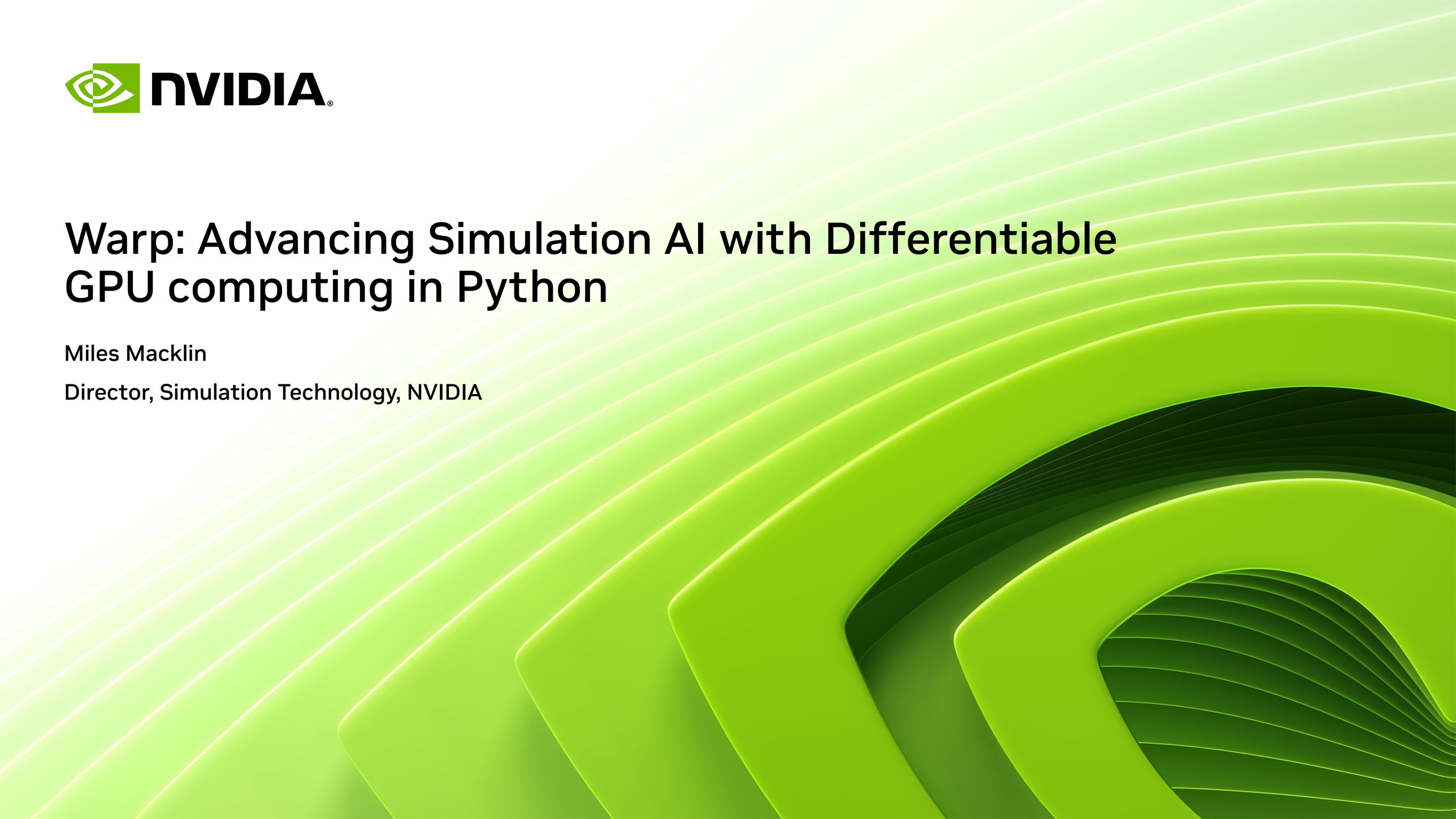


Warp: Advancing Simulation AI with Differentiable GPU computing in Python

Miles Macklin

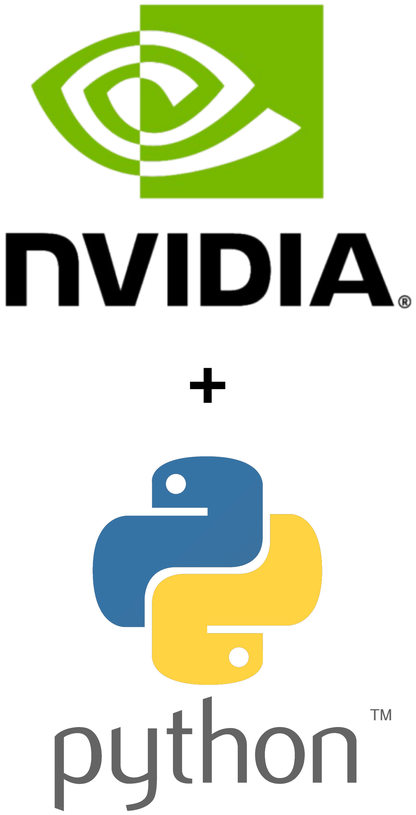
Director, Simulation Technology, NVIDIA

Agenda

- Introduction
- SDK Overview
- Case Studies
- Road Map
- Questions

Problem Statement

- CUDA is general purpose, efficient, but device centric:
  - C++ focused
  - Not specialized for simulation
  - Build everything yourself
  - Not differentiable
- Python is the lingua franca for AI
  - Low barrier to entry
  - Fast iteration and deployment
  - Rich ecosystem of DL frameworks
  - Often has poor performance
- How can we bridge the gap between CUDA and Python for simulation developers?



Introducing Warp

Differentiable Spatial Computing for Python

```
pip install warp-lang
```

1. GPU Kernels in Python

- Write CUDA kernels in 100% Python syntax
- Runtime JIT compilation
- Fast developer iteration

2. Spatial Computing

- Rich vector math library
- Mesh processing and queries
- Sparse volumes (OpenVDB)
- Hash grids

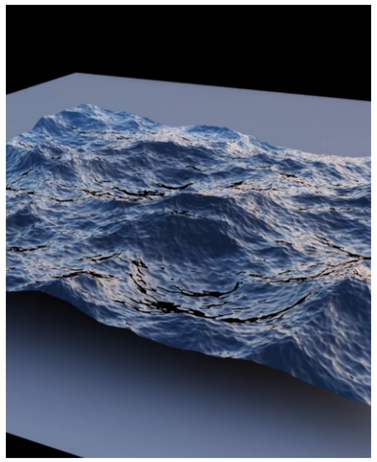
3. Differentiable Programming

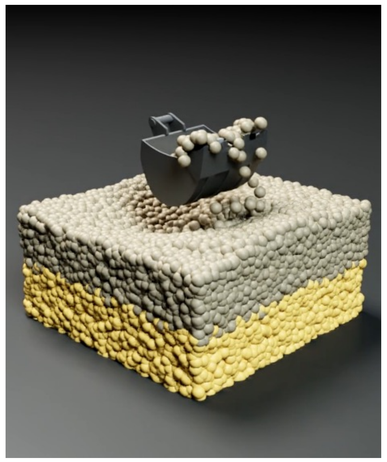
- Auto-differentiation
- Forward + backward kernel generation
- Interop with DL frameworks (e.g.: PyTorch, JAX)

4. Omniverse Integration

- OpenUSD import / export
- Runtime extensions for Isaac / Composer

```python
import warp as wp

@wp.kernel
def integrate(p: wp.array(dtype=wp.vec3),
              v: wp.array(dtype=wp.vec3),
              f: wp.array(dtype=wp.vec3),
              m: wp.array(dtype=float)):

    # thread id
    tid = wp.tid()

    # Semi-implicit Euler step
    v[tid] = v[tid] + (f[tid] * m[tid] + wp.vec3(0.0, -9.8, 0.0)) * dt
    x[tid] = x[tid] + v[tid] * dt

# kernel launch
wp.launch(integrate, dim=1024, inputs=[x, v, f, …], device="cuda:0")
```

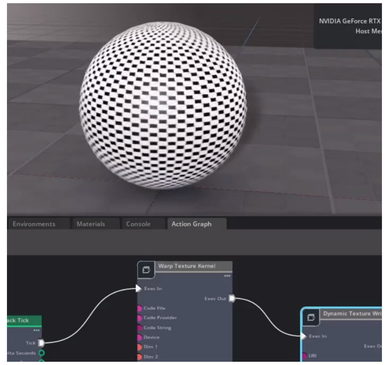




Warp Use Cases

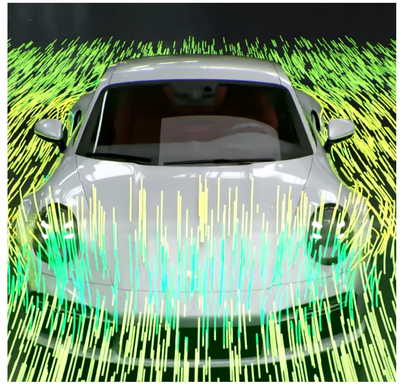
- Warp provides the building blocks for developers to create, accelerate and extend their own simulators

Data Processing



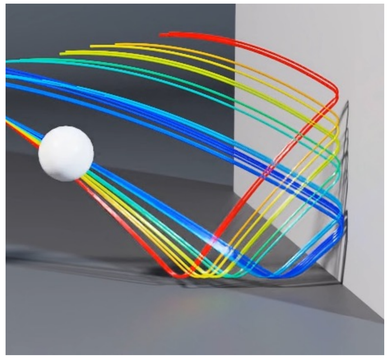
- Mesh processing
- Image processing
- Synthetic data generation

