Vulkan编程原理及通用并行计算

何小伟
中国科学院软件研究所

2023.6.4

大纲

- Vulkan简介及通用并行计算概述
- 如何利用Vulkan编写一个并行计算程序
- 如何降低Vulkan编程复杂度
- 如何实现Vulkan后端与PeriDyno框架的衔接
- Vulkan编程其他注意事项
- 测试案例展示

Vulkan简介及通用并行计算概述

- Vulkan简介
  - 由 Khronos Group 在在 GDC 2015 首次提出，2016首次发布
  - 被誉为"next generation OpenGL initiative", or "OpenGL next"
  - 以“编程麻烦”著称

Vulkan简介及通用并行计算概述

- Features:
  - Unified API: a single API for both desktop and mobile graphics devices
  - Cross platform: Android, Linux, BSD Unix, QNX, Haiku,[24] Nintendo Switch, Raspberry Pi, Stadia, Fuchsia, Tizen, and Windows 7, 8, 10, and 11.
  - Multi-threading friendly design: Direct3D 11 and OpenGL 4 were initially designed for use with single-core CPUs
  - Lower overhead
  - More direct control over the GPU
  - Lower CPU usage
  - …

Vulkan简介及通用并行计算概述

- The history of GPU API (Graphics/Compute)

1992 OpenGL
- Khronos Group
- Fixed Functions Pipeline
- Written in C

1995 DirectX
- Windows 95
- Stop supporting OpenGL

2002 DirectX
- Direct3D 9
- High-Level Shading Language(**HLSL**)

2003 OpenGL ES
- A subset of OpenGL

2004 OpenGL
- OpenGL 2.0
- OpenGL Shading Language(**GLSL**)

2007 CUDA
- Nvidia.
- Compute only
- C/C++

2009 OpenCL
- Apple Inc.
- Khronos Group
- Compute only
- C/C++

2014 Metal
- Apple Inc.
- Graphics&Compute
- C++/Objective C

2016 Vulkan
- AMD
- Khronos Group
- Graphics&Compute
- Low-level
- Written in C

Vulkan简介及通用并行计算概述

- 典型通用并行计算编程语言比较

	CUDA	OpenCL	Vulkan
支持混合编译（single source）	支持	目前不支持	不支持
支持状态输出，比如printf	支持	支持	支持
跨平台支持	差	一般	好
支持面向对象的封装	支持	支持	不支持
支持模板	支持	支持	不支持

如何利用Vulkan编写一个并行计算程序

- 已知数组A和B，求和并保存到数组C

CUDA

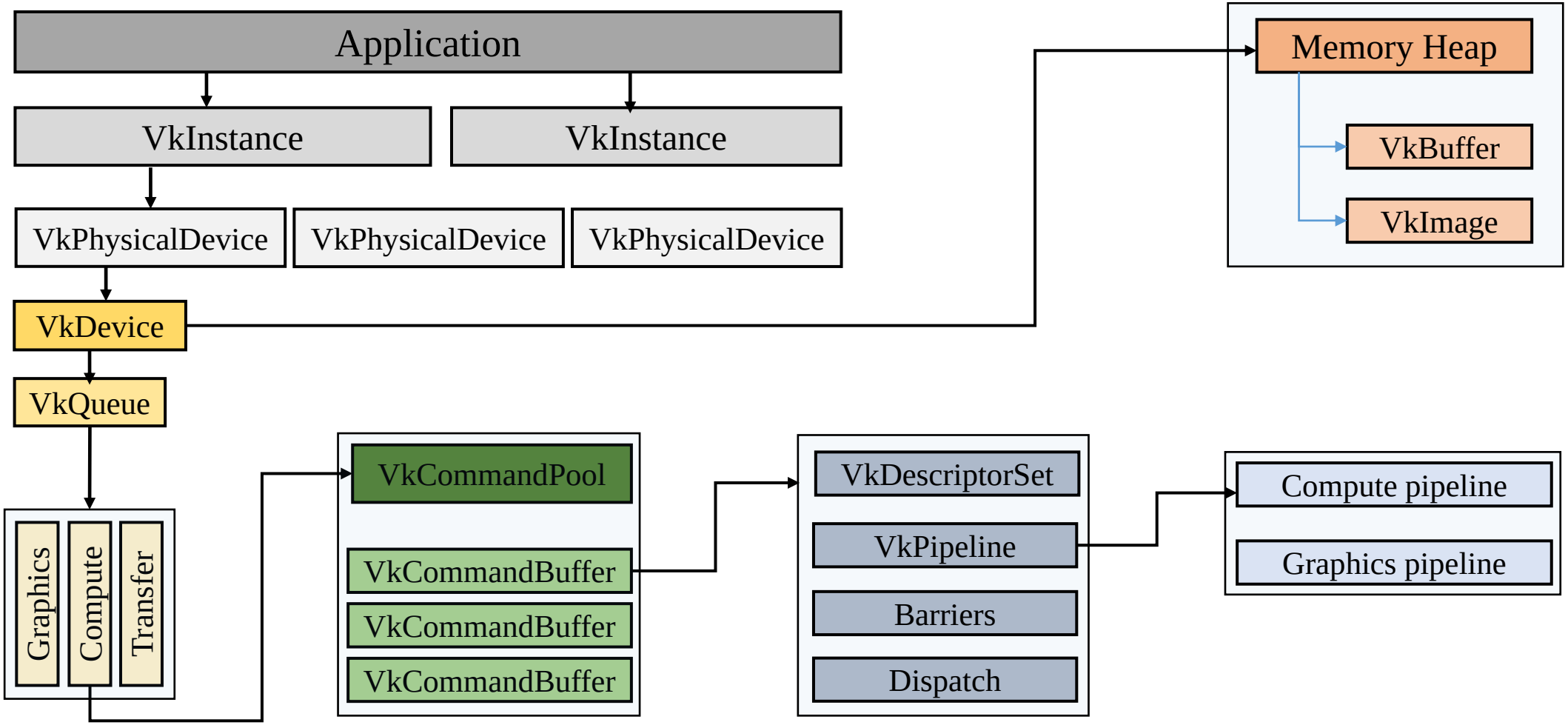
```
// CUDA Kernel
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    C[i] = A[i] + B[i];
}
```

