13th International Workshop on OpenCL and SYCL

IWOCL 2025

OpenCL

SYCL

Write Once, Deploy Many – 3D Rendering With SYCL Cross-Vendor Support and Performance Using Blender Cycles

Stefan Werner, Intel

Xavier Hallade, ph0b.com

Nikita Sirgienko and Sebastian Herholz, Intel

Malon Przemek, Codeplay Software

Agenda

- Introduction on Blender and Cycles
- Blender Cycles code overview
- Experimental but critical extensions
- Maintaining and Shipping Blender with SYCL
- Getting a multi-vendors build using SYCL
- Results

Blender

- 3D editing and rendering application with Millions of users
- Two renderers
  - EEVEE (GL/Vulkan/Metal) and Cycles (CPU/GPGPU)
- A Benchmark using Cycles
  - opendata.blender.org
- 3-4 versions to support in parallel
  - currently 3.6 LTS, 4.2 LTS, 4.4
- Broad end-users support
  - from 10y old laptops to latest and future high-end Workstation and Datacenter GPUs

History and Evolution of Cycles

- Path tracing physically based render engine
- Introduced in 2011 (Blender 2.61) supporting CPU and CUDA
- Initial implementation just one large kernel
- Refactored in 2021 (Blender 3.0) to a wavefront/microkernel approach (“Cycles X”)
  - higher occupancy
  - sorting between kernels for more coherent memory access
  - reduced compile time
  - lower register pressure
  - despite that, still large kernels

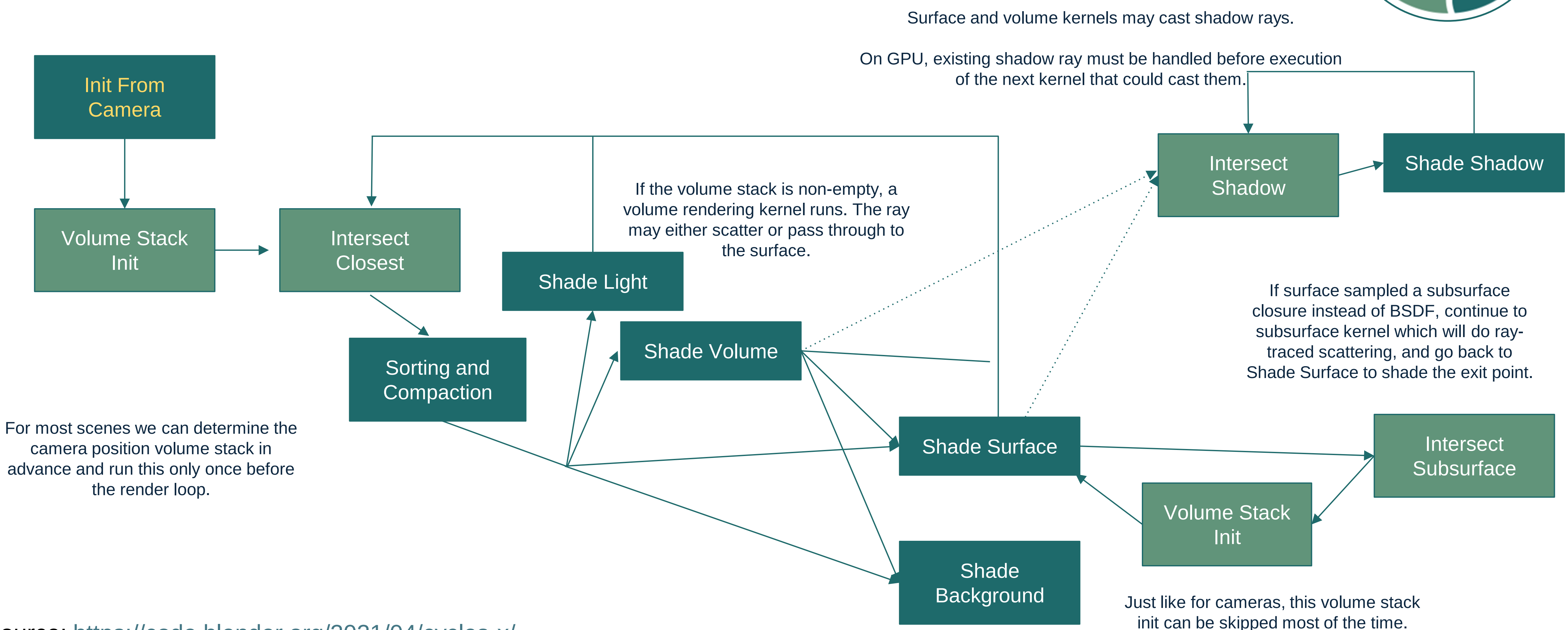
OpenCL and SYCL in Cycles

- Initial OpenCL support first released in 2015 (Blender 2.75)
- Split the kernel into a few smaller ones due to compiler bugs
- Still very unstable support, highly sensitive to driver versions
- Discrepancies between OpenCL and CUDA code
- Removed in 2021 (v3.0):
  "The combination of the limited Cycles split kernel implementation, driver bugs, and stalled OpenCL standard has made maintenance too difficult."
- v3.0 release with CPU, CUDA and HIP support
- SYCL backend added in 2022 (v3.3) for Intel Arc GPUs launch

Cycles Code overview

- Kernels written in C++ headers with own types and abstractions
  - 36 different kernels
  - state is periodically compacted and sorted
  - simple in-order queue
- *Almost* no differences across targets
- Backend specific code:
  - Compatibility header for kernels
  - Memory operations
  - Kernel launch
  - Error handling

Kernels Graph



Source: https://code.blender.org/2021/04/cycles-x/

compat.h snippet

```
#define ccl_gpu_thread_idx_x
(sycl::ext::oneapi::this_work_item::get_nd_item<1>().get_local_id(0))
#define ccl_gpu_global_id_x()
(sycl::ext::oneapi::this_work_item::get_nd_item<1>().get_global_id(0))
#define ccl_gpu_global_size_x()
(sycl::ext::oneapi::this_work_item::get_nd_item<1>().get_global_range(0))
#define ccl_gpu_warp_size
(sycl::ext::oneapi::this_work_item::get_sub_group().get_local_range()[0])
#define ccl_gpu_syncthreads()
sycl::ext::oneapi::this_work_item::get_nd_item<1>().barrier()
…
#define ccl_gpu_ballot(predicate) \
  (sycl::ext::oneapi::group_ballot(sycl::ext::oneapi::this_work_item::get_sub_group(),
predicate) \
       .count())
…
```

