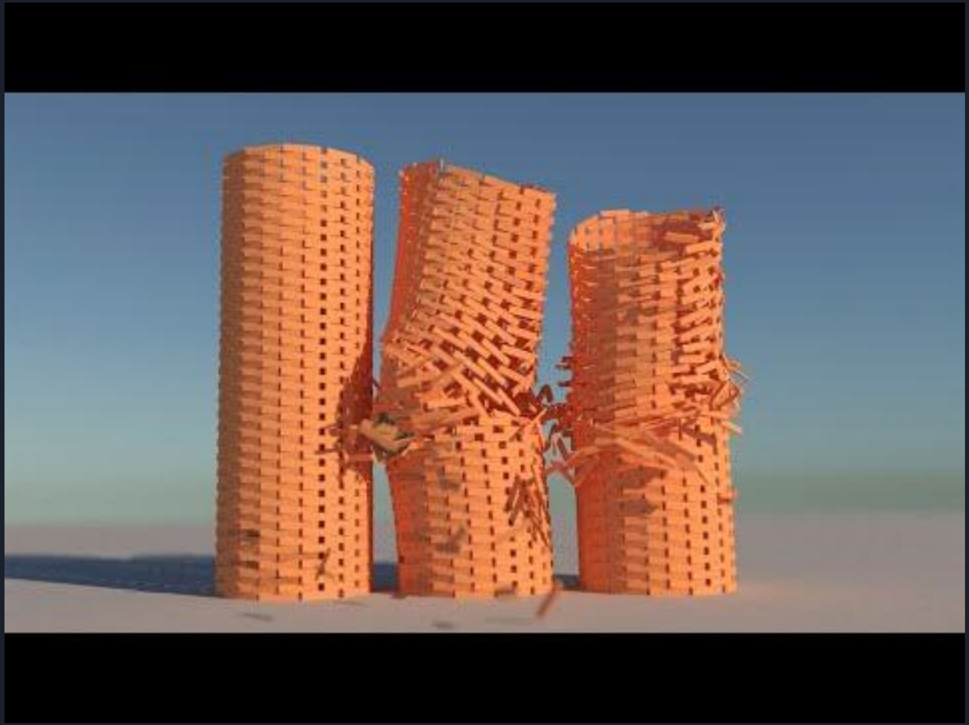
Pegase Engine

Simulating the real world

Aurèle MAHEO

Physic Engine



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
  - Play with fun techs: IA, Deep Learning, computer graphics, HPC, ...
  - 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 !

