

# NVIDIA GameWorks Animation Technologies in Unreal Engine 4

Kier Storey, Viktor Makoviychuk - NVIDIA

Ori Cohen, Benn Gallagher - Epic



# Overview

- Introduce new Gameworks physics-based animation technologies
  - PhysX immediate mode
  - NvCloth
- Provide brief technical details
- Describe how these technologies are used in UE4

# PhysX Immediate Mode



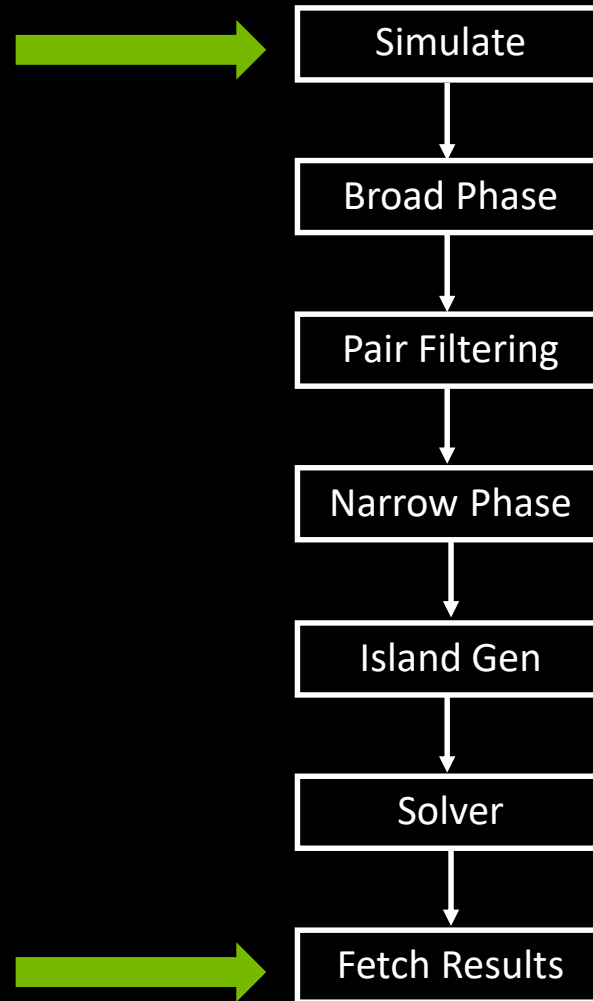
# Overview

- PhysX “Retained Mode”
- Basic Pipeline Stages
- What’s good about “Retained Mode”?
- What’s not-so-good about “Retained Mode”?
- What is “Immediate Mode”?
- How and when to use Immediate Mode

# PhysX “Retained Mode”

- Create PhysX scene and CPU dispatcher (thread pool)
- Create PhysX actors
- Create and associate PhysX shapes with actors
- Populate scene with actors
- Simulate scene, providing delta time-step
- Call fetchResults to gather simulation results

# PhysX “Retained Mode”



# What's good about PhysX “Retained Mode”?

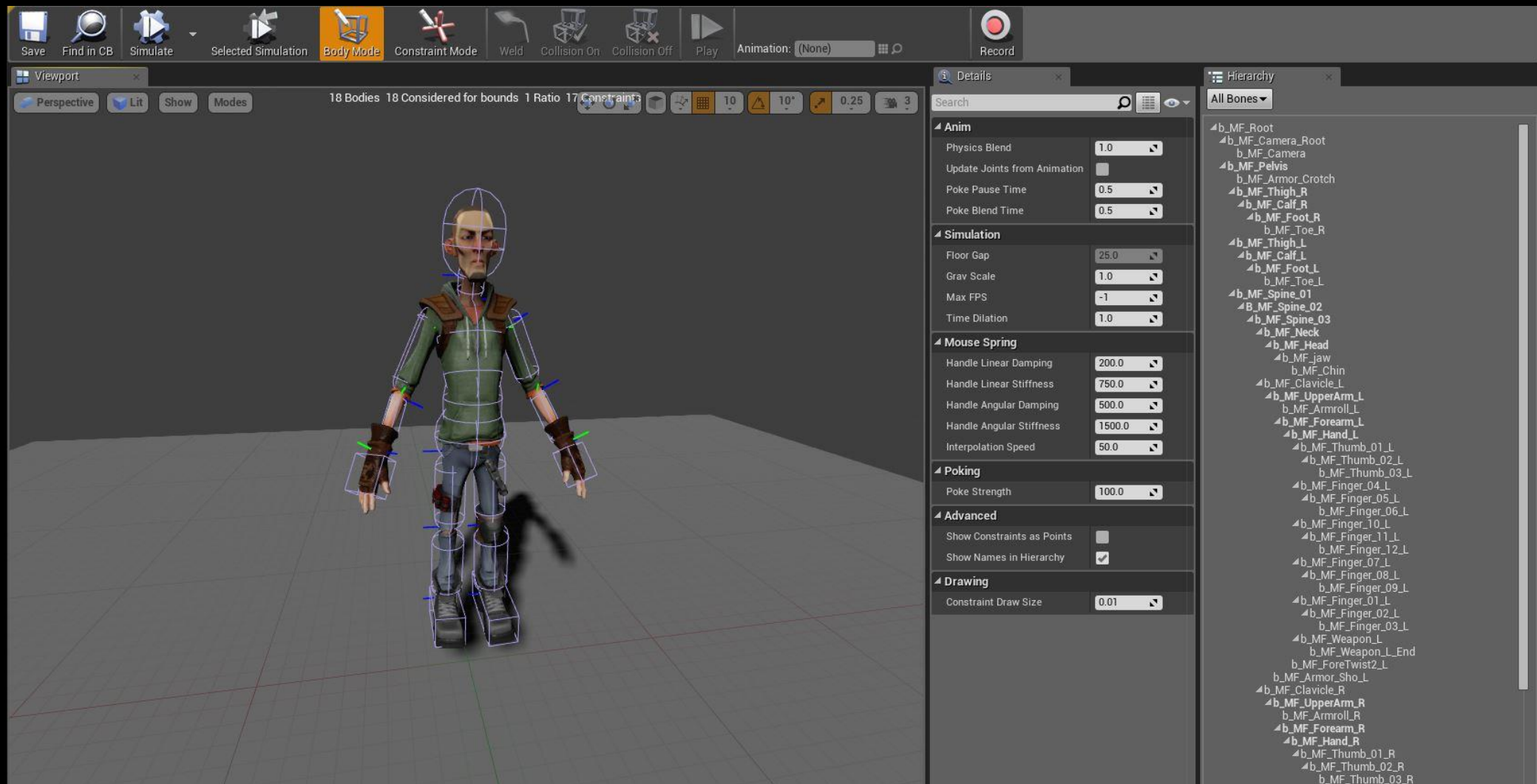
- Clean, easy-to-use API
- Efficient way to simulate lots of rigid bodies interacting with each-other
- Automatic multi-threading
- Automatically identifies subset of scene that needs simulating
- Leverages frame-frame coherence to improve performance and simulation stability
- Supports rich interactions between rigid bodies, clothing and particles.

# What's not-so-good about “Retained Mode”?

- Simulation has last word
  - Deactivating an actor does not mean it won't get simulated!
  - Deactivating actors that are re-activated in the simulation hurts performance.
- Filtering actor interactions isn't free
  - Broad phase overhead - it still finds all pairs
  - When a pair is found, filtering performed via callbacks
- Hard to determine cost of simulating a given body or group of bodies.
  - We can estimate (simplify scene, re-measure)
  - Or we could use multiple scenes but this adds per-scene overhead



