**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG2016/M40522**

**April 2017, Hobart, AU**

|  |  |
| --- | --- |
| **Source** | **Mitsubishi Electric** |
| **Status** | **Input Contribution** |
| **Title** | **Updates and Integration of Evaluation Metric Software for PCC** |
| **Authors** | **Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, Anthony Vetro** |

# Abstract

When evaluating the geometric distortions on point clouds, we proposed point-to-plane metrics (D2) that consider surface structure while keeping the approach with limited computation complexity. In this document, we describe an improved implementation on the proposed point-to-plane metric that runs faster than the earlier implementation. With the new implementation, the dependency on Point Cloud Library (PCL) is removed. In addition, the metric tool also reports point-to-point metric (D1) and RGB distortions that are to be used for the PCC development. As a unified and standalone tool, we recommend the MPEG 3DG group to take this codebase to report the D1, D2 and RGB distortions. The source code of the software is enclosed within this document.

# Proposed Point-to-Plane Metrics

In this section, we briefly recall the principals of point-to-plane we proposed during 117th MPEG meeting at Geneva [1].

## Principals of Point-to-Plane Distances

Let  and  denote the original and the compressed point cloud, respectively. Let’s consider to evaluate the compression errors, denoted as , in point cloud  relative to reference point cloud . This error may be referenced as  for quick reference in the AhG discussions.

The steps of computing point-to-point distance  and point-to-plane distance  are summarized as follows (shown in Fig. ).

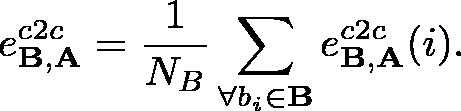
1) For each point  in point cloud , i.e., the black point in the figure, identify a corresponding point  in point cloud , i.e. the red point in the figure. Nearest neighbor is used to locate the corresponding point.

2) Take the unit normal vector  on the identified point in the reference point cloud , if available. Otherwise, the normal vector is estimated on the fly. We use -NN, with  as default approach to estimate the normal vectors.

3) Compute an error vector  by connecting the identified point  in reference point cloud  to point  in point cloud . The length of the error vector would actually lead to the point-to-point error, i.e.,

 (1)

Based on the point-to-point distance  for all points , the point-to-point (C2C) distance for the whole point cloud is defined as,

 (2)

where is the number of points in point cloud .

4) To get point-to-plane errors, we project the error vector  along the normal direction  and get a new error vector . In the end, the point-to-plane error is computed as,

 (3)

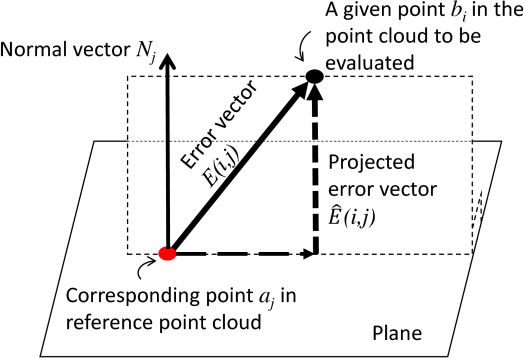
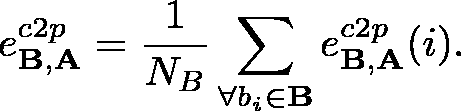


Figure 1: Point-to-point distance (C2C) vs. point-to-plane distance (C2P)

Finally, we define the point-to-plane (C2P) distance for the whole point cloud as,

 (4)

Compared to point-to-point distances, the point-to-plane distance measures a projected error vector along normal directions rather than measuring the error vector directly, that would impose larger penalty on the errors that do not follow the local plane surface. Given point cloud characterized by surfaces, point-to-plane distances would be better aligned with perceptual quality of point clouds.

A full description on the point-to-plane metric approach can be found in [1].

# Integration at Building Level (CMake Approach)

During 117th MPEG meeting at Geneva, 3DG group decided to include point-to-plane metric (referred as D2) in addition to point-to-point metric (D1) for dataset category C1 and C2.

There exist two independent implementations on the metric evaluation. Implementation 1 has been hosted in the MPEG SVN repository for some time. It reports D1 and RGB color distortions. Implementation 2 is based on MERL’s codebase. It reports D1, D2 and RGB distortions. Implementation 2 also incorporated a new way to convert the MSE to PSNR using the intrinsic resolutions. The source code of Implementation 2 is enclosed in this document.

In this section, a first effort to do the integration is described.

With a first approach, the shared procedure in the algorithm level is ignored and no integration at source code level between D1 and D2 was performed. Furthermore, it tries to integrate the compiling of D2 software within the compiling of the current “anchor” software. Hence, this approach basically involves editing the hierarchical CMakeLists.txt in the “anchor” software, and referred as a “cmake” approach. We understand that additional re-organization of the source code files may be also necessary to fit the file structures in PCL.

As a result, D1 and D2 would have their own executable after a successful building.

