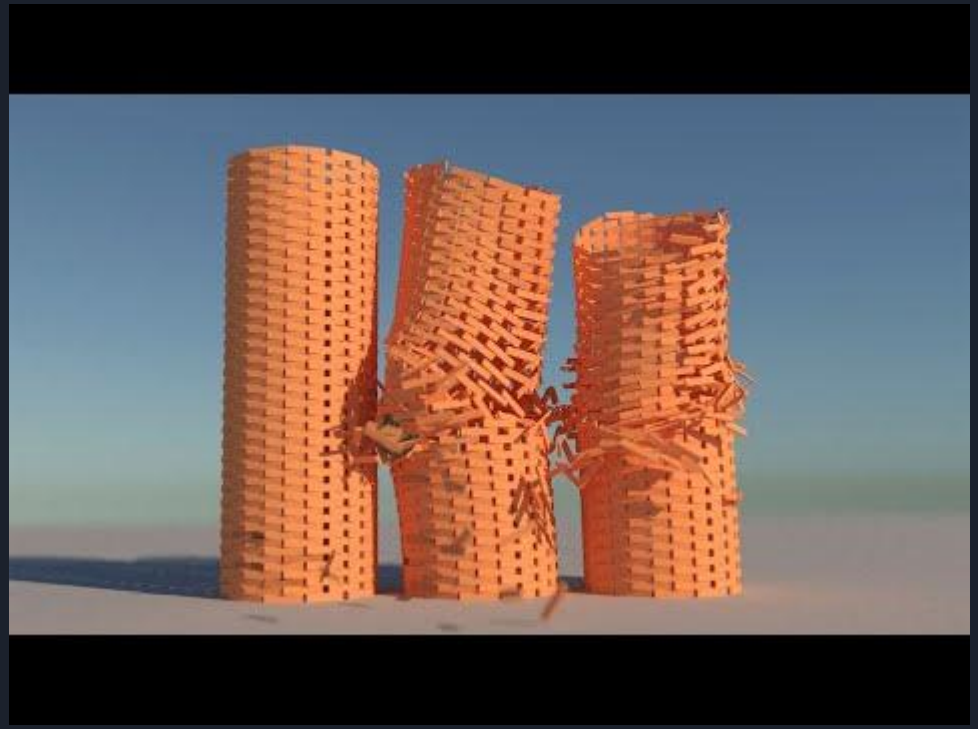
Pegase Engine

Simulating the real world

Aurèle MAHEO

Physic Engine



Physic Engine

Pegase Engine

- What ?
  - Physic engine (wants to be at least)
  - Aims at simulating a dynamic world
  - Combined with a graphic engine
- Why ?
  - Practice C++ language
  - Offer the gamers the most realistic game experience
  - Also applies to industry: civil engineering, aerospace, ship building, etc
  - Getting (one day) knowledgeable about:
    - Dynamics
    - Kinematics
    - Fluids dynamics
      - Hydrodynamics
      - Aerodynamics
  - Deal with lockdown
  - For fun ?

Pegase Engine

- Primary goals
  - Implement everything that makes what we call a physic engine
    - Core purpose : create objects and enable realistic movements and collisions
    - Applies laws of dynamics and kinematics
    - Start from simple objects to complex structures
    - Handle collisions
    - Simulate destructions of structures
    - Implement existing algorithms
    - Ambitious ? Really ??

Pegase Engine

- Secondary goals
  - Develop realistic graphics (light, textures, particles)
  - Build a research platform to experiment cool things
    - Design novel algorithms
    - Develop realistic graphics
    - IA: neural networks at the rescue
    - Put the hands in the hardware for fine tuning
  - Cool ideas are welcome
  - Go crazy !