# Common animation use-case



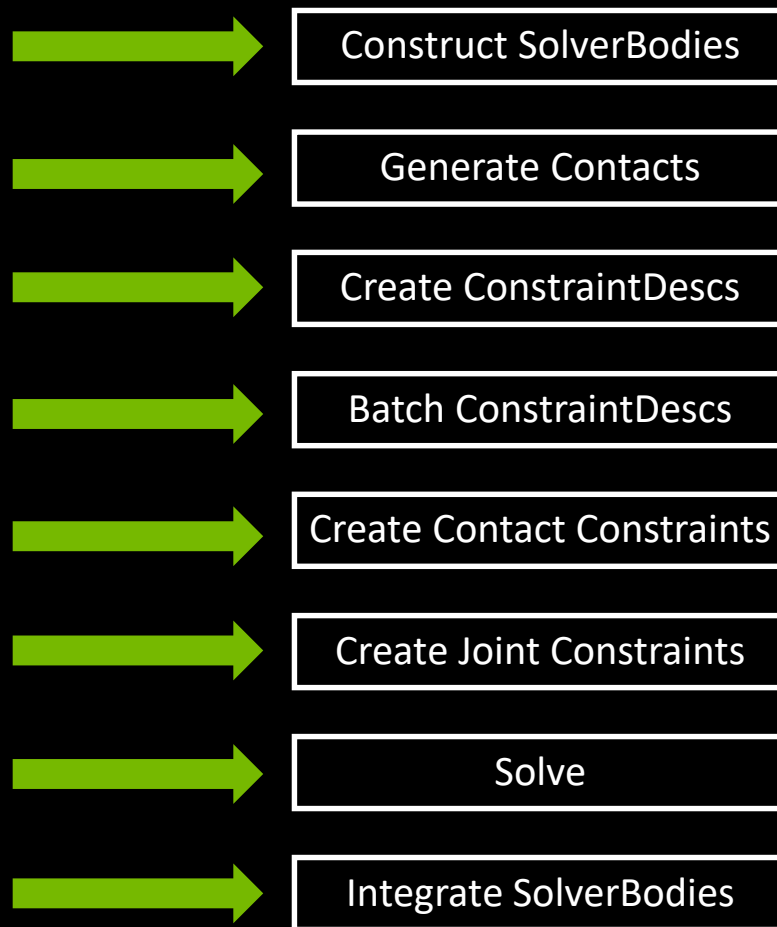
# Common Complaints

- Pushing all actors into one scene in an already complex world makes broad phase more expensive.
  - Can be alleviated by using PxAggregate feature but YMMV
- Island management for characters is wasted computation
  - Could be quite expensive in PhysX 3.3. Significantly less so in 3.4.
- Can't easily cull distant simulations
  - Can work around this but it requires hoop jumping (remove actors, disable simulation on bodies etc.)
- Hard to identify the cost of simulating a given character
- Scene-based simulation uses task chains. Can be fiddly to incorporate into animation pipeline
  - Commonly forced to split animation into pre-sim and post-sim stages.

# What is Immediate Mode?

- Exposes limited low-level functionality
  - Narrow phase/contact generation, Constraint prep, constraint solver and integration
- User responsible for:
  - Rigid body and shape management
  - Broad phase, filtering and pair management
  - Memory management, any threading etc.
- Intended use cases for immediate mode
  - Small component-based simulations
    - Secondary motion on characters, vehicles etc.

