**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG/N18068**

**October 2018, Macau SAR, CN**

|  |  |
| --- | --- |
| **Source** | Video |
| **Status** | Approved |
| **Title** | Reference View Synthesizer (RVS) manual |
|  |  |

# Introduction

This document describes the Reference View Synthesis (RVS) software that has been created for the 3DoF+ investigation. The Common Test Conditions [1] describe objective evaluation and expert viewing on views synthesized by RVS.

RVS overcomes most of the common challenges of view synthesis by using more input views. Common view synthesis datasets contain a huge number of views and corresponding depth maps. However, many depth image-based rendering [2] methods limit themselves in using a low number of reference views to synthesize virtual ones. Moreover, the baseline between the input views is often required to be small, restricting the view synthesis to positions close to the reference cameras. RVS overcomes this baseline problem thanks to multiple input views.

The second objective of RVS is to render objects with very tangential surfaces to the camera optical axis. For those surfaces, very few information about depth and color is available per pixel and thus, sometimes, the foreground might be put behind the background in synthesized views. Overcoming this problem, RVS is the select candidate to address the CTC properly.

This document is based on version 3.0 of RVS, which was built using SVS [3], originally developed by l’Université Libre de Bruxelles (ULB) - the French wing of Brussels University - in the context of the 3DLicorneA project, funded by Innoviris, Belgium. SVS was selected as a basis for RVS, because of its ability to have an arbitrary number of input views and showed also good performance on the ULB Unicorn and Technicolor data sets.

RVS 3.0 is a pure software architectural improvement over RVS 2.0.1 meaning that neither synthesis quality nor computational complexity have changed. It is planned to carry out experiments at the beginning of the AHG period. When successful, in the sense that a full CTC comparison delivers improved objective results, 3.1 will be released. The software library and command-line interfaces will remain compatible with this manual for all 3.*x*[.*y*] versions. Work on RVS 3.1 will concentrate on four areas: CPU speedup, exactness of OpenGL acceleration, blending quality and de-occlusion masking quality.

Table 1 provides a short overview of improvements per RVS version. Each version is tagged in git with a v*x*.*y*[.*z*] label. Please consult the git log for a complete overview of all minor changes.

Table 1: Progression of versions of RVS

|  |  |
| --- | --- |
| 1.0 | * The first full version of RVS according to the San Diego CTC. |
| 1.0.1 | * Complexity reduction by changing CTC configuration files at a small loss in terms of WS-PSNR: Precision 3 🡪 1. |
| 1.0.2 | * Restoration of 1.0 performance by small fixes to the image warping method. |
| 2.0 | * OpenGL acceleration * JSON metadata format extension   + Crop regions to support Padded ERP (input/output)   + Perspective projection incl. camera intrinsics * Simplify RVS configuration file (also JSON) * Removing compatibility for VSRS from RVS and into scripts extra/JSON\_to\_VSRS.py and extra/VSRS\_to\_JSON.py |
| 2.0.1 | * Minor fixes |
| 3.0 | * Extension of RVS for the Software Platform (Section 5: RVS API) * Support for masked output depth maps (key: MaskedDepthOutputFiles) * Support for YUV 4:0:0 depth I/O (key: DepthColorSpace) * Invalid depth value logic made optional (key: HasInvalidDepth) |

The RVS documentation consists of the following parts:

* This document:
  + Section 1: Introduction
  + Section 2: Getting started,
  + Section 3: Description of the method,
  + Section 4: Description of the configuration files,
  + Section 5: Description of the RVS API,
  + Section 6: Unit and integration tests,
  + Section 7: External libraries,
  + Section 8: Software coordination,
  + Appendix A: Linux/UNIX build script.
* Doxygen[[1]](#footnote-1) code comments,
* Example configuration files and generator,
* Other scripts in the extra/ source directory.

# Getting started

RVS accepts multiple input views (real cameras) and synthesizes and blends those to multiple output views (virtual cameras). There are two types of outputs:

* *Regular* output is inpainted (especially at the image edges) and may contain stretched textures for disoccluded regions where none of the input views have information.
* *Masked* output uses a texture validity metric to mark invalid pixels and paint them as neutral gray.

Regular output is more suitable for subjective testing (highest WS-PSNR against a ground truth reference), while masked output is designed with objective testing in mind.

## Building RVS

### Prerequisites

To build RVS it is necessary to have:

* A modern C/C++ compiler toolset with full C++11 support, such as:
  + GCC[[2]](#footnote-2) 4.9.x or newer,
  + Microsoft Visual Studio[[3]](#footnote-3) 2015 (VC14) or newer,
* CMake[[4]](#footnote-4) 3.7 or newer,
* OpenCV 3.x or newer.

OpenGL acceleration additionally requires OpenGL and glm libraries. The reader is referred to Section 7 for an informative overview of software licenses of external libraries.

We have confirmed that integration tests pass on the following combinations:

* Linux 64-bit + GCC 4.9.2 + OpenCV 3.4.1 (no OpenGL),
* Windows 10 64-bit + VC14 + OpenCV 3.1.0 (no OpenGL),
* Windows 10 64-bit + VC15 + OpenCV 3.4.1 + NVIDIA Quadro K420 (OpenGL).

