Vision SDK SUMat/UMat User Guide

ABSTRACT:

The document describes the SUMat/UMat container behavior inside VSDK.

KEYWORDS:

Vision SDK, APEX, ISP, DCU, SUMAT, UMAT

APPROVED:

AUTHOR	SIGN-OFF SIGNATURE #1	SIGN-OFF SIGNATURE #2
Rostislav Hulik		

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.8	10-March-17	Rostislav Hulik	First draft
1.0	30-March-17	Rostislav Hulik	First review done
1.1	6-september-17	Rostislav Hulik	OAL_Initialize/Deinitialize removed
1.2	13-February-18	Rostislav Hulik / Stephane Francois	Some clarifications around e0 and UMat
2.0	11-December-18	Rostislav Hulik	SUMat/UMat update

Table of Contents

Vision SDK SUMat/UMat User Guide 1

1 SUMat/UMat Concept 4

1.1 VSDK 1.2 Porting Guide 5

1.2 Application Library Dependency Explained 6

2 SUMat/UMat Basics 8

2.1 Buffer allocation 8

2.2 Element Access 9

2.3 ROI Operation, Padding 11

2.4 OpenCV Compatibility 13

2.5 Buffer Free 15

3 Tutorials 17

3.1 Basic use and OpenCV Compatibility 17

3.2 Memory allocation and its behavior 18

3.3 Memory mapping and access to SUMat/UMat memory 19

3.4 Memory mapping and access modifiers 20

3.6 SUMat/UMat in ACF 21

3.7 Use example with ACF 22

3.8 Use example with SDI 24

3.9 Use example with DCU 25

4 SUMat API 27

5 SMat API 35

6 UMat API 45

7 Mat API 48

1 SUMat/UMat Concept

The vsdk::SUMat image container is the data structure used in the whole VSDK to wrap data buffers. This is a virtual data container allowing for manipulating data which can be used by the host ARM core as well as by the hardware accelerators. The concept was taken from OpenCV (cv::UMat), however, SUMat class isn’t inherited from it and work without any reference to OpenCV. The implementation of SUMat is inspired from OpenCV cv::UMat with the added feature of providing Automotive quality metrics (MISRA, HIS) and using platform specific memory management for best performance when using hardware accelerators. The improved software quality feature forges the root for the name of the container: Safe Umat, in short SUMat.

To provide compatibility to OpenCV (use of the data containers in OCV functions) and keep access to hardware accelerators, the vsdk::UMat container should be used. This class inherits vsdk::SUMat and can be used in both, VSDK and OpenCV functions. In that case, by linking the application to OpenCV, it is much more difficult to guarantee safe operations in comparison to the use of SUMat. So, the use of UMat, and thus linking to OpenCV, is suggested only for prototyping and debug. For convenience, the SUMat/vsdk::UMat and vsdk::UMat/cv::UMat retypes can be cast arbitrarily.

vsdk::UMat/SUMat are using OAL allocator[1], platform dedicated memory allocator, internally so the buffer in is ensured to be contiguous, which means the buffers are compatible with all hardware accelerators on S32V234.

Classes summary:

`vsdk::SUMat`	`SUMat`	VSDK SUMat implementation
`vsdk::SMat`	`SMat`	VSDK Mat implementation
`vsdk::UMat`	`UMat`	VSDK UMat implementation
`vsdk::Mat`	`Mat`	VSDK Mat implementation
`cv::UMat`	`cv::UMat`	Original OpenCV UMat class
`cv::Mat`	`cv::Mat`	Original OpenCV Mat class

The SUMat/UMat and SMat/Mat behavior is identical except for the OpenCV compatibility, it means that vsdk::UMat/vsdk::Mat and vsdk::SUMat/vsdk::SMat are interchangeable. The next chapters of this document will only refer to vsdk::SUMat/vsdk::SMat, but it means that it also applies to vsdk::UMat/vsdk::Mat.

[1] Operation System Abstraction Layer (OAL) is a library which aims to provide portable OS-specific functions to be used in VSDK applications. The functions such as memory allocation, mutex mechanisms, task support and others are provided by the OAL in order to abstract the OS approach and provide common interface to the application. Please refer to VisionSDK_OAL_API_Specification.pdf for further details.

1.1 VSDK 1.2 Porting Guide

Following paragraph describes porting effort from older VSDK versions, where only vsdk::UMat container was present.

1.1.1 Safe application

Following lines describe porting of existing application (VSDK version < 1.3) to new SUMat/UMat in VSDK 1.3 while introducing safe code and removing OpenCV.

- Remove all `umat.hpp` includes and replace them by `sumat.hpp`
- Remove all OpenCV includes
- Replace all `vsdk::UMat` by `vsdk::SUMat`
- Replace all `vsdk::Mat` by `vsdk::SMat`
- Remove `$(SDK_ROOT)/libs/utils/umat/$(ODIR)/libumat.a` from BUILD.mk
- Add `$(SDK_ROOT)/libs/utils/sumat/$(ODIR)/libsumat.a into` BUILD.mk

1.1.2 OpenCV compatible application (minimal changes)

Following changes needs to be introduces while user wants to keep minimal changes into the application between VSDK version < 1.3 and VSDK 1.3:

- Add `$(SDK_ROOT)/libs/utils/sumat/$(ODIR)/libsumat.a into` BUILD.mk

1.2 Application Library Dependency Explained

1.2.1 Safe application

In order to compile safe application, only SUMat/SMat need to be used without any OpenCV function linked in. The SUMat/SMat are compatible with all VSDK base classes.

Main include file

```
#include “sumat.hpp”
```

Classes available

```
vsdk::SUMat
vsdk::SMat
```

Main library to be linked in (BUILD.mk)

```
$(SDK_ROOT)/libs/utils/sumat/$(ODIR)/libsumat.a
```

1.2.2 OpenCV compatible application

The OpenCV compatible application needs to have both, UMat/Mat and OpenCV linked in. UMat can be used in all VSDK base classes and can be recast to cv::UMat and be used in OpenCV functions.

Main include file

```
#include “umat.hpp”
```

Classes available

```
vsdk::SUMat
vsdk::SMat
vsdk::UMat
vsdk::Mat
cv::UMat
cv::Mat
```

Main library to be linked in (BUILD.mk)

```
$(SDK_ROOT)/libs/utils/sumat/$(ODIR)/libsumat.a  // necessary
$(SDK_ROOT)/libs/utils/umat/$(ODIR)/libumat.a    // necessary
-lopencv_core                                     // necessary
-lopencv_*                                        // any other opencv
                                                     lib needed
```

2 SUMat/UMat Basics

2.1 Buffer allocation

2.1.1 Related functions

```
vsdk::SUMat/UMat constructor
```

2.1.2 Behavior

Buffer allocation

- The constructor allocates the buffer specified in the parameters (size + type).
- The buffer is allocated **without virtual mapping**. This means it can be used as pure DMA buffer in ISP or ACF.
- If access from ARM is requested, an instance of vsdk::SMat must be created with specific flags.

Reference count

- Reference count is increased!

2.1.3 Code example

```
/* Standard allocation */
vsdk::SUMat matrix0(HEIGHT, WIDTH, VSDK_CV_8UC3);

/* Allocation with forced memory bank
   DDR0, DDR1, Single banked SRAM, Multi banked SRAM */
vsdk::SUMat matrix1(HEIGHT, WIDTH, VSDK_CV_16SC1, vsdk::USAGE_DDR0);
vsdk::SUMat matrix2(HEIGHT, WIDTH, VSDK_CV_32SC3, vsdk::USAGE_DDR1);
vsdk::SUMat matrix3(HEIGHT, WIDTH, VSDK_CV_8SC2, vsdk::USAGE_SSRAM);
vsdk::SUMat matrix4(HEIGHT, WIDTH, VSDK_MAKETYPE(CV_8S, 8),
vsdk::USAGE_MSRAM);
```

2.2 Element Access

2.2.1 Related functions

```
vsdk::SUMat/UMat::getMat
```

2.2.2 Behavior

Element access from ARM side

- Creates an instance of the vsdk::SMat with specific virtual mapping for ARM access.
- vsdk::SMat contains accessors for elements, rows etc.
- Multiple vsdk::SMat instances can be created from SUMat, must have the same cache settings! Otherwise, the empty matrix is returned.
    - Destroy all the previous mapped matrices in order to map with different settings

Reference count

- Reference count is increased each time Matrix is created.
- Reference count is decreased each time Matrix is destroyed.