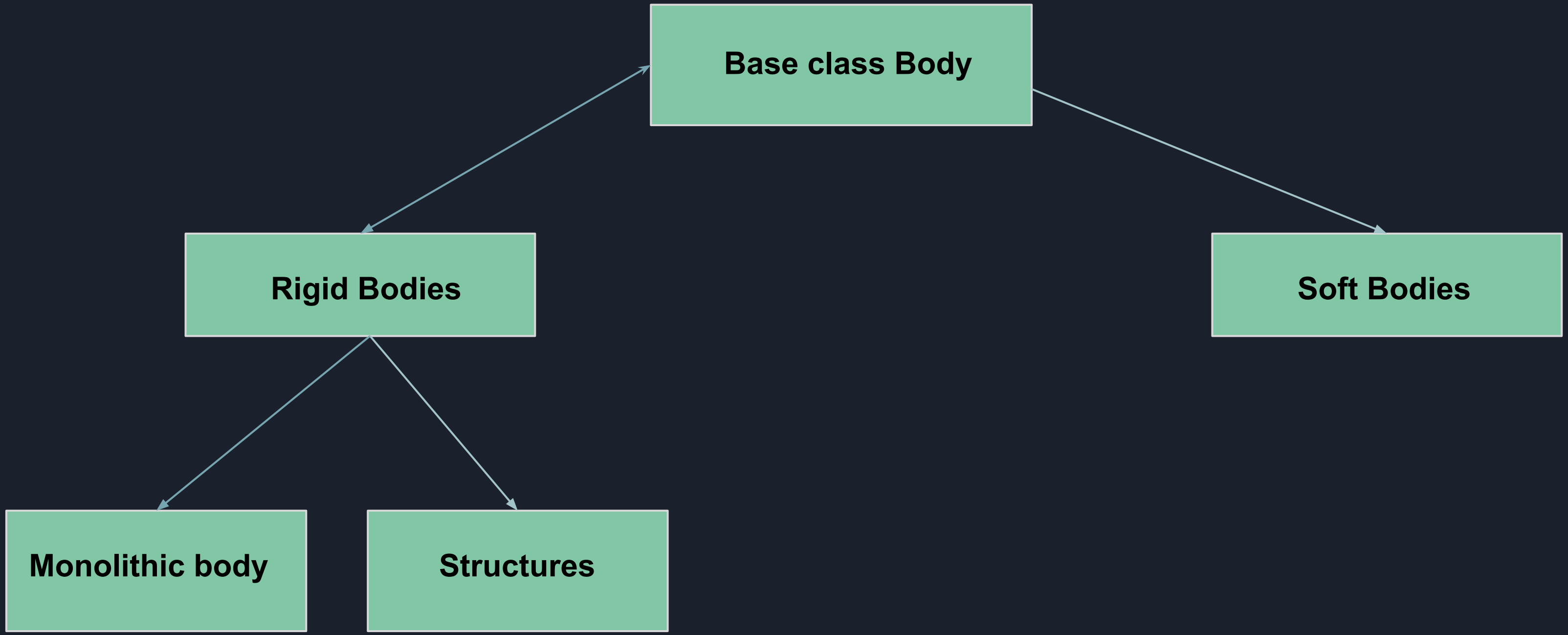
Pegase Engine

- 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



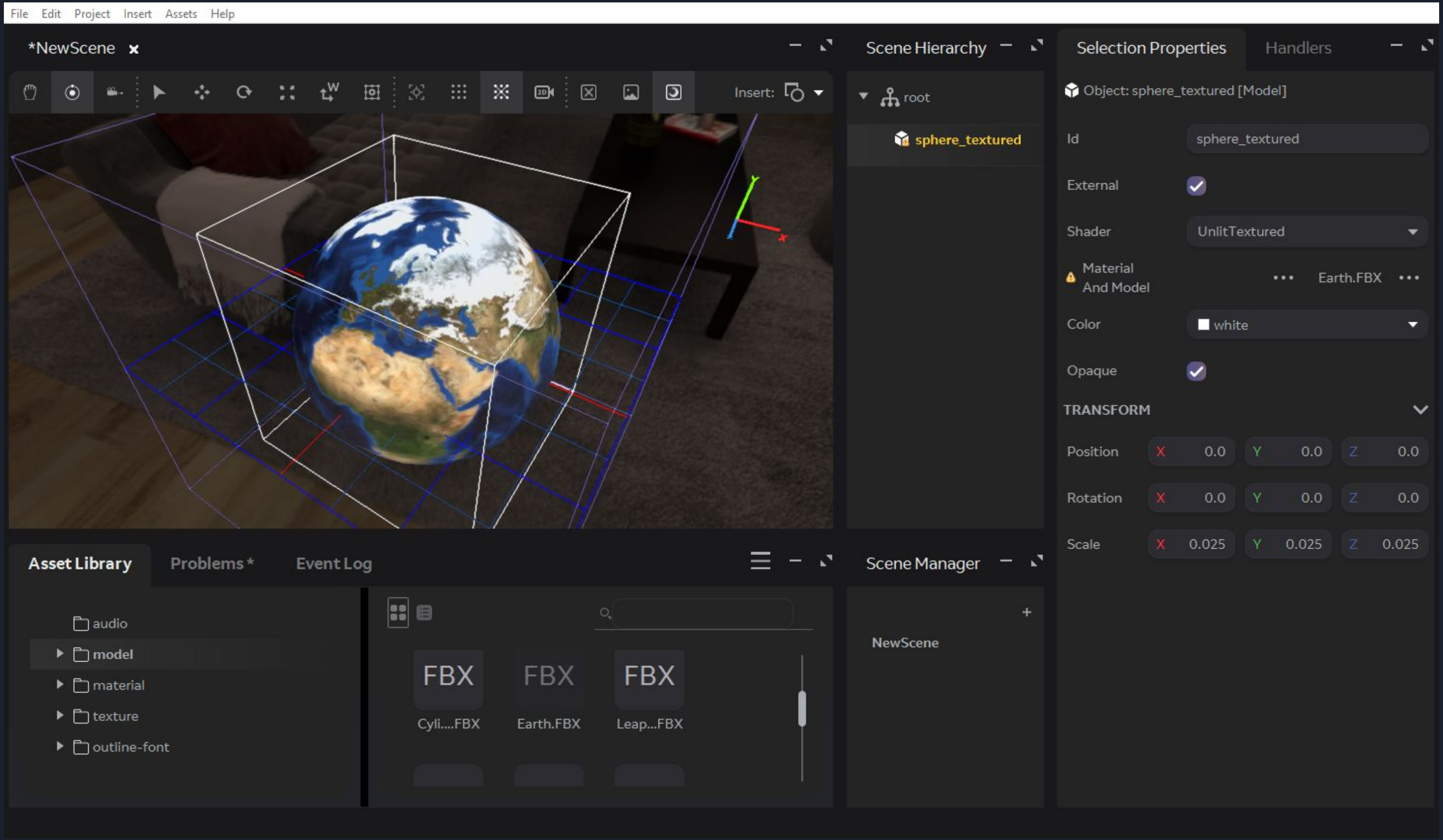
Expected features

- 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

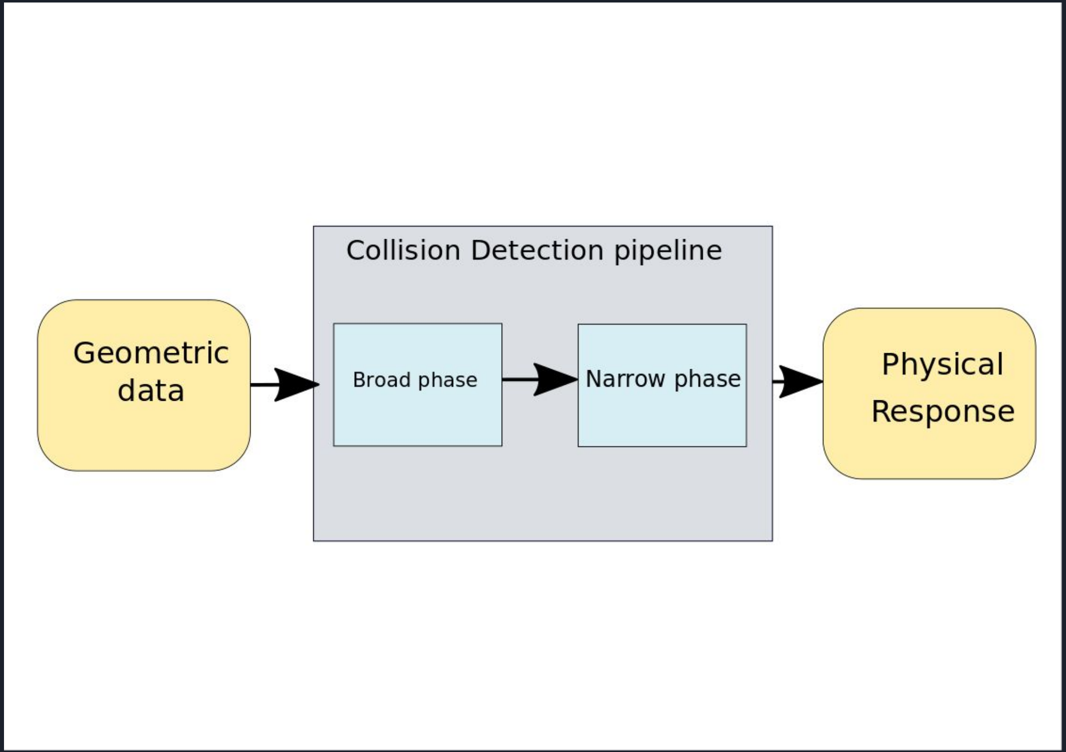
Expected product

- Applications
  - Video games
  - Industry companies (aerospace, cars, civil engineering)
- A product for:
  - To be used for developing video games
  - A framework for aircraft designers, car designers, civil engineers
- Develop a high level language for easy scene creation

Scene editor



Handling collisions



Handling collisions

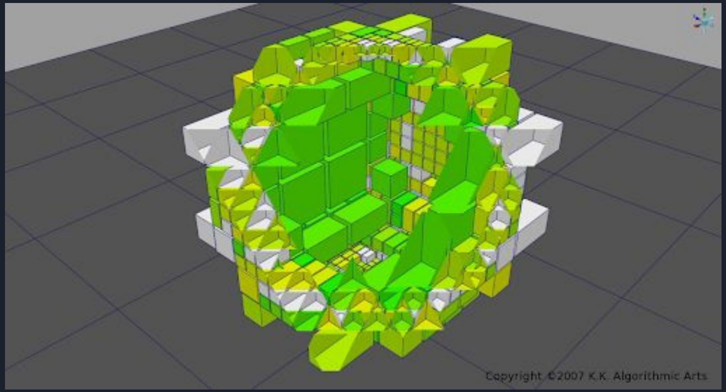
- Broad phase collision detection
  - Determines which physics bodies *might* intercept
- Narrow phase collision detection
  - Performs the actual detection, and resolves any collisions that have occured

Handling collisions

- Algorithms for Broad Phase collision detection
    - Brute Force: very exhaustive
        - Axis-Aligned-Bounding-Box (AABB)
        - Oriented-Bounding-Box (OBB)
    - Spatial Partitioning
        - Octree
        - Quadtree
        - K-d tree structure
        - Binary Space Partitioning (BSP)
    - Topological
        - Sweep and Prune
    - Spatial subdivision

Broad Phase collision detection

- Binary Space Partitioning (BSP)
  - Gives a representation of the scene and all objects using a tree data structure
  - Uses a BSP tree
  - BSP is a generalization of the k-d tree data structure
  - Performs recursive subdivisions



Handling collisions

- Algorithms for Narrow Phase collision detection
  - Feature-based algorithms: Voronoï Marching
  - Simplex-based algorithms: Gilbert-Johnson-Keerthi (GJK)
  - Bounding volume-based algorithms: Bounding Volume Hierarchies (BVH)
  - Separating Axis Theorem (SAT)

Narrow Phase collision detection

- Gilbert–Johnson–Keerthi (GJK) distance algorithm
  - Determines the minimum distance between 2 convex sets
  - Relies on a support function
  - Input
    - A set of points defining the convex polyhedron A
    - A set of points defining the convex polyhedron B
  - Output
    - The distance between polyhedra
    - 2 closest points
    - The 2 simplices of A and B containing the 2 closest points

Handling collisions

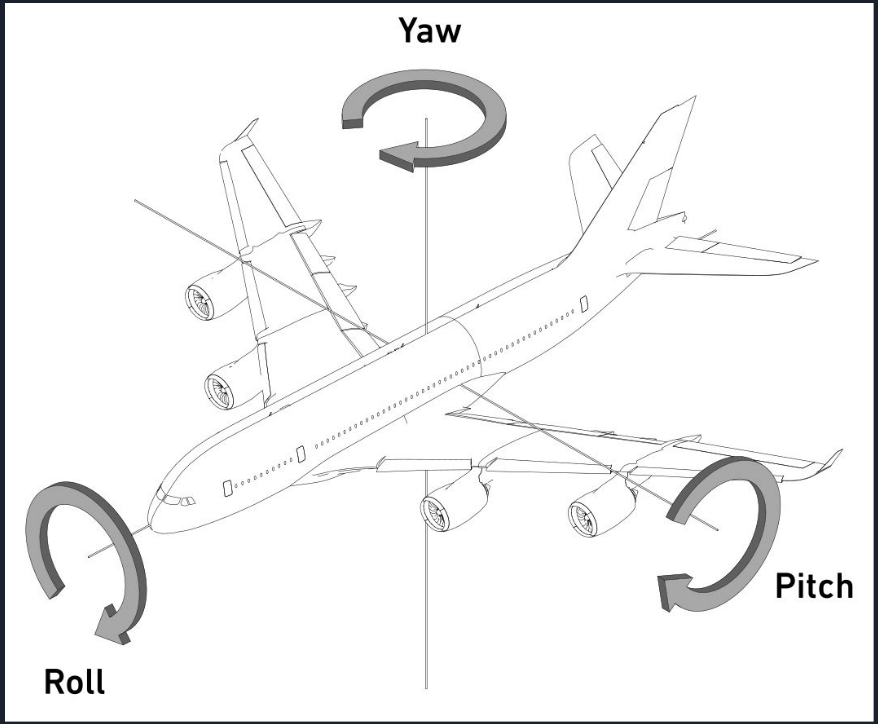
- Links
  - https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-32-broad-phase-collision-detection-cuda
  - https://hal.archives-ouvertes.fr/hal-00474171/document
  - https://www.merl.com/publications/docs/TR97-23.pdf
  - https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects

Handling collisions

- Research papers
  - Broad Phase collision detection
    - “Efficient Algorithms for Two-Phase Collision Detection”
    - “A Broad Phase Collision Detection Algorithm Adapted to Multi-cores Architectures”
    - “Dynamic Adaptation of Broad Phase Collision Detection Algorithms”
    - “Broad-Phase Collision Detection Using Semi-Adjusting BSP-trees”
    - “Hardware Accelerated Broad Phase Collision Detection for Real time Simulations”
    - “Parallelizing broad phase collision detection algorithms for sampling based path planners”