Simulation



- Rigid body dynamics
- Elasticity
- Fluid flow

Training



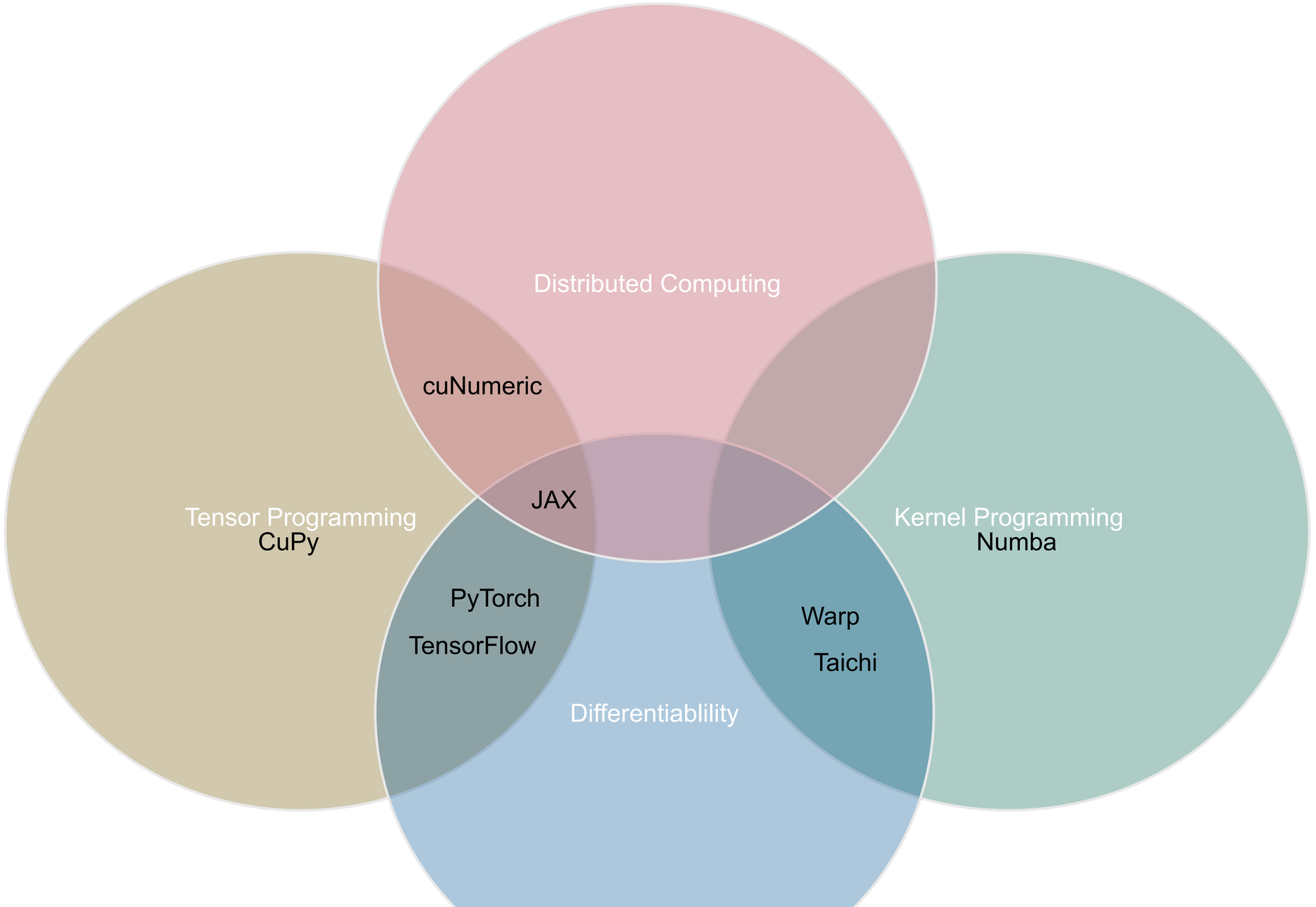
- Neural dynamics
- Parameter estimation
- Trajectory optimization
- Inverse problems

Scripting

```
@wp.kernel
def initialize_particles(
    particle_x: wp.array(dtype=wp.vec3), sm
):
    tid = wp.tid()

    # grid size
    nr_x = wp.int32(width / 4.0 / smoothing
    nr_y = wp.int32(height / smoothing_leng
    nr_z = wp.int32(length / 4.0 / smoothin

    # calculate particle position
    z = wp.float(tid % nr_z)
    y = wp.float((tid // nr_z) % nr_y)
    x = wp.float((tid // (nr_z * nr_y)) % n
```

- Loss/reward functions
- Custom forces
- Custom behaviors
- Custom boundaries

Python GPU Ecosystem

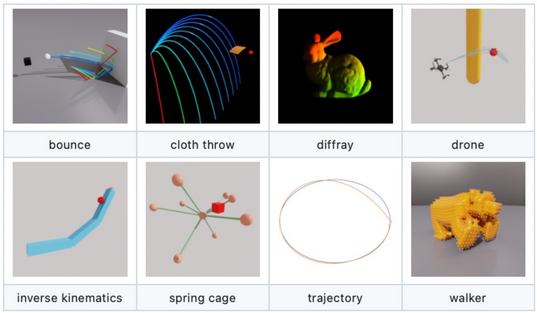


Warp SDK

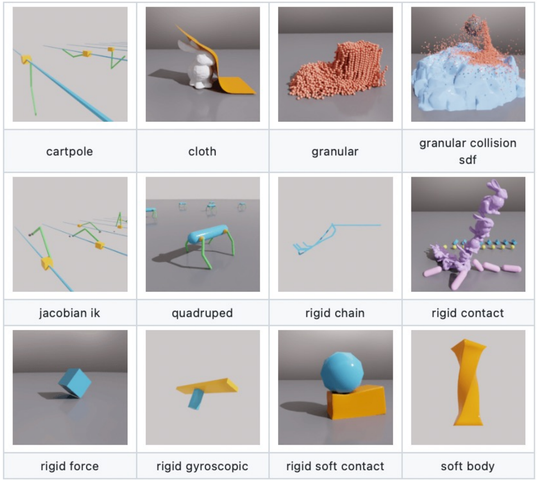
Warp Python Modules

- `warp.core`
  - Differentiable kernel coding for Python
  - Math, geometry, vector library
- `warp.sim`
  - Differentiable real-time simulation for robotic control + prediction
  - Rigid bodies, soft bodies, particles, cloth
  - URDF, MJCF, UsdPhysics parsers
- `warp.fem (early access)`
  - Differentiable PDE framework for heat transfer, diffusion, elasticity
  - Fast iteration, but offline focus
  - Not replacing existing codes, but potential to build on them
- `warp.llm (early access)`
  - AI agent specialized for Warp kernel coding
  - Generate simulation code directly from prompts
  - Generate loss functions from code -> optimization