Cache management

- When last of the matrices are destroyed, the cache flush is called on the buffer
- WARNING about cache flushing of sub-matrices - the cache flush is performed through the continuous area - this means the buffer is flushed across the whole span of original matrix!

2.2.3 Code example

```
vsdk::SUMat sumat(HEIGHT, WIDTH, VSDK_CV_8UC3);

/* Standard mapping & access */
{
  // OK
  vsdk::SMat mat0 = sumat.getMat(vsdk::ACCESS_RW | OAL_USAGE_CACHED);

  // OK, creates second matrix, shared buffer!
  vsdk::SMat mat1 = sumat.getMat(vsdk::ACCESS_RW | OAL_USAGE_CACHED);

  // NOT OK, fails because there are existing different mappings!
  vsdk::SMat mat2 = sumat.getMat(vsdk::ACCESS_RW | OAL_USAGE_NONCACHED);

  // access of elements
  int8_t *ptr = mat0.data;
  mat0.at<vsdk::Vec3u>(j, i)[0] = VALUE;
}
/* During the end block, all mats are destroyed, so the memory buffer is
flushed */

{
  // Now it's OK, it remaps the buffer since there is no cached mapping.
  vsdk::SMat mat0 = sumat.getMat(vsdk::ACCESS_RW | OAL_USAGE_NONCACHED);
}

/* Direct access without keeping the mat */
sumat.getMat(vsdk::ACCESS_WRITE | OAL_USAGE_NONCACHED).at<uint8_t>(0)
      = VALUE;
```

2.3 ROI Operation, Padding

2.3.1 Related functions

```
vsdk::SUMat/UMat constructor
```

2.3.2 Behavior

ROI specification

- ROI or padding is done by specifying the larger buffer and cutting the ROI inside.

Padding specification

- Padding is specified using ROI API
- The buffer isn't deallocated until last submatrix lasts. By that, it's possible to specify larger matrix in the constructor.

Reference count

- Reference count is increased when constructor is called.
- If ROI specified in one command (see first example), the reference count is 1 for ROIed sub-matrix (the larger matrix is dereferenced.

2.3.3 Code example

```
/* Padding specification. In the constructor, it creates larger
matrix,
   then, it's used inside the ROI constructor and immediately
destroyed.
   Padded matrix remains. */
vsdk::SUMat matrix_padded(
    vsdk::SUMat(HEIGHT+2*PADDINGY, WIDTH+2*PADDINGX, VSDK_CV_8UC1),
    vsdk::Rect(PADDINGX, PADDINGY, WIDTH, HEIGHT));

/* Separate definition of multiple ROIs*/
vsdk::SUMat matrix_main(HEIGHT, WIDTH, VSDK_CV_8UC1);
vsdk::SUMat matrix_small0(matrix_main, vsdk::Rect(0, 0, WIDTH/2,
HEIGHT/2);
vsdk::SUMat matrix_small1(matrix_main, vsdk::Rect(WIDTH/2, HEIGHT/2,
WIDTH/2, HEIGHT/2);

/* Writing into padding - we write into padding area */
{
  vsdk::SMat mat = matrix_padded.getMat(vsdk::ACCESS_RW |
OAL_USAGE_CACHED);
  mat.at<uint8_t>(-1, -1) = VALUE;
}
```

2.4 OpenCV Compatibility

2.4.1 Related functions

```
UMat/Mat inheritance, cast operators
```

2.4.2 Behavior

retype from cv::Mat/cv::UMat

- The retype operators exists between cv::UMat/Mat and vsdk::UMat/Mat.
- The retype operators exists between vsdk::UMat/Mat and vsdk::SUMat/SMat
- The buffer position is checked when retyped - if the buffer in the original is not in the OAL allocated memory, the buffer is reallocated and data copied.
    - The reason for that is the vsdk::UMat/SUMat always assures the contiguous data buffer to be used in DMA.

OpenCV functions

- vsdk::UMat and vsdk::Mat can be used inside the native opencv functions.
- There is small restriction - it must be explicitly retyped to the cv type - the openCV functions doesnt' have the API for cv::UMat, but for abstract class from which the buffer is detected. The automatic retype doesn't work in these cases.

2.4.3 Code example

```
vsdk::UMat vsdkumat(HEIGHT, WIDTH, VSDK_CV_8UC3);
cv::UMat   cvumat(HEIGHT, WIDTH, VSDK_CV_8UC3);

/* Use of openCV function with combined buffers */
/* Explicit retype must be present due to
   InputArray/OutputArray abstract class in the functions */
/* No realloc is done here, the blur is done inside of vsdkumat */

cv::blur(cvumat, (cv::UMat)vsdkumat, cv::Size(3, 3));

/* Reading file/buffer via opencv with implicit realloc to OAL */
vsdk::UMat in = cv::Mat(256, 256, VSDK_CV_8UC1,
in).getUMat(cv::ACCESS_READ);
vsdk::UMat in = cv::imread("in.png", 0).getUMat(cv::ACCESS_READ);
```

```
vsdk::SUMat vsdksumat(HEIGHT, WIDTH, VSDK_CV_8UC3);
vsdk::UMat  vsdkumat = vsdksumat;

cv::UMat   cvumat(HEIGHT, WIDTH, VSDK_CV_8UC3);

/* Use of openCV function with combined buffers */
/* Explicit retype must be present due to
   InputArray/OutputArray abstract class in the functions */
/* No realloc is done here, the blur is done inside of vsdkumat */

cv::blur(cvumat, (cv::UMat)vsdkumat, cv::Size(3, 3));

/* Reading file/buffer via opencv with implicit realloc to OAL */
vsdk::UMat in = cv::Mat(256, 256, VSDK_CV_8UC1,
in).getUMat(cv::ACCESS_READ);
vsdk::UMat in = cv::imread("in.png", 0).getUMat(cv::ACCESS_READ);
```

2.5 Buffer Free

2.5.1 Related functions

```
vsdk::SUMat/UMat destructor
```

2.5.2 Behavior

Freeing the memory

- When last referenced SUMat/SMat or its submatrix is destroyed, the buffer is freed/released.
- This is done even if the buffer was saved in the outside pointer variable - only SUMat/SMat structures are checked for reference count.

Function parameters

- It is strongly recommended to pass the SUMat/SMat by value to all the functions.
- Using references, the reference count mechanism is bypassed, which means the reference check is left to the user. Spontaneous buffer deallocation can happen when multiple pointers reference the same SUMat/SMat.

Reference count

- Reference count is decreased when SUMat/SMat is destroyed.