Vulkan

一言难尽

如何利用Vulkan编写一个并行计算程序

- Vulkan对象及其关系图



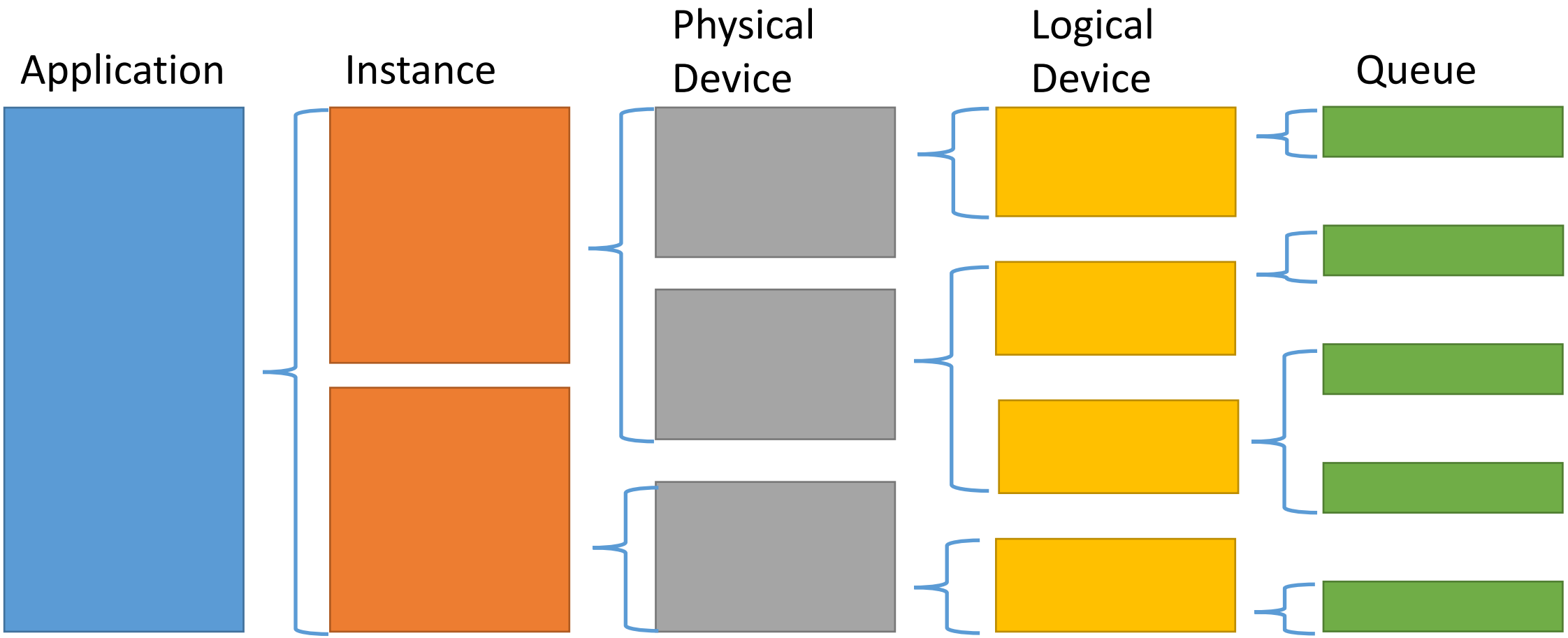
如何利用Vulkan编写一个并行计算程序

- Overview on developing a Vulkan application
  - Create an **instance**, pick a **physical device**, create a **logical device**, and pick a **queue**.
  - Allocate **memory and buffers** for the application.
  - Write the code for the **compute shader** on GLSL and compile it to **SPIR-V**.
  - Create **descriptor sets** and **compute pipelines**
  - Create **command buffer**, record commands to it.
  - **Dispatch** the command buffer.
  - Evaluate the results.

Bodurri KLAJDI. Comparative performance analysis of Vulkan and CUDA programming model implementations for GPUs, 2019.

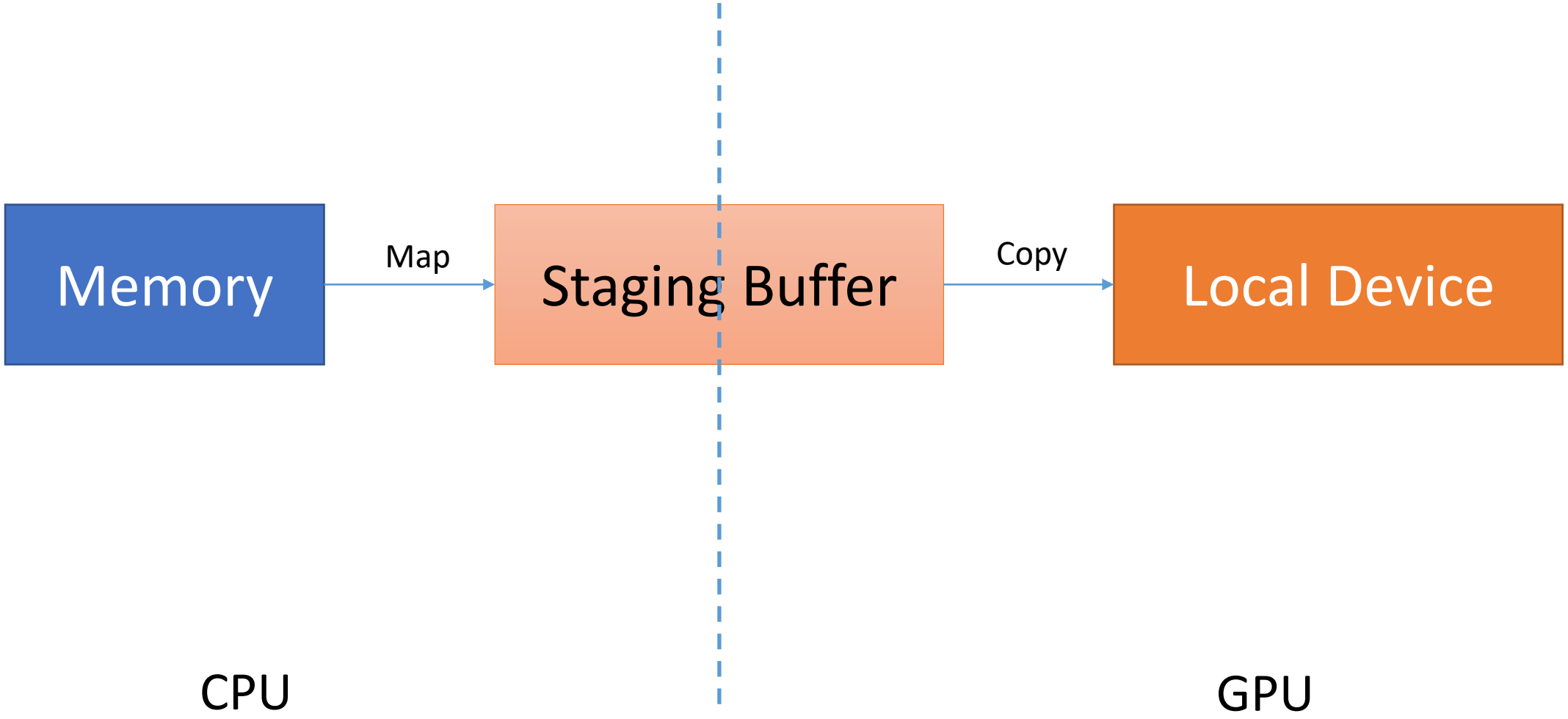
如何利用Vulkan编写一个并行计算程序

- Create an **instance**, pick a **physical device**, create a **logical device**, and pick a **queue**



如何利用Vulkan编写一个并行计算程序

- Allocate **memory and buffers** for the application.