# Immediate mode stages



# IMMEDIATE MODE PHYSICS

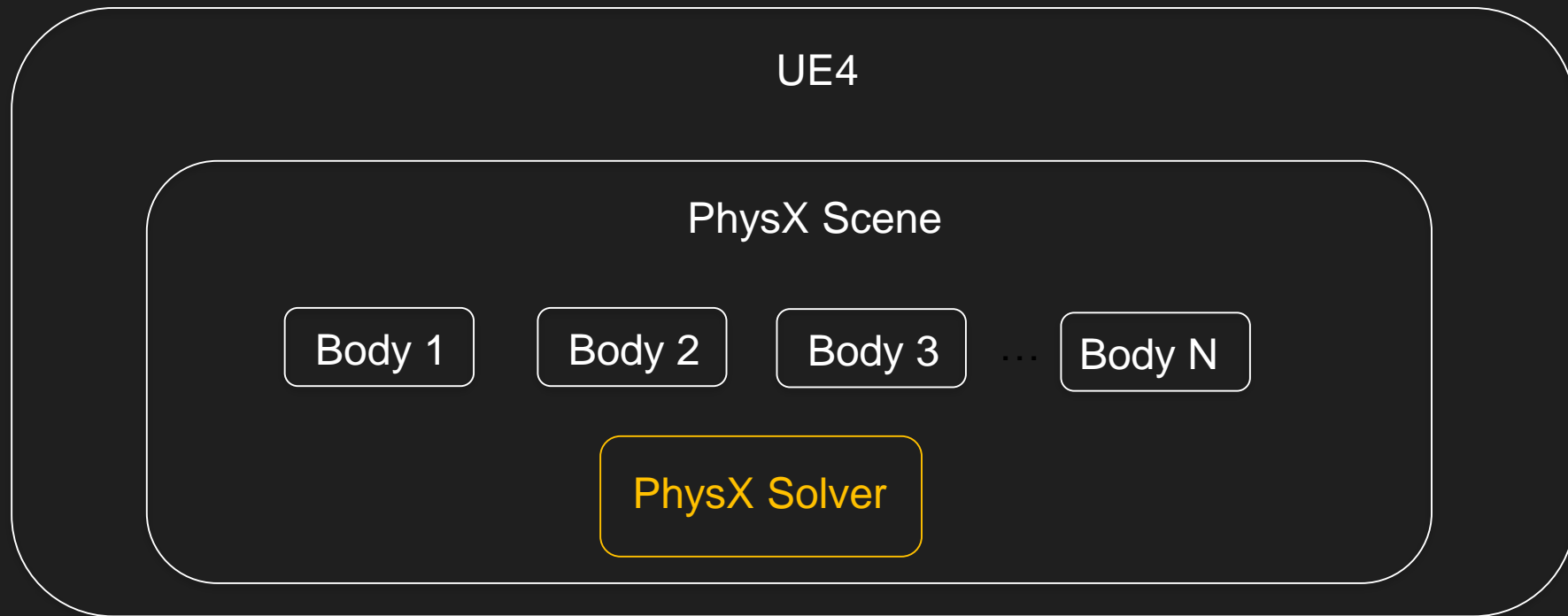
Ori Cohen

Senior Programmer

 @oricohen139

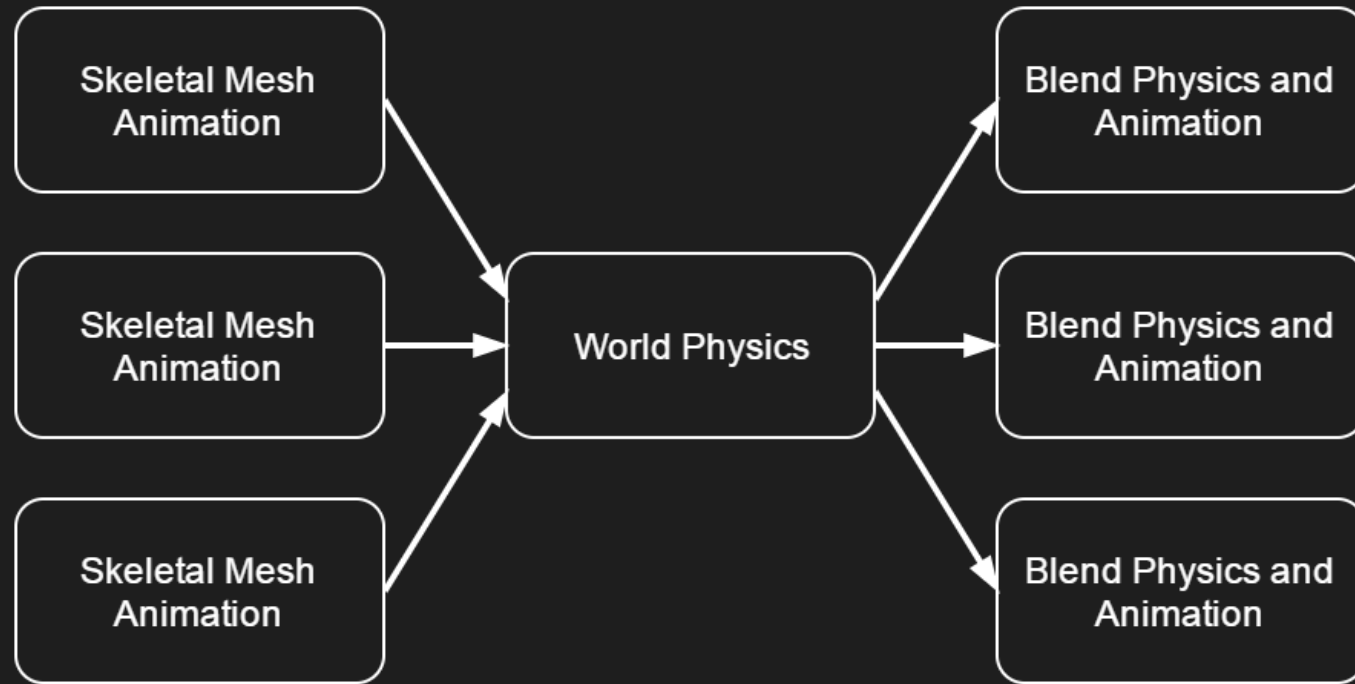
# IMMEDIATE MODE PHYSICS

## Classic Physics



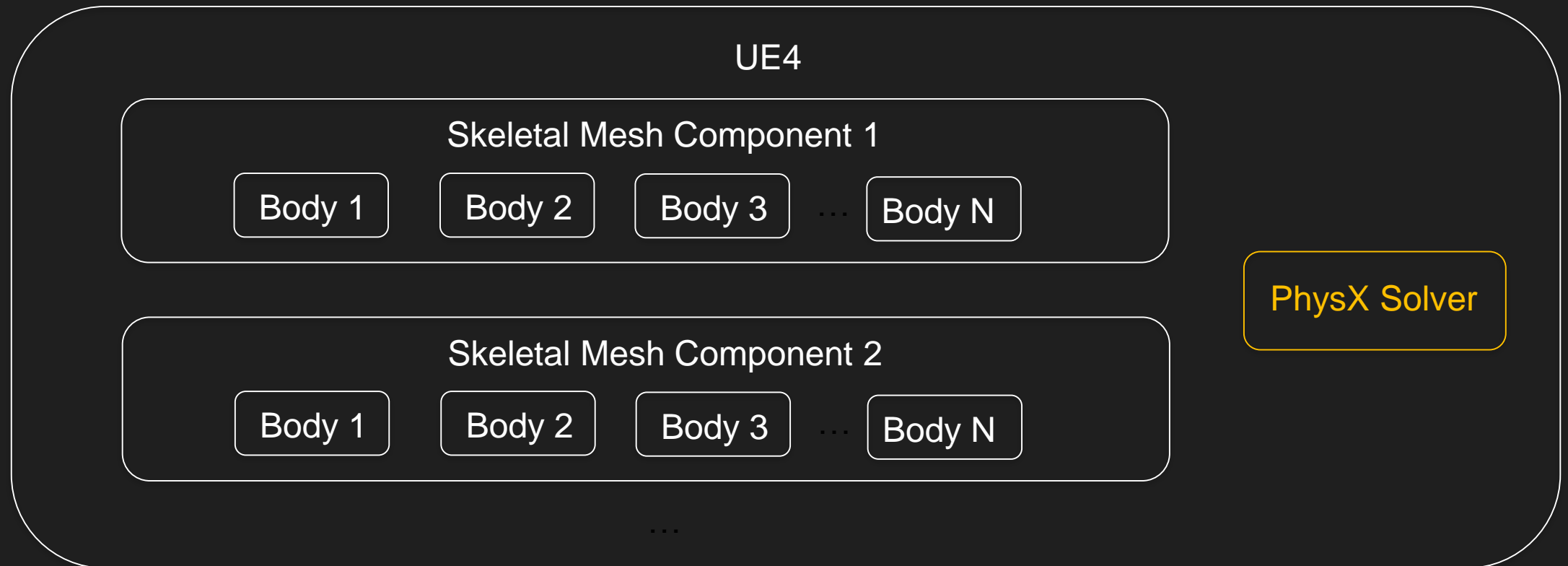
# IMMEDIATE MODE PHYSICS

## Expensive Sync Points



# IMMEDIATE MODE PHYSICS

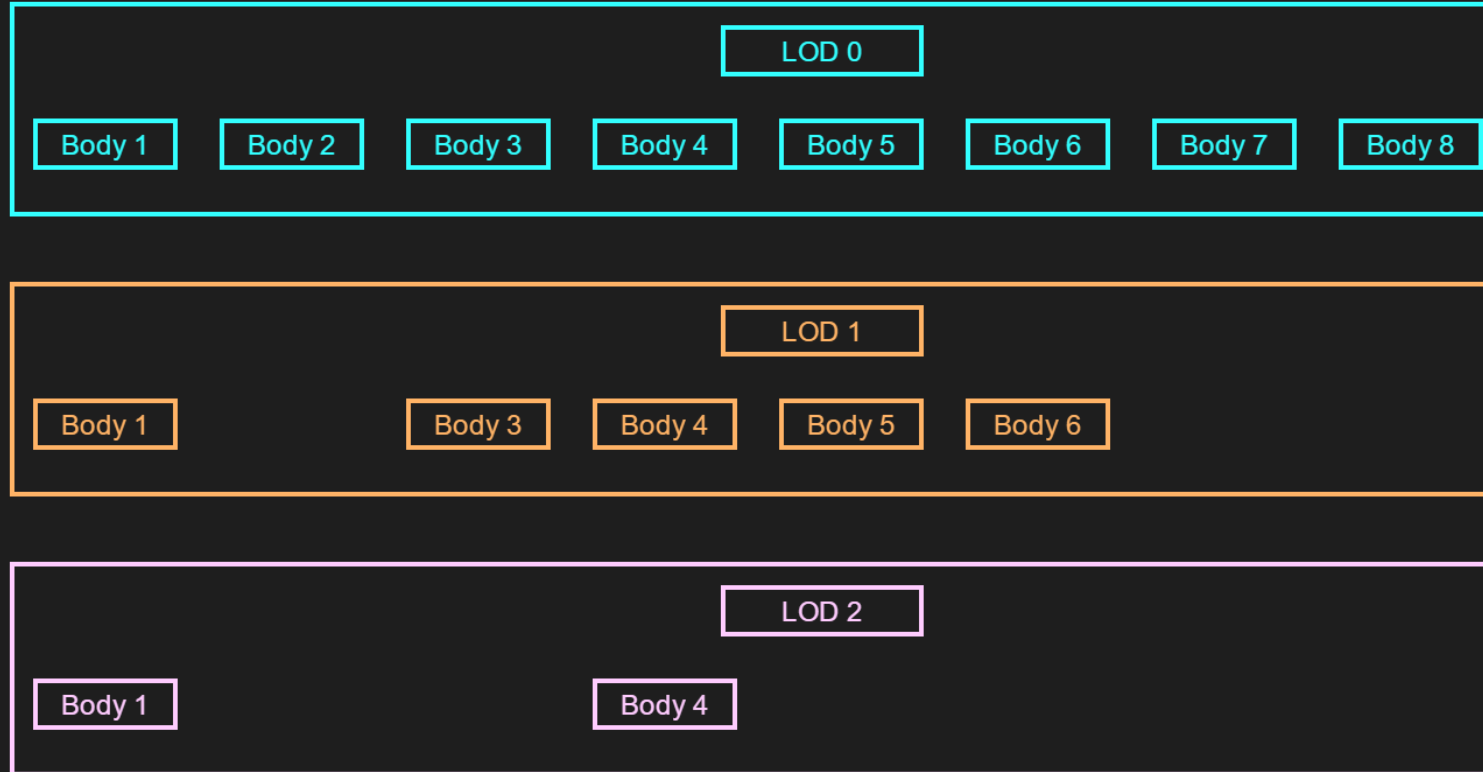