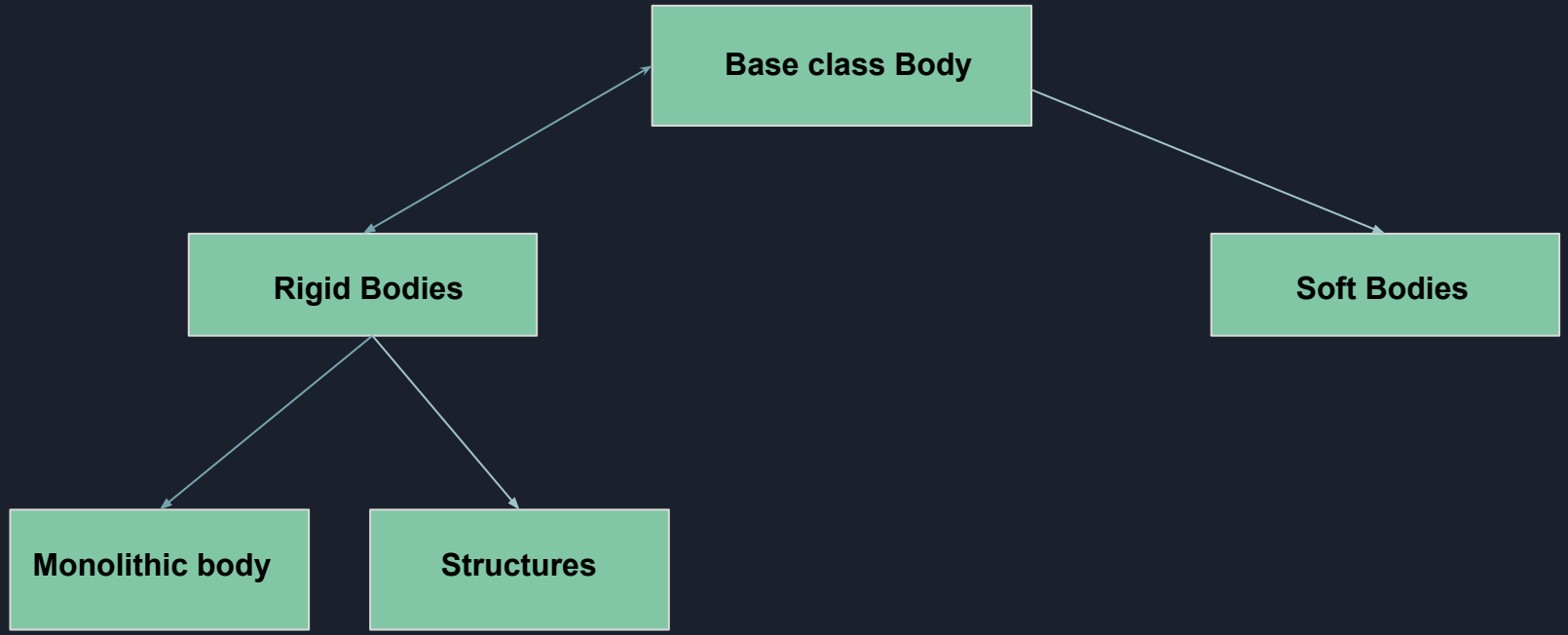
Pegase Engine

- Body
  - Base object of the scene
  - Complexity varies from basic solids (sphere, cube) to Real World structures (buildings, cars, aircrafts)
  - The real world includes rigid bodies (stones) and soft bodies (wheels), and gas (air, water)

Body

- Properties
  - Type (sphere, cube, complex shape)
  - Weight
  - Dimensions
  - Initial position (X,Y,Z)
  - Initial velocity (X,Y,Z)
  - Angular speed
  - Axis of rotation

