Tutorial Practice Session

Step 2: Graphs

Why graphs?

- **Most APIs (e.g., OpenCV) are based on function calls**
  - a function abstracts an algorithm that processes input data and produces output
  - the function can be optimized independent of all the other functionality
- **An application executes many function calls**
  - the call sequence of the function really defines a graph
- **There are limits to how much functions can be optimized**
  - but if you know that function B is called after A, you might be able to combine them to a new function that is much faster than time(A) + time(B)
- **By building first a graph of the function calls, and only executing later, we open up lots of optimization possibilities**
  - this is also how graphics APIs like OpenGL work

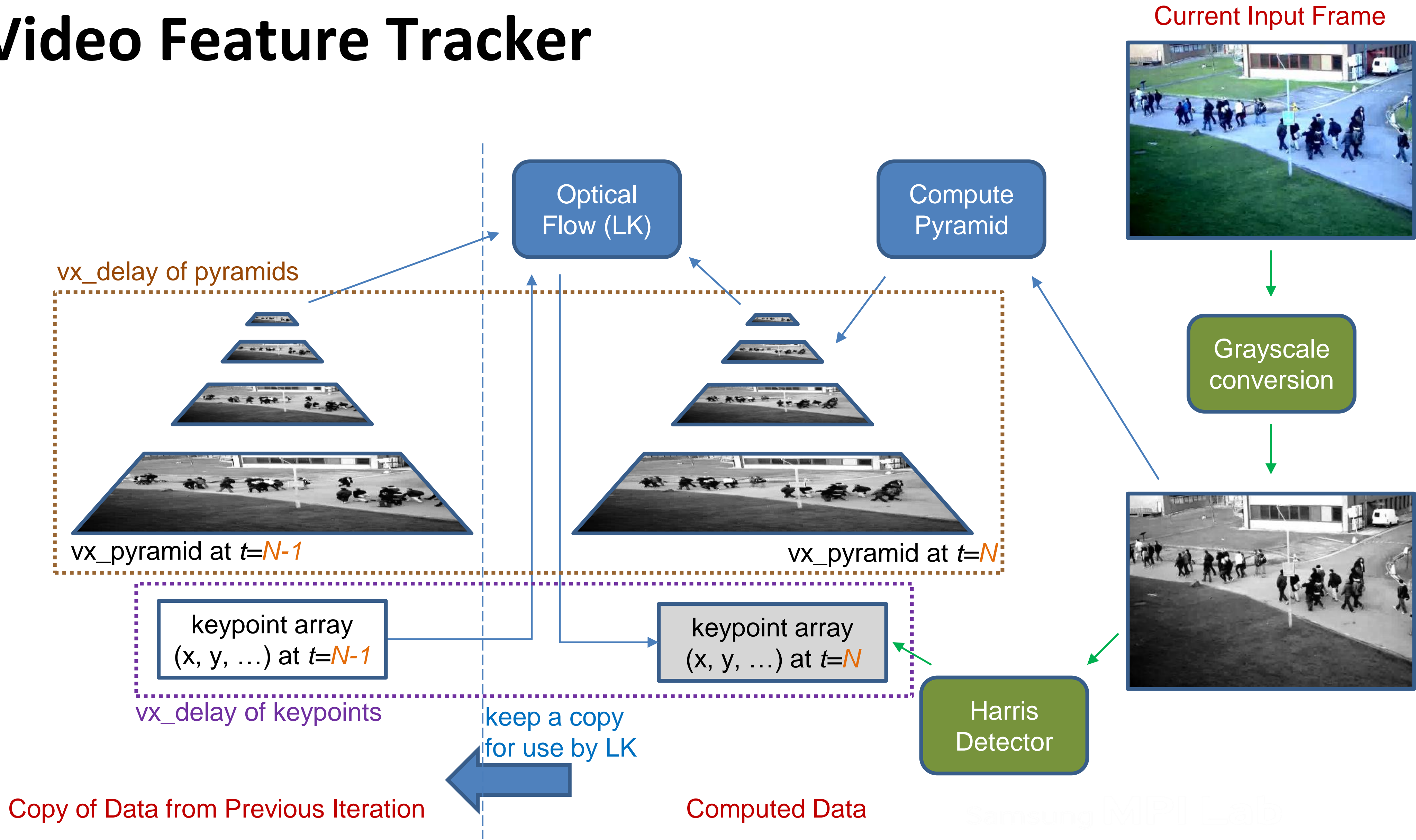
KHRONOS GROUP

Optimization opportunities

- **Fuse kernels**
  - while accessing the image, do many things at the same time
- **Process the images in tiles**
  - for better locality, memory access coherency
- **Parallelize**
  - with more work that is available, more parallelization opportunities

KHRONOS GROUP

Video Feature Tracker



Video Feature Tracker

Current Input Frame

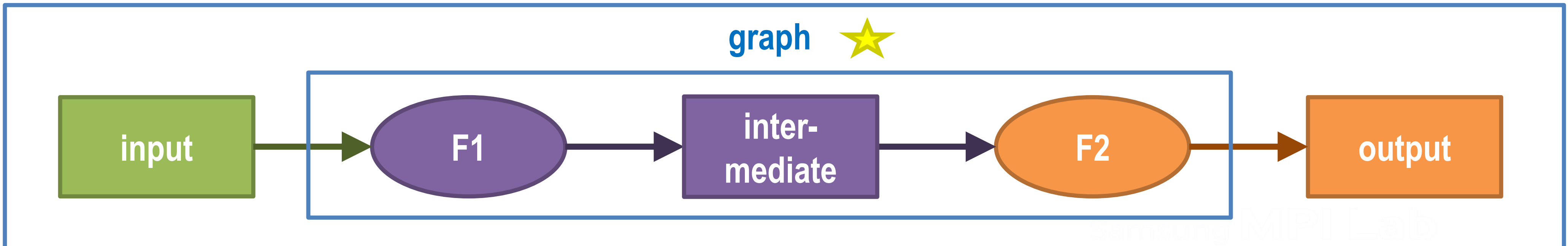
KHRONOS GROUP

OpenVX Code

```
vx_context context = vxCreateContext();
vx_image input = vxCreateImage( context, 640, 480, VX_DF_IMAGE_U8 );
vx_image output = vxCreateImage( context, 640, 480, VX_DF_IMAGE_U8 );

vx_graph graph = vxCreateGraph( context );
vx_image intermediate = vxCreateVirtualImage( graph, 640, 480, VX_DF_IMAGE_U8 );
vx_node F1 = vxF1Node( graph, input, intermediate );
vx_node F2 = vxF2Node( graph, intermediate, output );

vxVerifyGraph( graph );
vxProcessGraph( graph ); // run in a loop
```



