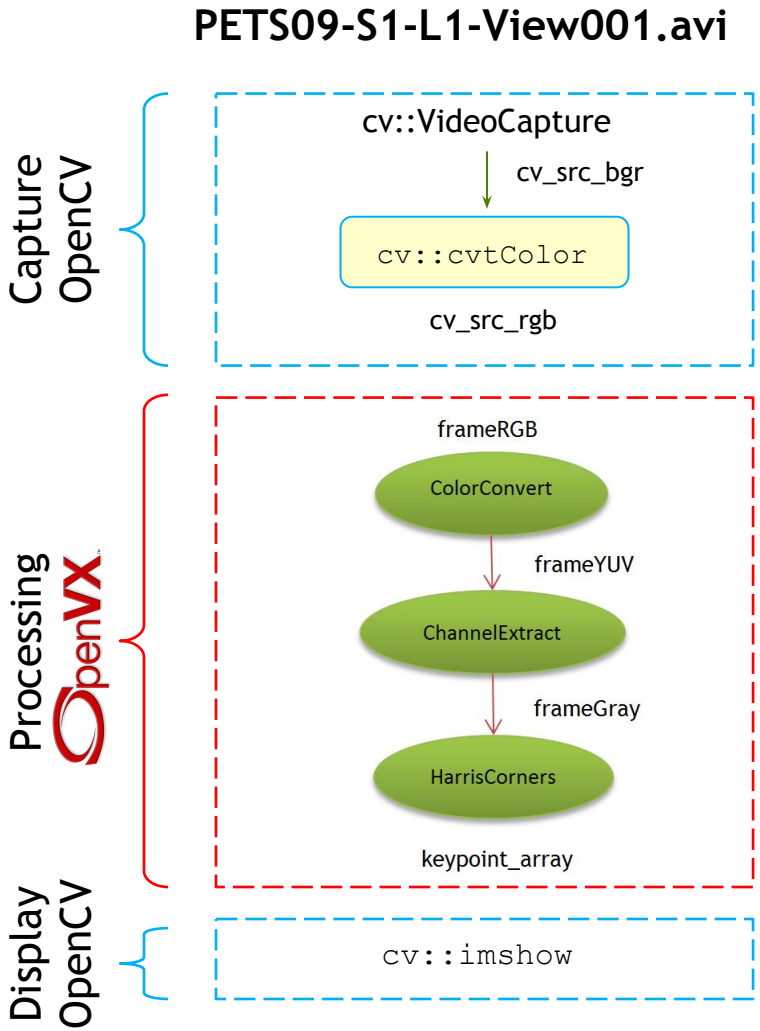
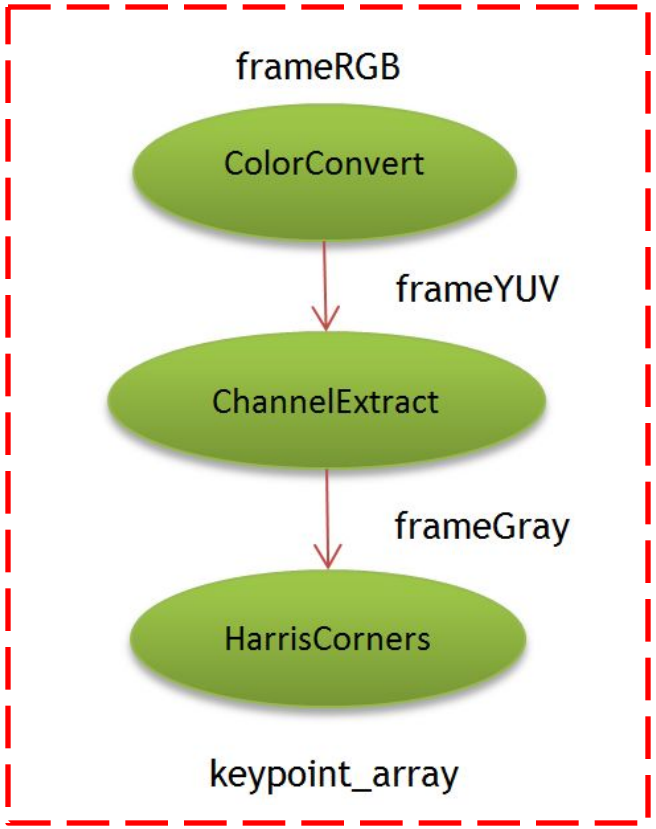
Tutorial Practice Session

Step 1: OpenVX Basic

Step 1: Keypoint detection



Step 1: OpenVX Concepts



- **The world**
  - vx_context
- **Error management**
- **Data object**
  - vx_image, vx_array, vx_scalar
  - Creation / Release
  - Read and write access
- **Vision functions**
  - Immediate execution mode
  - Retained execution mode (graph)

