Simulation in Games

Digital Media & Games

Derek Groen

Contents: 1st half

- Introduction to simulations.
  - What is a simulation?
  - Games are simulations.
- Why do we simulate?
  - Science perspective.
  - Gameplay perspective.
- Example of games used as serious simulations.

Contents: 2nd half

- Integrating simulation in games, an example:
  - Introducing the physics engine.
- Avenues for further enrichment of the game using simulation.
- Bonus: data-driven gaming.

What is a simulation

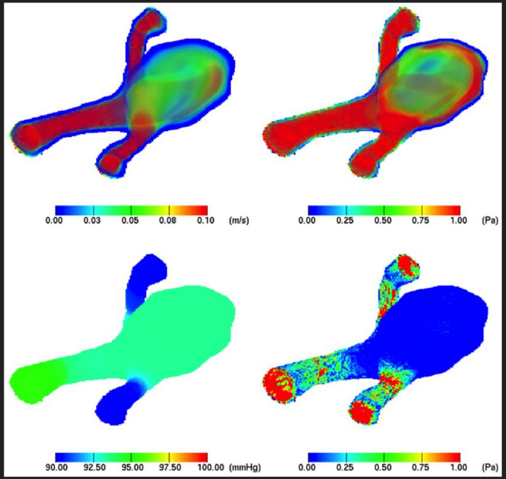
“Simulation is the imitation of the operation of a real-world process or system over time.

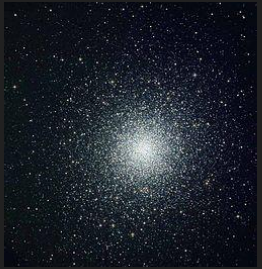
The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process.”

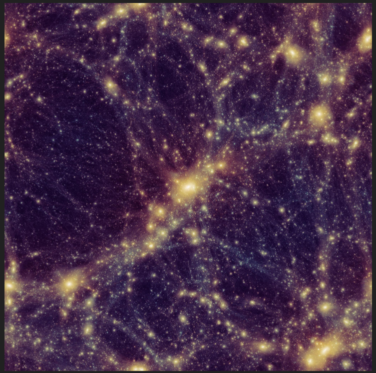
- Wikipedia.org

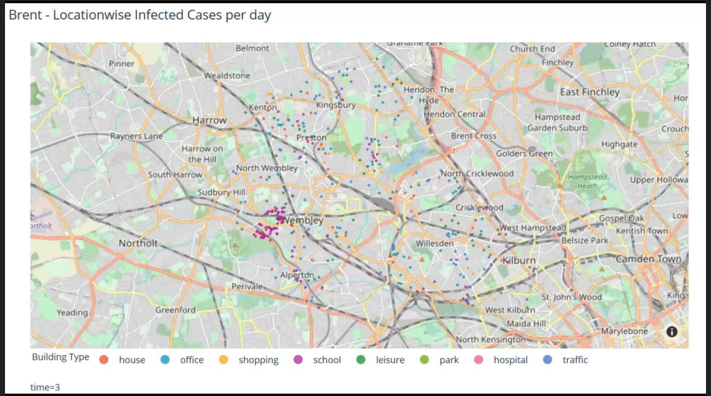
Simulations and research

- Simulations are of major importance in scientific research.
- >50% of research activities involve some kind of computation.
- Simulations can be combined to create more advanced simulations.



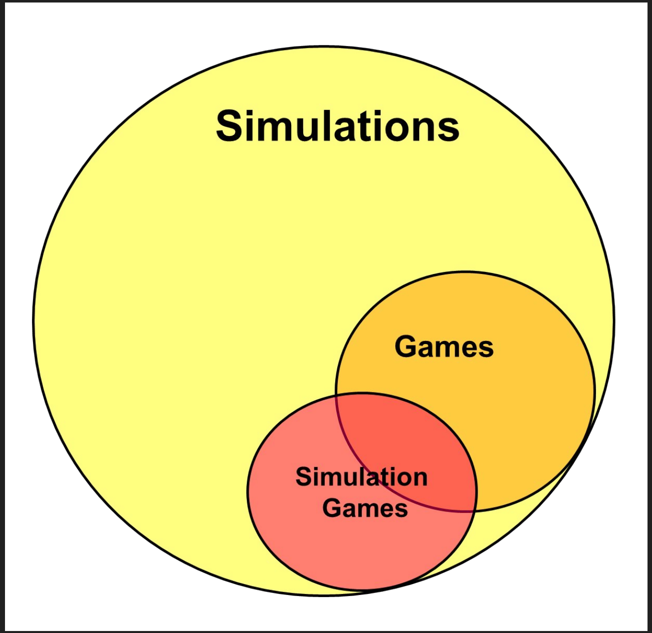






Games are simulations

- Games are usually an engaging rendition or combination of real-life environments.
    - Real life environments are familiar to players, and therefore work well to immerse the player in a game.
- Simulation games focus primarily on the simulation aspect.