2.5.3 Code example

```
{
  vsdk::SUMat subumat;
  {
    vsdk::SUMat sumat(HEIGHT, WIDTH, VSDK_CV_8UC3);
    subumat = vsdk::SUMat(sumat, vsdk::Rect(0, 0, ROIx, ROIy));
    {
        vsdk::SMat smat = sumat.getMat(vsdk::ACCESS_RW | 
OAL_USAGE_NONCACHED);
    } // here, the mat is destroyed, rf decreased, not freed

  } // Here, sumat is destroyed, But there's still subumat living as 
ROIed submatrix
} // Finally, all matrices are gone, deallocating.

/* Correct parameters */
void function_params0(vsdk::SUMat first, vsdk::SUMat &second);
void function_params1(vsdk::SMat first, vsdk::SMat &second);

/* Function calls */
/* Note the matrix is created when called the function.
   This flushes the "first" UMat when function returns */
/* It's not possible to do the same for second since it's
   the dereferenced parameter */
function_params1(first.getMat(vsdk::ACCESS_RW | OAL_USAGE_CACHED),
                 secondmat);
```

3 Tutorials

3.1 Basic use and OpenCV Compatibility

- The class has a public interface include/sumat.hpp and include/umat.hpp files, which can be included anywhere in the application code.
- The class and its dependencies are implemented in **libs/utils/sumat** and **libs/utils/umat** library
- When not defined differently, OpenCV still uses malloc allocator. However, vsdk::SUMat/UMat always ensures the contiguous data - if the non-contiguous data are detected, internal realloc is done and data are copied to the safe memory:

```
// Implicit type cast
{
    cv::UMat    image(YSIZE, XSIZE, CV_32SC1);
    vsdk::UMat  image_vsdk = image;
    vsdk::SUMat image_safe = image_vsdk;
    cv::UMat    image_cv = image_vsdk;
}
```

```
// If the read image isn't contiguous (allocated on heap) when retyped
to "image", this issue is detected, OAL memory is newly allocated and
data copied internally
// Fast data copy TBD
{
    vsdk::UMat image = cv::imread("in_color_256x256.png",
                       CV_LOAD_IMAGE_COLOR).getUMat(cv::ACCESS_RW);
    //   ... image is safe to be used by HW IP
    cv::imwrite("out_color_256x256.png", (cv::UMat)image);
}
```

3.2 Memory allocation and its behavior

- Memory is completely encapsulated in the SUMat/UMat class - there is no way how to pass an existing pointer to the constructor.
- The behavior is identical to the OpenCV implementation - when needed, the internal buffer is allocated via OAL and kept across all references until last SUMat/UMat instance with the same internal buffer is destroyed

```
{
    vsdk::SUMat image;        // An empty SUMat instance is created
    image = vsdk::SUMat(HEIGHT, WIDTH, DATA_TYPE);
                              // When created, memory is allocated via OAL
    {
        vsdk::SUMat image2 = image;
        // When assigned, no change happens in underlying structure,
        // reference count is increased
    }
} // When destroyed last reference, the data are freed from OAL heap
```

3.3 Memory mapping and access to SUMat/UMat memory

- The SUMat/UMat constructor does not map any memory into virtual memory space. It is only when vsdk::SMat/Mat (or cv::Mat) is created that a virtual mapping exists. Thanks to this, SUMat/UMat can be used purely by HW block without unnecessary mapping operations.
- Memory mapping is done by calling getMat(flags) method of SUMat/UMat.

```
// Simple data access from ARM
{
    vsdk::SMat image = sumat.getMat(OAL_USAGE_CACHED);
    // image is now accessible from ARM side by standard OpenCV accessors
    for (int i = 0; i < IMAGE_SIZE; ++i)
        image.at<char>(i) = 0;
}

// When image is destroyed (watch the lifetime!) and no other Mat instances
// are present (reference count), memory is tagged to be unmapped.
// Moreover, if the mapping was cached, the cache flush and invalidate
// operation is performed to ensure the data coherency.

// WARNING, Multiple different mappings are forbidden!
{
    vsdk::SMat image1 = sumat.getMat(OAL_USAGE_CACHED);
    vsdk::SMat image2 = sumat.getMat(OAL_USAGE_NONCACHED);
    // image2.data will be NULL in this part, image.empty() is true
}

// Correct usage:
{
    vsdk::SMat image1 = sumat.getMat(OAL_USAGE_CACHED);
} // when destroyed, matrix is flushed and unmapped
{
    vsdk::SMat image2 = sumat.getMat(OAL_USAGE_NONCACHED);
}

// WARNING, Watch the reference count:
{
    vsdk::SMat image1 = sumat.getMat(OAL_USAGE_CACHED);
    {
        vsdk::SMat image2 = sumat.getMat(OAL_USAGE_CACHED);
         // image2.data are ok
    } // memory is not unmapped, because there is still image1 living
    {
        vsdk::SMat image2 = sumat.getMat(OAL_USAGE_NONCACHED);
         // image2 is empty, because there is still living cached mapping
    }
}
```

3.4 Memory mapping and access modifiers

- The SUMat/UMat::getMat function also accepts the ACCESS_READ, ACCESS_WRITE and ACCESS_RW modifiers. This influences mainly the cache 'flush & invalidate' functions on mapping and unmapping.

The functionality is **NOT IMPLEMENTED** because OpenCV doesn't support it either. All Mat instances are mapped as ACCESS_RW.

```
vsdk::SMat image = umat.getMat(OAL_USAGE_CACHED | vsdk::ACCESS_READ);
// when destroyed, the umat data are not flusher (NOT IMPLEMENTED NOW)
```

3.6 SUMat/UMat in ACF

The APEX Core Framework is taking image buffer via vsdk::SUMat/UMat. The example below combines OpenCV image read, APEX processing and OpenCV image save:

```
// read the image via OpenCV, internally convert to vsdk UMat
// during conversion, the non-OAL memory is detected, UMat allocates OAL Memory
and copies data to be used in vsdk

vsdk::UMat image = cv::imread("in_color_256x256.png",
CV_LOAD_IMAGE_COLOR).getUMat(cv::ACCESS_RW);

if (!image.empty())
{
    // Init the rest of ports
    vsdk::UMat out(image.rows, image.cols, VSDK_CV_8UC3);
    vsdk::UMat dataThreshold(1, 1, VSDK_CV_8UC1);
    vsdk::UMat dataMarkColorChannel(1, 1, VSDK_CV_8UC1);

    // Init the algorithm parameters. Note the Mat is created just for this
    // call,
    // it's destroyed afterwards and flushed
    dataThreshold.getMat(OAL_USAGE_CACHED).at<unsigned char>(0) = THRESHOLD;
    dataMarkColorChannel.getMat(OAL_USAGE_CACHED).at<unsigned char>(0) =
                                                               COLOR_CHANNEL;

    // Init the ACF process
    APU_FAST9_COLOR process;

    lRetVal |= process.Initialize();
    lRetVal |= process.ConnectIO("INPUT", image);
    lRetVal |= process.ConnectIO("THRESHOLD", dataThreshold);
    lRetVal |= process.ConnectIO("MARK_COLOR_CHANNEL", dataMarkColorChannel);
    lRetVal |= process.ConnectIO("OUTPUT", out);

    // execute
    lRetVal |= process.Start();
    lRetVal |= process.Wait();

    // Save the output
    cv::imwrite("out_color_256x256.png", (cv::UMat)out);
}
```

3.7 Use example with ACF

3.7.1 Related functions

```
ACF_Process::ConnectIO
```

3.7.2 Behavior

Use in ACF Processes

- The ConnectIO interface in ACF allows for connecting SUMats/UMats to ACF Process Ports.
- ACF no longer flushes nor invalidates the connected SUMats/UMats prior to the ACF Process execution. The programmer must ensure no active SMat/Mat of the connected SUMats/UMats is mapped when ACF process starts.
- It is not possible to check the mapping of SUMat/UMat as there are currently no way to distinguish read-only access to write and read/write access because OpenCV 3.1 doesn’t support it.

