 ISO/IEC JTC 1/SC 29/WG 7 N0570

**ISO/IEC JTC 1/SC 29/WG 7  
MPEG 3D Graphics Coding   
Convenorship: AFNOR (France)**

**Document type:** Output Document

**Title:** V-DMC codec description

**Status:** Approved

**Date of document:** 2023-07-01

**Source:** ISO/IEC JTC 1/SC 29/WG 7

# Expected action: None

# Action due date: None

**No. of pages:** 29 (with cover page)

**Email of Convenor:** marius.preda @ imt . fr

**Committee URL:** [https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg7](https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3)

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 7 MPEG 3D Graphics Coding**

**ISO/IEC JTC 1/SC 29/WG 7 N** **0570**

**April 2023, Antalya, TR**

|  |  |
| --- | --- |
| **Title** | **Draft for V-DMC codec description** |
| **Source** | **WG 7, MPEG 3D Graphics Coding** |
| **Status** | **Approved** |
| **Serial Number** | **MDS22730** |

# Abstract

ISO/IEC MPEG (JTC 1/SC 29/WG 7) is studying the potential need for standardization of dynamic mesh coding technology with a compression capability that significantly exceeds that of the current approaches and will target to create the standard. The group is working together on this exploration activity in a collaborative effort known as the 3D Graphics Coding and Haptics Coding (3DGH) to evaluate compression technology designs proposed by their experts in this area.

This document is the V-DMC test model for dynamic mesh coding v4 (TMM v4) algorithm description. It describes the coding features that are under a coordinated test model study by 3DGH as potential dynamic mesh coding technology. The description of encoding strategies used in experiments for the study of the new technology in the TMM is also provided.

Ed. Notes (changes to V-DMC TMM release 4.0)

-----------Release 2.0-----------

m61094: add draco position compression

m61222: add draco uv-coordinate compression

m61002: add texture transfer improvement

m60812: add 1d displacement

m61034: add degenerate triangles

-----------Release 3.0-----------

m61585 [V-DMC][EE4.3] Report of EE 4.3 on Common Test Conditions for V-DMC

m61819 [V-DMC][EE4.4] Report for EE4.4 on base mesh coding

m61729 EE4.5 report (On texture transfer improvements)

m62189 EE4.7 report on displacement coding

m62072 [V-DMC][EE4.12] Report for EE4.12 on motion field coding

-----------Release 4.0-----------

m62540 [V-DMC-NEW] New background filling proposal for V-DMC reference software

m62541 [V-DMC-NEW] Background filling for skip mode

m62560 [V-DMC EE4.8 related] Optimization of orthoAtlas parameters

m62566 [V-DMC][EE4.12] Report of EE4.12 Motion coding context simplification

m62567 [V-DMC][EE4.12] Report of EE4.12 Motion predictor neighbors simplification

m62603 [V-DMC][EE4.4] Edgebreaker base mesh codec integration

m62604 [V-DMC][EE4.4] syntax proposal for the edgebreaker base mesh codec

m62632 [V-DMC] [New] On Bdrate calculation

m62634 [V-DMC] [EE4.7] Report on EE4.7 Test 1.1

m62652 [V-DMC][EE4.4-related] Metric instability in the evaluation of base mesh coding

m62674 [V-DMC][EE/CE] Report for EE4.4 on base mesh coding

m62676 [V-DMC][EE4.4] Position Enhancement Coding for Lossless Mesh Compression

m62722 [V-DMC][EE 4.4] Report on the SKIP type in base mesh coding

m62939 [V-DMC][EE4.7] Report on Test 3.4 (4:2:0 displacement video)

m63020 [V-DMC][EE4.7 Test 2.2] Block-Based Context-Adaptive Arithmetic Coding of Displacements

m63022 [V-DMC][EE4.7 Test 4.1] Block-Based Bypass Arithmetic Coding of Displacements

m63063 [V-DMC][EE4.13] Report for the EE4.13 on adding the V3C structure in the TMM

m63064 [V-DMC][EE4.3] VVenc/VVdec

m63066 [V-DMC][EE4.3] Update CTC conditions

m63257 [V-DMC][New] On displacement vector coding

m63297 [V-DMC][EE4.7 Test 1.2] Displacement Quantization for Each Level of Detail

m63491 [V-DMC] Padding and partial decoding of displacements

# Introduction

The test model for video-based dynamic mesh compression (TMM) is a new project that was started after the Call for Proposals (CfP) for Dynamic mesh coding [1]. The core encoding and decoding process for V-DMC were inherited from the solution [2] that demonstrated the highest compression efficiency among all proponents as was agreed during the 138th online MPEG meeting.

This document provides a description of the various encoding algorithms and best practices for pre and post processing used in TMM.

All the methods listed have been integrated into the main software branch of the mpeg-vmesh-tm [3]. The TMM solution is codec agnostic, however, in the current testing procedure, HM HEVC encoder implementation is used for video-based coding, and the implementation of the Edgebraeker is used for the geometry coding. Corresponding codecs are a part of a solution [3].

This document consists of two parts and provides a description of normative coding tools and a set of best practices (non-normative tools description).

# Description of coding features and encoding method

The technical details of each coding method used in V-DMC TMM test model are described in the following sub-sections. The description of the encoding strategies is also provided.

Figure 1. V-DMC TMM encoding structure.

At the encoding stage input set of the dynamic mesh frames is processed in the following manner.

First, the volumetric 3d data must be represented as a set of base mesh, and it’s corresponding refinement components.

At the decimation, stage mesh topology is reduced to the base mesh. The base mesh is coded with an existing coding solution such as draco or other Edgebreaker implementation.

At the subdivision stage the base mesh is subdivided using predefined algorithm. The subdivided mesh is later on fitted to the original input mesh surface and corresponding displacement vectors formulate a displacement component that is further transformed with a wavelet transform and the transformation coefficients may be further quantized.

An attribute map of the original mesh frame is re-parametrized in accordance with the decimated base mesh. An attribute map is coded using existing video coding solution.

Base mesh generation method, and atlas packing strategies are outside of the scope of the standard and corresponding current implementations are described in the best practices section of this document.

The block structure shown in Figure 1 is used for encoding while for decoding the block structure in Figure 2 is used.