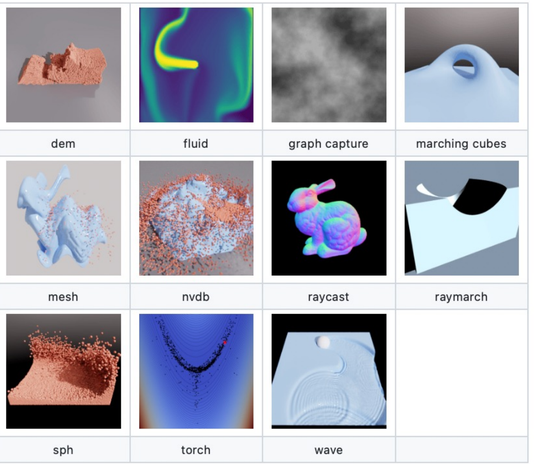
examples/optim



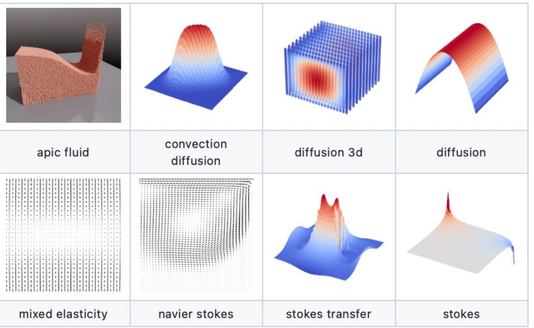
examples/sim



examples/core



examples/fem

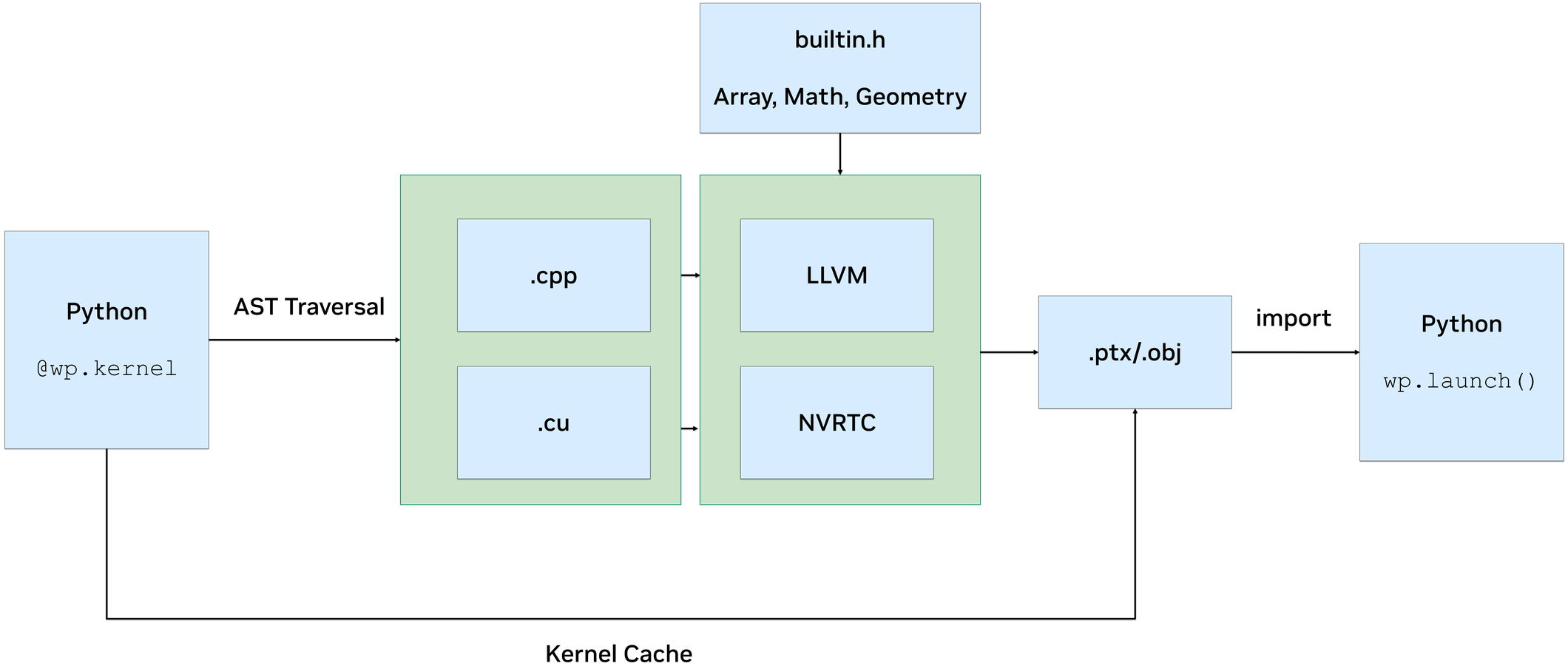


Warp Data Model

- Host/Device memory managed through wp.array type
- Builtin spatial math types similar to OpenCL/HLSL:
  - vec2, vec3, vec4, mat22, mat33, mat44, quat, transform
- Support for all common array protocols:
  - __array_interface__
  - __cuda_array_interface__
  - __dlplack__
- Zero-copy interop with PyTorch, JAX, NumPy:
  - wp.from_torch(), wp.to_torch()
  - wp.from_jax(), wp.to_jax()

```
# 1D array of int8 types:
a = wp.zeros(shape=[64], dtype=wp.int8)

# 2D array of fp16 vec3 types:
a = wp.zeros(shape=[64, 64], dtype=wp.vec3h)

# 3D array of fp64 mat44 types:
a = wp.zeros(shape=[64, 64, 64], dtype=wp.mat44d)
```

Warp Compilation Pipeline



Warp Execution Model

- Warp exposes a thin abstraction over CUDA
- Kernels are launched over an N-dimensional grid of threads
  - Max 4-dimensional grids
  - Mapping to blocks is handled internally and not exposed to user
  - Grid stride kernels used to scale to large thread counts
    - Max up to 2^31-1 on each dimension
- Pure SIMT model
  - No shared memory
  - No warp-level primitives, e.g.: __shfl_sync()
- Custom native functions
  - C++/CUDA snippets

```python
@wp.kernel
def divergence(u: wp.array2d(dtype=wp.vec2),
               div: wp.array2d(dtype=float)):

    # 2D thread indices
    i, j = wp.tid()

    # boundary conditions
    if i == grid_width - 1:
        return
    if j == grid_height - 1:
        return

    # compute divergence
    dx = (u[i + 1, j][0] - u[i, j][0]) * 0.5
    dy = (u[i, j + 1][1] - u[i, j][1]) * 0.5
    div[i, j] = dx + dy

# 2d kernel launch
wp.launch(divergence, dim=[512, 512], inputs=[u, div])
```

Example: 2D Divergence Calculation in Warp

Custom CUDA Code

Inserting native code snippets

- Add custom CUDA code snippets to Warp directly from Python via @wp.func_native decorator
- Allows easily dropping down to CUDA for access to:
  - Shared memory
  - Fine grained synchronization
  - Cooperative operations
- Custom native backward functions
- Ability to include larger C++/CUDA codes (header-only) coming soon

```
snippet = """
out[tid] = a * x[tid] + y[tid];
"""

adj_snippet = """
adj_a = x[tid] * adj_out[tid];
adj_x[tid] = a * adj_out[tid];
adj_y[tid] = adj_out[tid];
"""

@wp.func_native(snippet, adj_snippet)
def saxpy(
    a: wp.float32,
    x: wp.array(dtype=wp.float32),
    y: wp.array(dtype=wp.float32),
    out: wp.array(dtype=wp.float32),
    tid: int,
):
    ...
```

Example: Custom CUDA snippet as a Warp function

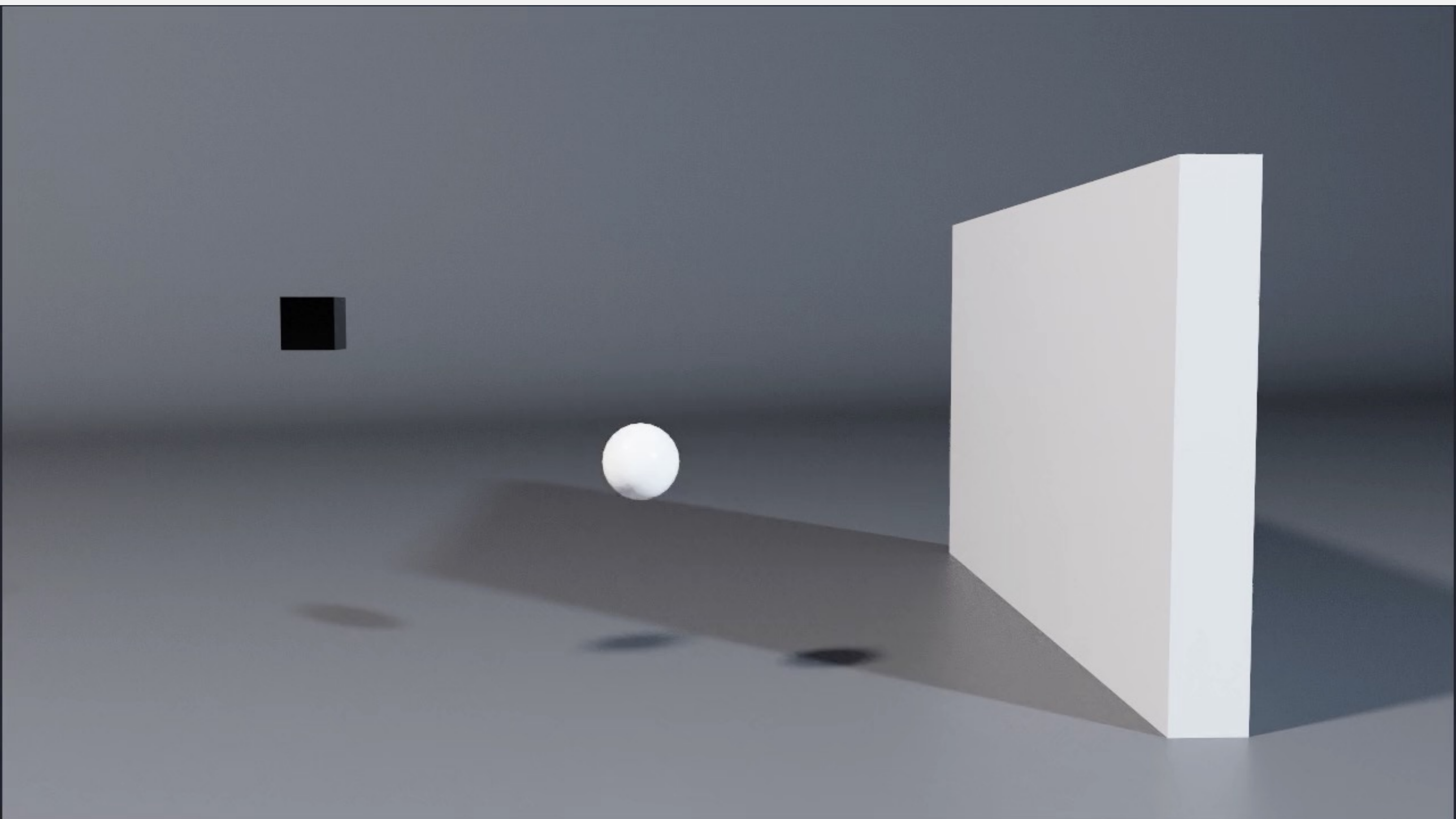
Automatic Differentiation

Differentiable Programming in Warp

- Reverse-mode automatic differentiation
- Kernel adjoint codes are generated automatically
- Store and reply kernels using wp.Tape()
- User-provided custom gradients
- Limitations:
  - First-order derivatives only
  - Dynamic loops and side-effects
- Capture entire backward pass in a CUDA graph

```
tape = wp.Tape()

with tape:
    wp.launch(kernel=kernel_1, dim=dim, inputs=[a], outputs=[b])
    wp.launch(kernel=kernel_2, dim=dim, inputs=[b], outputs=[c])
    wp.launch(kernel=kernel_3, dim=dim, inputs=[c], outputs=[loss])

# run backward
tape.backward(loss)

# gradients of loss w.r.t inputs
print(a.grad)
print(b.grad)
print(c.grad)
```

