oneIPL Technical Advisory Board

Tech session #3

February 17th, 2022

Agenda

- Introduction & open questions (10 min)
- Technical discussion (45 min)
    1. oneIPL Memory allocation and temporary images
    2. oneIPL Domains
    3. oneIPL Error handling mechanism
- 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**,
  Computer Vision Algorithm Architect
  Xinje

- **Yizhi Li,**
  Computer Vision Software Architect
  HuaRay

- **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

oneIPL specification

- SYCL 2020 – based on C++17
- oneIPL primitives - class data abstractions + functional API
- API shall be compatible with SYCL 2020 compliant compiler implementation
- Current provisional spec version is 0.5, the spec v0.6 is in progress

oneIPL specification

What's new is coming in oneIPL spec v0.6:

- Replace ipl::formats -> ipl::layouts
- Image constructors changed to remove dependency on implementation
- Default image allocator shall be USM
- Methods to image auxilliary classes moved to image API
- Switched to generic template parameters
- Gaussian filter with separated sigma for x and y axis
- Normalize without sycl::buffer in spec

Results of previous discussion of API

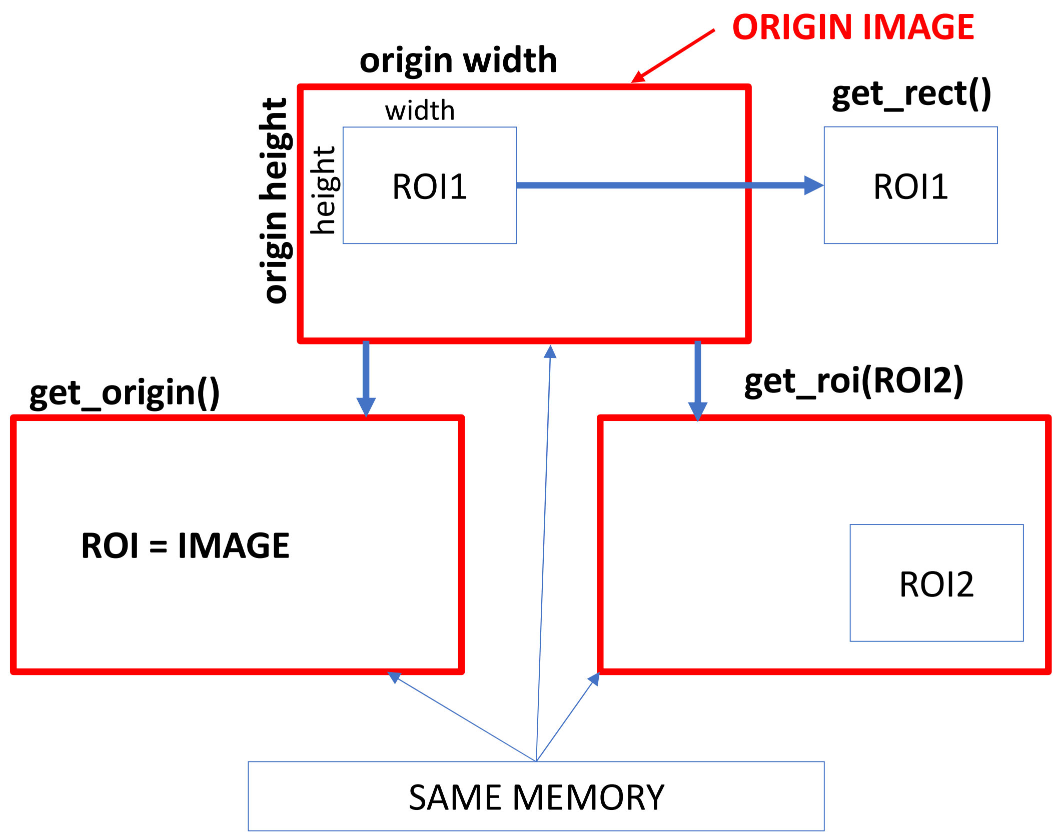
Accessors are introduced to directly return references to the data or use SYCL runtime to do that.

```
template <layouts Layout, typename DataT, typename AllocatorT>
class image final
{
public:
// … consructors, memory related (allocator, pitch, size, etc.)

    data_t* get_pointer();                // pointer (USM only)
    template <typename AccessDataT, sycl::access_mode AccessMode>
    __unspecified__ get_access(args);          // some access abstraction common
for USM and Image(Texture)
    template <typename AccessDataT, sycl::access_mode AccessMode>
    __unspecified__ get_access(sycl::handler& command_group_handler);
// __unspecified__ is not a language-related, but spec-related placeholder, since
the type can be defined differently for particular class specialization in the
implementation.

//  Size metadata access is reduced to minimal API (get rect)
// (Thanks for feedback to make that decision!)
    const roi_rect& get_rect() const; // ROI rectangle
    image           get_origin() const->image;     // returns origin image
    image           get_roi(const roi_rect& roi_rect) const; // returns new ROI
};

// Size accessors are in the roi_rect:
struct roi_rect
{
...
    std::size_t get_x_offset() const;
    std::size_t get_y_offset() const;
    std::size_t get_width() const;
    std::size_t get_height() const;
};
```