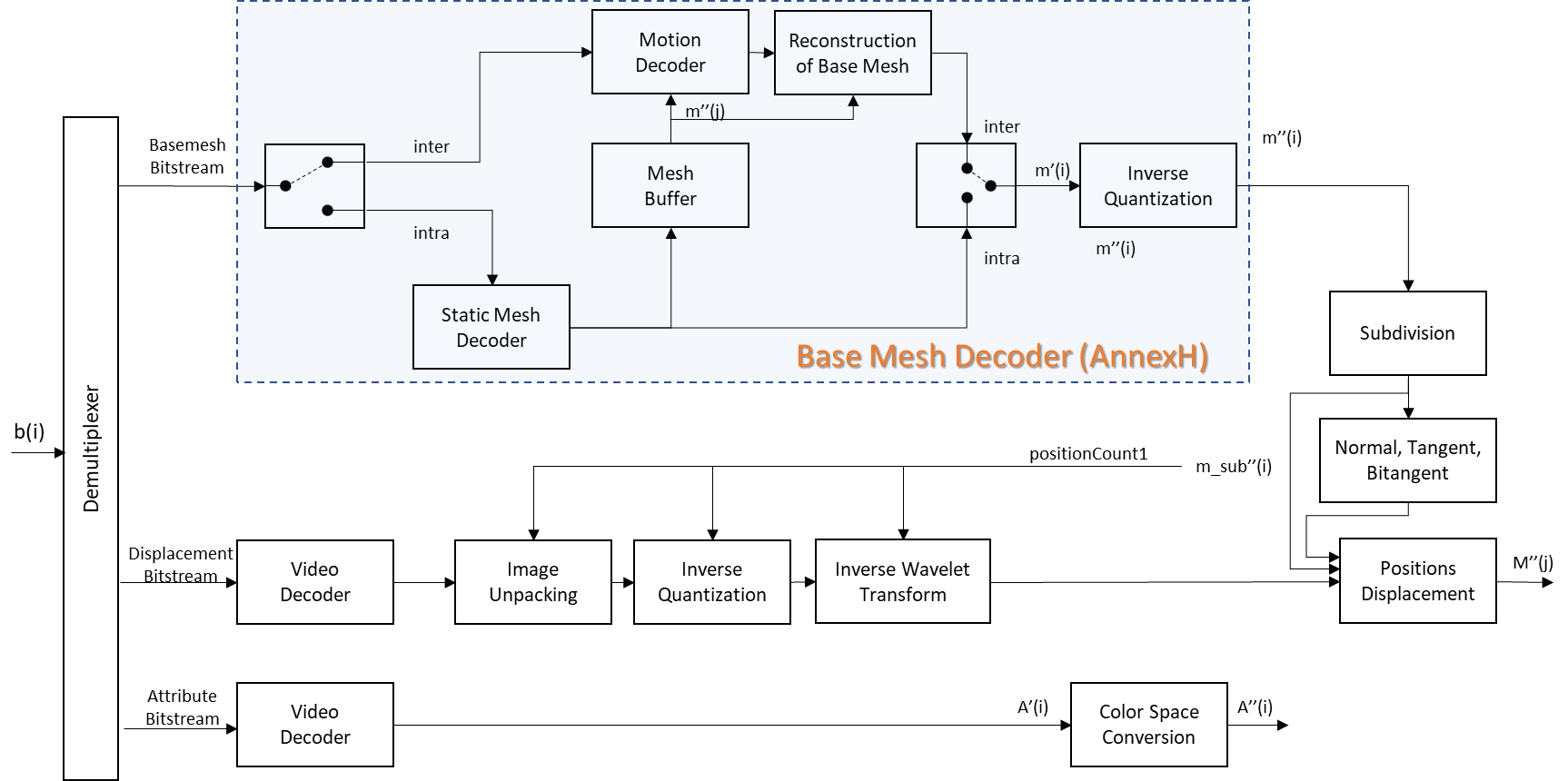


Figure 2. V-DMC TMM decoding structure

Decoding process starts form demultiplexing of the input compressed binary file into components that carry base mesh geometry, displacement geometry refinement, attribute, and metadata information sub-streams.

The base mesh bitstream is coded using two components a static mesh decoder indicated by v3c component, or a motion vector field decoder indicated by a v3c component. The motion field can be only coded for the registered meshes – thus limiting the application for constant connectivity and number of vertices in a base mesh.

The displacement component is decoded using a codec indicated in a v3c component, and if coded using video codec further image unpacking is applied. The displacement component coefficients are dequantized and converted to displacement values using inverse wavelet transform.

An attribute bitstream is decoded using a video decoder indicated in v3c component.

When the decoding of all the components is finished the mesh can be reconstructed.

# Pre processing for dynamic mesh coding

# Re-parametrization process for subdivided mesh

The proposed pre-processing scheme includes the following sub-blocks:

* Mesh decimation,
* Atlas parameterization, and
* Subdivision surface fitting.

The mesh decimation module uses a simplification techniqueto decimate the input mesh and produce the decimated mesh . The decimated mesh is then re-parameterized. The generated mesh is denoted as

Applying re-parameterization to the input mesh makes it possible to generate a lower number of patches. This reduces parameterization discontinuities and may lead to better RD performance. The subdivision surface fitting module takes as input the re-parameterized mesh and the input mesh and produces the base mesh together with a set of displacements . First, is subdivided by applying the subdivision scheme. The displacement field is computed by determining for each vertex of the subdivided mesh the nearest point on the surface of the original mesh .

# Dynamic mesh coding

# Dynamic mesh geometry component coding procedure

A diagram of a device

Description automatically generated

Figure 3. Mesh topology coding

# Dynamic mesh geometry component transformation in V-DMC TMM

The main advantage of the subdivided curve is that it has a subdivision structure that allows efficient compression, while it offers a faithful approximation of the original curve. The compression efficiency is obtained thanks to the following properties:

- The decimated/base curve has a low number of vertices and requires a limited number of bits to be encoded/transmitted.

- The subdivided curve is automatically generated by the decoder once the base/decimated curve is decoded (i.e., no need for any information other than the subdivision scheme type and subdivision iteration count).

- The displaced curve is generated by decoding the displacement vectors associated with the subdivided curve vertices. Besides allowing for spatial/quality scalability, the subdivision structure enables efficient transforms such as wavelet decomposition, which can offer high compression performance.

|  |  |  |
| --- | --- | --- |
| A picture containing text  Description automatically generated | A statue of a person  Description automatically generated | A picture containing text  Description automatically generated |
|  |  |  |
| Original mesh | Decimated mesh | Deformed mesh |

Figure 4. Mesh transformation process (1 frame)

# Base mesh coding process

Base mesh coding process is described using the diagram below. Overall there are three possible coding methods – intra using static mesh encoder, inter using predicitive motion field coding, and skip mode as a special case for inter coded motion field.

A black screen with red lines and a red sign

Description automatically generated

Figure 7: Base mesh coding methods.