Example: Auto-differentiation through multiple kernel launches



Mesh Data Structure

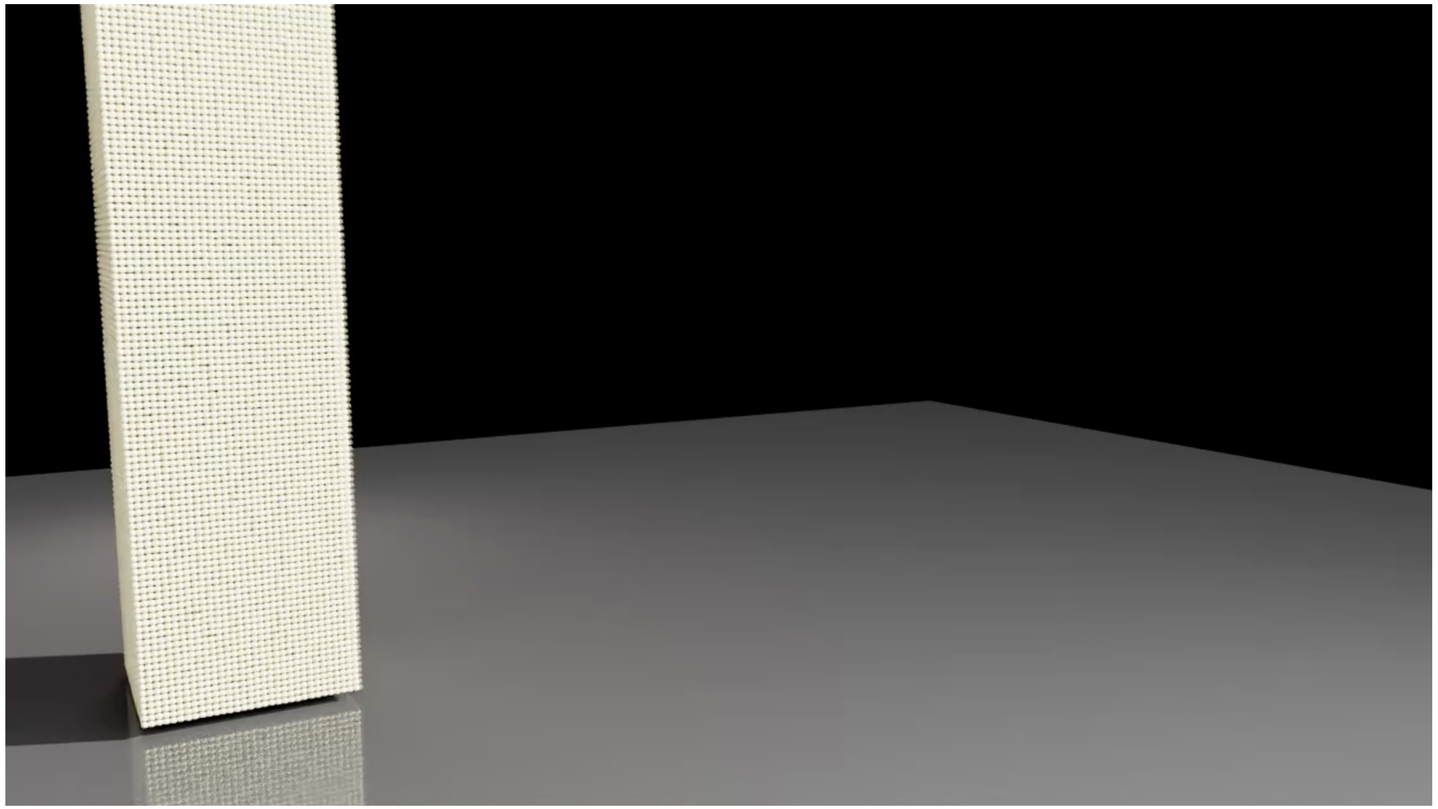
Accelerated Mesh Queries

- Warp mesh data structure:
  - wp.Mesh
- Built in mesh queries:
  - wp.mesh_query_point()
  - wp.mesh_query_ray()
  - wp.mesh_query_aabb()
- Fast inside/outside (sign) determination
- Fast GPU bounding volume hierarchy (BVH) refits:
  - mesh.refit()
- Example:
  - Neo-Hookean FEM elastic model
  - 60k particles versus 32k tris in 0.25ms
  - Dynamic signed-distance field (SDF) contact against meshes

Hash Grid Data Structure

Nearest Neighbor Point Queries

- Built-in hash-grid primitive
  - wp.HashGrid
- Fast neighbor finding / radius queries:
  - wp.hash_grid_query_point()
  - wp.hash_grid_query_next()
- GPU-side rebuild:
  - hashgrid.build()
- 10x faster than Open3D
- Example:
  - Discrete Element Method (DEM) in 200 lines of Python
  - 128k particles ~10ms/frame



Sparse Grid Data Structure

OpenVDB

- Support for NanoVDB sparse volumes
  - wp.Volume
- Great for narrow-band SDF collision
- Simple to use API:
  - wp.volume_sample_world(vol, xyz)
  - wp.volume_sample_local(vol, uvw)
  - wp.volume_lookup(vol, ijk)
  - wp.volume_transform(vol, xyz)
  - wp.volume_transform_inv(vol, xyz)

OpenVDB

Warp Sim

Differentiable Simulation for Robotics

- A framework for real-time, differentiable, robotic simulation
- Physical Models
  - Rigid bodies
  - Particles
  - Constraints
  - Geometry
  - Forces
- Physical State
  - Position (q)
  - Velocity (qd)
- Integrators
  - Symplectic Euler (semi-implicit)
  - XPBD (implicit)
  - Featherstone


```python
import warp as wp
import warp.sim

sim_dt = 1/60

builder = wp.sim.ModelBuilder()
wp.sim.parse_urdf("cartpole.urdf", builder)
model = builder.finalize()

integrator = wp.sim.SemiImplicitIntegrator()
state, next_state = model.state(), model.state()

for i in range(100):
    state.clear_forces()
    state = integrator.simulate(model, state, next_state, sim_dt)
    state, next_state = next_state, state
```

Example: Creating a cartpole simulation using warp.sim

Warp FEM

Differentiable Framework for PDEs

- Flexible finite element-based (FEM/DG) framework for
  - Diffusion
  - Convection
  - Fluid flow
  - Elasticity
- Define your weak-form PDE in high-level Python syntax:
  - wp.fem.grad()
  - wp.fem.div()
  - wp.fem.curl()
- Combine with geometry + shape functions:
  - wp.fem.TetMesh()
  - wp.fem.HexMesh()
  - wp.fem.make_polynomial_space()
- Integrate to assemble linear system:
  - wp.fem.integrate()
- Solve with block-sparse (BSR) solvers:
  - warp.sparse.BsrMatrix

Define PDE

```python
import warp.fem as fem

@fem.integrand
def diffusion_form(s: fem.Sample, u: fem.Field, v: fem.Field, nu: float):
    return nu * wp.dot(
        fem.grad(u, s),
        fem.grad(v, s),
)
```

Define Geometry

```python
geo = fem.Tetmesh(positions=positions, tet_vertex_indices=indices)
scalar_space = fem.make_polynomial_space(
    geo, degree=2, element_basis=fem.ElementBasis.SERENDIPITY)
```

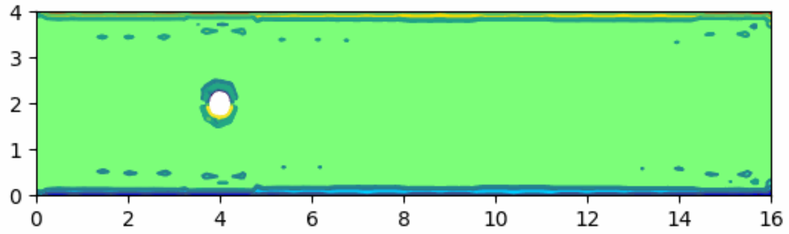
Assemble System

```python
trial = fem.make_trial(scalar_space)
test = fem.make_test(scalar_space)

matrix = fem.integrate(diffusion_form,
            fields={"u": trial, "v": test},
            values={"nu": 0.1})
```

