Speeding up rendering in Blender with Cycles X
Cycles

- Path tracing renderer, integrated in Blender
- Open source, Apache 2 license
- Goal: modern production renderer for everyone
- CPU and GPU rendering
- 10 years after the initial release, in need of an architecture refresh
Cycles X

New microkernel architecture for GPU rendering, with goals:

- Improved GPU occupancy and performance over megakernel
- Reduce runtime kernel compile times
- Reduce code duplication and maintenance cost
Devices

- GPU rendering on multiple operating systems and devices
  - CPU
  - CUDA and OptiX for NVIDIA GPUs
  - HIP for AMD GPUs
  - Metal for Apple
- Ongoing collaboration with developers from NVIDIA, AMD, Intel and Apple
- Device abstraction layer with shared C++ kernel code
Kernels
Path Tracing and Next Event Estimation

- Init From Camera
- Intersect Closest
- Shade Surface
- Shade Background

Main path:
- Intersect Shadow
- Shade Shadow

Shadow path:
- Initialization Kernel
- Intersection Kernel
- Shading Kernel
Kernel Scheduling

Path States

- Few million paths active at the same time
- Separate arrays for main and shadow paths
- Each path state is inactive or has an kernel assigned to execute next
- Structure-of-arrays storage
- Minimize amount of memory usage per state
Kernel Scheduling

Main Loop

Mark all paths as inactive

While any pixel samples to render:
  If fewer than half of paths are active:
    Gather array of inactive path indices
    Execute init_from_camera to activate paths

Find kernel that most active paths need to execute next
Gather array of active path indices with this kernel
Execute kernel
Kernel Scheduling

Compaction

- Path state array gets fragmented as paths become inactive
  - Less coherent memory access
  - More divergent execution
- Regularly compact states so all active paths are at start of the array
  - Before activating new main paths
  - Before kernels that can activate shadow paths
- Only when fragmentation is high enough
Kernel Scheduling

Shader Sorting

- Execute Shade Surface kernel sorted by shader for coherent execution
- Atomic shader counter with number of active paths for each shader
- Create active path arrays sorted by shader index, using prefix sum

Path states

Shader counter

Prefix sum

Sorted indices
Kernel Scheduling

Shadow Paths

- Main paths create fully independent shadow paths
- Handle transparency from surfaces and volumes
- Intersection and shading in separate kernels
  - Record N closest transparent intersections
  - Shade N intersections
  - Repeat
- Optimization for transparent hair
  - Transparency baked at curve control points
  - Accumulated immediately in intersection kernel
Kernels
Volumes and Subsurface Scattering

- Init From Camera
  - Volume Stack Init
  - Intersect Closest
  - Shade Volume
    - Shade Surface
      - Shade Background
        - Intersect Subsurface
      - Intersect Shadow
        - Shade Shadow
Kernel Scheduling

Subsurface Scattering

- Diffusion and random walk subsurface scattering
- Intersect Subsurface kernel traces many bounces within one object, without overhead of scheduling many kernels
- Shading and lighting at enter and exit points both handled with Shade Surface kernel
Kernel Scheduling

Shadow Mattes

- Differential rendering for creating shadow and indirect light mattes
- Object and lights in scene marked as synthetic or real
- Trace path with and without synthetic objects, and compute difference
- Split main path when hitting real object
Kernel Scheduling

OptiX

- Intersection kernels are OptiX, others pure CUDA
- Helped reduce OptiX runtime compilation to less than a second for many scenes
- For ray-tracing in shaders (ambient occlusion and bevel), still need to compile bigger kernel with OptiX
Progressive Rendering

- **Switch from more tile-oriented to progressive rendering**
  - Support rendering algorithms like path guiding

- **Fine tune scheduling**
  - Batching together more samples once the first few samples are done
    - Also for images with few very expensive pixels (long paths in hair)
  - Automatically pick resolution to meet target FPS in interactive rendering
  - Balancing cost of path tracing and denoising
  - Asynchronous updates for display with graphics APIs

- **Still support tiling for high resolution renders and many render passes**
  - Cached to disk, with denoising and post processing after rendering
Multi Device

- CPU + GPU and Multi GPU performance not scaling ideally yet
- Challenging for interactive and progressive render with unpredictable performance for each device
- Currently split image into parts and dynamically rebalance based on performance of previous sample
- Working on better load balancing by using small alternating slices, and tweaks to initial guess and rebalancing algorithm
Results

- Released in Blender 3.0
- On NVIDIA GPUs, 2-7x speedup over previous megakernel implementation
- Much improved viewport interactivity
Final Words

Work is continuing on improved rendering algorithms and more production features in the new architecture.