Complete version available in `intern/cycles/kernel/device/oneapi/compat.h`

Launching Kernels

```
try {
  queue->submit([&](sycl::handler &cgh) {
    switch (device_kernel) {
      case DEVICE_KERNEL_INTEGRATOR_RESET:
        oneapi_call(kg, cgh, global_size, local_size, args, oneapi_kernel_integrator_reset);
        break;
        …
    }
  }
}
```

`oneapi_call` goes through macros and templates leading to processed code such as:

```
void oneapi_kernel_integrator_reset(KernelGlobalsGPU *ccl_restrict kg,
                                    size_t kernel_global_size,
                                    size_t kernel_local_size,
                                    sycl::handler &cgh,
                                    int num_states)
{
  cgh.parallel_for<class kernel_integrator_reset>(
    sycl::nd_range<1>(kernel_global_size, kernel_local_size), [=](sycl::nd_item<1> item) {
      const int state = ccl_gpu_global_id_x();
      …
    })
  );
}
```

180K instructions shade_surface

- Evaluates user authored shader graph
- Shaders executed in a stack based virtual machine
- float[255] stack
- Switch statement for 98 node types
- Pre-sorting by shader ID to reduce divergence

- Kernel with high register pressure
- Despite sorting, still divergent due to Monte Carlo sampling and different light sources
- Long compile times, large binaries
- Challenging for compiler and hardware
- Execution mostly memory latency bound

Important extensions for Blender

- Bindless Textures
  - experimental, used in Blender 4.4
- Device Globals
  - experimental, very recent, targeting use in Blender 4.5
- free_memory (intel_device_info)
  - supported, used since Blender 4.2
- Vulkan Interoperability
  - experimental, very recent, targeting use in Blender 4.5
- Additional: group_local_memory, this_work_item, group_ballot

Bindless Textures

- The number of textures does not need to be known at compile time
- Access to fixed function hardware for texture interpolation and cache
- One call replaces four hundred of lines of code
- Textures can be stored in plain memory

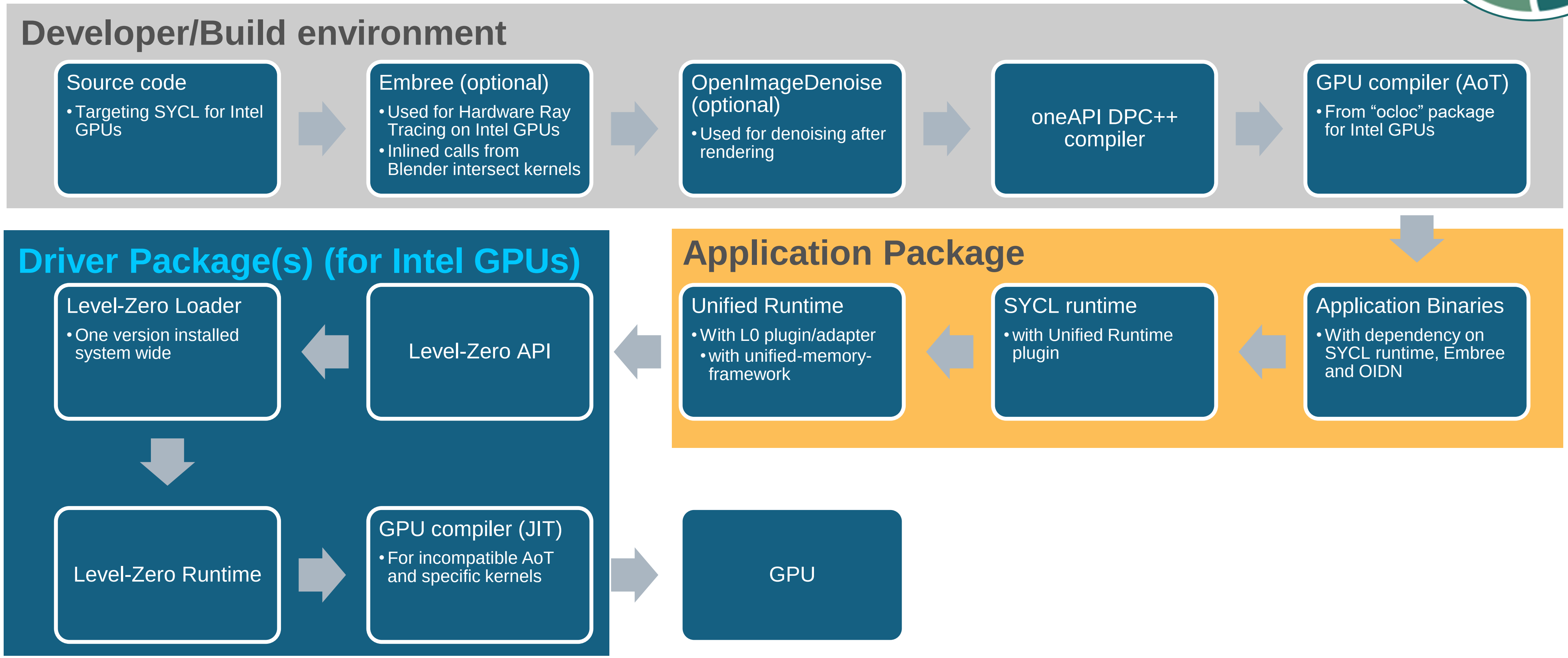
For a deeper dive into bindless textures, join this session on Friday, 11:15 – 11:45:

SYCL Interoperability with DirectX and Vulkan via Bindless Images
Duncan Brawley, Przemyslaw Malon, Jack Kirk, Georgi Mirazchiyski and Peter Žužek, Codeplay Software.

