# ANARI: ANAlytic Rendering Interface

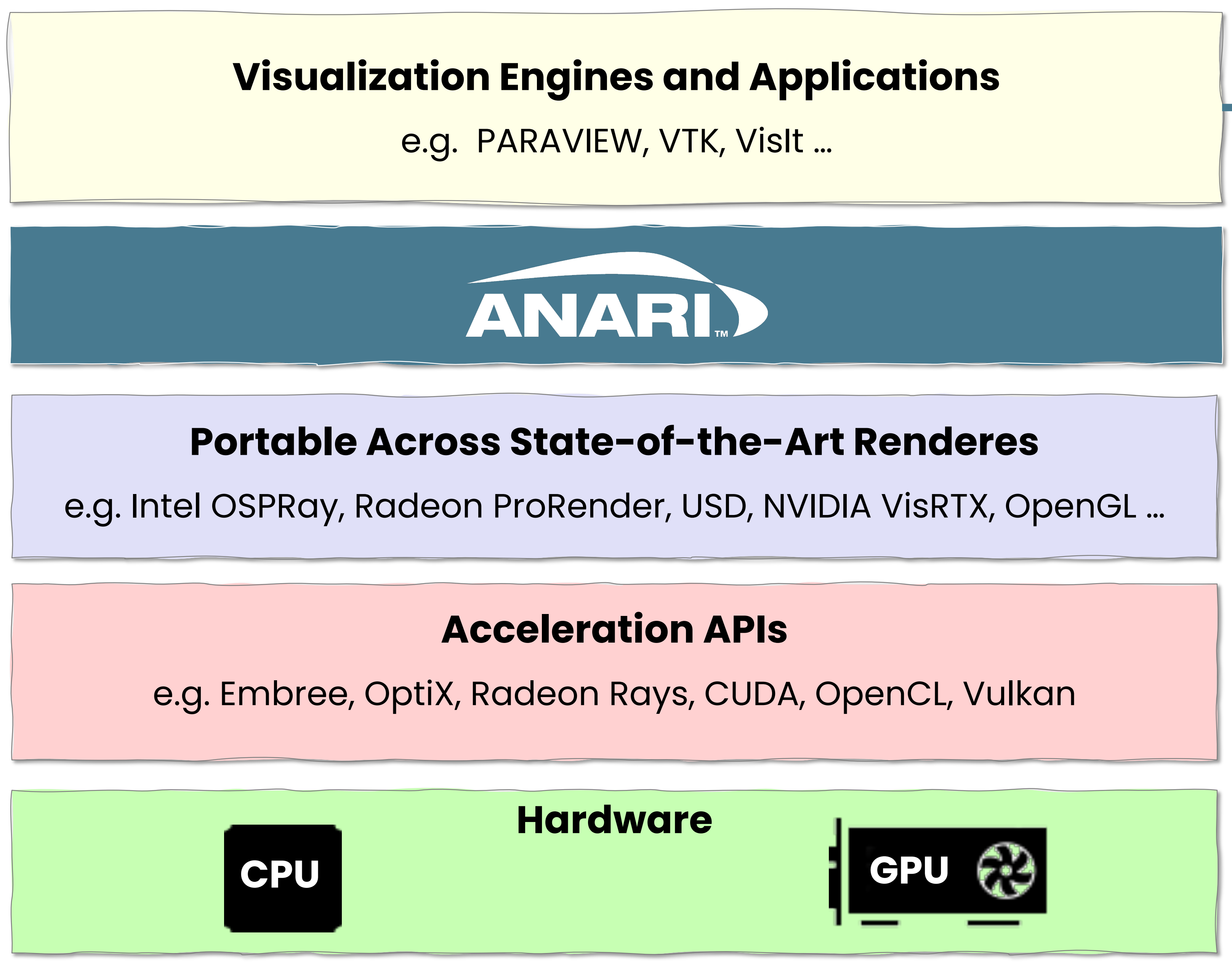
Portable, performant access to advanced 3D rendering engines

Kevin Griffin[1], Jeff Amstutz[1], Dave DeMarle[2], Johannes Gunther[2], Jakob Progsch[1], Bill Sherman[3], John Stone[4], Will Usher[2], Kees van Kooten[1]
[1]NVIDIA, [2]Intel Corporation, [3]NIST, [4]University of Illinois at Urbana-Champaign