# Static mesh coding

# Motion field coding

# Motion field predictive coding

The motion vector predictor is calculated in three steps in V-DMC test model 3.0 as shown below.

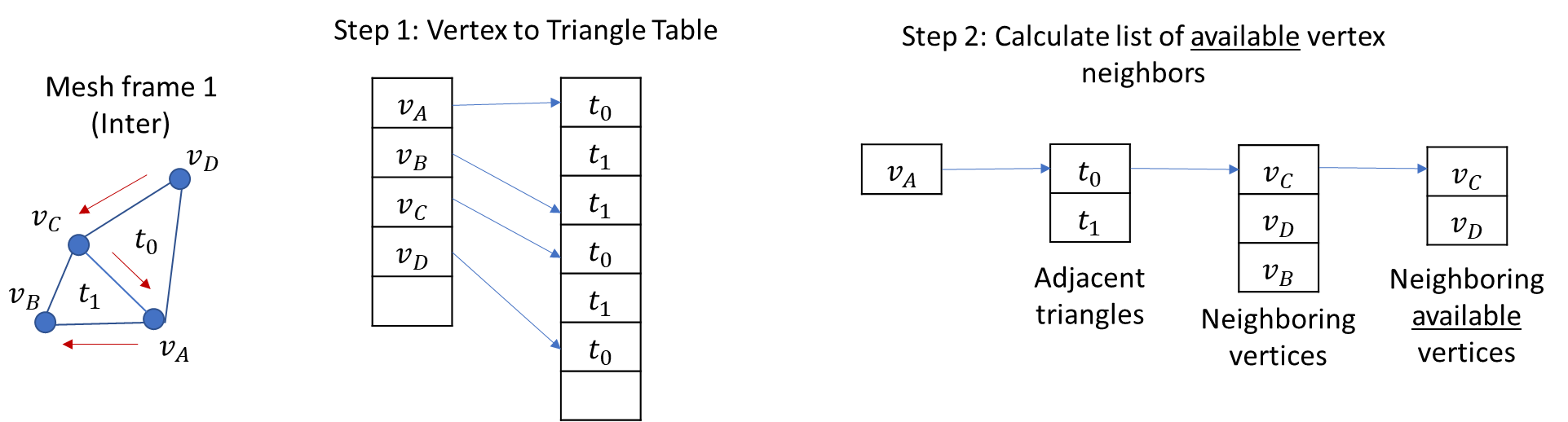


Figure 7: Calculation of motion vector predictor.

Step 1: Vertex to Triangle Adjacency table calculation

* Loop 1: Calculate the number of adjacent triangles for each vertex (variable number)
* Loop 2: Set up pointer table to reserve variable amount of memory for each vertex in the vertex triangle table
* Loop 3: Populate the vertex to triangle table with variable number of triangle neighbors for all vertices

Step 2: Calculate list of available vertex neighbors

* Assume vertex transmission order is D, C, A, B
* For vertex A
  + Loop 1: Calculate the list of neighboring vertices from the list of adjacent triangles
  + Loop 2: Prune the list to pick only available vertices

Step3: Motion vector predictor = average motion of available neighboring vertices

A group of arrows pointing to the right

Description automatically generated

Figure 7: Calculation of motion vector predictor with restricted search.

# Entropy coding for motion field

V-DMC uses a combination of unary and exp-golomb code [1] as illustrated in Figure 1 below. The yellow highlighted part is the unary part and the blue part is the exp-golomb part. The sign bit is coded separately.



The method is sharing the contexts across X, Y, Z and within the prefix part. It uses bypass coding for the suffix part. Figure 3 below shows the context memory used for coding the 3D vertex motion vector difference of (X = 15, Y = 12, Z = 9). The number of contexts required reduces to 8 (C0 – C7). Please note that B in Figure 3 indicates bypass coding.



# Mesh subdivision.

The process of mesh subdivision and displacement component calculation for a case with 1 dimensional displacements is described in Figure 5.

A diagram of a triangular structure

Description automatically generated

Figure 5. Mesh transformation process (1 triangular face)

Various subdivision schemes could be considered. A possible solution is the mid-point subdivision scheme, which at each subdivision iteration subdivides each triangle into 4 sub-triangles. New vertices are introduced in the middle of each edge. The subdivision process is applied independently to the geometry and to the texture coordinates since the connectivity for the geometry and for the texture coordinates is usually different. The sub-division scheme computes the position of a newly introduced vertex at the center of an edge , as follows:

where and are the positions of the vertices and .

The same process is used to compute the texture coordinates of the newly created vertex. For normal vectors, an extra normalization step is applied as follows:

here:

* , , and are the normal vectors associated with the vertices , , and , respectively.
* is the norm2 of the vector .

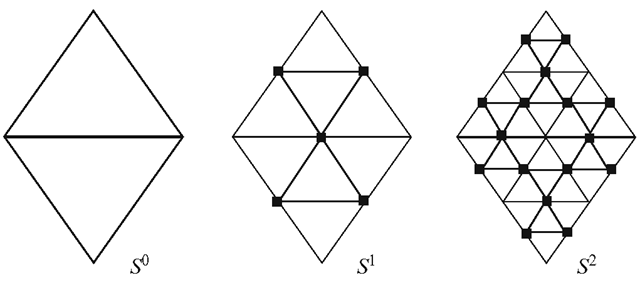


Figure 6. Mesh subdivision process

# Displacement coordinate system

The displacement field d(i) is defined in the same cartesian coordinate system as the input mesh. A possible optimization is to transform d(i) from this canonical coordinate system to a local coordinate system, which is defined by the normal to the subdivided mesh at each vertex.

The advantage of considering a local coordinate system for the displacements is the possibility to quantize more heavily the tangential components of the displacements compared to the normal component. In fact, the normal component of the displacement has more significant impact on the reconstructed mesh quality than the two tangential components.

# Displacement packing process

The following scheme is used to pack the wavelet coefficients into a 2D image:

* Traverse the coefficients from low to high frequency.
* For each coefficient, determine the index of the pixel block (e.g., ) in which it should be stored following a raster order for blocks.
* The position within the pixel block is computed by using a Morton order **Error! Reference source not found.** to maximize locality.

Other packing schemes could be used (e.g., zigzag order, raster order). The encoder could explicitly signal in the bitstream the used packing scheme (e.g., atlas sequence parameters). This could be done at patch, patch group, tile, or sequence level.

Two methods of padding can be applied – staring form the high LoD or from the low LoD – the difference in padding strategies is described in Figure 7.





Figure 7: Displacement packing process simulation results.

