3D APPLICATIONS

ParaView

VMD

... 

RENDERING ENGINES

Intel® OSPRay

AMD Radeon™ ProRender

NVIDIA VisRTX

Cycles

...
3D APPLICATIONS

Paraview

blender

VMD

"...for the faint of heart"

... ANARI ...

RENDERING ENGINES

Intel® OSPRay

AMD Radeon™ ProRender

NVIDIA VisRTX

Cycles

...
Working Group Launched March 2020

SDK v0.1.0
March 2022

V1.0 Provisional Specification
November 2021
Working Group Launched March 2020

V1.0 Provisional Specification November 2021

Industry Input and Feedback
- glTF-compatible Physically-based materials
- Improved object interface and error handling
- Directly mapped array parameters
- Revamped runtime feature queries
- Improved volume shading

V1.0 Final Specification August 2023

SDK v0.1.0 March 2022
SDK v0.2.0 July 2022
SDK v0.3.0 Feb 2023
SDK v0.7.0 August 2023

ANARI 1.0 Launch
Open-source SDK includes Conformance Test code

SDK v0.1.0
March 2022

SDK v0.2.0
July 2022

SDK v0.3.0
Feb 2023

SDK v0.7.0
August 2023

Adopters Program
v1.0

Adopters Agreement

Testing Process

Conformance Tests

1Q24

ANARI 1.0
Launch

V1.1 Spec
WIP

V1.0 Provisional
Specification
November 2021

Industry Input
and Feedback

- gITF-compatible Physically-based materials
- Improved object interface and error handling
- Directly mapped array parameters
- Revamped runtime feature queries
- Improved volume shading

Working Group
Launched
March 2020

All specification, SDK and Conformance Test development work done publicly on GitHub
API Design: Balancing Opposing Forces

API Uniformity

Feature Differentiation
API Design: Balancing Opposing Forces

**API Uniformity**
- Handle-based Objects
- Generic Parameters + Arrays
- Object/Array Updates
- Scene Hierarchy
- Concurrency + Parallelism
- API Synchronization Semantics
- Graphics/Compute API Interop
- ...

**Feature Differentiation**
API Design: Balancing Opposing Forces

**API Uniformity**
- Handle-based Objects
- Generic Parameters + Arrays
- Object/Array Updates
- Scene Hierarchy
- Concurrency + Parallelism
- API Synchronization Semantics
- Graphics/Compute API Interop
- ...

**Feature Differentiation**
- Supported API Extensions
- Performance (Frame/Update Latencies)
- Supported Hardware Features
- Image Quality
- Scene Size (Memory overhead, LoD, Out-of-core, Distributed, etc…)
- ...

This work is licensed under a Creative Commons Attribution 4.0 International License
API Design: Balancing Opposing Forces

**API Uniformity**
- Handle-based Objects
- Generic Parameters + Arrays
- Object/Array Updates
- Scene Hierarchy
- Concurrency + Parallelism
- API Synchronization Semantics
- Graphics/Compute API Interop
- ...

**Feature Differentiation**
- Supported API Extensions
- Performance (Frame/Update Latencies)
- Supported Hardware Features
- Image Quality
- Scene Size (Memory overhead, LoD, Out-of-core, Distributed, etc…)
- ...

?
API Design: Balancing Opposing Forces

API Uniformity

Handle-based objects

Generic Parameters + Arrays

Object/Array Updates

Scene Hierarchy

Concurrency + Parallelism

API Synchronization Semantics

Graphics/Compute API Interop

Feature Differentiation

Enabled Extensions

Frame/Update Latencies

Supported Hardware Features

Image Quality

Scene Size (Memory overhead, LoD, Out-of-core, Distributed, etc…)

…

only “what” and “when”

not “how”
ANARI Development Stack

- Scene Graphs
- 3D Applications
- 3D Rendering Engines
- Graphics + Compute APIs
- Hardware

C99 | C++ | Python | ...