### Obtaining the source code

The source code is located in the MPEG Git repository:

|  |  |
| --- | --- |
| URL | <http://mpegx.int-evry.fr/software/MPEG/Explorations/3DoFplus/RVS.git> |
| Tag | v3.0 |

MPEG experts may request access to the RVS project by sending an e-mail to:

* Bart Kroon [bart.kroon@philips.com](mailto:bart.kroon@philips.com) or
* The MPEG-I Visual reflector [mpeg-i-visual@lists.aau.at](mailto:mpeg-i-visual@lists.aau.at).

See Section 8 for rules on software coordination on improving RVS.

### Microsoft Windows

For Visual Studio it is possible to download an OpenCV installer with prebuilt libraries. After that, open the CMake GUI for RVS and click Configure. It will likely fail because it cannot find OpenCV. In that case click on the three dots next to OpenCV\_DIR and select the OpenCV directory that contains the OpenCV cmake files, e.g. C:\OpenCV-3.4.1\share\OpenCV. Then click Configure again and then Generate and Open Project. In Visual Studio select the Release build and press Ctrl+Shift+B to build the project.

### Linux/UNIX

For Linux OpenCV may be available as a system library, but it is generally more reliable (and faster) to build OpenCV specifically for the purpose of building RVS, while reducing system library dependencies to a minimum. Appendix A provides a suitable build script that results in an RVS build with support for raw streams and PNG images only.

## Running RVS

RVS is a command-line tool that expects the path to a configuration file as single command-line argument with optional flags.

# RVS CONFIGURATION\_file [OPTIONS…]

With options:

|  |  |
| --- | --- |
| --noopengl | OpenGL acceleration is enabled by default when built against OpenGL and glm libraries. It is possible to turn OpenGL off at runtime by adding this option. |
| --analyzer | This option enables the output of all intermediate synthesis and blending maps for analysis of RVS performance. |

Section 4.2 provides an overview of all parameters that can be specified in configuration files. RVS will print out all active parameters of the configuration file and content metadata file such that log files provide an accurate representation of the test conditions.

This CLI interface makes it possible to drag-and-drop a configuration file on a RVS shortcut, and makes it straightforward to run multiple configurations:

# find –name \\*.cfg | xargs RVS

See extra/compute\_job for a PBS[[5]](#footnote-5) compute server example.

## Configuration files

The software comes with example configuration files in the config\_files/ directory. Configuration files that match the CTC description [1] are included in config\_files/*sequence* directories. These files have been generated using the Python script extra/generate\_CTC.py.

# Description of the method

## View synthesis

### Image warping in SVS

In the original SVS [3], for each perspective camera input, a translation and rotation between each of them and the perspective target cameras is applied. The translation is obtained by moving each pixel according to its disparity and the rotation is obtained with a homography, where each pixel is displaced following the rotation of the camera. For a camera movement in the image plane, for each pixel , the translation in the image is given by the disparity

(1)

where is the translation vector of the camera, *d* is the depth at the pixel *p* and *f* is the focal length. In a movement along the Z axis of the camera, we have

(2)

Three degrees of freedom of translation are obtained by combining the two equations. Applying the affine transformation , a map is obtained, indicating the new position of each pixel in the new warped image.

The input view is divided in triangles with the pixels centers as vertices. The triangles are deformed using the affine transformation before being filled with interpolated colors. The colors are the tri-linear interpolations between each three vertices of the triangle. Discontinuities between objects creating disocclusions, and tangential surfaces may lead to triangles with very elongated shapes as we can see in the figure below. They will not be kept in the final result, as they get eliminated during the blending phase (see Section 3.2).



### Image warping in RVS

The largest difference between SVS and RVS, is that view synthesis in RVS is performed in a way that allows for generic projections:

1. Calculate a floating-point image coordinate map,
2. Unproject image coordinates to world coordinates with the input camera being the reference frame,
3. Apply a single affine transformation to make the virtual camera the reference frame,
4. Project the world coordinates onto the virtual image,
5. Warp the image according to the resulting map using the triangles method.

Currently two projection types are defined: perspective projection (Section 3.1.4) and equirectangular projection (Section 3.1.5).

### Coordinate system

The RVS world coordinate system is that of MPEG-I OMAF[[6]](#footnote-6).

* points forward (the reference direction for a viewer),
* points left,
* points up,

Hereby is the notation for Cartesian unit vectors such that. For an untransformed camera the origin is the cardinal point.

The definition of image coordinates is:

* The top-left corner is (0, 0),
* points right,
* points down.

### Perspective projection

Perspective projection requires an intrinsic matrix:

(3)

where all variables are in pixel units.

**Unprojection**

Taking into account the change of coordinate system, the projection equation is

(4)

where is the image position in pixel units.

**Projection**

The matching projection equation is

(5)

where is depth in meters and is the world position in meters. Please note that depth is typically stored as normalized disparities based on a configurable depth range, however in above equation is a real length.

### Equirectangular projection

For equirectangular projection the image is mapped on a horizontal angular range and vertical angular angle as specified in the JSON content metadata file.

**Unprojection**

For an image size, the spherical coordinates are:

(6)

(7)

The *ray direction* is:

(8)

and the world position is:

(9)

Whereby is the *ray length* which is the equivalent of depth for perspective projection. Please note that also ray length is stored as normalized disparities based on a configurable ray length range, however in the above equation is a real length.