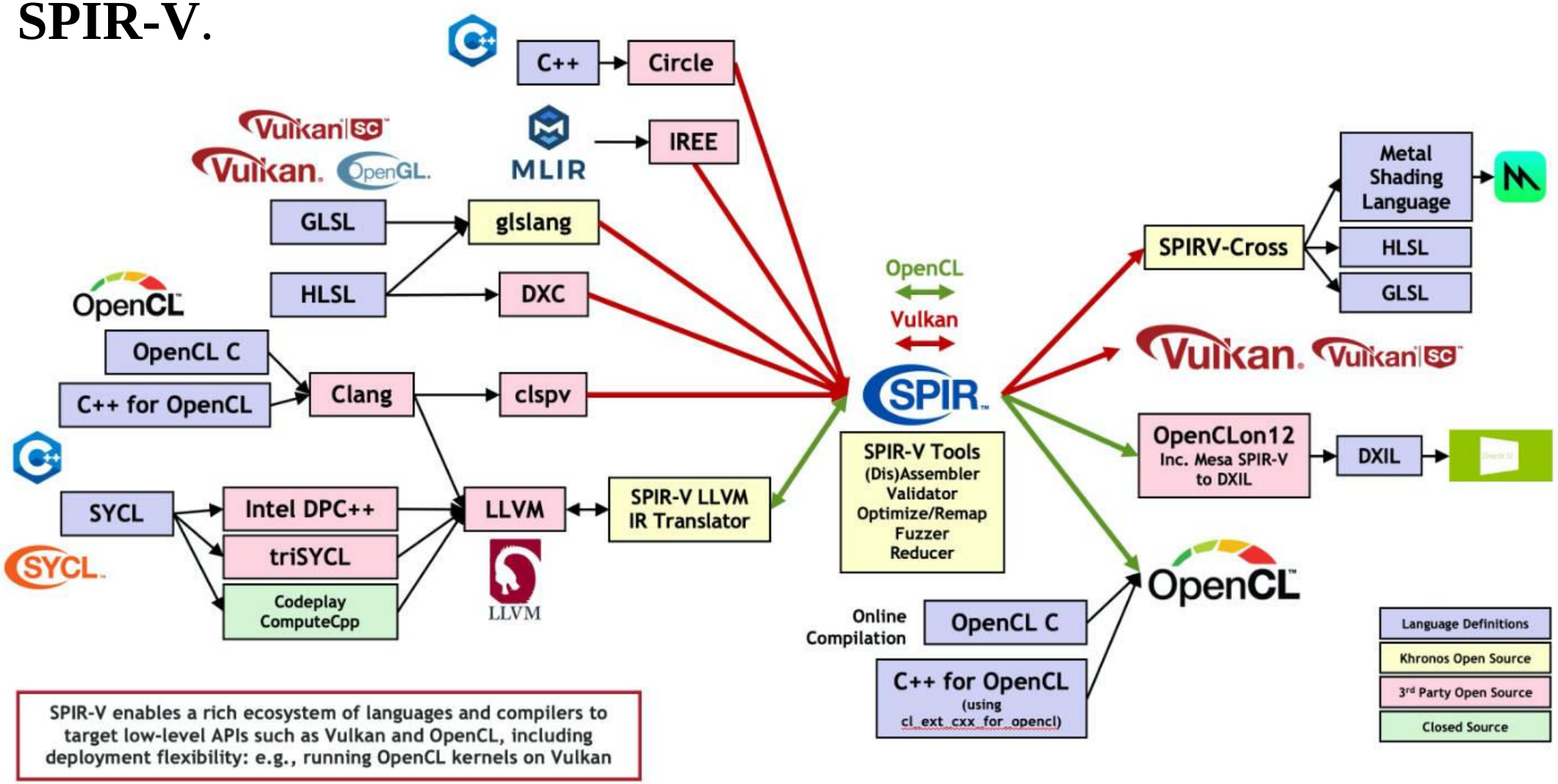
如何利用Vulkan编写一个并行计算程序

- Write the code for the **compute shader** on GLSL and compile it to **SPIR-V**.

```
#version 430

layout(std430, binding = 0) buffer VecA {
    float A[ ];
};

layout(std430, binding = 1) buffer VecB {
    float B[ ];
};

layout(std430, binding = 2) buffer VecC {
    float C[ ];
};

layout (push_constant) uniform PushConsts {
    uint count;
} pushConsts;

layout (local_size_x = 128) in;

void main()
{
    uvec3 id = gl_GlobalInvocationID;

    uint index = id.x;
    if (index >= pushConsts.count)
        return;

    C[index] = A[index] + B[index];
}
```

如何利用Vulkan编写一个并行计算程序

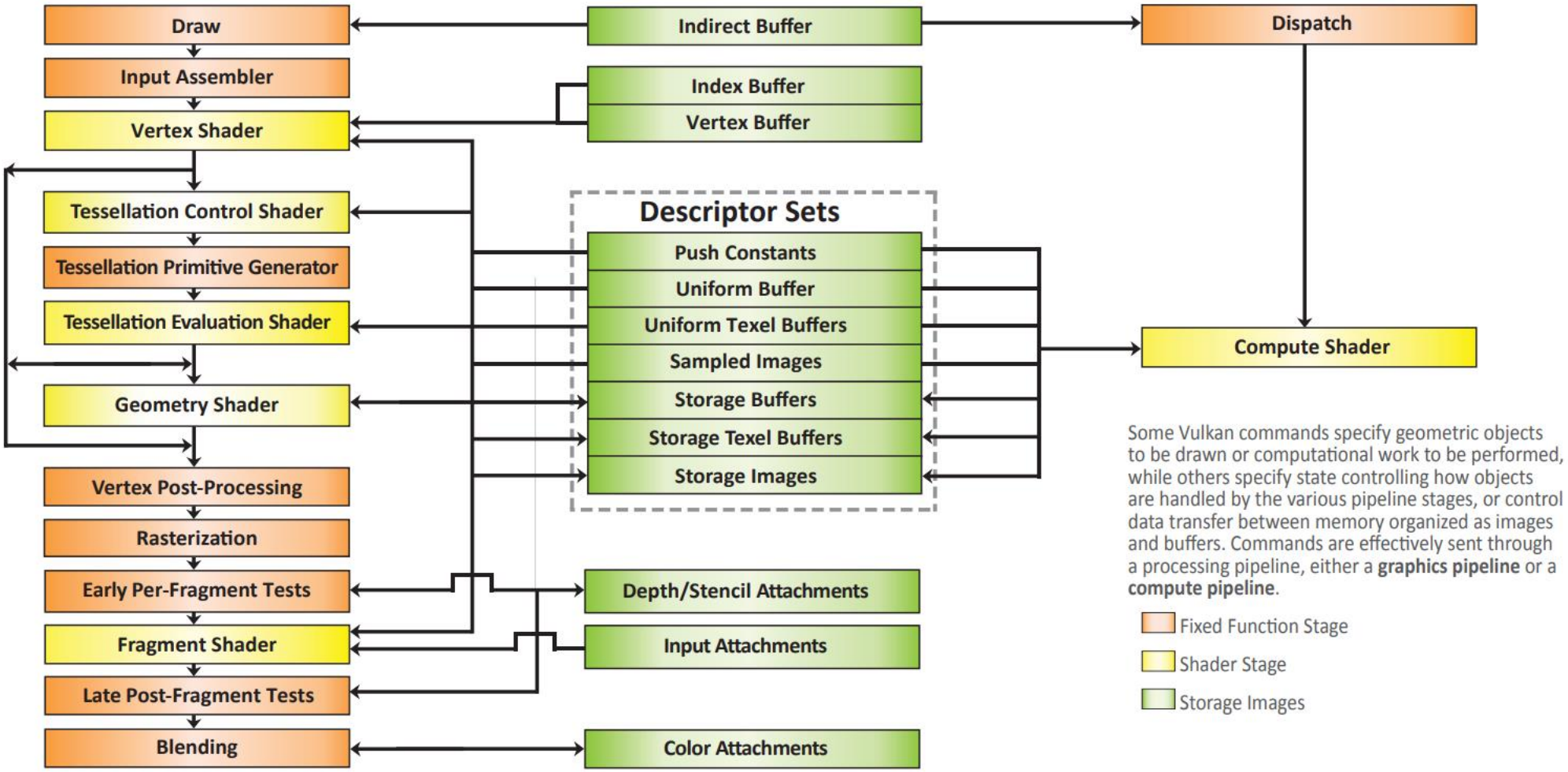
- Write the code for the **compute shader** on GLSL and compile it to **SPIR-V**.



https://www.khronos.org/spir/

如何利用Vulkan编写一个并行计算程序

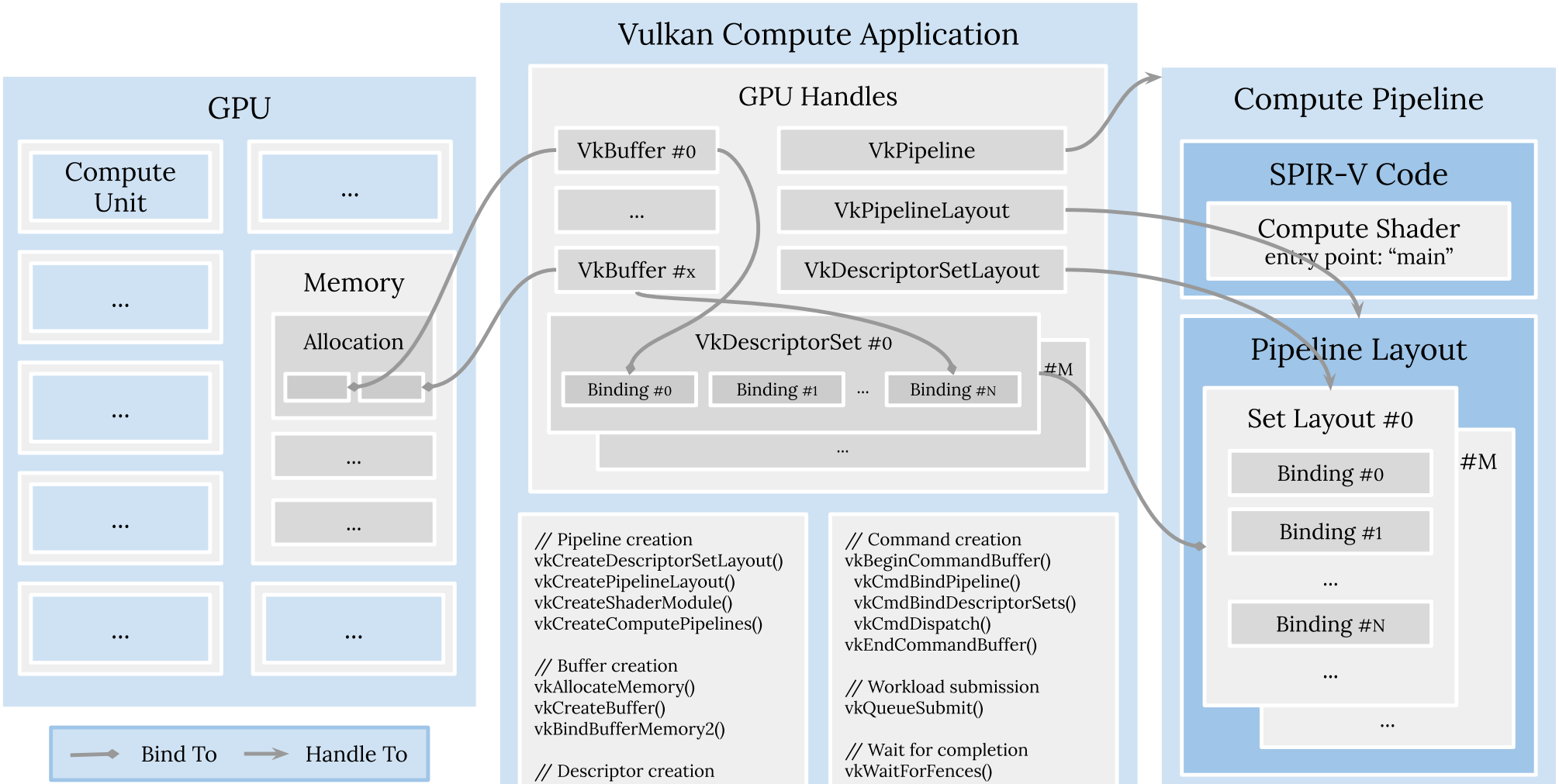
- Create **descriptor sets** and **compute pipelines**