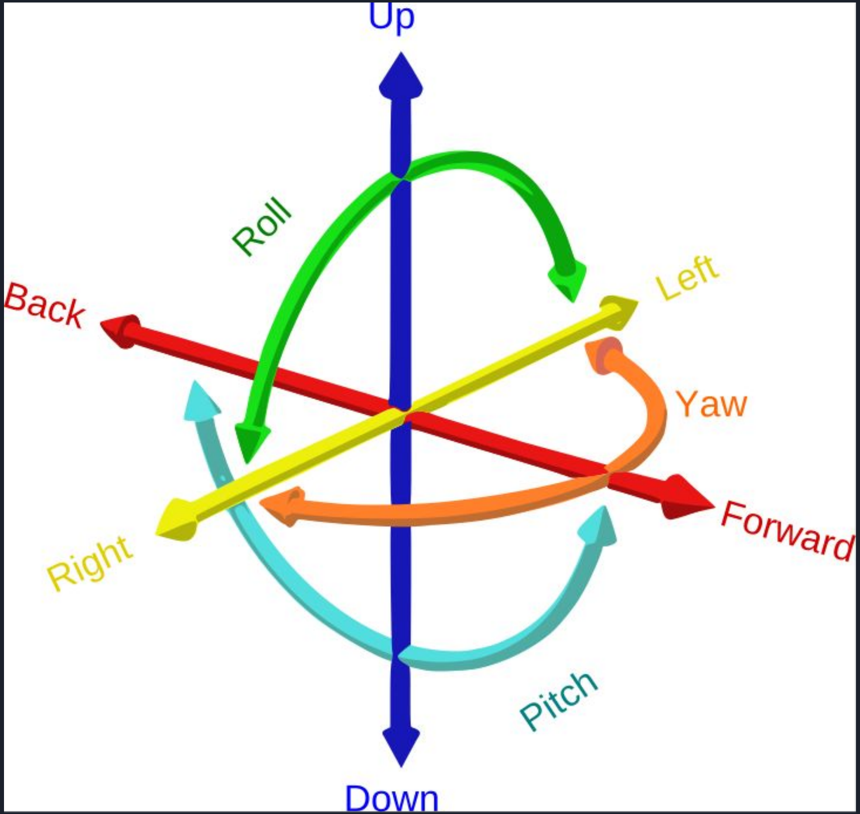
Body motion

- Principal axes:
  - Yaw: Vertical axis
  - Pitch: Transverse axis
  - Roll: Longitudinal axis

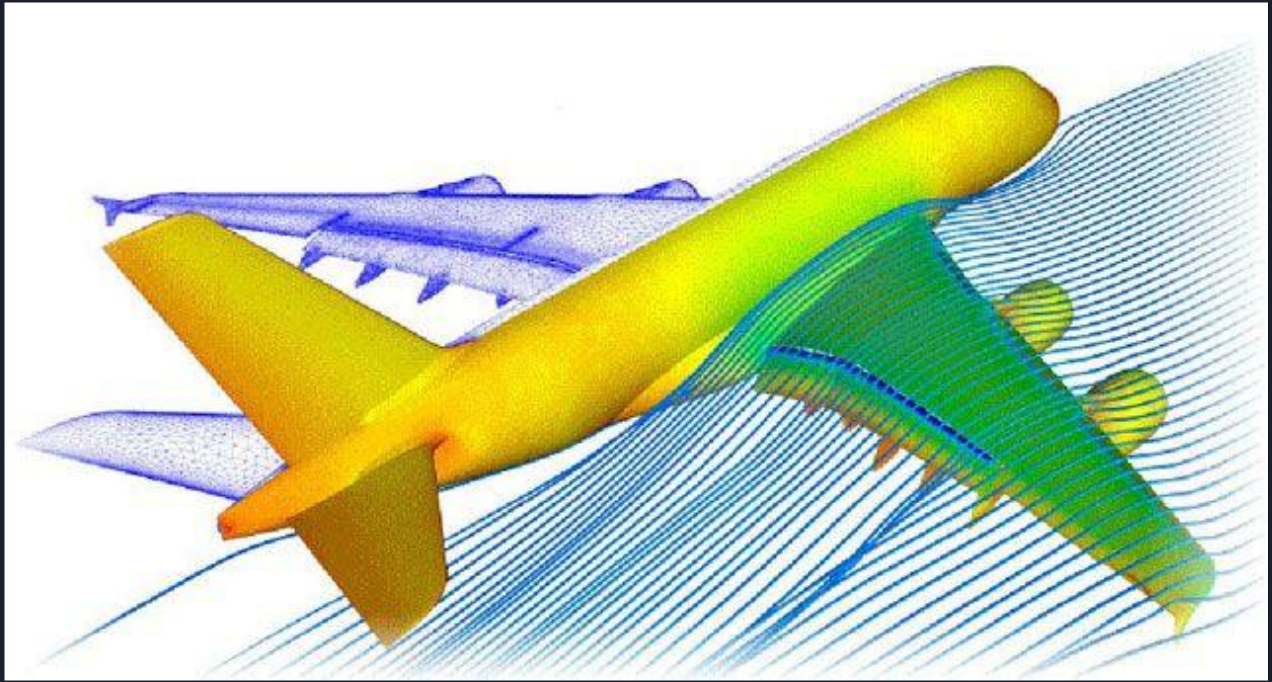


Aircraft motion

- Six degrees of freedom
  - Transational
    - Surge
    - Sway
    - Heave
  - Rotational
    - Yaw: Vertical axis
    - Pitch: Transverse axis
    - Roll: Longitudinal axis



Fluids dynamics



Fluids dynamics

- This branch of physics is modeled by Navier-Stokes equations:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = 0$$

$$\rho \frac{D\mathbf{u}}{Dt} = \rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \nabla\cdot\left\{\mu\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^{\mathrm{T}} - frac2/3(\nabla\cdot\mathbf{u})\mathbf{I}\right) + \zeta(\nabla\cdot\mathbf{u})\mathbf{I}\right\} + \rho\mathbf{g}$$

Fluids dynamics

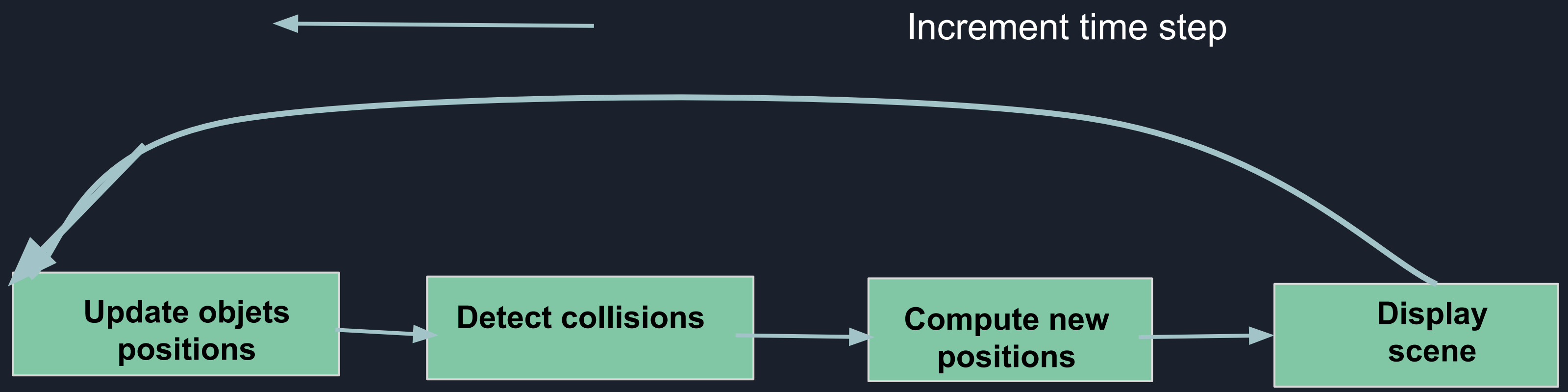
- Expected features:
  - Modeling of liquids and gas (water and air)
  - Model different levels of viscosity

Fluids dynamics

- Lattice Boltzmann methods (LBM)
  - New simulation technique for complex fluid systems
  - Advantages
    - Runs efficiently on massively parallel machines
  - Limitations