KHRONOS GROUP

TODO: Let’s create the graphs in exercise2…

```
207        ////////********
208        // Harris and optical flow algorithms require their own graph objects.
209        // The Harris graph needs to extract gray scale image out of input RGB,
210        // compute an initial set of keypoints, and compute an initial pyramid for use
211        // by the optical flow graph.
212        //
213        // TODO STEP 03:********
214        //   1. Create two graph objects: one for the Harris corner detector and
215        //      the other for feature tracking using optical flow using the
216        //      vxCreateGraph API.
217        //      We gave code for one graph; do similar for the other.
218        //   2. Use ERROR_CHECK_OBJECT to check the objects.
219        //      We gave one error check; do similar for the other.
220    //    vx_graph graphHarris = vxCreateGraph( context );
221    //    vx_graph graphTrack  = /* Fill in here */;
222    //    ERROR_CHECK_OBJECT( graphHarris );
223
```

KHRONOS GROUP

Example: Keypoint Detector

RGB Camera

VX_DF_IMAGE_RGB

vx_image
OpenVX

vxCreateImage

org.khronos.openvx.color_convert
OpenVX

vxColorConvertNode

vxCreateVirtualImage

VX_DF_IMAGE_IYUV

vx_image
OpenVX

VX_CHANNEL_Y

OpenVX
Node

org.khronos.openvx.channel_extract
OpenVX

vxChannelExtractNode

Display
Results

vxCreateVirtualImage

OpenVX
Data Object

VX_DF_IMAGE_U8

vx_image
OpenVX

configuration
parameters

vxCreateArray

vx_array
OpenVX

org.khronos.openvx.harris_corners
OpenVX

vxHarrisCornersNode

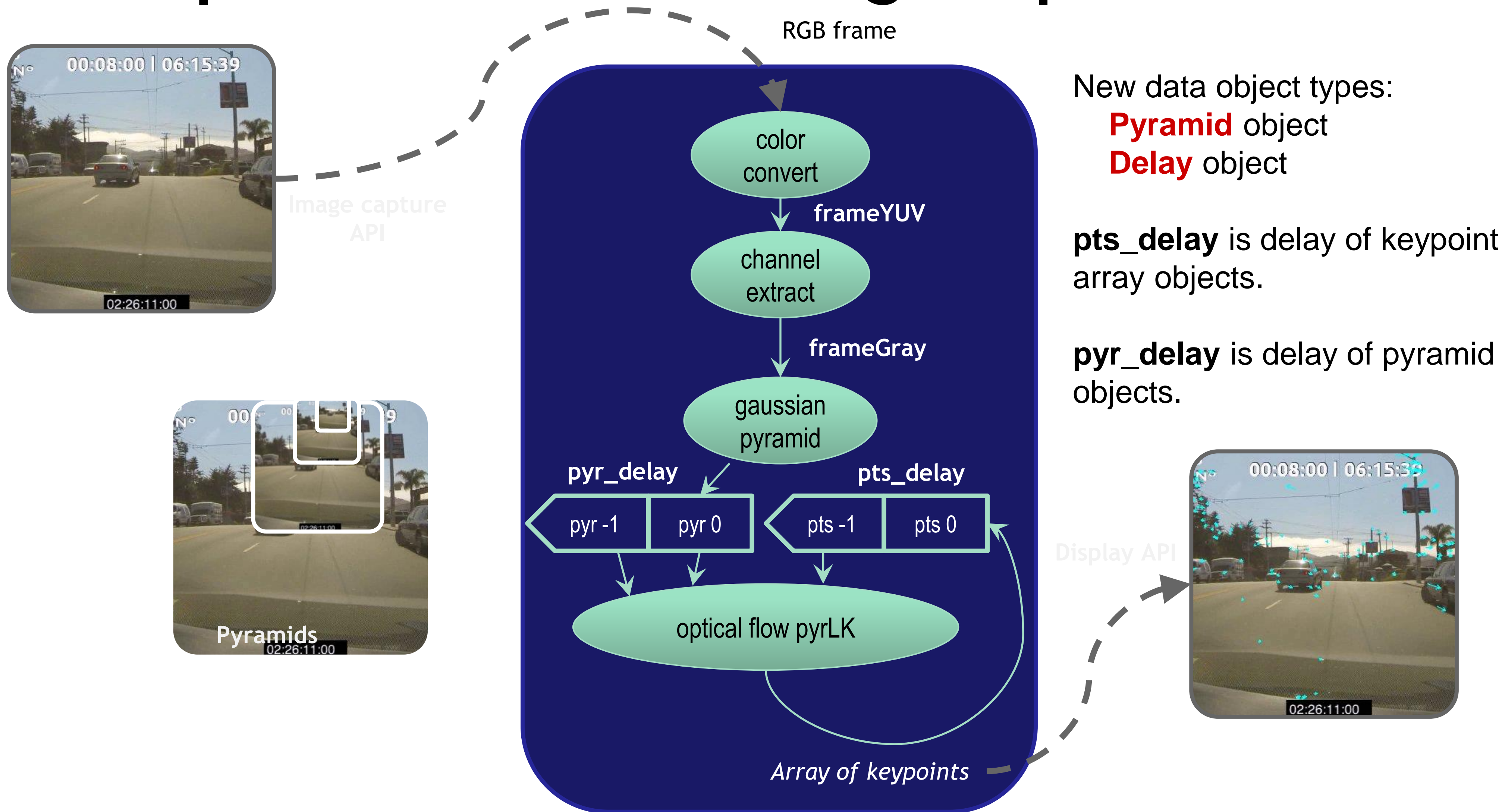
VX_TYPE_KEYPOINT

OpenVX Graph

KHRONOS GROUP

© Copyright Khronos Group 2016 -

Example: Feature Tracking Graph



New data object types:

- **Pyramid** object
- **Delay** object

pts_delay is delay of keypoint array objects.

pyr_delay is delay of pyramid objects.

