Development and Parallelization of a Physic Engine

José Ginés Picón López

Director: Domingo Giménez Cánovas

Departamento de Informática y Sistemas

Universidad de Murcia, Spain

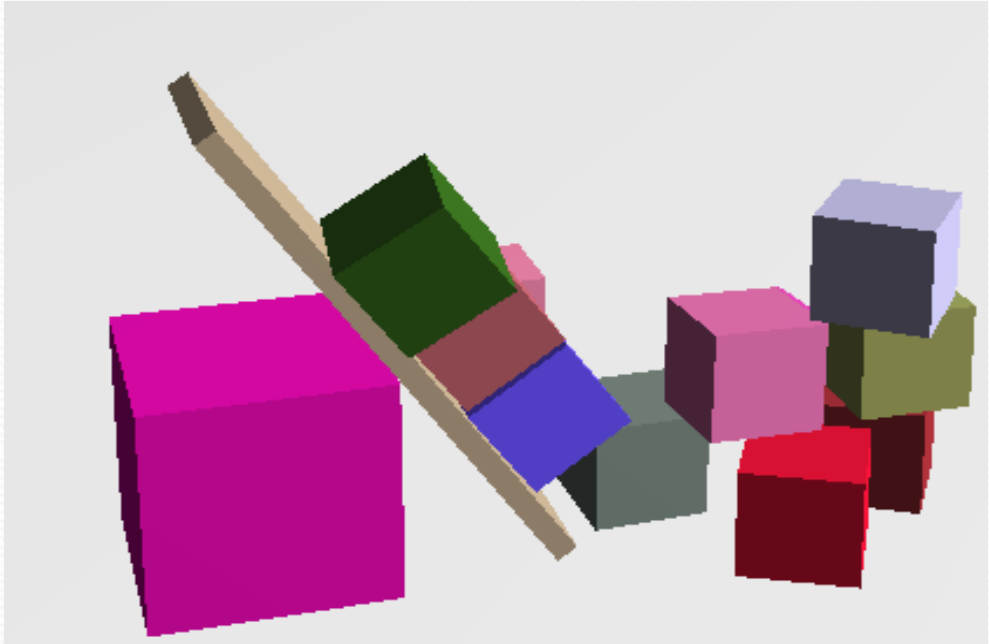
Motivation

- Improve the speed and realism of game physical simulation on next multi-core architectures.
- Achieve physic-based effect on a massive scale.

Snapshot of Midway's Stranglehold

What We are Building

- Fully parallelized *solid-rigid*, *iterative*, *impulse-based* physic engine for real-time applications.
- Optimized for multi-core architectures.



