oneIPL Technical Advisory Board

Tech session #1

December 16th, 2021

Agenda

- Introduction – all TAB members (10 min)
- Technical discussion (45 min)

  oneIPL overview:
  - Programming model
  - Execution model
  - Image processing pipelines
  - Image data abstraction
  - Memory model
- Closing words and next plans (5 min)

https://spec.oneapi.io/oneipl/latest/index.html - oneIPL specification (current version: v0.5)

The oneIPL TAB rules

DO NOT share any confidential information or trade secrets with the group

DO keep the discussion at a High Level

- Focus on the specific Agenda topics
- We are asking for feedback on features for the oneIPL specification (e.g. requirements for functionality and performance)
- We are NOT asking for the feedback on any implementation details

Please submit the feedback in writing on GitHub in accordance to Contribution Guidelines at spec.oneapi.io. This will allow Intel to further upstream your feedback to other standards bodies, including The Khronos Group SYCL specification.

Introduction of TAB members

- **Robert Schneider (PhD)**,
  Principal Key Expert
  Diagnostic Imaging
  Siemens Healthiness

- **SungShik Baik**,
  Principle Engineer, PC engineer
  Ultrasound System R&D
  Samsung Medison

- **Kangsik Kim**,
  Principle Engineer,
  Ultrasound signal processing architect
  Ultrasound System R&D
  Samsung Medison

- **Ashish Uthama**,
  Principal Software Engineer
  Image Processing
  Mathworks

- **Mark Rabotnikov**,
  Lead software engineer,
  Advanced Development group,
  Enterprise Diagnostics Informatics
  Philips

- **Tim van der Horst**,
  C++ Software Designer,
  Interventional Guided Therapy Systems R&D - Imaging & Image Processing
  Philips

- **Sohrab Amirghodsi**,
  Principal Compute Scientist
  Photoshop ART
  Adobe

- **Guoyi Zhou**,
  Head of the Medical Innovation Research Center
  SonoScape

- **Zhilei Zhu**,
  Xinje

- **Victor Getmanskiy**,
  oneIPL architect,
  Intel Performance Libraries,
  Intel

- **Maksim Shabunin**,
  AI Framework Engineer,
  OpenVINO Core Engineering / OpenCV,
  Intel

- **Sergey Ivanov**,
  AI Framework Engineer,
  OpenCV/G-API,
  Intel

oneAPI Image Processing Library (oneIPL)

Domains overview

oneAPI

Currently available in the specification

Can be considered in the future versions of the specification

Basics	Color conversions	Filters	Geometry	3D operations
Image	RGB(A) ↔ NV12	Sobel 3×3	Resize	Resize 3D
Allocators	RGB(A) ↔ RGBP	Gaussian	Mirror	Remap 3D
Accessors	RGBP ↔ NV12	Bilateral	Warp affine	Warp affine 3D
Type conversions	RGBA ↔ RGB	Median	Warp perspective	Median filter 3D
Batch processing	Other formats	Other filters	Other transforms	Other filters 3D

oneIPL programming language

- SYCL 2020 – based on C++17
- oneIPL primitives - class data abstractions + functional API
- API shall be compatible with SYCL 2020 compliant compiler implementation

oneIPL programming model

oneAPI

- Image data, layout, region of interest (ROI) are specified in **ipl::image class**. Layout, data type, and memory are defined at compile-time.
- The supported image formats and data types are defined by the matrix of combinations, each algorithm in the specification contain such matrix.
- Generic layouts is channel count – rows (1,3,4 channels). They are mapped to the formats: 1 – plane or grayscale, 3 – RGB, BGR, 4 – RGBA, BGRA, …
- Additional layouts supported selectively – 3 planes for R, G, B, subsampled YUV formats (like NV12), etc.
- Generic datatypes – 8u-32u – unsigned integer, 8s-32s – signed integer, fp16-fp64 – floating-point