© Copyright Khronos Group 2016 - Page 9

exercise2.cpp and configuration parameters…

```
114    //   lk_pyramid_levels       - number of pyramid levels for LK optical flow
115    //   lk_termination          - can be VX_TERM_CRITERIA_ITERATIONS or
116    //                              VX_TERM_CRITERIA_EPSILON or
117    //                              VX_TERM_CRITERIA_BOTH
118    //   lk_epsilon              - error for terminating the algorithm
119    //   lk_num_iterations       - number of iterations
120    //   lk_use_initial_estimate - turn on/off use of initial estimates
121    //   lk_window_dimension     - size of window on which to perform the algorithm
122    vx_uint32  width                   = gui.GetWidth();
123    vx_uint32  height                  = gui.GetHeight();
124    vx_size    max_keypoint_count      = 10000;
125    vx_float32 harris_strength_thresh  = 0.0005f;
126    vx_float32 harris_min_distance     = 5.0f;
127    vx_float32 harris_k_sensitivity    = 0.04f;
128    vx_int32   harris_gradient_size    = 3;
129    vx_int32   harris_block_size       = 3;
130    vx_uint32  lk_pyramid_levels       = 6;
131    vx_float32 lk_pyramid_scale        = VX_SCALE_PYRAMID_HALF;
132    vx_enum    lk_termination          = VX_TERM_CRITERIA_BOTH;
133    vx_float32 lk_epsilon              = 0.01f;
134    vx_uint32  lk_num_iterations       = 5;
135    vx_bool    lk_use_initial_estimate = vx_false_e;
136    vx_uint32  lk_window_dimension     = 6;
```

KHRONOS GROUP

TODO: create virtual images...

```
225        ////////********
226        // Harris and pyramid computation expect input to be an 8-bit image.
227        // Given that input is an RGB image, it is best to extract a gray image
228        // from RGB image, which requires two steps:
229        //   - perform RGB to IYUV color conversion
230        //   - extract Y channel from IYUV image
231        // This requires two intermediate OpenVX image objects. Since you don't
232        // need to access these objects from the application, they can be virtual
233        // objects that can be created using the vxCreateVirtualImage API.
234        //
235        // TODO STEP 04:********
236        //   1. Create an IYUV image and a U8 image (for Y channel) with the same
237        //      dimensions as the input RGB image. Note that the image formats for
238        //      IYUV and U8 images are VX_DF_IMAGE_IYUV and VX_DF_IMAGE_U8.
239        //      Note that virtual objects are specific to a graph, so you
240        //      need to create two sets, one for each graph.
241        //      We gave one fully in comments and you need to fill in missing
242        //      parameters for the others.
243        //   2. Use ERROR_CHECK_OBJECT to check the objects.
244        //      We gave one error check in comments; do similar for others.
245    //    vx_image harris_yuv_image       = vxCreateVirtualImage( graphHarris, width, height, VX_DF_IMAGE_IYUV );
246    //    vx_image harris_luma_image      = vxCreateVirtualImage( graphHarris, /* Fill in parameters */ );
247    //    vx_image opticalflow_yuv_image  = vxCreateVirtualImage( graphTrack,  /* Fill in parameters */ );
248    //    vx_image opticalflow_luma_image = vxCreateVirtualImage( /* Fill in parameters */ );
249    //    ERROR_CHECK_OBJECT( harris_yuv_image );
```

KHRONOS GROUP

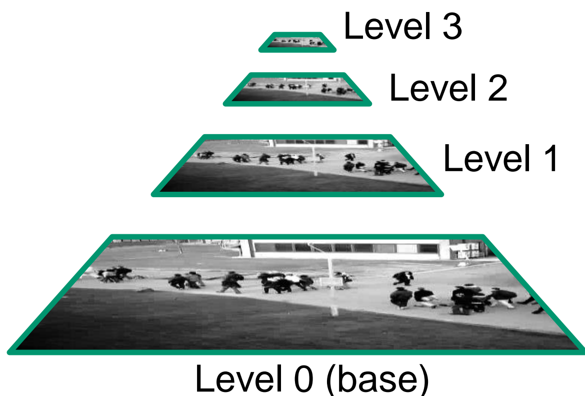
TODO: create scalar objects...

```
259        // TODO STEP 05:********
260        //   1. Create scalar data objects of VX_TYPE_FLOAT32 for strength_thresh,
261        //      min_distance, sensitivity, and epsilon. Set their
262        //      initial values to harris_strength_thresh, harris_min_distance,
263        //      harris_k_sensitivity, and lk_epsilon.
264        //      We gave code full code for one scalar in comments; fill in
265        //      missing arguments for other ones.
266        //   2. Similarly, create scalar objects for num_iterations and
267        //      use_initial_estimate with initial values: lk_num_iterations and
268        //      lk_use_initial_estimate. Make sure to use proper data types for
269        //      these parameters.
270        //      We gave code full code for one scalar in comments; fill in
271        //      missing arguments for the other.
272        //   3. Use ERROR_CHECK_OBJECT to check proper creation of objects.
273        //      We gave the error check for one scalar; do similar for other 5 scalars.
274        vx_scalar strength_thresh = vxCreateScalar(context, VX_TYPE_FLOAT32, &harris_strength_thresh);
275        vx_scalar min_distance = vxCreateScalar(context, VX_TYPE_FLOAT32, &harris_min_distance);
276        vx_scalar sensitivity = vxCreateScalar(context, VX_TYPE_FLOAT32, &harris_k_sensitivity);
277        vx_scalar epsilon = vxCreateScalar(context, VX_TYPE_FLOAT32, &lk_epsilon);
278        vx_scalar num_iterations = vxCreateScalar(context, VX_TYPE_UINT32, &lk_num_iterations);
279        vx_scalar use_initial_estimate = vxCreateScalar(context, VX_TYPE_BOOL, &lk_use_initial_estimate);
280        ERROR_CHECK_OBJECT(strength_thresh);
```

KHRONOS GROUP

Pyramid Data Object

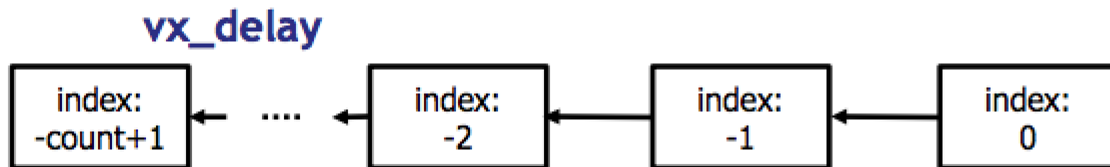
```
vx_pyramid vxCreatePyramid (
    vx_context context,
    vx_size levels,
    vx_float32 scale, // VX_SCALE_PYRAMID_HALF or VX_SCALE_PYRAMID_ORB
    vx_uint32 width,
    vx_uint32 height,
    vx_df_image format // VX_DF_IMAGE_U8
);
```