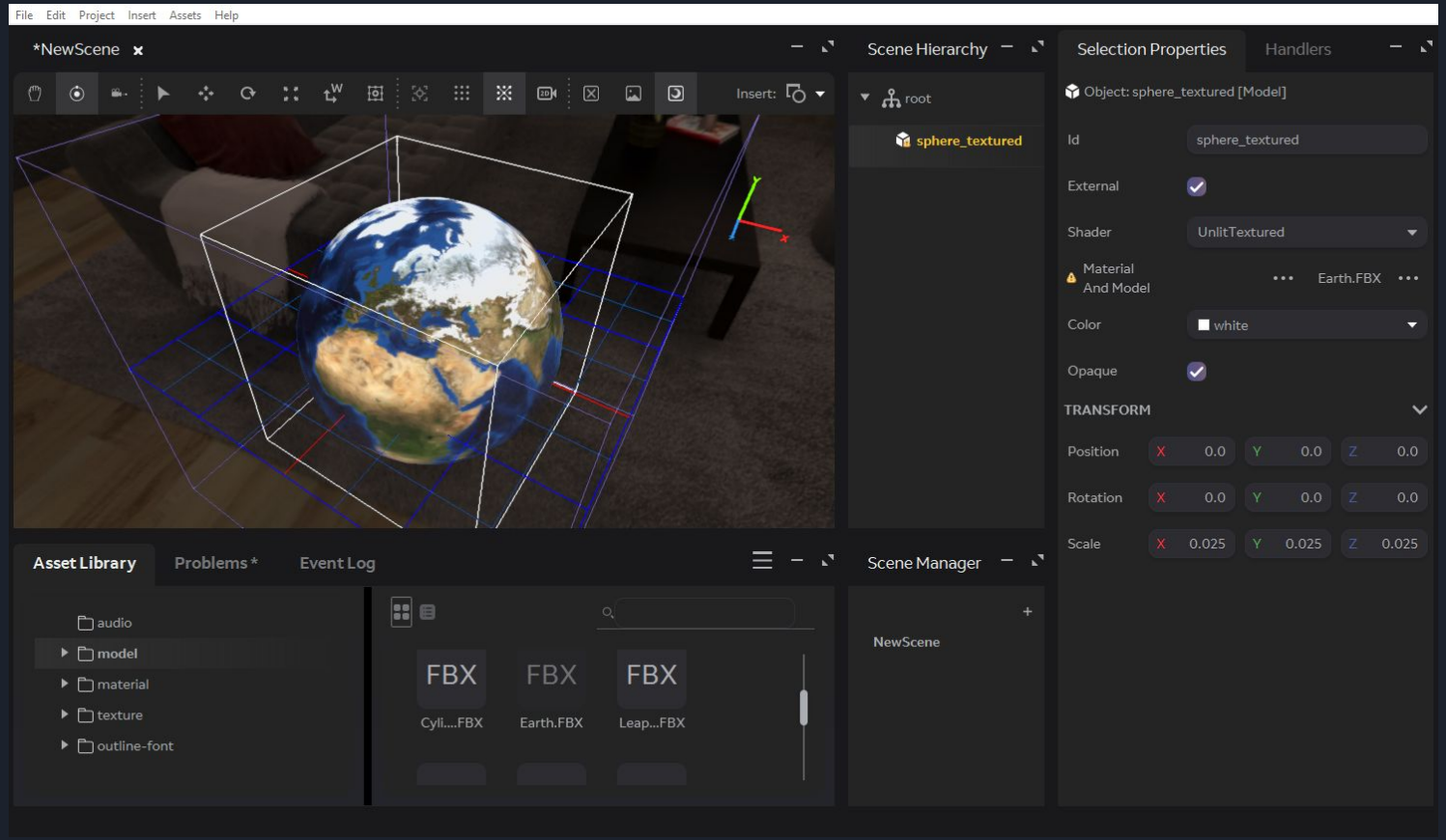
Bodies - Class Hierarchy



Expected features

- Handle collisions between
  - Rigid bodies (very soon)
  - Structures (quite soon)
  - Soft bodies (one day)
- Destruction of structures (one day)
- Handle gravity / no gravity
- Editor for easier scene creation and tuning
  1) create environment, bodies' properties
  2) simulate
- Support of multi-threading

