Portable and Scalable Rendering with ANARI™

July 26-28, 2024
Denver, Colorado
3D APPLICATIONS

ParaView

blender

VMD

Visual Molecular Dynamics

...

RENDERING ENGINES

Intel® OSPRay

AMD Radeon™ ProRender

NVIDIA VisRTX

Cycles Open Source Production Rendering

...

This work is licensed under a Creative Commons Attribution 4.0 International License
3D APPLICATIONS

ParaView

blender®

VMD Visual Molecular Dynamics

... 

RENDERING ENGINES

Intel® OSPRay

AMD Radeon™ ProRender

NVIDIA VisRTX

Cycles Open Source Production Rendering

...
3D APPLICATIONS

ParaView

blender

VMD

Visual Molecular Dynamics

...for the faint of heart

RENDERING ENGINES

Intel® OSPRay

AMD Radeon™ ProRender

NVIDIA VisRTX

Cycles

Open Source Production Rendering

...
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

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

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

SDK v0.7.0 August 2023

V1.0 Final Specification August 2023

ANARI 1.0 Launch
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

ANARI 1.0 Launch

Adopters Program v1.0
Adopters Agreement
Testing Process
Conformance Tests
1Q24
Open-source SDK includes Conformance Test code

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…)
  - ...

...
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**
- Extensions
- Supported API Extensions
- Scene Size (Memory overhead, LoD, Out-of-core, Distributed, etc...)
- Supported Hardware Features
- Image Quality
- Supported Hardware Features
- Performance (Frame/Update Latencies)
- only "**what**" and "**when**"
- not "**how**"
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 | ...

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

Scene Graphs

3D Applications

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 | ...
ANARI Library Usage

libanari

libanari_library_helide.so/.dll

helide

libanari_library_visrtx.so/.dll

VisRTX

libanari_library_ospray.so/.dll

OSPRay

libanari.so/.dll

...
Transparency Adding Layers

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.so/.dll

ANARI C API
anari.h

App

Compile time
Run time
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

```c
// 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");

// ...
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");

// ...
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…)
- ...

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

(Hardware: 4 worker nodes w/ 2× RTX 8000, low-end head node, 10-Gigabit Ethernet, screen size 2560 × 1080)

PRT landscape
Disney Hawaiian island

30 K instances, 4.3 B instanced triangles
39 M instances, 41 B instanced triangles
770 unique meshes, 500 MB image textures
3 M unique meshes, 804 MB baked Ptex textures
GPU memory usage on most loaded rank: 3.7 GB
GPU memory usage on most loaded rank: 25 GB
frame rate (averaged): 6.2 FPS (1 path/pixel)
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

HPG 2022
Elevating Research with ANARI

Data Parallel Path Tracing with Object Hierarchies

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

(Hardware: 4 worker nodes w/ 2x RTX 8000, low-end head node, 10-Gigabit Ethernet, screen size 2560 x 1080)

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 x 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 novel approach to rendering production-quality content with full path tracing on a data-distributed application. This approach allows for efficient use of modern GPUs in rendering complex scenes. We have implemented this approach on a data-parallel path tracer that we have developed within the ANARI project, which is designed for high-performance, real-time rendering applications.

**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 address the limitations of current methods. Our work focuses on using a data-parallel approach to efficiently load and render complex scenes. We have developed a data-parallel path tracer that can be used to render high-quality images in real-time. This approach allows for efficient use of modern GPUs in rendering complex scenes.

**Keywords**

Data parallelism, multi-GPU rendering, ray tracing, path tracing, object hierarchies.

Ingo Wald1  
Milan Jablonski2  
Stefan Zellmann2

1 NVIDIA  
2 ITT Forschungsgesellschaft mbH, Germany  
3 Computer Graphics Group, University of Cologne

Fig. 1. A high-resolution version of the Disney Moana island scene, with nearly 400 million triangles before instancing, 30 million instances, and 30 GB of textures. The scene is rendered on a single node with 8 GPUs, with a total of 32 GPUs used across 32 nodes. The scene is rendered at 4K resolution using a NVIDIA RTX 8000 and a NVIDIA RTX 4090.

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

VTK  ParaView  VisIt  Ascent  ...

C99  C++  Python  ...

Barney

CUDA  OptiX  MPI

RTX GPUs  Network HW

This work is licensed under a Creative Commons Attribution 4.0 International License
Elevating Research with ANARI

1B Spheres Wing Sim  
DNS Isosurface  
DNS Volume  
NekRs (Ascent)  
S3D (Ascent)

Elevating Research with ANARI

1B Spheres Wing Sim  
DNS Isosurface  
DNS Volume  
NekRs (Ascent)  
S3D (Ascent)

Elevating Research with ANARI

1B Spheres Wing Sim  
DNS Isosurface  
DNS Volume  
NekRs (Ascent)  
S3D (Ascent)

Elevating Research with ANARI

1B Spheres Wing Sim  
DNS Isosurface  
DNS Volume  
NekRs (Ascent)  
S3D (Ascent)
Elevating Research with ANARI

Standardized Data-Parallel Rendering Using ANARI

Ingo Wald\textsuperscript{*} NVIDIA
Stefan Zellmann\textsuperscript{1} University of Cologne
Jefferson Amstutz\textsuperscript{2} NVIDIA
Qi Wu\textsuperscript{3} University of California, Davis
Kevin Griffin\textsuperscript{4} NVIDIA
Milan Jaros\textsuperscript{5} IT4Innovations, VSB – Technical University of Ostrava
Stefan Wesner\textsuperscript{**} University of Cologne

Figure 1: Several examples of large sci-viz 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 eps 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 S3D simulation. d) Paraview performing data-parallel rendering on the airplane data set, using our data-parallel ANARI integration in preserver.

\textbf{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-viz 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 \textbf{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 the application communities of both have been active in the ANARI effort. 

© The Khronos\textsuperscript{®} Group Inc. 2024 - Page 55
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