Device Global

- Original CUDA code makes use of __constant__ globals
- Our initial implementation:
  - put them into a wrapper class
  - extra level of indirection for loads at runtime
  - Stored in regular global memory
- With device globals:
  - sycl::device_global no longer requires wrapper class
  - Can use dedicated constant cache on NVIDIA GPUs
  - More opportunities for compiler and hardware optimizations

Maintaining and Shipping Blender with SYCL



Application Package MUST run on current <u>and future</u> Drivers and Hardware

Blender Requirements

1. Open-Source, GPLv3 compatible application side components
2. No mandatory runtime dependencies outside of OS
   - Optionally calling into driver libraries: yes. Anything else: no
3. Support for a wide range of Operating Systems and GPUs:
   - Windows (x64 and arm64), Linux (also with “old” ABI), Mac OS
   - Nvidia, AMD, Intel, Apple GPUs... open to other OSes and GPUs
4. Compatible with vendor tools for debugging and profiling
5. Broad and long term hardware support
6. Compatible with future driver and hardware releases for 2+ years
7. Easy to download and setup in CI and on developer machines
8. Well documented application deployment
   - redistributables, driver requirements, OS support, bug tracking
9. No change of main application compiler and linker

Other Important features for Blender

1. Multiple AoT GPU binaries per target
   - currently supported only for Intel devices
2. Device binaries compression
   - must ensure compiler is built with `LLVM_ENABLE_ZSTD=FORCE_ON`
3. Hardware Ray Tracing
   - native SYCL library for Intel GPUs (Embree)
   - vendor specific for other GPUs (HIP RT, OptiX), not compatible with SYCL
   - Vulkan Ray Tracing is cross-platform, but cannot be efficiently mixed with SYCL

CMake Integration

- Written before any native CMake support
- `clang++ -fsycl` compiler called using `add_custom_command` and `cmake -E env`
- Used only for a standalone library: `cycles_kernel_oneapi`
- `CYCLES_ONEAPI_SYCL_TARGETS` values passed to `-fsycl-targets`
- `CYCLES_ONEAPI_SYCL_OPTIONS_`*`sycl_target`* value passed to `-Xsycl-target-backed=`*`sycl_target`*

implementation visible in `./intern/cycles/kernel/CMakelists.txt`

Compiling and Running on more GPUs

1. Use oneAPI DPC++ compiler with L0, CUDA and HIP support
2. Set Blender CMake options:

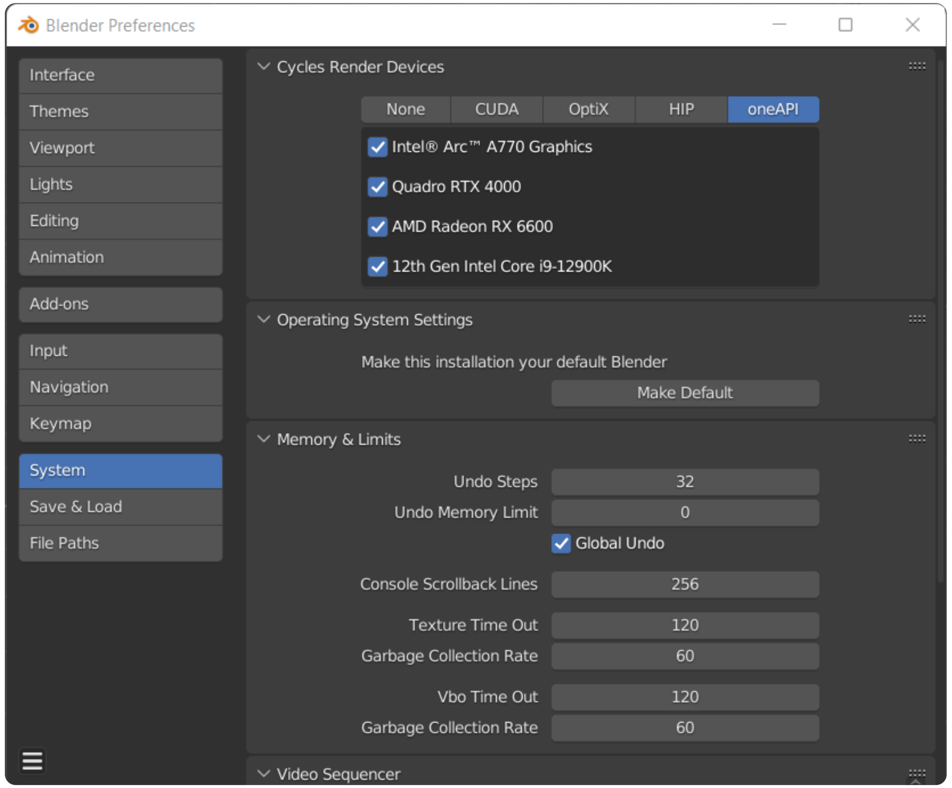
```
CYCLES_ONEAPI_SYCL_TARGETS=
amdgcn-amd-amdhsa;nvptx64-nvidia-cuda;spir64_gen
```

AMD
```
CYCLES_ONEAPI_SYCL_OPTIONS_amdgcn-amd-amdhsa=
--offload-arch=gfx1032
```

NVIDIA
```
CYCLES_ONEAPI_SYCL_OPTIONS_nvptx64-nvidia-cuda=
--offload-arch=sm_75
```