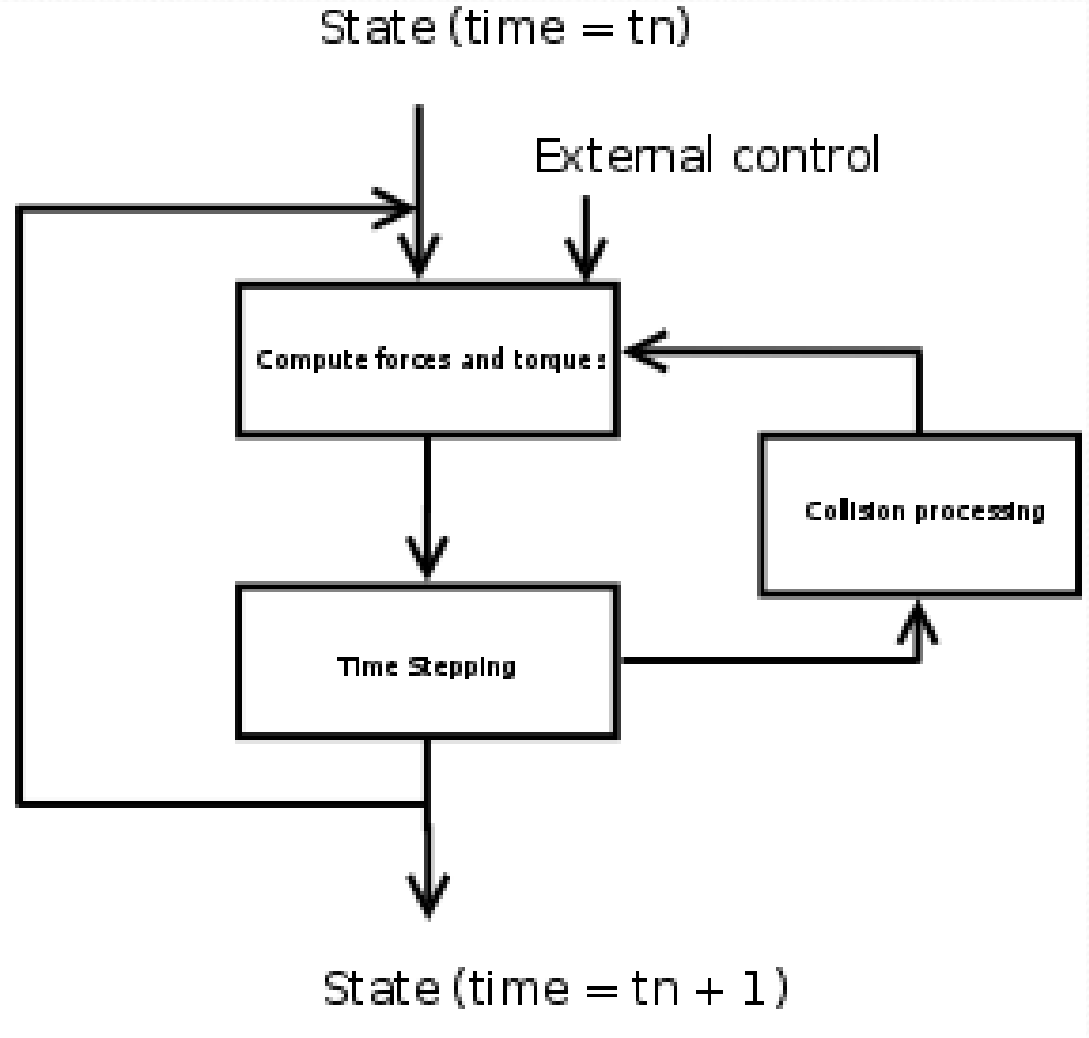
Libraries

- C++ Object-oriented design.
- Use of OpenGL and GLUT.
- OpenMP API for multi-platform shared-memory parallel programming.
- MPI message-passing interface.

Rigid Body Dynamics Overview

- 3 phases to every simulation clock tick.
  - Integrate position and velocities in response to forces and torques applied to.
  - Detect collisions.
  - Resolve collision.
- Integration is easily parallelized because there are no dependencies between objects.
- Collisions require a deeper study.

Physics Simulation Pipeline



Overview of the Engine

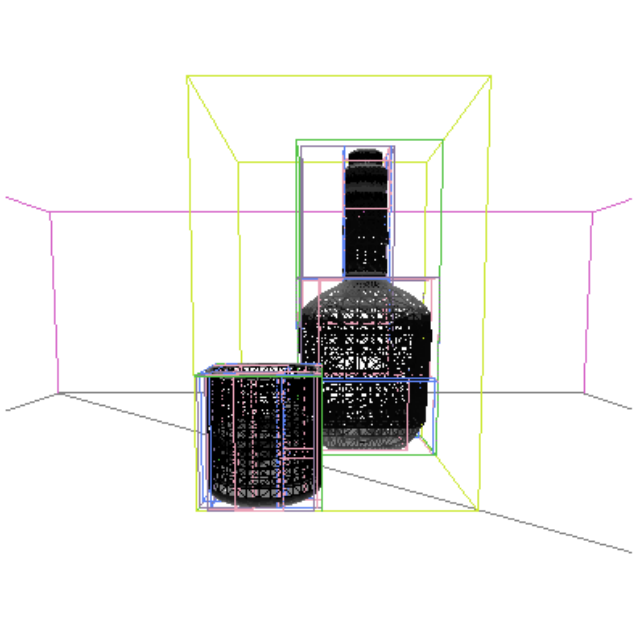
- The engine has four part:
  - ***The force and torque generators*** examine the current state of the game and calculate what forces need to be applied to.
  - ***The rigid-body simulator*** processes the movement in response to those forces.
  - ***The collision detector*** identifies collision and stores a set of contacts.
  - ***The collision resolver*** processes the set of contacts.

Collision Detection

- Collision detection concerns the problems of determining *if*, *when*, and *where* two object come in contact.
- Computer games involve simulation requiring that a large number of queries be performed at frame rates of about 30 to 60 frames per second.
- Collision detection can account for a large percentage of the time it takes to complete a game frame.

Bounding Volumes Hierarchies

- To accelerate collision, simple geometrical objects such as spheres and boxes are initially used to represent objects. More complex objects will be represented forming hierarchies.