## Immediate Physics





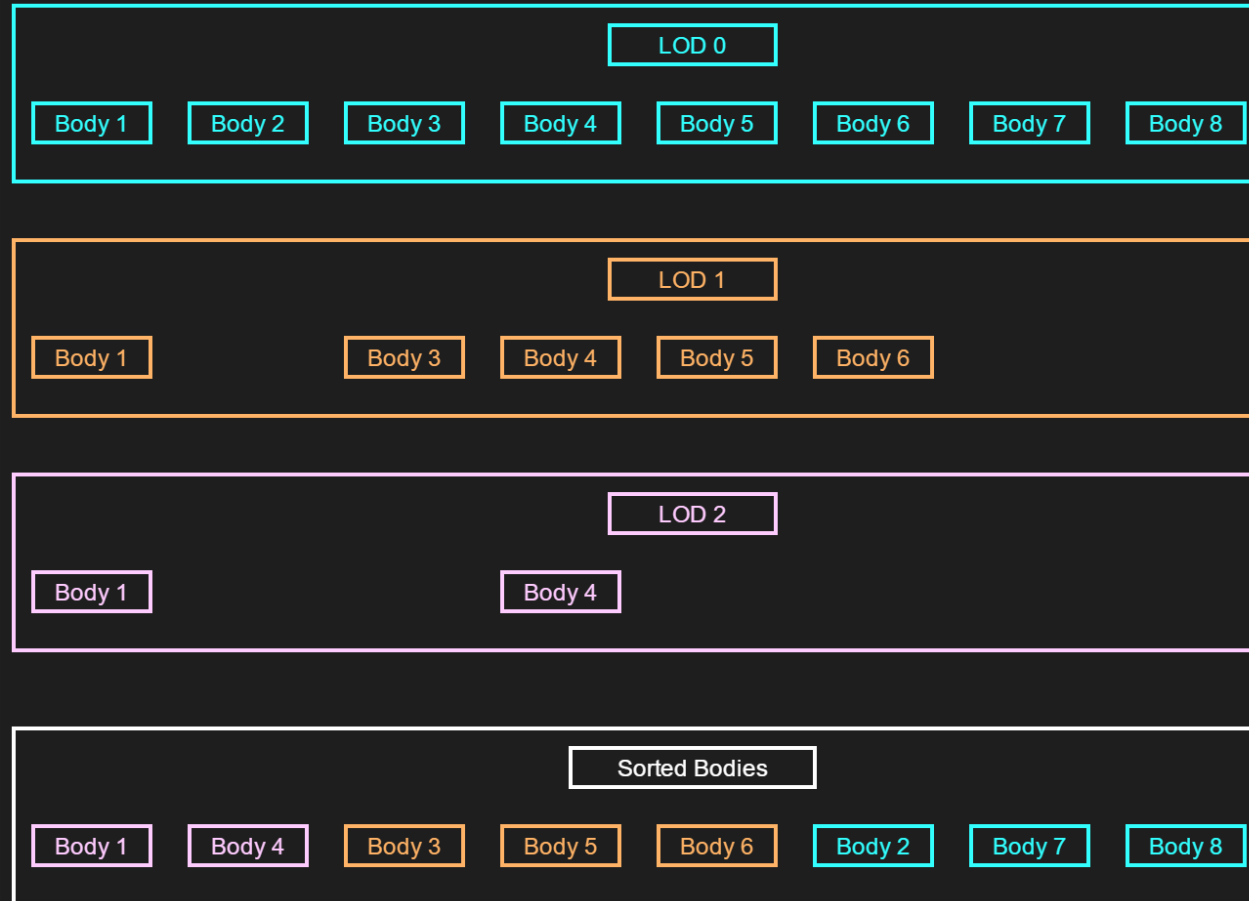
# IMMEDIATE MODE PHYSICS

LOD control



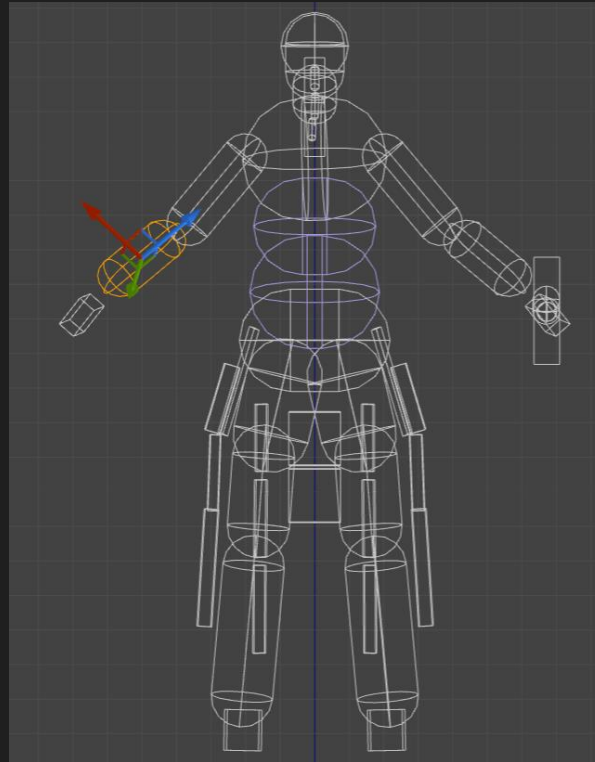
# IMMEDIATE MODE PHYSICS

## LOD control



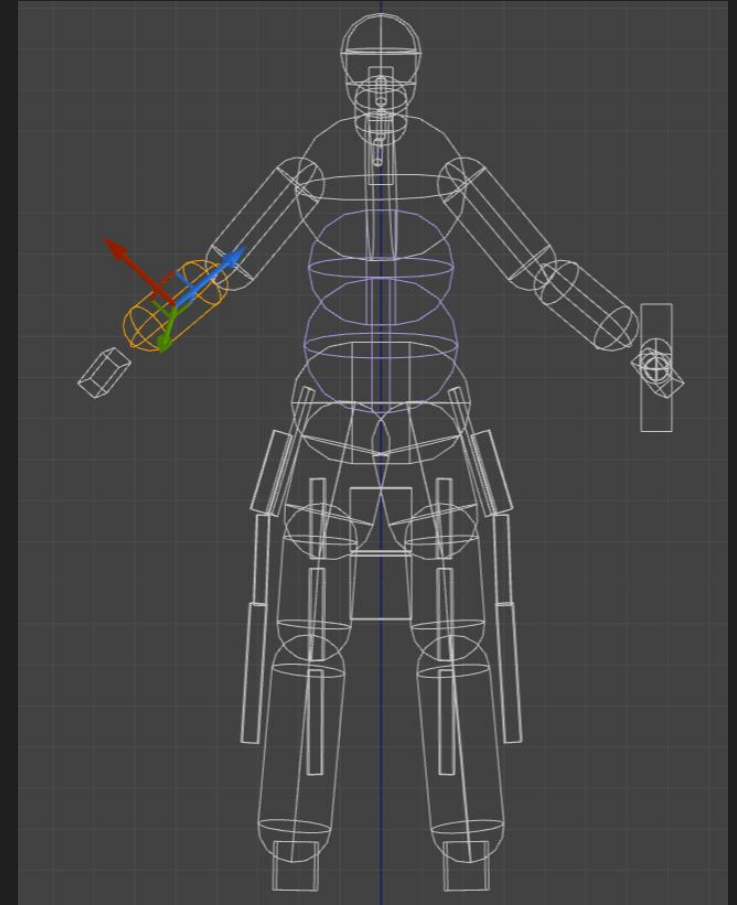
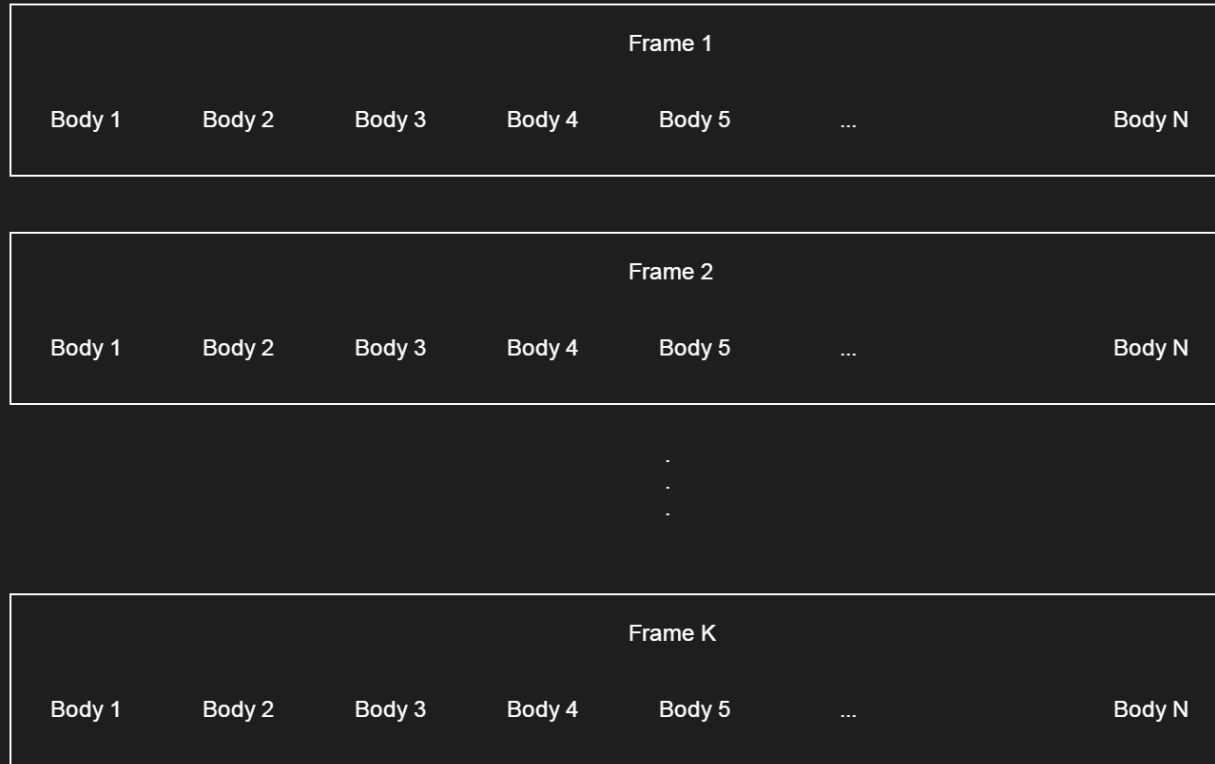
# IMMEDIATE MODE PHYSICS

Iteration Cache (cheap filtering)