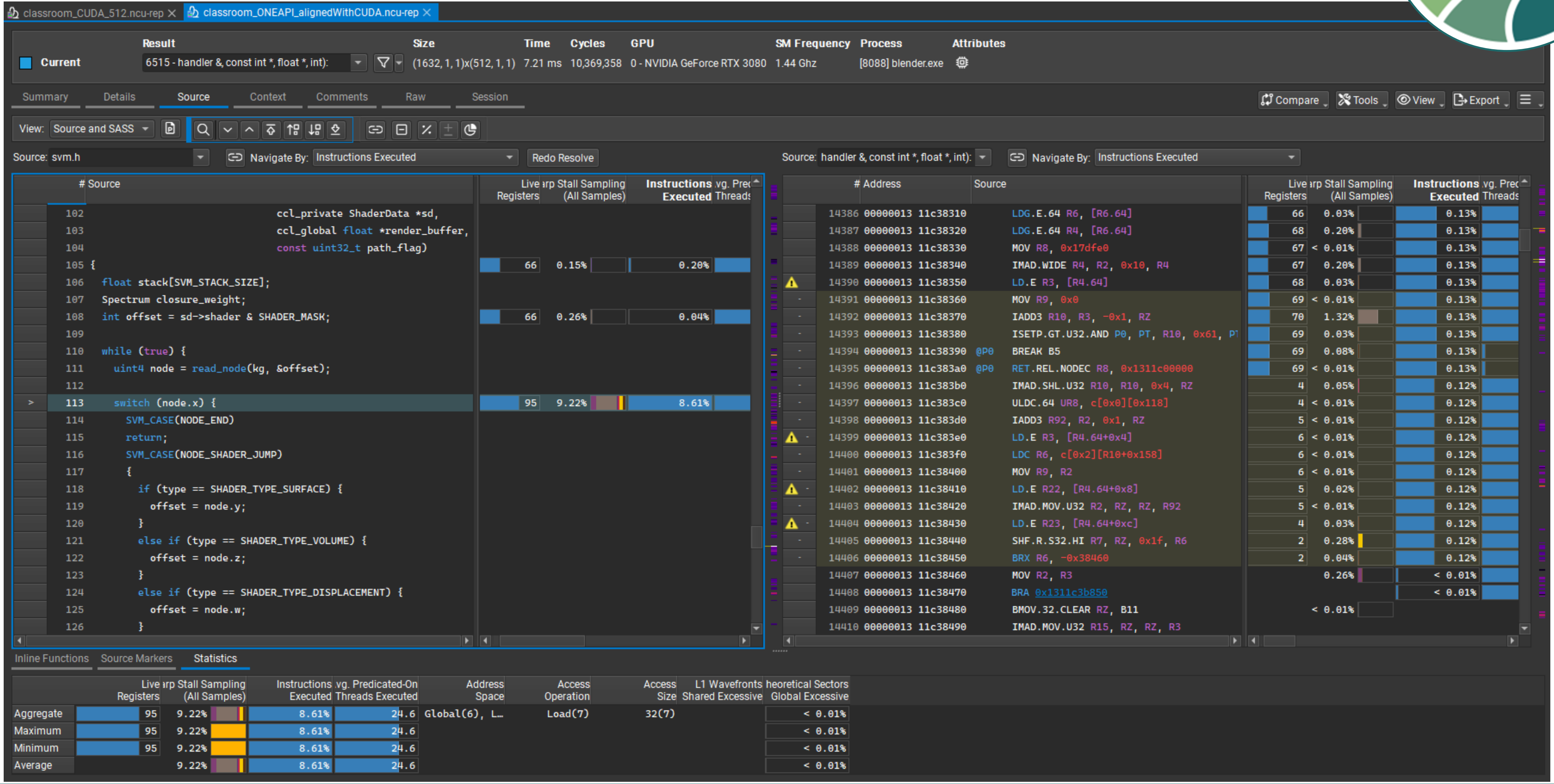
3. At runtime, set environment variable to allow using devices that aren't officially supported by Blender

```
CYCLES_ONEAPI_ALL_DEVICES=1
```