Simulation and games

- But effective simulation is key to make any game fun and engaging.
- Be it simple...

Game: Chess

- Or complex...

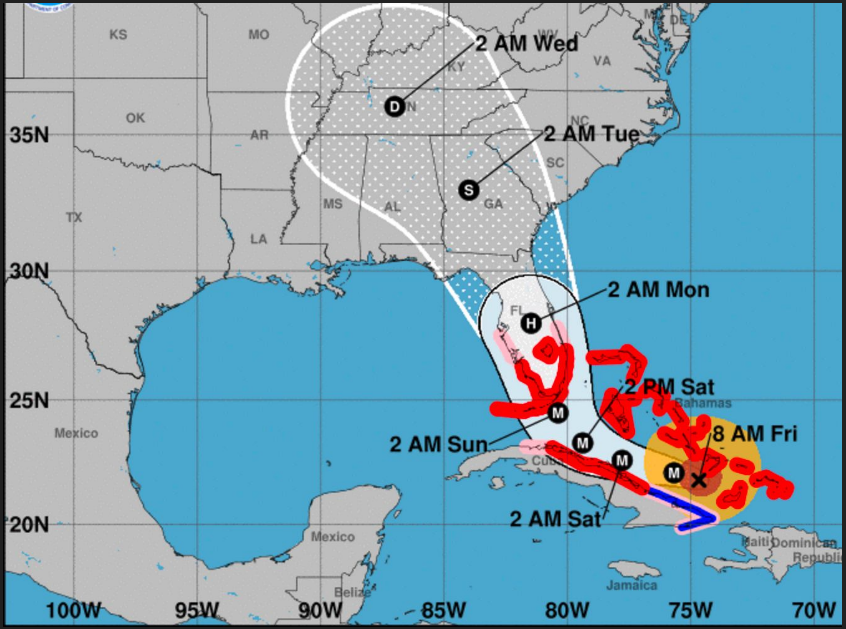
Game: Saints Row Part 5: Agents of Mayhem (PC)

The purpose of simulation in research

Why do we bother?

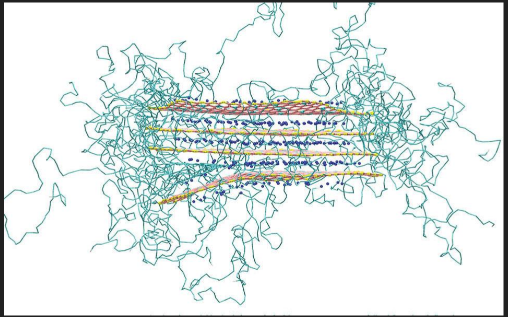
The purpose of simulation (research)

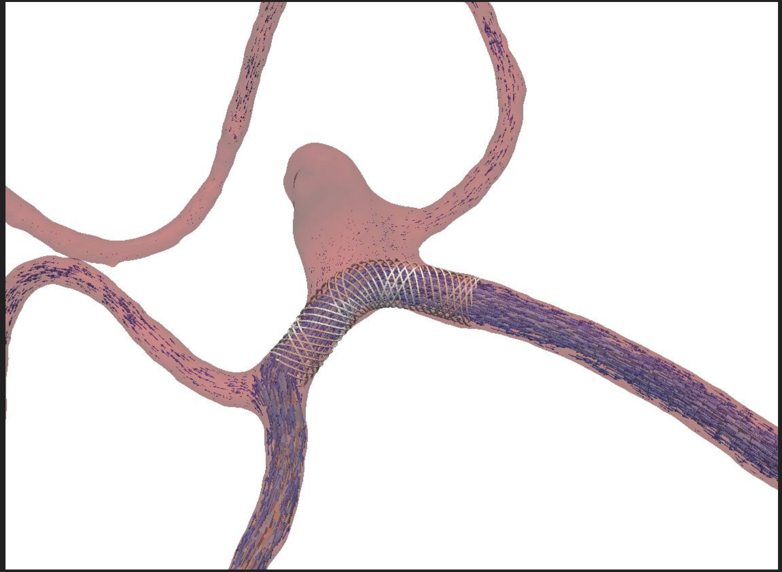
- Validating theory against observations,
  - e.g., validating Newton's law against star cluster observations.
- Predicting upcoming events,
  - e.g., weather/disaster prediction.



The purpose of simulation (research)

- Try to understand effects of (yet) non-existent scenarios, e.g.
  - what effect would treatment X have on patient Y?
  - How would Covid-19 spread change if we entered a full lockdown today?
- Quickly explore a wide range of possibilities,
  - ensembles.





The purpose of simulations (gameplay)

- Make game environment and settings more realistic.
  - E.g., Dwarf Fortress simulates a full world history process.
- Create realistic game responses.
  - See video in the break.
- Create seamless aging effects of structures and/or people.