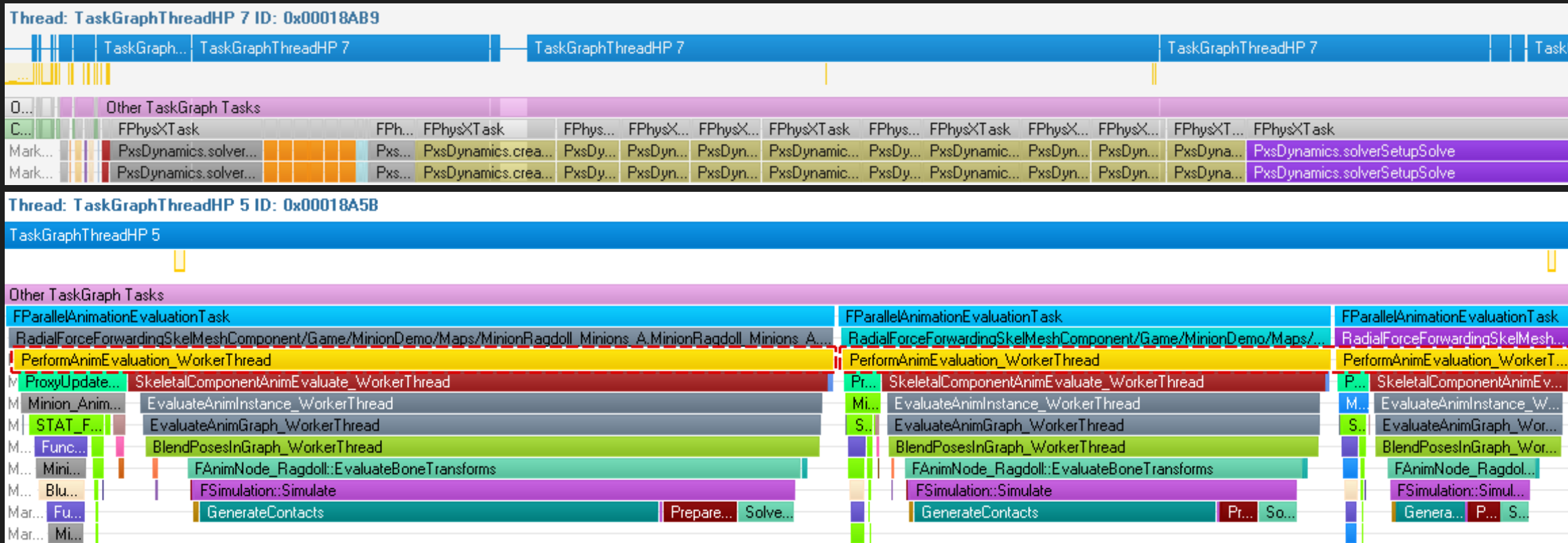
# IMMEDIATE MODE PHYSICS

## Iteration Cache (cheap filtering)



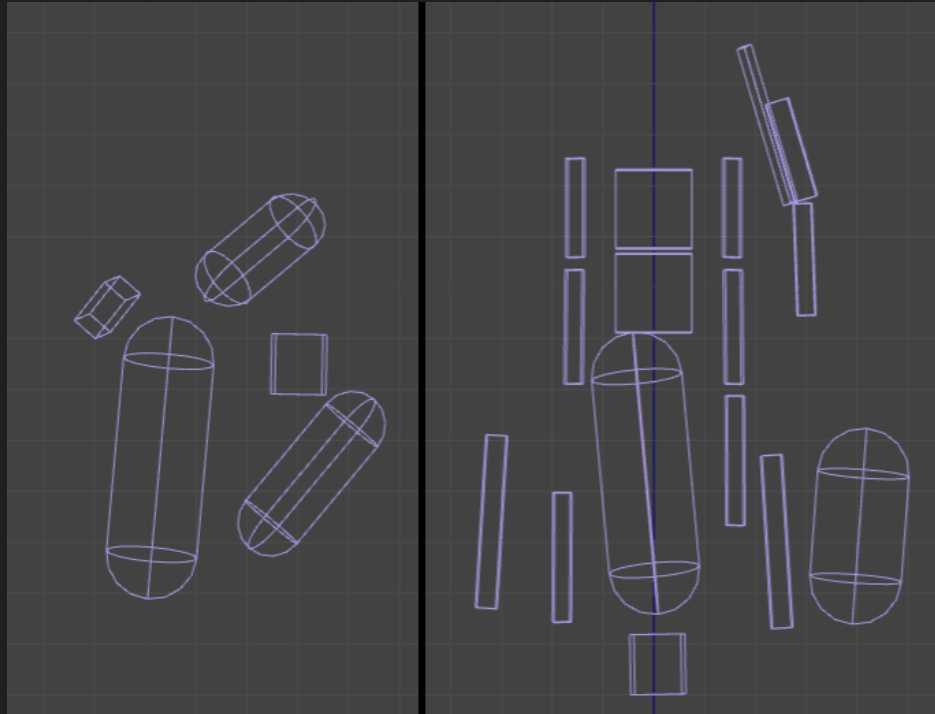
# IMMEDIATE MODE PHYSICS

## Control Over Scheduling



# IMMEDIATE MODE PHYSICS

## Control Over Memory



character 1

character 2

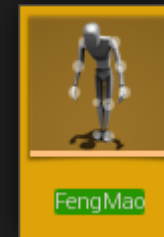
# IMMEDIATE MODE PHYSICS

Isn't this just AnimDynamics?



# IMMEDIATE MODE PHYSICS

## Immediate Mode workflow







UNREAL  
ENGINE

# IMMEDIATE MODE PHYSICS

## Performance of individual characters

Budget [character_lod0] - MergedTaskGraphThreads			
	CallCount	InclusiveAvg	InclusiveMax
Cloth Total	1	0.09 ms	0.15 ms
PerformAnimEvaluation_WorkerThread	1	0.24 ms	0.39 ms
FSimulation.Simulate	1	0.13 ms	0.25 ms
Anim Dynamics Physics Update			
Total (of 1.00 ms)		0.34 ms	0.54 ms
Budget [character_lod0] - GameThread			
	CallCount	InclusiveAvg	InclusiveMax
AnimGameThreadTime	3	0.07 ms	0.13 ms
BlueprintUpdateAnimation	1	0.00 ms	0.00 ms
Total (of 0.25 ms)		0.07 ms	0.13 ms

# IMMEDIATE MODE PHYSICS

x2 speed up over AnimDynamics





UNREAL  
ENGINE

# NvCloth

Viktor Makoviychuk



# Simulation challenges

## Very Hard Task!

- Tight time budget
- Should be no stretching and penetrations even with large timesteps
- Zoo of platforms
- Large accelerations and velocities of animated characters

# Limitations of the current solutions

- APEX Clothing and PhysX Cloth use different copies of the same embedded solver
  - Additional work and harder to support, potential source of errors
  - Any bug fix or new feature should be integrated to APEX and PhysX
- General solutions
  - Too complex integration
  - Overhead could affect performance

# NvCloth

Immediate mode low-level cloth solver with some additional functionality

- Tiny and easy to integrate
- High performance due to direct and tight integration without overhead introduced by serialization, managing rendering data and other layers
- More control on the simulation pipeline for our customers

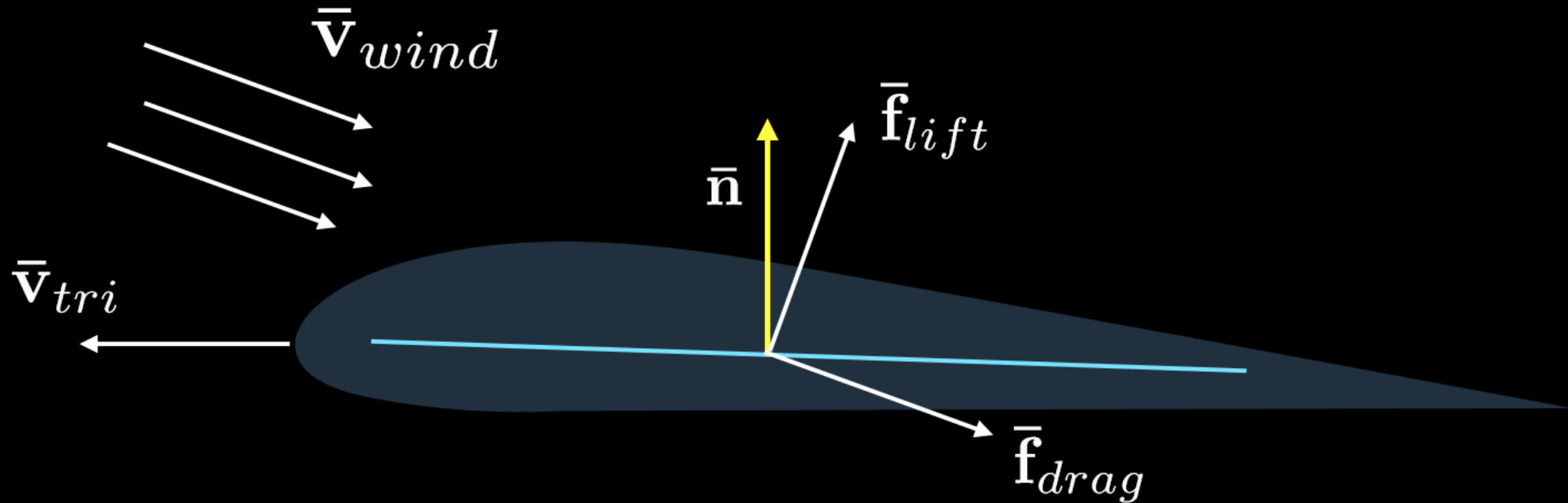


# Features

- All of the features supported by APEX/PhysX low-level cloth solver
- New air resistance model
- DX11 compute path in addition to the CPU and GPU CUDA solvers
  - In beta testing stage, not all of the features are supported yet