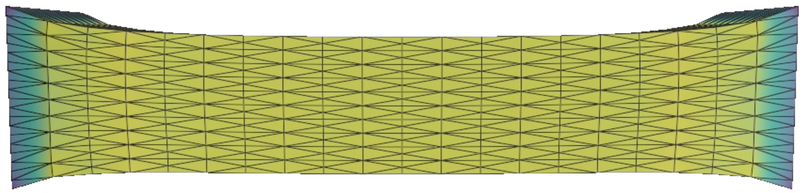
Example: Diffusion PDE bilinear form

Warp FEM

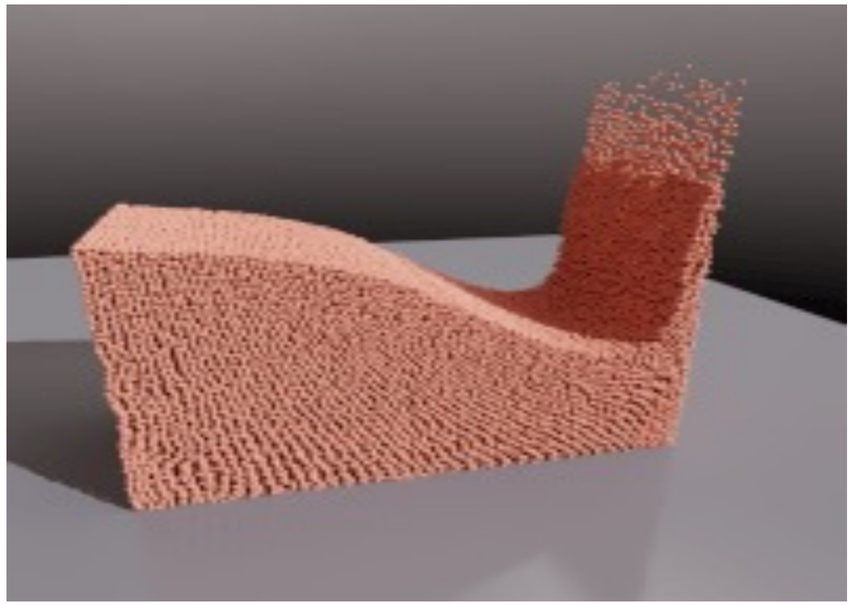
Examples



Example: Navier—Stokes flow simulation



Example: Neo-Hookean Elasticity



Example: Hybrid Eulerian—Lagrangian fluids

Case Studies

Neural Constitutive Laws

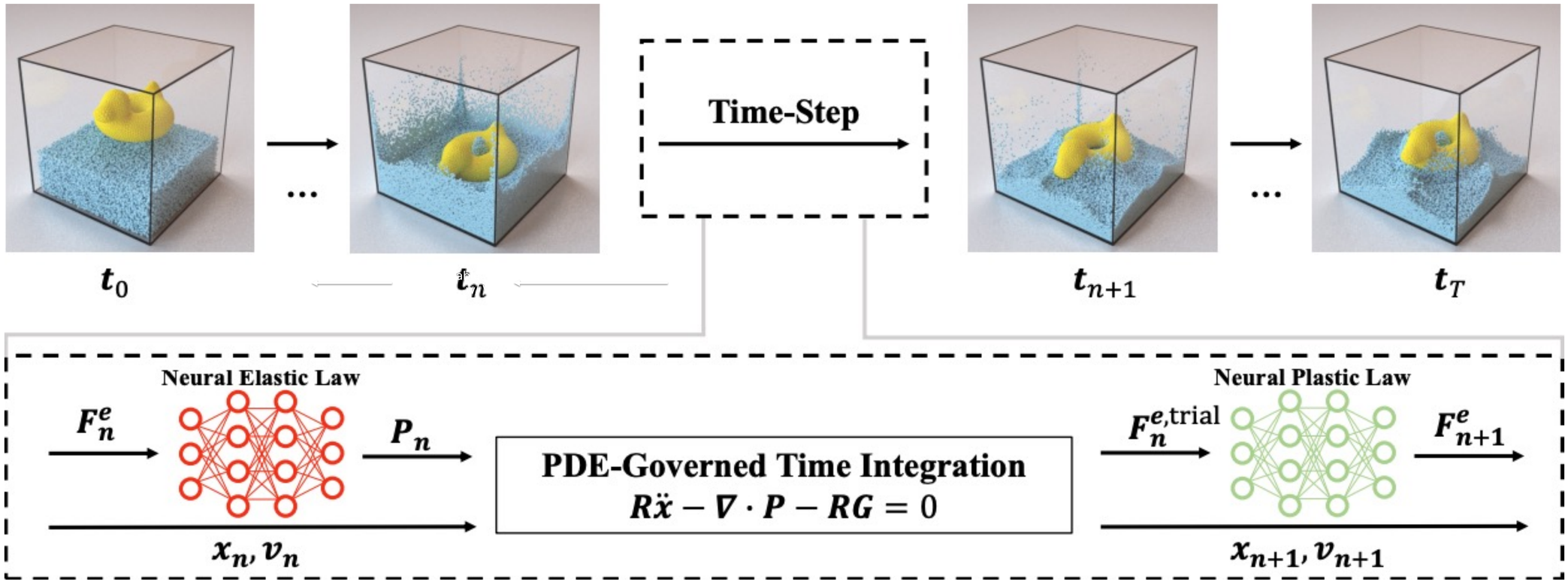
Learned Elastic Modeling

- Goal: Learn constitutive material models from video
- Research from MIT CSAIL @ ICML 2023
- Differentiable MPM solver written in Warp
- Use to train DNN constitutive model to match captured deformation
- Generalizes to 30x larger environments, unseen geometries and trajectories
- Handles multi-physics environments: coupled rigid body–fluid simulations, phase transitions
- https://github.com/PingchuanMa/NCLaw
- [Ma et al. 2023]

Learning Neural Constitutive Laws

Neural Constitutive Laws

Pipeline



Neural Networks w/ PyTorch

Differentiable MPM Simulation w/ Warp

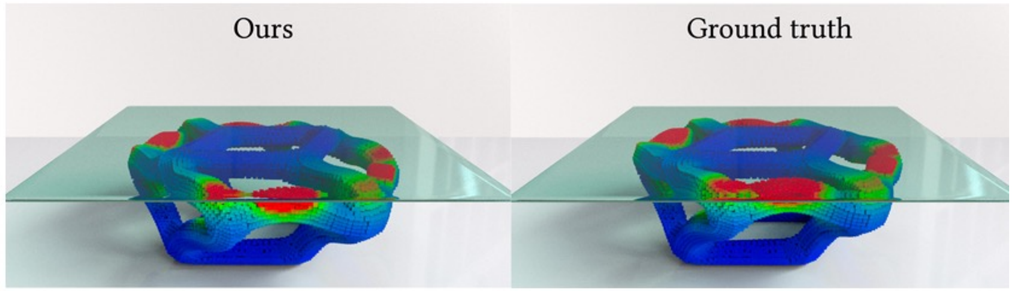
Neural Networks w/ PyTorch

Neural Stress Fields

Reduced-Order Elasto-plasticity and Fracture

- Goal: Accelerated MPM solvers for elastoplastic simulation

- Created differentiable MPM solver in Warp
- Learn a reduced neural basis to represent stress fields
- Dimension reduction of up to 100,000x
- 10x faster simulation
- Generalizes to a broad range of materials
- Joint work from UCLA, CMU, MIT, UToT
- https://zeshunzong.github.io/reduced-order-mpm/
- [Zhong et al. 2023]



Large Scale Fluid Simulation

Modulus

XLB + Warp

XLB + GH200 + Warp + nvCOMP

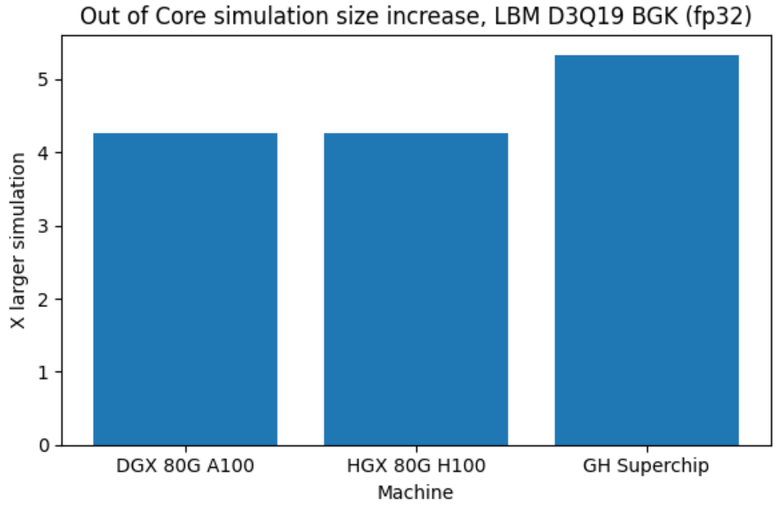
XLB is a Lattice Boltzmann fluid dynamics solver developed by Autodesk Research.

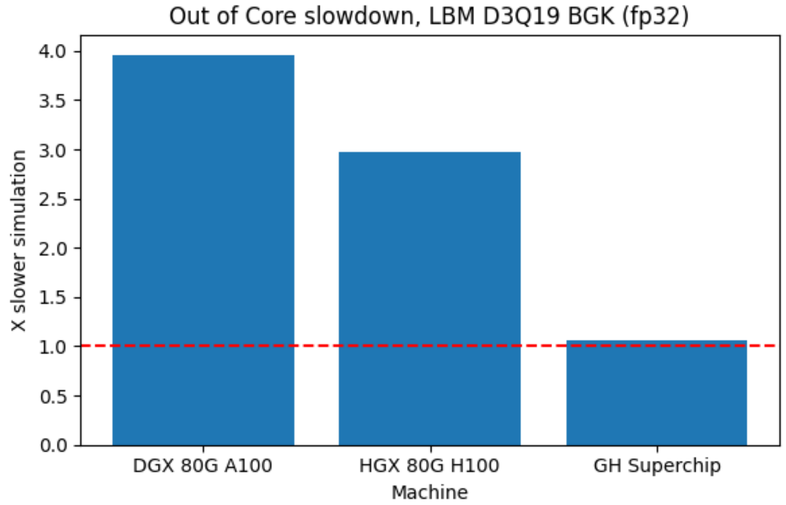
Developed in Python using JAX with key features:

- Easily extendable to add new methods
- Scalable across distributed systems
- ML friendly allowing integration with physics-based ML models
- Topological optimization and inverse problems

XLB is being restructured to allow Warp as a compute backend. This results in ~3.8x speed improvement and ~3.1x larger simulations