Game: Sims IV (right with mod)

The purpose of simulations (gameplay)

- Add intelligence to the non-player entities (more on that in the next lecture).
- Enable animations to be more dynamic/interactive than with pre-recorded animations.
- Have complex materials interact realistically.
  - E.g., hair waving in the wind.

The purpose of simulations (gameplay)

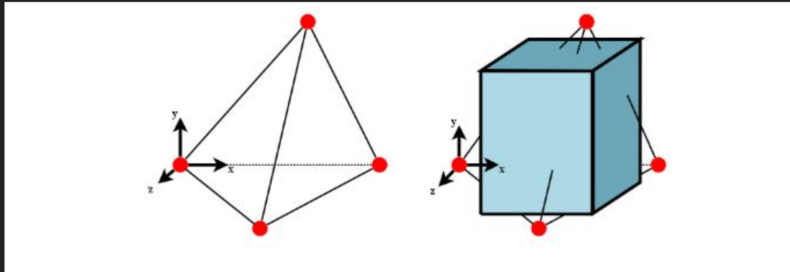


Figure 8: A tetrahedron and box defined using a local coordinate system. The local coordinate systems makes it possible to place the box independently on the particles positions.

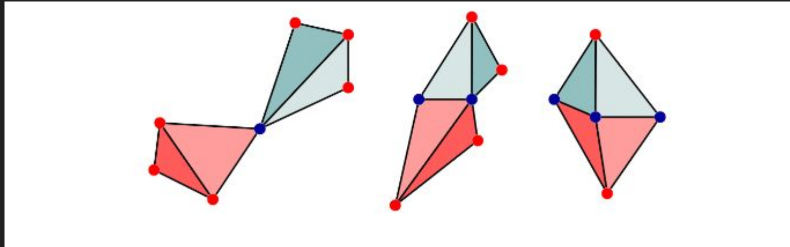
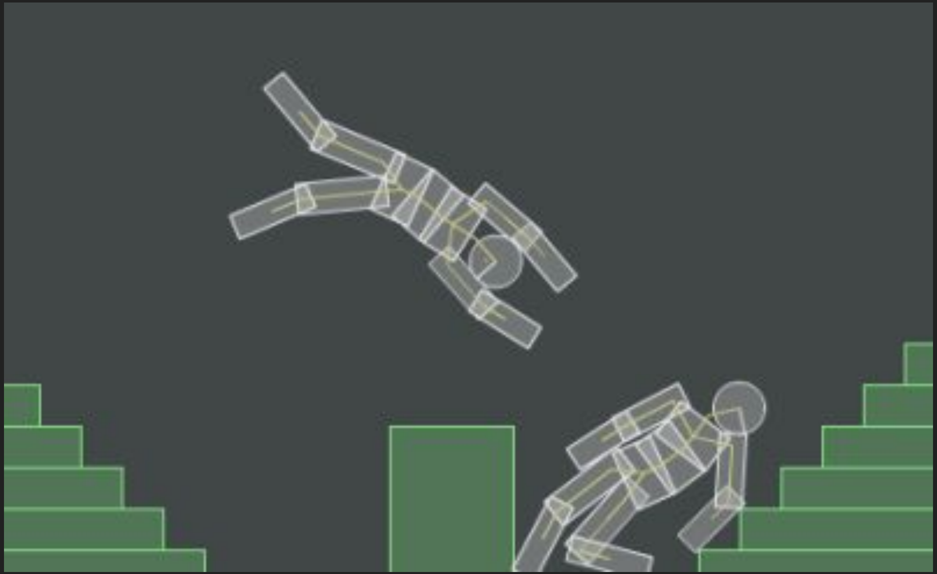


Figure 9: Three ways to model joints using particles. The first is a ball joint where the two boxes share one particle. Second is a hinge joint where two particles are shared. The last shows three shared particles which yield one rigid object.

Comparison of Ragdoll methods - http://image.diku.dk/projects/media/glimberg.engel.07.pdf

- Realistic death scenes.



Engine: Box2dFlashAS3

The purpose of simulations (gameplay)

- Realistic death scenes.