Context

- **Context**
  - OpenVX world: need to be created first
  - All objects belong to a context

```
vx_context context = vxCreateContext();
```

Error Management

- **Methods return a status**
  - `vx_status` returned: `VX_SUCCESS` when no error

```
if( vxuColorConvert( context, input, output ) != VX_SUCCESS) { /* Error */ }
```

- **Explicit status check**
  - Object creation: use `vxGetStatus` to check the object

```
vx_context context = vxCreateContext();
if( vxGetStatus( (vx_reference)context) != VX_SUCCESS ) { /* Error */ }
```

- **More info from the log callback**

```
void logCallback( vx_context c, vx_reference r, vx_status s,
                  const vx_char string[] )
{ /* Do something */ }
...

vxRegisterLogCallback( context, logCallback, vx_false_e );
```

Data objects

- **The application gets only references to objects, not the objects**
  - References should be released by the application when not needed
  - Ref-counted object destroyed by OpenVX when not referenced any more

```
vx_image img = vxCreateImage( context, 640, 400, VX_DF_IMAGE_RGB );
// Use the image
vxReleaseImage( &img );
```

- **Object-Oriented Behavior**
  - strongly typed (good for safety-critical applications)
  - OpenVX are really pointers to structs
    - any object may be down-cast to a vx_reference, e.g., for passing to vxGetStatus()
- **Opaque**
  - Access to content explicit and temporary (access, edit, commit)
    - No permanent pointer to internal data
  - Needed to handle complex memory hierarchies
    - DSP local memory
    - GPU dedicated memory

Image Access (1/3) : Overview

- **Access limited in time**
  - vxAccessImagePath: get access (Read, Write, Read & Write)
  - vxCommitImagePatch: release the access
- **Two modes**
  - MAP: OpenVX controls *address* and *memory layout*

```
void * ptr = NULL;
vx_imagepatch_addressing_t addr;
vx_rectangle_t rect = { 0u, 0u, width, height };
vxAccessImagePatch( img, &rect, plane, &addr, &ptr, VX_READ_AND_WRITE );
// Access data in ptr
vxCommitImagePatch( img, &rect, plane, &addr, ptr );
```

  - COPY: The application controls *address* and *memory layout*

```
void * ptr = &my_array[0];
vx_imagepatch_addressing_t addr = {  /* to fill */ };
vx_rectangle_t rect = { 0u, 0u, width, height };
vxAccessImagePatch( img, &rect, plane, &addr, &ptr, VX_READ_AND_WRITE );
// Access data in my_array
vxCommitImagePatch( img, &rect, plane, &addr, ptr );
```

Image Access (2/3) : Patch

```
typedef struct _vx_rectangle_t {
    vx_uint32 start_x;          /*!< \brief The Start X coordinate. */
    vx_uint32 start_y;          /*!< \brief The Start Y coordinate. */
    vx_uint32 end_x;            /*!< \brief The End X coordinate. */
    vx_uint32 end_y;            /*!< \brief The End Y coordinate. */
} vx_rectangle_t;
```

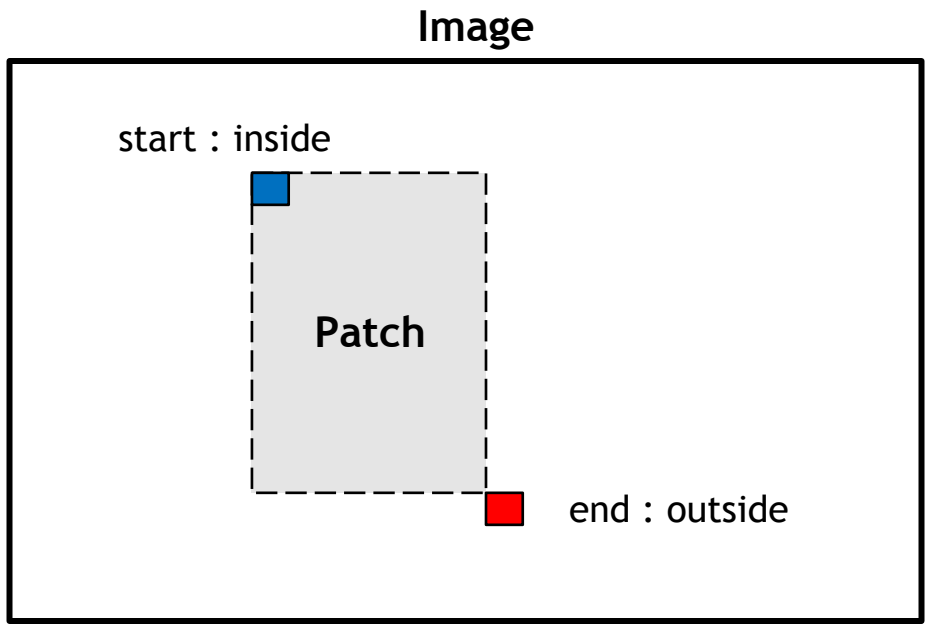


Image Access (3/3) : Memory Layout

```
typedef struct _vx_imagepatch_addressing_t {
    vx_uint32 dim_x;
    vx_uint32 dim_y;
    vx_int32  stride_x;
    vx_int32  stride_y;
    vx_uint32 scale_x;
    vx_uint32 scale_y;
    vx_uint32 step_x;
    vx_uint32 step_y;
} vx_imagepatch_addressing_t;
```