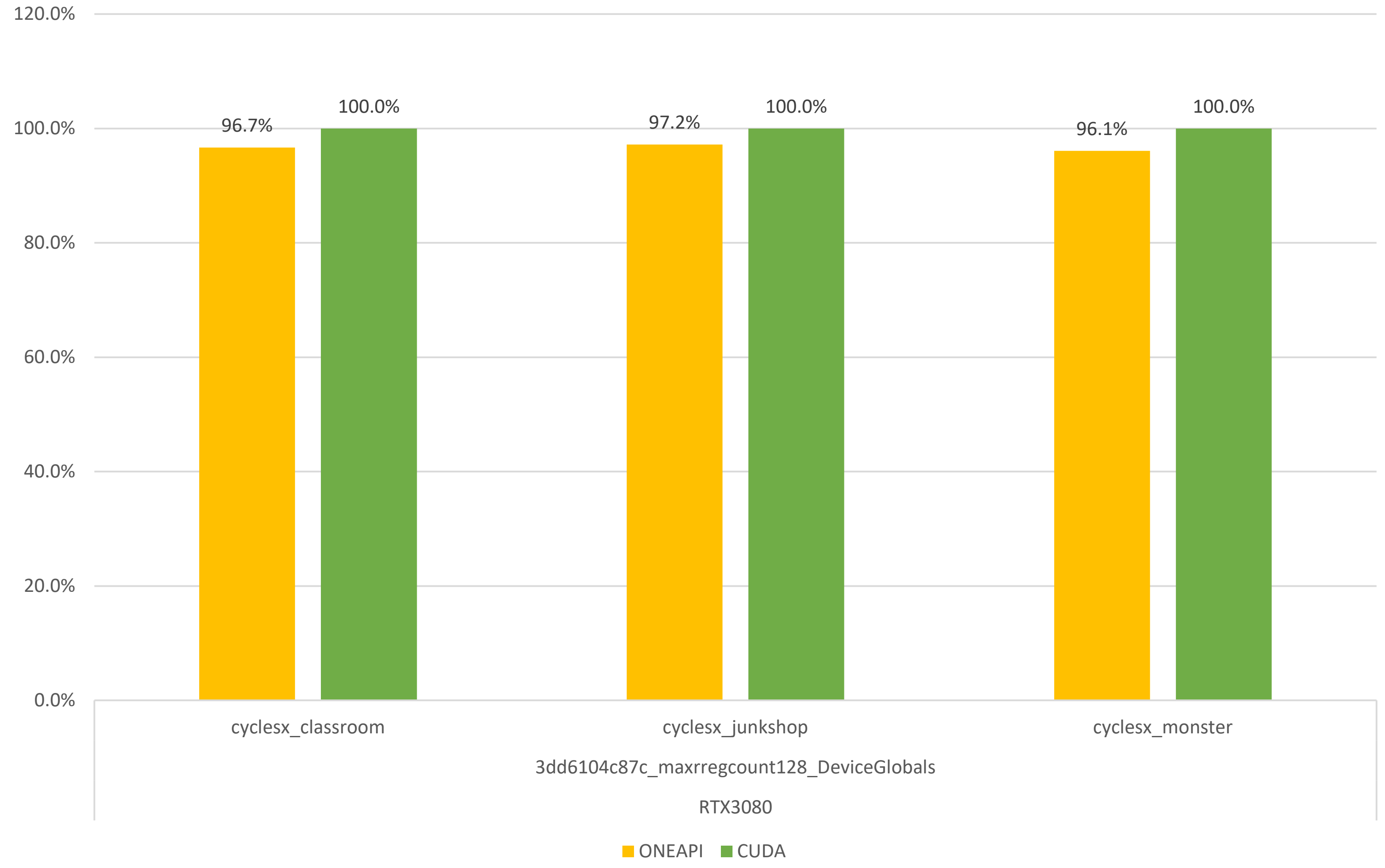


implementation visible in `./intern/cycles/kernel/CMakelists.txt`

Tooling


classroom_CUDA_512.ncu-rep
classroom_ONEAPI_alignedWithCUDA.ncu-rep
Result | Size | Time | Cycles | GPU | SM Frequency | Process | Attributes
Current | 6515 - handler &, const int *, float *, int): | (1632, 1, 1)x(512, 1, 1) | 7.21 ms | 10,369,358 | 0 - NVIDIA GeForce RTX 3080 | 1.44 Ghz | [8088] blender.exe
Summary | Details | Source | Context | Comments | Raw | Session
Compare | Tools | View | Export
View: Source and SASS
Source: svm.h | Navigate By: Instructions Executed | Redo Resolve
Source: handler &, const int *, float *, int): | Navigate By: Instructions Executed
Source | Live Registers | Warp Stall Sampling (All Samples) | Instructions Executed | Avg. Predicated-On Threads
102 ccl_private ShaderData *sd,
103 ccl_global float *render_buffer,
104 const uint32_t path_flag)
105 { | 66 | 0.15% | 0.20%
106 float stack[SVM_STACK_SIZE];
107 Spectrum closure_weight;
108 int offset = sd->shader & SHADER_MASK; | 66 | 0.26% | 0.04%
109
110 while (true) {
111 uint4 node = read_node(kg, &offset);
112
113 switch (node.x) { | 95 | 9.22% | 8.61%
114 SVM_CASE(NODE_END)
115 return;
116 SVM_CASE(NODE_SHADER_JUMP)
117 {
118 if (type == SHADER_TYPE_SURFACE) {
119 offset = node.y;
120 }
121 else if (type == SHADER_TYPE_VOLUME) {
122 offset = node.z;
123 }
124 else if (type == SHADER_TYPE_DISPLACEMENT) {
125 offset = node.w;
126 }
Address | Source
14386 00000013 11c38310 LDG.E.64 R6, [R6.64]
14387 00000013 11c38320 LDG.E.64 R4, [R6.64]
14388 00000013 11c38330 MOV R8, 0x17dfe0
14389 00000013 11c38340 IMAD.WIDE R4, R2, 0x10, R4
14390 00000013 11c38350 LD.E R3, [R4.64]
14391 00000013 11c38360 MOV R9, 0x0
14392 00000013 11c38370 IADD3 R10, R3, -0x1, RZ
14393 00000013 11c38380 ISETP.GT.U32.AND P0, PT, R10, 0x61, PT
14394 00000013 11c38390 @P0 BREAK B5
14395 00000013 11c383a0 @P0 RET.REL.NODEC R8, 0x1311c00000
14396 00000013 11c383b0 IMAD.SHL.U32 R10, R10, 0x4, RZ
14397 00000013 11c383c0 ULDC.64 UR8, c[0x0][0x118]
14398 00000013 11c383d0 IADD3 R92, R2, 0x1, RZ
14399 00000013 11c383e0 LD.E R3, [R4.64+0x4]
14400 00000013 11c383f0 LDC R6, c[0x2][R10+0x158]
14401 00000013 11c38400 MOV R9, R2
14402 00000013 11c38410 LD.E R22, [R4.64+0x8]
14403 00000013 11c38420 IMAD.MOV.U32 R2, RZ, RZ, R92
14404 00000013 11c38430 LD.E R23, [R4.64+0xc]
14405 00000013 11c38440 SHF.R.S32.HI R7, RZ, 0x1f, R6
14406 00000013 11c38450 BRX R6, -0x38460
14407 00000013 11c38460 MOV R2, R3
14408 00000013 11c38470 BRA 0x1311c3b850
14409 00000013 11c38480 BMOV.32.CLEAR RZ, B11
14410 00000013 11c38490 IMAD.MOV.U32 R15, RZ, RZ, R3
Inline Functions | Source Markers | Statistics
| Live Registers | Warp Stall Sampling (All Samples) | Instructions Executed | Avg. Predicated-On Threads Executed | Address Space | Access Operation | Access Size | L1 Wavefronts Shared Excessive | Theoretical Sectors Global Excessive
Aggregate | 95 | 9.22% | 8.61% | 24.6 | Global(6), L... | Load(7) | 32(7) | < 0.01%
Maximum | 95 | 9.22% | 8.61% | 24.6 | < 0.01%
Minimum | 95 | 9.22% | 8.61% | 24.6 | < 0.01%
Average | 9.22% | 8.61% | 24.6 | < 0.01%