exercise2.cpp <Select Symbol>

```
114     //   lk_pyramid_levels        - number of pyramid levels for LK optical flow
115     //   lk_termination           - can be VX_TERM_CRITERIA_ITERATIONS or
116     //                                VX_TERM_CRITERIA_EPSILON or
117     //                                VX_TERM_CRITERIA_BOTH
118     //   lk_epsilon               - error for terminating the algorithm
119     //   lk_num_iterations        - number of iterations
120     //   lk_use_initial_estimate  - turn on/off use of initial estimates
121     //   lk_window_dimension      - size of window on which to perform the algorithm
122     vx_uint32  width                 = gui.GetWidth();
123     vx_uint32  height                = gui.GetHeight();
124     vx_size    max_keypoint_count    = 10000;
125     vx_float32 harris_strength_thresh = 0.0005f;
126     vx_float32 harris_min_distance   = 5.0f;
127     vx_float32 harris_k_sensitivity  = 0.04f;
128     vx_int32   harris_gradient_size  = 3;
129     vx_int32   harris_block_size     = 3;
130     vx_uint32  lk_pyramid_levels     = 6;
131     vx_float32 lk_pyramid_scale      = VX_SCALE_PYRAMID_HALF;
```

Delay Data Object




```
vx_delay vxCreateDelay
 (
    vx_context context,
    vx_reference exemplar,
    vx_size count
 );
```

Example:

```
vx_pyramid exemplar = vxCreatePyramid(context, …);
vx_delay pyr_delay = vxCreateDelay(context, (vx_reference)exemplar, 2);
vxReleasePyramid(&exemplar);

…
vx_pyramid pyr_0 = (vx_pyramid)vxGetReferenceFromDelay(pyr_delay, 0);
vx_pyramid pyr_1 = (vx_pyramid)vxGetReferenceFromDelay(pyr_delay, -1);
…

vxAgeDelay(pyr_delay);
```

KHRONOS GROUP

TODO: create delay objects…

```
160         // TODO STEP 01:********
161         //    1. Use vxCreatePyramid API to create a pyramid exemplar with the
162         //       same dimensions as the input image, VX_DF_IMAGE_U8 as image format,
163         //       lk_pyramid_levels as levels, and lk_pyramid_scale as scale.
164         //       We gave code for this in comments.
165         //    2. Use vxCreateArray API to create an array exemplar with
166         //       keypoint data type with num_keypoint_count as capacity.
167         //       You need to add missing parameters to code in comments.
168         //    3. Use vxCreateDelay API to create delay objects for pyramid and
169         //       keypoint array using the exemplars created using the two steps above.
170         //       Use 2 delay slots for both of the delay objects.
171         //       We gave code for one in comments; do similar for the other.
172         //    4. Release the pyramid and keypoint array exemplar objects.
173         //       We gave code for one in comments; do similar for the other.
174         //    5. Use ERROR_CHECK_OBJECT/STATUS macros for proper error checking.
175         //       We gave few error checks; do similar for the others.
176     //      vx_pyramid pyramidExemplar = vxCreatePyramid( context, lk_pyramid_levels,
177     //                                                    lk_pyramid_scale, width, height, VX_DF_IMAGE_U8 );
178     //      ERROR_CHECK_OBJECT( pyramidExemplar );
179     //      vx_delay pyramidDelay   = vxCreateDelay( context, ( vx_reference )pyramidExemplar, 2 );
180     //      ERROR_CHECK_OBJECT( pyramidDelay );
181     //      ERROR_CHECK_STATUS( vxReleasePyramid( &pyramidExemplar ) );
182     //      vx_array keypointsExemplar = vxCreateArray( /* Fill in parameters */ );
183     //      vx_delay keypointsDelay = vxCreateDelay( /* Fill in parameters */ );
184
```

KHRONOS GROUP

TODO: create delay objects…

```
160     // TODO STEP 01:********
161     //   1. Use vxCreatePyramid API to create a pyramid exemplar with the
162     //      same dimensions as the input image, VX_DF_IMAGE_U8 as image format,
163     //      lk_pyramid_levels as levels, and lk_pyramid_scale as scale.
164     //      We gave code for this in comments.
165     //   2. Use vxCreateArray API to create an array exemplar with
166     //      keypoint data type with num_keypoint_count as capacity.
167     //      You need to add missing parameters to code in comments.
168     //   3. Use vxCreateDelay API to create delay objects for pyramid and
169     //      keypoint array using the exemplars created using the two steps above.
170     //      Use 2 delay slots for both of the delay objects.
171     //      We gave code for one in comments; do similar for the other.
172     //   4. Release the pyramid and keypoint array exemplar objects.
173     //      We gave code for one in comments; do similar for the other.
174     //   5. Use ERROR_CHECK_OBJECT/STATUS macros for proper error checking.
175     //      We gave few error checks; do similar for the others.
176     vx_pyramid pyramidExemplar = vxCreatePyramid(context, lk_pyramid_levels,
177         lk_pyramid_scale, width, height, VX_DF_IMAGE_U8);
178     vx_array keypointsExemplar = vxCreateArray(context, VX_TYPE_KEYPOINT,
179         max_keypoint_count);
180     ERROR_CHECK_OBJECT(pyramidExemplar);
181     ERROR_CHECK_OBJECT(keypointsExemplar);
182     vx_delay pyramidDelay = vxCreateDelay(context, (vx_reference)pyramidExemplar, 2);
183     vx_delay keypointsDelay = vxCreateDelay(context, (vx_reference)keypointsExemplar, 2);
184     ERROR_CHECK_OBJECT(pyramidDelay);
185     ERROR_CHECK_OBJECT(keypointsDelay);
186     ERROR_CHECK_STATUS(vxReleasePyramid(&pyramidExemplar));
187     ERROR_CHECK_STATUS(vxReleaseArray(&keypointsExemplar));
```