Problems observed with this approach:

1. Difficult to edit the CMakeFiles.txt

Because the “anchor” software depends on the Point Cloud Library (PCL) in source code level, the complex structure makes it rather difficult to edit the build instruction files (CMakeLists.txt). This was making troubles in implementing this approach.

1. Difficult to compile

It takes several hours to build the anchor code, and the integrated metric. Because the “anchor” software has source code level dependency on the Point Cloud Library (PCL), it takes several hours to build the code. Basically over 90 applications are compiled and produced from the “make” command. Most of them are not related to PCC anchor codec, but the tools in the original PCL library.

As of today (Wed, Mar 29, 2017) before uploading this document, there are still compiling problems on the software:

$ date

Wed Mar 29 09:04:58 EDT 2017

$ svn update

Updating '.':

At revision 1068.

$ svn info

Path: .

Working Copy Root Path: /data/tian/pc\_psnr/pcc-mp3dg\_after117

URL: http://wg11.sc29.org/svn/repos/MPEG-04/Part16-Animation\_Framework\_eXte nsion\_(AFX)/trunk/3Dgraphics/3DG-PCC/trunk/pcc-mp3dg

Relative URL: ^/MPEG-04/Part16-Animation\_Framework\_eXtension\_(AFX)/trunk/ 3Dgraphics/3DG-PCC/trunk/pcc-mp3dg

Repository Root: http://wg11.sc29.org/svn/repos

Repository UUID: 94298a81-5874-47c9-aab8-bfb24faeed7f

Revision: 1067

Node Kind: directory

Schedule: normal

Last Changed Author: rmekuria

Last Changed Rev: 1065

Last Changed Date: 2017-03-17 07:57:36 -0400 (Fri, 17 Mar 2017)

1. Two executable for D1, and RGB

There are two executable that reports D1 and RGB distortions. One executable is produced by Implementation 1. The other executable is from Implementation 2. Both executable report D1 and RGB distortions, which is rather confusing.

1. RGB color distortion – single pass

With Implementation 1 in this approach, the RGB distortion is reported from a single-pass procedure. Note that the geometric distortions (D1 and D2) are all reported from a two-pass procedure in both Implementation 1 and Implementation 2.

# Integration at Source Code Level (C/C++ Approach)

This section describes a second approach to implement the integration, which does not face the risks with the CMake approach. The source code of the integration, i.e., Implementation 2, is enclosed in this document. Note that D1 is not newly integrated in this implementation. But RGB distortion is newly integrated.

## A Unified Metric Tool (D1, D2, and RGB)

In addition to the concerns with the integration at “cmake” level, there are good reasons and motivations to do the integration in source code level instead of “cmake” level.

The fundamental motivation is that the computation of D1 is a sub-procedure of the computation of D2.

By separating D1 and D2 at source code level (C/C++), it imposes risks that mismatches may arise for the shared procedure between D1 and D2. In the future, changes in either implementation, people have to run verification tests to ensure no mismatch. They may have to align the code changes in the two metric implementations.

By keeping two executable for D1 and D2, it would lead to redundant computation time (the same running time of D1 computation) as well. That is a waste of computing resource when dealing with the large dataset of MPEG PCC. In addition, people have to script twice when collecting the output data.

Moreover, the common procedure also exist between geometry distortions (D1 & D2) and RGB distortions, since they all need to find the corresponding points in the two input point clouds.

Hence, we recommend to have all distortions, D1, D2, RGB (and reflectance for C3) integrated at the source code level. The reflectance for C3 still need to be tested after the dataset is being fixed.

## A Standalone Metric Tool (From Anchor)

Comparing to the “CMake Approach”, a second merit of the proposed integration approach, is to make the metric software outside of the “anchor” software.

“Anchor” software is an important development of 3DG group, having its unique value for this group project. It should provide a reasonable state-of-the-art performance on point cloud compression. On the contrary, PCL, as a open source project, is intended to be a collection of tools for point cloud processing. Although the “anchor” is based on PCL, we should focus on MPEG targets rather than the open source project targets.

Furthermore, the metric tool is an independent software. There is not a need for the metric software to rely on the “anchor” software. In terms of algorithm, only the kd-tree search from PCL was utilized in addition to loading point clouds. In the improved implementation, we totally removed the dependency of metric software from the PCL.

In the end, by binding the metric software with the “anchor”, it would not bring any benefits, but carry much extra burden to maintain/develop the metric tool, e.g., the problems observed on the “CMake Approach”

A newer organization of the MPEG SVN, that was discussed during AhG meetings between 116th and 117th MPEG, was preferred since it complies with the guidelines discussed above. This structure is partially reproduced below for references.