Layout	8u	8s	16u	16s	32u	32s	64u	64s	fp16	fp32	fp64
1 channel											
3 channels											
4 channels											
3 planes											
NV12		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

33 generic image formats

extra formats

oneIPL programming model

oneAPI

- Each function has own type and format combination support matrix in the specification.
- Generic formats are usually supported for multiple devices.
- Some data types are device specific (FP16, FP64). Specification doesn't have the requirement that such API shall work as fallback. For example, if FP16 is not supported on device, it shall not be implemented via FP32. For that case error reporting is used.
- Example: resize_lanczos (support multichannel, NV12, doesn't support 3 planes)

Layout	8u	8s	16u	16s	32u	32s	64u	64s	fp16	fp32	fp64
1 channel	√	√	√	√	√	√	√	√	√	√	√
3 channels	√	√	√	√	√	√	√	√	√	√	√
4 channels	√	√	√	√	√	√	√	√	√	√	√
3 planes	X	X	X	X	X	X	X	X	X	X	X
NV12	√	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

oneIPL programming model

oneAPI

- Image data, layout, region of interest (ROI) are specified in **ipl::image class**. Layout, data type, and memory are defined at compile-time.
- Algorithm is a function called `<function>` which takes input image and output image. Additional arguments are passed as runtime class called `<function>_spec`. API has fixed arguments; spec structure can be extended.
- Example of Gaussian algorithm API

```
template <typename ComputeT = float,
          formats Format,
          typename DataT,
          typename SrcAllocatorT,
          typename DstAllocatorT>
sycl::event gaussian(sycl::queue&                                 queue,
                     image<Format, DataT, SrcAllocatorT>&         src,
                     image<Format, DataT, DstAllocatorT>&         dst,
                     const gaussian_spec_t<Format, DataT, ComputeT>& spec,
                     const std::vector<sycl::event>&              dependencies = {})
```

oneIPL programming model

- ComputeT – type in which computations are done. E.g., implementation can provide faster solution with less precision using sycl::half rather than double data type.
- Each function takes sycl::queue and vector of dependencies.
- Function returns sycl::event which can be used in dependencies vector of other calls.

```
template <typename ComputeT = float,
          formats Format,
          typename DataT,
          typename SrcAllocatorT,
          typename DstAllocatorT>
sycl::event gaussian(sycl::queue&                                         queue,
                     image<Format, DataT, SrcAllocatorT>&                 src,
                     image<Format, DataT, DstAllocatorT>&                 dst,
                     const gaussian_spec_t<Format, DataT, ComputeT>&      spec,
                     const std::vector<sycl::event>&                      dependencies = {})
```

oneIPL execution model

- oneIPL APIs follows SYCL xPU ideology. Each API is able to execute on the range of the devices, if device is supporting required features.
- oneIPL provides primitive algorithms so there is no covering of unsupported paths by different implementation. oneIPL algorithm shall not substitute unsupported type by the different supported type if it impacts the result. If the result precision doesn't matter, user still can write universal API covering all paths depending on device, so the ideology is that API shall be minimal for the range of the devices.
- Example: 2 devices CPU and GPU – float (fp32) is supported, double (fp64) is not supported on GPU, sycl::half (fp16) is supported on GPU.

```
(void)resize_lanczos<float>(queue, input_image, output_image);

(void)resize_lanczos<sycl::half>(queue, input_image, output_image); // valid only for GPU
(void)resize_lanczos<double>(queue, input_image, output_image);     // valid only for CPU
```