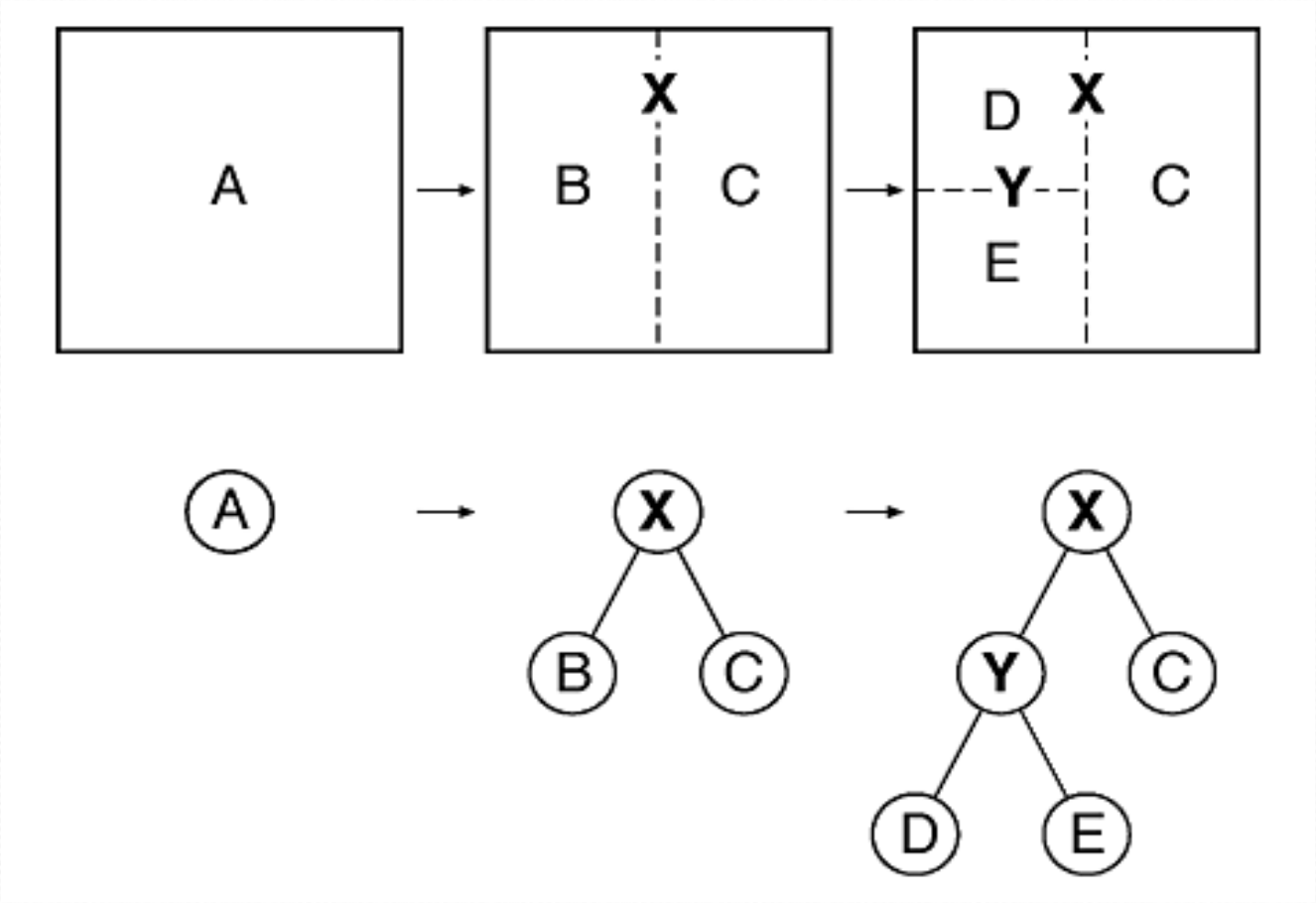
Spatial Partitioning Techniques

- Spatial Partitioning techniques allow us to divide space into regions and testing if objects overlap the same region of space.
- We use a tree structure for representing collision detection: *the binary space partitioning tree* (BSP).
- A BSP can be used to partition space independently from objects in the space.

BSP Trees

- A leaf node of the tree consists of a single object.
- At each step, it checks pairs of nodes.
- If the bounding volumes at the two nodes do not overlap, then none of the objects in the first subtree can collide with any object in the second subtree.

BSP Tree Example



Parallelization

- Initially, the engine will be parallelized using the fork-join paradigm in which the program consist of alternating serial and parallel sections.
- This is attractive because it allows us to start with a serial program and later parallelize portions of the code.
- We will use OpenMP.

Parallelization

- The majority of modules will be parallelized via loop parallelization.
- Each pair of BSP subtrees represents independent computation and can be perform in parallel.
- Sometimes reordering data leads to more parallelism.

References

Baraff, David. Witkin, Andrew.
Physically Based Modeling: Principles and Practice Course Notes.
Siggraph '01 course notes. 2001.
http://www.pixar.com/companyinfo/research/pbm2001/

Davis, Tom. Woo, Mason. Neider, Jackie. Shreiner, Dave.
OpenGL Programming Guide.
Addison-Wesley. 2004.

Intel Technology Journal.
High-Performance Physical Simulation on Next-Generation Architecture with Many Cores.
http://developer.intel.con/technology/itj/index.htm.

Van Verth, James M. Bishop, Lars M.
Essential Mathematics for Games.
Morgan Kaufmann. 2004.

Schneider, Philip. Eberly, David H..
Geometric Tools for Computer Graphics.
Morgan Kaufmann. 2002.