Cycles is an open source project and everyone is welcome to get involved. https://www.cycles-renderer.org/development/

Many thanks to all contributors over the years!
AGENDA

Motivation

Hydra

Implementation

Hydra in Omniverse
MOTIVATION

- Blackbird Hydra Render Delegate from Tangent Animation
  
  https://github.com/tangent-opensource/hdBlackbird

- Difficult to keep fork up-to-date with upstream

- Goal to make a Hydra Render Delegate part of mainline Cycles
HYDRA ARCHITECTURE
HYDRA ARCHITECTURE

- UsdImaging
- Scene Delegate
- Render Delegate
- Render Index
- hdCycles
- ...
HYDRA CLASSES AND PRIMITIVES

- **HdRenderDelegate**
  - **HdRprim**
    - **HdMesh**
    - **HdBasisCurve**
    - **HdPoints**
    - **HdVolume**
  - **HdSprim**
  - **HdBprim**
    - **HdCamera**
    - **HdMaterial**
    - **HdLight**
  - **HdRenderParam**
  - **HdRenderPass**
  - **HdRenderBuffer**
  - **HdCyclesMesh**
    - `ccl::Mesh`
  - **HdCyclesBasisCurve**
    - `ccl::Hair`
  - **HdCyclesPoints**
    - `ccl::PointCloud`
  - **HdCyclesVolume**
    - `ccl::Volume`
  - **HdCyclesCamera**
    - `ccl::Camera`
  - **HdCyclesMaterial**
    - `ccl::Shader`
  - **HdCyclesLight**
    - `ccl::Light`
  - **HdCyclesSession**
    - `ccl::Session`
IMPLEMENTATION
SCENE UPDATES
Multithreading

- Cycles already multithreading ready
- Rendering tasks run on separate render thread
- Hydra notifies Render Delegate when scene primitive changed
- Protect access to scene data structures
  - See C++ RAII wrapper called SceneLock in the code
- Only pull necessary data from scene delegate
SCENE UPDATES

Primitive Implementation

1. Lock access to data structures
2. Check which parts changed through dirty bits
3. Pull data from scene delegate and convert it
4. Update owned Cycles object
5. Notify the Cycles scene that it was updated

```cpp
void HdCyclesLight::Sync(
    HdSceneDelegate *sceneDelegate,
    HdRenderParam *renderParam,
    HdDirtyBits *dirtyBits)
{
    const SceneLock lock(renderParam);
    if (*dirtyBits & DirtyBits::DirtyTransform)
    {
        const ccl::Transform tfm = convert_transform(
            sceneDelegate->GetTransform(GetId()));
        light_->set_tfm(tfm);
    }
    ...
    if (light_->is_modified())
    {
        light_->tag_update(lock.scene);
    }
    *
    dirtyBits = DirtyBits::Clean;
}
```
SCENE UPDATES

Materials

- Declare supported material types via `HdRenderDelegate::GetMaterialRenderContexts`
- Hydra only notifies Render Delegate when supported material changed
- Create Cycles shader graph from known nodes

```cpp
const auto diff = graph->create_node<DiffuseBsdfNode>();
...
const auto glossy = graph->create_node<GlossyBsdfNode>();
...
const auto mix = graph->create_node<MixClosureNode>();
mix->set_fac(0.010f);
...
graph->connect(diff->output("BSDF"), mix->input("Closure1"));
graph->connect(glossy->output("BSDF"), mix->input("Closure2"));
```
RENDEERING
Render Pass and Render Buffer Bprims

- **Hydra calls** `HdRenderPass::_Execute` method every frame
- Update Cycles session with requested AOVs and camera settings
- **Cycles writes results with its** `ccl::OutputDriver` class
  - Copies rendered image data into the Bprims associated with the requested AOVs
- Optimization when only color requested and operating in OpenGL:
  - **Cycles can write results to a GPU resource directly via OpenGL interop with its** `ccl::DisplayDriver` class
RENDERING

Render Pass Implementation

1. Try and lock access to data structures
2. Check if AOV bindings changed and if so update them
3. Update camera
4. Check if scene was changed in a way that requires a reset, due to updated AOV bindings, modified viewport, ... and reset session if so
5. Start render thread if it is not running already
6. Optionally present current frame if not using AOVs

```cpp
googletagmanager(“gtm-020223p”)

void HdCyclesRenderPass::_Execute(
    const HdRenderPassStateSharedPtr &renderPassState,
    const TfTokenVector &renderTags)
{
    auto scene = renderParam_->GetCyclesScene();
    auto session = renderParam_->GetCyclesSession();

    if (scene->mutex.try_lock())
    {
        if (renderParam_->GetAovBindings() != renderPassState->GetAovBindings()) {
            renderParam_->SyncAovBindings();
        }

        if (auto camera = renderPassState->GetCamera()) {
            camera->ApplyCameraSettings(scene->camera);
        }

        if (scene->need_reset()) {
            ...
            session->reset(...);
        }

        scene->mutex.unlock();
    }

    session->start();
}

session->draw();
```
**RENDER SETTINGS**

- **Change via** `HdRenderDelegate::SetRenderSetting`
- Hydra keeps track of changes with monotonically increasing render settings "version"
- Apply all render settings in one go right before potential Cycles session reset
HYDRA IN OMNIVERSE
NOW AVAILABLE

- https://developer.blender.org
- Can build against the USD/Hydra version needed in your target application
- CMake variables to build:

  WITH_BLENDER=0
  WITH_CYCLES_HYDRA_RENDER_DELEGATE=1
  USD_INCLUDE_DIRS=<path to your USD build>/include
  USD_LIBRARY_DIR=<path to your USD build>/lib
THANK YOU