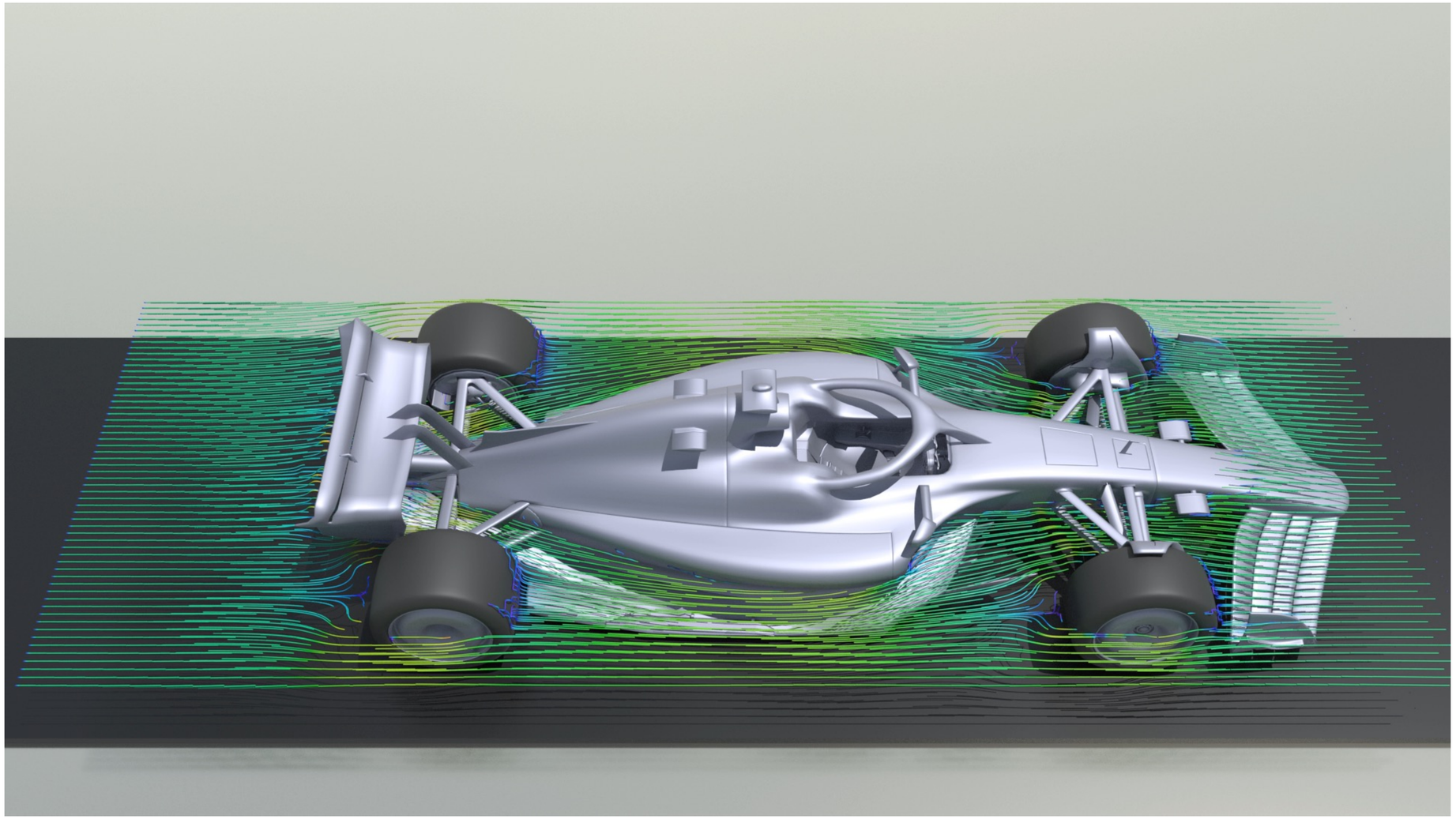
Combining XLB, Warp, nvCOMP and our Grace Hopper super chips allows for efficient out of core memory simulations. This allows for ~5x larger simulations with minimal impact on performance.





Large Scale Fluid Simulation

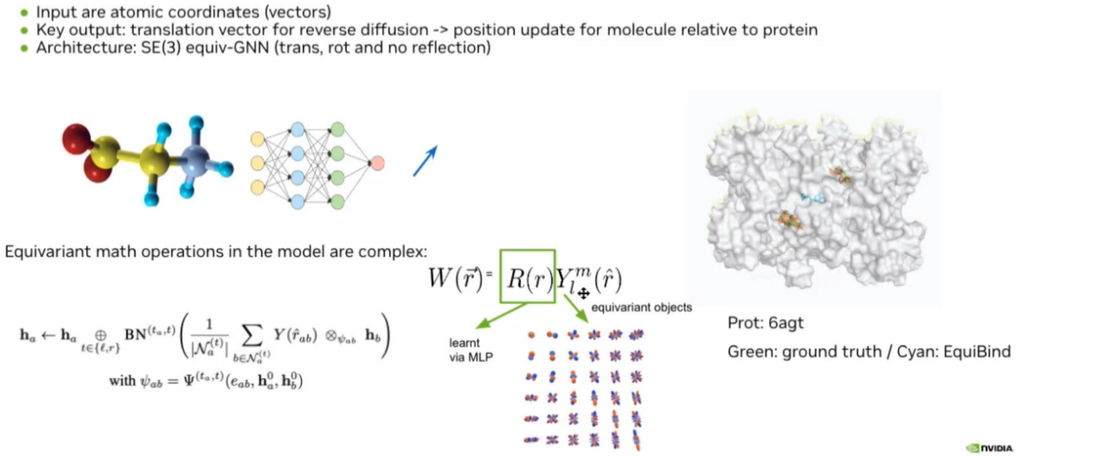
Aerodynamic Simulation

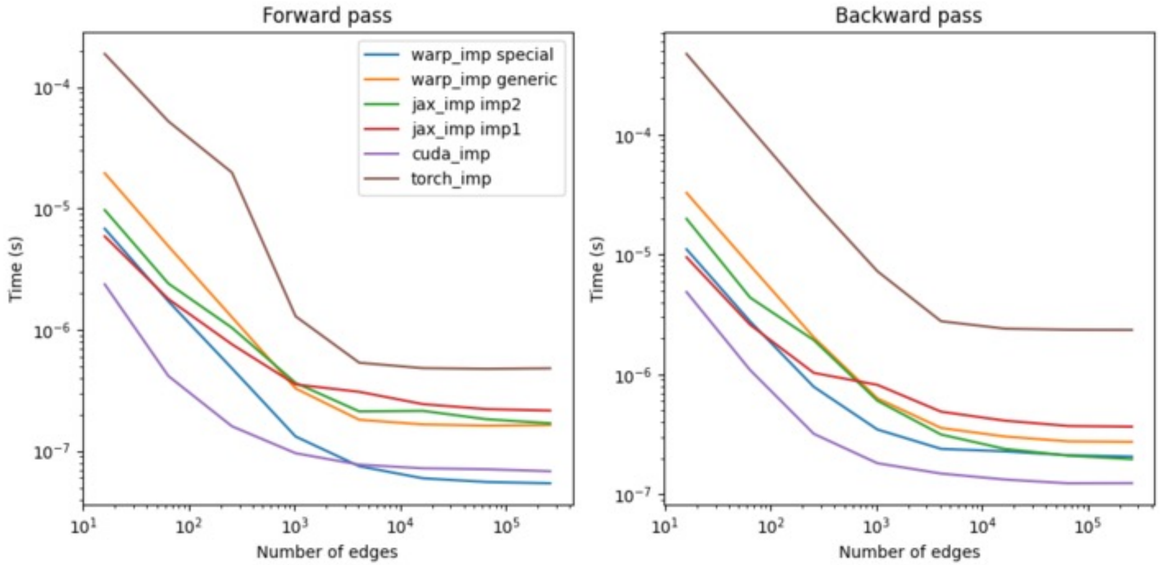


Differentiable Molecular Dynamics

Protein Docking

- Goal: Accelerate protein docking simulation
- Implement equivariant operators
  - Rotation / translation invariant
- Fastest ML framework implementation in Warp:
  - 10x faster than Torch
  - Near native CUDA performance at a fraction of the effort
- Create customizable differentiable kernels at runtime for specific models, e.g.: DiffDock*
  - Runtime kernel specialization for different datatypes, dimensions, etc.
  - Faster than fixed-parameter hand rolled CUDA
- * Corso et al. [2023]



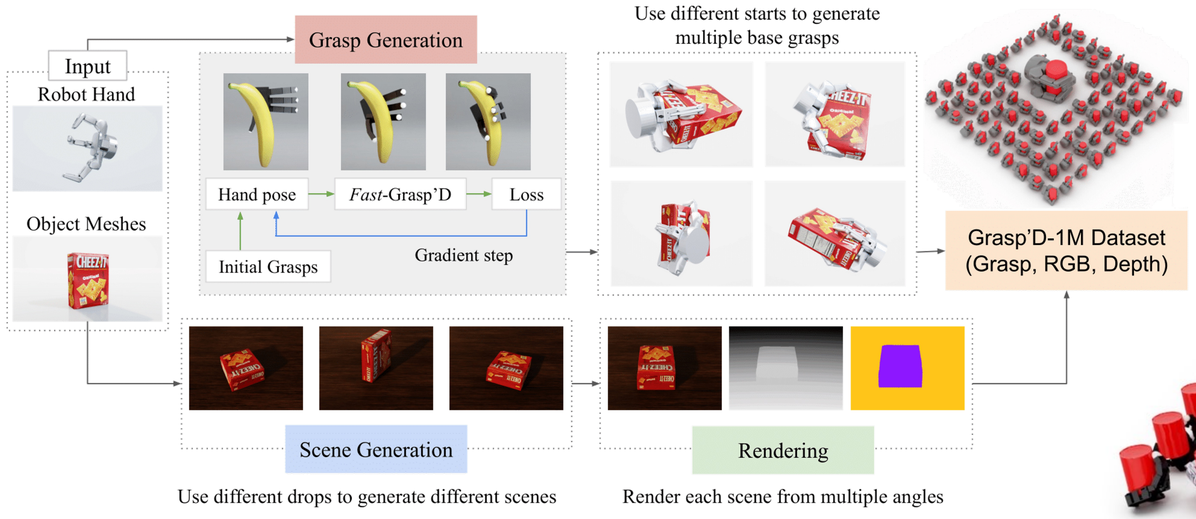


Fast-Grasp’D

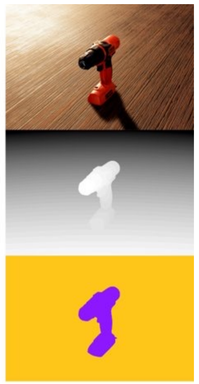
Dexterous Multi-finger Grasp Generation

- Goal: Generate realistic and stable grasps for robotic hand / object pairs

- Research from UofT + NVIDIA
- Differentiable SDF contact with rigid bodies
- 20x faster than previous SOTA
- Generated over 1M high quality grasps in 140 GPU hours over multiple hand types