# Air resistance model

- Treat each triangle as thin airfoil to generate lift + drag forces



# Code example: initialization

```
#include <NvCloth/Factory.h>
#include <NvCloth/Fabric.h>
#include <NvCloth/Solver.h>
#include <NvCloth/Cloth.h>

#include <NvClothExt/ClothFabricCooker.h>

//Initialization only once per application
nv::cloth::InitializeNvCloth(myAllocator, myErrorCallback, myAssertHandler,
nullptr);

nv::cloth::Factory* factory = NvClothCreateFactoryCPU();
nv::cloth::Solver* solver = factory->createSolver();
```

# Code examples: creating a cloth

```
MyClothMeshClass myMesh = ...;
nv::cloth::ClothMeshDesc meshDesc = myMesh.getClothMesh();

nv::cloth::Vector<int32_t>::Type phaseTypeInfo;
nv::cloth::Fabric* fabric =
    NvClothCookFabricFromMesh(factory, meshDesc, physx::PxVec3(0.0f, -9.8f, 0.0f),
                              &phaseTypeInfo, false);

std::vector<physx::PxVec4> particlesCopy;
particlesCopy.resize(meshDesc.points.count);
for (int i = 0; i < meshDesc.points.count; i++)
    particlesCopy[i] = physx::PxVec4(myMesh.points[i], 1.0f);

//Create the cloth from the initial positions/masses and the fabric
nv::cloth::Cloth* cloth = factory->createCloth(
    nv::cloth::Range<physx::PxVec4>(&particlesCopy.front(),
    &particlesCopy.back() + 1),
    *fabric);
```

# Code examples: setting it up

```
// Setting cloth properties
cloth->setGravity(physics::PxVec3(0.0f, -9.8f, 0.0f));
cloth->setDragCoefficient(0.5f);
cloth->setLiftCoefficient(0.6f);

std::vector<nv::cloth::PhaseConfig> phases(fabric->getNumPhases());
for (int i = 0; i < (int)phases.size(); i++)
{
    phases[i].mPhaseIndex = i; // Set index to the corresponding set

    phases[i].mStiffness = 1.0f;
    phases[i].mStiffnessMultiplier = 1.0f;
    phases[i].mCompressionLimit = 1.0f;
    phases[i].mStretchLimit = 1.0f;
}
cloth->setPhaseConfig(CreateRange(phases));

//Add the cloth to the solver for simulation
solver->addCloth(cloth);
```

# Code examples: update loop

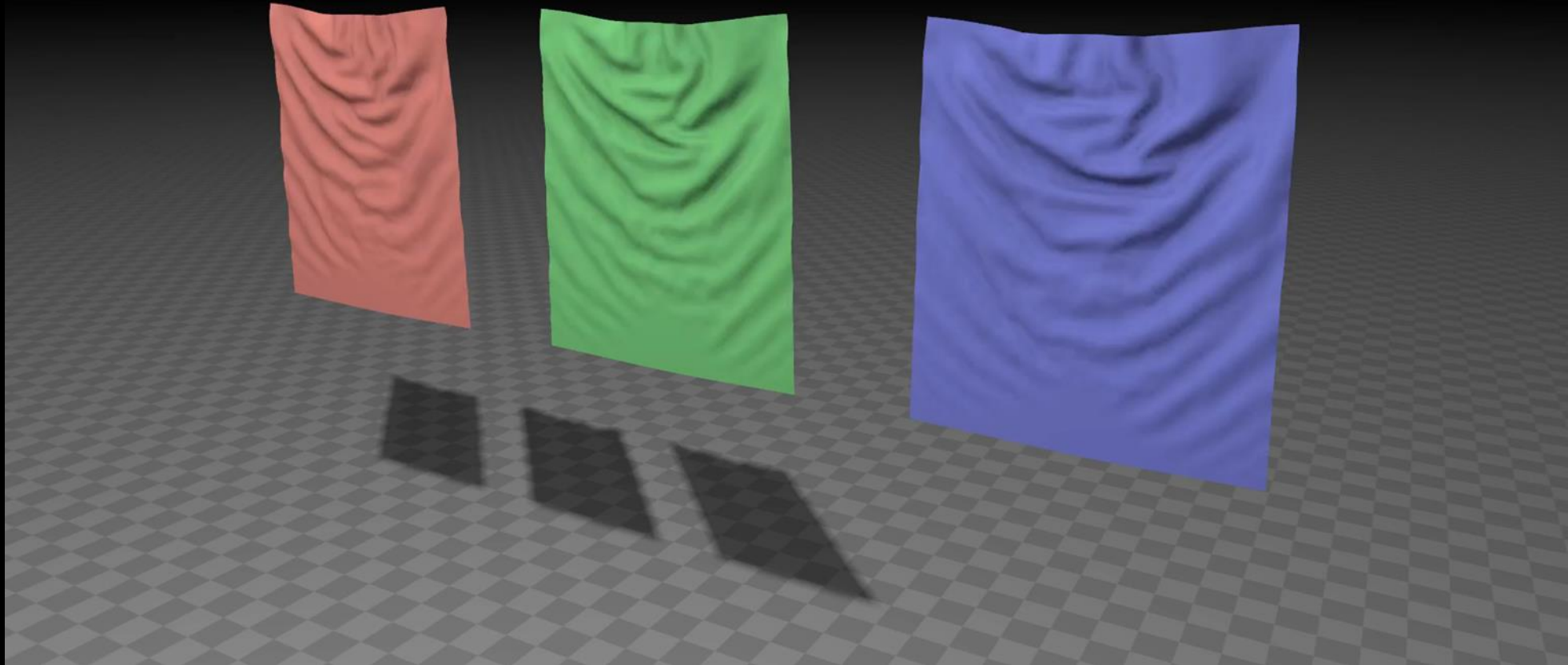
```
// Simulation loop
for (int i = 0; i < 600; i++)
{
    solver->beginSimulation(1.0f / 60.0f);
    for (int i = 0; i < solver->getSimulationChunkCount(); i++)
        solver->simulateChunk(i);
    solver->endSimulation();

    cloth->setWindVelocity(GetGlobalWind());
}

//Cleanup
solver->removeCloth(cloth);

//Delete all the created objects
NV_CLOTH_DELETE(cloth);
fabric->decRefCount();
NV_CLOTH_DELETE(solver);
NvClothDestroyFactory(factory);
```

# Result



# Release

Github link: <https://github.com/NVIDIAGameWorks/NvCloth>



# UE4 integration




# Questions?

[vmakoviychuk@nvidia.com](mailto:vmakoviychuk@nvidia.com)

[mtamis@nvidia.com](mailto:mtamis@nvidia.com)