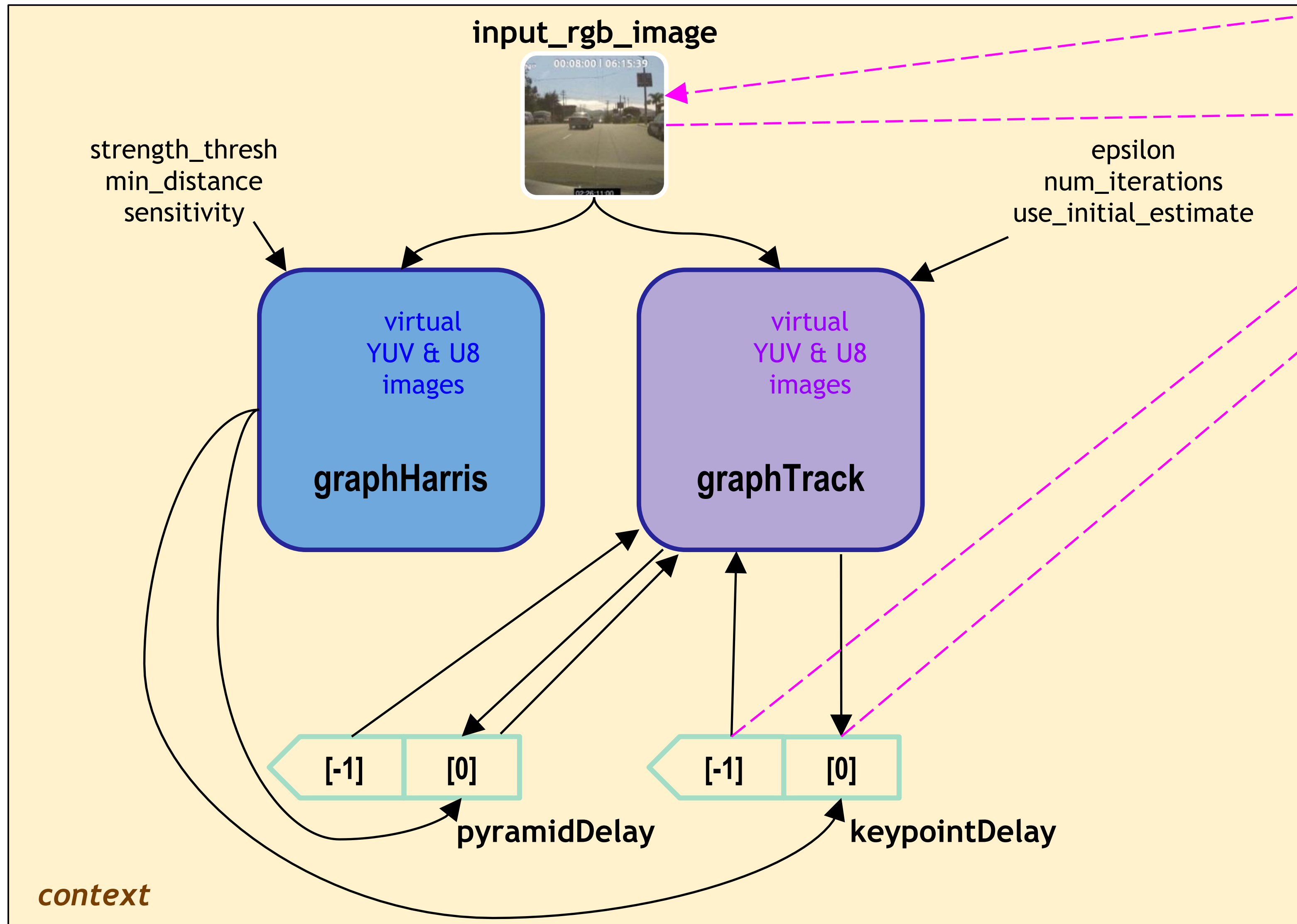
KHRONOS GROUP

TODO: access objects from delay objects…

```
186     /////////*********
187     // An object from a delay slot can be accessed using vxGetReferenceFromDelay API.
188     // You need to use index = 0 for the current object and index = -1 for the previous object.
189     //
190     // TODO STEP 02:********
191     //   1. Use vxGetReferenceFromDelay API to get the current and previous
192     //      pyramid objects from pyramid delay object. Note that you need
193     //      to typecast the vx_reference object to vx_pyramid.
194     //      We gave code for one in comments; do similar for the other.
195     //   2. Similarly, get the current and previous keypoint array objects from
196     //      the keypoint delay object.
197     //      We gave code for one in comments; do similar for the other.
198     //   3. Use ERROR_CHECK_OBJECT for proper error checking.
199     //      We gave one error check; do similar for the others.
200 //     vx_pyramid currentPyramid  = ( vx_pyramid ) vxGetReferenceFromDelay( pyramidDelay, 0 );
201 //     vx_pyramid previousPyramid = ( vx_pyramid ) vxGetReferenceFromDelay( /* Fill in parameters */ );
202 //     vx_array currentKeypoints  = ( vx_array )   vxGetReferenceFromDelay( /* Fill in parameters */ );
203 //     vx_array previousKeypoints = ( vx_array )   vxGetReferenceFromDelay( keypointsDelay, -1 );
204 //     ERROR_CHECK_OBJECT( currentPyramid );
```