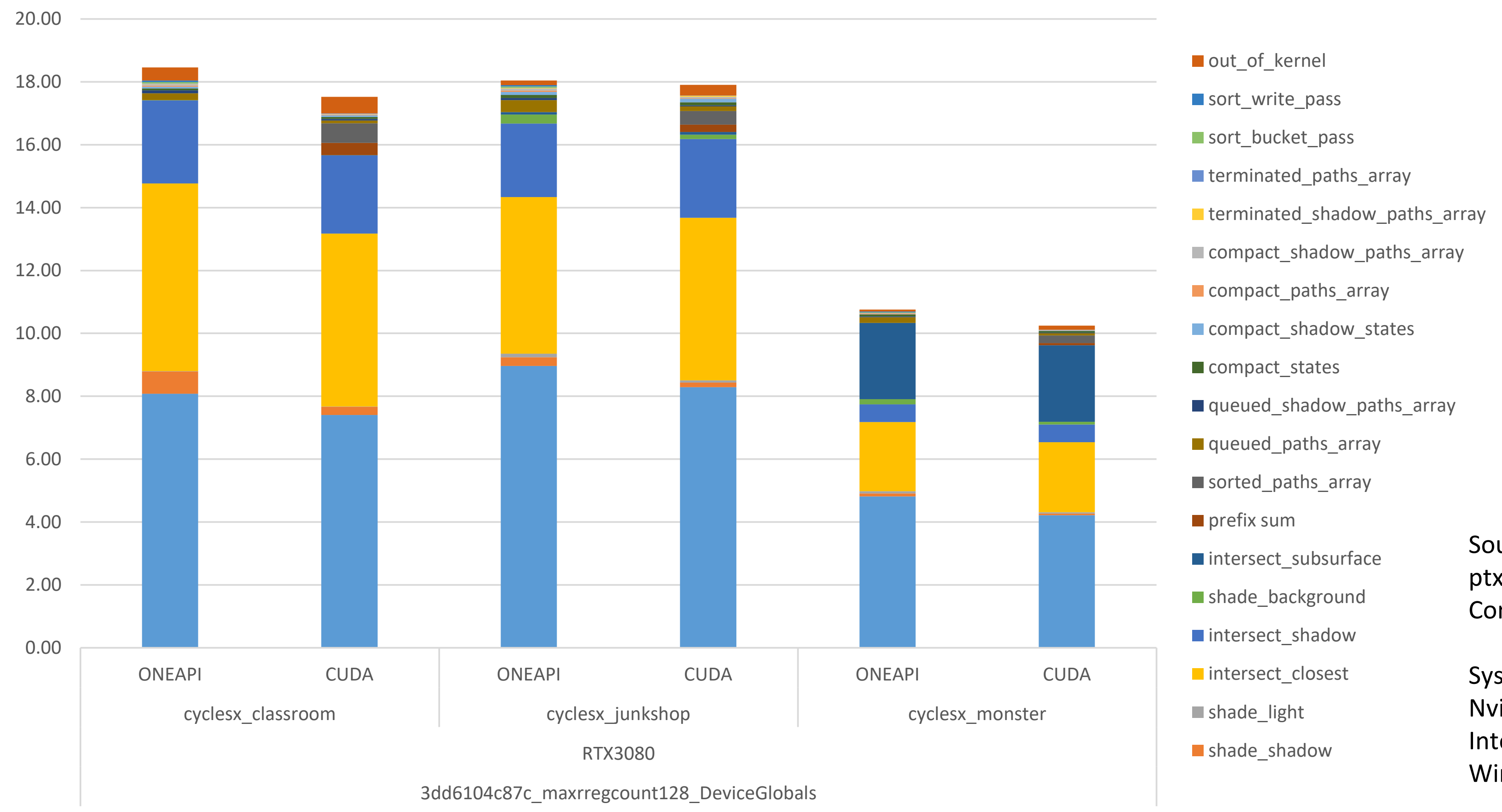
Scores of Blender Benchmark scenes on Nvidia RTX3080 normalized on CUDA device results



Source: Blender 4.5 alpha 3dd6104c87c with -Xcuda-ptxas --maxrregcount=128 and Device Globals
Compiled with CUDA 12.8 SDK