https://www.khronos.org/files/vulkan11-reference-guide.pdf

如何利用Vulkan编写一个并行计算程序

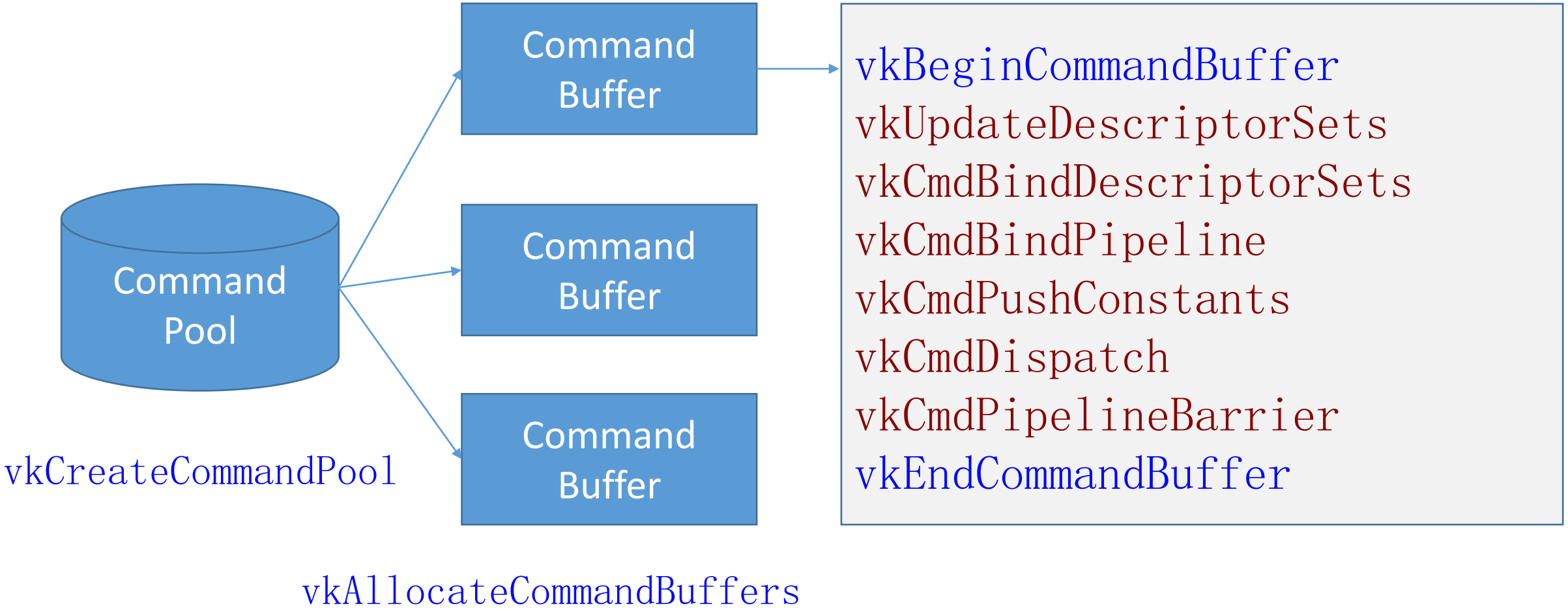
- Create **descriptor sets** and **compute pipelines**



https://www.lei.chat/posts/what-is-vulkan-compute/

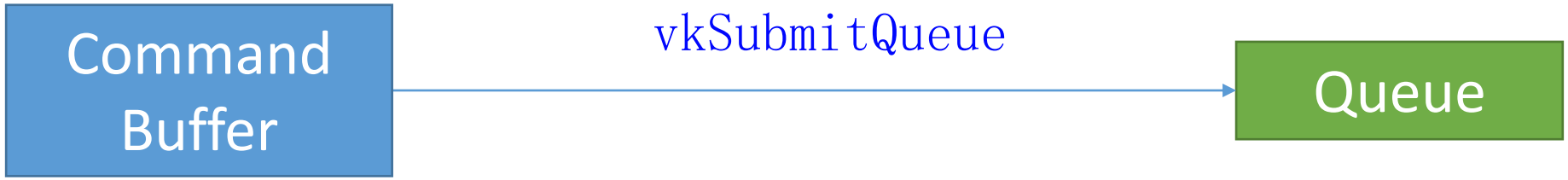
如何利用Vulkan编写一个并行计算程序

- Create **command buffer**, record commands to it.



如何利用Vulkan编写一个并行计算程序

- **Dispatch** the command buffer.



如何利用Vulkan编写一个并行计算程序

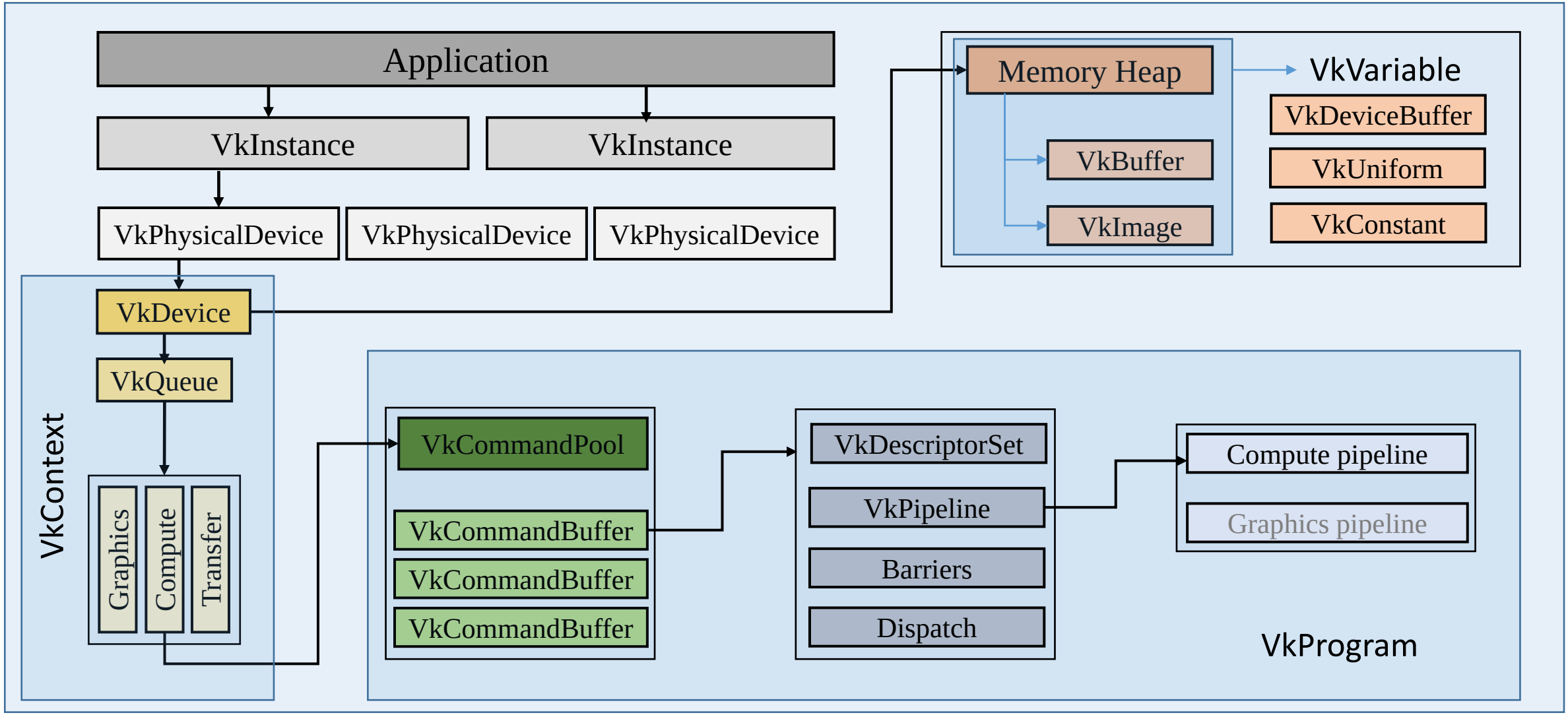
- 完整案例参见
  - examples/Vulkan/Tutorials/App_VulkanNative