oneIPL Memory Model

oneIPL spec supports different type of memory assigned via AllocatorT or provided via pointer by user to image constructor:

- **host USM**
- **device USM,**
- **shared USM,**
- **image (texture).**

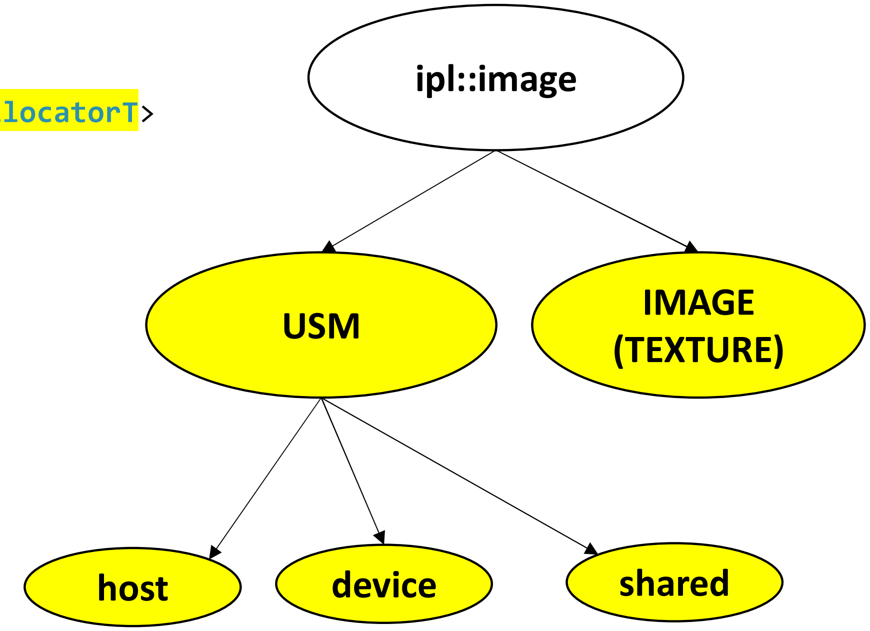
```
template <layouts Layout, typename DataT, typename AllocatorT>
class image final;
```