# CLOTHING PIPELINE

Benn Gallagher  
Senior Programmer  
 @Exelcior

# CLOTHING PIPELINE - MOTIVATION

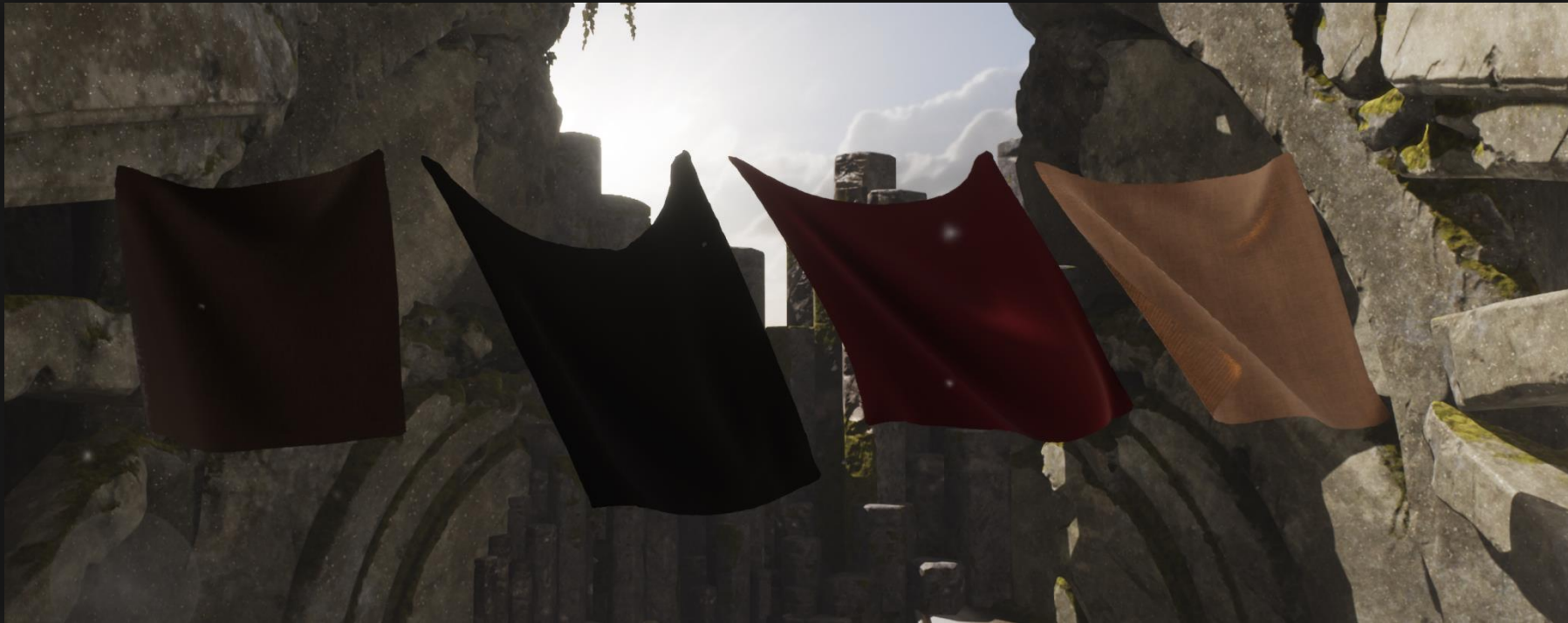
APEX wasn't an ideal solution as UE4 grew





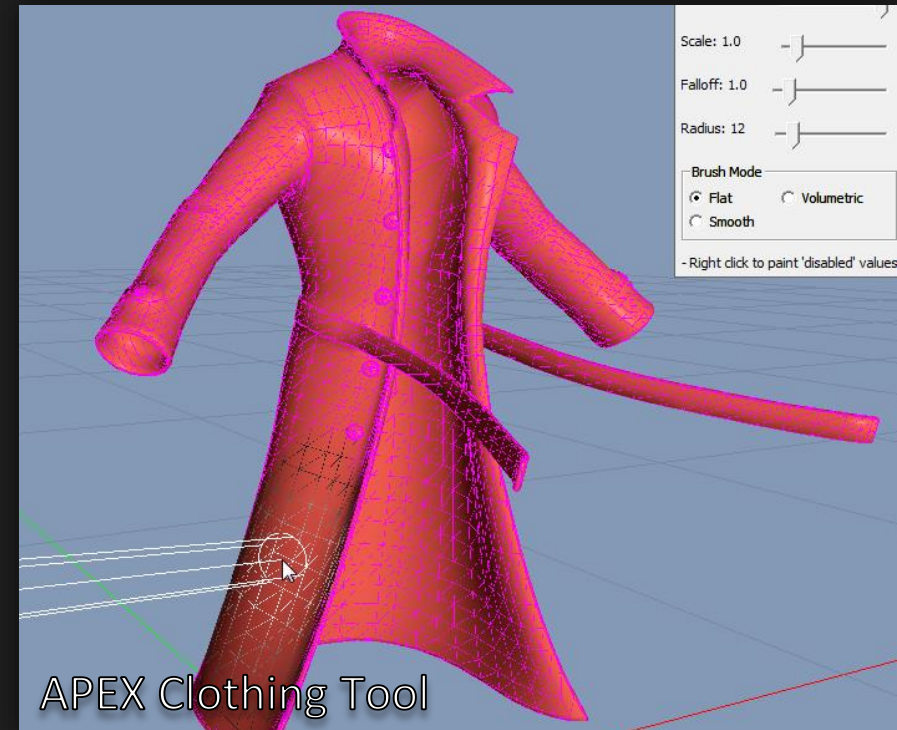
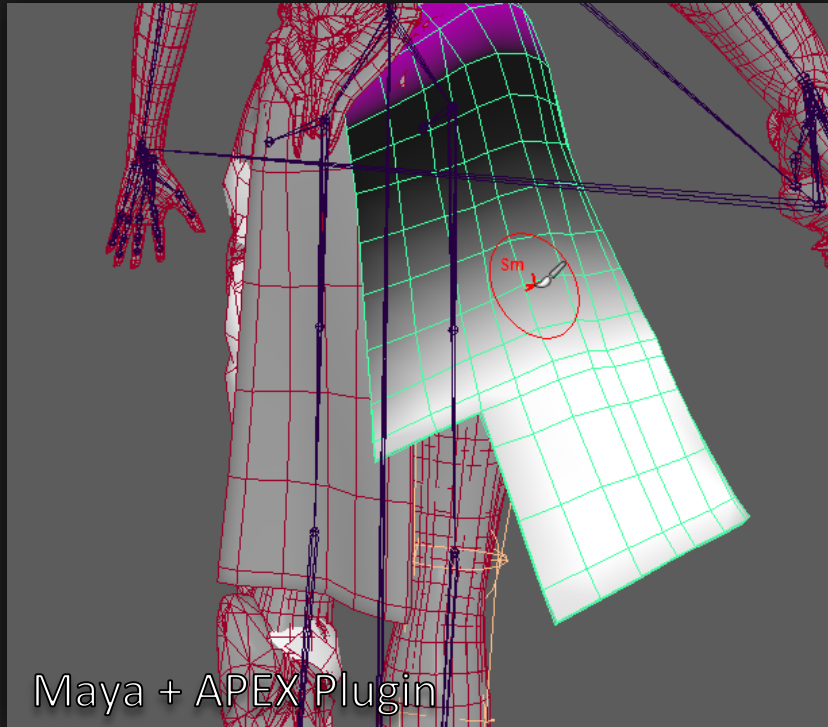
# CLOTHING PIPELINE - MOTIVATION