KHRONOS GROUP

exercise2 in a nut-shell

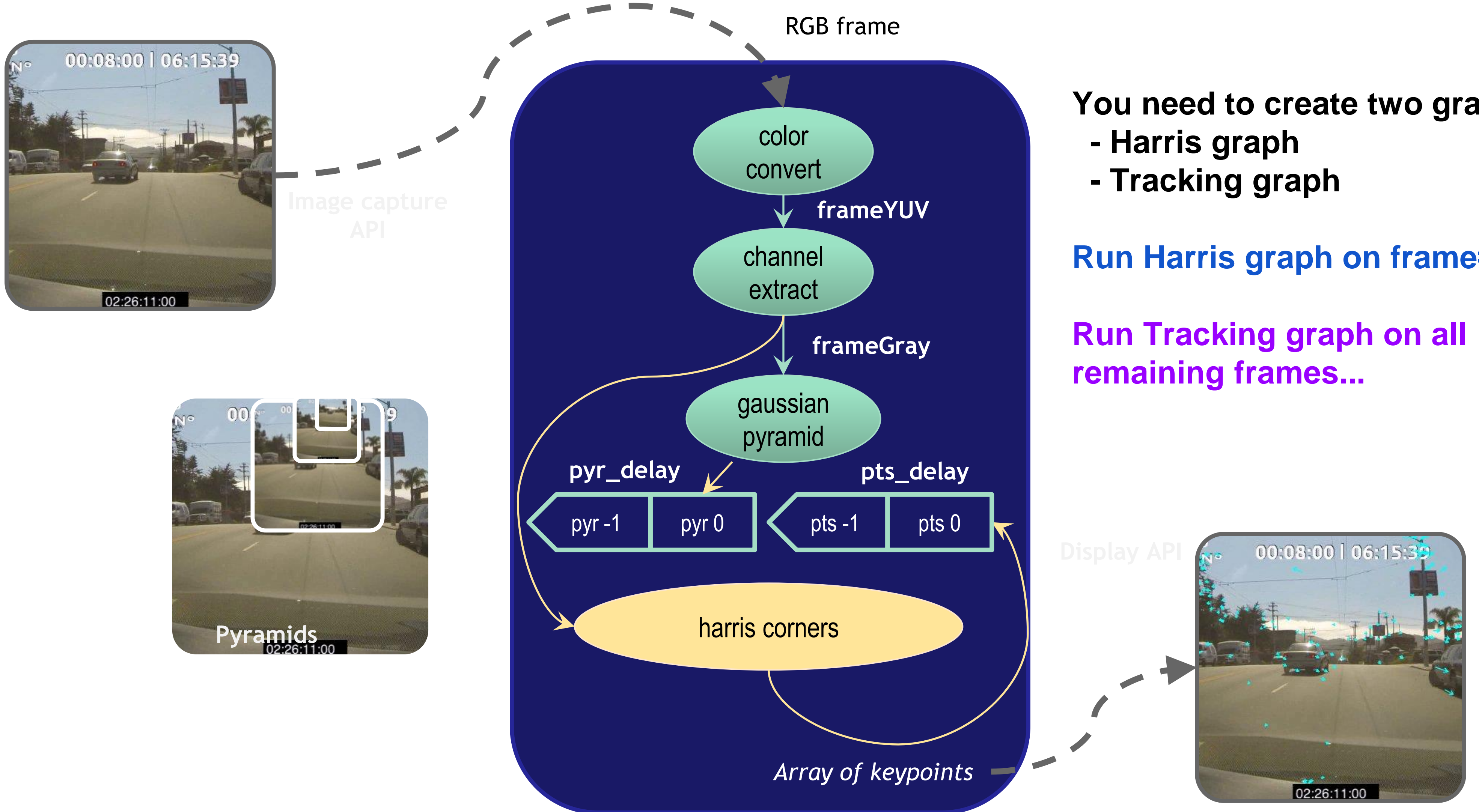


create all data objects and graphs within a context.

verify graphs.

process each frame:
- read input frame
- process graph:
  - harris, if frame#0
  - track, otherwise
- display results
- age delay objects

Harris on frame#0 ...



You need to create two graphs:
- **Harris graph**
- **Tracking graph**

Run Harris graph on frame#0.

Run Tracking graph on all remaining frames...

© Copyright Khronos Group 2016

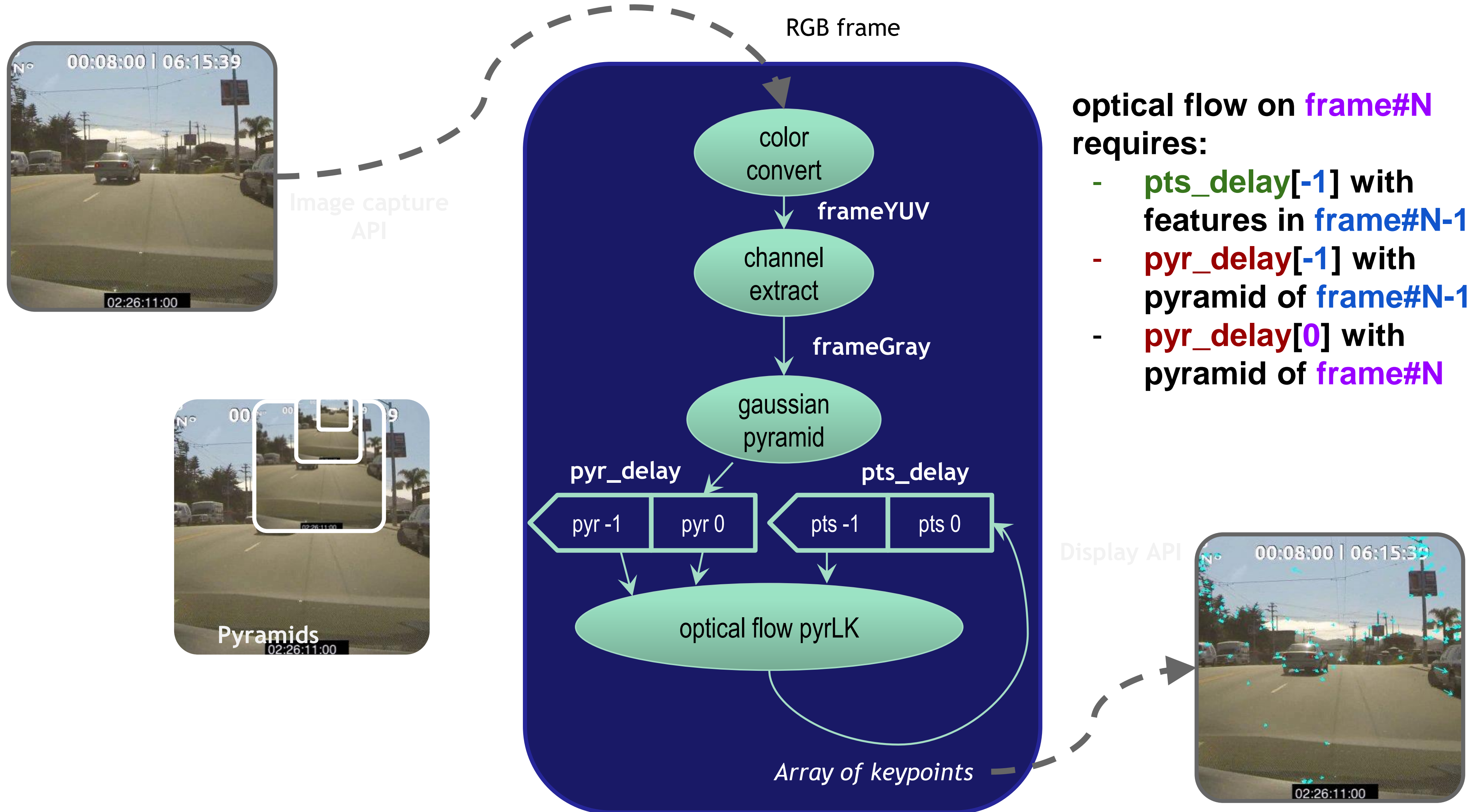
TODO: setup Harris graph…

```
288        // TODO STEP 06:********
289        //   1. Use vxColorConvertNode and vxChannelExtractNode APIs to get gray
290        //      scale image for Harris and Pyramid computation from the input
291        //      RGB image. Add these nodes into Harris graph.
292        //      We gave code in comments with a missing parameter for you to fill in.
293        //   2. Use vxGaussianPyramidNode API to add pyramid computation node.
294        //      You need to use the current pyramid from the pyramid delay object.
295        //      We gave code in comments with a missing parameter for you to fill in.
296
297
298
299
300
301
302
303
```

```
304    //     vx_node nodesHarris[] =
305    //     {
306    //         vxColorConvertNode( graphHarris, input_rgb_image, harris_yuv_image ),
307    //         vxChannelExtractNode( graphHarris, /* Fill in parameter */, VX_CHANNEL_Y, harris_luma_image ),
308    //         vxGaussianPyramidNode( graphHarris, /* Fill in parameter */, currentPyramid ),
309    //         vxHarrisCornersNode( graphHarris, /* Fill in missing parameters */, currentKeypoints, NULL )
310    //     };
311    //     for( vx_size i = 0; i < sizeof( nodesHarris ) / sizeof( nodesHarris[0] ); i++ )
312    //     {
313    //         ERROR_CHECK_OBJECT( nodesHarris[i] );
314    //         ERROR_CHECK_STATUS( vxReleaseNode( &nodesHarris[i] ) );
315    //     }
316    //     ERROR_CHECK_STATUS( vxReleaseImage( &harris_yuv_image ) );
317    //     ERROR_CHECK_STATUS( vxReleaseImage( &harris_luma_image ) );
318    //     ERROR_CHECK_STATUS( vxVerifyGraph( /* Fill in parameter */ ) );
```