**Projection**

The ray length and ray direction are trivially determined as

(10)

(11)

making use of the fact that valid ray lengths are

Finally, spherical angles are then estimated from:

(12)

(13)

Please note that the only difference between equirectangular projection and other omnidirectional projections is the mapping between spherical coordinates and image coordinates.

### Camera extrinsics

For 3DoF+ test materials the extrinsics are directly read from the JSON metadata files as a position and rotation vector. The Euler angles are internally converted to a rotation matrix according to:

Pose traces are comma-separated value files with the same six columns as the CTC tables and JSON metadata files: X, Y, Z, Yaw, Pitch, Roll.

The two rotations and two translations to transform from an input camera to a virtual camera are combined into a single affine transformation: x 🡪 Rx + t.

## View blending

All the synthesized images, corresponding to each input view, will be blended together. This is done by comparing a per-pixel quality, determined by its depth and the shape of the triangle it lays in: a pixel of good quality has a low depth and belongs to a triangle with a regular shape.

Taking the pixel with the maximal quality would give a sharper result, while taking the weighted mean is more resistant to errors in the depth maps, color differences between the input views. With the multispectral blending method, high and low frequencies are separated to apply the most adapted blending to each frequency. The low frequencies are blended with the weighted mean, and the high frequencies are blended by choosing the pixel of highest weight. The low frequencies are obtained by applying a mean blur with kernel size of 10% of the image size. The final color of a pixel is hence:

(14)

With (15)

(16)

where *n* is the number of input views, the depth at this pixel for the synthesized view , the color of the pixel in view *i* for the low and high frequencies, the quality of the triangle the pixel lays in, and a parameter. The factor prioritizes foreground objects even if they do not appear in all the input views. The factor removes the elongated triangles. Hence, the disocclusions can be filled with data from other input views. The quality must satisfy where when the triangle keeps its shape during the warping phase.

## Inpainting

A very simple inpainting is done. We propagate the color of the nearest pixel using a Manhattan distance. This could be improved but was not the main interest so far.

# Description of the configuration files

The input views and virtual views are described using a JSON format that is extended from the 3DoF+ CfTM. The view synthesis task is described in the RVS configuration file.

## JSON content metadata format

The test material including camera parameters are described by a single JSON file. Such files are included in the config\_files/ source directory for the ClassroomVideo, TechnicolorMuseum, TechnicolorHijack, and ULB Unicorn test materials. Subsequent sections provide some examples followed by a description of the parameters.

RVS 3.0 is able to read RVS 2.0 content metadata files.

### Equirectangular projection example

This is the first part of the TechnicolorMuseum file:

{

"Version": "3.0",

"Content\_name": "TechnicolorMuseum",

"BoundingBox\_center": [0, 0, 1.65],

"Fps": 30,

"Frames\_number": 300,

"Informative": {

"Cameras\_number": 24,

"RigRadius": 0.3

},

"cameras": [{

"Name": "v0",

"Position": [0.0777, -0.1429, 1.3872],

"Rotation": [-61.4676, 40.3759, 0],

"Depthmap": 1,

"Background": 1,

"Resolution": [2048, 2048],

"Hor\_range": [-90, 90],

"Ver\_range": [-90, 90],

"Projection": "Equirectangular",

"Depth\_range": [0.5, 25],

"BitDepthColor": 10,

"BitDepthDepth": 10,

"ColorSpace": "YUV420",

"DepthColorSpace": "YUV420"

}, {

…

### Perspective projection example

This is the first part of the file generated from Plane\_B'/cams\_unicorn\_vsrs.txt using a first generator, now replaced by Python scripts. The reason for the unit change is that depth ≥ 1000 (meters) is defined as infinity.

{

"Version": "3.0",

"Content\_name": "ULB\_Unicorn",

"BoundingBox\_center": [0, 0, 0],

"Fps": 1,

"Frames\_number": 1,

"Informative": {

"Converted\_by": "VSRS\_to\_JSON.m",

"Original\_units": "mm",

"New\_units": "m"

},

"cameras": [{

"Name": "cam\_000003250438",

"Position": [0.0511, -0.001832, -0.04965],

"Rotation": [-0.5563, 0.72404, 0.66459],

"Depthmap": 1,

"Background": 0,

"Depth\_range": [0.5, 2],

"Resolution": [1920, 1080],

"Projection": "Perspective",

"Focal": [1094.7, 1094.7],

"Principle\_point": [940.2, 540.3],

"BitDepthColor": 8,

"BitDepthDepth": 16,

"ColorSpace": "YUV420",

"DepthColorSpace": "YUV420"

}, {

…

### Viewport camera for pose traces (viewport) videos

To synthesize viewport videos according to a pose trace (path) a suitable viewport camera has to be added that determines the neutral position, resolution, field of view, etc.:

{

"Name": "viewport",

"Position": [0, 0, 1.65],

"Rotation": [0, 0, 0],

"Depthmap": 1,

"Background": 1,

"Resolution": [2048, 2048],

"Projection": "Perspective",

"Focal": [1024, 1024],

"Principle\_point": [1024, 1024],

"Depth\_range": [0.5, 25],

"BitDepthColor": 10,

"BitDepthDepth": 10,

"ColorSpace": "YUV420",

"DepthColorSpace": "YUV420"