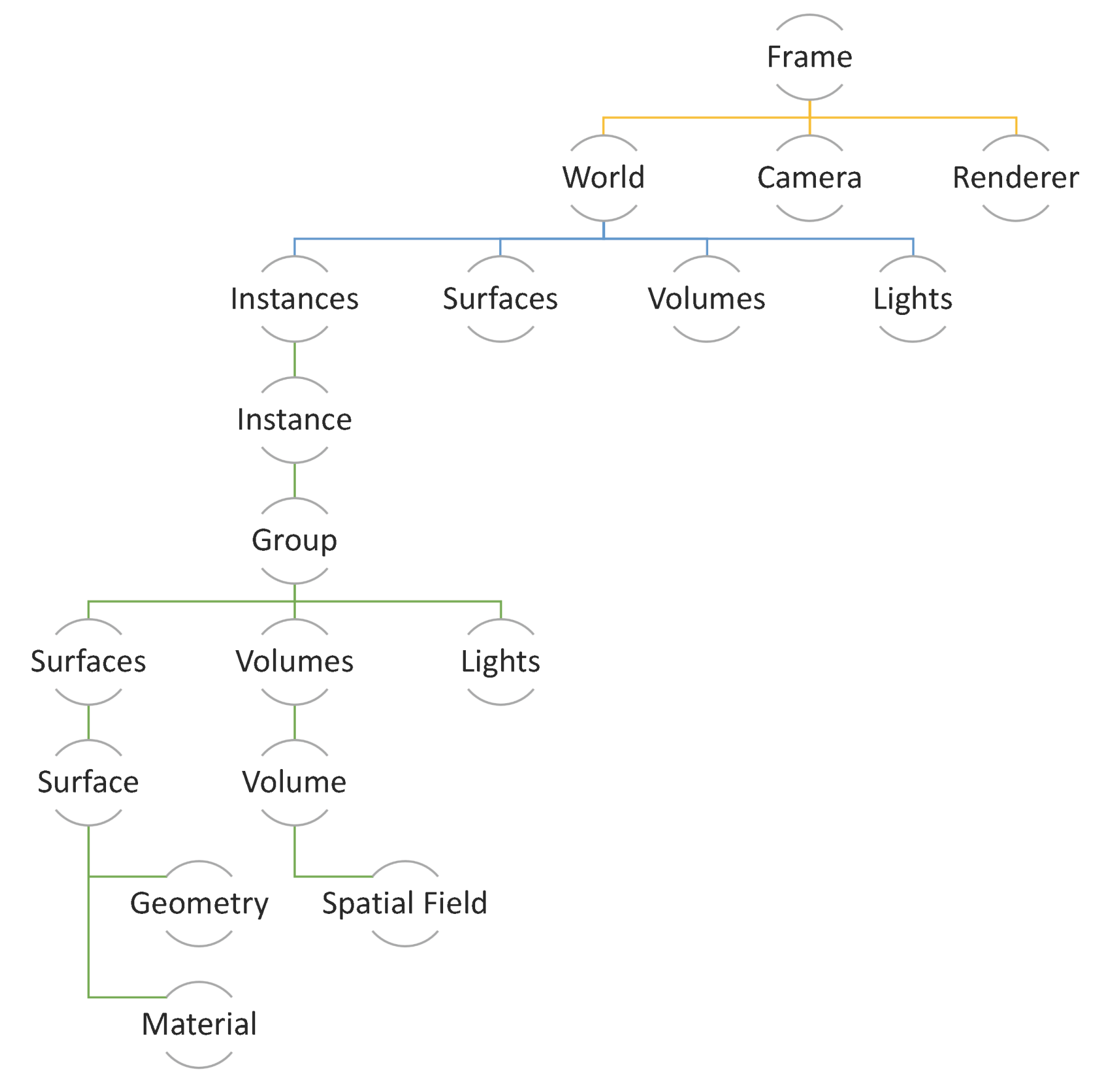
## Motivation

The ANARI (Analytic Rendering Interface) API enables users to build the description of a scene to generate imagery, rather than specifying the details of the rendering process, providing simplified visualization application development and cross-vendor portability to diverse rendering engines, including those using state-of-the-art ray tracing.