https://dexgrasp.github.io



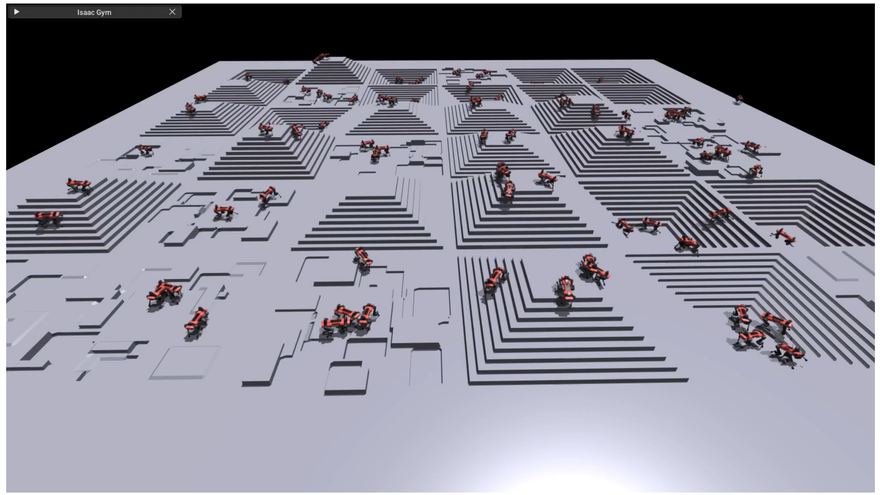
Aerial Vehicle Simulation

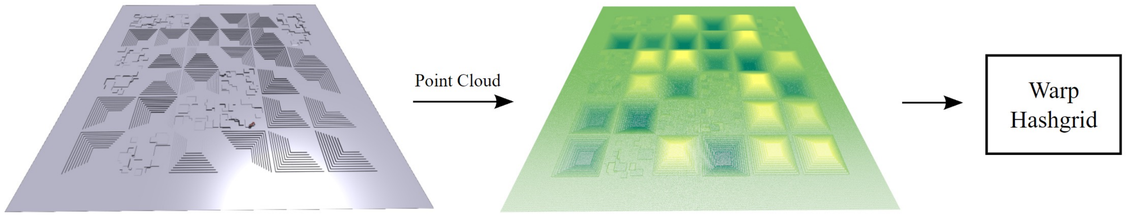
Differentiable Dynamics simulation for Real-Time Control

Robotic Perception

ANYmal Quadruped Training

- Goal: Robot perception and processing for locomotion
- Extract the point cloud around 4000 robots using the built-in Warp hash grid
- ~12-16M points, 20ms for all the robots -> 27k FPS total
- Additional LIDAR simulation via. mesh ray-casts
- Deployed on Jetson AGX Orin hardware





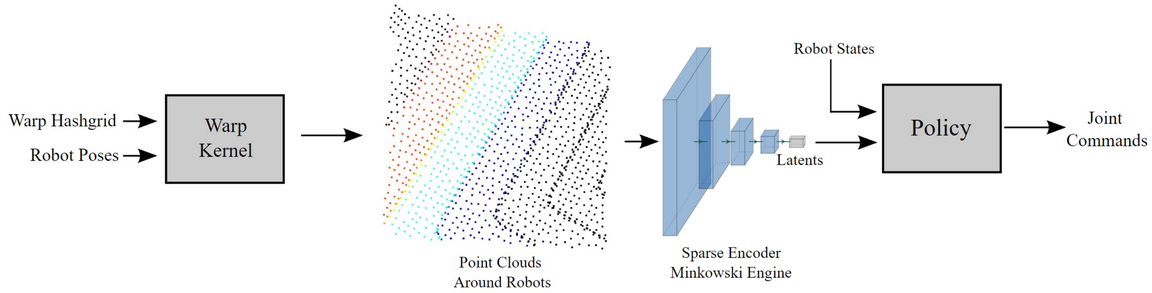


Image Courtesty Anybiotics ANYmal

Roadmap

Road Map

Math Library Interop

- Integration with nvmath Python libraries
- Device-side versions of CUDA libraries:
  - cuBLASDx
  - cuFFTDx
- e.g. invoke FFT directly from inside Warp kernels
- Fully fused execution with user code
- See Math Libraries Python talk on Wednesday!

Deep Dive into Math Libraries [S62162]

Arthy Sundaram, Senior Product Manager, NVIDIA

Harun Bayraktar, Director of Engineering, Math Libraries, NVIDIA

NVIDIA's GPU-accelerated Math Libraries, which are part of the CUDA Toolkit and the HPC SDK, are constantly expanding, providing industry-leading performance and coverage of common compute workflows across AI, ML, and HPC. We'll do a deep dive...

Add to Schedule

Wednesday, Mar 20 | 9:00 AM - 9:50 AM PDT
(Thursday, Mar 21 | 5:00 AM - 5:50 AM NZDT)

Hilton Almaden 1 (L1)

```python
import nvmath as nvm
import warp as wp

FFT_fwd = nvm.device.fft(type='c2c', size=size, precision=wp.float32, direction='forward', dsl='warp')
FFT_inv = nvm.device.fft(type='c2c', size=size, precision=wp.float32, direction='inverse', dsl='warp')

@wp.kernel
def compute(input: wp.array(dtype=float),
            output: wp.array(dtype=float),
            fwd: FFT_fwd,
            inv: FFT_inv):

    i = wp.tid()

    # epilogue
    thread_data = user_func(input[i])

    # execute foward FFT
    fwd(FFT_fwd, thread_data)

    # frequency space convolution
    thread_data /= scale

    # execute inverse FFT
    inv(FFT_inv, thread_data)

    # prologue
    output[i] = thread_data
```

Example: Fused Forward/Reverse FFT

Road Map

Warp Neural Networks

- Leverage TensorCore operations (MMA) inside Warp kernels, e.g.:
  - wp.linear()
  - wp.map()
- Ideal for building fused neural network inference in Warp kernels
- Keep all intermediate values local to kernel
- Network inference in-situ

```
@wp.kernel
def eval(layer0: Layer,
         layer1: Layer,
         loss: wp.array(dtype=float)):

    # setup feature vectors / input
    x = wp.vector(dtype=wp.float16, length=DIM_IN)
    x[0] = 0.0
    x[1] = 1.0
    x[2] = 2.0
    x[3] = 3.0

    # network eval
    z = wp.linear(layer0.weights, layer0.bias, DIM_HID, z)
    z = wp.map(relu, z)
    z = wp.linear(layer1.weights, layer1.bias, DIM_OUT, z)
    z = wp.map(relu, z)

    # compute loss
    l = wp.length(z)


    # write loss
    wp.atomic_add(loss, 0, l)
```

Example: Fused MLP Inference

Road Map

Warp Neural Networks

```python
@wp.kernel
def eval_tetrahedra(args):

    tid = wp.tid()

    ...

    act = activation[tid]

    k_mu = materials[tid, 0]
    k_lambda = materials[tid, 1]
    k_damp = materials[tid, 2]

    Ds = wp.mat33(x10, x20, x30)
    Dm = pose[tid]

    inv_rest_volume = wp.determinant(Dm) * 6.0
    rest_volume = 1.0 / inv_rest_volume

    alpha = 1.0 + k_mu / k_lambda - k_mu / (4.0 * k_lambda)

    # scale stiffness coefficients to account for area
    k_mu = k_mu * rest_volume
    k_lambda = k_lambda * rest_volume
    k_damp = k_damp * rest_volume

    # F = Xs*Xm^-1
    F = Ds * Dm
    dFdt = wp.mat33(v10, v20, v30) * Dm

    col1 = wp.vec3(F[0, 0], F[1, 0], F[2, 0])
    col2 = wp.vec3(F[0, 1], F[1, 1], F[2, 1])
    col3 = wp.vec3(F[0, 2], F[1, 2], F[2, 2])

    # ------------------------------
    # Neo-Hookean (with rest stability [Smith et al 2018])

    Ic = dot(col1, col1) + dot(col2, col2) + dot(col3, col3)

    # deviatoric part
    P = F * k_mu * (1.0 - 1.0 / (Ic + 1.0)) + dFdt * k_damp
    H = P * wp.transpose(Dm)

    f1 = wp.vec3(H[0, 0], H[1, 0], H[2, 0])
    f2 = wp.vec3(H[0, 1], H[1, 1], H[2, 1])
    f3 = wp.vec3(H[0, 2], H[1, 2], H[2, 2])

    J = wp.determinant(F)

    s = inv_rest_volume / 6.0
    dJdx1 = wp.cross(x20, x30) * s
    dJdx2 = wp.cross(x30, x10) * s
    dJdx3 = wp.cross(x10, x20) * s

    f_volume = (J - alpha + act) * k_lambda
    f_damp = (wp.dot(dJdx1, v1) + wp.dot(dJdx2, v2) + wp.dot(dJdx3, v3)) * k_damp

    f_total = f_volume + f_damp

    f1 = f1 + dJdx1 * f_total
    f2 = f2 + dJdx2 * f_total
    f3 = f3 + dJdx3 * f_total
    f0 = (f1 + f2 + f3) * (0.0 - 1.0)

    # apply forces
    wp.atomic_sub(f, i, f0)
    wp.atomic_sub(f, j, f1)
    wp.atomic_sub(f, k, f2)
    wp.atomic_sub(f, l, f3)
```