Scene editor

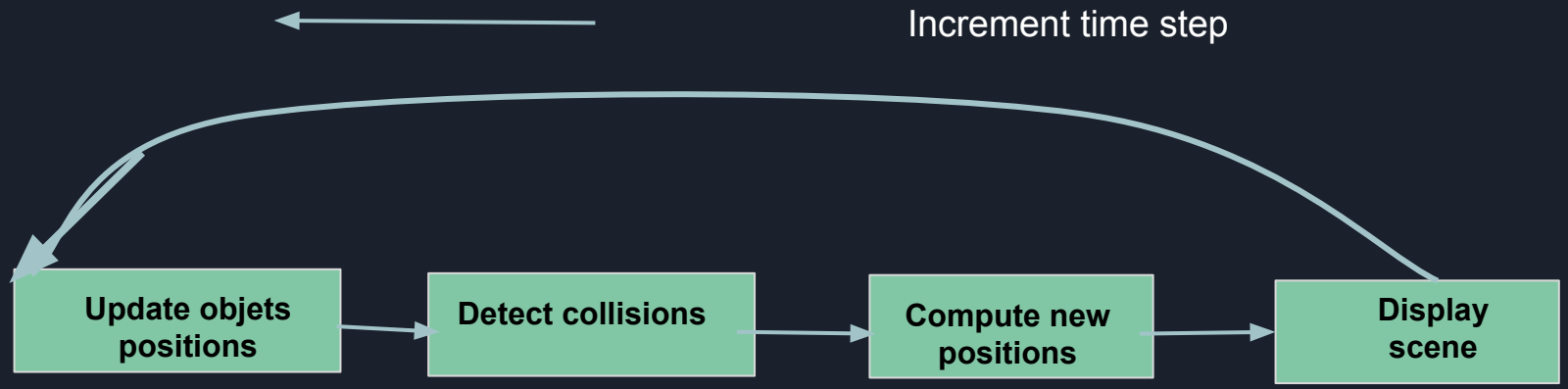


Handling collisions

- Various algorithms to implement collisions:
  - AABB Tree
  - GJK (Gilbert–Johnson–Keerthi) algorithm
  - SAT algorithm
  - ….

The engine

- The main loop: closed circuit



The engine

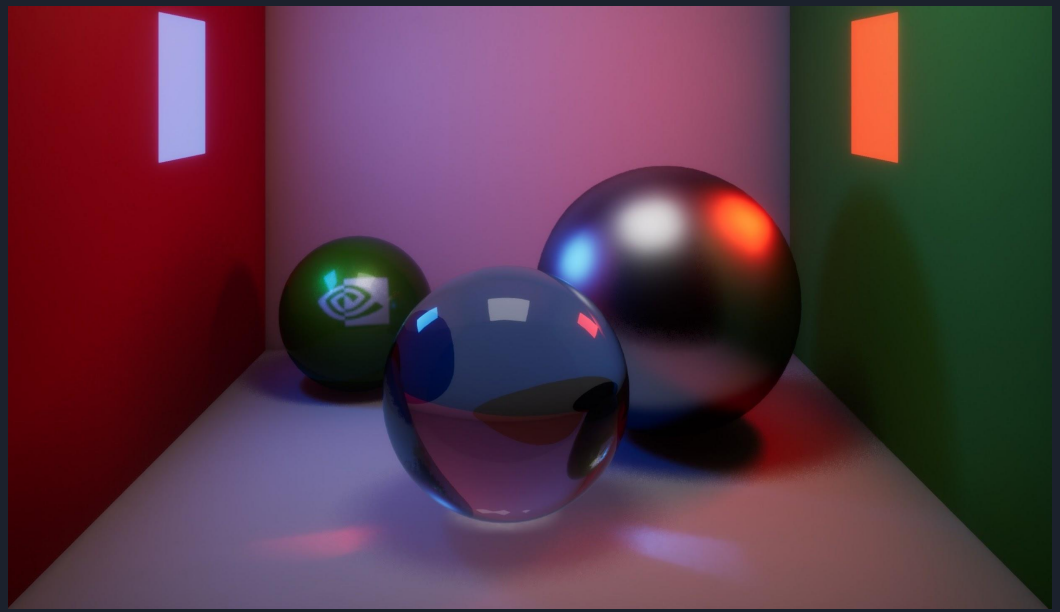
- Language: C++
- Load the scene
  - Load scene elements (bodies, plan) from JSON file as input stream
- Design patterns:
  - Singleton: ensure objects are instantiated only once
  - Observer: implement observer to monitor system as observee
  - Flyweight: Share common properties for a huge number of objects in order to save memory
- Maths: express bodies' behavior
  - Rotation matrix
  - Vector3: express object position (x,y,z)
  - Quaternions: Mathematical object to express rotation axis and velocity

The engine

- Utils: tools for debugging the engine
  - Memory manager: prevent memory leaks
  - Smart pointers: handle creation and destruction of objects automatically
  - Logger: bug tracker
    - Print file, function name and line number
    - different levels of criticality (INFO,DEBUG,WARN,ERROR)
- Media manager: Import objects from files
  - 3D objects
  - Textures
- Plugin manager: easily add new features (shaders,...)