System:
Nvidia RTX 3080 with drivers 560.94
Intel Core i9-10980XE
Windows 24H2

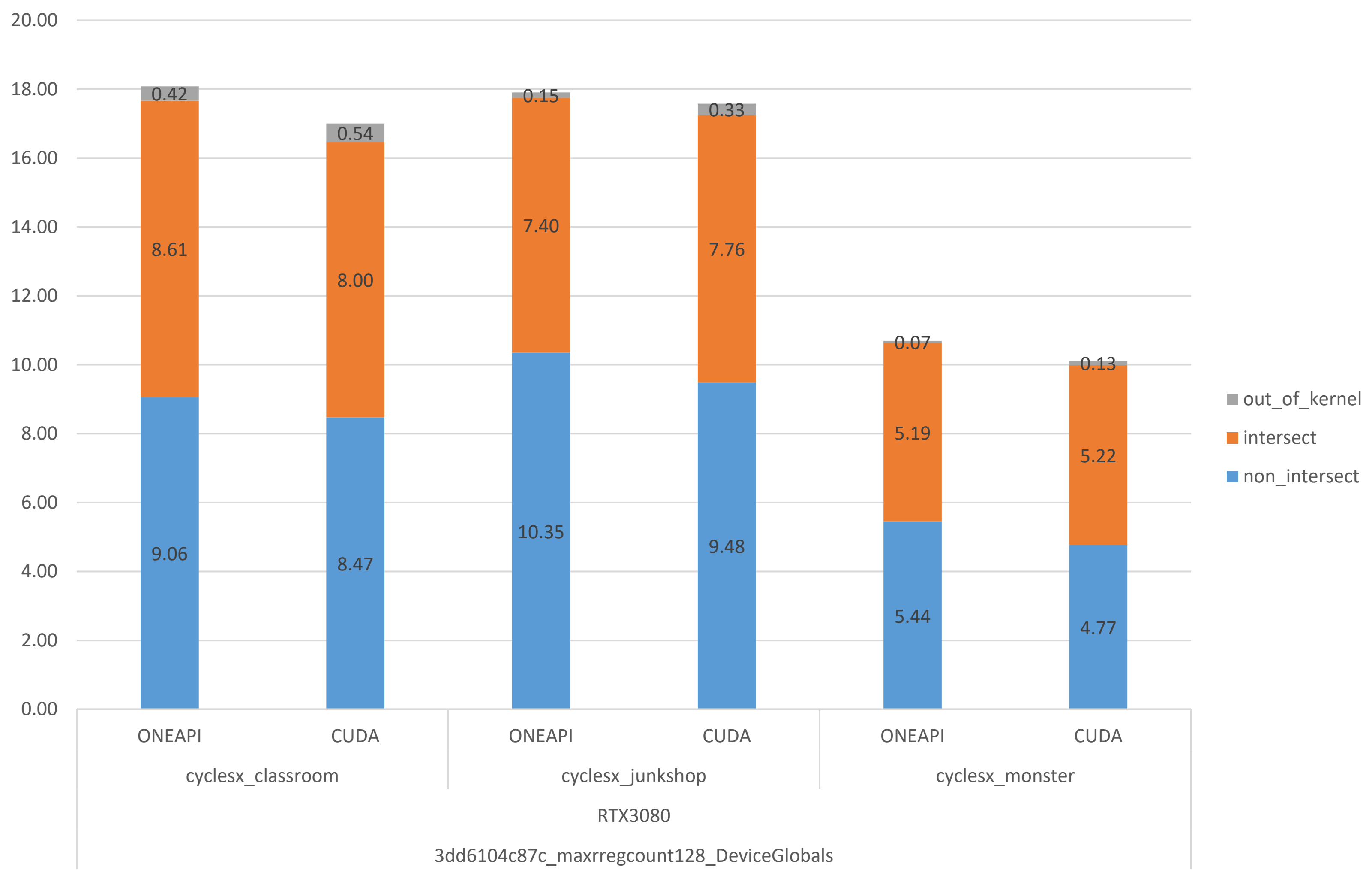
Per-Kernels execution in seconds



Source: Blender 4.5 alpha 3dd6104c87c with -Xcuda-ptxas --maxrregcount=128 and Device Globals
Compiled with CUDA 12.8 SDK

System:
Nvidia RTX 3080 with drivers 560.94
Intel Core i9-10980XE
Windows 24H2

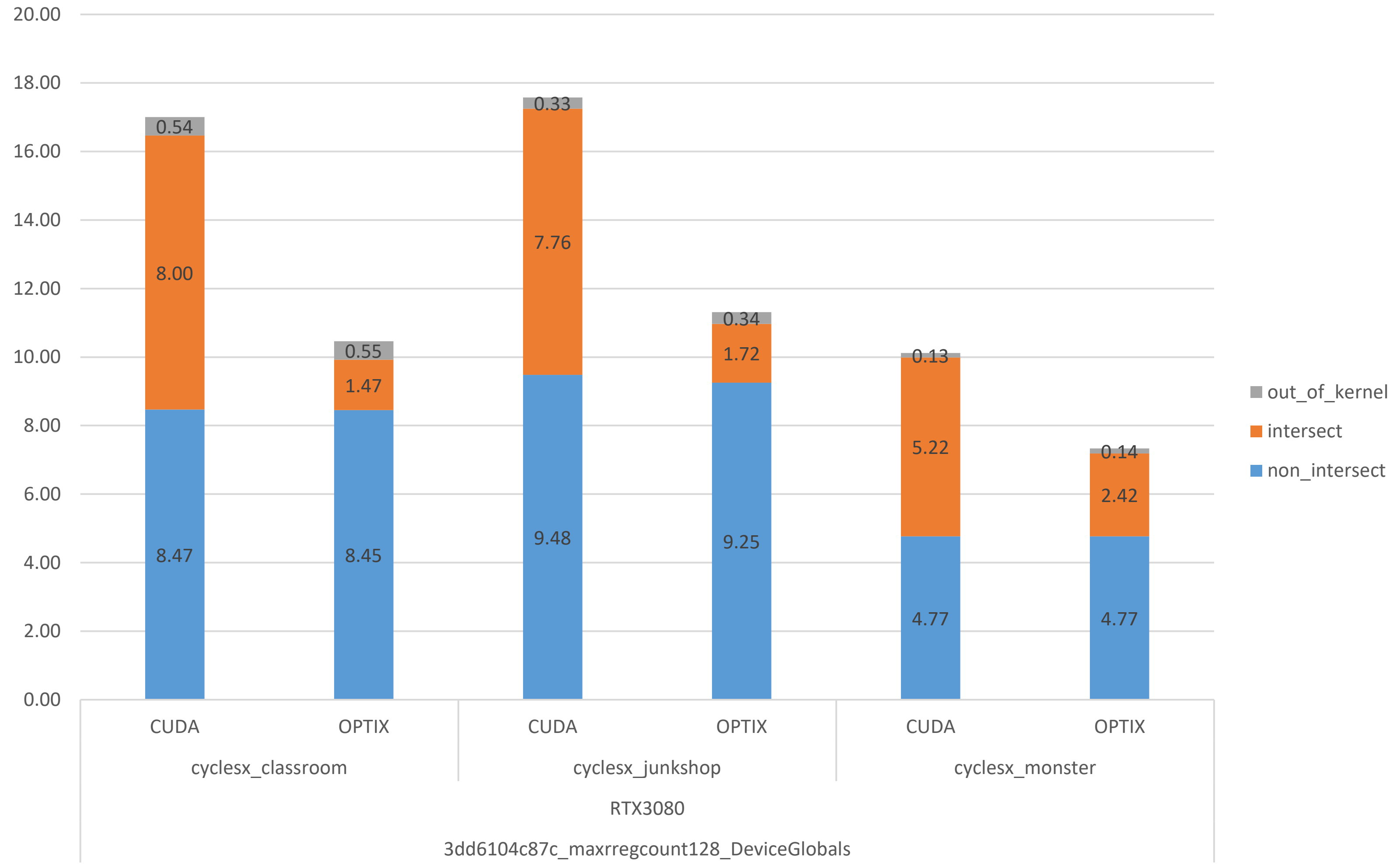
Per-Kernels execution in seconds, simplified