# Attribute coding

# Attribute transfer

The attribute transfer module aims at computing a new attribute map based on the input mesh and the input texture map that is better suited for the reconstructed deformed mesh . The proposed algorithm proceeds as follows (*cf.* Figure 8):

* For each pixel of the attribute map to be generated, compute its texture coordinates .
* Determine if the point in the texture space belongs to a triangle of .
* If does not belong to any triangle, mark this pixel as an empty pixel that could optionally be filled by a padding algorithm.
* Otherwise, if belongs to a triangle defined by the three vertices , then:
  + Mark the pixel as filled.
  + Compute the barycentric coordinates of the point according to the triangle in the 2D parametric space.
  + Compute the 3D point associated with the point by using the barycentric coordinates and the 3D positions associated with the triangle in 3D space.
  + Find the nearest 3D point located on the triangle of the original mesh.
  + Compute the barycentric coordinates of according to in 3D space.
  + Compute the point associated with by using the barycentric coordinates with the 2D parametric coordinates associated with the three vertices of .
  + Use the texture coordinates to sample the original texture map and compute the attribute value .
  + Assign the attribute value to the pixel of the attribute map .

The empty pixels are padded by combining the Push-Pull algorithm with the sparse linear padding algorithm.

Graphical user interface, website

Description automatically generated

Figure 8: Attribute transfer process.

# Atlas map generation for attribute

# Orthographic patch attribute creation

orthoAtlas is a tool to create texture mapping coordinates by using orthographic projections. The implementation is divided into two parts: Patch Creation and Patch Packing.

Patch Creation

A screenshot of a computer

Description automatically generated

*Figure 9: Overview of the patch creation process*.

In the patch creation stage, first some properties of the triangles are calculated, like the adjacent list of neighboring triangles and the triangles normal and surface area. The triangle neighborhood can be defined by sharing a single vertex (params.useVertexCriteria = **true**) or sharing an edge (params.useVertexCriteria = **false**). For each triangle, the normal and surface area in 3D is calculated. Notice that the triangles’ surface area and normal can be derived from the projected areas, as shown by the formulas below (this information will be later used for the derivation of the triangle stretch distortion).

A screenshot of a computer screen

Description automatically generated

*Figure 10: Calculation of normal and surface area using projected areas*

Now the triangle are first divided into cluster of triangles that are connected by their neighbors, also known as connected components. We have two methods to initiate the clustering algorithm, that is, to choose the seed for the connected component: (1) (params.bUseSeedHistogram = **true**) The most frequent orientation is obtained by checking the triangle’s categories’ histogram. The seed is the triangle, whose normal is closest aligned with the most frequent orientation and has not been added to a connected component; (2) (params.bUseSeedHistogram = **false**) The seed uses the first available triangle (i.e., not in a connected component list) is used. Next, the neighboring triangles are added to the connected component list following certain criteria (note that some of the same criteria has been described by Nokia in their response to the call for proposal []):

* + If the triangle’s category is the same as the connected component orientation
  + If the angle between the last inserted normal and the current normal is smaller than a certain threshold (params.strongGradientThreshold)
    - params.strongGradientThreshold = **180** deactivates this criteria
  + If the area of the patch is smaller than a fraction (params.maxCCAreaRatio) of the total area
    - params.maxCCAreaRatio = **1.0** deactivates this criteria
  + If the number of triangles is smaller than a threshold (params.maxNumFaces)
    - params.maxNumFaces = **max<int> (or maxNumFaces > mesh.triangles().size)** deactivates this criteria

Currently the selection criteria for adding triangles to the connected components is mostly unused to avoid smaller patches but could be turned on if necessary.

At this point, the number of patches is significantly high, with also a lot of small-sized connected components. Then like UVAtlas, we perform a merge operation to reduce the number of connected components (cc). The merge function will merge neighboring connected components depending on a novel COST function, defined by the cc’s perimeter and ortho stretch (the stretch caused by orthographic projection., , which will be explained later). The following algorithm describes the merge operation:

1. Create an ordered list of connected components
   * Order the list by (1) smallest number of triangles, (2) average normal (weighted by the triangle area) most aligned with the orientation
   * Each connected component starts with a cost:
2. While the list is not empty:
   1. Remove the top element of the list ()
   2. Obtain the list of candidate neighbors (, where is the list of connected components that share an edge/vertex with )
   3. For each neighboring candidate:
      * Calculate the merging cost:
      * If the merge operation with one of the neighbors (connected components that share at least one edge) reduces the cost
        + (),
      * then select the neighbor that minimizes the joint cost
        + ()
   4. If a neighbor was selected for merge, update the connected component () and resort the list.

The stretch function is a measure of distortion due to the orthographic projection and can be derived by the ratio of the projected areas in each direction. This is inspired by the stretch measure used in UVAtlas, and the formula for orthographic projection is shown below.

A diagram of a triangle with lines and numbers

Description automatically generated

*Figure 11: orthoStretch derivation*

*A screenshot of a computer

Description automatically generated*

Finally, the merged connected components are then checked for overlapping projected vertices and surfaces. The overlapped areas are broken into new connected components, which are then sent to the patch packing module. The figure below shows the impact of the clustering algorithm in terms of the new proposed stretch metric and the number of connected components.

A person standing in front of a black background

Description automatically generated

*Figure 12: From left to right, initial isolated triangle, then combined in the initial connected components, followed by the merge and the orthographic projection segmentation.*

## Patch Packing

A screenshot of a computer

Description automatically generated

Figure 13: Overview of the patch packing process

At first, an overall frame scale is determined to maximize the patch occupancy in the texture map surface. The frame scale is determined as follows:

Initially, the adjustment factor is set to 1.0, If packing is not successful, the scaling is reduced by 10% (i.e. adjustment = adjustment \* 0.9) to provide more space for the projected connected components. Patch scaling can be done on top of the frame scaling to adapt the texture size on a per-patch basis. This can allow for specific areas with greater details to use more texture space. If the patch cannot be packed, then its size is reduced by 10%. If the total size reduction surpasses 50% of the initial patch size, then the packing is deemed to have failed, and a new frame scaling is calculated.