The engine

- The main loop: closed circuit



The engine

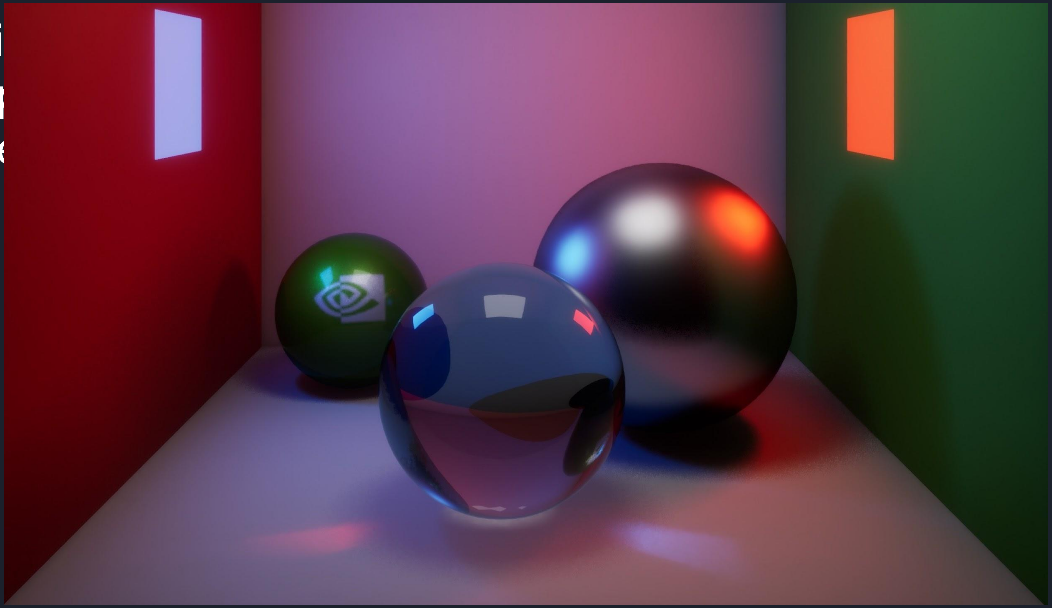
- Language: C++
- Load the scene
  - Load scene elements (bodies, plan) from JSON file as input stream
- Design patterns:
  - Singleton: ensure objects are instantiated only once
  - Observer: implement observer to monitor system as observee
  - Flyweight: Share common properties for a huge number of 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 from 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,...)

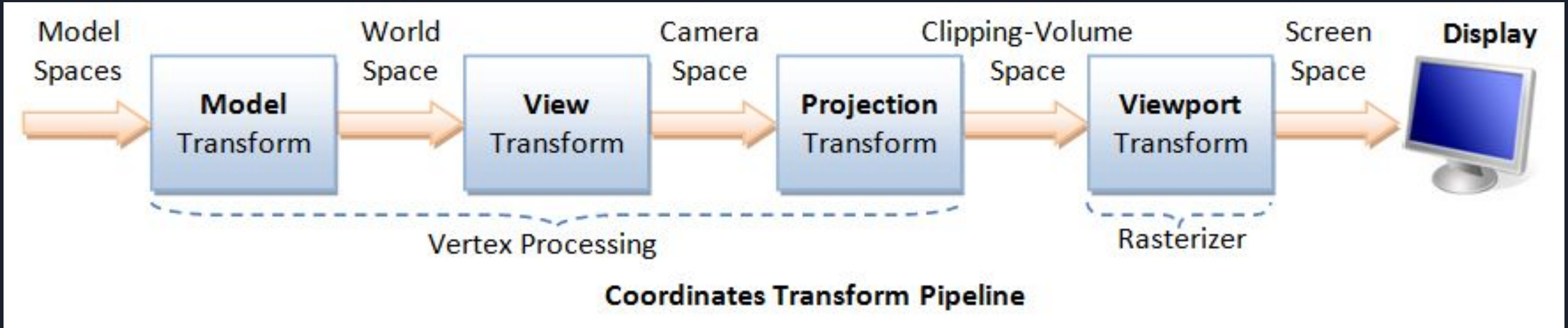
Beautify the engine

- Renderi
    - Se
    - Use



Beautify the engine

- Parallelize the graphic pipeline ?

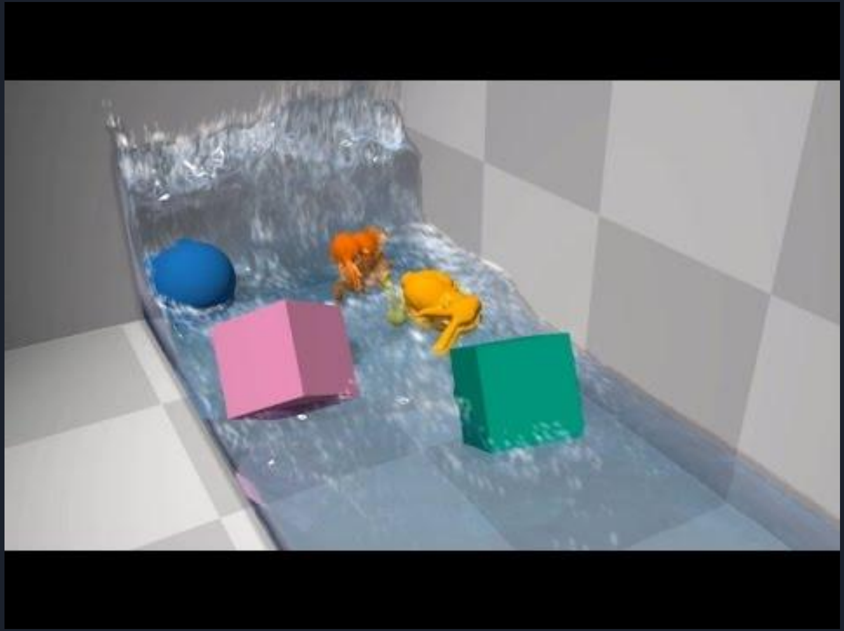


Beautify the engine

- Implement a dashboard for graphic properties (light, texture, etc)