Beautify the engine

- Rendering: OpenGL vs DirectX
  - Separate physics from rendering
  - Use Ray Tracing ?

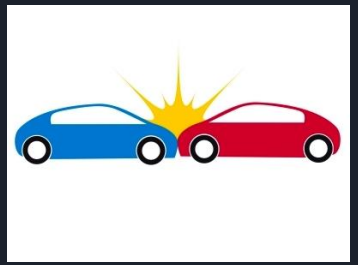


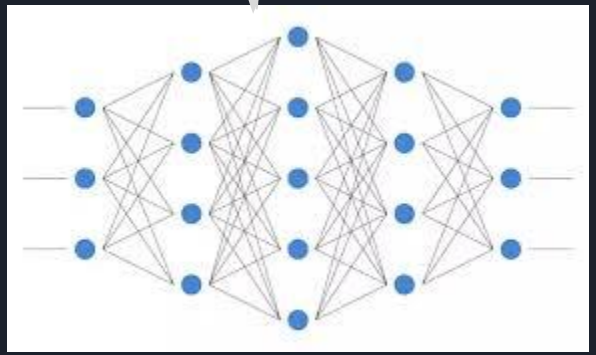
Make the engine smarter

- How can we use AI to enhance physics ?
  - How to train neural networks in order to enable more realistic scenes ?
  - Train a neural network to compute a better scene