Traditional Elastic Model

Neural Network

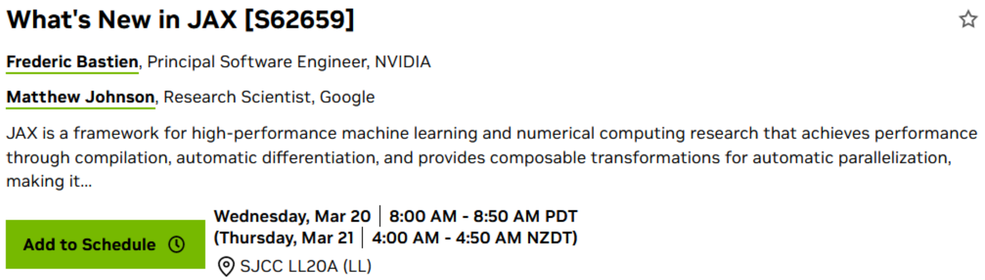
```python
@wp.kernel
def eval_tetrahedra(args):

    ...

    # setup feature vectors
    x = wp.vector(dtype=wp.float16, length=DIM_IN)

    # evaluate layers
    z = wp.linear(layer0.weights, layer0.bias, DIM_HID, x)
    z = wp.map(relu, z)
    z = wp.linear(layer1.weights, layer1.bias, DIM_OUT, z)
    z = wp.map(relu, z)

    # apply forces
    wp.atomic_sub(f, i, f0)
    wp.atomic_sub(f, j, f1)
    wp.atomic_sub(f, k, f2)
    wp.atomic_sub(f, l, f3)
```

Learned Constitutive Model

Road Map

Warp Kernels in JAX

- Wrap Warp kernels as JAX primitives with one line of code
- Invoke JAX primitive inside @jax.jit code
- Very convenient way to mix JAX and Warp
- Available now in wp.jax_experimental
- Sharding, vmap(), pmap() support coming soon
- See JAX session tomorrow!



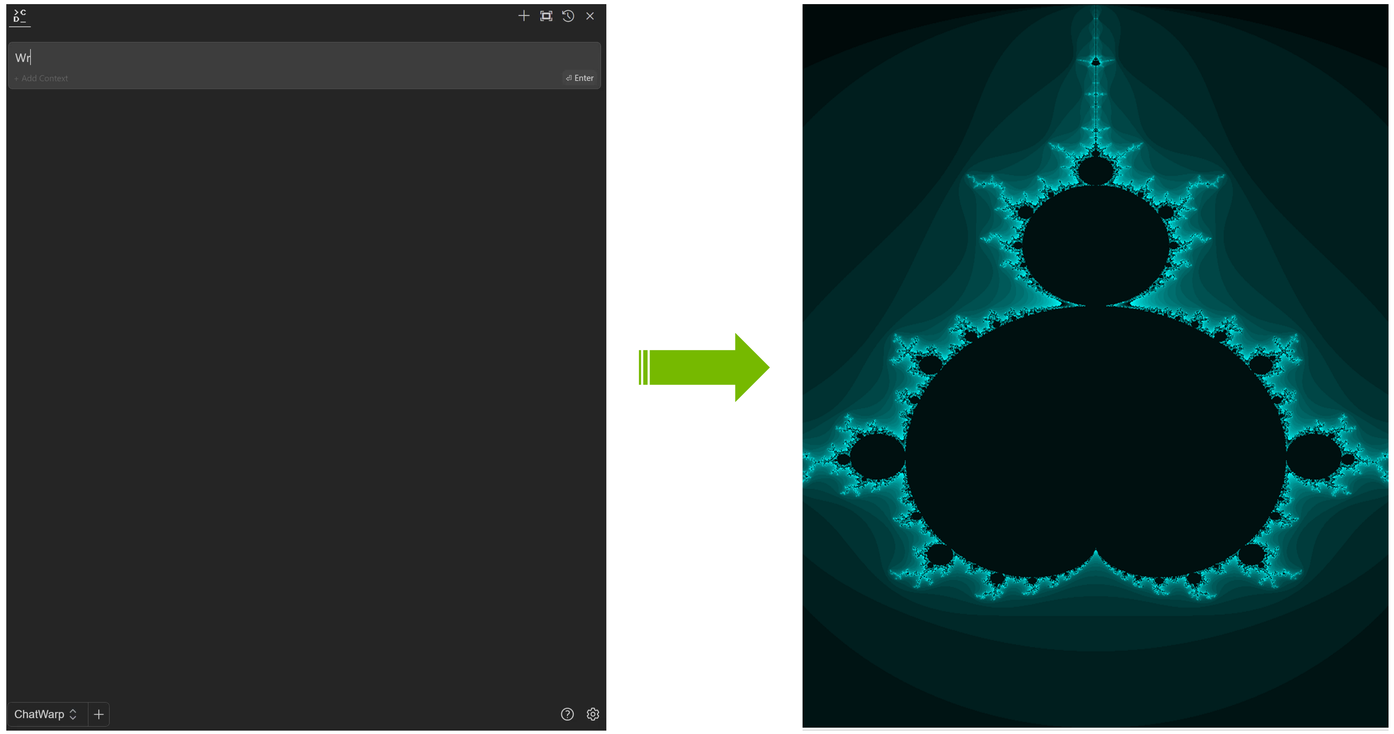

```python
import warp as wp
import jax
import jax.numpy as jp

# import Warp->JAX adapter
from warp.jax_experimental import jax_kernel

@wp.kernel
def triple_kernel(input: wp.array(dtype=float), output: wp.array(dtype=float)):
    tid = wp.tid()
    output[tid] = 3.0 * input[tid]

# create a JAX primitive from a Warp kernel
jax_triple = jax_kernel(triple_kernel)

# use the Warp kernel in a JAX jit'd function
@jax.jit
def f():
    x = jp.arange(0, 64, dtype=jp.float32)
    return jax_triple(x)

print(f())
```

Example: Warp kernel wrapped as a JAX primitive

Road Map

ChatWarp

- ChatWarp: a specialized LLM for generative Warp coding
- Combined RAG + Fine Tuning
- Prompt: Write a Warp kernel that a Mandelbrot texture
- Natural language -> loss function



Example: Warp Mandelbrot Generation using LLM

Next Steps

Installation:

```
pip install warp-lang
```

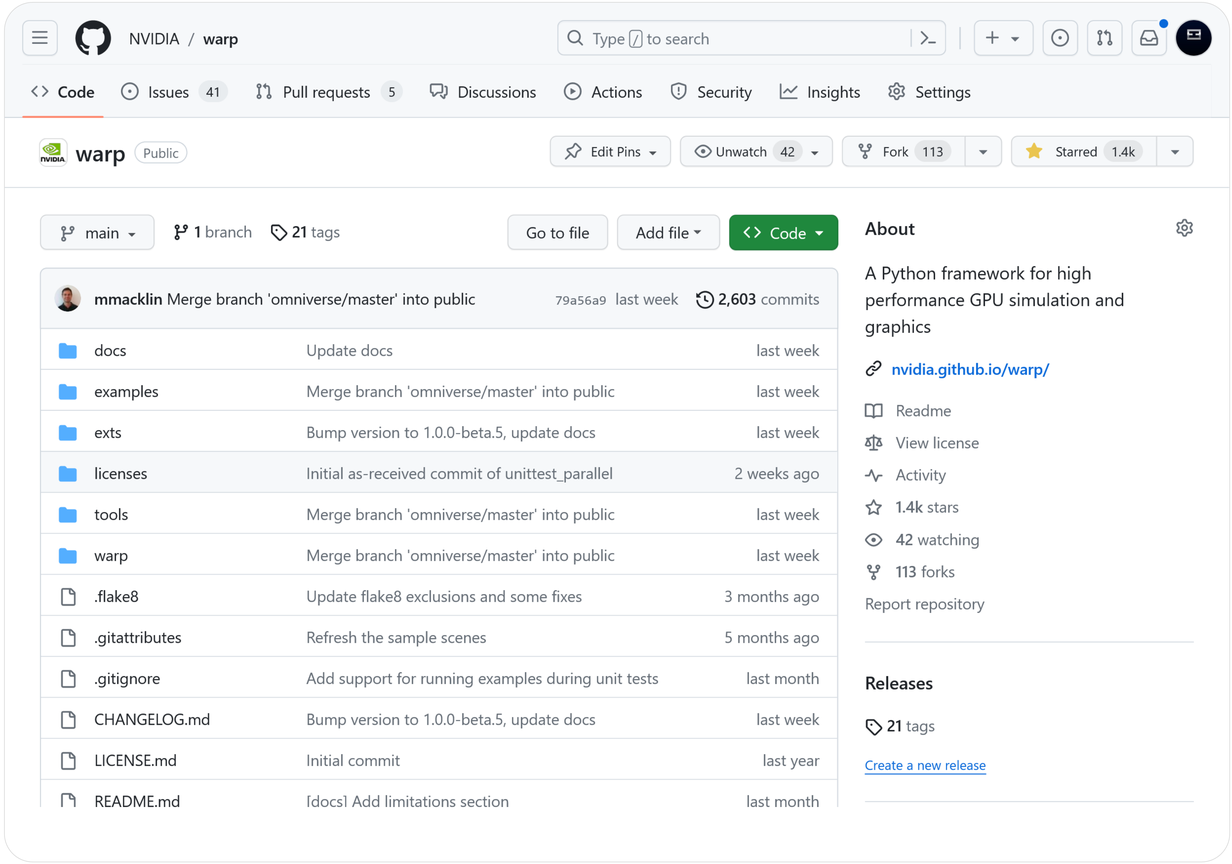
Platform support:

- Windows, Linux, macOS
- CUDA 11.5+, x86, aarch64, Jetson/Tegra

Deployment:

- Released as a standalone, open-source library
- Binary packages with monthly release cadence on PyPi
- Diverse set of example scripts
- omni.warp available in Omniverse extensions registry

- Links:
  - Repo: https://github.com/NVIDIA/warp
  - Docs: https://nvidia.github.io/warp/



Questions?