oneIPL execution model

oneAPI

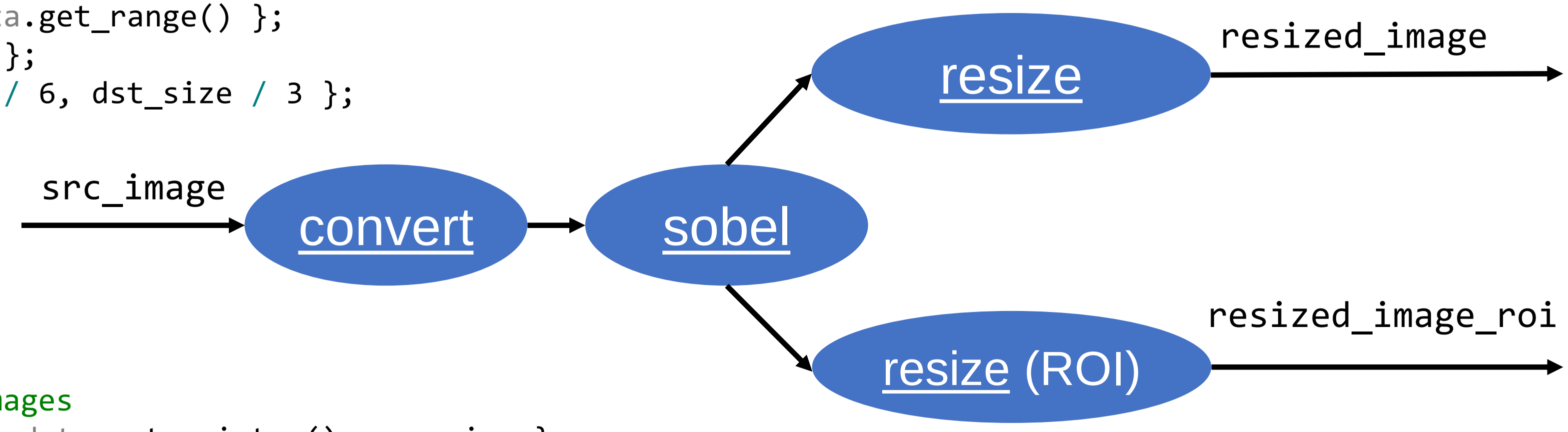
- Execution mode is asynchronous. Algorithms submitted to sycl::queue and returns the control flow. Execution is scheduled by runtime taking into account the dependencies vector.
- For arguments having type ipl::image dependencies are handled automatically. In example below execution is sequential convert → sobel_3x3 → resize_lanzos, since output of previous function is an input of next function.

```
// Source RGBA image data and destination images
image_t<formats::rgba, shared_usm_allocator_t> src_image{ queue,
                                                          src_image_data.get_pointer(),
                                                          src_size,
                                                          shared_allocator };
// Gray image data
image_t<formats::plane, device_usm_allocator_t> gray_image{ src_size, device_allocator };
// Sobel image data
image_t<formats::plane, device_usm_allocator_t> sobel_image{ src_size, device_allocator };
// Resized image data
image_t<formats::plane, shared_usm_allocator_t> resized_image{ dst_size, shared_allocator };

// Run pipeline: convert to grayscale -> sobel -> resize_lanczos
(void)convert(queue, src_image, gray_image);
(void)sobel_3x3(queue, gray_image, sobel_image);
(void)resize_lanczos(queue, sobel_image, resized_image);
```