ACF vs OpenCV incompatibilities

- As OpenCV doesn't have the integer type 32U, it is not possible to specify this datatype within SUMats/UMats. ACF was altered to not check the signess of 32-bit datatypes. This means that int32_t and uint32_t types are represented in SUMat/UMat by 32S type with no distinction.
- As OpenCV doesn't offer a way to express dimY, it has been added artificially into the vsdk::SUMat/UMat by using step[2]. It's not managed automatically in the constructors, as there is no interface to specify it. So, it must be handled manually (see 3.7.3 Code example below). Here is the description of the different level of Step:
  - Step[0] = Span, number of bytes to next line. Within ACF convention, it’s e0.y*e0.x*(element size)*width,
  - Step[1] = Horizontal size of the pixel in bytes. Within ACF convention, it’s e0.x*(element size),
  - Step[2] = Vertical size of the pixel. Within ACF convention, it is e0.y, it is equal 1 by the constructor and needs to be updated if e0.y is different than 1.
- It’s important to note that Step[2] is not detected and not compatible with OpenCV UMat. This means that when retyping vsdk::UMat to cv::UMat, Step[2] data are lost and the cv::UMat will have a corrupt size afterwards. In case the buffer needs to be translated to OpenCV, it is necessary to go back to the original constructor representation by having Step[2] back at 1 and the vertical dimension holding back the e0.y.

3.7.3 Code example

```
/* ACF process graph connection */
vsdk::SUMat input(HEIGHT, WIDTH, VSDK_CV_8UC1);
vsdk::SUMat output(HEIGHT, WIDTH, VSDK_CV_8UC1);

BLUR_PROCESS process;
process.Initialize();
process.ConnectIO("INPUT", input);
process.ConnectIO("THRESHOLD", output);
process.start();
process.wait();

/* Element dimension setting */
// Predefined X dim
vsdk::SUMat sumat0(HEIGHT, WIDTH, VSDK_CV_8UC1);

// Arbitrary X dim (after constructor, step[1] will be equal to
// WIDTH*16
vsdk::SUMat sumat1(HEIGHT, WIDTH, VSDK_CV_MAKETYPE(VSDK_CV_8U, 16);

// Arbitrary Y dim (non-automatic!!!)
vsdk::SUMat sumat2(HEIGHT * YDim, WIDTH, VSDK_CV_8UC1);

sumat2.rows    /= YDim;  // Rows are set to correct width
#ifdef APEX2_EMULATE
sumat2.step[0] *= YDim;  // Emulator compatibility
#endif
sumat2.step[2] *= YDim;   // Pixel height setup

// After those steps, the buffer can be attached correctly
// to the ACF

// Original DataDescriptor approach:
// icp::DataDescriptor(WIDTH, HEIGHT, icp::DATATYPE_8U, xDim, yDim);
```

3.8 Use example with SDI

3.8.1 Related functions

```
sdi_grabber::FramePop()
```

3.8.2 Behavior

Use in SDI

- The FramePop of the sdi_grabber now returns the SDI_Frame, which contains the SUMat instance.
- The returned SDI_Frame contains vital information of the buffer removed from the FDMA buffer pool.
- After the processing, the SDI_Frame must be pushed back to the buffer pool. **User is responsible of not using any of SUMat instances nor it's ROIs after the FramePush(SDI_Frame).**
    - The reason is that the buffer was put into the FDMA buffer pool again and will be rewritten by the camera feed.
    - On the other hand, it cannot be freed accidentally since SDI keeps the SUMat instance, so the reference won't decrease below 1 until SDI is destroyed

3.8.3 Code xample

```
/* SDI frame processing */
sdi_grabber lGrabber;

// ... SDI init

// Grabbing loop
while(1)
{
  SDI_Frame lFrame = lGrabber.FramePop();

  // UMat is in lFrame.UMat
  if (!lFrame.mUMat.empty()
  {
    // UMat processing
  }

  // We need to push the buffer back
  lGrabber.FramePush(lFrame);
}
```

3.9 Use example with DCU

3.9.1 Related unctions

```
FrameOutputDCU::PutFrame(SUMat)

FrameOutputV234Fb::PutFrame(SUMat)
```

3.9.2 Behavior

Use in DCU interface

- The DCU classes have the interface for putting the SUMat onto the screen.
- The SUMat parameter must have the same size as the DCU settings.
- TheS UMat must not be mapped into Mat! The DCU buffer doesn't flush the cache prior to display, which can lead on some platforms to the wrong displayed data.

3.9.3 Code example

```
/* DCU init */
#ifdef __STANDALONE__
    io::FrameOutputDCU
        output(1280,
               720,
               io::IO_DATA_DEPTH_08,
               CHNL_CNT);
#else
    io::FrameOutputV234Fb
        output(1280,
               720,
               io::IO_DATA_DEPTH_08,
               CHNL_CNT);
#endif

// Output buffer allocation
vsdk::SUMat output_sumat = vsdk::SUMat(720,
           1280,
           VSDK_CV_8UC3);

/* ... write into the output ... */

// output the buffer on the screen
output.PutFrame(output_sumat);
```

4 SUMat API

The following chapter describes the API of SUMat used in VSDK. Please note the API is similar to OpenCV, so the OpenCV documentation can be used:

cv::UMat reference

Please also note the vsdk SUMat is not inherited from these structures, so the API can be narrowed. The SUMat can be cast to UMat, from where conversion functions between vsdk and cv are available.

SUMat member	Parameters	Comment

`SUMat(`
`vsdk::UMatUsageFlags usageFlags =`
`vsdk::UMatUsageFlags::USAGE_DEFAULT);`
	IN | usageFlags | Usage flags, if need to be specified	Default constructor
		UMatUsageFlags can specify the memory pool where the sumat is allocated.
		• USAGE_DDR0
		• USAGE_DDR1
		• USAGE_SSRAM
		• USAGE_MSRAM

`SUMat(int32_t rows,`
`int32_t cols,`
`int32_t type,`
`vsdk::UMatUsageFlags`
`usageFlags =`
`vsdk::UMatUsageFlags::USAGE_DEFAULT);`
	IN | rows | Number of rows (height)
	IN | cols | Number of cols (width)
	IN | type | Type of element: VSDK_CV8UC1, VSDK_CV64FC3, VSDK_CV32SC(12) etc.
		Constructs 2D matrix of the specified size and type

	IN type Usage flags, if need to be specified	UMatUsageFlags can specify the memory pool where the sumat is allocated. • USAGE_DDR0 • USAGE_DDR1 • USAGE_SSRAM • USAGE_MSRAM

`SUMat(int32_t ndims,`
`     const int32_t* sizes,`
`     int32_t type,`
`     vsdk::UMatUsageFlags usageFlags`
`=`
`vsdk::UMatUsageFlags::USAGE_DEFAULT);`
	IN ndims Number of dimensions
	IN sizes Number of bytes in each dimension
	IN type Type of element: VSDK_CV8UC1, VSDK_CV64FC3, VSDK_CV32SC(12) etc.
	IN usageFlags Usage flags, if need to be specified
		Constructs n-dimensional matrix
		UMatUsageFlags can specify the memory pool where the sumat is allocated.
		• USAGE_DDR0
		• USAGE_DDR1
		• USAGE_SSRAM
		• USAGE_MSRAM

`SUMat(const vsdk::SUMat& m);`	IN m Original matrix	Copy constructor

`SUMat(const vsdk::SUMat& m,`
`      const vsdk::Range& rowRange,`
`      const vsdk::Range&`
`      colRange=Range::all());`
	IN m Original matrix
	IN rowRange Row range
	IN colRange Column range
		Creates a matrix header for a part of the bigger matrix

`SUMat(const vsdk::SUMat& m,`
`      const vsdk::Rect& roi);`
	IN m Original matrix
	IN roi ROI specified by a Rectangle class
		Creates a matrix header for a part of the bigger matrix

