Physics 11

CITM

Course notes

Intro | Before we start…

- Yours truly: David de la Torre, aerospace engineer.

- Class/course **delegate** → spokesperson, main contact for general class stuff.

- Rules during livestream classes:
  - Write freely on the chat (questions/answers). OBVIOUSLY stuff about the course.
  - Do not speak randomly at will over the mic (you are a metric f-ton of people!).
  - Raise hand to speak, I will (almost immediately) give you permission to.
  - DO interrupt me during theory/practice classes to clarify concepts. Nobody knows everything, and that includes me. So don't be afraid to ask; we'll all learn!

Intro | Course contents

- Theory class (livestream): how game physics engines work + build your own physics engine.
  - Basic vector algebra, kinematics, dynamics
  - Integrator
  - Collisions
  - Gravity
  - Aerodynamics, hydrodynamics
  - Harmonic equations (springs)
  - Complex systems (ropes, cloth, fluids, ragdolls, soft bodies, etc.) → just introduction, maybe, we'll see.

- Practice class (on-site): use "commercial" (open-source) physics engines.
  - Box2D (2D physics) → https://box2d.org
  - Bullet Physics (3D physics) → https://bulletphysics.org

Intro | Projects

- Project #1 Apollo: build a 2D space game using your own physics engine.
  - Spaceship (manual impulses/forces) + planet (collisions) + ocean (hydrodynamics) + atmosphere (aerodynamics) + extra dynamics if you want.
- Project #2 Pinball: build a 2D pinball game using Box2D.
- Project #3 Racecar: build a 3D racing game using Bullet Physics.

Intro | Projects

- Projects are done in groups of 2/3 people.
  - SAME GROUP MEMBERS FOR ALL PROJECTS. ‘cause CoViD sucks.
- Focus is on simulation/code quality, then gameplay.
- Tools: Visual Studio, Box2D/Bullet lib and documentation.
- Deliveries should be before 23:59 of the delivery day.
- Every project should be different.

- **Crashing or not playable won’t be accepted for grading.**

Intro | Evaluation

- 25% Project Apollo: Spaceship.
- 15% Project Box 2D: Pinball.
- 30% Project Bullet: Racecar.
- 20% Final exam: theory and practice stuff (physics & coding).
- 10% Attitude (class exercises, presentations, etc.).

- Recovery exam: upgrades the 20% of final exam, if you fail the course.

Intro | Miscellanea

- The course will have:
  - Blackboard-style class, explanations, some equations and cute drawings.
  - Lots of videos & images. Also live gameplays (to teach you stuff, not to play).
  - Streaming sessions will not be recorded → PRINT SCREEN IS YOUR FRIEND.
  - Only simple PPT slides on campus → PAY ATTENTION TO CLASS.
- What do you want to learn? If you have any interesting proposals/requests (games to analyze/play, topics, resources, projects, etc.), let's discuss!
  - Fortnite is BANNED from this course (just kidding not really).

Game physics approach

- Engineering/Science: the physics themselves are the final objective.
  - Precise, accurate simulations: faithfully re-create the behavior of the *real* universe.

- Game development: the physics are just tools subordinated to the game.
  - Fast approximated models: make *your* game universe look good to the player.

"Art is not about being the most accurate; it's about being convincing"

→ Fake it 'till you make it.

Basic algebra stuff: you *should* know this

- How to work with vectors:
  - From vector components to magnitude/angle and vice-versa.
  - Vector operations (add/subtract, normalize, dot product, cross product).
  - Compute normal vector to plane, compute distance point-plane, etc.
  - Code and work with vector variables in C++ → by array type or using classes.

- Reference frames (Euclidean geometry):
  - Definition: origin point + reference plane + principal vector + direction of motion.
  - Change reference frames: translation, rotation, composition.

Kinematics: where are the things

- Point-mass kinematics (**linear** movement):
  - Position x, velocity v, acceleration a: physical meaning (applied to videogames).
  - Derivative of the above quantities: what does it mean?
  - Integral of the above quantities: what does it mean?