Memory in image class is controlled via template parameter **AllocatorT.**

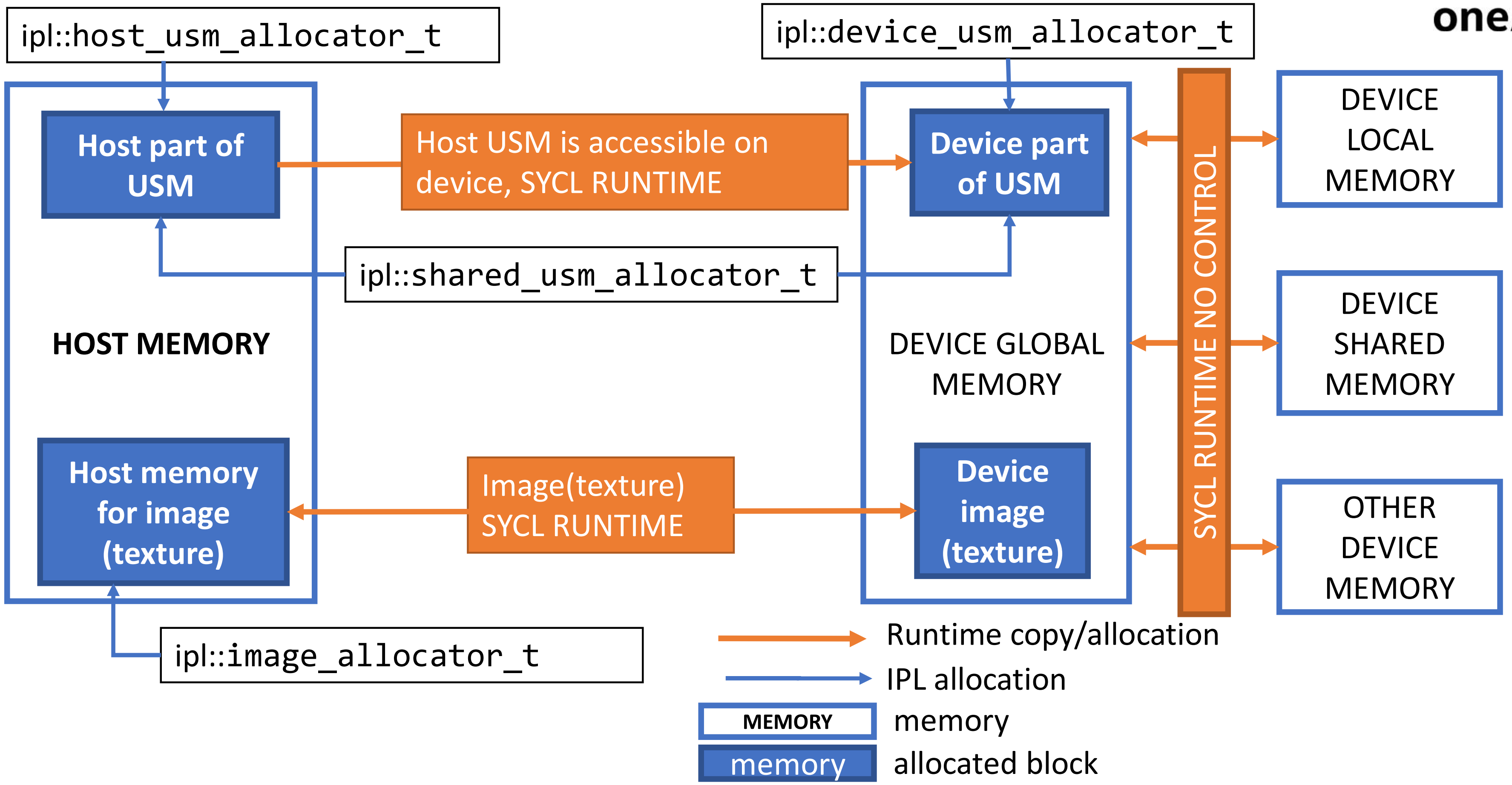
AllocatorT is an allocator type, **shared by default**:
In USM case - targeted to host, shared and device.
In image (texture) case - always a host allocator.

USM allocator has a major difference from std::allocator.

Allocator cannot be constructed without context, so device context is required. It complicates the APIs, since allocator has no default constructor.



oneIPL Memory Model



oneIPL Memory Model

Important change. In spec v0.5 default allocator was implementation-defined. For compatibility in spec v0.6 the default, allocator is selected as shared USM.

Possible implementation – trait select_image_allocator or direct use shared_allocator_t as default value for template parameter:

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

oneIPL Memory Model – temporary images

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

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

ipl::image<layouts:channel4, uint8_t, host_usm_allocator_t> host_image{ size, host_allocator };
ipl::image<layouts:channel4, uint8_t, shared_usm_allocator_t> shared_image{ size, shared_allocator };
ipl::image<layouts:channel4, uint8_t, device_usm_allocator_t> device_image{ size, device_allocator };
ipl::image<layouts:channel4, uint8_t, image_allocator_t> texture_image{ size, image_allocator };
```

Images in pipeline which shall not be transferred to the host must have device allocator as argument.

oneIPL Memory Model – temporary images

oneAPI

- If the memory is initially on host:

```
ipl::image<layouts:channel4, uint8_t, shared_usm_allocator_t> shared_image{ q, p_host, size, shared_allocator };
ipl::image<layouts:channel4, uint8_t, device_usm_allocator_t> tmp_image_1{ size, device_allocator };
ipl::image<layouts:channel4, uint8_t, device_usm_allocator_t> tmp_image_2{ size, device_allocator };
ipl::[function_1] (queue, shared_image, device_image_1);

// if the operation doesn't change the layout, swapping I/O for 2 temporary device images
// 2 images on device are required, no inplace operations!
ipl::[function_2] (q, tmp_image_1, tmp_image_2);
ipl::[function_3] (q, tmp_image_2, tmp_image_1);
ipl::[function_4] (q, tmp_image_1, tmp_image_2);

// final image can be written back to initial shared image, so the host pointer would be overwritten by rutime
ipl::[function_N] (queue, tmp_image_2, shared_image);
ipl::image<layouts:channel4, uint8_t, image_allocator_t> texture_image{ size, image_allocator };
```

oneIPL Memory Model – temporary images

oneAPI

- If the memory is initially on device:

```
ipl::image<layouts:channel4, uint8_t, device_usm_allocator_t> device_image{ q, p_device, size, device_allocator };
ipl::image<layouts:channel4, uint8_t, device_usm_allocator_t> tmp_image{ size, device_allocator };
ipl::image<layouts:channel4, uint8_t, shared_usm_allocator_t> shared_image{ size, shared_allocator };
ipl::[function_1] (queue, shared_image, device_image_1);

// if the operation doesn't change the layout, swapping I/O for 2 device images
// 2 images on device are required, no inplace operations!
ipl::[function_2] (q, device_image, tmp_image);
ipl::[function_3] (q, tmp_image, device_image);
ipl::[function_4] (q, device_image, tmp_image);

// final image can be written back to shared image, and data will be accessible on host
ipl::[function_N] (queue, tmp_image, shared_image);
ipl::image<layouts:channel4, uint8_t, image_allocator_t> texture_image{ size, image_allocator };
```

oneIPL Memory Model

- Image can be mapped over memory, but some cases shall be supported by device associated with 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<layouts:channel4, std::uint8_t, shared_usm_allocator_t> shared_image{ queue, device_ptr, size };
ipl::image<layouts:channel4, std::uint8_t, device_usm_allocator_t> device_image{ queue, shared_ptr, size };

// layout is not supported by image(texture) – would not work
ipl::image<layouts::plane,  uint8_t, image_allocator_t> texture_image_bad{ queue, host_ptr, size };

// layout is supported by image (texture) – works!!!
ipl::image<layouts:channel4,  uint8_t, image_allocator_t> texture_image{ queue, host_ptr, size };
```

oneIPL Domains

Currently available in the specification v0.6

Can be considered in the future versions of the specification

Basics

- Image
- Allocators
- Accessors
- Type conversions
- Batch processing

Color conversions

- RGB(A) ↔ NV12
- RGB(A) ↔ RGBP
- RGB(A) ↔ GRAY
- RGBA ↔ RGB
- Other formats

Filters

- Sobel 3×3
- Gaussian
- Bilateral
- Median
- Other filters

Geometry

- Resize
- Mirror
- Warp affine
- Warp perspective
- Other transforms

3D operations

- Resize 3D
- Remap 3D
- Warp affine 3D
- Median filter 3D
- Other filters 3D

Color conversions Domain

- Header : **include/oneapi/ipl/convert.hpp**
- Conversion functions in spec v0.6:

oneapi::ipl::rgba_to_rgb
oneapi::ipl::rgb_to_rgba
oneapi::ipl::gray_to_rgb
oneapi::ipl::rgb_to_gray
oneapi::ipl::rgb_to_rgbp
oneapi::ipl::rgbp_to_rgb

Subsampled formats:

oneapi::ipl::rgb_to_nv12
oneapi::ipl::nv12_to_rgb
oneapi::ipl::rgb_to_i420
oneapi::ipl::i420_to_rgb

Possible future extension to:

oneapi::ipl::yv12_to_rgb (YUV422)
oneapi::ipl::yvu9_to_rgb (YUV444)