KHRONOS GROUP

Tracking can be started from frame#1 …



optical flow on frame#N requires:

- **pts_delay[-1] with features in frame#N-1**
- **pyr_delay[-1] with pyramid of frame#N-1**
- **pyr_delay[0] with pyramid of frame#N**

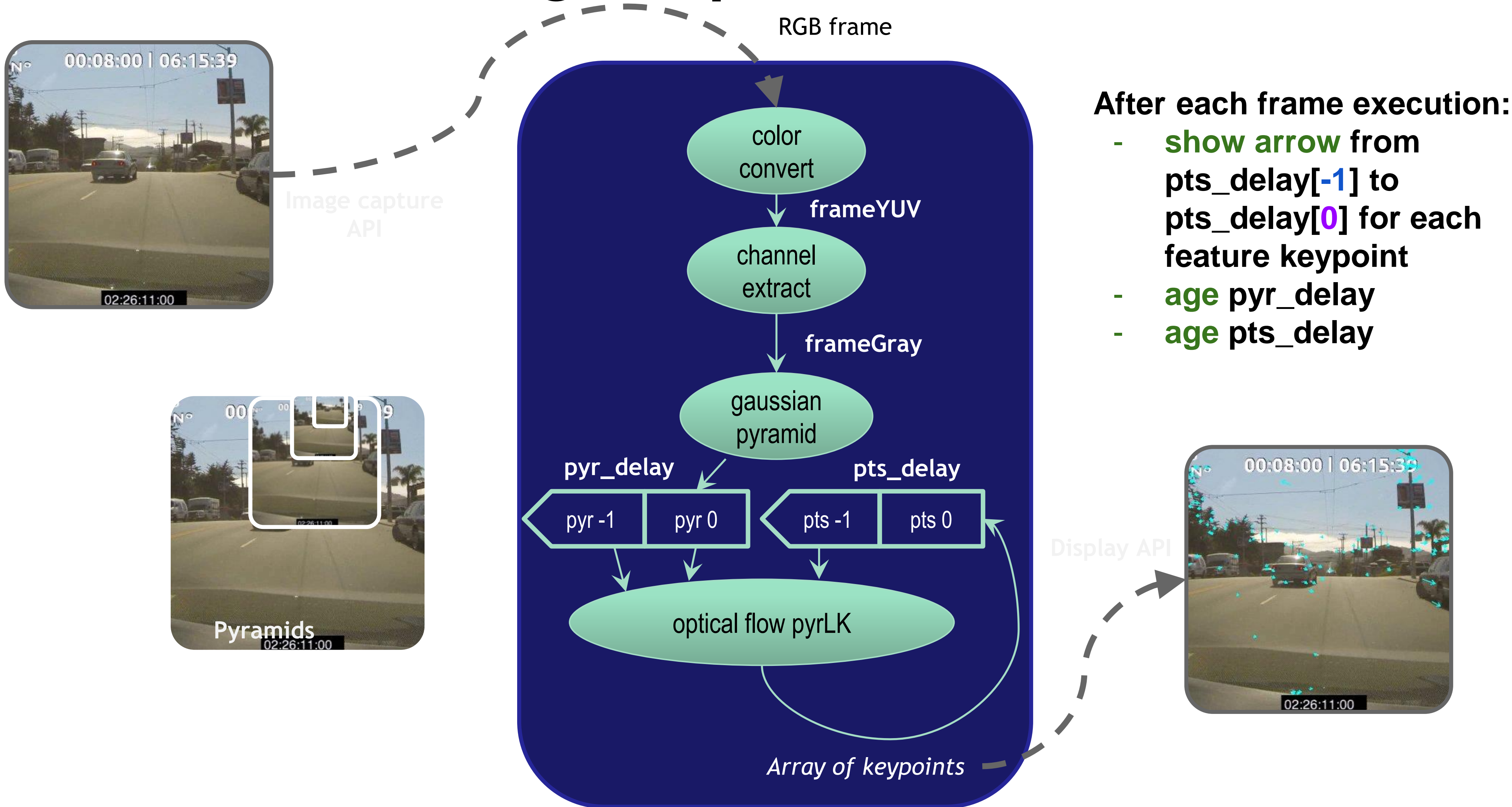
© Copyright Khronos Group 2016

TODO: setup Feature Tracking graph…

```
325        // TODO STEP 07:********
326        //   1. Use vxColorConvertNode and vxChannelExtractNode APIs to get a gray
327        //      scale image for Harris and Pyramid computation from the input
328        //      RGB image. Add these nodes into Harris graph.
329        //      We gave the code in comments for color convert node; do similar
330        //      one for the channel extract node.
331        //   2. Use vxGaussianPyramidNode API to add pyramid computation node.
332        //      You need to use the current pyramid from the pyramid delay object.
333        //      Most of the code is given in the comments; fill in the missing parameter
334
335
336
337
338
339
340
341
```

```
342     //    vx_node nodesTrack[] =
343     //    {
344     //        vxColorConvertNode( graphTrack, input_rgb_image, opticalflow_yuv_image ),
345     //        vxChannelExtractNode( graphTrack, /* Fill in parameters */ ),
346     //        vxGaussianPyramidNode( graphTrack, /* Fill in parameter */, currentPyramid ),
347     //        vxOpticalFlowPyrLKNode( graphTrack, /* Fill in parameters */ )
348     //    };
349     //    for( vx_size i = 0; i < sizeof( nodesTrack ) / sizeof( nodesTrack[0] ); i++ )
350     //    {
351     //        ERROR_CHECK_OBJECT( nodesTrack[i] );
352     //        ERROR_CHECK_STATUS( vxReleaseNode( &nodesTrack[i] ) );
353     //    }
354     //    ERROR_CHECK_STATUS( vxReleaseImage( &opticalflow_yuv_image ) );
355     //    ERROR_CHECK_STATUS( vxReleaseImage( &opticalflow_luma_image ) );
356     //    ERROR_CHECK_STATUS( vxVerifyGraph( /* Fill in parameter */ ) );
357
```