如何降低Vulkan编程复杂度

- 封装Vulkan底层细节
  - Vulkan对象封装
  - Kernel函数封装
  - 数据对象封装

如何降低Vulkan编程复杂度

- Catalyzer逻辑架构图



如何降低Vulkan编程复杂度

- Kernel函数定义：__global__ —> VkProgram

CUDA

```
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    C[i] = A[i] + B[i];
}
```

```
VecAdd<<<dim, block>>>(A, B, C);
```

Vulkan

```
auto kernel = std::make_shared<VkProgram>(
    BUFFER(float),          //Array A
    BUFFER(float),          //Array B
    BUFFER(float),          //Array C
    CONSTANT(uint));
kernel->load(getAssetPath() +
    "shaders/glsl/tutorials/VecAdd.comp.spv");
```

```
VkConstant<uint> N(num);
kernel->flush(
    vkDispatchSize(num, 128),
    dA.handle(),
    dB.handle(),
    dC.handle(),
    &N);
```

如何降低Vulkan编程复杂度

- VkProgram形参列表

• VkConstant	• CONSTANT
• VkUniform	• UNIFORM
• VkDeviceArray	• BUFFER
• VkDeviceArray2D	• BUFFER2D
• VkDeviceArray3D	• BUFFER3D
• ArrayList	• ?

GPU数据

传递给VkProgram的参数宏定义

如何降低Vulkan编程复杂度

- VkProgram运行机理

- 即时模式：

```
kernel("UpdateVelocity")->flush(
    vkDispatchSize(mCenter.size(), WORKGROUP_SIZE),
    &mVelocity,
    &mAngularVelocity,
    &mForceExt,
    &mTorqueExt,
    &mGravity,
    &mMassInv,
    &mInertiaWorldInv,
    &mBodyType,
    &mSolverState,
    &mTotalNum);
```

- 缓存模式： single shader

1、缓存

```
kernel("UpdateVelocity")->begin();
kernel("UpdateVelocity")->enqueue(
    vkDispatchSize3D(info.nx, info.ny, info.nz, 8),
    &vel_u,
    &vel_v,
    &vel_w,
    &mPressure,
    &mDensity,
    &mSDF,
    &mArrayInfo,
    &constDt);
kernel("UpdateVelocity")->end();
```

2、执行

```
kernel("UpdateVelocity")->update(true);
```

如何降低Vulkan编程复杂度

- VkMultiProgram

```
auto& solverFunc = this->createKernelGroup("ElasticitySolver");
solverFunc.add("SolveOneStep0", SolveOneStep0);
solverFunc.add("SolveOneStep1", SolveOneStep1);
solverFunc.add("Constrain", ConstraintFunc);
```

1、创建kernel函数

2、写入参数

```
solverFunc["SolveOneStep0"]->write(
    mVertexBuffer.handle(),
    this->inVertex()->getDataPtr()->handle(),
    this->inInitialVertex()->getDataPtr()->handle(),
    nbrIds.mIndex.handle(),
    nbrIds.mElements.handle(),
    &nbrIds.mInfo,
    &mTotalNum);

solverFunc["SolveOneStep1"]->write(
    this->inVertex()->getDataPtr()->handle(),
    mVertexBuffer.handle(),
    this->inInitialVertex()->getDataPtr()->handle(),
    nbrIds.mIndex.handle(),
    nbrIds.mElements.handle(),
    &nbrIds.mInfo,
    &mTotalNum);

solverFunc["Constrain"]->write(
    this->inVertex()->getDataPtr()->handle(),
    &mSphere,
    &mTotalNum);
```

```
for (int t = 0; t < 10; t++)
{
    solverFunc["SolveOneStep0"]->dispatch(
        vkDispatchSize(this->inVertex()->size(), WORKGROUP_SIZE));

    solverFunc["SolveOneStep1"]->dispatch(
        vkDispatchSize(this->inVertex()->size(), WORKGROUP_SIZE));

    solverFunc["Constrain"]->dispatch(
        vkDispatchSize(this->inVertex()->size(), WORKGROUP_SIZE));
}
```

3、Dispatch

如何降低Vulkan编程复杂度

- 简化后的案例参见：
  - examples/Vulkan/Tutorials/App_Catalyzer

如何降低Vulkan编程复杂度

- 已知数组A和B，求和并保存到数组C

App_VulkanNative

编程耗时10小时

Applicaiton: 7 hours

Shader: 5 min

Debug: 3 hours

App_Catalyzer

编程耗时13分钟

Applicaiton: 6 min

Shader: 5 min

Debug: 2 min

如何实现Vulkan后端与PeriDyno框架的衔接

- src://源码根目录
  - Core //基础数据结构,GPU无关
    - Backend: //GPU后端
      - Vulkan
      - Cuda
  - Framework //引擎框架 GPU无关
  - Dynamics: //仿真库根目录
  - Topology: //拓扑结构库，GPU相关
  - Rendering //渲染根目录
    - Engine //渲染引擎
    - GUI

