CMSC 335
COMPUTER GRAPHICS

LECTURE 11

• PHYSICALLY-BASED ANIMATION
ANIMATION

• **Animation** is the act of imparting life, motion, etc. into a scene

• Many approaches exist for animation (e.g., keyframe or motion capture)

• Many techniques are used to adjust animations (e.g., timing or deformations)
APPLICATION LOOP WITH DELTA-TIME AND FRAME LIMITING

1. target_frame_time = 0.17f
2. while isRunning() do
3.   realdt = time - last_time
4.   appdt = realdt * app_time_factor
5.   // Process inputs
6.   // Update world with appdt
7.   // Generate outputs
8.   // Frame limiting
9.   while realdt < targetft do
10.   doSomethingSmall()
PHYSICALLY-BASED ANIMATION

- **Physically-based animation** solves the differential equations of motion for an object.

- Physics in games involves these two basic elements:
  - Object-object interaction (Geometry)
    - Collision detection
    - Collision response
  - Mechanics (Calculus)
    - Object movement
REVIEW OF LINEAR (NEWTONIAN) MECHANICS

• Newton's second law of motion – force is mass by acceleration
  \[ \vec{F} = m\vec{a} \]

• For position \( \vec{x} \):
  • Velocity \( \vec{v} = \dot{\vec{x}} \) (first derivative with respect to time)
  • Acceleration \( \vec{a} = \dot{\vec{v}} = \ddot{\vec{x}} \) (second derivative with respect to time)

• In games however, we are trying to compute the next time steps position \( \vec{x}' \), thus, we need the anti-derivative, i.e., integration
EULER INTEGRATION

• Euler integration – use current velocity to alter position
  \[ \hat{x}' = \hat{x} + \hat{v} \Delta t \]
  \[ \hat{v}' = \hat{v} + \hat{a} \Delta t \]

• Semi-implicit Euler integration – use next velocity to alter position
  \[ \hat{v}' = \hat{v} + \hat{a} \Delta t \]
  \[ \hat{x}' = \hat{x} + \hat{v}' \Delta t \]
PHYSICALLY-BASED ANIMATION
Essentially, physically-based animation simulates the evolution of state.

- **State** is defined as all of the properties that change over time, encoded as a vector $\vec{x}$:
  - e.g., position, velocity, color, etc.
- Differential equations describe the change in state.
Example

- Point mass with gravity
  - State $\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix}$ and mass $m$
  - Force of gravity: $\ddot{F} = m \ddot{g} = m \ddot{a}$

- Explicitly solving for position:
  - $\ddot{v} = \int \ddot{g} \, dt = \dddot{v}_0 + \dddot{g} t$
  - $\ddot{p} = \int (\dddot{v}_0 + \dddot{g} t) \, dt = \dddot{p}_0 + \dddot{v}_0 t + \frac{1}{2} \dddot{g} t^2$

- Typically, equations are not this nice and we apply Euler Integration over a small enough time step
  - $\dddot{x}' = \dddot{x} + \dddot{x} \Delta t = \begin{bmatrix} \dddot{p} \\ \dddot{v} \\ \dddot{g} \end{bmatrix}$
SIMULATION

• Since we will not typically have a representation of the equations, at each time step:
  • Calculate the forces acting upon a body
  • Compute the acceleration
    \[ \ddot{a} = \frac{\vec{F}}{m} \]
  • Numerically integrate
• Collisions complicate this process a bit
SIMULATION WITH COLLISIONS

- Physics loop
  \[ t \leftarrow \Delta t \]
  \[ \text{while } t > 0 \text{ do} \]
  \[ \text{setAccelerationFromForces()} \]
  \[ \ddot{x}_{n+1} \leftarrow \dot{x}_n + \ddot{x}_n t \]
  \[ \text{if collision}(\dddot{x}_n, \dddot{x}_{n+1}) \text{ then} \]
  \[ c \leftarrow \text{firstCollisionTime()} \]
  \[ \ddot{x}_{n+1} \leftarrow \dot{x}_n + \ddot{x}_n c \]
  \[ \text{collisionResponse()} \]
  \[ t \leftarrow t - c \]
  \[ \text{else} \]
  \[ t \leftarrow 0 \]
  \[ \ddot{x}_n \leftarrow \ddot{x}_{n+1} \]
RESTING