# Background filling for attribute map

# Smoothed PushPull (SPP)

|  |  |
| --- | --- |
| For each pixel location (x, y) in the area of padding:  image(x,y,c) = ( image(x-1,y-1,c) +  image(x+1,y-1,c) +  image(x-1,y+1,c) +  image(x+1,y+1,c) +  image(x ,y-1,c) +  image(x ,y+1,c) +  image(x-1,y ,c) +  image(x+1,y ,c) ) + 4 ) >> 3; |  |

A diagram of a flowchart

Description automatically generated

# Sparse Linear model based padding method

The sparse linear model helps to create a smoother transition between patches than the original method. The algorithm of this process is as follows:

* Let be the attribute associated with the pixel at location (*i,j*), and with the full resolution occupancy . is equal to 1 if the pixel is inside a point cloud patch, i.e. is a full pixel, and 0 if not, i.e. is an empty pixel.
* Let be the set of empty pixels, the set of full pixels, and the set of all pixels.
* Let be the set of neighboring pixels of the pixel
* We formulate the padding as a minimization problem, which tries to find the values of empty pixels E such that the obtained padded image is as smooth as possible. More precisely, we would like to find the colors , such that the following cost function is minimized:

where is the number of available neighbors of the pixel . For interior pixels, equals 4. For pixels on the boundary of the image is lower than 4.

# Harmonic background filling method

Harmonic Background Filling, we create a linear model and optimize its solution on a multi-resolution framework. We define our system using a 7-point Laplacian with Neumann constraints, but solve the linear system using Gauss-Seidel relaxation while initializing the solution from lower dimension representation. The occupancy map defines the positions that need to be updated, and also guides the multi-resolution creation. In the case of dyadic sub-sampling of the higher resolution image, the occupied value is carried over to the lower resolution if any of its corresponding 4 positions in high-resolution are occupied. The linear system is solved in lower resolution, and the solution is replicated to the higher resolution as an initial step for the optimization problem. We solve the linear system using Gauss-Seidel relaxation using 10-pass. Another advantage of this approach is that it can optimize the value considering all the video layers together, that is, the optimization can be spatial and temporal as well.

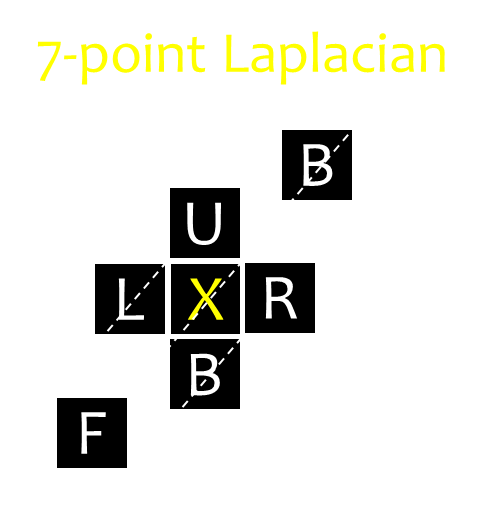


Figure 13: 7 tap Laplacian filter

# References

[1] CfP for Dynamic Mesh Coding, ISO/IEC JTC 1/SC 29/WG 7 MDS21000\_WG07\_N00231, Online, October 2021.

[2] [V-CG] Apple’s Dynamic Mesh Coding CfP Response, ISO/IEC JTC 1/SC 29/WG 7 m59281 Online, April 2022

[3] <http://mpegx.int-evry.fr/software/MPEG/dmc/mpeg-vmesh-tm>

**Annex A: Objective Evaluation Metrics & Usage of Metric Software**

***A.1 Point-based metric***

The point-based metric converts the reference and distorted meshes into two point clouds by applying the sampling procedure described in section A.1.1.

***A.1.1 Mesh sampling***

The point cloud is created by performing ray-casting in the axis direction (x,y,z), depending on the normal of the triangle. A hit test determines if the casted ray hits the triangle, then the color is obtained by barycentric interpolation (to determine the UV coordinate of the point), and then bilinear interpolation (to get the RGB value from texture map). The normal of the triangle is computed as the cross product of its two edges and normalized to have a unit length. All the points obtained by sampling the triangle inherit its normal vector.

***A.1.2 Geometric Distortions***

Let and denote the original and the compressed point cloud, respectively. Consider evaluating the compression errors, denoted as in point cloud relative to reference point cloud. The steps to compute both point-to-point error (D1) and point-to-plane error (D2) for geometric errors are summarized in the following and illustrated in the below figure.

For each point in point cloud, i.e., the black point in the figure, identify a corresponding point in point cloud, i.e. the red point in the figure. Nearest neighbour is used to locate the corresponding point. In particular, a KD-tree search is used to perform the nearest neighbour search in order to reduce the computation complexity.

***A.1.2.1 Computing D1***

Determine an error vector by connecting the identified point in reference point cloud  to point in point cloud . The length of the error vector is the point-to-point error, i.e.,

Based on the point-to-point distances for all points , the point-to-point error (D1) for the whole point cloud, withas the number of points in point cloud, is defined as:

The peak value for PSNR is clipped at 100.

***A.1.2.2 Computing D2***

Project the error vector along the normal direction and get a new error vector. In this way, the point-to-plane error is computed as,

The point-to-plane error (D2) for the whole point cloud is then defined as,

A diagram of a graph

Description automatically generated

Figure 14. Illustration of point-to-point distance (D1) and point-to-plane distance (D2)

***A.1.3 Geometric PSNR Calculation***

The geometric PSNR value is computed as:

where is the maximum length of the sequence bounding box (maxBBLength) as defined in Table 1, and is the symmetric mean squared point-to-point () or point-to-plane () error, which are obtained by considering the maximum distortions and computed as follows:

and .

For dynamic content, the peak value is unchanged over the frames of a sequence.

The peak value for PSNR is clipped at 100.

***A.1.4 Attribute Distortions***

The attribute PSNR value is computed as:

For color attributes, the MSE for each of the three color components is calculated. A conversion from RGB space to YUV space is conducted using ITU-R BT.709, since YUV space correlates better with human perception. A symmetric computation of the distortion is utilized, in the same way as is done for geometric distortions. The maximum distortion between the two passes is selected as the final distortion. Since the color attributes for all test data have a bit depth of 8 bits per point, the peak value for PSNR calculation is 255.

***A.2 Image-based metric***

The computation of the lossy metric is based on image MSE/PSNR processing [7][8]. An overview of the approach is given in **Error! Reference source not found.**. For each frame, the reference and the distorted models are rendered for several view directions , using an orthographic projection (see section A.2.1). The images obtained from the rendering of reference and distorted models are then compared using some adapted image MSE/PSNR metrics (see section A.2.3). The results are averaged over a set of view directions for the frame and over the frames of the sequence.

Diagram

Description automatically generated

Figure 15. Overview of the image-based metric. Ref/DisColor are the color images/buffers. Ref/DisMask are binary images where pixel[i,j]=1 if there exist a projection in associated color buffer, and 0 otherwise. Ref/DisDepth are the depth buffers. All the buffers have same dimensions of 2048x2048 pixels.

