INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 04 MPEG VIDEO CODING

ISO/IEC JTC 1/SC 29/WG 04 m64709

October 2023, Hannover, DE

Title **[MIV] Implementation of MIV DSDE sub-profile in TMIV**

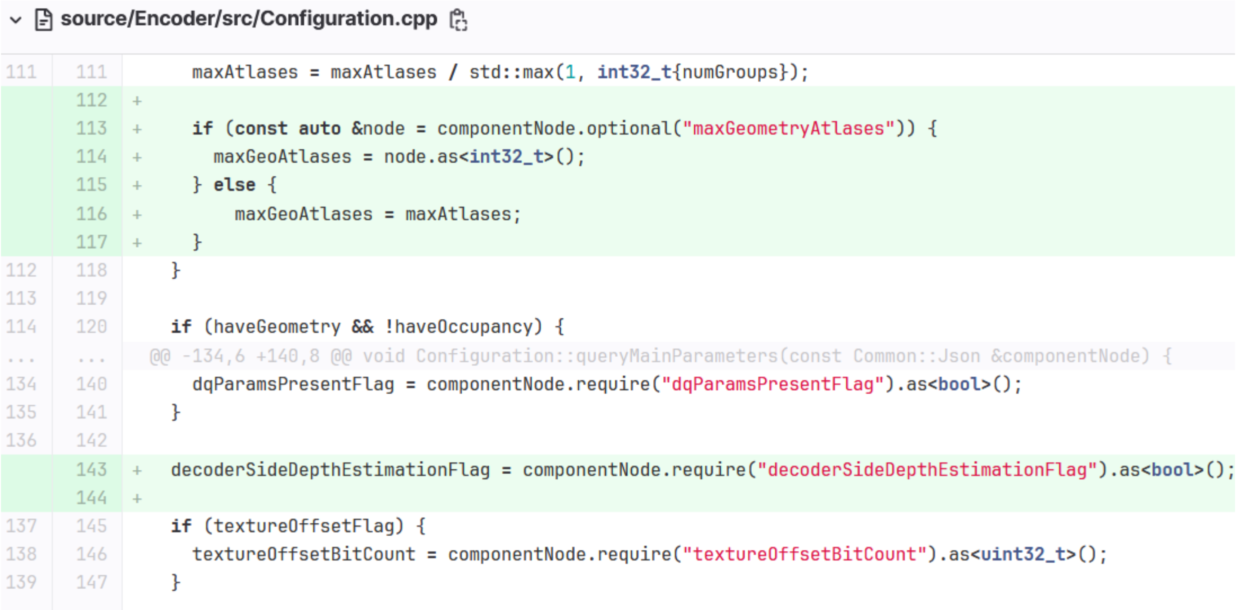
Source **PUT, ETRI**
Authors **Błażej Szydełko, Adrian Dziembowski, Gwangsoon Lee, Jun Young Jeong**

Abstract

In this contribution we propose an implementation of basic features of the MIV Extended DSDE sub-profile, i.e., input depth map assistance (IDMA). The proposal does not add any syntax, but adds the possibility of transmitting a partial geometry information, i.e., depth maps for a subset of basic views (called “essential views”).

1 Implementation

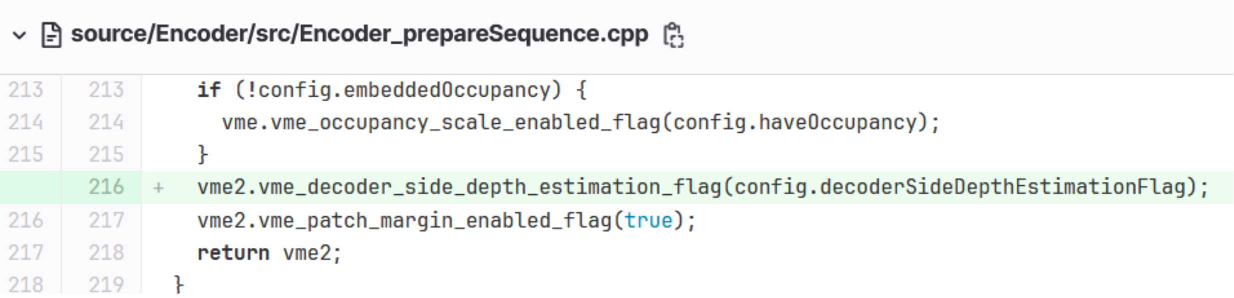
Source code changes:



source/Encoder/src/Configuration.cpp

```
     maxAtlases = maxAtlases / std::max(1, int32_t{numGroups});
+
+    if (const auto &node = componentNode.optional("maxGeometryAtlases")) {
+      maxGeoAtlases = node.as<int32_t>();
+    } else {
+        maxGeoAtlases = maxAtlases;
+    }
   }

   if (haveGeometry && !haveOccupancy) {
@@ -134,6 +140,8 @@ void Configuration::queryMainParameters(const Common::Json &componentNode) {
     dqParamsPresentFlag = componentNode.require("dqParamsPresentFlag").as<bool>();
   }

+  decoderSideDepthEstimationFlag = componentNode.require("decoderSideDepthEstimationFlag").as<bool>();
+
   if (textureOffsetFlag) {
     textureOffsetBitCount = componentNode.require("textureOffsetBitCount").as<uint32_t>();
   }
```

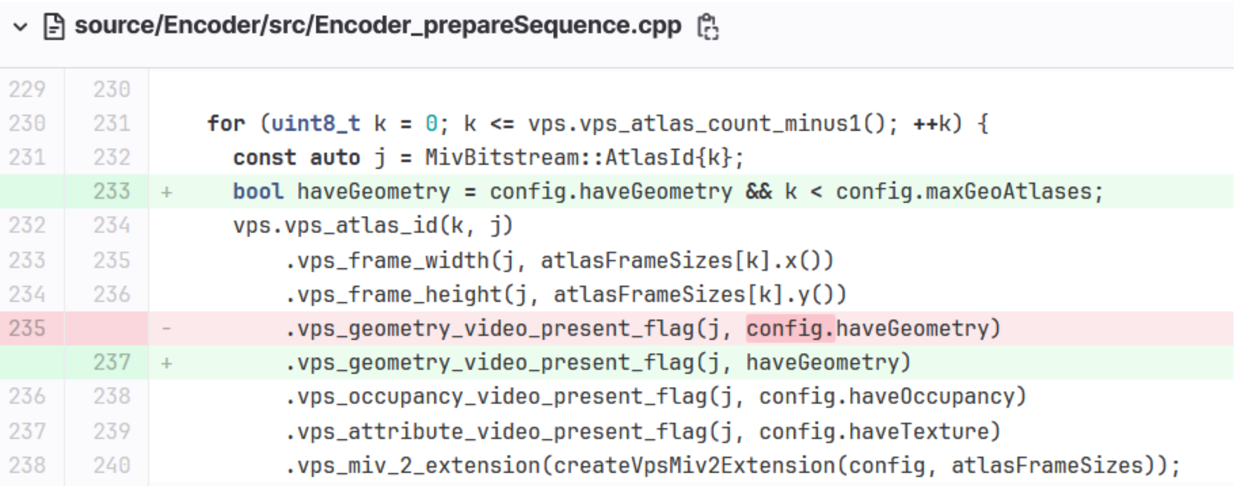
(createVpsMiv2Extension)



source/Encoder/src/Encoder_prepareSequence.cpp

```
      if (!config.embeddedOccupancy) {
        vme.vme_occupancy_scale_enabled_flag(config.haveOccupancy);
      }
+     vme2.vme_decoder_side_depth_estimation_flag(config.decoderSideDepthEstimationFlag);
      vme2.vme_patch_margin_enabled_flag(true);
      return vme2;
    }
```

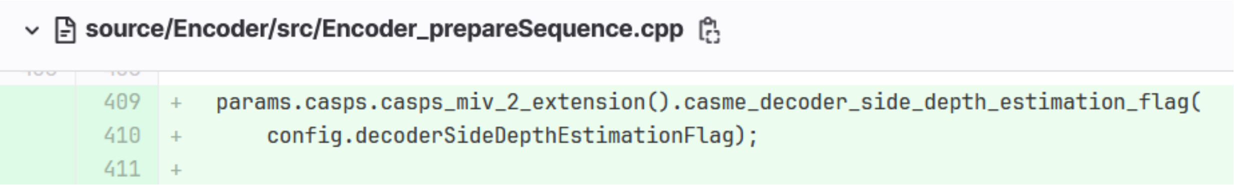
(createV3cParameterSet)



source/Encoder/src/Encoder_prepareSequence.cpp

```

      for (uint8_t k = 0; k <= vps.vps_atlas_count_minus1(); ++k) {
        const auto j = MivBitstream::AtlasId{k};
+       bool haveGeometry = config.haveGeometry && k < config.maxGeoAtlases;
        vps.vps_atlas_id(k, j)
            .vps_frame_width(j, atlasFrameSizes[k].x())
            .vps_frame_height(j, atlasFrameSizes[k].y())
-           .vps_geometry_video_present_flag(j, config.haveGeometry)
+           .vps_geometry_video_present_flag(j, haveGeometry)
            .vps_occupancy_video_present_flag(j, config.haveOccupancy)
            .vps_attribute_video_present_flag(j, config.haveTexture)
            .vps_miv_2_extension(createVpsMiv2Extension(config, atlasFrameSizes));
```