SUMat(const vsdk::SUMat& m, const vsdk::Range* ranges);	IN m Original matrix IN ranges Ranges specified by a list	Creates a matrix header for a part of the bigger matrix
vsdk::SUMat operator()(const vsdk::Range* ranges) const;	IN ranges Ranges specified by a list RETURN ROIed SUMat	Returns a SUMat from specified ROI
vsdk::SUMat& operator = (const SUMat& m);	IN m Original matrix RETURN Assigned matrix	Assign operator
vsdk::SUMat operator()(vsdk::Range rowRange, vsdk::Range colRange) const;	IN rowRange Original matrix row range IN colRange Original matrix column range RETURN Assigned submatrix	Assign ROI operator
vsdk::SUMat operator()(const vsdk::Rect& roi) const;	IN roi ROI rectangle RETURN Assigned submatrix	Assign ROI operator
~SUMat();		Destructor - calls release(), decrements the counter before freeing the buffer.
int32_t type() const;	RETURN Element type (similar to VSDK_CVMAT_TYPE)	Returns element type, similar to VSDK_CVMAT_TYPE(cvmat->type)
uint64_t total() const;	RETURN Total number of matrix elements	Returns the total number of matrix elements
uint64_t elemSize() const;	RETURN Total element size in bytes	

				Returns element size in bytes (e.g. 3 channel 16bit pixel will return 6)
int8_t isContinuous() const;	RETURN		Returns true **if** the matrix data is continuous	Returns true if the matrix data is continuous (i.e. when there are no gaps between successive rows). Similar to VSDK_CVIS_MAT_CONT(cvmat->type).
int8_t isSubmatrix() const;	RETURN		Returns true if the matrix is a submatrix of another matrix	Returns true if the matrix is a submatrix of another matrix
uint64_t elemSize1() const;	RETURN		Returns the size of element channel in bytes.	Returns the size of element channel in bytes (e.g. 3 channel 16 bit pixel will return 2).
int32_t depth() const;	RETURN		Returns element type, similar to VSDK_CVMAT_DEPTH(cvmat->type)	Returns element type, similar to VSDK_CVMAT_DEPTH(cvmat->type)
int32_t channels() const;	RETURN		Returns element type, similar to VSDK_CVMAT_CN(cvmat->type)	Returns element type, similar to VSDK_CVMAT_CN(cvmat->type)
uint64_t step1(int32_t i=0) const;	IN RETURN	i 	Step index Number of channels in the step	Returns step/elemSize1() - i.e. number of channels in the step
int8_t empty() const;	RETURN		Returns true if matrix data is NULL	Returns true if matrix data is NULL
vsdk::SUMat row(int32_t y) const;	IN RETURN	y 	Index of the row to be returned SUMat containing row ROI	

<table>
<tr><td></td><td></td><td>Returns a new matrix header for the specified row</td></tr>
<tr><td>vsdk::SUMat col(int32_t x) const;</td><td><table><tr><td>IN</td><td>x</td><td>Index of the column to be returned</td></tr><tr><td>RETURN</td><td></td><td>SUMat containing column ROI</td></tr></table></td><td>Returns a new matrix header for the specified column</td></tr>
<tr><td>vsdk::SUMat rowRange(
   int32_t startrow,
   int32_t endrow) const;</td><td><table><tr><td>IN</td><td>startrow</td><td>Starting row</td></tr><tr><td>IN</td><td>endrow</td><td>End row</td></tr><tr><td>RETURN</td><td></td><td>SUMat containing row ROI</td></tr></table></td><td>Returns a new matrix header for the specified row span</td></tr>
<tr><td>vsdk::SUMat rowRange(const vsdk::Range& r) const;</td><td><table><tr><td>IN</td><td>r</td><td>Range specifying row span</td></tr><tr><td>RETURN</td><td></td><td>SUMat containing row ROI</td></tr></table></td><td>Returns a new matrix header for the specified row span</td></tr>
<tr><td>vsdk::SUMat colRange(int32_t startcol,
                    int32_t endcol) const;</td><td><table><tr><td>IN</td><td>startcol</td><td>Starting column</td></tr><tr><td>IN</td><td>endcol</td><td>End column</td></tr><tr><td>RETURN</td><td></td><td>SUMat containing specified ROI</td></tr></table></td><td>Returns a new matrix header for the specified column span</td></tr>
<tr><td>vsdk::SUMat colRange(const vsdk::Range& r) const;</td><td><table><tr><td>IN</td><td>r</td><td>Column range</td></tr><tr><td>RETURN</td><td></td><td>SUMat containing specified ROI</td></tr></table></td><td>Returns a new matrix header for the specified column span</td></tr>
<tr><td>vsdk::SUMat diag(int32_t d=0) const;</td><td><table><tr><td>IN</td><td>d</td><td>Diagonal specification (see description)</td></tr><tr><td>RETURN</td><td></td><td>SUMat containing specified diagonal</td></tr></table></td><td>Returns a new matrix header for the specified diagonal

• d=0 - the main diagonal
• >0 - a diagonal from the lower half
• <0 - a diagonal from the upper half</td></tr>
</table>

<table>
<tr>
<td><code>int32_t checkVector(int32_t elemChannels,
int32_t depth=-1,
int8_t requireContinuous=true) const;</code></td>
<td>
<table>
<tr><td>IN</td><td>elemChannels</td><td>Query number of channels</td></tr>
<tr><td>IN</td><td>depth</td><td>Query depth</td></tr>
<tr><td>IN</td><td>requireContinuous</td><td>Query is continuous?</td></tr>
<tr><td>RETURN</td><td></td><td>Returns N if the matrix is 1-channel (N x ptdim) or ptdim-channel (1 x N) or (N x 1); negative number otherwise</td></tr>
</table>
</td>
<td>Returns N if the matrix is 1-channel (N x ptdim) or ptdim-channel (1 x N) or (N x 1); negative number otherwise.</td>
</tr>
<tr>
<td><code>vsdk::SMat getMat(int32_t flags) const;</code></td>
<td>
<table>
<tr><td>IN</td><td>flags</td><td>Access flags</td></tr>
<tr><td>RETURN</td><td></td><td>vsdk::SMat instance (mapped for access)</td></tr>
</table>
</td>
<td>Returns a Mat class with a concrete buffer mapping
<ul>
<li>The access flag must be always specified (note for v3.1 only RW flag is supported - all other access flags are rewritten inside the function)
<ul><li>ACCESS_READ</li><li>ACCESS_WRITE</li><li>ACCESS_RW</li></ul></li>
<li>The buffer mapping must be always specified, otherwise the empty matrix is returned.
<ul><li>OAL_USAGE_CACHED</li><li>OAL_USAGE_NONCACHED</li></ul></li>
</ul>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td><ul><li>Also, only one mapping can be present at the time. If different mapping is requested while there exist another Mat, the Mat returned will be empty. All Mats with different mapping must be destroyed before the call.</li></ul></td>
</tr>
<tr>
<td><code>int32_t rows, cols;</code></td>
<td><table><tr><td>MEMBER</td><td>rows</td><td>matrix height</td></tr><tr><td>MEMBER</td><td>cols</td><td>matrix width</td></tr></table></td>
<td></td>
</tr>
<tr>
<td><code>vsdk::UMatData* u;</code></td>
<td><table><tr><td>MEMBER</td><td>u</td><td>UMatData structure containing allocation info. Common structure for all derived matrices.</td></tr></table></td>
<td></td>
</tr>
<tr>
<td><code>vsdk::MatStep step;</code></td>
<td><table><tr><td>MEMBER</td><td>step[3]</td><td>Internal span dimensions:<ul><li>step[0] Row span in bytes (number of bytes between two vertical elements)</li><li>step[1] Element span in bytes (number of bytes between two neighboring elements)</li><li>step[2] Element Y span - number of rows containing to the one element</li></ul></td></tr></table></td>
<td>WARNING! The step[2] is added to the original OpenCV definition. When pixel Y size is needed, the matrix must be allocated with height*DimY and the step changed accordingly (not automatical!)<ul><li>SUMat matrix(height*dimY, 1, type);</li><li>matrix.step[0] *= dimY;</li><li>matrix.step[2] = dimY;</li><li>matrix.rows = matrix.rows/ dimY;</li></ul></td>
</tr>
</table>

			(pixel Y dimension)	

5 SMat API

The following chapter describes the API of SMat used in VSDK. Please note the API is similar to OpenCV, so the OpenCV documentation can be used:

cv::Mat reference

Please also note the vsdk SMat is not inherited from these structures, so the API can be narrowed. The SMat can be cast to Mat, from where conversion functions between vsdk and cv are available.