Back end

Computer scene in first place





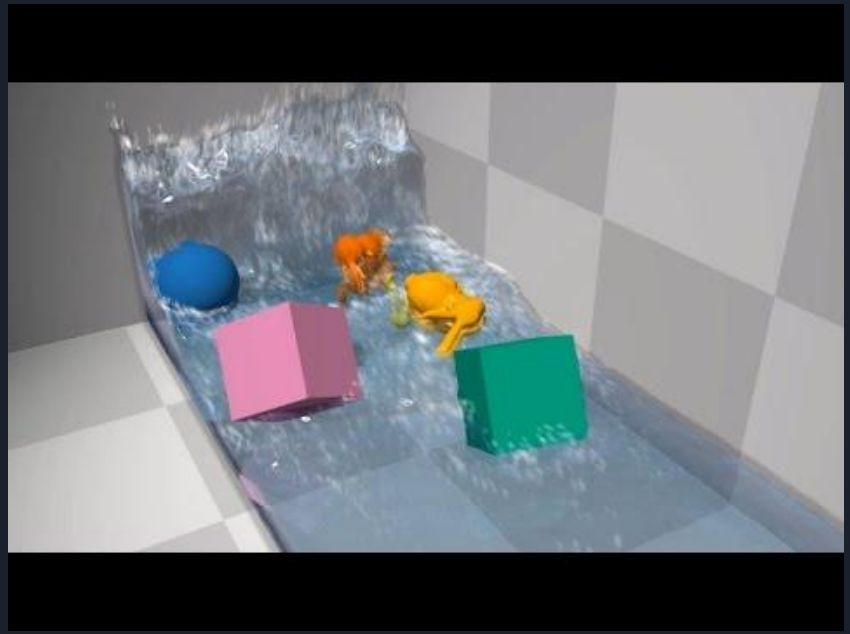
Enhanced scene

Make the engine smarter

- Example of use of a neural network for computer graphics

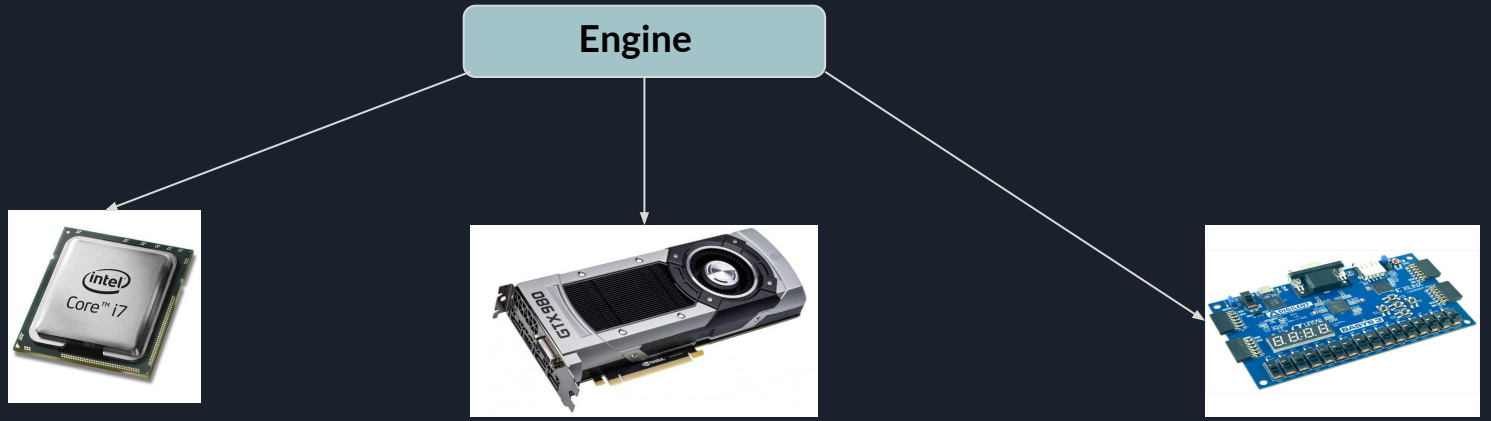
Accelerate the engine

- This kind of software implies heavy computations
  - Examples :
    - N-body simulation
    - Fluids dynamics: simulate liquids
- We need compute cores at the rescue



Accelerate the engine

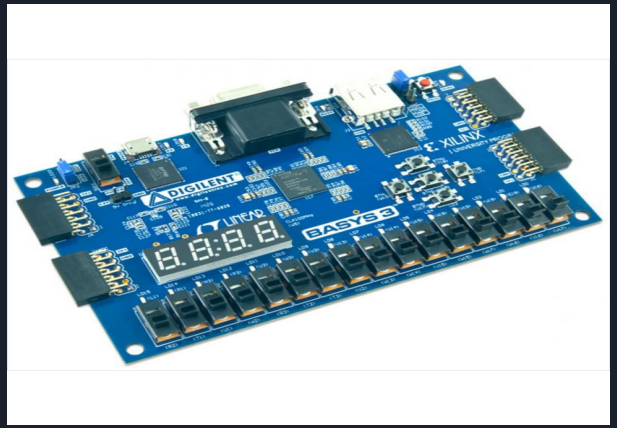
- Support of multi-threading to accelerate computations
  - Leverage multi-core platforms and speed up computations
  - Support of various HW platforms (CPU, GPU, FPGA,...)



  - Experiment different parallel programming models
    - threads, tasks, etc: MPI, OpenMP, CUDA
  - Design runtime

Accelerate the engine

- Research topic: design processors dedicated to handle calculations of physics
- Actually exists: Physics Processing Units (PPU)
  - https://github.com/NVIDIAGameWorks/PhysX
- Program FPGA (Field Programmable Gate Array) and use it as a prototype to perform calculations