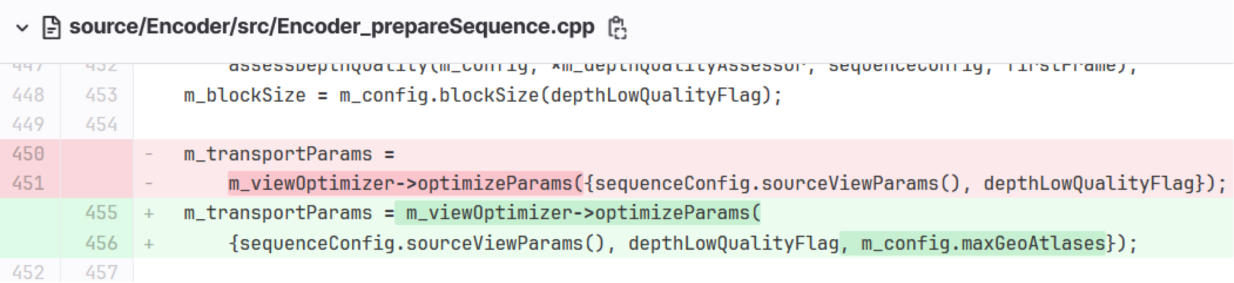
createEncoderParams



source/Encoder/src/Encoder_prepareSequence.cpp

```
+     params.casps.casps_miv_2_extension().casme_decoder_side_depth_estimation_flag(
+         config.decoderSideDepthEstimationFlag);
+
```