}

### Center view camera

It is also useful to define a central view with the field of view of the scene unless such a source view already exists (like v0 of ClassroomVideo). For TechnicolorMuseum:

{

"Name": "center",

"Position": [0, 0, 1.65],

"Rotation": [0, 0, 0],

"Depthmap": 1,

"Background": 1,

"Resolution": [4096, 2048],

"Projection": "Equirectangular",

"Hor\_range": [-180, 180],

"Ver\_range": [-90, 90],

"Depth\_range": [0.5, 25],

"BitDepthColor": 10,

"BitDepthDepth": 10,

"ColorSpace": "YUV420",

"DepthColorSpace": "YUV420"

}

### Description of the parameters

Table 2: Description of the parameters of the content metadata format

|  |  |
| --- | --- |
| Version | The version of this format as a string to allow future software to interpret these files correctly. |
| Content\_name | The name of the test material. |
| BoundingBox\_center | The center of the viewing zone in meters. The metadata files included with RVS have a “viewport” and “center” camera at this position. |
| Fps | Frame rate in Hz |
| Frames\_number | Number of frames per sequence |
| Informative | Everything under this key will remain unspecified |
| Cameras | An array of cameras typically including source views and intermediate views. |
| Name | The camera name, e.g. “v0” |
| Position | The position [x, y, z] in meters with x forward, y left and z up. |
| Rotation | The rotation [yaw, pitch, roll] in degrees |
| Depthmap | Required to be 1 by RVS. |
| Background | Ignored by RVS. |
| Resolution | The number of pixels in a view [width, height] |
| Crop\_region | RVS supports padded ERP in/out through this parameter. Crop\_region specifies a region of interest (ROI) in the frame. Focal and Principle\_point (perspective projection) relate to the full (uncropped) frame. Hor\_range and Ver\_range (equirectangular projection) relate to the cropped frame. |
| Projection | Currently Equirectangular or Perspective. RVS is easily extensible for other projections. |
| Focal | The focal length [fx, fy] in pixels for perspective projection. |
| Principle\_point | The principle point [px, py] in pixels for perspective projection. |
| Hor\_range | Horizontal angular range in degrees for equirectangular projection. The maximum range is [-180, 180]. |
| Ver\_range | Vertical angular range in degrees for equirectangular projection. The maximum range is [-90, 90]. |
| Depth\_range | Depth maps are stored as normalized disparities with a range [near, far]. The definition of depth depends on the projection type. For perspective projection it is the projection of a world point on the forward axis of the camera [x\_near, x\_far]. For equirectangular projection, it is the radius from the cardinal point [r\_near, r\_far]. |
| HasInvalidDepth | true [default]: Depth level 0 indicates “invalid depth”.  false: Depth level 0 corresponds to “far”. |
| BitDepthColor | The bit depth for color images. RVS supports 1 to 16 bit fixed point values and 32-bit floating point (the internal format). |
| BitDepthDepth | The bit depth for depth maps. RVS supports 1-16 bit normalized disparities and 32-bit floating point depth (the internal format). |
| ColorSpace | The color space for raw texture streams. RVS only supports “YUV420”. This field does not apply to image files such as PNG and OpenEXR. |
| DepthColorSpace | The color space for raw depth streams. RVS supports “YUV420” and “YUV400”. This field does not apply to image files such as PNG and OpenEXR. |

## RVS configuration file

The RVS 3.0 configuration file is also based on JSON and relies much more on the metadata files. Thanks to this the RVS 3.0 configurations are generally shorter than the RVS 1.0.2 files. This section provides a couple of examples followed by a description of all parameters.

RVS 3.0 is able to read RVS 2.0 configuration files.

### Typical configuration file

This first example synthesizes view v0 from v7 and v8 using default parameters.

{

"Version": "3.0",

"InputCameraParameterFile": "config\_files/ClassroomVideo.json",

"VirtualCameraParameterFile": "config\_files/ClassroomVideo.json",

"InputCameraNames": ["v7", "v8"],

"VirtualCameraNames": ["v0"],

"ViewImageNames": [

"ClassroomVideo/v7\_4096\_2048\_420\_10b.yuv",

"ClassroomVideo/v8\_4096\_2048\_420\_10b.yuv"],

"DepthMapNames": [

"ClassroomVideo/v7\_4096\_2048\_0\_8\_1000\_0\_420\_10b.yuv",

"ClassroomVideo/v8\_4096\_2048\_0\_8\_1000\_0\_420\_10b.yuv"],

"OutputFiles": ["v0vs\_from\_v7v8\_4096\_2048\_420\_10b.yuv"]

}

### Pose trace example

This second example is more elaborate and synthesizes a viewport video and 180º video according to the positions and rotations of a pose trace.

