point, as described in 9.4.6 “reconstruct 3D point position” [V-PCC\_SCD\_d39]

add point to borderPoints[p]

for each patch, p

for each neighbouringPatches[p], q

    for all 3D border points, b

        if b in borderBoundingBox[p]

(ur,vr,dr) = generate2D( p, r )

if ( | dr – dilatedGeoFrame[p][ur][vr][0] | < 10 )

neighborDepth[ur][vr]= dr

for each patch, p

for u = 0..Patch2dSizeX[ p ] − 1, v = 0..Patch2dSizeY[ p ] – 1, the following applies:

if(oFrame[u][v][0]&(!oFrame[u-1][v][0] | !oFrame[u+1][v][0] |

!oFrame[u][v1][0] | !oFrame[u][v+1][0] )

for uW = -size.size,  uW = -size..size

distance = sqrt( du^2+dv^2+( neighborDepth[du][dv] - dilatedGeoFrame[p][u][v][0])^2)

      if( distance > minDistance )

nearPoint = (du,dv, neighborDepth[du][dv])

minDistance = distance

if( minDistance in [ 2; MaxDistance] )

for i = 1.. minDistance-1

generate3DPoints( u + i \* (nearPoint[0] – u ) / minDistance ) + 0.5,

v + i \* (nearPoint[1] – v ) / minDistance ) + 0.5,

d + i \* (nearPoint[2] – d ) / minDistance ) + 0.5 )

# Conclusions

The Patch block filtering brings gains in term of metrics, computation times and memory usage.

We recommend that this process is adopted in V-PCC specification.

# References

[[w18665](http://wg11.sc29.org/doc_end_user/current_document.php?id=70087&id_meeting=179)] Common Test Conditions for PCC, Gothenburg, Sweden, July 2019, ISO/IEC JTC1/SC29 WG11.

[[w18666](http://wg11.sc29.org/doc_end_user/current_document.php?id=70088&id_meeting=179)] V-PCC Test Model v7, Gothenburg, Sweden, July 2019, ISO/IEC JTC1/SC29 WG11.