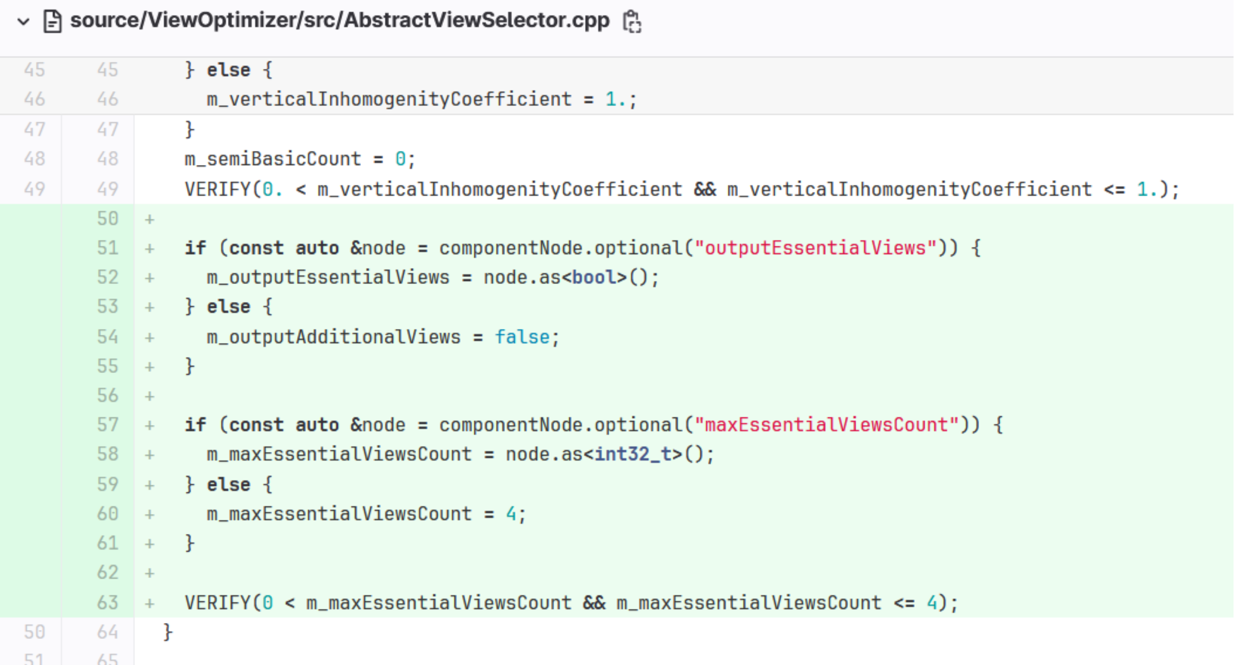
Encoder::Impl::prepareSequence



source/Encoder/src/Encoder_prepareSequence.cpp

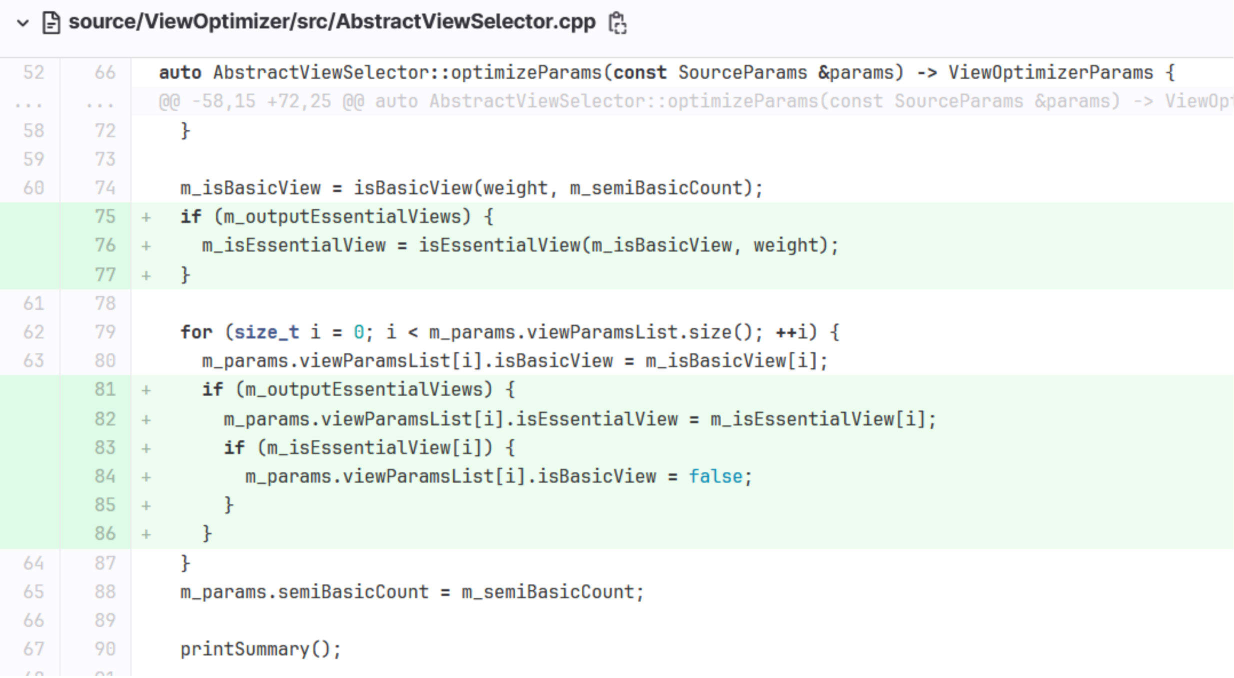
```
        assessDepthQuality(m_config, *m_depthQualityAssessor, sequenceConfig, firstFrame);
    m_blockSize = m_config.blockSize(depthLowQualityFlag);

-   m_transportParams =
-       m_viewOptimizer->optimizeParams({sequenceConfig.sourceViewParams(), depthLowQualityFlag});
+   m_transportParams = m_viewOptimizer->optimizeParams(
+       {sequenceConfig.sourceViewParams(), depthLowQualityFlag, m_config.maxGeoAtlases});