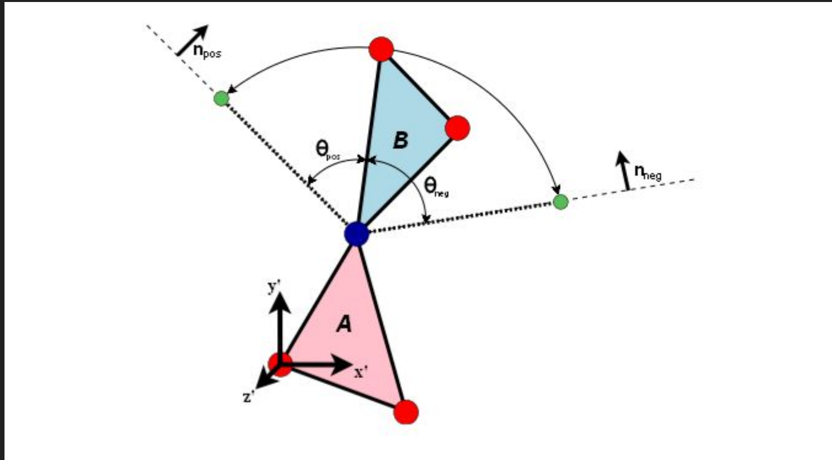
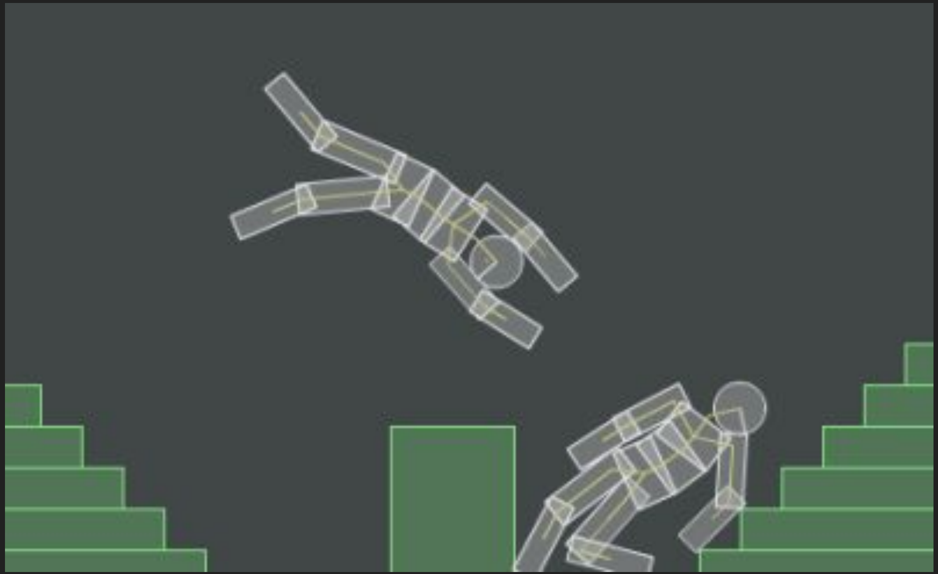


Figure 11: The confinement planes can be calculated by rotating the reference particle in B by the given angle limits.



Comparison of Ragdoll methods -
http://image.diku.dk/projects/media/glimberg.engel.07.pdf

Engine: Box2dFlashAS3

Can games be used as simulations?

- Fifa in scouting and football management.
- Sim City and smart cities…?
- Plague Inc and epidemiology…..?

- Tough trade-off between realism and entertainment: games can’t always be trusted.

Screenshot of Plague Inc.

One for the break

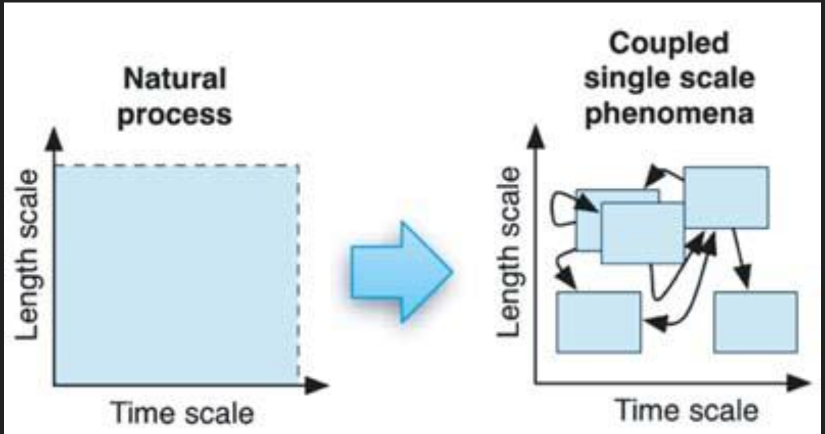
https://www.youtube.com/watch?v=2z__y42rir8