Feature Tracking Graph …



After each frame execution:

- **show arrow from pts_delay[-1] to pts_delay[0] for each feature keypoint**
- **age pyr_delay**
- **age pts_delay**

TODO: run a graph...

```
380
381         ////////********
382         // Now that input RGB image is ready, just run a graph.
383         // Run Harris at the beginning to initialize the previous keypoints.
384         //
385         // TODO STEP 08:********
386         //   1. Run a graph using vxProcessGraph API. Select Harris graph
387         //      if the frame_index == 0 (i.e., the first frame of the video
388         //      sequence), otherwise, select the feature tracking graph.
389         //   2. Use ERROR_CHECK_STATUS for error checking.
390
```

KHRONOS GROUP

TODO: display the tracking results…

```
393         ////////********
394         // To mark the keypoints in display, you need to access the output
395         // keypoint array and draw each item on the output window using gui.DrawArrow().
396         //
397         // TODO STEP 09:********
398         //   1. Use vxGetReferenceFromDelay API to get the current and previous
399         //      keypoints array objects from the keypoints delay object.
400         //      Make sure to typecast the vx_reference object to vx_array.
401         //      We gave one for the previous previous keypoint array in comments;
402         //      do a similar one for the current keypoint array.
403         //   2. OpenVX array object has an attribute that keeps the current
404         //      number of items in the array. The name of the attribute is
405         //      VX_ARRAY_ATTRIBUTE_NUMITEMS and its value is of type vx_size.
406         //      Use vxQueryArray API to get number of keypoints in the
407         //      current keypoint array data object, representing number of
408         //      corners detected in the input RGB image.
409         //      IMPORTANT: Read number of items into "num_corners"
410         //      because this variable is displayed by code segment below.
411         //      We gave most part of this statement in comment; just fill in the
412         //      missing parameter.
413         //   3. The data items in output keypoint array are of type
414         //      vx_keypoint_t (see "VX/vx_types.h"). To access the array
415         //      buffer, use vxAccessArrayRange with start index = 0,
416         //      end index = number of items in the array, and usage mode =
417         //      VX_READ_ONLY. Note that the stride returned by this access
418         //      call is not guaranteed to be sizeof(vx_keypoint_t).
```

TODO: delay aging…

```
462         /////////*********
463         // Flip the current and previous pyramid and keypoints in the delay objects.
464         //
465         // TODO STEP 10:********
466         //   1. Use vxAgeDelay API to flip the current and previous buffers in delay objects.
467         //      You need to call vxAgeDelay for both two delay objects.
468         //   2. Use ERROR_CHECK_STATUS for error checking.
469     //          ERROR_CHECK_STATUS( vxAgeDelay( /* Fill in parameter */ ) );
470     //          ERROR_CHECK_STATUS( vxAgeDelay( /* Fill in parameter */ ) );
471
```

KHRONOS GROUP

TODO: delay aging… then it’ll be ready to run

```
462         ////////********
463         // Flip the current and previous pyramid and keypoints in the delay objects.
464         //
465         // TODO STEP 10:********
466         //   1. Use vxAgeDelay API to flip the current and previous buffers in delay objects.
467         //      You need to call vxAgeDelay for both two delay objects.
468         //   2. Use ERROR_CHECK_STATUS for error checking.
469 //          ERROR_CHECK_STATUS( vxAgeDelay( /* Fill in parameter */ ) );
470 //          ERROR_CHECK_STATUS( vxAgeDelay( /* Fill in parameter */ ) );
471
```

KHRONOS GROUP

TODO: query performance counters…

```
////////********
// Query graph performance using VX_GRAPH_ATTRIBUTE_PERFORMANCE and print timing
// in milliseconds. Note that time units of vx_perf_t fields are nanoseconds.
//
// TODO STEP 11:********
//   1. Use vxQueryGraph API with VX_GRAPH_ATTRIBUTE_PERFORMANCE to query graph performance.
//      We gave the attribute query for one graph in comments. Do the same for the second graph.
//   2. Print the average and min execution times in milliseconds. Use the printf in comments.
//    vx_perf_t perfHarris = { 0 }, perfTrack = { 0 };
//    ERROR_CHECK_STATUS( vxQueryGraph( graphHarris, VX_GRAPH_ATTRIBUTE_PERFORMANCE, &perfHarris, sizeof( perfHarris ) ) );
//    ERROR_CHECK_STATUS( vxQueryGraph( /* Fill in parameters here for get performance of the other graph */ );
//    printf( "GraphName NumFrames Avg(ms) Min(ms)\n"
//            "Harris    %9d %7.3f %7.3f\n"
//            "Track     %9d %7.3f %7.3f\n",
//            ( int )perfHarris.num, ( float )perfHarris.avg * 1e-6f, ( float )perfHarris.min * 1e-6f,
//            ( int )perfTrack.num,  ( float )perfTrack.avg  * 1e-6f, ( float )perfTrack.min  * 1e-6f );
```

KHRONOS GROUP

TODO: release all objects…

```
////////********
// Release all the OpenVX objects created in this exercise, and make the context as the last one to release.
// To release an OpenVX object, you need to call vxRelease<Object> API which takes a pointer to the object.
// If the release operation is successful, the OpenVX framework will reset the object to NULL.
//
// TODO STEP 12:********
//   1. For releasing all other objects use vxRelease<Object> APIs.
//      You have to release 2 graph objects, 1 image object, 2 delay objects,
//      6 scalar objects, and 1 context object.
//   2. Use ERROR_CHECK_STATUS for error checking.
//    ERROR_CHECK_STATUS( vxReleaseContext( &context ) );
```

KHRONOS GROUP

Q & A

KHRONOS GROUP

Samsung MPI Lab