```

AbstractViewSelector



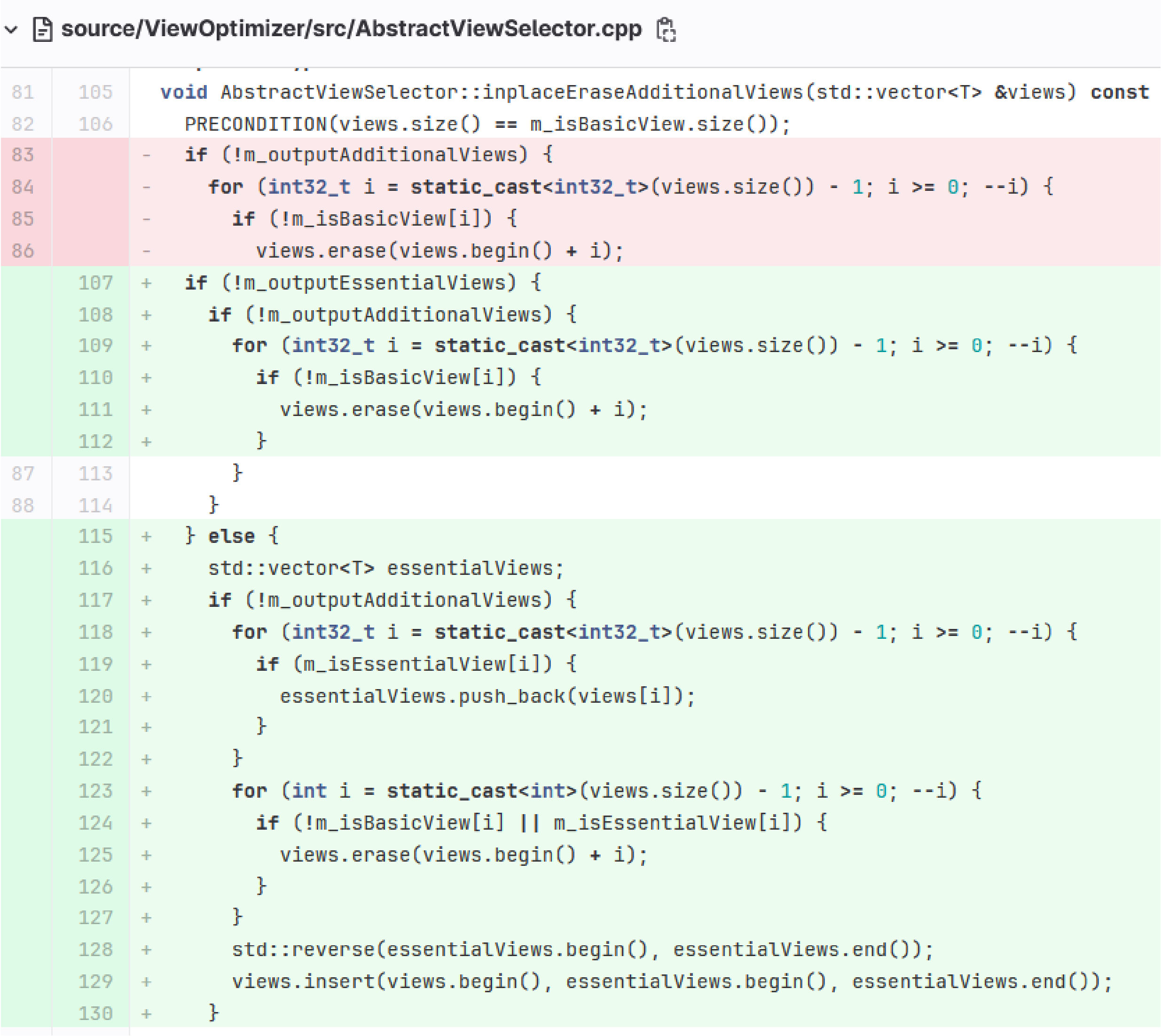
source/ViewOptimizer/src/AbstractViewSelector.cpp

```diff
     } else {
       m_verticalInhomogenityCoefficient = 1.;
     }
     m_semiBasicCount = 0;
     VERIFY(0. < m_verticalInhomogenityCoefficient && m_verticalInhomogenityCoefficient <= 1.);
+
+    if (const auto &node = componentNode.optional("outputEssentialViews")) {
+      m_outputEssentialViews = node.as<bool>();
+    } else {
+      m_outputAdditionalViews = false;
+    }
+
+    if (const auto &node = componentNode.optional("maxEssentialViewsCount")) {
+      m_maxEssentialViewsCount = node.as<int32_t>();
+    } else {
+      m_maxEssentialViewsCount = 4;
+    }
+
+    VERIFY(0 < m_maxEssentialViewsCount && m_maxEssentialViewsCount <= 4);
   }
```