The number of views is fixed to 16 and the resolution of image, depth and mask buffers is set to 2048x2048 pixels. The Figure 16 gives an example of rendered image buffers for the 16 view directions.

Graphical user interface, application

Description automatically generated

Figure 16. Example of color images generated for the basketball player using 16 views.

***A.2.1 Rendering of one view***

Graphical user interface, application

Description automatically generated

Figure 17. Rendering of image buffers for a given view direction vdi.

The rendering is part of the metric and is implemented inside the metric software.

The rendering of one view is illustrated in Figure 17. The pixels of the image could be obtained by ray tracing or rasterization of the mesh.

The bounding sphere is obtained by summing the axis aligned bounding box of the distorted and reference objects and taking the diagonal and center of the resulting bounding box.

The view directions always points toward the center of the bounding sphere in 3D space.

The mesh is rendered using an *orthogonal projection*. The projection plane for the direction is the plane tangent to the bounding sphere and perpendicular to the view direction vector.

The mesh is rendered using *clockwise back-face culling* which suits all the models of the anchor in terms of visual rendering.

Note: if the provided answer does not preserve faces orientations based on clockwise orientations it will get low MSE/PSNR scores for the metric.

The rendering step generates the color, the mask, and the depth buffers.

* The color buffer contains for each pixel i,j the RGB value, of the nearest projected triangle.
  + In case of a textured mesh, the RGB color is obtained by bilinear interpolation of the texture map using triangle UV coordinates.
  + In case of color per vertex meshes (no texture map), the vertex colors are blended using barycentric coordinates.
* The mask buffer contains for each pixel i,j a binary value set to 1 if a projection for this pixel exists and 0 otherwise.
* The depth buffer contains for each pixel i,j the distance from the projection plane to the 3D surface in 3D space.

***A.2.2 Positioning of views***

The positioning of the 16 views, that is to say the set of orientations, is obtained by using a [Fibonacci sphere lattice](http://extremelearning.com.au/evenly-distributing-points-on-a-sphere/) [11]. This distribution aims at generating points over a sphere in an evenly spaced manner. Once one has the points , the directions are the vectors passing through these points and pointing toward the center of the sphere. The Figure 16 gives an example of generated images for 16 views using this method.

The Fibonacci sphere samples are computed as follows:

const double pi = std::atan(1.0) \* 4;

// golden angle in radians

Const float phi = (float)(pi \* (3. - std::sqrt(5.)));

// glm::vec3 a vector of three floats, each entry a direction

std::vector<glm::vec3> camDir;

for (size\_t i = 0; i < targetNbSamples; ++i) {

float y = 1 - (i / float(targetNbSamples - 1)) \* 2; // y goes from 1 to - 1

float radius = std::sqrt(1 - y \* y); // radius at y

float theta = phi \* i; // golden angle increment

float x = std::cos(theta) \* radius;

float z = std::sin(theta) \* radius;

camDir.push\_back(glm::vec3(x, y, z));

}

A special attention is also given to generate the up vectors (see Figure 17). The up vector determines, for a given direction , the rotation of the camera on the direction axis, and by side effect the final orientation of the model in the image. At the end, we obtain the camera matrix using the well-known [LookAt function from OpenGL](https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml) [12].

For a given view direction viewDir (), the viewUp () vector is defined as follows:

if (glm::abs(viewDir) == glm::vec3(0, 1, 0))

viewUp = glm::vec3(0, 0, 1);

else

viewUp = glm::vec3(0, 1, 0);

In other terms, whenever view direction is not a north or south pole, we use y axis vector as the up vector , otherwise we use the z axis vector as the up vector .

Those values were selected so most of the models visually renders head at top of image when rendered by side or front views. The special case of poles is the simplest positive default value for .

***A.2.3 distortion for one view***

The calculus of the distortion is based on the general Mean Squared Error (MSE) formula. Let be a sample of an original image, a sample of a distorted image and the width of both buffers in pixels. The for the two images is calculated as follows:

In the rest of this document, we will decline this formula to our needs, especially restricting processing to parts where the mask generated from the reference and distorted models is equal to one. Let be a matrix of same size of and (Each mask buffer contains for each pixel i,j a binary value set to 1 if a projection for this pixel exists and 0 otherwise. See section B.2.1).

Let be the width of the buffers in pixels. The combined number of projected pixels of all the view directions for one frame is:

A conversion from RGB space to YUV space is performed by using ITU-R BT.709, since YUV space correlates better with human perception.

In the following the color image/buffer is considered YUV.

Let be a sample of the color image/buffer of the reference model for view direction , a sample of the color image/buffer of the distorted model for view direction . The color MSE for the YUV images, noted , is calculated as follows on each component in :

Let be a sample of the depth image/buffer of the reference model for view direction , a sample of the depth image/buffer of the distorted model for view direction . Let be the dynamic range of the depth signal initialized with the maximum between the diagonals of the bounding boxes of both models. The depth MSE for the depth images/buffers, noted , is calculated as follows:

This depth renormalization to 255 is used to get geometric MSEs and PSNRs comparable to the color ones.

The final metric results per frame are the followings:

And the respective associated with are computed with the following formula:

The metric tool reports also the ratio of unmatched samples defined as follows:

where is given by:

Additionally, the metric tool reports also the ratio () of the bounding box diagonal of the reference and the diagonal of the bounding box of the distorted mesh as follows:

***A.3 Calculation of MSE and PSNR for a sequence***

Metrics for a sequence are obtained by computing the **Minimum**, the **Maximum**, the **Mean**, the **Variance,** and the **Standard deviation** of the metric evaluated on each frame of the sequence.