引擎框架与底层解耦

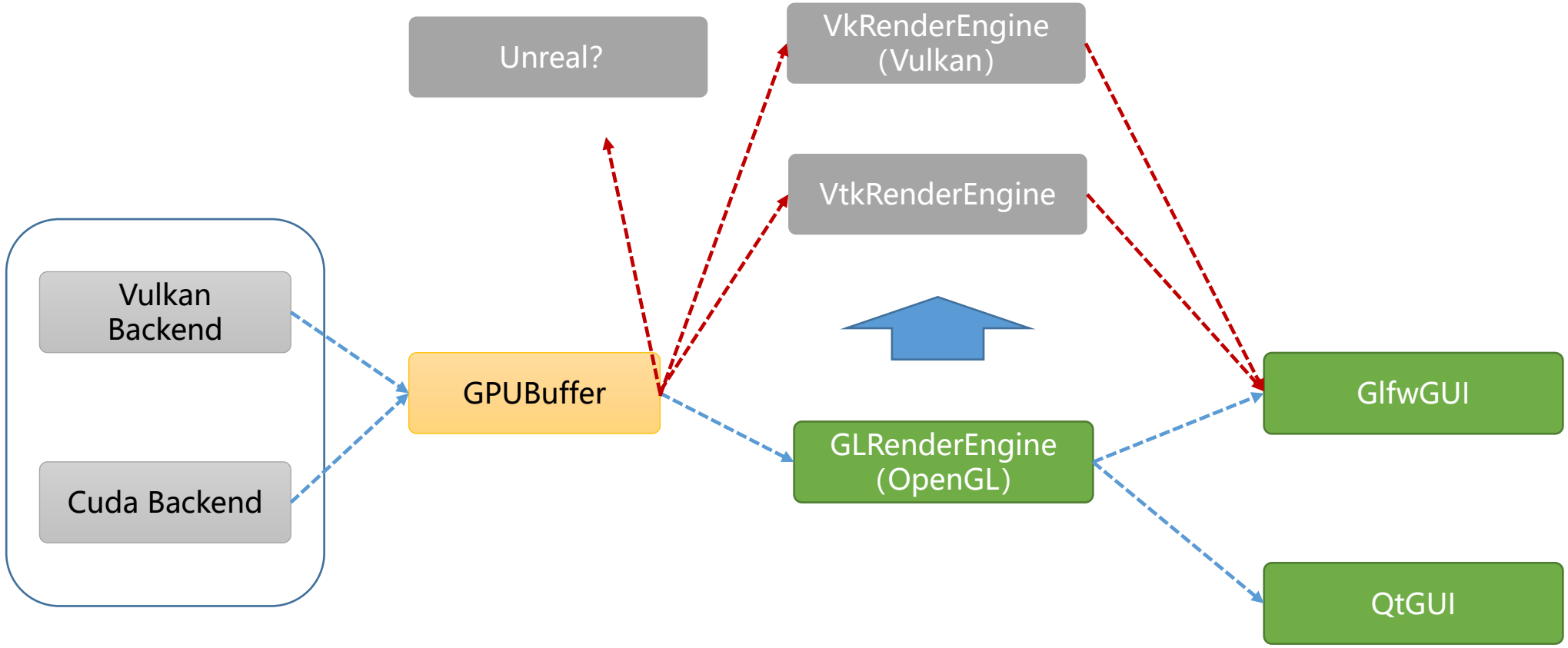
如何实现Vulkan后端与PeriDyno框架的衔接

- Cuda与Vulkan计算后端接口的统一

- 举例：Array

如何实现Vulkan后端与PeriDyno框架的衔接

- Cuda与Vulkan渲染后端接口的统一



Vulkan编程其他注意事项

- 数据结构对齐
  - Vec3f：16字节对齐
  - 自定义结构：float pad0填充4字节

```
struct ContactPair
{
    Vec3f pos0;
    Vec3f pos1;

    Vec3f normal0;
    Vec3f normal1;

    int id0;
    int id1;

    ConstraintType cType;
    float pad0;
};
```

Vulkan编程其他注意事项

- 不支持混合编译，导致代码维护成本较高

```
struct ContactPair
{
    Vec3f pos0;          //16字节对齐
    Vec3f pos1;          //16字节对齐

    Vec3f normal0;       //16字节对齐
    Vec3f normal1;       //16字节对齐

    int id0;             //4字节
    int id1;             //4字节

    ContactType cType;   //4字节
    float distance;      //4字节
};
```

Host

```
struct ContactPair
{
    vec4 pos0;
    vec4 pos1;

    vec4 normal0;
    vec4 normal1;

    int id0;
    int id1;

    uint cType;
    float distance;
};
```

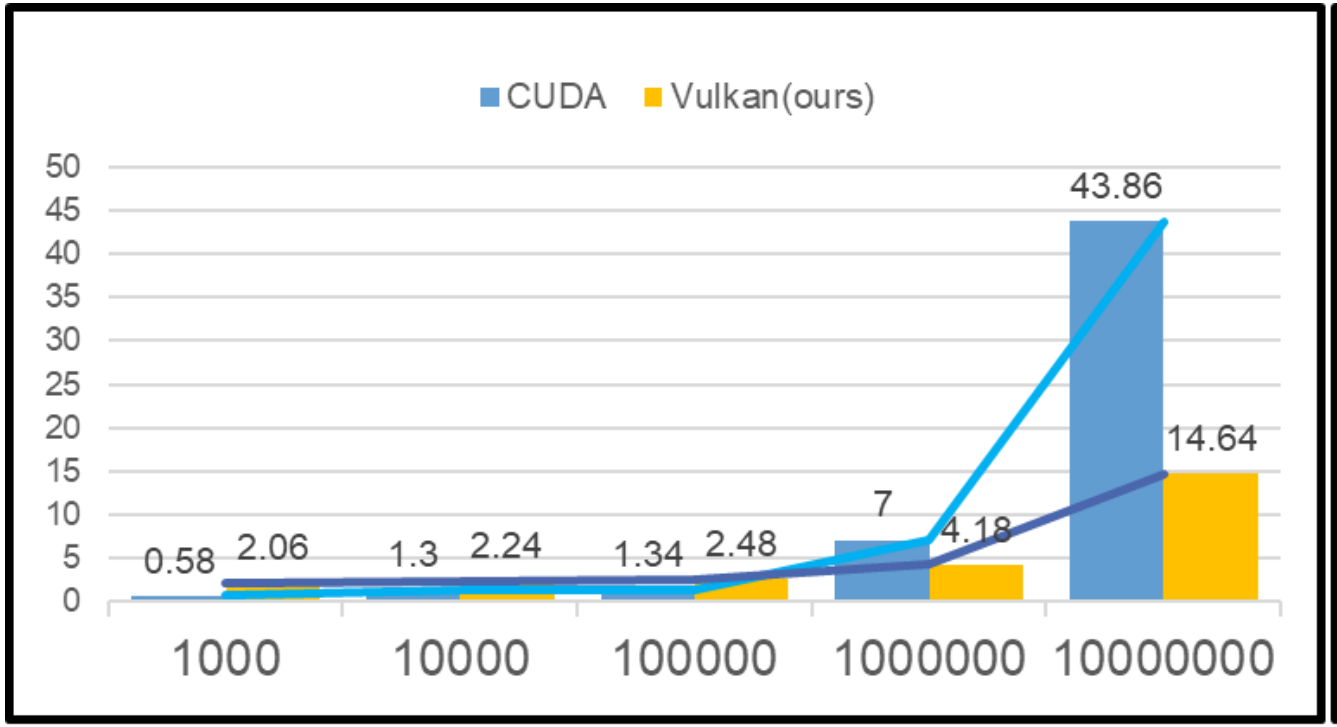
Device

Vulkan编程其他注意事项

- Vulkan其他局限性
  - 不支持C++高级特性
  - 不支持float原子操作
  - 不支持高级模板特性
  - 不同芯片支持的Vulkan特性差异较大，导致编程困难
  - 性能与便捷性较难兼顾