{

"Version": "3.0",

"InputCameraParameterFile": "config\_files/TechnicolorMuseum.json",

"VirtualCameraParameterFile": "config\_files/TechnicolorMuseum.json",

"InputCameraNames": ["v5"],

"VirtualCameraNames": ["center", "viewport"],

"VirtualOverrides": {

"Resolution": [2048, 2048],

"Hor\_range": [-90, 90]},

"VirtualPoseTraceName": "config\_files/(…)/TechnicolorMuseum-PoseTrace.csv",

"ViewImageNames": ["TechnicolorMuseum/v5\_2048\_2048\_420\_10b.yuv"],

"DepthMapNames": ["TechnicolorMuseum/v5\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv"],

"OutputFiles": [

"TechnicolorMuseum\_PoseTrace.yuv",

"TechnicolorMuseum\_PoseTrace\_Viewport.yuv"],

"NumberOfFrames": 300

}

### Description of the parameters

Version (string)

Corresponds to the version of RVS “3.0”. This field allows future versions of RVS to interpret how these files can be read.

InputCameraParameterFile (string)

Name of the JSON content metadata file which includes input cameras parameters.

CameraNames (array of *N* strings)

Names (identifiers) of the *N* input cameras in InputCameraParameterFile to be used for synthesizing each virtual view. *N* inputs views result in *M* virtual output views. The names have to match with the Name field.

ViewImageNames (array of *N* strings)

File paths of the input color images in same order as CameraNames. The input images can be in YUV format or PNG/JPEG/EXR/etc. format.

DepthMapNames (array of *N* strings)

File paths of the input depth maps in same order as CameraNames. The input images have to be in YUV format or PNG/JPEG/EXR/etc. format.

VirtualCameraParameterFile (string)

File path of the text file which includes the output cameras parameters. Can be the same as InputCameraParameterFile because the metadata files may provide both source and intermediate view positions.

VirtualCamerasNames (array of *M* strings)

Names (identifiers) of the virtual cameras in VirtualCameraParameterFile that are to be synthesized. *N* inputs views result in *M* virtual output views.

OutputFiles (array of *M* strings)

File paths of the regular (inpainted) output virtual views. When absent this type of output is omitted.

MaskedOutputFiles (array of *M* strings)

File paths of the masked output files (for inpainting). When absent this type of output is omitted.

OutputMasks (array of *M* strings)

File paths of the output masks (for inpainting). When absent this type of output is omitted.

DepthOutputFiles (VirtualCameraNumber string)

File paths of the output depth maps (for inpainting). When absent this type of output is omitted.

MaskedDepthOutputFiles (VirtualCameraNumber string)

File paths of the output depth maps with de-occlusion regions masked gray. When absent this type of output is omitted.

ValidityTheshold (float)

Threshold on the validity of a pixel to inpaint it. The default value is 5000.

Precision (int)

Internal oversampling factor. Working image size = Precision \* virtual size. The default value is 1. Computational complexity and memory consumption of RVS goes up with the square of Precision. This parameter has no effect on the OpenGL code path.

ColorSpace (string)

Working color space, either “YUV” or “RGB”. The default value is “YUV”.

ViewSynthesisMethod (string)

View synthesis method. For now only “Triangles” is supported.

BlendingMethod (“Simple”/” MultiSpectral”)

Blending method to blend the synthesized views from all the output. (Default Simple).

BlendingFactor (float)

When BlendingMethod is set to Simple. If set to -1, the blending will keep only the best pixel across the views. If set to , the color will be the weighted mean over the views:

The default value is 5.0.

BlendingLowFreqFactor (float)

When BlendingMethod is set to Multispectral. Blending factor for low frequencies.

BlendingHighFreqFactor (float)

When BlendingMethod is set to Multispectral. Blending factor for high frequencies.

StartFrame (int)

Start frame (Default 0).

NumberOfFrames (int)

Number of frames (Default 1).

VirtualPoseTraceName (string)

File path to a pose trace to induce a camera transformation for each frame and view. This functionality is used to mimic typical behavior of a viewer with a headset. A pose trace file is a comma-separated file with six columns: X, Y, Z, Yaw, Pitch Roll that match in definition with the Position and Rotation camera parameters. The header has to match otherwise RVS will reject the file.

InputOverrides (object)

Any key-value pairs specified in this object will override those keys of the camera parameters. For instance, to switch to 16-bit depth maps it is not necessary to create another metadata file. Instead it is possible to add InputOverrides { BitDepthDepth 16 } to the RVS configuration file. Other uses may be switching input resolutions.

VirtualOverrides (object)