Num of (logical) pixels in a row

Num of (logical) pixels in a column

Num of bytes between the beginning of 2 successive pixels

Num of bytes between the beginning of 2 successive lines

Sub-sampling :
1 *physical* pixel every 'step' *logical* pixel
scale = VX_SCALE_UNITY / step

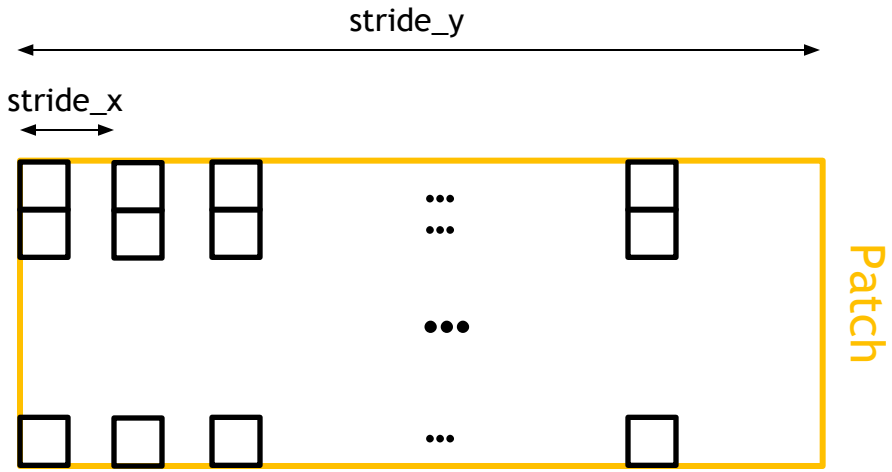


Image Created from a Handle

- **Image is created from application memory**
  - Import an external image *without forcing a copy*
    - *if exotic alignment or memory layout,*
      the OpenVX implementation may need to create an internal copy
- **Image is always *mapped* at the original address and memory layout**

```
vx_imagepatch_addressing_t src_addr;
src_addr.dim_x    = src_width;
src_addr.dim_y    = src_height;
src_addr.stride_x = 3*sizeof( vx_uint8 );
src_addr.stride_y = cv_src_rgb.step;

void *src_ptrs[]  = { cv_src_rgb.data };

vx_image img = vxCreateImageFromHandle( context, VX_DF_IMAGE_RGB, &src_addr,
                                        src_ptrs, VX_IMPORT_TYPE_HOST );
```

Miscellaneous

- Array
  - Variable number of elements, but fixed maximum capacity

```
vx_array array = vxCreateArray( context, VX_TYPE_KEYPOINT, 10000 );
```

  - Access philosophy is similar to the image (MAP / COPY)

```
vx_size num;
vxQueryArray( array, VX_ARRAY_ATTRIBUTE_NUMITEMS, &num, sizeof(num) );

vx_keypoint_t * ptr = NULL; // access in MAP mode
vx_size stride;
vxAccessArrayRange( array, 0, num, &stride, (void **)&ptr, VX_READ_ONLY );
/* Access */
vxCommitArrayRange( array, 0, num, ptr );
```

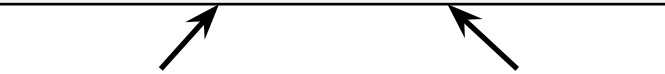
- Scalar

```
vx_float32 distance  = 5.f;
vx_scalar s_distance = vxCreateScalar( context, VX_TYPE_FLOAT32, &distance
);
```

Vision Functions: Immediate Execution Mode

- **RGB -> YUV**

```
vxuColorConvert( context, frameRGB, frameYUV );
```



- **YUV -> Y**

```
vxuChannelExtract( context, frameYUV, VX_CHANNEL_Y, frameGray );
```

- **Harris corner**
  - strength_thresh : 0.0005f
  - min_distance : 5.0f
  - sensitivity : 0.04f
  - gradient_size : 3
  - block_size : 3

```
vxuHarrisCorners( context, frameGray, s_strength_thresh, s_min_distance,
                  s_k_sensitivity, gradientSize, blockSize,
                  keypoint_array, NULL )
```