This work is licensed under a Creative Commons Attribution 4.0 International License
ANARI Development Stack

- 3D Applications
  - Scene Graphs

- 3D Rendering Engines

- Graphics + Compute APIs

- Hardware
  - GPUs
  - CPUs
  - ...
ANARI Development Stack

3D Applications

Scene Graphs

3D Rendering Engines

Graphics + Compute APIs

OptiX  DirectX  Vulkan  OpenGL  Embree  Metal  ...

Hardware

GPUs  CPUs  ...

This work is licensed under a Creative Commons Attribution 4.0 International License
# ANARI Development Stack

## 3D Applications

| Scene Graphs |

## 3D Rendering Engines

| VisRTX | VisGL | IndeX | OSPRay | Radeon ProRender | Cycles | ... |

## Graphics + Compute APIs

| OptiX | DirectX | Vulkan | OpenGL | Embree | Metal | ... |

## Hardware

| GPUs | CPUs | ... |
ANARI Development Stack

3D Applications

Scene Graphs

3D Rendering Engines

- VisRTX
- VisGL
- IndeX
- OSPRay
- Radeon ProRender
- Cycles
- ...

Graphics + Compute APIs

- OptiX
- DirectX
- Vulkan
- OpenGL
- Embree
- Metal
- ...

Hardware

- GPUs
- CPUs
- ...

C99 | C++ | Python | ...

This work is licensed under a Creative Commons Attribution 4.0 International License
ANARI Development Stack

3D Applications

Scene Graphs

3D Rendering Engines
- VisRTX
- VisGL
- IndeX
- OSPRay
- Radeon ProRender
- Cycles
- ...

Graphics + Compute APIs
- OptiX
- DirectX
- Vulkan
- OpenGL
- Embree
- Metal
- ...

Hardware
- GPUs
- CPUs
- ...

C99 | C++ | Python | ...

This work is licensed under a Creative Commons Attribution 4.0 International License
ANARI Development Stack

3D Applications

Scene Graphs

3D Rendering Engines
- VisRTX
- VisGL
- IndeX
- OSPRay
- Radeon ProRender
- Cycles
- ...

Graphics + Compute APIs
- OptiX
- DirectX
- Vulkan
- OpenGL
- Embree
- Metal
- ...

Hardware
- GPUs
- CPUs
- ...

C99 | C++ | Python | ...

This work is licensed under a Creative Commons Attribution 4.0 International License
ANARI Library Usage

- helide
  - libanari_library_helide.so/.dll
- VisRTX
  - libanari_library_visrtx.so/.dll
- OSPRay
  - libanari_library_ospray.so/.dll
- ...
Transparetly Adding Layers

- **helide**
  - libanari_library_helide.so/.dll

- **VisRTX**
  - libanari_library_visrtx.so/.dll

- **OSPRay**
  - libanari_library_ospray.so/.dll

- **debug device layer**
  - libanari_library_debug.so/.dll

- **libanari**
  - libanari.so/.dll

- **ANARI C API**
  - anari.h

- **App**

- Compile time
- Run time
What’s in the ANARI-SDK?
What’s in the ANARI-SDK?

**Code Gen Tools**
(Extension Queries + Headers)

- Blender Add-on
- CTS
- hdAnari
- Simple Examples
- anariViewer
  - anari_viewer
  - anari_test_scenes
- Python Bindings
- C++ Wrappers (anari_cpp)

**ANARI Front-end Library**
(From Specification)

- Helium
- Helide (Example Implementation)
- Debug Layer
- Sink Device
- Remote Device
What’s in the ANARI-SDK?