## Methodology


Visualization Engines and Applications
e.g. PARAVIEW, VTK, VisIt ...
ANARI
Portable Across State-of-the-Art Renderes
e.g. Intel OSPRay, Radeon ProRender, USD, NVIDIA VisRTX, OpenGL ...
Acceleration APIs
e.g. Embree, OptiX, Radeon Rays, CUDA, OpenCL, Vulkan
Hardware
CPU
GPU

## Scene Description in Memory



**Frame**: The root object of the scene.

**Camera**: Configures the projection of the rendering used to view the **World**.

**Renderer**: Holds parameters relating to the rendering algorithm.

**World**: Holds arrays of the drawable objects of the scene either directly or via an array of **Instances** each containing a **Group**.

## Backend Loading and Initialisation

```
const char *libraryName = GetDeviceType();
m_library = anariLoadLibrary(libraryName, avtANARIDevice::StatusCallback);
m_device = anariNewDevice(m_library, "default");
if(m_device)
{
    anari::setParameter(m_device, m_device, "debug", true);
    anariCommit(m_device, m_device);

    debug5 << "[ANARI::OptiX] Done loading module and creating device" << std::endl;
    if(m_dataType == DataType::VOLUME)
        ExecuteVolume();
    else
        ExecuteSurface();
}
else
{
    std::string msg("[ANARI::OptiX] Could not load device for OptiX module.");
    EXCEPTION1(VisItException, msg);
}
```

## API Calls

```
ANARIWorld avtANARIVolumeDevice::CreateWorld(ANARIDevice *device, ...)
{
   // Instances can apply transforms to groups for placement in the World
   ANARIInstance instance = anariNewInstance(*device);

   if(instance != nullptr)
   {
      ...
      // Create World
      ANARIWorld world = anariNewWorld(*device);

      // Instance
      ANARIArray1D array1D = anariNewArray1D(*device, &instance, nullptr, ANARI_INSTANCE, 1);

      anari::setAndReleaseParameter(*device, world, "instance", array1D);
      anariRelease(*device, instance);
      ...
      return world;
   }
}
```

## Scene Setup and Rendering

```
ANARIFrame frame = anariNewFrame(m_device);

int screenSize[2] = {width, height};
anari::setParameter(m_device, m_frame, "size", ANARI_UINT32_VEC2, &screenSize[0]);
ANARIDataType format = ANARI_UFIXED8_RGBA_SRGB;
anari::setParameter(m_device, m_frame, "color", ANARI_DATA_TYPE, &format);
ANARIDataType depthFormat = ANARI_FLOAT32;
anari::setParameter(m_device, m_frame, "depth", ANARI_DATA_TYPE, &depthFormat);

bool t = true;
anari::setParameter(m_device, m_frame, "channelColor", ANARI_BOOL, &t);
anari::setParameter(m_device, m_frame, "channelDepth", ANARI_BOOL, &t);

anari::setParameter(m_device, m_frame, "camera", m_camera);
anari::setParameter(m_device, m_frame, "world", m_world);
anari::setParameter(m_device, m_frame, "renderer", m_renderer);
anariCommit(m_device, m_frame);

// Render frame
anariRenderFrame(m_device, m_frame);
anariFrameReady(m_device, m_frame, ANARI_WAIT);

// Map framebuffer
const void *renderedFrameBuffer = anari::map(m_device, m_frame, "color");
const void *depthBuffer = anari::map(m_device, m_frame, "depth");
```