***A.4 Software usage for computing the point cloud and image-based metrics***

The “equ” and “ibsm” metrics are symmetrical, but the “pcc” is not. To ensure the numerical stability of the results among participants it is required that:

* the **--modelA and --MapA** are used for the **reference** model**.**
* the **--modelB and --mapB** are used for the **distorted** model**.**

***A.4.1 Lossless condition***

Lossless condition with log to console:

./mm compare \

--mode equ \

--inputModelA ref.obj –inputMapA ref.png \

--inputModelB dis.obj –inputMapB dis.png > summary.txt

***A.4.1 Lossy condition***

Lossy condition on dequantized models using the point-based metric with log to console and statistics reporting into files:

./mm \

reindex --sort oriented -i ref.obj -o ID:ref\_reordered END

sample --mode grid --gridSize 512 \

--useNormal --useFixedPoint --minPos "$globalMinPos" --maxPos "$globalMaxPos" \

--bilinear -i ID:ref\_reordered -m ref.png -o ID:pcRef END\

reindex --sort oriented -i dis.obj -o ID:dis\_reordered END

sample --mode grid --gridSize 512 \

--useNormal --useFixedPoint --minPos "$globalMinPos" --maxPos "$globalMaxPos" \

--bilinear -i ID:dis\_reordered -m dis.png -o ID:pcDis END\

compare --mode pcc \

--inputModelA ID:pcRef --inputModelB ID:pcDis \

--outputCsv perFrame.csv > summary.txt

Lossy condition on dequantized models using the image-based metric with log to console and statistics reporting into files:

./mm compare --mode ibsm \

--inputModelA ref.obj --inputMapA ref.png \

--inputModelB dis.obj --inputMapB dis.png \

--outputCsv perFrame.csv > summary.txt

**Attention:** Lossy condition shall be applied on dequantized models.

**Annex B: TM-D Software configuration**

***B.1 General***

This section provides several examples of configuration files for the TMM encoder, decoder, and MMetric software.

***B.1.1 Lossless AI condition***

Encoder:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai-ll.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

srcMesh: /Streams/dataset/basketball\_player\_fr%04d\_qp12\_qt12.obj

srcTex: /Streams/dataset/basketball\_player\_fr%04d.png

startFrameIndex: 1

frameCount: 300

positionBitDepth: 12

texCoordBitDepth: 12

target: 1

minCCTriangleCount: 0

minPosition: -725.812988 -483.908997 -586.02002

maxPosition: 1252.02002 1411.98999 1025.34998

textureParametrizationWidth: 2048

textureParametrizationHeight: 2048

textureParametrizationGutter: 16

textureParametrizationQuality: QUALITY

ai\_subdivIt: 0

ai\_sdeform: 1

ai\_forceNormalDisp: 0

ai\_unifyVertices: 1

ai\_deformNormalThres: -2.0

ai\_deformNNCount: 1

ai\_deformFlipThres: -2.0

ai\_useInitialGeom: 1

ai\_smoothMotion: 0

ai\_smoothMethod: 1

# skipped 1 elements: 'defined ${fitsubdiv\_with\_mapping} and !${fitsubdiv\_with\_mapping}' is false

# with fitsubdiv\_with\_mapping = '(undef)'

# skipped 10 elements: 'defined ${pp} and ${pp} < 1.0 and ( defined ${fitsubdiv\_inter} and ${fitsubdiv\_inter})' is false

# with pp = '1'

# and fitsubdiv\_inter = '(undef)'

invertOrientation: 0

baseMeshTexCoordBitDepth: 10

gofMaxSize: 32

liftingBias: 0.3333,0.3333333333333333,0.3333333333333333

unifyVertices: 0

textureVideoWidth: 2048

textureVideoHeight: 2048

textureVideoEncoderConvertConfig: /mpeg-vmesh-tm/cfg/hdrconvert/bgr444toyuv420.cfg

textureVideoDecoderConvertConfig: /mpeg-vmesh-tm/cfg/hdrconvert/yuv420tobgr444.cfg

analyzeGof: 0

subdivIsBase: 1

baseIsSrc: 1

encodeGeometryVideo: 0

ai\_fitSubdiv: 0

liftingIterationCount: 0

textureTransferEnable: 0

textureBGR444: 1

unifyVertices: 0

dracoMeshLossless: 1

textureVideoEncoderConfig: /mpeg-vmesh-tm/cfg/hm/texture-ai-ll.cfg

baseMeshPositionBitDepth: 12

baseMeshTexCoordBitDepth: 12

Decoder:

# This file was automatically generated from:

# ./scripts/../cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai-ll.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

startFrameIndex: 1

textureVideoDecoderConvertConfig: /mpeg-vmesh-tm/cfg/hdrconvert/yuv420tobgr444.cfg

MMetric:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai-ll.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

startFrameIndex: 1

frameCount: 300

srcMesh: /Streams/dataset/basketball\_player\_fr%04d\_qp12\_qt12.obj

srcTex: /Streams/dataset/basketball\_player\_fr%04d.png

positionBitDepth: 12

texCoordBitDepth: 12

minPosition: -725.812988 -483.908997 -586.02002

maxPosition: 1252.02002 1411.98999 1025.34998

pcc: 1

ibsm: 1

gridSize: 1024

resolution: 1977.833008

***B.1.2 Lossy AI condition***

Encoder:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

imesh: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d\_qp10\_qt12.obj

itex: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d.png

fstart: 1051

fcount: 300

gdepth: 10

tdepth: 12

qt: 12

target: 0.03

cctcount: 8

minPosition: -0.475553989 -1.4576 -0.284981996

maxPosition: 481.324005 1023.37 659.137024

width: 2048

height: 2048

quality: QUALITY

gutter: 16

ai\_subdivIt: 3

ai\_sdeform: 1

ai\_forceNormalDisp: 0

ai\_unifyVertices: 1

ai\_deformNormalThres: -2.0

ai\_deformNNCount: 1

ai\_deformFlipThres: -2.0

ai\_useInitialGeom: 1

ai\_smoothMotion: 0

ai\_smoothMethod: 1

# skipped 1 elements: 'defined ${fitsubdiv\_with\_mapping} and !${fitsubdiv\_with\_mapping}' is false

# with fitsubdiv\_with\_mapping = '(undef)'

# skipped 10 elements: 'defined ${pp} and ${pp} < 1.0 and ( defined ${fitsubdiv\_inter} and ${fitsubdiv\_inter})' is false