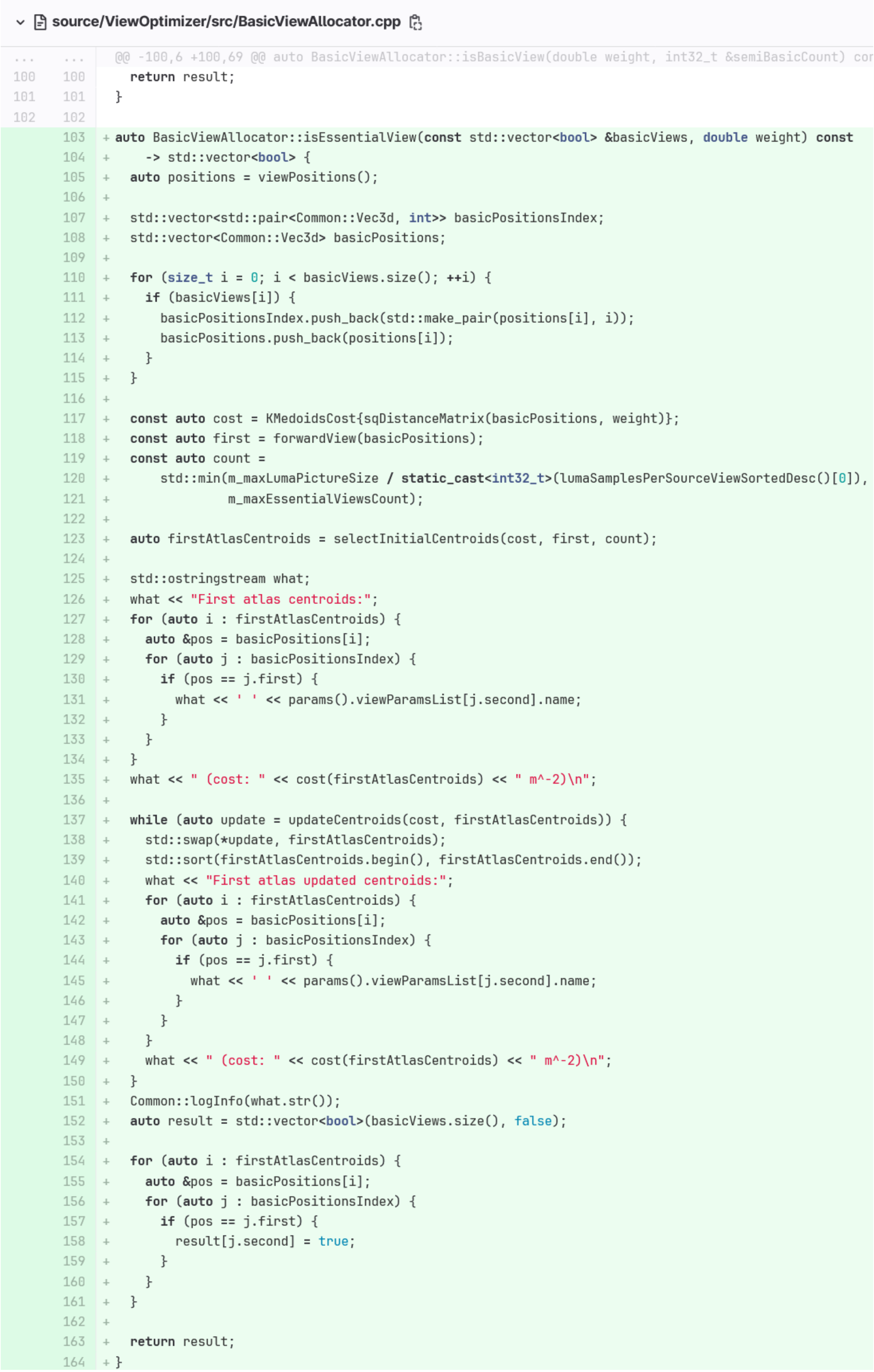
source/ViewOptimizer/src/AbstractViewSelector.cpp

```diff
 auto AbstractViewSelector::optimizeParams(const SourceParams &params) -> ViewOptimizerParams {
 @@ -58,15 +72,25 @@ auto AbstractViewSelector::optimizeParams(const SourceParams &params) -> ViewOpt
   }
 
   m_isBasicView = isBasicView(weight, m_semiBasicCount);
+  if (m_outputEssentialViews) {
+    m_isEssentialView = isEssentialView(m_isBasicView, weight);
+  }
 
   for (size_t i = 0; i < m_params.viewParamsList.size(); ++i) {
     m_params.viewParamsList[i].isBasicView = m_isBasicView[i];
+    if (m_outputEssentialViews) {
+      m_params.viewParamsList[i].isEssentialView = m_isEssentialView[i];
+      if (m_isEssentialView[i]) {
+        m_params.viewParamsList[i].isBasicView = false;
+      }
+    }
   }
   m_params.semiBasicCount = m_semiBasicCount;
 
   printSummary();
```



source/ViewOptimizer/src/AbstractViewSelector.cpp

```diff
 void AbstractViewSelector::inplaceEraseAdditionalViews(std::vector<T> &views) const
   PRECONDITION(views.size() == m_isBasicView.size());
-  if (!m_outputAdditionalViews) {
-    for (int32_t i = static_cast<int32_t>(views.size()) - 1; i >= 0; --i) {
-      if (!m_isBasicView[i]) {
-        views.erase(views.begin() + i);
+  if (!m_outputEssentialViews) {
+    if (!m_outputAdditionalViews) {
+      for (int32_t i = static_cast<int32_t>(views.size()) - 1; i >= 0; --i) {
+        if (!m_isBasicView[i]) {
+          views.erase(views.begin() + i);
+        }
       }
     }
+  } else {
+    std::vector<T> essentialViews;
+    if (!m_outputAdditionalViews) {
+      for (int32_t i = static_cast<int32_t>(views.size()) - 1; i >= 0; --i) {
+        if (m_isEssentialView[i]) {
+          essentialViews.push_back(views[i]);
+        }
+      }
+      for (int i = static_cast<int>(views.size()) - 1; i >= 0; --i) {
+        if (!m_isBasicView[i] || m_isEssentialView[i]) {
+          views.erase(views.begin() + i);
+        }
+      }
+      std::reverse(essentialViews.begin(), essentialViews.end());
+      views.insert(views.begin(), essentialViews.begin(), essentialViews.end());
+    }
```