SMat member	Parameters	Comment

`SMat member function`
`SMat();`
		Constructor, creates and initializes the SMat
		These are various constructors that form a matrix. As noted in the Automatic Allocation, often the default constructor is enough, and the proper matrix will be allocated by an OpenCV function. The constructed matrix can further be assigned to another matrix or matrix expression or can be allocated with SMat::create. In the former case, the old content is de-referenced.

`SMat member function`
`SMat(int32_t rows,`
`    int32_t cols,`
`    int32_t type);`
	IN | rows | Number of rows (height)
	IN | cols | Number of cols (width)
	IN | type | Type of element: VSDK_CV_8UC1, VSDK_CV_64FC3, VSDK_CV_32SC(12) etc.
		Constructs 2D matrix of the specified size and type

SMat member function SMat(int32_t ndims, const int32_t* sizes, int32_t type);	IN	ndims	Number of dimensions	Constructs n-dimensional matrix
	IN	sizes	Number of bytes in each dimension	
	IN	type	Type of element: VSDK_CV_8UC1, VSDK_CV_64FC3, VSDK_CV_32SC(12) etc.	
SMat member function SMat(const vsdk::SMat& m);	IN	m	Original matrix	Array that (as a whole or partly) is assigned to the constructed matrix. No data is copied by these constructors. Instead, the header pointing to m data or its sub-array is constructed and associated with it. The reference counter, if any, is incremented. So, when you modify the matrix formed using such a constructor, you also modify the corresponding elements of m . If you want to have an independent copy of the sub-array, use cv::Mat::clone().
SMat member function SMat(int32_t rows, int32_t cols, int32_t type, void* data, uint64_t step=vsdk::SMat::AUTO_STEP);	IN	rows	Number of rows in a 2D array.	Creates a matrix based on existing buffer.
	IN	cols	Number of columns in a 2D array.	
	IN	type	Array type. Use VSDK_CV_8UC1, ..., VSDK_CV_64FC4 to create 1-4 channel matrices, or VSDK_CV_8UC(n), ..., VSDK_CV_64FC(n) to create multi-channel (up to VSDK_CV_CN_MAX channels) matrices.	
	IN	data	Pointer to the existing data	
	IN	step	Row step (number of bytes each matrix row occupies).	

			If the parameter is missing (set to AUTO_STEP), no padding is assumed and the actual step is calculated as cols*elemSize(). See SMat::elemSize.	

SMat member function
SMat(int32_t ndims,
    const int32_t* sizes,
    int32_t type,
    void* data,
    const uint64_t* steps=0);
	IN	ndims	Array dimensionality.	Creates a multi-dimensional matrix based on existing buffer.

	IN	sizes	Array of integers specifying an n-dimensional array shape.	
	IN	type	Array type (see above)	
	IN	data	Pointer to the existing data	
	IN	steps	Row steps (number of bytes each matrix row occupies). If the parameter is missing (set to AUTO_STEP), no padding is assumed and the actual step is calculated as cols*elemSize(). See SMat::elemSize.	

SMat member function
SMat(const vsdk::SMat& m,
    const vsdk::Range& rowRange,
    const vsdk::Range& colRange=vsdk::Range::all());
	IN	m	Original matrix	Creates a matrix header for a part of the bigger matrix

	IN	rowRange	Range of the m rows to take. As usual, the range start is inclusive and the range end is exclusive. Use Range::all() to take all the rows.	
	IN	colRange	Range of the m columns to take. Use Range::all() to take all the columns.	
SMat member function SMat(const vsdk::SMat& m, const vsdk::Rect& roi);	IN	m	Original matrix	Creates a matrix header for a part of the bigger matrix
	IN	roi	Region of interest to be taken into account.	

SMat member function SMat(const vsdk::SMat& m, const vsdk::Range* ranges);	IN m Original matrix IN ranges Array of selected ranges of m along each dimensionality.	Creates a matrix header for a part of the bigger matrix
SMat member function ~SMat();		Destructor - calls release()
SMat member function vsdk::SMat& operator = (const vsdk::SMat& m);	IN m Matrix to be assigned - right hand side.	Assignment operator.
SMat member function vsdk::SMat row(int32_t y) const;	IN Y A 0-based row index. RETURN New row matrix	Creates a matrix header for the specified matrix row.
SMat member function vsdk::SMat col(int32_t x) const;	IN X A 0-based column index. RETURN New column matrix	Creates a matrix header for the specified matrix column.
SMat member function vsdk::SMat rowRange(int32_t startrow, int32_t endrow) const;	IN startrow An inclusive 0-based start index of the row span. IN endrow An exclusive 0-based ending index of the row span. RETURN New ROI matrix	Creates a matrix header for the specified row span.
SMat member function vsdk::SMat rowRange(const vsdk::Range& r) const;	IN R Range structure containing both the start and the end indices. RETURN New ROI matrix	Creates a matrix header for the specified row span.
SMat member function vsdk::SMat colRange(int32_t startcol, int32_t endcol) const;	IN startcol An inclusive 0-based start index of the column span.	

	IN	endcol	An exclusive 0-based ending index of the column span.	Creates a matrix header for the specified column span.
	RETURN		New ROI matrix	
SMat member function vsdk::SMat colRange(const vsdk::Range& r) const;	IN	R	Range structure containing both the start and the end indices.	Creates a matrix header for the specified column span.
	RETURN		New ROI matrix	
SMat member function vsdk::SMat diag(int32_t d=0) const;	IN	d	Index of diagonal	Extracts a diagonal from a matrix. Index of the diagonal, with the following values (d value): • \`d=0\` is the main diagonal. • \`d>0\` is a diagonal from the lower half. For example, d=1 means the diagonal is set immediately below the main one. • \`d<0\` is a diagonal from the upper half. For example, d=-1 means the diagonal is set immediately above the main one.
	RETURN		New ROI matrix	

SMat member function
vsdk::SMat operator()(
    vsdk::Range rowRange,
    vsdk::Range colRange) const;
	IN	rowRange	Start and end row of the extracted submatrix. The upper boundary is not included. To select all the rows, use Range::all().	Extracts a rectangular submatrix.

	IN	colRange	Start and end column of the extracted submatrix. The upper boundary is not included. To select all the columns, use Range::all().	
	RETURN		New ROI matrix.	

SMat member function vsdk::SMat operator()(const vsdk::Rect& roi) const;	IN	roi	Extracted submatrix specified as a rectangle.	Extracts a rectangular submatrix.
	RETURN			
SMat member function vsdk::SMat operator()(const vsdk::Range* ranges) const;	IN	ranges	Array of selected ranges along each array dimension.	Extracts a rectangular submatrix.
	RETURN			
SMat member function int32_t type() const;	RETURN		Element type (similar to VSDK_CV_MAT_TYPE)	Returns element type, similar to VSDK_CV_MAT_TYPE(cvmat->type)
SMat member function uint64_t total() const;	RETURN		Total number of matrix elements	Returns the total number of matrix elements
SMat member function uint64_t elemSize() const;	RETURN		Total element size in bytes	Returns element size in bytes (e.g. 3 channel 16bit pixel will return 6)
SMat member function int8_t isContinuous() const;	RETURN		Returns true iff the matrix data is continuous	Returns true iff the matrix data is continuous (i.e. when there are no gaps between successive rows). Similar to VSDK_CV_IS_MAT_CONT(cvmat->type).
SMat member function int8_t isSubmatrix() const;	RETURN		Returns true if the matrix is a submatrix of another matrix	Returns true if the matrix is a submatrix of another matrix
SMat member function uint64_t elemSize1() const;	RETURN		Returns the size of element channel in bytes.	Returns the size of element channel in bytes (e.g. 3 channel 16 bit pixel will return 2).