- Rigid-body kinematics (**angular** movement):
  - Angular position θ, angular velocity ω, angular acceleration α: physical meaning.
  - Derivative, integral… you know the drill.

- Relationship between the above stuff (*spoiler: linear == angular)

Dynamics: how the things move

- **Newton**'s Laws:
  - 1st law: law of inertia. $F = 0 \rightarrow v = cte$; $\tau = 0 \rightarrow \omega = cte$
  - 2nd law: change of momentum. $F = ma$; $\tau = I\alpha$
  - 3rd law: action-reaction. $F_{A\rightarrow B} = -F_{B\rightarrow A}$; $\tau_{A\rightarrow B} = -\tau_{B\rightarrow A}$
- Conservation (or not) of:
  - Linear/angular **momentum**: $p = mv$; $L = I\omega$
  - Mechanical **energy**: $E_m = E_k + E_p = \frac{1}{2}mv^2 + E_p$; $E_m = \frac{1}{2}I\omega^2 + E_p$
- Apply the above to linear & angular kinematics.

Integrator: move the things

- Implicit Euler (1º): $x \mathrel{+}= v\,\Delta t;$ $v \mathrel{+}= a\,\Delta t$
- Symplectic Euler (1º): $v \mathrel{+}= a\,\Delta t;$ $x \mathrel{+}= v\,\Delta t$
- **Velocity-Verlet** (2º): $x \mathrel{+}= v\,\Delta t + 0.5\,a\,\Delta t^2;$ $v \mathrel{+}= a\,\Delta t$
  - Störmer-Verlet → $v \cong \bar{v}_{now} = \frac{\bar{v}_{old}+\bar{v}_{new}}{2};$ $\bar{v}_{old} = \frac{x_{old}+x}{2};$ $\bar{v}_{new} = \frac{x+x_{new}}{2}$

    …after some algebra: $x \mathrel{+}= 2x - x_{old} + a\,\Delta t^2.$
- Runge Kutta (4º, 6º, 8º, etc.): uses accurate derivatives, but it is slower*.
  - Adaptive RK: automatically check integration error and pick the best order.

- Pros/cons of each method, relationship with framerate ($\Delta t = 1/FPS$).

Framerate: how the things flicker on screen

- VIDEOGAMES ARE NOT CONTINUOUS → DISCRETE FRAMES.
- Strategies to set/determine FPS: **fixed, variable, semi-fixed,** etc.
    - Advantages, drawbacks, random shenanigans.
    - How to implement each.
- **Sub-stepping**: Why? When? How?
- The spiral of death: Why? How to avoid it.

Collisions: how the things crash

- Physical meaning of collisions: why cannot we simulate/compute them directly?
  - Official solution → conservation of momentum and energy → system of equations.

- The above is useless → how do we solve collisions in videogames?
  - **Simple** collision with static/kinematic object (+ damping) → gg ez.
  - **Multiple** dynamic collisions → like playing dark souls with a banana controller. Blindfolded.
    - Solve via position/velocity or via impulses.
    - Iterative strategies vs. Gauss-Seidel-like methods.

- **Tunneling** effect (bullets and projectiles): how to avoid it!

Gravity: how the things… ¿fall?

- **General** gravitational force → planets, spaaaaace, etc. → $F_g = -G\frac{m_1 m_2}{r^2}\vec{r}$
  - **Local** gravity force → the above when on ground → $F_g \cong mg$

- Center of mass: where the gravity force is applied.
- How to implement gravity into videogames: cheats and shortcuts.

- Gravity is NOT CONSTANT. Constant is boring. Do not be boring.

Aerodynamics: how the things fly

- Computational fluid dynamics: oh, you want to simulate this? Ha-ha-ha-no.

- Simplified aerodynamics: equations of **Lift** and **Drag**.
  - Lift: $L = \frac{1}{2}\rho v^2 S C_L$ → makes things fly, sometimes.
  - Drag: $D = \frac{1}{2}\rho v^2 S C_D$ → makes things go slower, and dat is gut.
  - Location of the above forces: center of pressure.