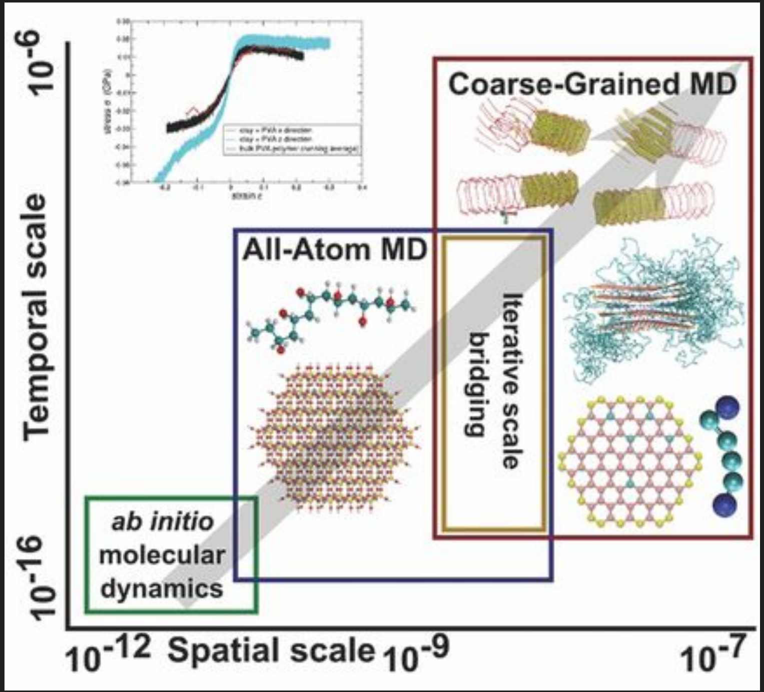
Contents: 2nd half

- Basics simulation concepts.
- Integrating simulation in games.
- Common types of simulation.
- Avenues for further enrichment of the game using simulation.
- Bonus: data-driven gaming.

Basic simulation concepts

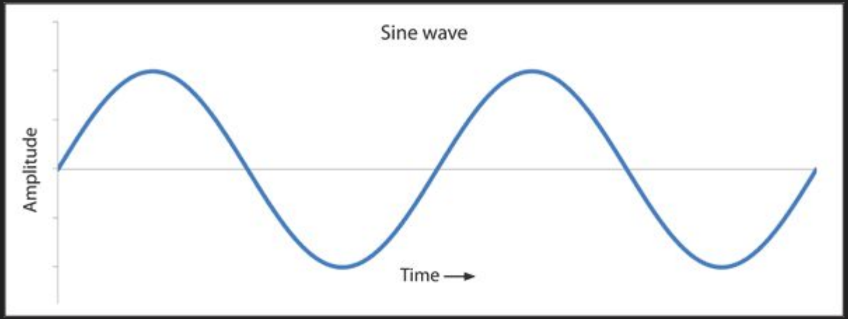
- Simulations modify a *system* over a length of *time*.
- Two key concepts:
    - Length scale
    - Time scale



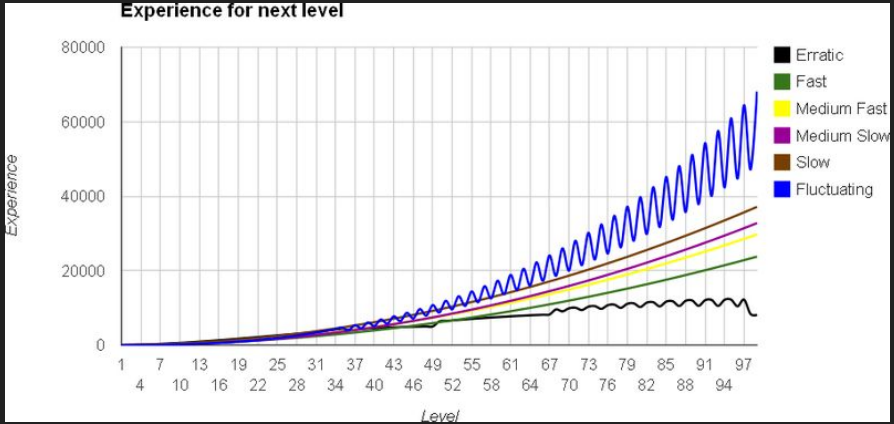


Basic simulation concepts: simple functions

E.g., wave functions which can work well for price fluctuations over time.



Curves for level progressions.