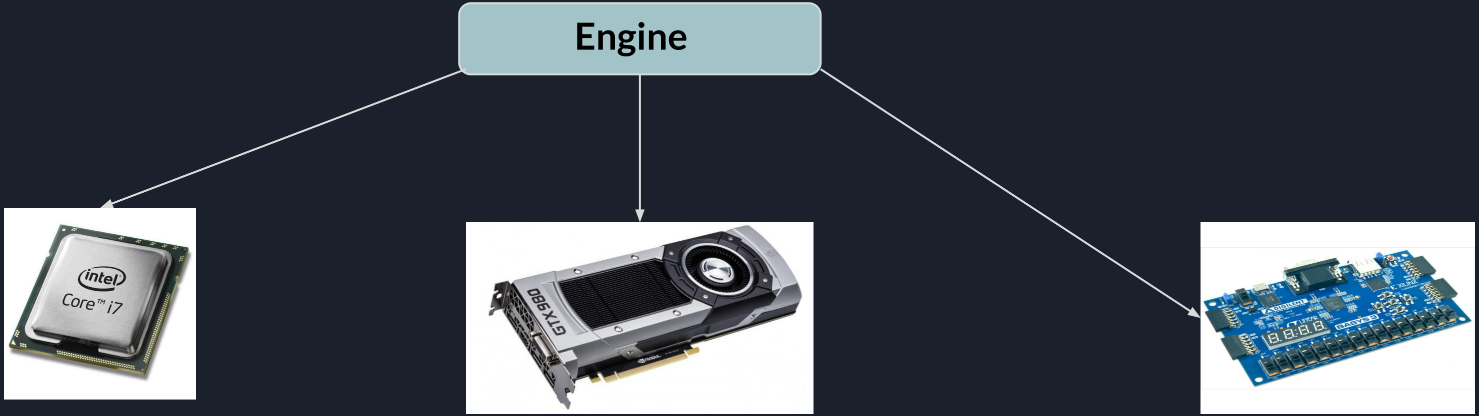
Speed up the engine

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



Speed up the engine

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



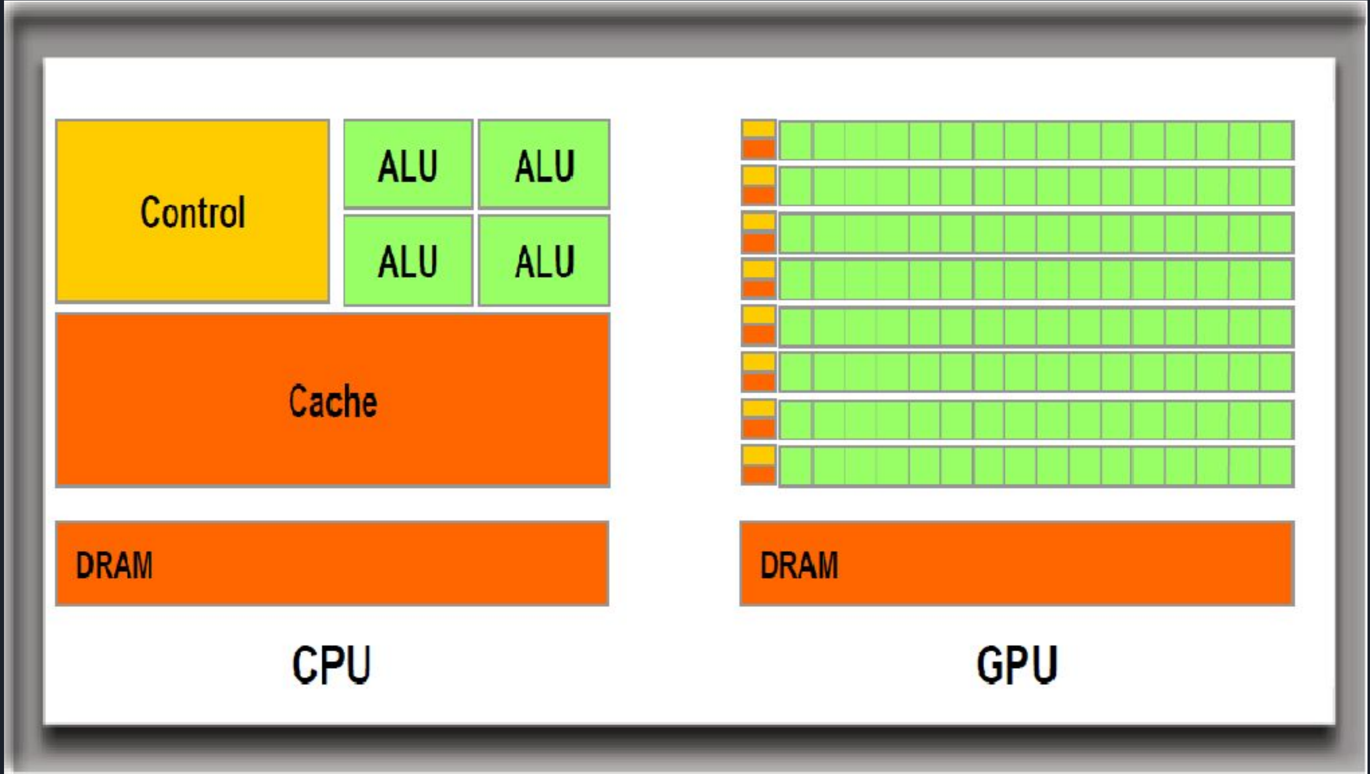
  - Experiment different parallel programming models
    - threads, tasks, etc: MPI, OpenMP, CUDA
  - Design runtime

Speed up the engine

- Focus on a Graphic Processing Units (GPU)
- GPUs are essential components for graphics rendering and game physics
  - Originally used as graphic cards
  - Provides hardware acceleration for graphics
    - Computing graphic display
    - 3D graphic rendering
    - Support for programmable shaders
    - Other graphic functions
    - Ray tracing
  - Computations
    - GPUs very suited for embarrassingly parallel problems
    - Floating Point precision
    - Applications:
      - Computational Fluids Dynamics
      - Computational Biology
      - Molecular dynamics
      - Video signal processing
      - Cryptography

Speed up the engine

- CPU versus GPU microarchitecture



Speed up the engine

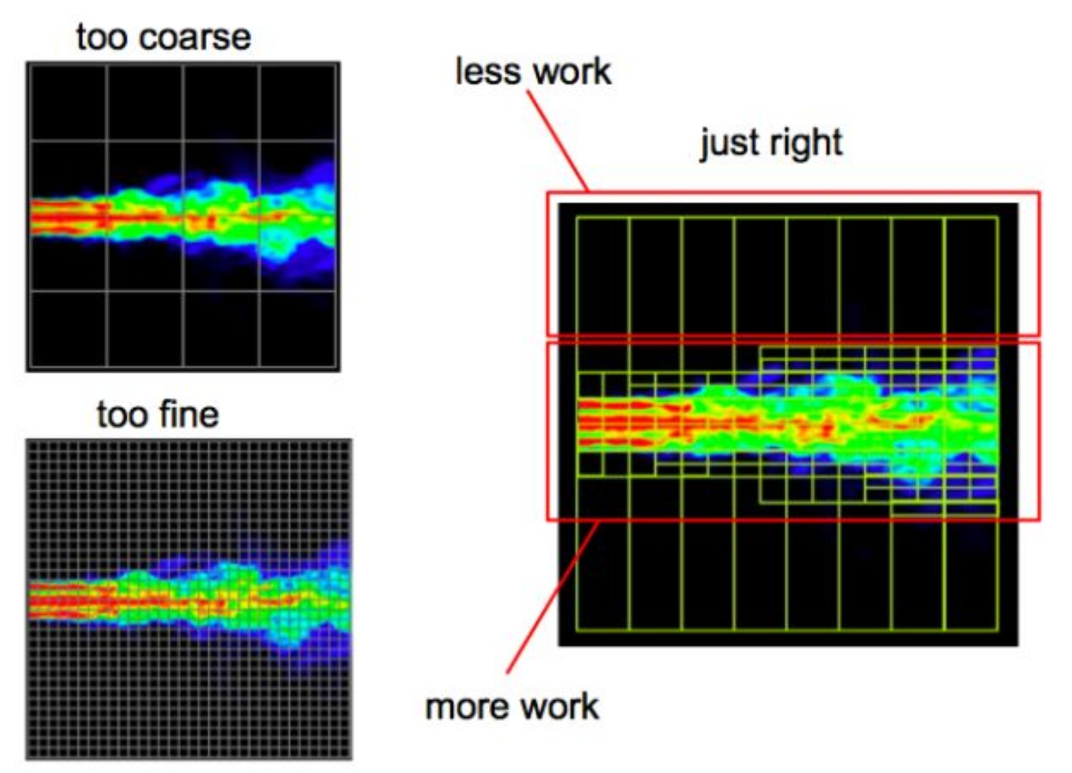
- Have an insight of a GPU microarchitecture (Nvidia GeForce)

Speed up the engine

- Have an insight of a GPU microarchitecture
  - GPUs consist of numerous compute cores
  - Consists of Streaming Multiprocessors (SMs)
  - GPU Nvidia Geforce microarchitectures
    - Fermi (2010)
    - Kepler (2012)
    - Maxwell (2014)
    - Pascal (2016)
    - Turing (2018)
    - Ampere (2020 ?)
  - Features different hardware capabilities
    - Compute cores
    - Ray Tracing cores
    - Tensor cores (AI)