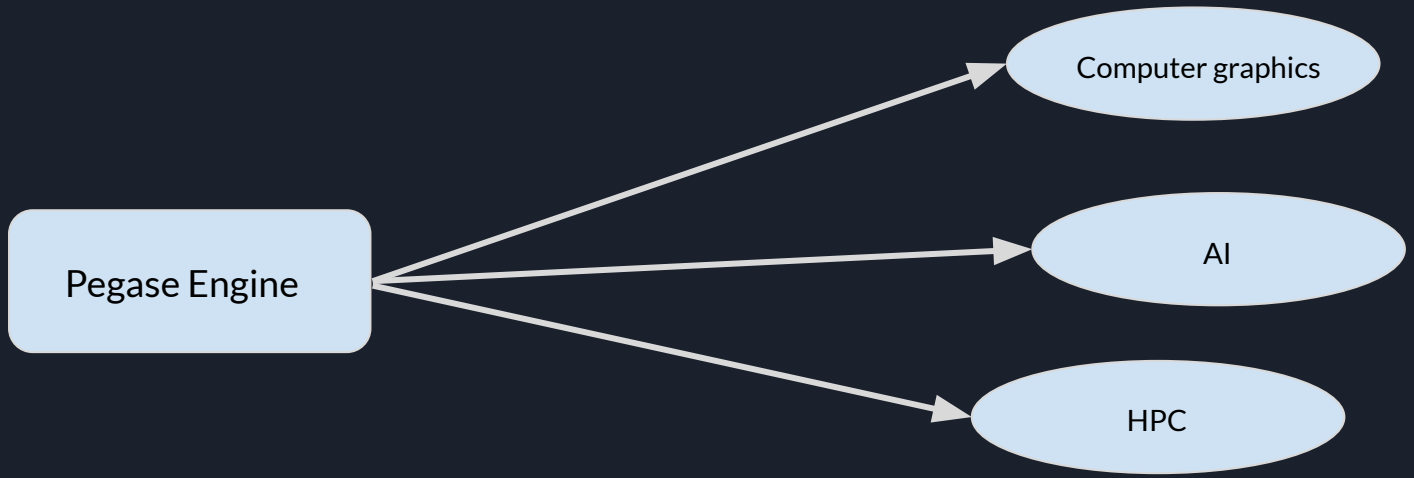


Accelerate the engine

- Large scale experiments

Let’s sum it up

- Pegase Engine aims at being a physics engine
- Pegase Engine is also a research platform



Links

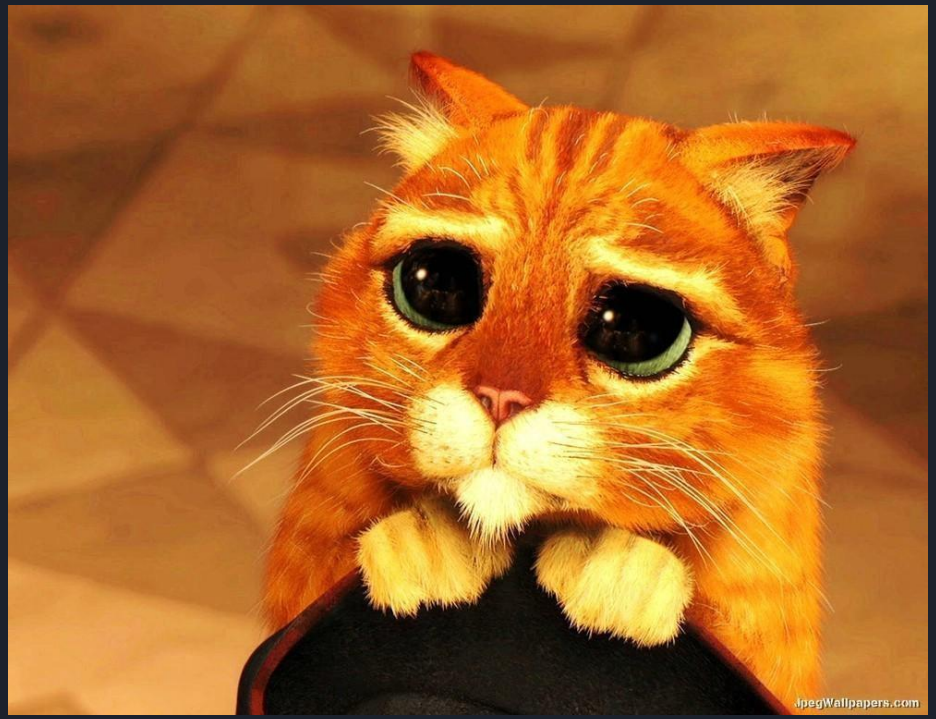
- ReactPhysics3D
  - https://www.reactphysics3d.com
  - https://github.com/DanielChappuis/reactphysics3d.git

My department hires .....

…. brave volunteers

… who will receive my eternal gratitude

And the project manager is nice



very nice

Interested ?

- What do you need ?
  - A terminal
  - Git (versioning tool => open source)
  - An editor (vim, IDE)

- $ git clone https://github.com/aurelemaheo/pegaseengine
- Let’s get started
- ….
- ….
- And welcome on board !