## ANARI Rendering Results I

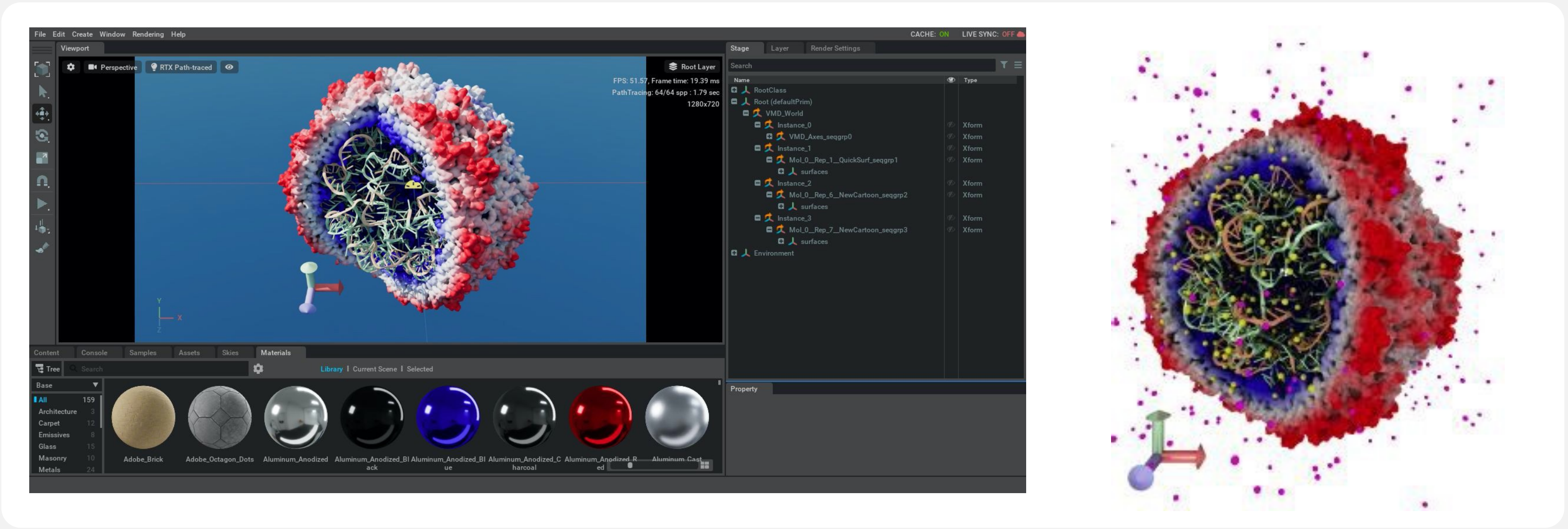


**Figure 1**: VMD visualization, using the Intel OSPRay ANARI device, of satellite tobacco mosaic virus capsid and its interior RNA, with surrounding solvent ions.

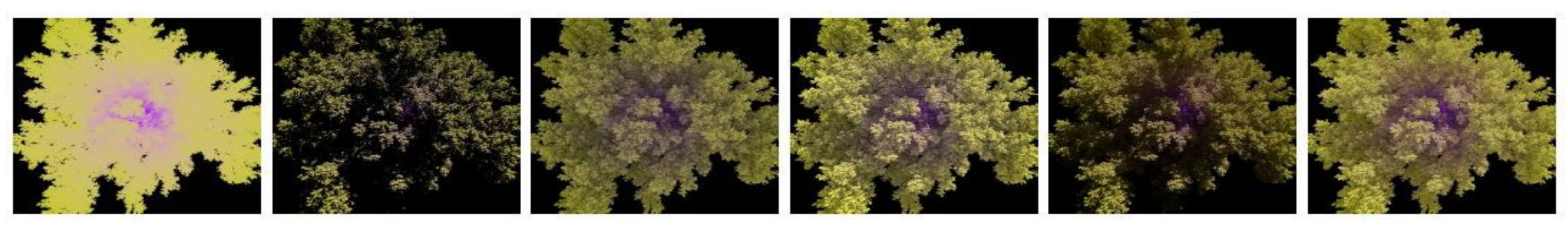


**Figure 5**: Comparison of lighting techniques for a complex and crowded visualization of the results of a diffusion-limited aggregation simulation, rendered using ANARI and the NVIDIA VisRTX back-end device. The lighting techniques, from left, are: raycasting of surface color, directional lighting and shadows, ambient occlusion lighting, directional lighting with ambient occlusion, directional lighting with path traced indirect lighting, and directional lighting combined with ambient occlusion and path traced indirect lighting.