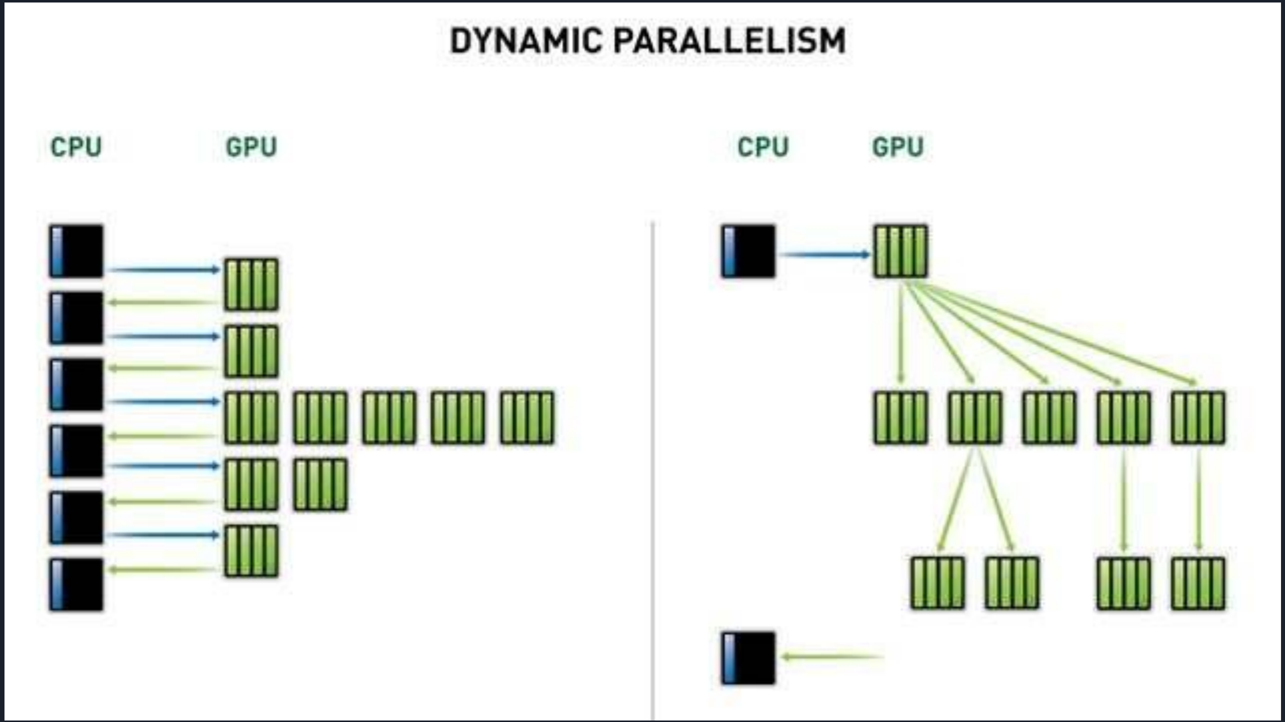
Speed up the engine

- Dynamic parallelism



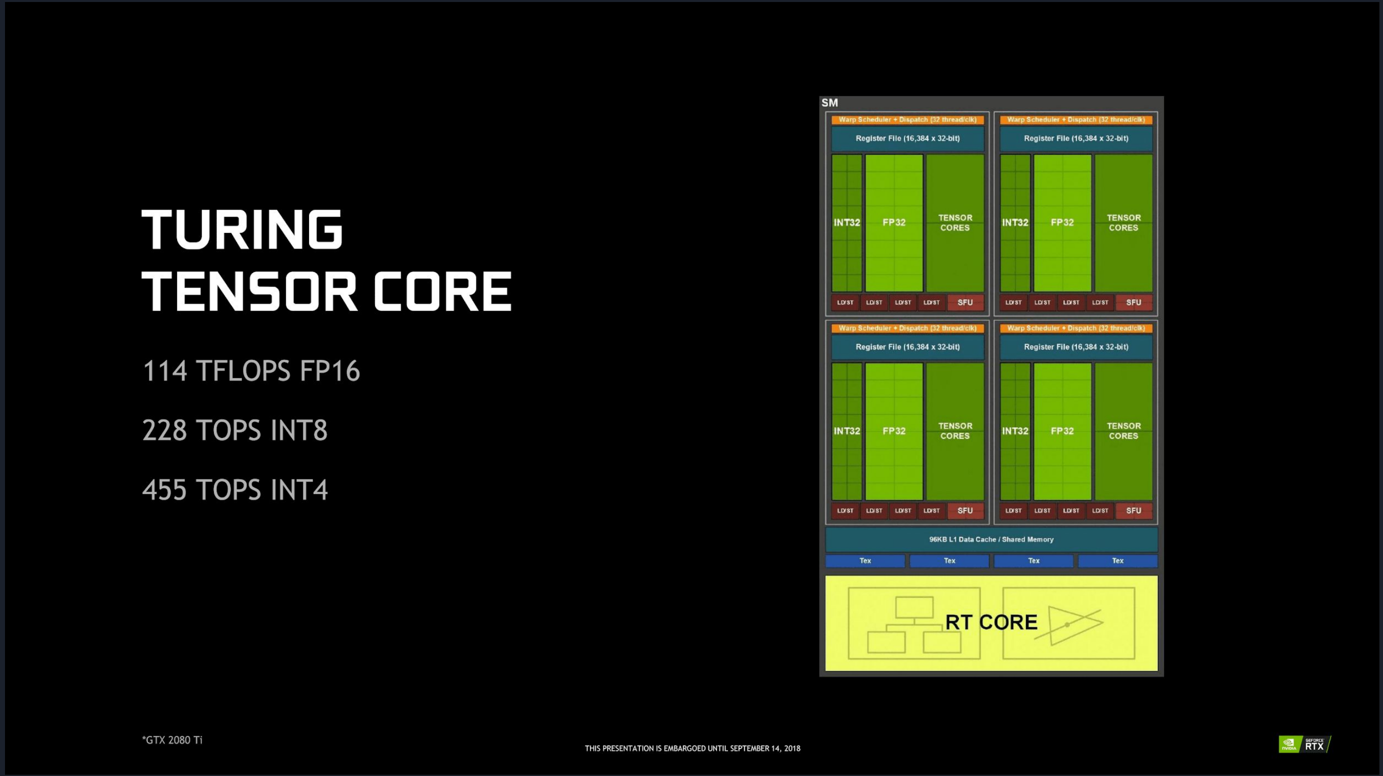
Speed up the engine

- Dynamic parallelism



Speed up the engine

- Use last generations of NVIDIA GPUs to accelerate neural networks



Speed up the engine

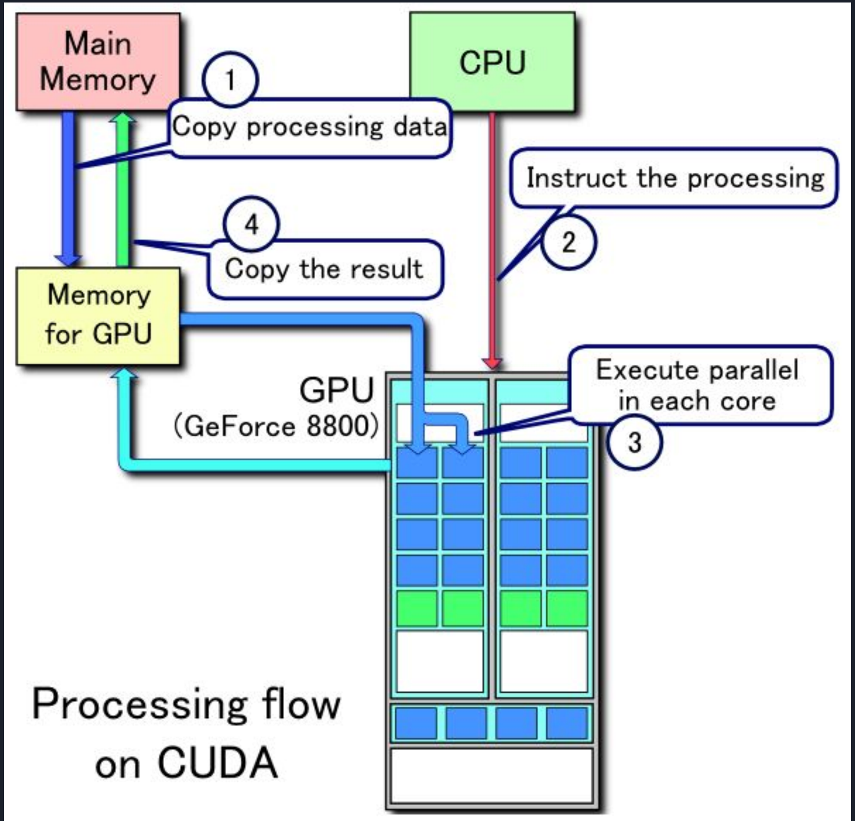
- What is a Tensor Core ?
  - Different from a regular GPU
  - New type of processor core
  - Performs a type of specialized matrix math, suitable for deep learning
- Links
  - “GPU Tensor Cores for fast Arithmetic Reductions”

Speed up the engine

- How to program GPUs to speed up computations ?
    - Consider the GPU as a GPGPU (General Purpose GPU)
    - Introducing CUDA (Compute Unified Device Architecture)
        - Parallel computing platform and API
        - Works with programming languages such as C / C++ / Fortran
        - Considers the CPU as a host and the GPU as a device
        - Data offloading from the host to the device have to be explicit