SMat member function int32_t depth() const;	RETURN		Returns element type, similar to VSDK_CV_MAT_DEPTH(cvmat->type)	Returns element type, similar to VSDK_CV_MAT_DEPTH(cvmat->type)
SMat member function int32_t channels() const;	RETURN		Returns element type, similar to VSDK_CV_MAT_CN(cvmat->type)	Returns element type, similar to VSDK_CV_MAT_CN(cvmat->type)
SMat member function uint64_t step1(int32_t i=0) const;	IN	i	Step index	Returns step/elemSize1() - i.e. number of channels in the step
	RETURN		Number of channels in the step	
SMat member function int8_t empty() const;	RETURN		Returns true if matrix data is NULL	Returns true if matrix data is NULL
SMat member function int32_t checkVector(int32_t elemChannels, int32_t depth=-1, int8_t requireContinuous=true) const;	IN	elemChannels	Query number of channels	Returns N if the matrix is 1-channel (N x ptdim) or ptdim-channel (1 x N) or (N x 1); negative number otherwise.
	IN	depth	Query depth	
	IN	requireContinuous	Query is continuous?	
	RETURN		Returns N if the matrix is 1-channel (N x ptdim) or ptdim-channel (1 x N) or (N x 1); negative number otherwise	
SMat member function int32_t rows, cols;	MEMBER	rows	matrix height	
	MEMBER	cols	matrix width	

SMat member function uint8_t *data	MEMBER	data	Underlying virtual mapping for buffer	
SMat member function vsdk::UMatData* u;	MEMBER	u	UMatData structure containing allocation info. Common structure for all derived matrices.	
SMat member function vsdk::MatStep step;	MEMBER	step[3]	Internal span dimensions: • step[0] Row span in bytes (number of bytes to go to the next line) • step[1] Element span in bytes (number of bytes between two neighboring elements) • step[2] Element Y span - number of rows containing to the one element (pixel Y dimension)	WARNING! The step[2] is added to the original OpenCV definition. When pixel Y size is needed, the matrix must be allocated with height*DimY and the step changed accordingly (it is not automatic!) • UMat matrix(height*dimY, width, type); • *matrix.step[0] *= dimY;(emulator only)* • matrix.step[2] *= dimY; • matrix.rows /= dimY;

SMat member function
uint8_t* ptr(int32_t i0=0);
const uint8_t* ptr(int32_t i0=0) const;
	IN	i0	A 0-based row index.	Returns a pointer to the specified matrix row.

	RETURN		Pointer to the buffer	
SMat member function uint8_t* ptr(int32_t i0, int32_t i1); const uint8_t* ptr(int32_t i0, int32_t i1) const;	IN	i0	A 0-based row index.	Returns a pointer to the specified matrix row.
	IN	i1	A 0-based column index.	
	RETURN		Pointer to the buffer	

SMat member function uint8_t* ptr(int32_t i0, int32_t i1, int32_t i2); const uint8_t* ptr(int32_t i0, int32_t i1, int32_t i2) const;	IN i0 A 0-based row index. IN i1 A 0-based column index. IN i2 A 0-based channel index. RETURN Pointer to the buffer	Returns a pointer to the specified matrix row.
SMat member function uint8_t* ptr(const int32_t* idx); const uint8_t* ptr(const int32_t* idx) const;	IN idx A 0-based array of indices. RETURN Pointer to the buffer	Returns a pointer to the specified matrix row.
SMat member function template<typename _Tp> _Tp* ptr(int32_t i0=0);template<typename _Tp> const _Tp* ptr(int32_t i0=0) const;	IN i0 A 0-based row index. RETURN Pointer to the buffer	Returns a pointer to the specified matrix row.
SMat member function template<typename _Tp> _Tp* ptr(int32_t i0, int32_t i1); template<typename _Tp> const _Tp* ptr(int32_t i0, int32_t i1) const;	IN i0 A 0-based row index. IN i1 A 0-based column index. RETURN Pointer to the buffer	Returns a pointer to the specified matrix row.
SMat member function template<typename _Tp> _Tp* ptr(int32_t i0, int32_t i1, int32_t i2); template<typename _Tp> const _Tp* ptr(int32_t i0, int32_t i1, int32_t i2) const;	IN i0 A 0-based row index. IN i1 A 0-based column index. IN i2 A 0-based channel index. RETURN Pointer to the buffer	Returns a pointer to the specified matrix row.

SMat member function
template<typename _Tp>
_Tp& at(int32_t i0=0);
template<typename _Tp>
const _Tp& at(int32_t i0=0) const;
	IN i0 Index along the dimension 0
	RETURN Element at specified index
		Returns a reference to the specified array element.

SMat member function template<typename _Tp> _Tp& at(int32_t i0,	IN i0 Index along the dimension 0 IN i1 Index along the dimension 1	

int32_t i1);
template<typename _Tp>
const _Tp& at(int32_t i0,
int32_t i1) const;
	RETURN		Element at specified index	Returns a reference to the specified array element.

SMat member function template<typename _Tp> _Tp& at(int32_t i0, int32_t i1, int32_t i2); template<typename _Tp> const _Tp& at(int32_t i0, int32_t i1, int32_t i2) const;	IN IN IN RETURN	i0 i1 i2	Index along the dimension 0 Index along the dimension 1 Index along the dimension 2 Element at specified index	Returns a reference to the specified array element.
SMat member function template<typename _Tp> _Tp& at(const int32_t* idx); template<typename _Tp> const _Tp& at(const int32_t* idx) const;	IN RETURN	idx	Array of SMat::dims indices Element at specified index	Returns a reference to the specified array element.

SMat member function
template<typename _Tp>
_Tp& at(vsdk::Point pt);
template<typename _Tp>
const _Tp& at(vsdk::Point pt) const;
	IN
	RETURN
		pt	Element position specified as Point(j,i)
			Element at specified index
				Special versions for 2D arrays (especially convenient for referencing image pixels)

6 UMat API

The following chapter describes the API of UMat used in VSDK. Please note the API is similar to OpenCV, so the OpenCV documentation can be used:

cv::UMat reference

Please also note the vsdk UMat is inherited from vsdk::SUMat. Conversion functions between vsdk and cv are available. Following table does not contain inherited members.

SUMat member	Parameters			Comment

`UMat(`
`vsdk::UMatUsageFlags usageFlags =`
`vsdk::UMatUsageFlags::USAGE_DEFAULT);`
	IN	usageFlags	Usage flags, if need to be specified	Default constructor
				
				UMatUsageFlags can specify the memory pool where the sumat is allocated.
				