Color conversions Domain example

```
shared_usm_allocator_t usm_allocator{ queue };

// Source gray image data
image<layouts::plane, std::uint8_t> src_image{
     queue, src_image_data.get_pointer(), src_size, usm_allocator };
// Destination rgba image data
image<layouts::channel4, std::uint8_t> dst_image{ src_size };

oneapi::ipl::gray_to_rgb(queue, src_image, dst_image);

// Destination rgba image data with custom alpha-channel value = 127
oneapi::ipl::gray_to_rgb(queue, src_image, dst_image, color_conversion_spec{127});
```

Color conversions Domain API

```
template <typename SrcImageT,
          typename DstImageT>
sycl::event gray_to_rgb(sycl::queue&                                          queue,
                        SrcImageT&                                            src,
                        DstImageT&                                            dst,
                        const color_conversion_spec<typename DstImageT::data_t>& spec         = {},
                        const std::vector<sycl::event>&                       dependencies = {});

template <typename ComputeT  = float,
          typename SrcImageT,
          typename DstImageT>
sycl::event rgb_to_gray(sycl::queue&                                          queue,
                        SrcImageT&                                            src,
                        DstImageT&                                            dst,
                        const color_conversion_spec<typename DstImageT::data_t>& spec         = {},
                        const std::vector<sycl::event>&                       dependencies = {});
```

- Conversion functions have common spec:

```
template <typename DataT>
class color_conversion_spec {
public:
    constexpr color_conversion_spec(DataT a_value = max_color_v<DataT>) noexcept;
    constexpr DataT get_alpha_value() const noexcept;
};
```

- Alpha value is used to fill an alpha channel if required for conversion to RGBA

Filtering Domain

- Header : **include/oneapi/ipl/filter.hpp**
- Filtering functions in spec v0.6:

oneapi::ipl::sobel_3x3

oneapi::ipl::gaussian

Possible future extension to:

oneapi::ipl::bilateral

oneapi::ipl::sobel

oneapi::ipl::filter

oneapi::ipl::filter_box

oneapi::ipl::filter_median

...

Filtering Domain API

```
template <typename ComputeT = float,
          typename SrcImageT,
          typename DstImageT>
sycl::event gaussian(sycl::queue&                           queue,
                     SrcImageT&                             src,
                     DstImageT&                             dst,
                     const gaussian_spec_t<…>& spec,
                     const std::vector<sycl::event>& dependencies = {})
```

Functions have individual parameters:

```
template <…>
class gaussian_spec : public border_spec_base<…> {
...
    explicit gaussian_spec(std::size_t   radius,
                           sigma_t       sigma,
                           border_types  border = border_types::repl,
                           crop_types    crop   = crop_types::on);
    explicit gaussian_spec(std::size_t   radius,
                           border_types  border = border_types::repl,
                           crop_types    crop   = crop_types::on);
...
}
```

Gaussian filter specific parameters

Transformations Domain

- Header : **include/oneapi/ipl/transform.hpp**
- Conversion functions in spec v0.6:

oneapi::ipl::resize_bilinear

oneapi::ipl::resize_bicubic

oneapi::ipl::resize_lanczos

oneapi::ipl::resize_supersampling

oneapi::ipl::mirror

Possible future extension to:

oneapi::ipl::resize_nearest

oneapi::ipl::warp_affine_bilinear

…

Transformations Domain API

```
template <typename ComputeT = float,
          typename SrcImageT,
          typename DstImageT
          >
sycl::event resize_bicubic(sycl::queue&                          queue,
                           SrcImageT&                            src,
                           DstImageT&                            dst,
                           const resize_bicubic_spec<ComputeT>& spec         = {},
                           const std::vector<sycl::event>&      dependencies = {})
```

- Functions have individual specs:

```
template <typename ComputeT = float>
class resize_bicubic_spec : public border_spec_base<> {
public:
    explicit resize_bicubic_spec(ComputeT     b_factor,
                                 ComputeT     c_factor,
                                 border_types border = border_types::repl,
                                 crop_types   crop   = crop_types::on)
…
}
```

Bicubic specific parameters

Error Handling – common flow

- oneAPI error handling mechanism is C++ exceptions.

```
Common oneAPI application flow:
    try {
        oneapi::ipl::[algorithm]([args]);
    }
    catch (...) {
        exception_handler();
    }
```