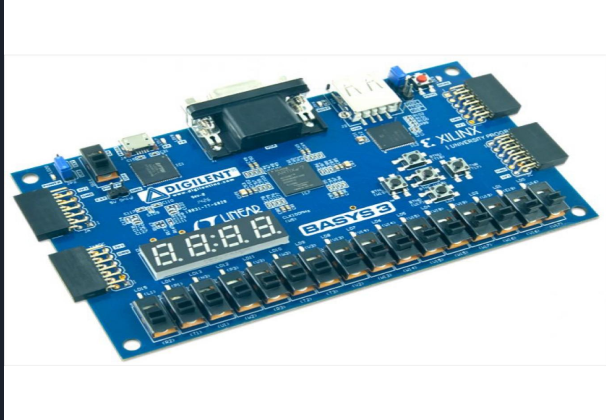
Speed up the engine

- CUDA processing flow



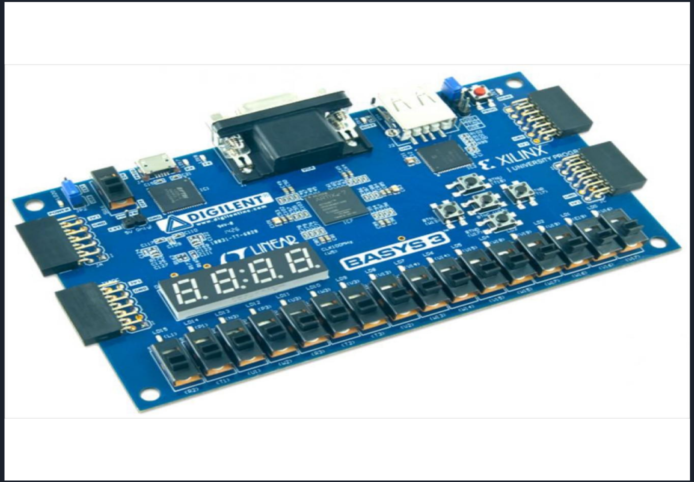
Speed up the engine

- 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



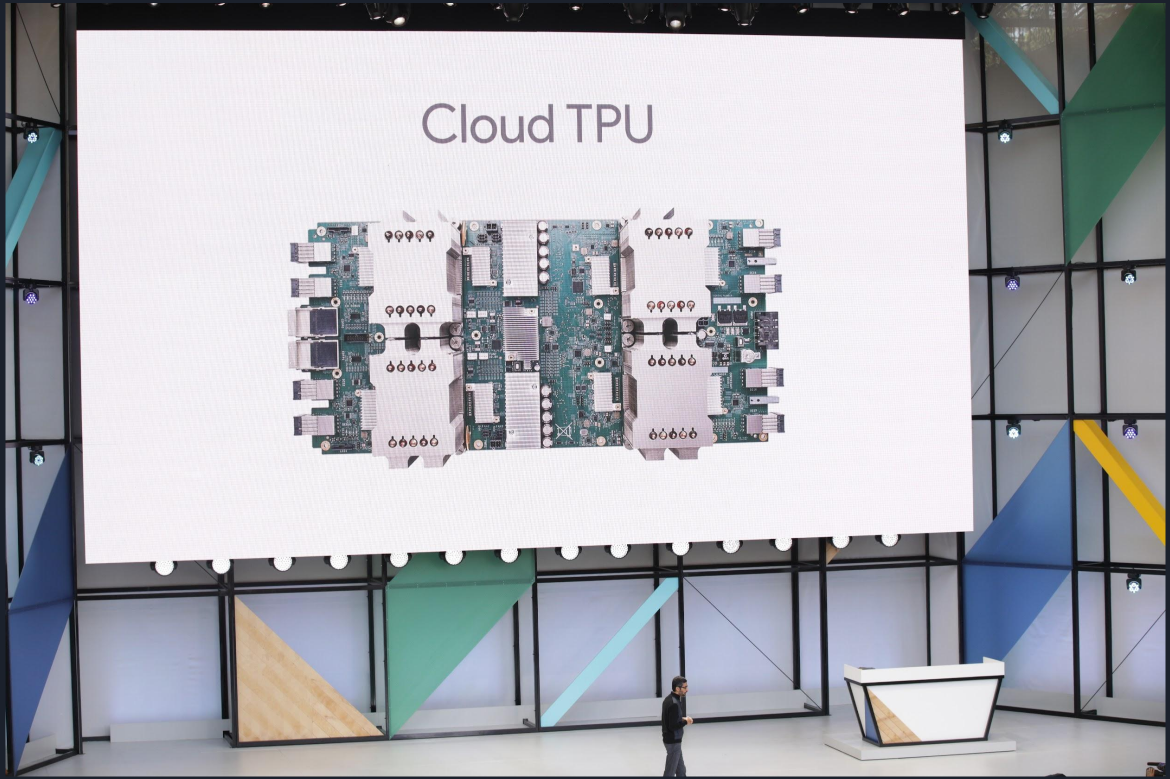
Speed up the engine

- FPGA (Field Programmable Gate Array)
  - Electronic component designed to be configured
  - Configuration done by a Hardware Description Language (HDL)
  - Hardware composed of reconfigurable interconnects
  - Used to process very specific applications
  - Very popular in embedded systems’ business



Speed up the engine

- Google TPU (Tensor Processing Unit)



Speed up the engine

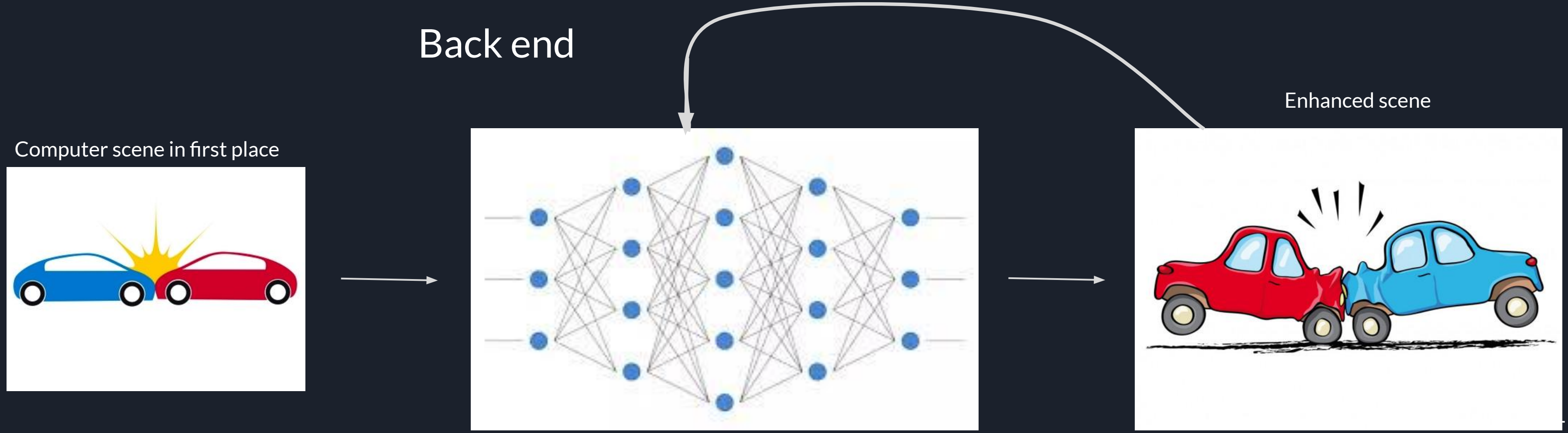
- Large scale experiments

Speed up the engine

- Cloud technologies
  - Alternative to supercomputers or local HW accelerators
  - Many existing solutions
    - Amazon AWS
      - Compute: AWS Lambda, EC2
      - AI: AWS SageMaker
      - Storage: S3
    - OVH
      - Compute: GPU, IOPS
    - Scaleway

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



Make the engine smarter

- Use Deep Learning for collision detection

Make the engine smarter

- Example of use of a neural network for computer graphics

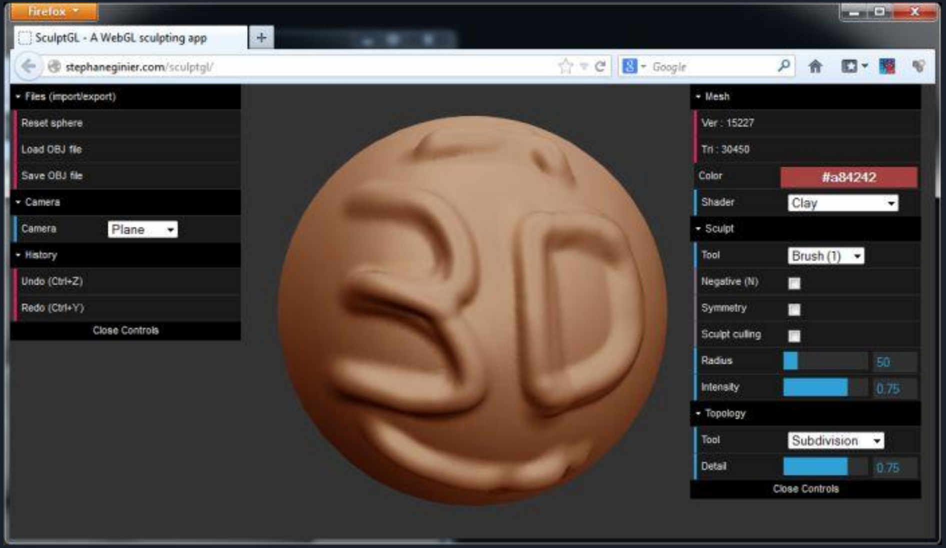
Make the engine smarter

- Orchestrator
  - Scheduler for best use of:
    - HW resources (GPUs, FPGA, etc)
    - Collision algorithms
  - Define scene patterns
    - N-Body simulation
    - Fluid dynamics
  - Decide whether or not is it relevant to use IA techniques