案例展示

- Catalyzer与thrust性能比较：Reduction

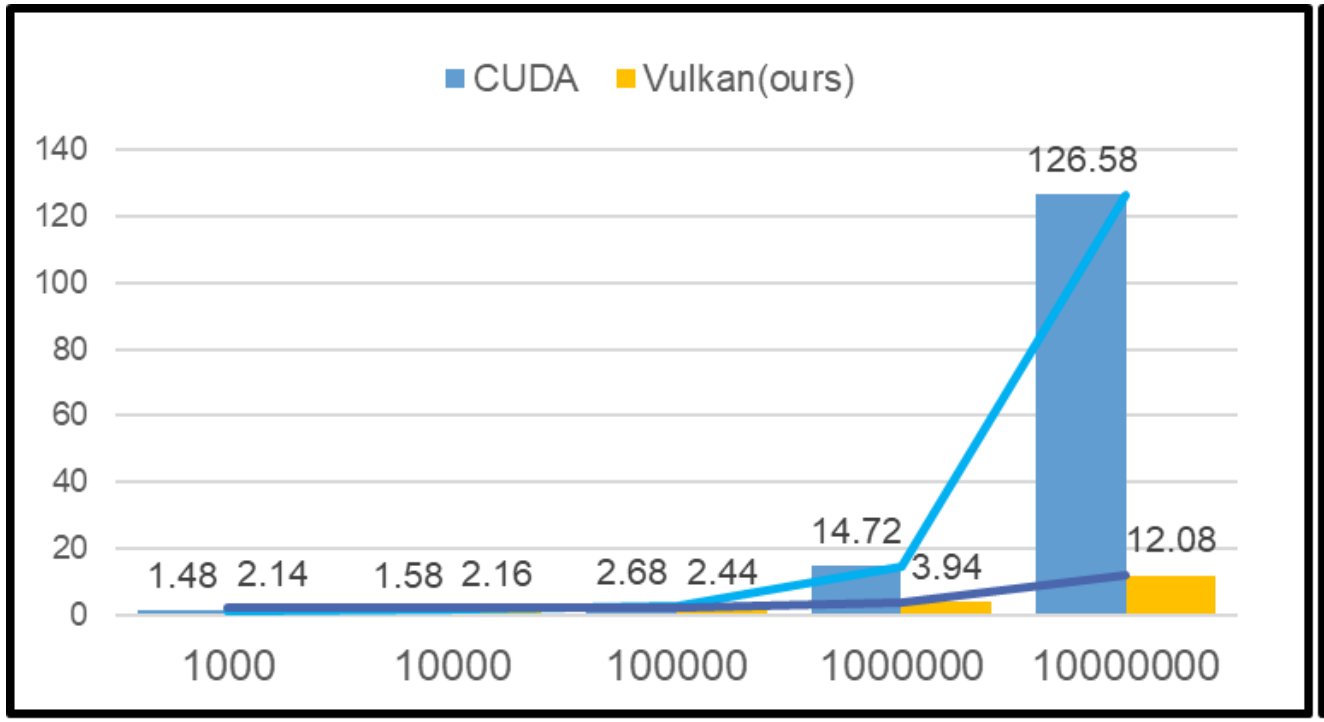


方法 \ 数据量	10^3	10^4	10^5	10^6	10^7
CUDA	0.58	1.3	1.34	7	43.86
Vulkan	2.06	2.24	2.48	**4.18**	**14.64**

数据为运行50次的时间平均值；单位：ms

案例展示

- Catalyzer与thrust性能比较：Inclusive scan

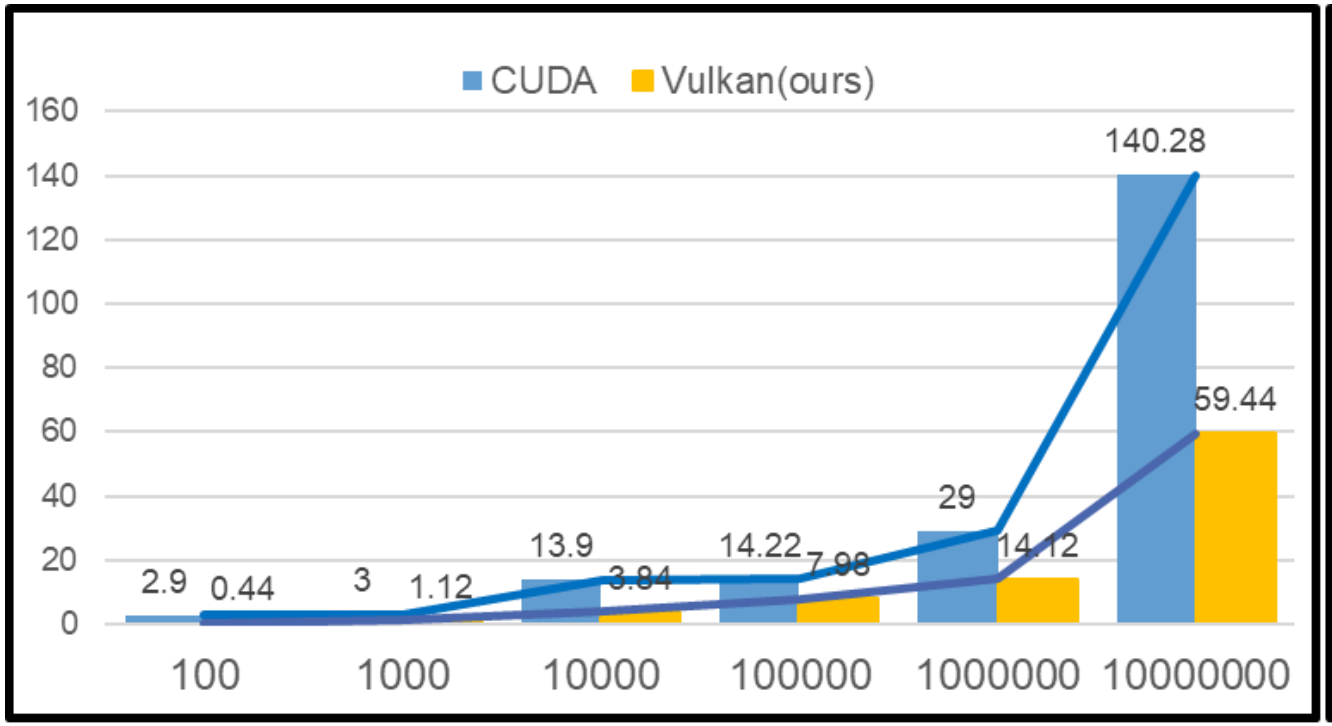


方法 \ 数据量	10^3	10^4	10^5	10^6	10^7
CUDA	1.48	1.58	2.68	14.72	126.58
Vulkan	2.14	2.16	2.44	3.94	12.08

数据为运行50次的时间平均值；单位：ms

案例展示

- Catalyzer与thrust性能比较：双调排序

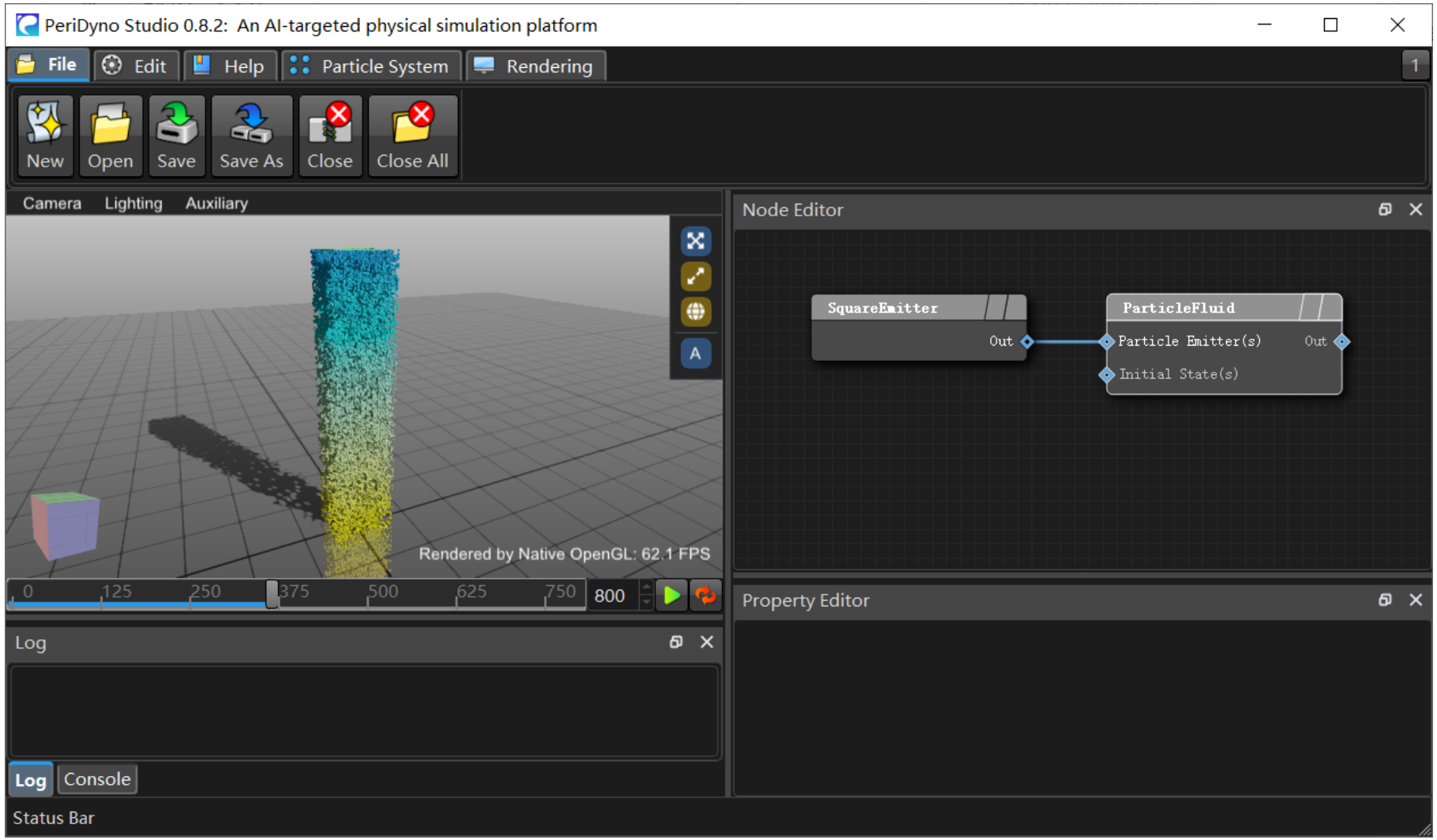


数据量 / 方法	10^2	10^3	10^4	10^5	10^6	10^7
std	0.1	0.5	5.4	65.24	600.88	5947
CUDA	2.9	3	13.9	14.22	29	140.3
Vulkan	0.44	1.120	3.840	7.98	14.12	59.44