## ANARI Rendering Results II

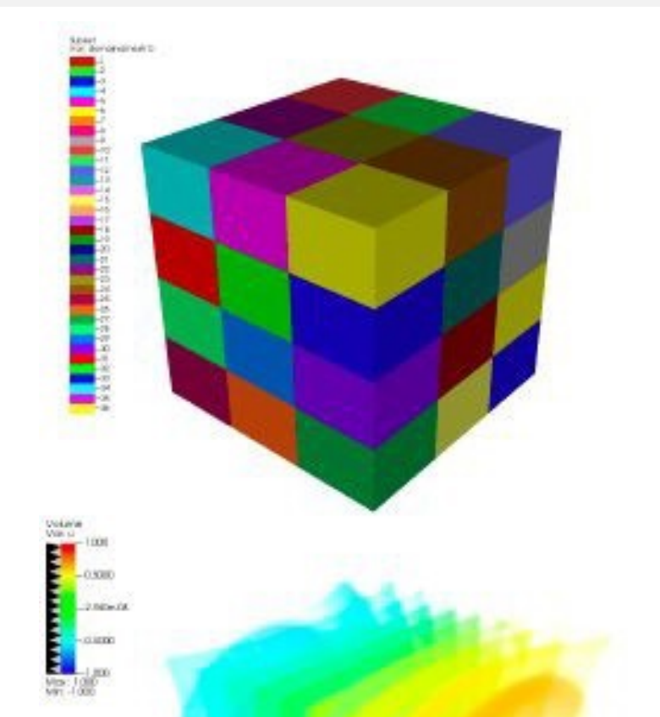


**Figure 2**: VisIt parallel volume rendering using NVIDIA VisRTX back-end.

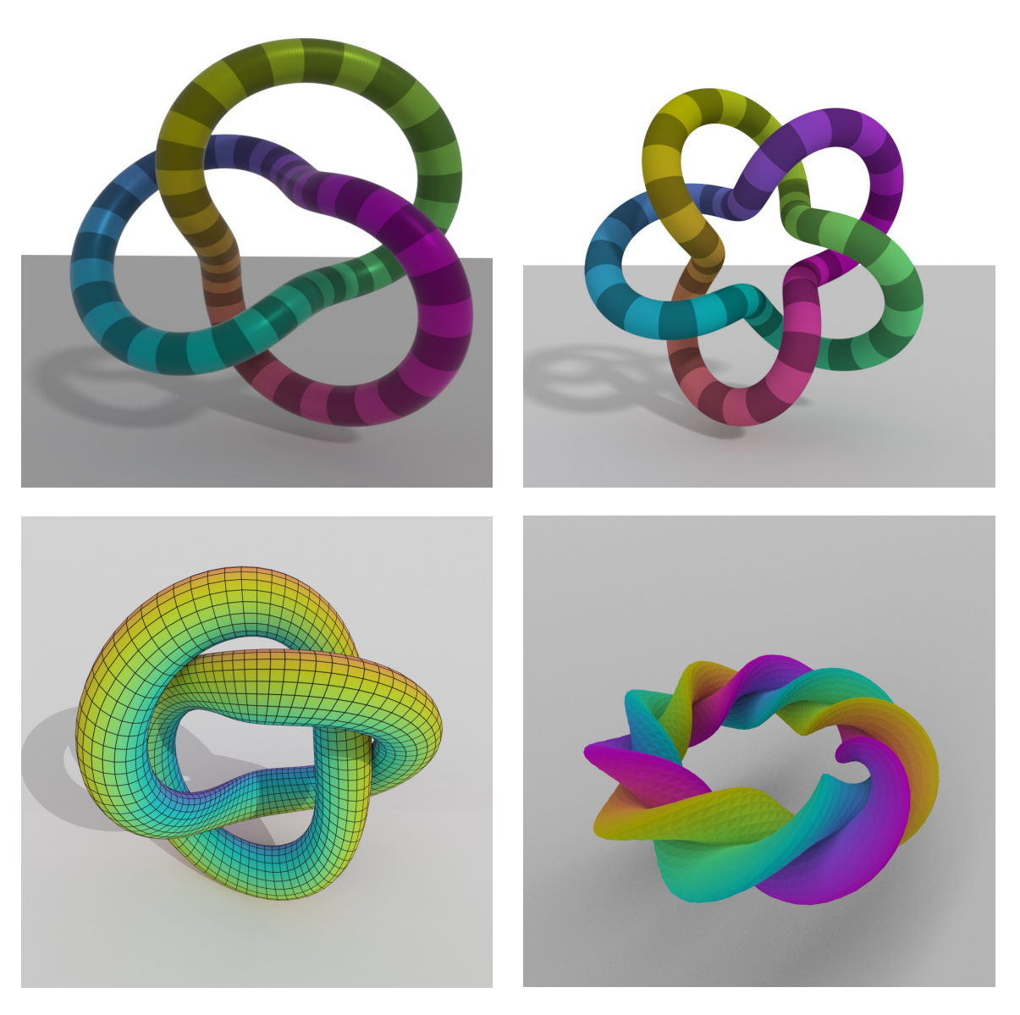


**Figure 3**: ANARI renderings of knots (top, Intel OSPRay) and parametric surfaces (bottom left, NVIDIA VisRTX and bottom right, AMD Radeon ProRender).

**Figure 4**: NVIDIA VisRTX Path traced rendering of the San Miguel scene © Guillermo M. Leal Llaguno (https://casual-effects.com/data).