# with pp = '0.03'

# and fitsubdiv\_inter = '(undef)'

invorient: 0

tqp: 10

gofmax: 32

dqb: 0.3333,0.3333333333333333,0.3333333333333333

normuv: 0

unifvertices: 0

texwidth: 2048

texheight: 2048

cscencconfig: /mpeg-vmesh-tm/cfg/hdrconvert/bgr444toyuv420.cfg

cscdecconfig: /mpeg-vmesh-tm/cfg/hdrconvert/yuv420tobgr444.cfg

analyzeGof: 0

# skipped 1 elements: '${pp} >= 0.24 and ${pp} < 1.0' is false

# with pp = '0.03'

# skipped 2 elements: 'not defined ${pp} or ${pp} == 1.0' is false

# with pp = '0.03'

gvencconfig: /mpeg-vmesh-tm/cfg/hm/ctc-hm-displacements-map-ai-main10.cfg

tvencconfig: /mpeg-vmesh-tm/cfg/hm/ctc-hm-texture-ai.cfg

liftingIt: 3

gqp: 10

tvqp: 47

dqp: 30,42,42

Decoder:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

fstart: 1051

cscdecconfig: /mpeg-vmesh-tm/cfg/hdrconvert/yuv420tobgr444.cfg

normuv: 0

MMetric:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

fstart: 1051

fcount: 300

srcMesh: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d\_qp10\_qt12.obj

srcTex: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d.png

qp: 10

qt: 12

minPosition: -0.475553989 -1.4576 -0.284981996

maxPosition: 481.324005 1023.37 659.137024

pcc: 1

ibsm: 1

gridSize: 1024

***B.1.3 Lossy LD condition***

Encoder:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

imesh: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d\_qp10\_qt12.obj

itex: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d.png

fstart: 1051

fcount: 300

gdepth: 10

tdepth: 12

qt: 12

target: 0.03

cctcount: 8

minPosition: -0.475553989 -1.4576 -0.284981996

maxPosition: 481.324005 1023.37 659.137024

width: 2048

height: 2048

quality: QUALITY

gutter: 16

ai\_subdivIt: 3

ai\_sdeform: 1

ai\_forceNormalDisp: 0

ai\_unifyVertices: 1

ai\_deformNormalThres: -2.0

ai\_deformNNCount: 1

ai\_deformFlipThres: -2.0

ai\_useInitialGeom: 1

ai\_smoothMotion: 0

ai\_smoothMethod: 1

# skipped 1 elements: 'defined ${fitsubdiv\_with\_mapping} and !${fitsubdiv\_with\_mapping}' is false

# with fitsubdiv\_with\_mapping = '(undef)'

# skipped 10 elements: 'defined ${pp} and ${pp} < 1.0 and ( defined ${fitsubdiv\_inter} and ${fitsubdiv\_inter})' is false

# with pp = '0.03'

# and fitsubdiv\_inter = '(undef)'

invorient: 0

tqp: 10

gofmax: 32

dqb: 0.3333,0.3333333333333333,0.3333333333333333

normuv: 0

unifvertices: 0

texwidth: 2048

texheight: 2048

cscencconfig: /mpeg-vmesh-tm/cfg/hdrconvert/bgr444toyuv420.cfg

cscdecconfig: /mpeg-vmesh-tm/cfg/hdrconvert/yuv420tobgr444.cfg

analyzeGof: 1

# skipped 1 elements: '${pp} > 0.24 and ${pp} < 1.0' is false

# with pp = '0.03'

# skipped 2 elements: 'not defined ${pp} or ${pp} == 1.0' is false

# with pp = '0.03'

# skipped 1 elements: 'defined ${fitsubdiv\_inter}' is false

# with fitsubdiv\_inter = '(undef)'

gvencconfig: /mpeg-vmesh-tm/cfg/hm/ctc-hm-displacements-map-ld-main10.cfg

# skipped 1 elements: 'defined ${fitsubdiv\_inter} and ${fitsubdiv\_inter} == 1' is false

# with fitsubdiv\_inter = '(undef)'

tvencconfig: /mpeg-vmesh-tm/cfg/hm/ctc-hm-texture-ai.cfg

liftingIt: 3

gqp: 10

tvqp: 49

dqp: 32,44,44

Decoder:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

fstart: 1051

cscdecconfig: /mpeg-vmesh-tm/cfg/hdrconvert/yuv420tobgr444.cfg

normuv: 0

MMetric:

# This file was automatically generated from:

# ./scripts/../cfg/cfg-site-default.yaml

# ./scripts/../cfg/cfg-tools.yaml

# ./scripts/../cfg/cfg-cond-ai.yaml

# ./scripts/../cfg/cfg-cond-ld.yaml

# ./scripts/../cfg/sequences.yaml

# ./scripts/../cfg/cfg-site.yaml

fstart: 1051

fcount: 300

srcMesh: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d\_qp10\_qt12.obj

srcTex: /Streams/dataset/longdress\_voxelized/longdress\_fr%04d.png

qp: 10

qt: 12

minPosition: -0.475553989 -1.4576 -0.284981996

maxPosition: 481.324005 1023.37 659.137024

pcc: 1

ibsm: 1

gridSize: 1024

**Annex C: Visualization of the voxelized mesh content**

# For the dataset content, the vertices attributes (3D position and texture coordinate) are provided in integer format. For correct visualization of these voxelized meshes, a dequantization procedure that converts the integer values to floating point values should be conducted. The mmetric software provides the capability to dequantize the voxelized meshes by applying the following command line:

**mm** \

**dequantize** --inputModel ${voxelizedMesh} --outputModel ${outMesh} \

--qp ${qp} --qt ${qt} \

--minPos ${globalPosMin} --maxPos ${globalMaxPos} \

--minUV “0.0 0.0” --maxUV “1.0 1.0” \

--useFixedPoint

# The values for qp (quantization parameter for vertex 3D coordinate) and qt (quantization parameters for texture coordinate) is equivalent to the options of Geometry Precision and Texture Coordinate Precision respectively, and are indicated in Error! Reference source not found.. The values for globalPosMin and globalPosMax are provided in Table 1.

Table 1 Conversion parameters for the Mesh sequences

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test material**  **dataset filename** | **globalPosMin** | | | **globalMaxPos** | | | **maxBBLength** |
| **(x)** | **(y)** | **(z)** | **(x)** | **(y)** | **(z)** |
| Longdress | -0.475553989 | -1.4576 | -0.284981996 | 481.324005 | 1023.37 | 659.137024 | 1024.8276 |
| Soldier | -0.366236001 | 1.10722005 | 0.224947006 | 508.764008 | 1023.37 | 637.421997 | 1022.26277995 |
| Basketball\_player | -725.812988 | -483.908997 | -586.02002 | 1252.02002 | 1411.98999 | 1025.34998 | 1977.833008 |
| Dancer | -902.244995 | -486.196991 | -670.518005 | 621.093994 | 1576.04004 | 738.028992 | 2062.237031 |
| Mitch | -588.255981 | 5.80515003 | -469.799011 | 734.567993 | 1829.69995 | 697.179016 | 1823.89479997 |
| Thomas | -265.006989 | -4.04448986 | -248.710999 | 320.546997 | 1820.93005 | 400.225006 | 1824.97453986 |
| Football | -0.000159517003 | 3.32326999e-06 | 0.000132931003 | 1024 | 980.619995 | 966.692993 | 1023.96268540018 |
| Levi | -0.780686975 | -0.0424938016 | -0.594317973 | 0.857237995 | 1.90897 | 0.687259018 | 1.9514638016 |