Visualize the engine

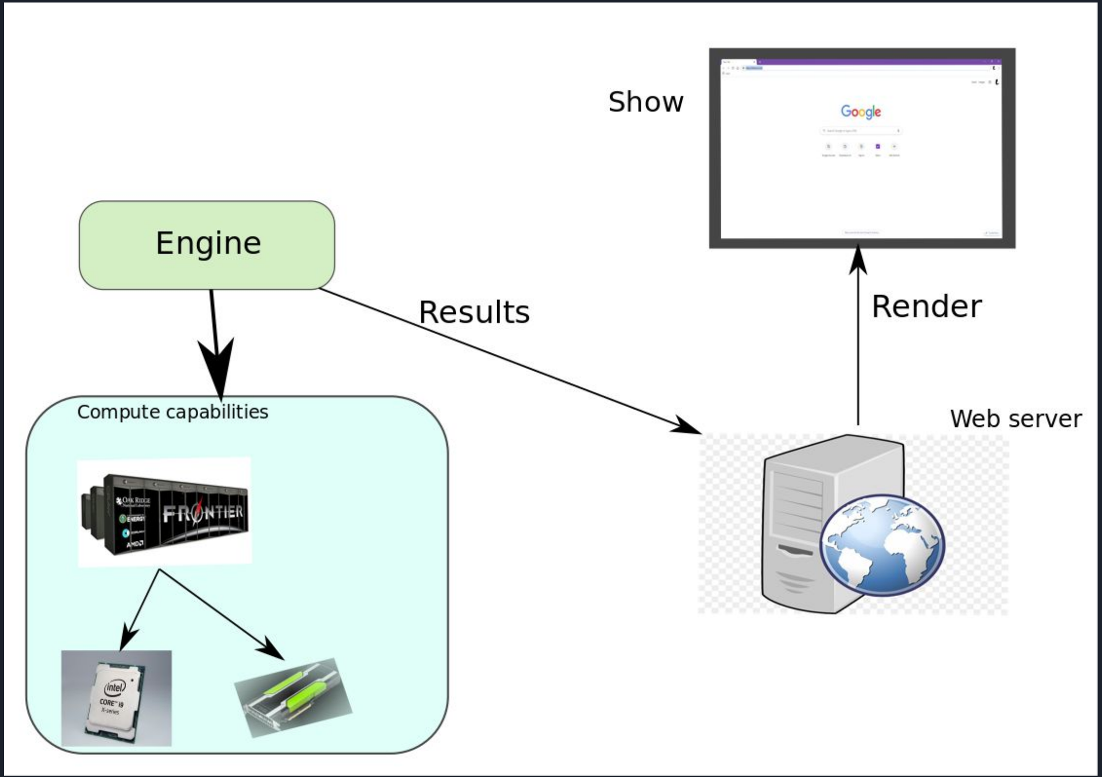
- Render the engine via a web browser, using 3D Javascript technos (WebGL)

- Use compute capabilites for the physics, and use web server for the rendering



Visualize the engine

- Web renderer infrastructure



Work Packages (*)

- WP1 : First working engine
- WP2 : Multithreading support
- WP3 : Web based renderer
- WP4 : Algorithms for collision detection
- WP5 : Orchestrator for collision detection
- WP6 : Implement structure bodies
- WP7 : Handle 3D complex objects
- WP8 : Support soft bodies
- WP9 : Support fluids dynamics
- WP10 : Scene editor
- WP11 : Enhance computer graphics
- WP12 : Use AI to enhance physics
- WP13 : Design a parallel runtime
- WP14 : design a custom HW platform for our engine

(*) Each work package includes a release

Work Packages (*)

- **WP1 : First working engine**
  - Load scene from external source,
    - Scene containing bodies and their properties is loaded in memory
  - Implement the main loop of the engine
    - Iterate the scene at each time step
    - Implement math structures to ensure bodies’ motions
  - Implement a first version of the renderer using OpenGL
    - The 3D scene is displayed
  - Study existing algorithms for basic collisions
    - Broad Phase and Narrow Phase collision detection
  - Implement collisions between monolithic rigid bodies
    - Implement the basic collision detection pipeline (broad phase and narrow phase collision detection)

(*) Each work package includes a release

Work Packages (*)

- **WP2 : Multithreading support**
  - Implement multithreading for physics calculations
    - Adapt code for GPU computing using CUDA language
    - See how we can parallelize narrow phase calculations
      - Parse list of object pairs and perform collision handling, in a parallel manner
  - Depends if we have a GPU available.

(*) Each work package includes a release

Work Packages (*)

- **WP3 : Web based renderer**
  - Implement a web version of the renderer
  - Investigate Javascript technos (WebGL)

- **WP4 : Implement structure bodies**
  - Study existing algorithms handling structures
  - Study collisions between rigid bodies and structures

- **WP5 : Handle 3D complex objects**
  - Handle complex 3D Meshes from external files

(*) Each work package includes a release

Work Packages (*)

- **WP6 : Support soft bodies**
    - Study collisions with soft bodies
        - Rigid body / soft body
        - Soft body / soft body

- **WP7 : Support fluids dynamics**
    - Study interactions between fluids and rigid bodies
    - Study interaction between fluids and soft bodies
    - Study interactions between fluids with different densities / viscosities

(*) Each work package includes a release

Work Packages (*)

- **WP8 : Scene editor**
  - Develop a scene editor GUI
    - Create scene using drag & drop of objects
    - Describe bodies' properties via the GUI
    - Simulate

- **WP9 : Enhance computer graphics**
  - Study Graphic APIs (OpenGL, Direct3D, Metal, ..)
    - Light, Textures, particles
  - Study how to benefit from Ray Tracing

- **WP10 : Use AI to enhance physics**
  - Use Artificial Intelligence to enhance scene realism
    - Study how we can use neural networks to enhance realism of collisions

(*) Each work package includes a release

Work Packages (*)

- **WP11: Design a parallel runtime**
  - Study parallel programming models (MPI, OpenMP, OpenACC, task based models) to parallelize computations
  - Study different parallel paradigms (processes, threads, tasks)
  - Design a custom runtime to implement the programming models

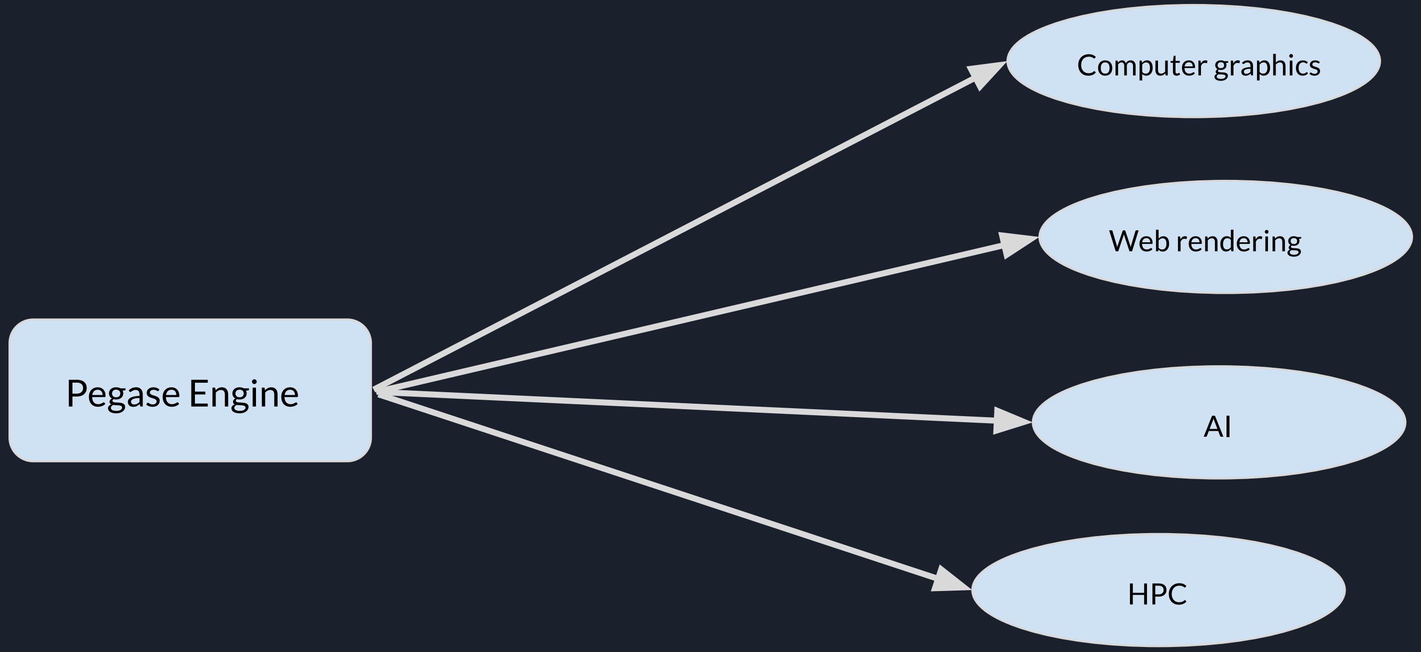
- **WP12: design a custom HW platform for our engine**
  - Study hardware platforms such as FPGA or existing PPUs in order to accelerate physics computations
  - Study recent GPUs or TPUs in order to accelerate neural networks to be implemented in the engine

- **WP13: Orchestrator**

(*) Each work package includes a release

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 !