<table>
<thead>
<tr>
<th>Blender Add-on</th>
<th>CTS</th>
<th>hdAnari</th>
<th>Simple Examples</th>
<th>anariViewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python Bindings</td>
<td>C++ Wrappers (anari_cpp)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ANARI Front-end Library (From Specification)

<table>
<thead>
<tr>
<th>Helium</th>
<th>Debug Layer</th>
<th>Sink Device</th>
<th>Remote Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helide (Example Implementation)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Code Gen Tools
(Extension Queries + Headers)
ANARI Development Stack

3D Applications

Scene Graphs

3D Rendering Engines

- VisRTX
- VisGL
- IndeX
- OSPRay
- Radeon ProRender
- Cycles
- ...

Graphics + Compute APIs

- OptiX
- DirectX
- Vulkan
- OpenGL
- Embree
- Metal
- ...

Hardware

- GPUs
- CPUs
- ...

This work is licensed under a Creative Commons Attribution 4.0 International License
What’s in the ANARI-SDK?

Blender Add-on

CTS

hdAnari

Simple Examples

anariViewer

anari_viewer

anari_test_scenes

Python Bindings

C++ Wrappers (anari_cpp)

ANARI Front-end Library (From Specification)

Helium

Debug Layer

Sink Device

Remote Device

Helide (Example Implementation)
What’s in the ANARI-SDK?

- Blender Add-on
- CTS
- hdAnari
- Simple Examples
- anariViewer
  - anari_viewer
  - anari_test_scenes
- Python Bindings
- C++ Wrappers (anari_cpp)
- ANARI Front-end Library (From Specification)
  - Helium
  - Debug Layer
  - Sink Device
  - Remote Device
- Helide (Example Implementation)
What’s in the ANARI-SDK?

- **Blender Add-on**
- **CTS**
- **hdAnari**
- **Simple Examples**
- anariViewer
  - anari_viewer
  - anari_test_scenes
- **Python Bindings**
- **C++ Wrappers** (anari_cpp)
- **ANARI Front-end Library** (From Specification)
- **Helium**
- **Debug Layer**
- **Sink Device**
- **Remote Device**
- **Helide** (Example Implementation)
What's in the ANARI-SDK?

- Blender Add-on
- CTS
- hdAnari
- Simple Examples
- anariViewer
- Python Bindings
- C++ Wrappers (anari_cpp)
- ANARI Front-end Library (From Specification)
- Helium
- Debug Layer
- Sink Device
- Remote Device
- Helide (Example Implementation)

Code Gen Tools
(Extension Queries + Headers)
## What’s in the ANARI-SDK?

<table>
<thead>
<tr>
<th>Code Gen Tools</th>
<th>Simple Examples</th>
<th>Python Bindings</th>
<th>C++ Wrappers</th>
<th>ANARI Front-end Library</th>
<th>Blender Add-on</th>
<th>CTS</th>
<th>hdAnari</th>
<th>anariViewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Extension Queries + Headers)</td>
<td></td>
<td></td>
<td></td>
<td>(From Specification)</td>
<td></td>
<td></td>
<td></td>
<td>anariViewer</td>
</tr>
</tbody>
</table>

- **Blender Add-on**
- **CTS**
- **hdAnari**
- **Simple Examples**
- **anariViewer**
  - anari_viewer
  - anari_test_scenes
- **Python Bindings**
- **C++ Wrappers** (anari_cpp)
- **ANARI Front-end Library** (From Specification)
- **Helium**
- **Debug Layer**
- **Sink Device**
- **Remote Device**
- **Helide** (Example Implementation)
What’s in the ANARI-SDK?

- Blender Add-on
- CTS
- hdAnari
- Simple Examples
- anariViewer
- anari_viewer
- anari_test_scenes
- Python Bindings
- C++ Wrappers (anari_cpp)
- ANARI Front-end Library (From Specification)
- Helium
- Debug Layer
- Sink Device
- Remote Device
- Helide (Example Implementation)
What’s in the ANARI-SDK?