* Today’s MPEG SVN structure:

http://wg11.sc29.org/svn/repos/MPEG-04/Part16-Animation\_Framework\_eXtension \_(AFX)/trunk/3Dgraphics/3DG-PCC/

· branches/

· tags/

· trunk/

* Proposed MPEG SVN structure:

http://wg11.sc29.org/svn/repos/MPEG-04/Part16-Animation\_Framework\_eXtension \_(AFX)/trunk/3Dgraphics/3DG-PCC/

· PccAppAnchor/[branches, tags, trunk] <=> corresponds to the current /MPEG -04/Part16-Animation\_Framework\_eXtension\_(AFX)/trunk/3Dgraphics/3DG-PCC [trunk, branches, tags]

· PccAppMetrics/[branches, tags, trunk] ==> Metrics application based on MERL codebase (standalone)

· PccAppRenderer/[branches, tags, trunk] ==> Technicolor renderer applica tion (standalone)

· PccAppTestModel/[branches, tags, trunk]

# Software Related Development/Integration

In this section, we summarize the software development and integration work done since the 117th MPEG meeting. Ver0.06 is uploaded with this revised document. No impacts on the reported D1 or D2. See the changes.txt to find out the modifications relative to ver0.05.

## Software Development/Integration

A list of features is highlighted below.

* Removal the dependency on PCL library, making the compiling and running faster.
* Integrated D1, D2 and RGB distortion all in a single, unified C/C++ framework.
* Updated RGB to YUV conversion with BT.709 formula.
* Wrote plain code to load point cloud in .ply format.
* Moved normal estimation outside of the metric tool.
* Provided standalone compiling instructions and example scripts to run the tool.

In the efforts to remove the dependency on PCL, we now use an open source project "nanoflann”, which is dedicated for KD-tree based nearest neighbor searching. Two headers files are imported.

git clone https://github.com/jlblancoc/nanoflann.git

commit: 16696bd113c50c7c32d176b4826a5ba0d05cd9d4

Author: Travis CI <joseluisblancoc@gmail.com>

Date: Thu Mar 2 19:06:28 2017 +0000

We also write plain code to load point clouds. With the new I/O functions, now it is straightforward to adapt the software for new point data types.

For the normal used during point-to-plane computation, we now assume they are provided by an external means that is a typical practice in our experiments. That is, the KNN normal estimation from the previous implementation is dropped in the new codebase. A polished version just for normal estimation from the old codebase could be provided upon request.

A newer functionality added is the capability to report the distortions from color attributes. A conversion from RGB space to YUV space is conducted as YUV space is preferred with respect to human perceptual quality based on the practice in image/video quality assessments.

It should be noted that during color distortion evaluation, a two-pass processing is enforced, in the same way just as the case for geometric distortions (point-to-point and point-to-plane).The worse measurement is selected as the final metric. This is different from the RGB distortion reporting in current MPEG SVN software. It uses a single pass method.

We propose to use the two-pass method for RGB in the same way as D1 and D2.

Note that the new implementation still uses Boost library to interpret the command line parameters. OpenMP is utilized for multi-threading in the computations to speed up the computation.

### Example Command Lines

Example command lines to use the metric tool are provided below.

./pc\_error -a pointcloud1.ply

This will load the point cloud and report NN distances – the intrinsic resolution to be used for PSNR computation.

./pc\_error -a pointcloudOrg.ply -b pointcloudDec.ply

This will produce point-to-point metric.

./pc\_error -a pointcloudOrg.ply -b pointcloudDec.ply -n normalOrg.ply

This will produce point-to-point & point-to-plane metric. The normal is provided in normalOrg.ply for the original point cloud. It could be the same as pointcloudOrg.ply.

./pc\_error -a pointcloudOrg.ply -b pointcloudDec.ply -n normalOrg.ply -c

This will produce point-to-point, point-to-plane as well as RGB metric.

./pc\_error -a pointcloudOrg.ply -b pointcloudDec.ply -n normalOrg.ply -c -r intrinsicFloatNumber

This will produce D1, D2 as well as RGB metric, using an imported intrinsic resolution for PSNR computation rather than an internally determined intrinsic resolution.

## Meta Data Collection, Normals and Intrinsic Resolutions

The above developed software is being used in collecting intrinsic resolutions and anchor statistics. Note that the normal generation is being performed using the previous version of the software, which has a built-in normal estimation.

Both normal data and intrinsic data could be provided during the meeting or within an editing period after Hobart meeting. At least another party than MERL is needed as crosscheckers.

# Conclusions and Recommendations

In this document, we revised the metric software tool to make it more complete by integrating other features required for point cloud compression. We also removed the dependency on the PCL library making it easier to compile and use. We recommend:

* Utilize the enclosed codebase to report D1, D2 and RGB distortions for the CfP;
* Put the enclosed codebase in a dedicated MPEG SVN folder;
* Use the two-pass approach for RGB distortions as done for D1and D2 computations;
* Confirm the normals associated with each data file;
* Confirm the intrinsic resolution to be used for PSNR calculation; and
* Produce an updated output document on the metrics D1, D2, and RGB from this meeting.

# References:

[1] Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro, “Evaluation metrics for point cloud compression,” in *MPEG M39966*, 2017.