- It may be surprising to know that getting an object to stop and rest is surprisingly difficult, because of numerical error and being in a state of many collisions
  - Velocity threshold or distance threshold
  - Analyze acceleration in direction of normal compared with a threshold ($|\vec{a} \cdot \hat{n}| < h$)
- Important that at rest an object is no longer simulated to reduce computation cost
Air resistance $F_a = -d\ddot{v}$
- Opposite direction as velocity
- $d \in [0,1]$ is a viscosity constant
- Can also be related to velocity squared

Wind $F_w = d\ddot{w}$

Friction response to collisions
PARTICLE SYSTEMS

• Set of independent point-masses
  • State should include position and velocity of each particle
  • Can include temperature, age, color, etc

• Pre-allocate memory for the system, e.g., an array of size 10000.

• Used to simulate affects, e.g., water, fire, smoke, etc.
GENERATORS

• Initial conditions for the particles need to vary and we want to regenerate particles after they "die"

• Generators create values/vectors based on a distribution
GENERATORS

• Distributions
  • Uniform – all values within a range \([a, b]\) are equally likely
  • Gaussian (normal) – defined by a mean \(\mu\) and variance \(\sigma^2\)
**EXAMPLE GENERATORS**

- **Point generator**
  - $\vec{p}_0 = \vec{p}$ | $\vec{p}$ is an input value
  - $\vec{v}_0 = \vec{v}\vec{r}$ | $\vec{v}$ is input and $\vec{r}$ is a random vector

- **Directed generator**
  - $\vec{p}_0 = \vec{p}$ | $\vec{p}$ is an input value
  - $\vec{v}_0 = \vec{v}(\vec{r} + \hat{n})$ | $\vec{v}, \hat{n}$ are input and $\vec{r}$ is a small random vector

- **Disc generator**
  - $\vec{p}_0 = \vec{r}$ | $\vec{r}$ is a random point within a circle
  - $\vec{v}_0 = \hat{n}$ | $\hat{n}$ is input

- **Get creative!**
ATTRACTION AND REPULSION

• Gravity toward a point
  • \( \mathbf{F}_g = -\frac{g m_1 m_2}{r^2} \hat{u} \), where \( \hat{u} \) is direction away from the point (or \( \hat{u} \) is toward and \( g \) is negative)
  • Alterations:
    • Ignore second mass
    • Be proportional to \( \frac{1}{r} \)

• Repulsor from a point
  • \( \mathbf{F}_g = \frac{g m_1 m_2}{r^2} \hat{u} \)
  • Anything goes, e.g., towards/away from lines, potential fields, etc
FLOCKING SYSTEMS

- Point-masses that interact
- Need to determine set of neighbors to interact with and forces from those interactions
- Example would be simulating crowds of people
SPRING-MASS SYSTEMS

• Point-masses connected by springs
• Need to use Fourth Order Runge-Kutta for numerical stability
• Examples would be cloth or hair simulations
RIGID BODY MECHANICS

- Volumes with rotational physics, i.e., torques and angular velocities
- Need good collision detection and response mechanism
- Very useful, e.g., fighting mechanics or cars in racing games
IMPLEMENTATION CONCERNS
INSTANCED RENDERING

• Instanced rendering, uses the same model (vertex buffer) to render at various transforms (stored in a different buffer)

• Much more efficient than separate model rendering
COMPUTE SHADERS

• The programmable rasterization pipeline allows for general computations to occur

• A compute shader allows easy parallelization of the integration, etc.
RECOMMENDED TEXTS