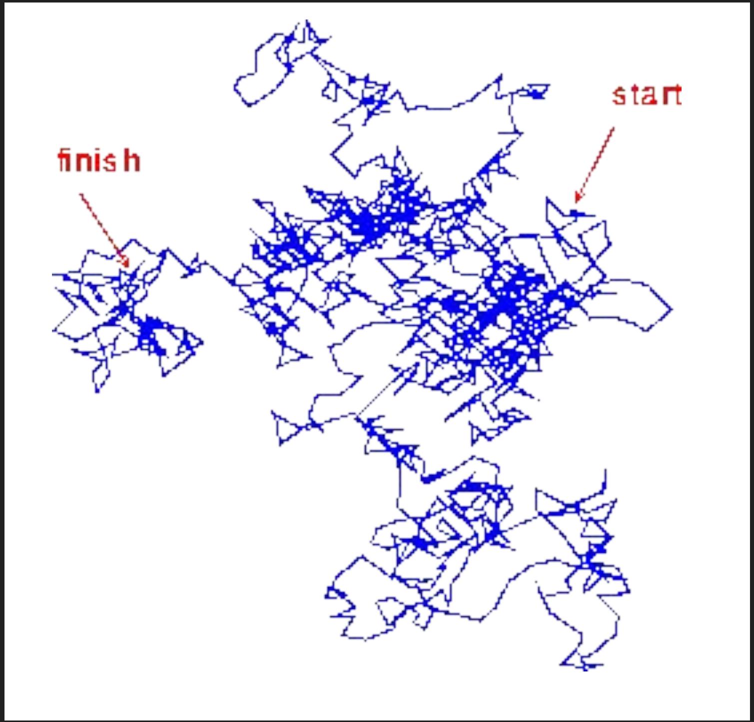


Game: Pokemon series

Basic simulation concepts

Random walk

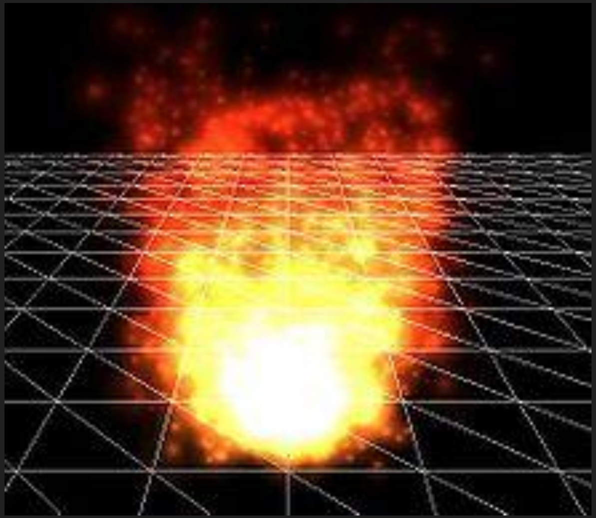
(more on this during the AI in Games lecture!)



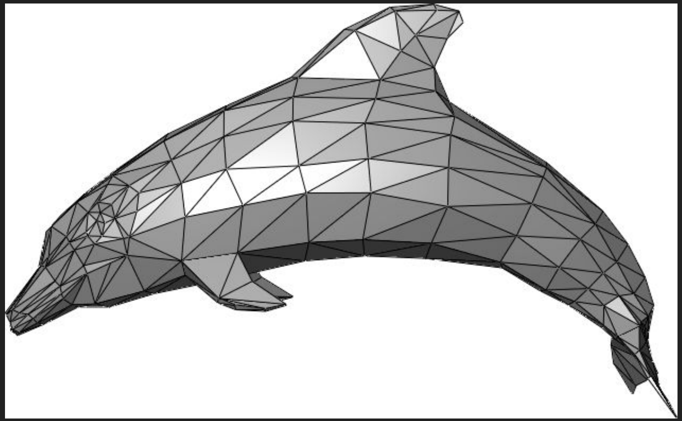
Basic simulation concepts: particle & mesh

Particles work well for explosions, gravity, gases, and some types of fluids.

Meshes work well for shapes, solids and some type of fluids.



Engine: 3dengfx



Wikipedia (user: Chrschn)

Basic simulation concepts

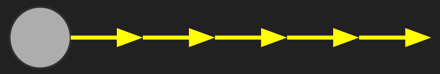
- Time is advanced *step-by-step*, a simple example in Python:

```
x = SimulationSystem()
t_end = 10.0

while t < t_end:
  x.evolve(duration=1) # do one simulation step.
  t = t+1
  # Now the system has evolved one time step.
# Here in the code the system has evolved to t=9.
```

Example: moving ball

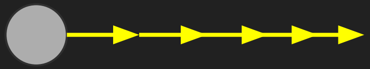
```
t = 0, t_end = 10, x = 0, y = 10,
vx = 10 # m/s
vy = 0  # m/s
while t < t_end:
  print(t,x,y)
  x = x + 1*vx
  y = y + 1*vy
  t = t+1
  # Now the system has evolved one time step.
# Here in the code the system has evolved to t=9.
```



```
Output (t,x,y):
0 0  10
1 10 10
2 20 10
3 30 10
```

Example: moving ball with air resistance

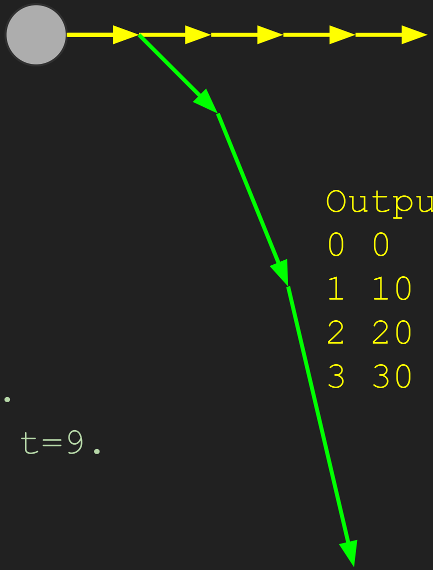
```
t = 0, t_end = 10, x = 0, y = 10,
vx = 10 # m/s
vy = 0  # m/s
air_drag_coefficient = 0.1 # [m/s^2]
while t < t_end:
  print(t,x,y)
  x = x + 1*vx
  y = y + 1*vy
  t = t+1
  vx = vx * (1.0 - air_drag_coefficient)
```



```
Output (t,x,y):
0 0    10
1 10   10
2 19   10
3 27.9 10
```

How do you introduce gravity?

