INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG04 MPEG VIDEO CODING

ISO/IEC JTC1/SC29/WG04/M74751
Oct. 2025, Geneva, CH

Title	**[GSC][Informative] IV-SSIM integration into mpeg-gsc-metrics software**
Source	**Poznan University of Technology**
	Electronics and Telecommunications Research Institute
Authors	**Jakub Stankowski, Adrian Dziembowski**

1 Abstract

This document describes the integration of IV-SSIM (Immersive Video – Structural Similarity Index Metric) into mpeg-gsc-metrics software.

2 Implementation description

The calculation of IV-SSIM metric has been added to mpeg-gsc-metrics software in way that minimized the possibility of introduction of errors. We used implementation taken from QMIV v2.0 software together with corresponding data structures.

The synthesizer (and reference) image is copied from data container used by mpeg-gsc-metrics (template <typename T, int N = 3> Image) into QMIV native (xPicP). QMIV internally operates on integer images (to avoid floating point related inaccuracies). Therefore, if floating point image is provided (the image container in mpeg-gsc-metrics allows for such an option) pixel values are scaled to 16-bit unsigned range and rounded.

All metric calculation code is copied directly from QMIV. In order to avoid build system incompatibilities we removed all vectorized (SSE, AVX, etc.) and parallelized implementations. This results in significantly slower calculations, however metric results are correct.

3 Verification and testing

The implementation has been tested and verified against QMIV as reference. For several test GSCs we extracted the content of rendered images (both compared and reference) and passed those data to QMIV. The IV-SSIM metric results are the same.

Moreover, we verified the correctness of image copy between containers by calculating PSNR metric using mpeg-gsc-metrics internal routine and QMIV derived code path. Both results matched.

4 Image data conversion source code

Below we provide the code used to convert between mpeg-gsc-metrics and QMIV image structures:

```
template <typename T, int32_t N>
void convertBuffer( PMBB::xPicP& Dst, const Image<T, N>& Src ) {
  using namespace PMBB;

  const int32 Width     = Dst.getWidth();
  const int32 Height    = Dst.getHeight();
  const int32 BitDepth  = Dst.getBitDepth();
  const int32 DstStride = Dst.getStride();

  for ( int32 c = 0; c < 3; c++ ) {
    const Plane<T>& SrcPlane = Src.plane( c );
    uint16*         DstPtr   = Dst.getAddr( eCmp( c ) );

    for ( int32 y = 0; y < Height; y++ ) {
      for ( int32 x = 0; x < Width; x++ ) {
        T SrcA = SrcPlane.get( x, y );
        if constexpr ( std::is_same_v<T, uint8_t> || std::is_same_v<T, uint16_t> ) {
          DstPtr[x] = SrcA;
        } else if constexpr ( std::is_same_v<T, float> ) {
          DstPtr[x] = round( SrcA * xBitDepth2MaxValue( BitDepth ) );
        }
      }
      DstPtr += DstStride;
    }
  }
}
```

5 IV-SSIM metric calculation source code

Below we provide the code used to calculate IV-SSIM metric within mpeg-gsc-metrics software:

```
template <typename T, int32_t N>
void Metric::computeIVSSIM( Results& Result, const Image<T, N>& imageA, const Image<T,
N>& imageB ) {
  using namespace PMBB;

  const int32 width  = imageA.width();
  const int32 height = imageA.height();

  int32 bitDepth = NOT_VALID;
  if constexpr ( std::is_same_v<T, uint8_t> ) {
    bitDepth = 8;
  } else if constexpr ( std::is_same_v<T, uint16_t> ) {
    bitDepth = 16;
  } else if constexpr ( std::is_same_v<T, float> ) {
    bitDepth = 16;
  } else {
    static_assert( false, "Unsupported Image<T, N> type" );
  }

  static_assert( N == 3, "Unsupported number of components" );

  // create PMBB image buffers
```

```
  xPicP picA( { width, height }, bitDepth, 4 );
  xPicP picB( { width, height }, bitDepth, 4 );

  // create IVSSIM calculator
  xIVSSIM procIvSsim;
  procIvSsim.create( { width, height }, bitDepth, 4, false );
  procIvSsim.createThrdPoolIntf( nullptr,
                                 0 );  // create empty thread pool interface - don't
use parallel processing here
  procIvSsim.setStructSimParams( xSSIM::eMode::BlockAveraged, false, 8, 4 );
  procIvSsim.initRowBuffers( height );

  // copy/convert data from Image<T, N> to xPicP
  convertBuffer<T, N>( picA, imageA );
  assert( picA.check( "imageA" ) );
  picA.extend();

  convertBuffer<T, N>( picB, imageB );
  assert( picB.check( "imageB" ) );
  picB.extend();

  // calculate RGB IV-SSIM
  procIvSsim.setCmpWeightsSearch( xIVSSIM::c_EqualCmpWeights );
  procIvSsim.setCmpWeightsAverage( xIVSSIM::c_EqualCmpWeights );
  double rgbIvSsim = procIvSsim.calcPicIVSSIM( &picA, &picB );

  // convert colorspace (in-place)
  xColorSpace::ConvertRGB2YCbCr( picA.getAddr( eCmp::LM ),  //
                                 picA.getAddr( eCmp::CB ),  //
                                 picA.getAddr( eCmp::CR ),  //
                                 picA.getAddr( eCmp::R ),   //
                                 picA.getAddr( eCmp::G ),   //
                                 picA.getAddr( eCmp::B ),   //
                                 picA.getStride(),          //
                                 picA.getStride(),          //
                                 picA.getWidth(),           //
                                 picA.getHeight(),          //
                                 picA.getBitDepth(),        //
                                 eClrSpcLC::BT709 );
  picA.extend();

  xColorSpace::ConvertRGB2YCbCr( picB.getAddr( eCmp::LM ),  //
                                 picB.getAddr( eCmp::CB ),  //
                                 picB.getAddr( eCmp::CR ),  //
                                 picB.getAddr( eCmp::R ),   //
                                 picB.getAddr( eCmp::G ),   //
                                 picB.getAddr( eCmp::B ),   //
                                 picB.getStride(),          //
                                 picB.getStride(),          //
                                 picB.getWidth(),           //
                                 picB.getHeight(),          //
                                 picB.getBitDepth(),        //
                                 eClrSpcLC::BT709 );
  picB.extend();

  // calculate YCbCr IV-SSIM
  procIvSsim.setCmpWeightsSearch ( xIVSSIM::c_DefaultCmpWeights );
  procIvSsim.setCmpWeightsAverage( xIVSSIM::c_DefaultCmpWeights );
  double ycbcrIvSsim = procIvSsim.calcPicIVSSIM( &picA, &picB );
```

```
  // store results
  Result.rgbIvSsim_ = rgbIvSsim;
  Result.yuvIvSsim_ = ycbcrIvSsim;

  // cleanup
  procIvSsim.destroy();
}
```