- When to use Lift/Drag. Effects on game physics.

Hydrodynamics: how the things float

- Computational fluid dynamics: …yeah, you know the drill.

- Simplified hydrodynamics: Lift, Drag, **Buoyancy**.
  - Lift: same as aero, but not really. And only if you go faaaast.
  - Drag: mainly Stokes' drag: $D = -bv; \quad b = wikipedia^{TM}$.
  - Buoyancy (makes things float): $F_b = \rho g V - mg$.
  - Location of the above forces: center of pressure.

- When to use all of the above. Effects on game physics.

Springs: how the things boing-boing

- Hooke's law (force of a spring): $F_k = -k\Delta x$

- Peasant solution: simple harmonic oscillator ($x = A\cos\omega t$) → boring.
  - Use damped harmonic oscillator solution (recommended): $x = Ae^{-\beta t}\cos\omega t$.
  - No other external forces allowed on the body (not even gravity): isolated system!
  - Mostly used for hair, other boing-boing body parts, and dynamic pseudo-animations.

- Pro-gamer solution: cram F_k into the integrator → may explode. Cool.
  - (In-)accuracy of the integrator, how to fix. Apply damping.

Complex systems: how the things ragdoll

- Kinematic or dynamic restrictions between bodies: make stuff stick together.
  - Types: fixed/welded, pivot/revolute/hinge, distance, linear/slider/prismatic, 6dof, etc.

- How to solve:
  - Iterative methods: solve conditions, one by one, until "convergence" (= you ran out of Δt).
  - System of equations: solve all conditions at the same time.
    - Gauss-Seidel, Jacobi method, SOR/SSOR, etc.
  - A.I./Neural networks: because overkill is fun.

- Examples: rope, cloth, soft-body, structures, pseudo-fluids, ragdolls, etc.

Network: how the things do online physics

- Online games: lots of clients → need synchronization!
  - To avoid cheating → physics are run by the server, then broadcasted to clients.
  - Clients send user input info, receive world state updates → no physics on clients?
- Problem: massive lag in user input:
  - Player presses W → 2 seconds later the character starts moving. Unacceptable!
- Solution → predictive simulation:
  - Client sends user inputs to server, but also starts processing the physics by itself.
  - After a while, the server sends to client any corrections as required → sync, sync, sync!

Experimental physics: the future things

- Voxel physics: Minecraft on steroids.
  - Same physical principles and equations, but everything is 3D cube objects.
  - VERY good for destruction-based games. Might get CPU-expensive if not careful.
  - https://www.youtube.com/watch?v=zo0mOmv9jW8

- Machine learning: make Skynet do physics for you.
  - Replace physics equations by neural networks & co.
  - Lots of SLOW training using faithful simulations first → but only need to do it once.
  - Once trained, NN are FAST. May fail on unexpected/untrained scenarios.
  - https://sites.google.com/view/learning-to-simulate/

SIGGRAPH: the even more future things

- Where the cutting-edge research on CG (not only game physics) is presented:
  - https://www.youtube.com/watch?v=EhDr3Rs5fTU
  - https://www.youtube.com/watch?v=jYdMKdRUq_8
  - https://www.youtube.com/watch?v=iOUr536wtUs
  - https://www.youtube.com/watch?v=y96jk-eUCgI
  - https://www.youtube.com/watch?v=16CHDQK4W5k
  - https://www.youtube.com/watch?v=vn-WzWm74Pc
  - https://www.youtube.com/watch?v=eXJi7pkUZn0
  - https://www.youtube.com/watch?v=Rzj3k3yerDk

GDC: how game developers do the things

- Actual game developers presenting and discussing their work in detail.
- Miscellaneous topics; some of them are physics/animation-related.
- Check out especially the animation and the programming talks.
- Learn from the work of the masters!

https://www.youtube.com/channel/UC0JB7TSe49lg56u6qH8y_MQ