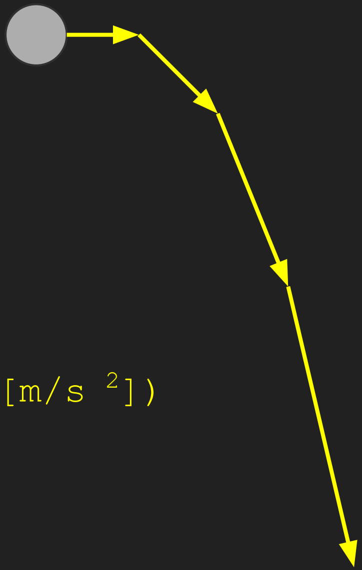
```
t = 0, t_end = 10, x = 0, y = 10,
vx = 10 # m/s
vy = 0  # m/s
while t < t_end:
  print(t,x,y)
  x = x + 1*vx
  y = y + 1*vy
  t = t+1
  # Now the system has evolved one time step.
# Here in the code the system has evolved to t=9.
```



```
Output (t,x,y):
0 0  10
1 10 10
2 20 10
3 30 10
```

Example: moving ball with gravity

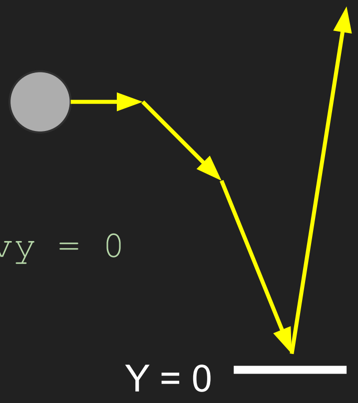
```
t = 0, t_end = 10, x = 0, y = 10,
vx = 10 # m/s
vy = 0  # m/s
while t < t_end:
  print(t,x,y)
  x = x + 1*vx
  y = y + 1*vy
  t = t+1
  vy -= 9.81 # Gravitational force (G=-9.81 [m/s^2])
```



```
Output (t,x,y):
0 0  10
1 10 10
2 20 0.19
3 30 -19.43
4 40 -48.86
```

Bouncing against a surface

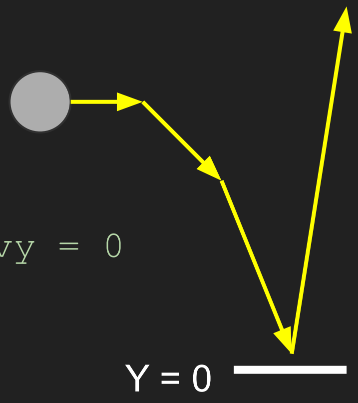
```
e = 1.0 # no energy loss in bounce
ball_radius = 2
t = 0, t_end = 10, x = 0,  y = 20, vx = 10, vy = 0
while t < t_end:
  print(t,x,y,vy)
  x = x + 1*vx
  y = y + 1*vy
  t = t+1
  vy -= 9.81 # Gravitational force (G=-9.81 [m/s ²])
  if  y < ball_radius:
    vy = -vy*e
    y = ball_radius
```




```
Output (t,x,y,vy):
0 0  20    0
1 10 20    -9.81
2 20 10.19 -19.62
3 30 2     27.43
4 40 29.43 17.62
```

Bouncing against a surface

```
e = 1.0 # no energy loss in bounce
ball_radius = 2
t = 0, t_end = 10, x = 0,  y = 20, vx = 10, vy = 0
while t < t_end:
  print(t,x,y,vy)
  x = x + 1*vx
  y = y + 1*vy
  t = t+1
  vy -= 9.81 # Gravitational force (G=-9.81 [m/s ²])
  if  y < ball_radius:
    vy = -vy*e
    y = ball_radius
```



Output (t,x,y,vy):

0	0	20	0
1	10	20	-9.81
2	20	10.19	-19.62
3	30	2	27.43
4	40	29.43	17.62

What happens if we change E? And if we change G?

Gravity and super mario

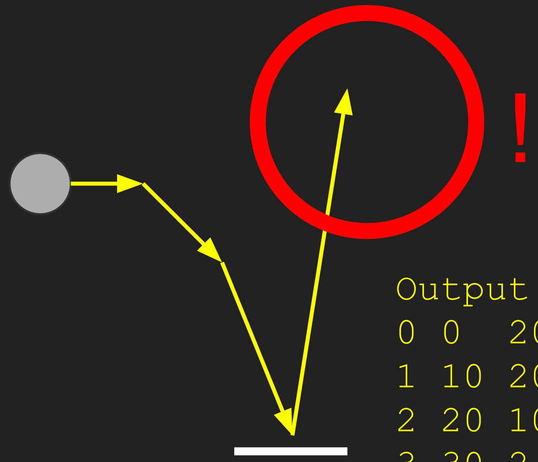
- Since we’re doing games, not science, we can play with constants.