- oneIPL additionally has a requirements to implement compile-time checks which are possibly based on template parameters

Error Handling – common flow

```
try {
    oneapi::ipl::[algorithm]([args]);
}
catch (oneapi::ipl::exception const& e) {
    std::cout << "\t\tCaught oneIPL exception: "
        << e.what() << std::endl;
}
catch (sycl::exception const& e) {
    std::cout << "\t\tCaught synchronous SYCL exception: "
        << e.what() << std::endl;
}
catch (std::exception const& e) {
    std::cout << "\t\tCaught synchronous STL exception: "
        << e.what() << std::endl;
}
```

Standard ASYNC exception handling

```
    auto my_exception_handler = [](sycl::exception_list exceptions) {
        for (std::exception_ptr const& e : exceptions) {
            try {
                std::rethrow_exception(e);
            }
            catch (sycl::exception const& e) {
                std::cout << "Caught asynchronous SYCL exception:"
                    << e.what() << std::endl;
            }
            catch (std::exception const& e) {
                std::cout << "Caught asynchronous STL exception:"
                    << e.what() << std::endl;
            }
        }
    };
    // create execution queue on my gpu device with exception handler
attached
    sycl::queue my_queue(my_device, my_exception_handler);
```

Error Handling – exception types

- oneAPI error handling mechanism is C++ exceptions.
- Exception types:

oneapi::ipl::exception – base class for oneIPL exceptions

sycl::exception – base class for SYCL exceptions

std::exception – base class for non-SYCL and non-library exceptions

Error Handling – exception types

Exception class	Description
oneapi::ipl::exception	Abstract class, base for all other oneIPL exception classes
oneapi::ipl::logic_error	Abstract class, base for all oneIPL exception classes on logic errors (that are a consequence of faulty logic within the program)
oneapi::ipl::runtime_error	Abstract class, base for all oneIPL exception classes on runtime errors (that are due to events beyond the scope of the program)
oneapi::ipl::invalid_argument	Reports a problem when arguments to the routine were rejected
oneapi::ipl::domain_error	Reports a problem when inputs are outside of the domain on which an operation is defined
oneapi::ipl::out_of_range	Reports a problem when there is an attempt to access element(s) out of defined range
oneapi::ipl::range_error	Reports a problem when a result of a computation cannot be represented by the destination type
oneapi::ipl::bad_alloc	Base class for all oneIPL exception classes for bad allocation errors, reports a problem when an allocation function has failed to allocate storage
oneapi::ipl::host_bad_alloc	Reports a problem that occurred during memory allocation on the host
oneapi::ipl::device_bad_alloc	Reports a problem that occurred during memory allocation on a specific device
oneapi::ipl::unsupported_device	Reports a problem when the routine is not supported on a specific device
oneapi::ipl::unimplemented	Reports a problem when a specific routine has not been implemented for the specified parameters
oneapi::ipl::uninitialized	Reports a problem when an object has not been initialized

Error Handling – example from spec

Example of errors description in oneIPL spec. Compile-time checks which are possible to be done comparing template parameters and exceptions with conditions.

compile-time memory layout check	Indicates an error when image memory layout is not supported.
compile-time data type check	Indicates an error when image data type is not supported.
compile-time compute data type check	Indicates an error when compute data type is not supported.
invalid_argument exception	Indicates an error when one of the pitch values is not divisible by size of component data type in bytes.
unimplemented exception	Indicates an error when border type is not supported.
unimplemented exception	Indicates an error when bicubic filter factors are not supported.

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 meetings)
- The next technical discussion: March 3rd

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	December 16th, 2021
2) oneIPL Image data abstraction	1) HW images and data formats and types coverage 2) IPL image data abstraction 3) Interoperability with USM	February 3rd, 2022
3) oneIPL Library design details	1) Memory allocation and temporary images 2) Domains 3) Error handling mechanism	February 17th, 2022
4) oneIPL Functions overview	1) Interoperability with other oneAPI libraries 2) ML oriented APIs for image preprocessing 3) Data type support in the functions 4) Reference code and optimized backends	March 3rd, 2022

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