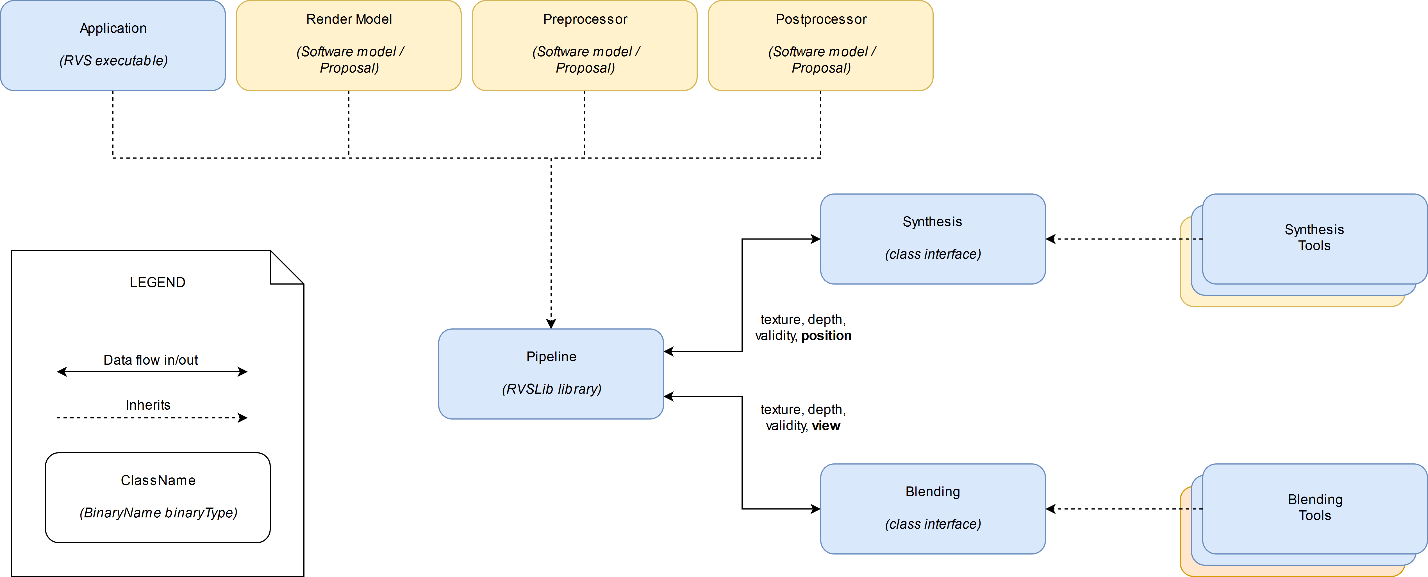
Similar to InputOverrides this allows for overriding virtual camera parameters. In above example it is used to reduce the field of view of the center view to 180º by overriding Resolution and Hor\_range.

# Description of the RVS API

From 3.0 onwards RVS can be used as a library instead of an executable (Figure 1). The main application (RVS) has the same performance as before, but proponents can create software that links against RVSLib to interface with RVS at a lower level. This allows for:

* In-memory operations (reducing disk usage),
* Access to floating point color, texture, quality and validity maps,
* Access to OpenGL pixel buffer objects when OpenGL support is enabled,
* Analysis of intermediate synthesis and blending result,
* Adding new projection types (for instance fish eye cameras).

The idea is that evidence and proposals may use RVS in more creative ways without altering RVS itself.



*Figure 1: Extension of RVS for the Software Platform*

## RVS Application

The main class of RVS is called Pipeline. In the branch this class is split in three classes:

* rvs::Pipeline (part of RVSLib) is the processing model but performs no I/O
* rvs::Application (part of RVS executable) derives from rvs::Pipeline and performs file I/O
* rvs::Analyzer (part of RVS executable) derives from rvs::Application and outputs intermediate results to disc for analysis.

Calling RVS from the command-line constructs the rvs::Application except when --analyzer is passed in which case the rvs::Analyzer is constructed. Apart from the --analyzer option the command-line interface of RVS and the behavior of the RVS application has not changed.

## RVS Library

* + 1. **Namespaces**

We have introduced namespaces to indicate classes of RVS that are part of the RVSLib interface. The rvs and json namespaces are part of the interface and may change only for each major version of RVS. The rvs::detail namespace is not part of the interface and may change for each minor version.

* The rvs namespace contains most of the RVS logic.
* The json namespace contains the JSON parser and is generically applicable.
  + 1. **Pipeline**

The main class is rvs::Pipeline. The idea is to derive from this class and overload multiple methods. The rvs::Application and rvs::Analyzer classes serve as examples of how to use rvs::Pipeline.

|  |  |
| --- | --- |
| getConfig | Interface to provide access to the configuration.  Implemented by Application using file I/O. |
| loadInputView | Interface for loading a source view.  Implemented by Application using file I/O. |
| wantColor | Does the derived class want to save color?  Implemented by Application to depend on configuration file. |
| wantMaskedColor | Does the derived class want to save masked color?  Implemented by Application to depend on configuration file. |
| wantMask | Does the derived class want to save the validity mask itself?  Implemented by Application to depend on configuration file. |
| wantDepth | Does the derived class want to save the depth map?  Implemented by Application to depend on configuration file. |
| wantMaskedDepth | Does the derived class want to save the depth map?  Implemented by Application to depend on configuration file. |
| saveColor | Interface for saving a regular (inpainted) synthesis result.  Implemented by Application using file I/O. |
| saveMaskedColor | Interface for saving a masked synthesis result.  Implemented by Application using file I/O. |
| saveMask | Interface for saving the validity mask.  Implemented by Application using file I/O. |
| saveDepth | Interface for saving the synthesized depth map.  Implemented by Application using file I/O. |
| saveMaskedDepth | Interface for saving a masked version of the synthesized depth map.  Implemented by Application using file I/O. |
| onIntermediate… SynthesisResult | Interface for making intermediate result available for pruning or analysis. Pipeline calls this function after synthesizing each single input view. |
| onIntermediate… BlendingResult | Interface for making intermediate result available for pruning or analysis. Pipeline calls this function after synthesizing a single input view. |
| onFinal… BlendingResult | Interface for making the final blending result available for pruning or analysis at floating-point precision. In case of OpenGL the view data is on the GPU as pixel buffer objects. In case of CPU the view data is passed as a parameter. |
| createBlender | Create a view blender to blend the warped input views to one virtual view. The default implementation supports the RVS 3.0 blending tools but the RVS API allows a proponent to add a blending tool without altering RVS. |
| createSynthesizer | Create a single-view synthesizer. The default implementation supports the 3.0 synthesis tool but this interface allows a proponent to add a blending tool without altering RVS. |
| createSpaceTransformer | By extending the rvs::SpaceTransformer it is possible to mix in new projection types such as fish eye cameras. The RVS 3.0 projections are equirectangular and perspective. |