source/ViewOptimizer/src/BasicViewAllocator.cpp

```diff
@@ -100,6 +100,69 @@ auto BasicViewAllocator::isBasicView(double weight, int32_t &semiBasicCount) cor
   return result;
 }
 
+auto BasicViewAllocator::isEssentialView(const std::vector<bool> &basicViews, double weight) const
+    -> std::vector<bool> {
+  auto positions = viewPositions();
+
+  std::vector<std::pair<Common::Vec3d, int>> basicPositionsIndex;
+  std::vector<Common::Vec3d> basicPositions;
+
+  for (size_t i = 0; i < basicViews.size(); ++i) {
+    if (basicViews[i]) {
+      basicPositionsIndex.push_back(std::make_pair(positions[i], i));
+      basicPositions.push_back(positions[i]);
+    }
+  }
+
+  const auto cost = KMedoidsCost{sqDistanceMatrix(basicPositions, weight)};
+  const auto first = forwardView(basicPositions);
+  const auto count =
+      std::min(m_maxLumaPictureSize / static_cast<int32_t>(lumaSamplesPerSourceViewSortedDesc()[0]),
+               m_maxEssentialViewsCount);
+
+  auto firstAtlasCentroids = selectInitialCentroids(cost, first, count);
+
+  std::ostringstream what;
+  what << "First atlas centroids:";
+  for (auto i : firstAtlasCentroids) {
+    auto &pos = basicPositions[i];
+    for (auto j : basicPositionsIndex) {
+      if (pos == j.first) {
+        what << ' ' << params().viewParamsList[j.second].name;
+      }
+    }
+  }
+  what << " (cost: " << cost(firstAtlasCentroids) << " m^-2)\n";
+
+  while (auto update = updateCentroids(cost, firstAtlasCentroids)) {
+    std::swap(*update, firstAtlasCentroids);
+    std::sort(firstAtlasCentroids.begin(), firstAtlasCentroids.end());
+    what << "First atlas updated centroids:";
+    for (auto i : firstAtlasCentroids) {
+      auto &pos = basicPositions[i];
+      for (auto j : basicPositionsIndex) {
+        if (pos == j.first) {
+          what << ' ' << params().viewParamsList[j.second].name;
+        }
+      }
+    }
+    what << " (cost: " << cost(firstAtlasCentroids) << " m^-2)\n";
+  }
+  Common::logInfo(what.str());
+  auto result = std::vector<bool>(basicViews.size(), false);
+
+  for (auto i : firstAtlasCentroids) {
+    auto &pos = basicPositions[i];
+    for (auto j : basicPositionsIndex) {
+      if (pos == j.first) {
+        result[j.second] = true;
+      }
+    }
+  }
+
+  return result;
+}
```

Configuration changes – already implemented :

“decoderSideDepthEstimationFlag” (mandatory):

- A65: false
- G65: true
- IDMA: true

“maxGeometryAtlases” (optional, by default = maxAtlases):