Blender Add-on | CTS Template | hdAnari | Simple Examples | anariViewer
Python Bindings | C++ Wrappers (anari_cpp) | Sink Device | Remote Device

ANARI Front-end Library (From Specification)

Helium | Debug Layer | Helide (Example Implementation) | anari_viewer | anari_test_scenes
What’s in the ANARI-SDK?

- Blender Add-on
- CTS
- hdAnari
- Simple Examples
- anariViewer
- Python Bindings
- C++ Wrappers (anari_cpp)
- ANARI Front-end Library (From Specification)
- Helium
- Debug Layer
- Sink Device
- Remote Device
- Helide (Example Implementation)
What’s in the ANARI-SDK?

Blender Add-on

CTS

hdAnari

Simple Examples

anariViewer

anari_viewer

anari_test_scenes

Python Bindings

C++ Wrappers (anari_cpp)

ANARI Front-end Library (From Specification)

Helium

Debug Layer

Sink Device

Remote Device

Helide (Example Implementation)

Code Gen Tools

(Extension Queries + Headers)
API Design: Devices

• ANARI is a C API, with available C++ type safe wrappers
API Design: Devices

• ANARI is a C API, with available C++ type safe wrappers

• Devices are the main object which handles all API calls from the application
  ○ Devices are the instance of the 3D engine that the app is making API calls against
  ○ Devices (usually) come from shared libraries loaded at runtime
API Design: Devices

• ANARI is a C API, with available C++ type safe wrappers
• Devices are the main object which handles all API calls from the application
  ○ Devices are the instance of the 3D engine that the app is making API calls against
  ○ Devices (usually) come from shared libraries loaded at runtime

// Load implementation from libanari_library_visrtx.so/.dll
ANARILibrary lib = anariLoadLibrary("visrtx");

// Create instance of VisRTX from the library
ANARIDevice device = anariNewDevice(lib, "default");
API Design: Objects

• Objects are represented by *opaque handles* and are:
  ○ Reference counted
  ○ Configured with *parameters* (from app to device)
  ○ Introspected with *properties* (from device to app)
API Design: Objects

• Objects are represented by *opaque handles* and are:
  ○ Reference counted
  ○ Configured with *parameters* (from app to device)
  ○ Introspected with *properties* (from device to app)

• Parameter updates are *transactional* using object “commits” to signal object state change
API Design: Objects

• Objects are represented by *opaque handles* and are:
  ○ Reference counted
  ○ Configured with *parameters* (from app to device)
  ○ Introspected with *properties* (from device to app)

• Parameter updates are *transactional* using object “commits” to signal state change

• Parameters are *unidirectional*: values flow into the object, not out
  ○ Applications are responsible for keeping values around they want to “remember” (e.g. to display in a UI)
API Design: Objects

// Create an object that does not need a subtype
ANARIWorld world = anariNewWorld(device);

// Create an object that is subtyped
ANARICamera camera = anariNewCamera(device, "perspective");
API Design: Objects

// Create an object that does not need a subtype
ANARIWorld world = anariNewWorld(device);

// Create an object that is subtyped
ANARICamera camera = anariNewCamera(device, "perspective");

// Parameterize camera using values from the application
anariSetParameter(device, camera, "position", ANARI_FLOAT32_VEC3, &cam_pos);
anariSetParameter(device, camera, "direction", ANARI_FLOAT32_VEC3, &cam_view);
anariSetParameter(device, camera, "up", ANARI_FLOAT32_VEC3, &cam_up);
API Design: Objects

// Create an object that does not need a subtype
ANARIWorld world = anariNewWorld(device);

// Create an object that is subtyped
ANARICamera camera = anariNewCamera(device, "perspective");

// Parameterize camera using values from the application
anariSetParameter(device, camera, "position", ANARI_FLOAT32_VEC3, &cam_pos);
anariSetParameter(device, camera, "direction", ANARI_FLOAT32_VEC3, &cam_view);
anariSetParameter(device, camera, "up", ANARI_FLOAT32_VEC3, &cam_up);