Source: Blender 4.5 alpha 3dd6104c87c with -Xcuda-ptxas --maxrregcount=128 and Device Globals
Compiled with CUDA 12.8 SDK

System:
Nvidia RTX 3080 with drivers 560.94
Intel Core i9-10980XE
Windows 24H2

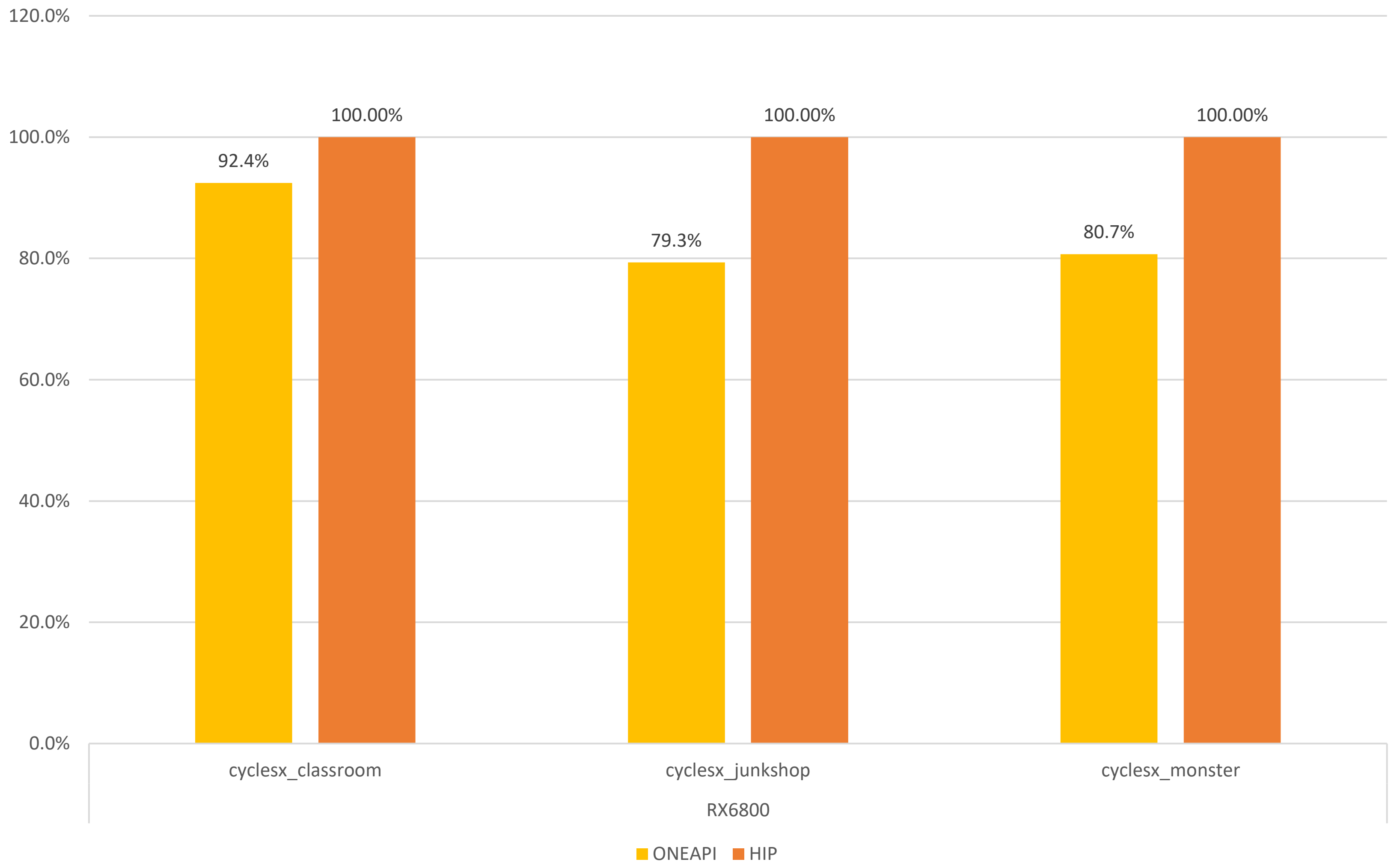
CUDA vs OptiX, Per-Kernels execution in seconds



Source: Blender 4.5 alpha 3dd6104c87c with -Xcuda-ptxas --maxrregcount=128 and Device Globals Compiled with CUDA 12.8 SDK

System:
Nvidia RTX 3080 with drivers 560.94
Intel Core i9-10980XE
Windows 24H2

Scores of Blender Benchmark scenes on AMD RX6800 normalized on HIP device results



Source: Blender 4.5 alpha 3dd6104c87c with Device Globals
ROCM 6.31

System:
AMD Radeon RX 6800 with drivers 24.3.0.60301
Intel Core i9-13900K
Ubuntu 24.04

Conclusion

- SYCL shipping in production through Blender for Intel GPUs since 2022, and getting better every year
- Large real-world codebase able to target Level-Zero, HIP, CUDA devices with competitive performance on Linux and Windows
- Open-Source implementation: projects.blender.org/blender/blender/src/branch/main/intern/cycles
- Key features are available only through extensions at the moment
  - Whether you're implementing SYCL or using SYCL, don’t overlook them

Tips and Tricks

- Math functions can be native (fast) or from library (slow)
  - -ffast-math, sycl::native::*, etc have an influence
  - verify by inspecting PTX assembly
  - red flag: .f64 instructions when using only single precision float
  - Godbolt for small reproducers: godbolt.org/z/Kc7xjr8aG
- Play with `-Xcuda-ptxas --maxrregcount=N`
- Differentiating targets can still be done
  - `#ifdef __NVPTX__, __AMDGPU__, __SPIRV__`

13th International Workshop on OpenCL and SYCL

IWOCL 2025

OpenCL SYCL

Write Once, Deploy Many – 3D Rendering With SYCL Cross-Vendor Support and Performance Using Blender Cycles

Stefan Werner, Intel
Xavier Hallade, ph0b.com

Nikita Sirgienko and Sebastian Herholz, Intel
Malon Przemek, Codeplay Software