oneIPL Image processing pipelines

oneAPI

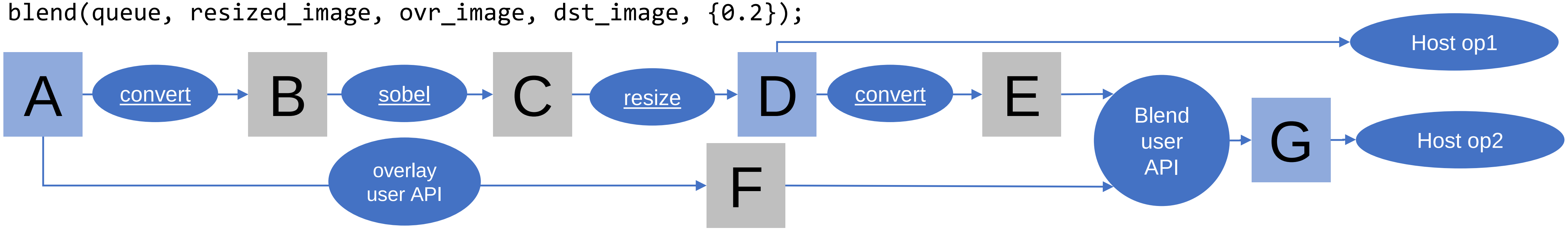
```
template <formats Format>
using image_t = image<Format, std::uint8_t>;

// Size of source and destination buffers after color conversion
const sycl::range<2> src_size{ src_image_data.get_range() };
const auto           dst_size{ 2 * src_size };
const roi_rect       dst_roi_rect{ dst_size / 6, dst_size / 3 };

// Create queue
sycl::queue queue;

// Create allocator
shared_usm_allocator_t allocator{ queue };

// Source RGBA image data and destination images
image_t<formats::rgba>  src_image{ src_image_data.get_pointer(), src_size };
image_t<formats::plane> gray_image{ src_size, allocator };                    // Gray image data
image_t<formats::plane> gray_image_roi{ gray_image, { src_size / 2 } };       // Gray image ROI
image_t<formats::plane> sobel_image{ src_size, allocator };                   // Sobel image data
image_t<formats::plane> sobel_image_roi{ sobel_image, { src_size * 2 / 3 } }; // Sobel image ROI
image_t<formats::plane> resized_image{ dst_size, allocator };                 // Resized image data
auto                    resized_image_roi = resized_image.get_roi(dst_roi_rect); // Resized image ROI

// Run pipeline: convert to grayscale -> sobel -> resize_lanczos -> resize_lanczos
(void)convert(queue, src_image, gray_image_roi);
(void)sobel_3x3(queue, gray_image, sobel_image);
(void)resize_lanczos(queue, sobel_image_roi, resized_image);
(void)resize_lanczos(queue, sobel_image_roi, resized_image_roi);
```