// Commit set parameters to the camera for use in the next rendered frame
anariCommitParameters(device, camera);
API Design: Rendering Frames

// Render one frame
anariRenderFrame(device, frame);

// Wait on the frame to be completed (anariMapFrame() will block if needed)
anariFrameReady(device, frame, ANARI_WAIT);

// Get pointer to the pixels in the color channel
uint32_t width = 0, height = 0;
ANARIDataType type = ANARI_UNKNOWN;
uint32_t *pixels =
    (uint32_t *)anariMapFrame(device, frame, "channel.color", &width, &height, &type);

// Consume the pixels, in this case writing them to a file
writePNG("anari_frame.png", pixels, type, width, height);

// Unmap the pixel buffer and move on to the next frame
anariUnmapFrame(device, frame, "channel.color");

// ...
// Render one frame
anariRenderFrame(device, frame);

// Wait on the frame to be completed (anariMapFrame() will block if needed)
anariFrameReady(device, frame, ANARI_WAIT);

// Get pointer to the pixels in the color channel
uint32_t width = 0, height = 0;
ANARIDataType type = ANARI_UNKNOWN;
uint32_t *pixels =
    (uint32_t *)anariMapFrame(device, frame, "channel.color", &width, &height, &type);

// Consume the pixels, in this case writing them to a file
writePNG("anari_frame.png", pixels, type, width, height);

// Unmap the pixel buffer and move on to the next frame
anariUnmapFrame(device, frame, "channel.color");

// ...
// Render one frame
anariRenderFrame(device, frame);

// Wait on the frame to be completed (anariMapFrame() will block if needed)
anariFrameReady(device, frame, ANARI_WAIT);

// Get pointer to the pixels in the color channel
uint32_t width = 0, height = 0;
ANARIDataType type = ANARI_UNKNOWN;
uint32_t *pixels =
    (uint32_t *)anariMapFrame(device, frame, "channel.color", &width, &height, &type);

// Consume the pixels, in this case writing them to a file
writePNG("anari_frame.png", pixels, type, width, height);

// Unmap the pixel buffer and move on to the next frame
anariUnmapFrame(device, frame, "channel.color");

// ...
API Design: Rendering Frames
Additional Topics

- Details of specific object subtype extensions
  - Geometries, materials, samplers, lights, spatial fields, volumes, cameras…
- Device introspection – detecting extensions + parameter information
- Asynchronous operations: rendering vs. scene updates, thread safety
- Multi-frame and multi-device application architecture
- Array ownership semantics + content updates
- Performance considerations
- Diversity of implementation approaches and design choices
API Design: Balancing Opposing Forces

**API Uniformity**
- Handle-based Objects
- Generic Parameters + Arrays
- Object/Array Updates
- Scene Hierarchy
-Concurrency + Parallelism
-API Synchronization Semantics
- Graphics/Compute API Interop
...

**Feature Differentiation**
- Supported API Extensions
- Performance (Frame/Update Latencies)
- Supported Hardware Features
- Image Quality
- Scene Size (Memory overhead, LoD, Out-of-core, Distributed, etc…)
...

This work is licensed under a Creative Commons Attribution 4.0 International License
© The Khronos® Group Inc. 2024 - Page 58
ANARI is *Portable* (API Uniformity)
ANARI is **Scalable** (Feature Differentiation)

- By not prescribing “how” things are rendered, implementations can scale…
  - Image Quality (lighting, materials, etc.)
  - Available HW (multi-GPU, multi-node)
  - Render Rate
  - Scene Size (geometry, volumes, instances)
  - Animation Update Rate
Elevating Research with ANARI

Data Parallel Path Tracing with Object Hierarchies

INGO WALD, NVIDIA, USA
STEVEN G. PARKER, NVIDIA, USA

PBR landscape
30 K instances, 4.3 B instanced triangles
70 unique meshes, 500 MB image textures
GPU memory usage on most loaded rank: 3.7 GB
frame rate (averaged): 6.2 FPS (1 path/pixel)