				• USAGE_DDR0
				• USAGE_DDR1
				• USAGE_SSRAM
				• USAGE_MSRAM

`UMat(int32_t rows,`
`     int32_t cols,`
`     int32_t type,`
`     vsdk::UMatUsageFlags`
`     usageFlags =`
`vsdk::UMatUsageFlags::USAGE_DEFAULT);`
	IN	rows	Number of rows (height)	Constructs 2D matrix of the specified size and type

	IN	cols	Number of cols (width)	
	IN	type	Type of element: VSDK_CV8UC1, VSDK_CV64FC3, VSDK_CV32SC(12) etc.	
	IN	type	Usage flags, if need to be specified	

<table>
<tr><td></td><td></td><td>UMatUsageFlags can specify the memory pool where the sumat is allocated.
• USAGE_DDR0
• USAGE_DDR1
• USAGE_SSRAM
• USAGE_MSRAM</td></tr>
<tr><td>UMat(int32_t ndims,
     const int32_t* sizes,
     int32_t type,
     vsdk::UMatUsageFlags usageFlags
=
vsdk::UMatUsageFlags::USAGE_DEFAULT);</td><td><table><tr><td>IN</td><td>ndims</td><td>Number of dimensions</td></tr><tr><td>IN</td><td>sizes</td><td>Number of bytes in each dimension</td></tr><tr><td>IN</td><td>type</td><td>Type of element: VSDK_CV8UC1, VSDK_CV64FC3, VSDK_CV32SC(12) etc.</td></tr><tr><td>IN</td><td>usageFlags</td><td>Usage flags, if need to be specified</td></tr></table></td><td>Constructs n-dimensional matrix
UMatUsageFlags can specify the memory pool where the sumat is allocated.
• USAGE_DDR0
• USAGE_DDR1
• USAGE_SSRAM
• USAGE_MSRAM</td></tr>
<tr><td>UMat(const vsdk::UMat& m);</td><td><table><tr><td>IN</td><td>m</td><td>Original matrix</td></tr></table></td><td>Copy constructor</td></tr>
<tr><td>UMat(const vsdk::SUMat& m);</td><td><table><tr><td>IN</td><td>m</td><td>Original matrix</td></tr></table></td><td>Copy & cast constructor</td></tr>
<tr><td>UMat(const cv::UMat& m);</td><td><table><tr><td>IN</td><td>m</td><td>Original matrix</td></tr></table></td><td>Copy & cast constructor</td></tr>
<tr><td>UMat(const vsdk::UMat& m,
      const vsdk::Range& rowRange,
      const vsdk::Range&
      colRange=Range::all());</td><td><table><tr><td>IN</td><td>m</td><td>Original matrix</td></tr><tr><td>IN</td><td>rowRange</td><td>Row range</td></tr><tr><td>IN</td><td>colRange</td><td>Column range</td></tr></table></td><td></td></tr>
</table>

		Creates a matrix header for a part of the bigger matrix

`UMat(const vsdk::UMat& m,`
`const vsdk::Rect& roi);`
	IN | m | Original matrix
	IN | roi | ROI specified by a Rectangle class
		Creates a matrix header for a part of the bigger matrix

`UMat(const vsdk::UMat& m,`
`const vsdk::Range* ranges);`
	IN | m | Original matrix
	IN | ranges | Ranges specified by a list
		Creates a matrix header for a part of the bigger matrix

`vsdk::UMat& operator =`
`(const SUMat& m);`
	IN | m | Original matrix
	RETURN | | Assigned matrix
		Assign operator

`vsdk::UMat& operator =`
`(const cv::UMat& m);`
	IN | m | Original matrix
	RETURN | | Assigned matrix
		Assign operator

`vsdk::UMat operator cv::UMat()`
`const;`
		Recast operator

`~UMat();`		Destructor - calls release(), decrements the counter before freeing the buffer.

7 Mat API

The following chapter describes the API of Mat used in VSDK. Please note the API is similar to OpenCV, so the OpenCV documentation can be used:

cv::Mat reference

Please also note the vsdk Mat is inherited from SMat, and conversion functions between vsdk and cv are available. The API list was narrowed to show only non-inherited members.

SMat member	Parameters	Comment

`Mat member function`
`Mat();`
		Constructor, creates and initializes the Mat
		These are various constructors that form a matrix. As noted in the Automatic Allocation, often the default constructor is enough, and the proper matrix will be allocated by an OpenCV function. The constructed matrix can further be assigned to another matrix or matrix expression or can be allocated with Mat::create. In the former case, the old content is de-referenced.

`Mat member function`
`Mat(int32_t rows,`
`    int32_t cols,`
`    int32_t type);`
	IN rows Number of rows (height)
	IN cols Number of cols (width)
	IN type Type of element: VSDK_CV_8UC1, VSDK_CV_64FC3, VSDK_CV_32SC(12) etc.
		Constructs 2D matrix of the specified size and type

`Mat member function`
`Mat(int32_t ndims,`
`    const int32_t* sizes,`
`    int32_t type);`
	IN ndims Number of dimensions
	IN sizes Number of bytes in each dimension
		Constructs n-dimensional matrix

	IN type Type of element: VSDK_CV_8UC1, VSDK_CV_64FC3, VSDK_CV_32SC(12) etc.	

`Mat member function`
`Mat(const vsdk::SMat& m);`
	IN m Original matrix	Array that (as a whole or partly) is assigned to the constructed matrix. No data is copied by these constructors. Instead, the header pointing to m data or its sub-array is constructed and associated with it. The reference counter, if any, is incremented. So, when you modify the matrix formed using such a constructor, you also modify the corresponding elements of m . If you want to have an independent copy of the sub-array, use cv::Mat::clone().

`Mat member function`
`Mat(const cv::Mat& m);`
	IN m Original matrix	Array that (as a whole or partly) is assigned to the constructed matrix. No data is copied by these constructors. Instead, the header pointing to m data or its sub-array is constructed and associated with it. The reference counter, if any, is incremented. So, when you modify the matrix formed using such a constructor, you also modify the corresponding elements of m . If you want to have an independent copy of the sub-array, use cv::Mat::clone().

`Mat member function`
`Mat(int32_t rows,`
`    int32_t cols,`
`    int32_t type,`
`    void* data,`
`    uint64_t`
`    step=vsdk::SMat::AUTO_STEP);`
	IN rows Number of rows in a 2D array.
	IN cols Number of columns in a 2D array.
	IN type Array type. Use VSDK_CV_8UC1, ..., VSDK_CV_64FC4 to create 1-4 channel matrices, or VSDK_CV_8UC(n), ..., VSDK_CV_64FC(n) to create multi-channel (up to
		Creates a matrix based on existing buffer.

			VSDK_CV_CN_MAX channels) matrices.	
	IN	data	Pointer to the existing data	
	IN	step	Row step (number of bytes each matrix row occupies). If the parameter is missing (set to AUTO_STEP), no padding is assumed and the actual step is calculated as cols*elemSize(). See SMat::elemSize.	

Mat member function
Mat(int32_t ndims,
    const int32_t* sizes,
    int32_t type,
    void* data,
    const uint64_t* steps=0);
	IN	ndims	Array dimensionality.	Creates a multi-dimensional matrix based on existing buffer.

	IN	sizes	Array of integers specifying an n-dimensional array shape.	
	IN	type	Array type (see above)	
	IN	data	Pointer to the existing data	
	IN	steps	Row steps (number of bytes each matrix row occupies). If the parameter is missing (set to AUTO_STEP), no padding is assumed and the actual step is calculated as cols*elemSize(). See SMat::elemSize.	

Mat member function
Mat(const vsdk::SMat& m,
    const vsdk::Range& rowRange,
    const vsdk::Range&
colRange=vsdk::Range::all());
	IN	m	Original matrix	Creates a matrix header for a part of the bigger matrix

	IN	rowRange	Range of the m rows to take. As usual, the range start is inclusive and the range end is exclusive. Use Range::all() to take all the rows.	

	IN colRange Range of the m columns to take. Use Range::all() to take all the columns.	

`Mat member function`
`Mat(const vsdk::SMat& m,`
`    const vsdk::Rect& roi);`
	IN m Original matrix
	IN roi Region of interest to be taken into account.
		Creates a matrix header for a part of the bigger matrix

`Mat member function`
`Mat(const vsdk::SMat& m,`
`    const vsdk::Range* ranges);`
	IN m Original matrix
	IN ranges Array of selected ranges of m along each dimensionality.
		Creates a matrix header for a part of the bigger matrix

`Mat member function`
`~Mat();`
		Destructor - calls release()

`Mat member function`
`vsdk::Mat& operator =`
`          (const vsdk::SMat& m);`
	IN m Matrix to be assigned - right hand side.	Assignment operator.

`Mat member function`
`vsdk::Mat& operator =`
`          (const cv::Mat& m);`
	IN m Matrix to be assigned - right hand side.	Assignment operator.

`Mat member function`
`Operator cv::Mat();`
		Recast operator.