- IDMA: 1

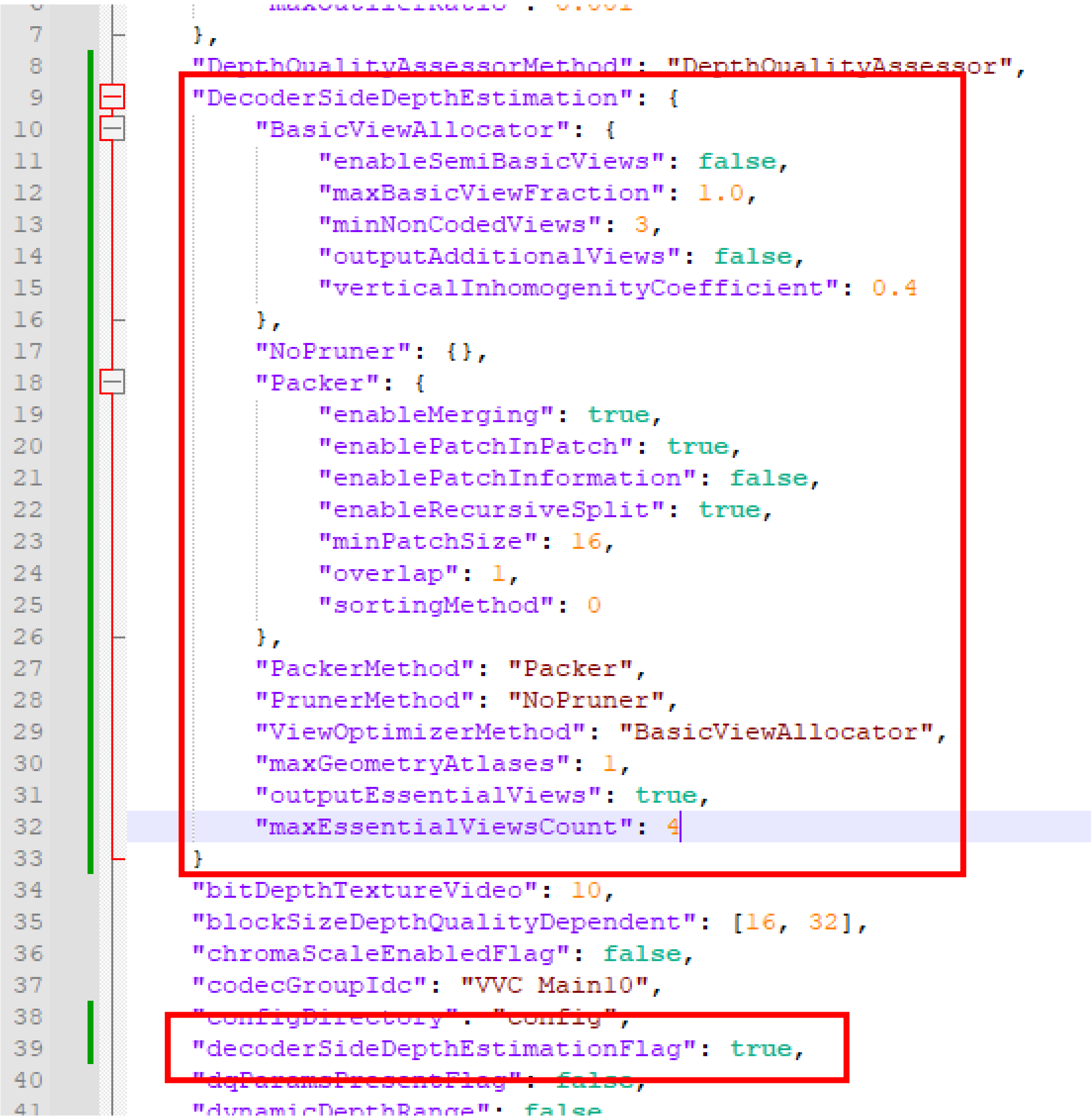
“ViewAllocator”:

- “outputEssentialViews” (optional, by default: false)
  - IDMA: true
- “maxEssentialViewsCount” (optional)
  - IDMA: 4

Compatibility:

A65 and G65 bitstreams are exactly the same as for TMIV17

Possible configuration changes:



```
        "maxOutlierRatio": 0.001
    },
    "DepthQualityAssessorMethod": "DepthQualityAssessor",
    "DecoderSideDepthEstimation": {
        "BasicViewAllocator": {
            "enableSemiBasicViews": false,
            "maxBasicViewFraction": 1.0,
            "minNonCodedViews": 3,
            "outputAdditionalViews": false,
            "verticalInhomogenityCoefficient": 0.4
        },
        "NoPruner": {},
        "Packer": {
            "enableMerging": true,
            "enablePatchInPatch": true,
            "enablePatchInformation": false,
            "enableRecursiveSplit": true,
            "minPatchSize": 16,
            "overlap": 1,
            "sortingMethod": 0
        },
        "PackerMethod": "Packer",
        "PrunerMethod": "NoPruner",
        "ViewOptimizerMethod": "BasicViewAllocator",
        "maxGeometryAtlases": 1,
        "outputEssentialViews": true,
        "maxEssentialViewsCount": 4
    }
    "bitDepthTextureVideo": 10,
    "blockSizeDepthQualityDependent": [16, 32],
    "chromaScaleEnabledFlag": false,
    "codecGroupIdc": "VVC Main10",
    "configDirectory": "config",
    "decoderSideDepthEstimationFlag": true,
    "dqParamsPresentFlag": false,
    "dynamicDepthRange": false
```

2 Recommendation

We recommend integrating the proposal into TMIV 18.

3 Acknowledgement

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00207, Immersive Media Research Laboratory).