Disney Moana island
39 M instances, 41 B instanced triangles
7 M unique meshes, 904 MB baked PTex textures
GPU memory usage on most loaded rank: 25 GB
frame rate (averaged): 7.9 FPS (1 path/pixel)

Fig. 1. Two screenshots from a data-parallel path tracer built using the techniques described in this paper; showing multi-bounce path tracing, textures, alpha textures, area- and environment lighting, etc., on two non-trivial models each distributed across 4 nodes and 8 GPUs. Despite intentionally low-end network infrastructure, at 2560 × 1080 pixels and one path per pixel these two examples run at 6.2 and 7.9 frames per second, respectively (images shown are converged over multiple frames).

We propose a new approach to rendering production-grade content with full path tracing in a data-distributed environment.

Abstract

We propose a novel approach to data-parallel path tracing on single-node multi-GPU hardware that builds on ray tracing, but which aims—to quote—"to keep it simple, stupid." We do this by avoiding any attempts at reducing the memory of trace or forward operations performed, and instead focus on always using all GPU aggregate compute and bandwidth to efficiently trace and ray every GPU. We show that—assuming simplicity—is both feasible and doable and that when run on optimal data centers/loads hardware, the resulting framework not only achieves good performance and scalability, but also comes with significantly fewer limitations, assumptions, and overheads than one GPU is under similar or similar-to-similar environments.

Data Parallel Multi-GPU Path Tracing using Ray Queue Cycling

(author's pre-print, with some added material)

Ingo Wald1
Milan Jarolík2
Stefan Zellmann3
1 NVIDIA
2 eHuman, VŠB – Technical University of Ostrava, Ostrava, Czech Republic
3 University of Cologne

Fig. 1. A high-resolution version of the Disney Moana Island model, with nearly 400 million triangles before instancing, 31 million instancers, and 33 GB of textures, for a total of 2.6 GB of model data excluding acceleration structures. At 2560 × 1080 pixels and 8 paths per pixel, we method runs this at 7.9 frames per second (FPS) on a single node with 8 GPUs in a GPU-PDIM (pixel-distributed interconnect) configuration, built with 8 × 768 GB GPUs, and at 6.8 FPS, respectively, on an RTX 3080 with 8 queues class. GPUs with high frame rates on PCs. An important feature of our method is that it is almost entirely oblivious to how geometry gets partitioned across GPUs, and does not require any spatial or object-space efficient assignment whatsoever. Right: A live color image where an object's color correlates with GPU ID; output shows how little synchronization is necessary on a node.

This work is licensed under a Creative Commons Attribution 4.0 International License
ANARI Software Stack

- 3D Applications
- 3D Rendering Engines
- Graphics + Compute APIs
- Hardware

C99 | C++ | Python | ...
Barney Software Stack

VTK | ParaView | VisIt | Ascent | ...

Barney

CUDA | OptiX | MPI

RTX GPUs | Network HW
Elevating Research with ANARI
Elevating Research with ANARI

Standardized Data-Parallel Rendering Using ANARI

Ingo Wald\textsuperscript{1} NVIDIA
Stefan Zeilmann\textsuperscript{1} University of Cologne
Jefferson Amstutz\textsuperscript{2} NVIDIA
Qi Wu\textsuperscript{1} University of California, Davis
Kevin Griffin\textsuperscript{1} NVIDIA
Milan Jaros\textsuperscript{3} IT4Innovations, VSB – Technical University of Ostrava
Stefan Wesner\textsuperscript{*} University of Cologne

Figure 1: Several examples of large sci-vis data being rendered using the data-parallel ANARI paradigm proposed in this paper. From left to right: a) Roughly one billion color-mapped spheres, rendered using HayStack and BANARI. b) The roughly 500GB airplase data set, with volume path tracing on 128 GPUs, also using HayStack and BANARI. c) An iso-surface rendered during an in-situ Ascent session, while attached to an SSD simulation. d) ParaView performing data-parallel rendering on the airplase data set, using our data-parallel ANARI integration in preserver.