oneIPL Image processing pipelines

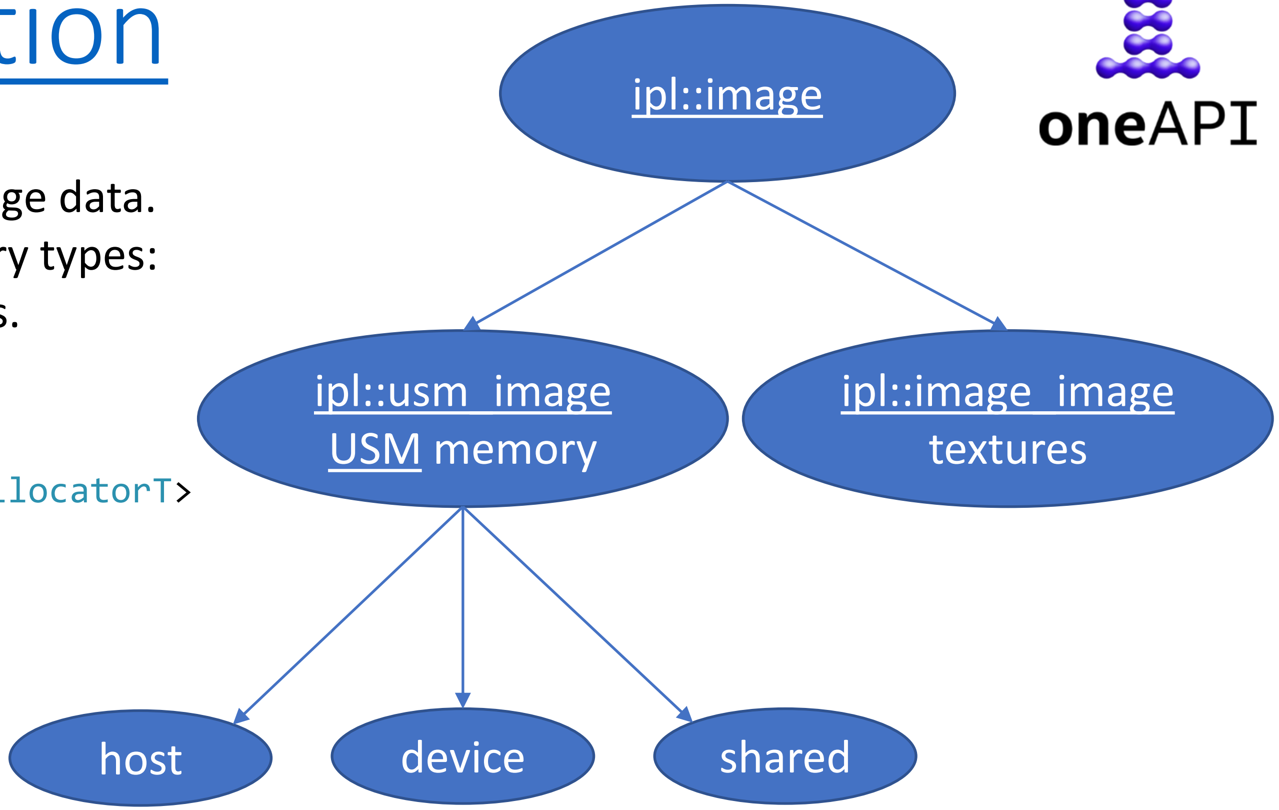
```
// Creation of I/O for pipeline steps
// Source RGBA image data and destination images (USM-based)
ipl::image<formats::rgba,  uint8_t> src_image{ src_image_data.get_pointer(), src_size, image_alloc }; // A (image)
ipl::image<formats::plane, uint8_t> gray_image{ src_size, device_alloc };                              // B (device USM)
ipl::image<formats::plane, uint8_t> sobel_image{ src_size, device_alloc };                             // C (device USM)
ipl::image<formats::plane, uint8_t> resized_image{ dst_size, shared_alloc };                           // D (shared USM)
ipl::image<formats::rgba,  uint8_t> res_rgba_image{ dst_size,  image_alloc };                          // E (image)
ipl::image<formats::rgba,  uint8_t> ovr_image{ dst_size, image_alloc };                                // F (image)
ipl::image<formats::rgba,  uint8_t> dst_image{ dst_size, image_alloc };                                // G (image)

// Run pipeline: convert to grayscale -> sobel -> resize_lanczos
generate_overlay(queue, src_image, ovr_image.get_usm_pointer(), dst_size);
convert(queue, src_image, gray_image);
sobel_3x3(queue, gray_image, sobel_image);
resize_lanczos(queue, sobel_image, resized_image);
convert(queue, resized_image, res_rgba_image);
blend(queue, resized_image, ovr_image, dst_image, {0.2});
```

Legend		
	A	Input/output
	B	Temporary memory
	operation	Operation



oneIPL Image Abstraction

oneapi::ipl::image class is basic data abstraction for image data. oneIPL provides single abstraction over different memory types: host, device, shared and special GPU memory – textures.




```
template <formats Format, typename DataT, typename AllocatorT>
class image final
{
public:
    using image_impl_t::image_impl_t;
    using allocator_t = AllocatorT;
    using pixel_t = pixel_layout_t<DataT, Format>;

    image(const image_impl_t& image_impl);
    image(image_impl_t&& image_impl);
    auto operator=(const image_impl_t& image_impl)->image&;
    auto operator=(image_impl_t&& image_impl)->image&;

    auto get_whole_image() const->image;
    auto get_roi(const roi_rect& roi_rect) const->image;
};
```