# Unit and integration tests

When changing versions of prerequisites, it is important to run the tests. While the unit tests check specific aspects of the code, the integration tests are based on achieving PSNR thresholds when synthesizing from multiple source views to a destination view (possibly the same).

The tests require the ULB Unicorn Plane B' with Kinect depth maps and (the first frames of) some views of the 3DoF+ sequences [1] to be placed in subdirectories of the RVS project. The metadata files are included with the RVS source for convenience.

Required ULB Unicorn data files should be placed at:

Plane\_B'/Plane\_B'\_Texture/

Kinect\_z0300y0307x0370.yuv

Kinect\_z0300y0325x0438.yuv

Kinect\_z0300y0343x0506.yuv

Plane\_B'/Plane\_B'\_Depth/

Kinect\_z0300y0307x0370.yuv

Kinect\_z0300y0343x0506.yuv

One early test case ULB\_Unicorn\_10b requires 10-bit conversions of ULB Unicorn Plane B'. Those files can be generated using HDRTools according to the configurations supplied with [1].

Required 3DoF+ data files are:

ClassroomVideo/

v0\_4096\_2048\_420\_10b.yuv v0\_4096\_2048\_0\_8\_1000\_0\_420\_10b.yuv

v7\_4096\_2048\_420\_10b.yuv v7\_4096\_2048\_0\_8\_1000\_0\_420\_10b.yuv

v8\_4096\_2048\_420\_10b.yuv v8\_4096\_2048\_0\_8\_1000\_0\_420\_10b.yuv

TechnicolorMuseum/

v0\_2048\_2048\_420\_10b.yuv v0\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv

v2\_2048\_2048\_420\_10b.yuv v2\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv

v5\_2048\_2048\_420\_10b.yuv v5\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv

v6\_2048\_2048\_420\_10b.yuv

v13\_2048\_2048\_420\_10b.yuv v13\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv

v17\_2048\_2048\_420\_10b.yuv v17\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv

v19\_2048\_2048\_420\_10b.yuv v19\_2048\_2048\_0\_5\_25\_0\_420\_10b.yuv

TechnicolorHijack/

v1\_4096\_4096\_420\_10b.yuv v1\_4096\_4096\_0\_5\_25\_0\_420\_10b.yuv

v4\_4096\_4096\_420\_10b.yuv v4\_4096\_4096\_0\_5\_25\_0\_420\_10b.yuv

v9\_4096\_4096\_420\_10b.yuv

To run the integration tests,

* Linux/UNIX: make all test
* Visual Studio: build ALL\_BUILD and RUN\_TESTS.

# External libraries

This section provides an informative overview of software licenses. The purpose is to facilitate a study of this aspect, but no rights can be derived from the information in this section.

* RVS is designed to *dynamically* link to the OpenCV library. A small non-controversial subset of OpenCV (see Appendix A) is used extensively throughout the code base. Removing this dependency is possible but neither ULB nor Philips wish to undertake this large task.
* Optional OpenGL acceleration will require:
  + Dynamically linking to a system-supplied OpenGL implementation and the glm library.
  + glLoadGen has been used to generate a source and header file.
  + Optional performance analysis using RenderDoc will include a header file.

# Software coordination

Software coordinator:

Bart Kroon

[bart.kroon@philips.com](mailto:bart.kroon@philips.com)

Any improvements during the AHG period (with support on the MPEG-I Visual reflector) will result in version 3.1, 3.2, etc. Minor fixes will be numbered 3.0.1, 3.0.2, etc. The major number is increased with each meeting cycle unless nothing changes.

Participants that wish to obtain RVS may request the software coordinator for Reporter access (read only).

Participants that wish to improve RVS should write an input document describing the improvement and provide evidence for the improvement. As soon as the input document (m*xxxxx*) is registered, a participant may request the software coordinator for Developer (read/write) access to the RVS git project and to create a branch feature/m*xxxxx* based on the current release. After that the participant may work on the branch and possibly coordinate the work with other participants. Participants shall not work on the v*x*.*y*-dev branches nor on the master.

Then name RVS refers both the project, the executable and the library. When using the software interface to add tools to RVS, then these tools and their use have to be clearly described. We recommend that content providers that provide new test material with other types of projections, keep these in a separate software project that uses RVS as a library. We recommend that contributions that (also) improve RVS in other ways contribute a branch to RVS.