Replaced with lower level solver & clothing framework



# CLOTHING PIPELINE - MOTIVATION

Content creation requires external tools

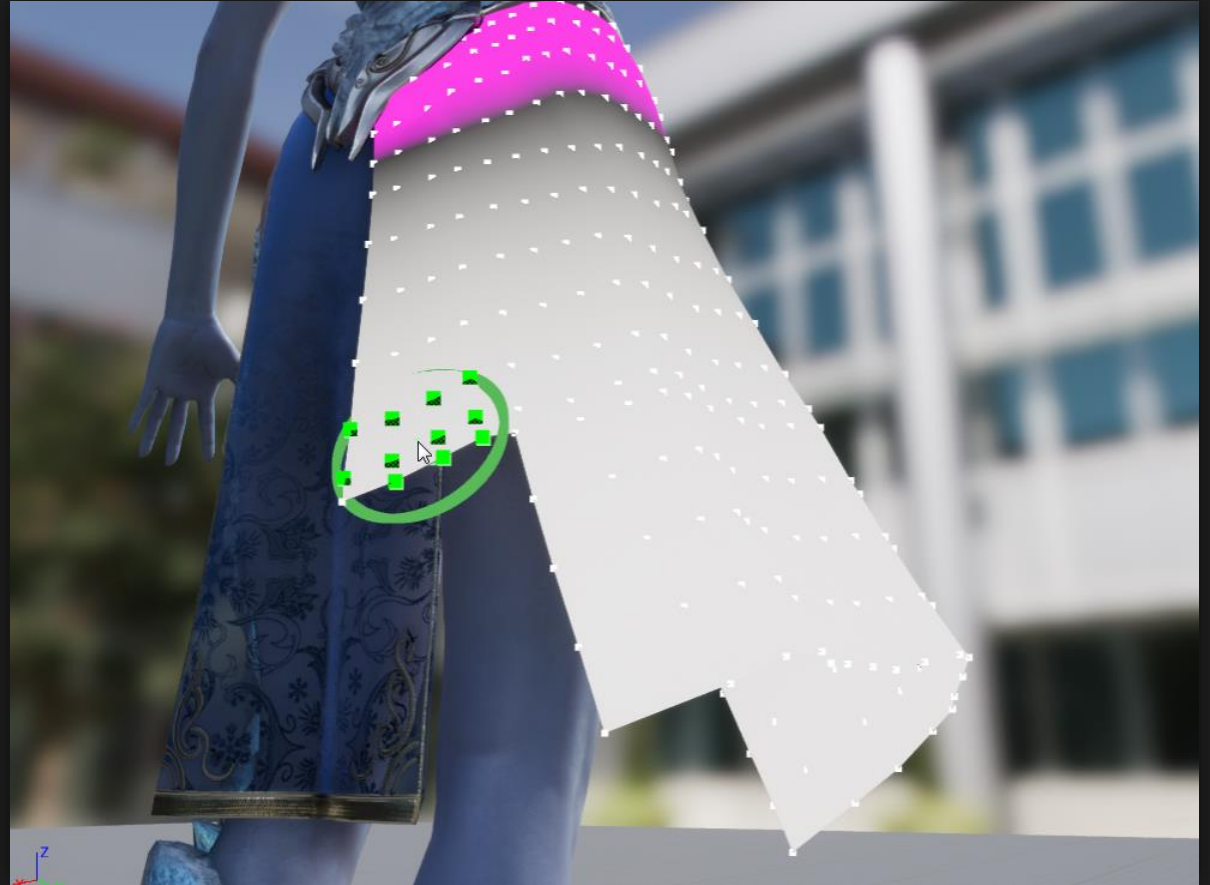


# CLOTHING PIPELINE - TOOLS

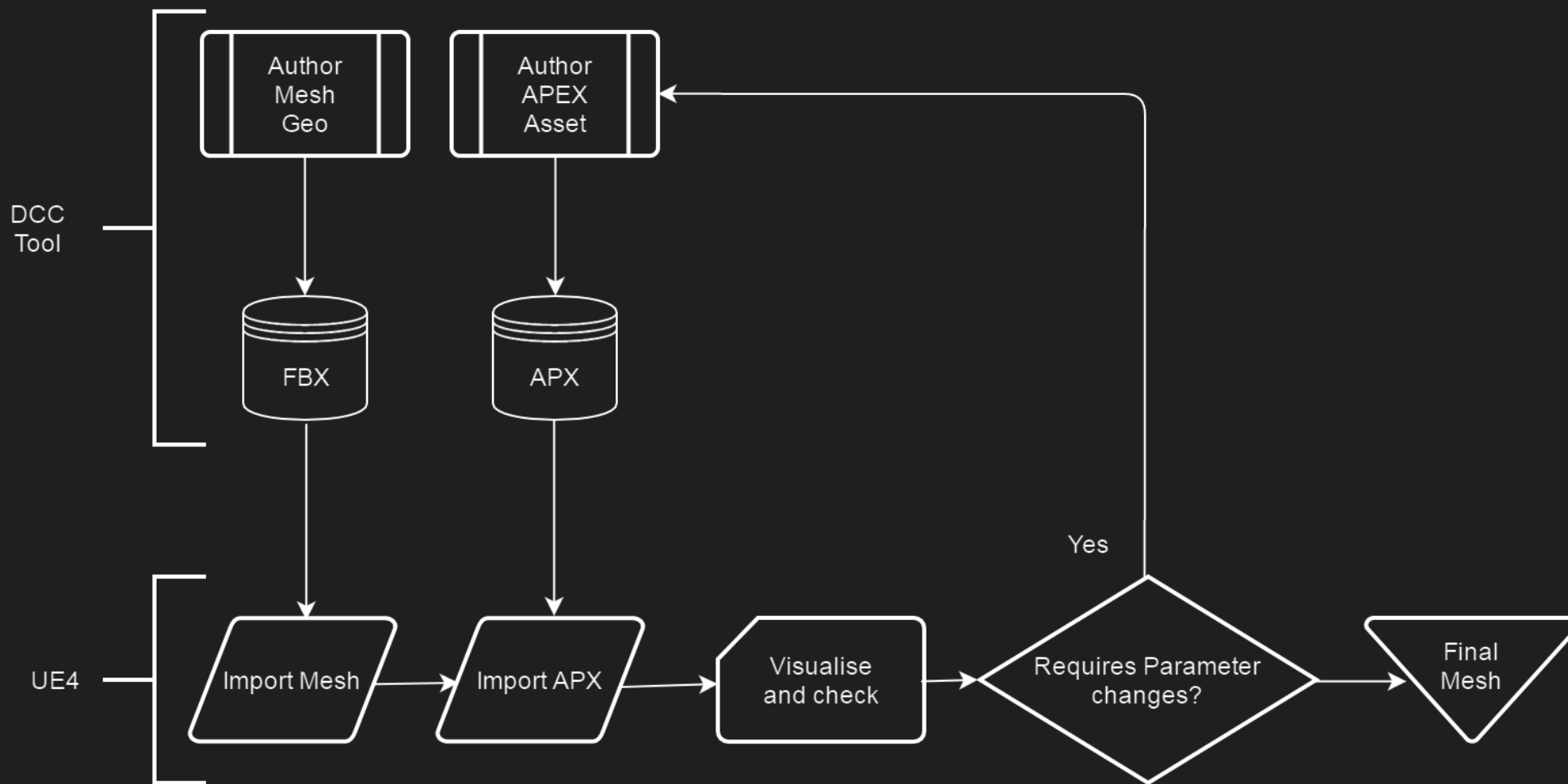
Aiming for fast iteration

Familiarity to other tools

Extensible so we can expand  
on the toolset

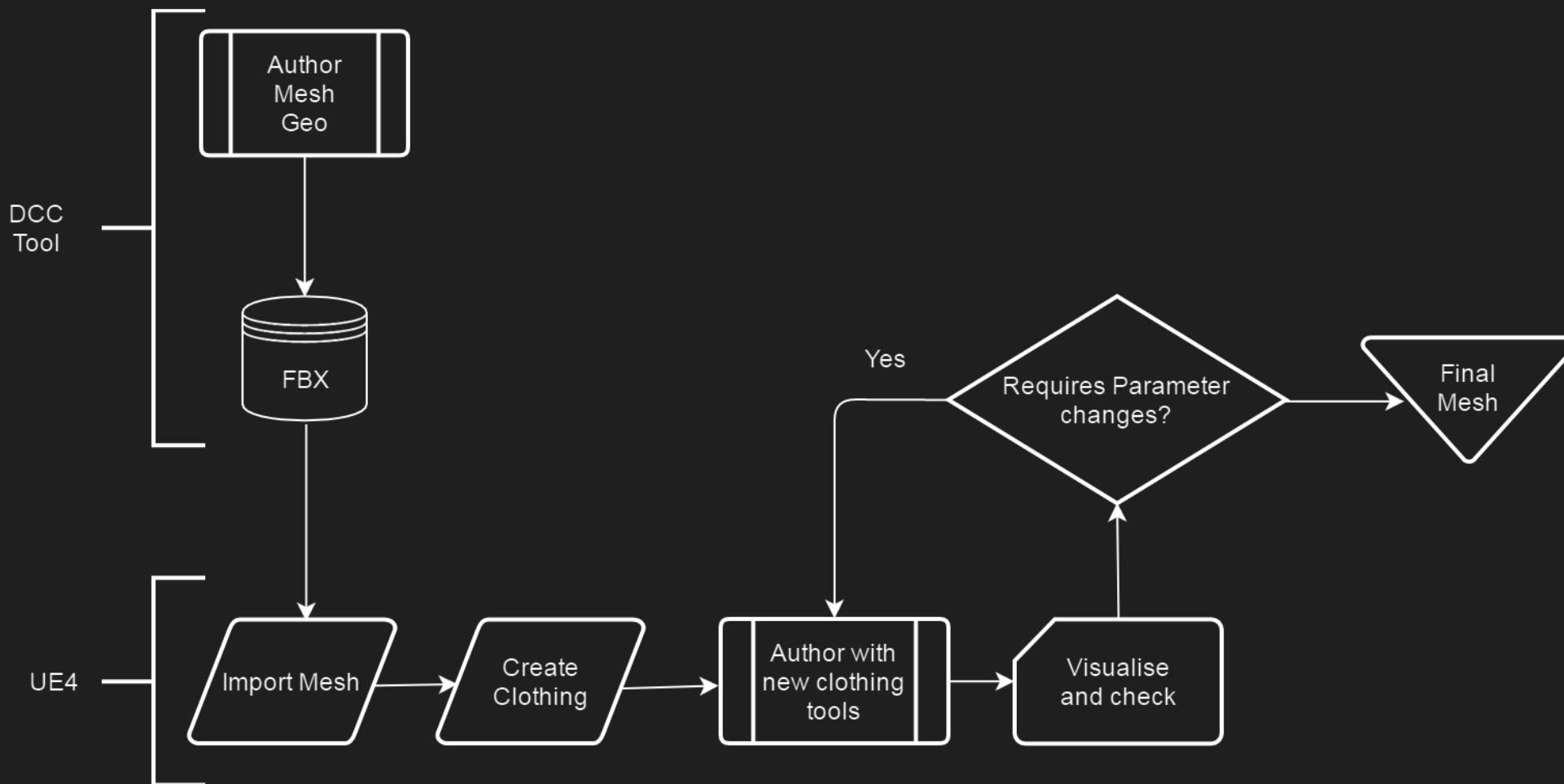


# CLOTHING PIPELINE – OLD PIPELINE





# CLOTHING PIPELINE – NEW PIPELINE





UNREAL  
ENGINE