oneIPL Memory Model

oneAPI

- oneIPL supports different type of memory – device, host, shared and special GPU texture (image) memory. Memory type is accessible explicitly via user allocation or via allocator passed to **ipl::image** constructor, if no external memory is provided. Allocator argument is a special tag. By default, allocator is selected as shared or texture depending on supported formats depending on device.

```
template <formats Format, typename DataT, typename AllocatorT =
select_image_allocator_t<Format, DataT>>
class image;
```

oneIPL Memory Model

oneAPI

- oneIPL supports 4 allocators which targets image memory allocation on host, device, shared or as a texture (image):

```
host_usm_allocator_t   host_allocator{ queue };   // memory is on host
shared_usm_allocator_t shared_allocator{ queue }; // memory is on device and host
device_usm_allocator_t device_allocator{ queue }; // memory is temporary (device only)
image_allocator_t      device_allocator{ queue }; // memory is texture (image)

ipl::image<formats::rgba,  uint8_t, host_usm_allocator_t> host_image{ size, host_allocator };
ipl::image<formats::rgba,  uint8_t, shared_usm_allocator_t> shared_image{ size, shared_allocator };
ipl::image<formats::rgba,  uint8_t, device_usm_allocator_t> device_image{ size, device_allocator };
ipl::image<formats::rgba,  uint8_t, image_allocator_t> texture_image{ size, image_allocator };

// doesn't work, since texture in SYCL 2020 supports only 4-channel images
ipl::image<formats::plane,  uint8_t, image_allocator_t> texture_image{ size, image_allocator };
```

oneIPL Memory Model

oneAPI

- Image can be mapped over memory, but some cases shall be supported by device defined by sycl::queue object:

```
auto* device_ptr = sycl::malloc_device<std::uint8_t>(size[0] * size[1], queue);
auto* shared_ptr = sycl::malloc_shared<std::uint8_t>(size[0] * size[1], queue);

ipl::image<formats::rgba, std::uint8_t, shared_usm_allocator_t> shared_image{  queue,
device_ptr, size, shared_allocator };
ipl::image<formats::rgba, std::uint8_t, device_usm_allocator_t> device_image{  queue,
shared_ptr, size, device_allocator };

// if texture from device memory is supported by device, copy occurred, if not – would not work
ipl::image<formats::plane,  uint8_t, image_allocator_t> texture_image{ queue, device_ptr,
size, image_allocator };
```

Next Steps

- All materials and minutes of meetings will be published on [GitHub](https://github.com/oneapi-src/oneAPI-tab) and will be available for the offline review (the offline feedback of invited TAB members will be also processed and discussed on next TAB meeting)
- The next technical discussion:

February 3rd (ww6)

Find more on https://spec.oneapi.io/versions/latest/introduction.html#contribution-guidelines
https://github.com/oneapi-src/oneAPI-tab

oneIPL Technical Advisory Board meetings

The goal is to provide the feedback and define future development of the specification.

First topics planned to discuss are at the table below, but it might be adjusted later.

Topic	Plan	Date
1) oneIPL overview	1) Programming model 2) Execution model 3) Image processing pipelines 4) Image data abstraction 5) Memory model	
2) oneIPL Image data abstraction	1) HW images and data formats and types coverage 2) IPL image data abstraction 3) Interoperability with USM 4) Memory allocation and temporary images	February 3rd, 2022
3) oneIPL Library design details	1) Domains 2) Reference code and optimized backends 3) Error handling mechanism 4) Interoperability with other oneAPI libraries	
4) oneIPL Functions overview	1) ML oriented APIs for image preprocessing 2) Data type support in the functions 3) Color formats and conversions	

Resources

- https://www.oneapi.io/spec/ - oneAPI Specification
- https://spec.oneapi.io/oneipl/latest/index.html - oneIPL specification (current version: v0.5)
- https://github.com/oneapi-src/oneAPI-tab - GitHub with oneAPI TAB materials
- https://spec.oneapi.io/versions/latest/introduction.html#contribution-guidelines - oneAPI Specification contribution guidelines