Upon acceptance of multiple improvements of RVS, say m1, m2, m3, etc., it is decided in which order the branches will be merged into the next release (3.0). The software coordinator will merge the first branch (m1) into the development branch (v3.0-dev) running tests and possibly making corrections. Then the second participant (m2) should merge v3.0-dev into m2 and contact the software coordinator as soon as this is ready. The software coordinator will then merge the updated second branch (m2) into the development branch, etc. When ready the software coordinator will inform the group that the v3.0-dev branch is ready for reviewing and then 3.0 will be released.

All branches that relate to input contributions will be deleted after the MPEG meeting.

# Acknowledgements

The original authors of RVS are:

* Sarah Fachada [Sarah.Fernandes.Pinto.Fachada@ulb.ac.be](mailto:Sarah.Fernandes.Pinto.Fachada@ulb.ac.be)
* Daniele Bonatto [Daniele.Bonatto@ulb.ac.be](mailto:Daniele.Bonatto@ulb.ac.be)
* Arnaud Schenkel [arnaud.schenkel@ulb.ac.be](mailto:arnaud.schenkel@ulb.ac.be)
* Bart Kroon [bart.kroon@philips.com](mailto:bart.kroon@philips.com)
* Bart Sonneveldt [bart.sonneveldt@philips.com](mailto:bart.sonneveldt@philips.com)

|  |  |
| --- | --- |
| Université libre de Bruxelles  <http://www.ulb.ac.be/> | This work is supported by Innoviris, the Brussels Institute for Research and Innovation, Belgium, under contract numbers 2015-DS-39a/b & 2015-R-39c/d, 3DLicorneA. |
| Koninklijke Philips N.V.  <https://www.philips.com/> | This work is part of Philips’ research into 3DoF+ and 6DoF video. |

# Appendix A: Linux/UNIX build script

This example script creates a minimal build of OpenCV and RVS for use on compute servers. There is support for PNG images, but not for other types of images or OpenGL.

#!/bin/bash

**trap** exit ERR

**set** -x

root=*$PWD*

OpenCV\_url=https://github.com/opencv/opencv

OpenCV\_tag=3.4.1

RVS\_url=http://mpegx.int-evry.fr/software/MPEG/Explorations/3DoFplus/RVS.git

RVS\_tag=v3.0

# Checkout OpenCV

[ -d "*$root*/OpenCV.source" ] || **git** clone *$OpenCV\_url* "*$root*/OpenCV.source"

**cd** "*$root*/OpenCV.source"

**git** checkout *$OpenCV\_tag*

# Configure OpenCV

**mkdir** -p "*$root*/OpenCV.build"

**cd** "*$root*/OpenCV.build"

**cmake** "*$root*/OpenCV.source" \

-DBUILD\_SHARED\_LIBS:BOOL=ON \

-DBUILD\_LIST:STRING=core,imgcodecs,imgproc,photo \

-DBUILD\_PNG=ON \

-DBUILD\_ZLIB=ON \

-DWITH\_CUDA=OFF \

-DWITH\_OPENCL=OFF \

-DWITH\_OPENGL=OFF \

-DWITH\_IPP=OFF \

-DWITH\_LAPACK=OFF \

-DWITH\_GTK=OFF \

-DWITH\_1394=OFF \

-DWITH\_TIFF=OFF \

-DWITH\_JPEG=OFF \

-DWITH\_JASPER=OFF \

-DWITH\_WEBP=OFF \

-DWITH\_OPENEXR=OFF \

-DENABLE\_PRECOMPILED\_HEADERS=OFF \

-DCMAKE\_INSTALL\_PREFIX:PATH="*$root*/OpenCV"

# Build OpenCV

**make** -j16

**make** -j16 install

# Checkout RVS

[ -d "*$root*/RVS.source" ] || **git** clone *$RVS\_url* RVS.source

**cd** "*$root*/RVS.source"

**git** checkout *$RVS\_tag*

# Configure RVS

**mkdir** -p "*$root*/RVS.build"

**cd** "*$root*/RVS.build"

**cmake** "*$root*/RVS.source" -DOpenCV\_DIR="*$root*/OpenCV/share/OpenCV"

# Build RVS

**cd** "*$root*"

**make** -j16 -C "*$root*/RVS.build"

**ln** -sf "*$root*/RVS.build/RVS" .

# References

[1] *Common Test Conditions on 3DoF+ and Windowed 6DoF*, ISO/IEC JTC1/SC29/WG11 MPEG/N18089, October 2018, Macau SAR, CN.

[2] *Depth-image-based rendering (dibr), compression, and transmission for a new approach on 3d-tv*, Christoph Fehn, Stereoscopic Displays and Virtual Reality Systems XI. International Society for Optics and Photonics, 2004, vol. 5291, pp. 93–105.

[3] *Depth Image Based View Synthesis with Multiple Reference Views for Virtual Reality,* Sarah Fachada, Daniele Bonatto, Arnaud Schenkel, Gauthier Lafruit, July 2018, 3DTV-CON.

1. <https://www.stack.nl/~dimitri/doxygen/> [↑](#footnote-ref-1)
2. <https://gcc.gnu.org/> [↑](#footnote-ref-2)
3. <https://visualstudio.microsoft.com/> [↑](#footnote-ref-3)
4. <https://cmake.org/> [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Portable_Batch_System> [↑](#footnote-ref-5)
6. <https://mpeg.chiariglione.org/standards/mpeg-i/omnidirectional-media-format> [↑](#footnote-ref-6)