数据为运行50次的时间平均值；单位：ms

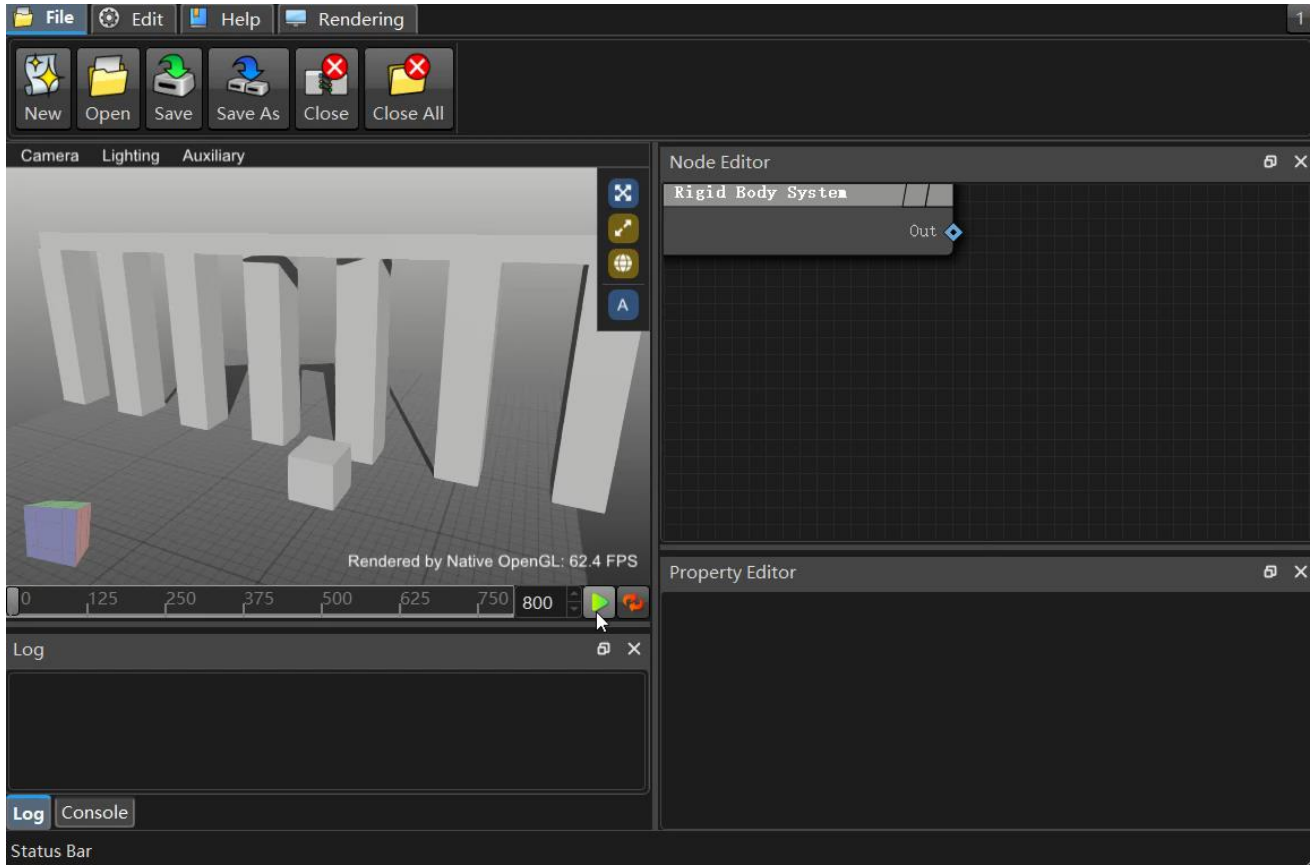
案例展示

- 粒子发射器

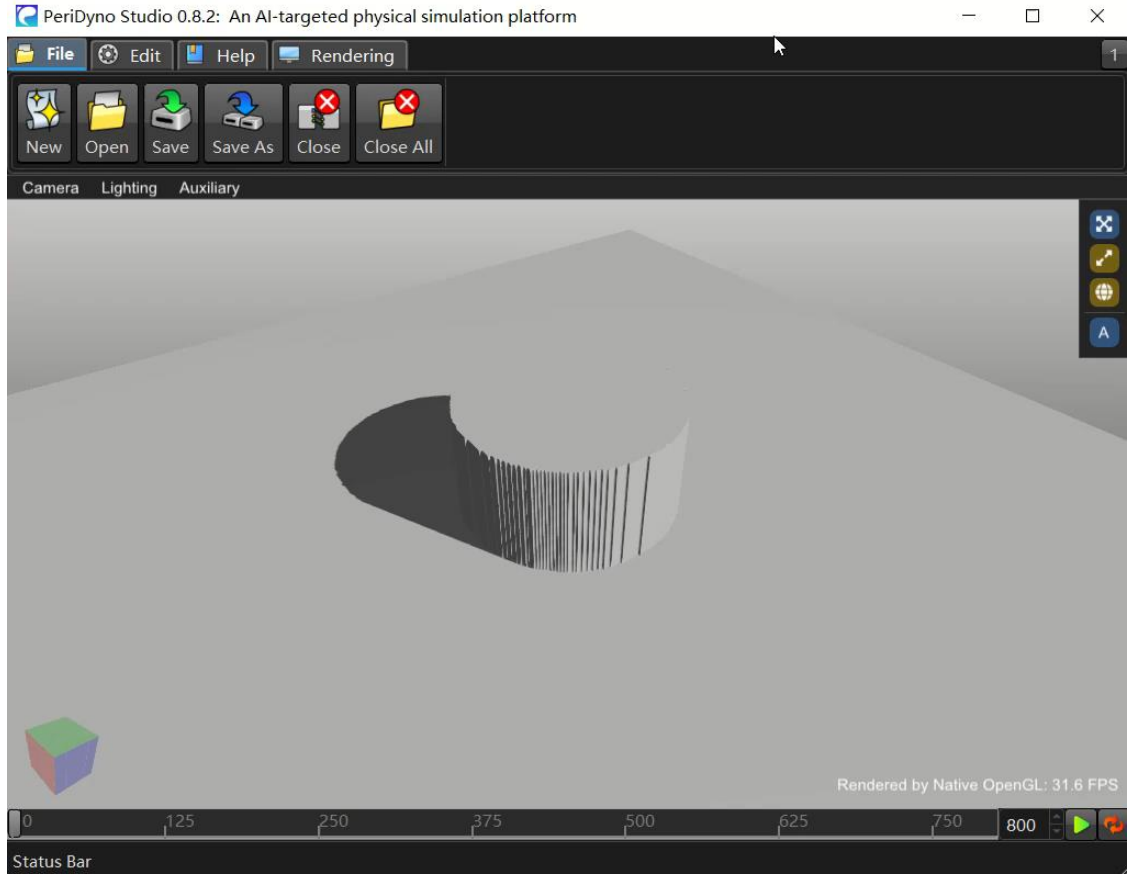


examples/Vulkan/SPH/Qt_ParticleFluid

案例展示



刚体动力学



波动方程

Further Reading

- https://vulkan-tutorial.com/
- https://vkguide.dev/
- https://github.com/SaschaWillems/Vulkan

结语

欢迎优秀同学(保研/考研)加入

数量有限，先到先到

邮箱：xiaowei@iscas.ac.cn；个人主页：www.peridynamics.com