ABSTRACT
We propose and discuss a paradigm that allows for expressing data-parallel rendering with the classically non-parallel ANARI API. We propose this as a new standard for data-parallel sci-vis rendering, describe two different implementations of this paradigm, and use multiple sample integrations into existing apps to show how easy it is to adopt this paradigm, and what can be gained from doing so.

1 INTRODUCTION
Visualization is about more than rendering, but rendering nevertheless plays a large role in many vis tools. Rendering is hard: it was already a hard problem when all such tools relied on a single common API (e.g. OpenGL); today, it is further complicated involved in rendering, such as cameras or data arrays containing geometry, materials, colors, etc. These objects ultimately represent a generic interface to the private implementation of the back-end, where the mechanics of rendering frames is left up to the implementation.

ANARI is not a silver bullet, though. Even with a single agreed-upon API, different implementations can and will still differ in what features exactly they will support (and in which form). Thus, applications still need to be aware of which specific implementation they may be running on—and either adopt a least common denominator approach, or have some application features only available from specific ANARI vendors. Still, this standardization is encouraging as ANARI is already seeing adoption even in VTK and VTK-m, and we hope contributions from related open-source projects (e.g. Bullet) will follow.
Call to Action

- Try out the ANARI-SDK: (https://github.com/KhronosGroup/ANARI-SDK)
  - Make a “hello world” ANARI program with C++ or Python
  - Integrate the API with your research application(s)
  - Try out the various implementations: VisRTX/GL, OSPRay, Visionaray, Barney, Cycles, …
Call to Action

- Try out the ANARI-SDK: (https://github.com/KhronosGroup/ANARI-SDK)
  - Make a “hello world” ANARI program with C++ or Python
  - Integrate the API with your research application(s)
  - Try out the various implementations: VisRTX/GL, OSPRay, Visionaray, Barney, Cycles, …

- Explore implementing ANARI using your renderer/engine
  - Reference the ‘helide’ device inside the SDK or other FOSS implementations
Call to Action

- **Try out the ANARI-SDK:** ([https://github.com/KhronosGroup/ANARI-SDK](https://github.com/KhronosGroup/ANARI-SDK))
  - Make a “hello world” ANARI program with C++ or Python
  - Integrate the API with your research application(s)
  - Try out the various implementations: VisRTX/GL, OSPRay, Visionaray, Barney, Cycles, ...

- **Explore implementing ANARI using your renderer/engine**
  - Reference the ‘helide’ device inside the SDK or other FOSS implementations

- **Get involved with the standard**
  - Join the ANARI Working Group (for Khronos members)
    - Weekly WG meetings are Wednesdays @ 10am Pacific
  - Join the ANARI Advisory Panel (for Khronos non-members)
    - Mailing list + on-demand online discussions
  - Open issues on GitHub, both the SDK + specification
Call to Action

● Try out the ANARI-SDK: (https://github.com/KhronosGroup/ANARI-SDK)
  ○ Make a “hello world” ANARI program with C++ or Python
  ○ Integrate the API with your research application(s)
  ○ Try out the various implementations: VisRTX/GL, OSPRay, Visionaray, Barney, Cycles, ...

● Explore implementing ANARI using your renderer/engine
  ○ Reference the ‘helide’ device inside the SDK or other FOSS implementations

● Get involved with the standard
  ○ Join the ANARI Working Group (for Khronos members)
    ■ Weekly WG meetings are Wednesdays @ 10am Pacific
  ○ Join the ANARI Advisory Panel (for Khronos non-members)
    ■ Mailing list + on-demand online discussions
  ○ Open issues on GitHub, both the SDK + specification