Game	Frames	Time (s)	Height of Mario (pixels)	Distance of Fall (pixels)	Distance of Fall (m)	Acceleration (m/s^2)	Acceleration (g)
Super Mario Bros.	15	0.5	39	292	11.4	91.28	9.31
Super Mario Bros. 2	12	0.4	45	255	8.6	107.95	11
Super Mario Bros. 3	15	0.5	35	265	11.5	92.31	9.42
Super Mario World	15	0.5	38	193	7.7	61.92	6.32
Super Mario 64	10	0.33	86	217	3.8	69.22	7.06
Super Mario Sunshine	23	0.77	119	988	12.7	43.05	4.4
Super Paper Mario	12	0.4	288	748	4	49.47	5.05

http://hypertextbook.com/facts/2007/mariogravity.shtml

Time step issues

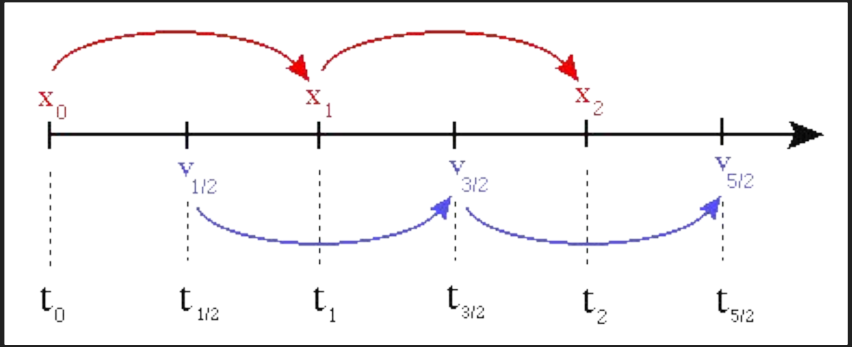
- Here is where our simple code goes particularly wrong.




```
Output (t,x,y,vy):
0 0  20    0
1 10 20    -9.81
2 20 10.19 -19.62
3 30 2     27.43
4 40 29.43 17.62
```

Time step issues

- Two main ways to make particle codes more accurate:
  a. Use a smaller timestep.
  b. Use a more accurate *integration scheme*.
- Common integration schemes:
  - Euler (shown earlier) → 1st order accurate.
  - Leapfrog → 2nd order accurate.
  - Runge-Kutta → 4th order accurate.



Leapfrog method (source: drexel.edu)

Advanced example: N-body with leapfrog

The gravitational N-body problem

read t=0 snapshot from input

calculate energy dt= 0.001

initialize t_{end} = 1

WHILE $t < t_{end}$ **DO**

Calculate $a_i(r_i)$

Advance positions $r_i \rightarrow r_{i+1}$

Calculate $a_{i+1}(r_{i+1})$

Advance velocities $v_i \rightarrow v_{i+1}$

t += dt

nsteps++

IF (nsteps%10 = 0)

calculate energy

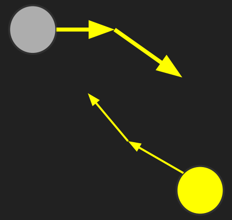
print diagnostics (energy, energy error)

ENDIF

END WHILE

dump final snapshot

Source: www.nbabel.org



$$\mathbf{F}_i \equiv m_i \mathbf{a}_i = m_i G \sum_{j=1, j\neq i}^{N} m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}.$$

Newton's Second Law

Game: Super Mario Galaxy

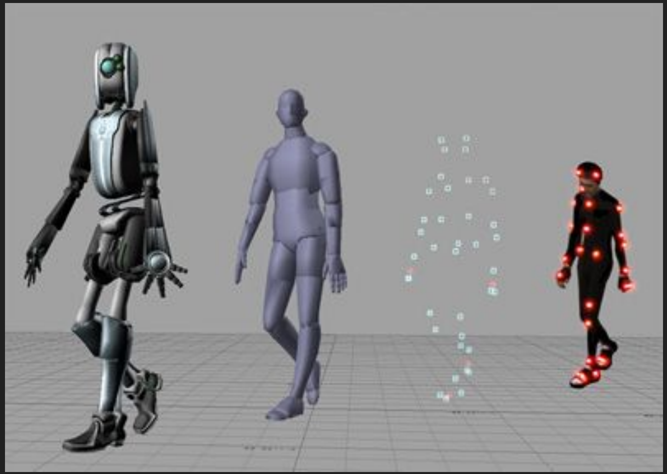
https://repl.it/@djgroen/Stars-and-planets
(and accompanying tutorial)

Integrating simulations in games

- Trade-offs:
  - Short development time.
  - Good performance.
  - Realistic results.
- External libraries vs. self-written code
  - With libraries no need to reinvent the wheel.
  - Libraries often save (a lot of) development time.
  - Self-written code forces you to understand the simulation model.
  - Self-written code is easier to modify & install.

Data-driven gaming

- Sometimes it’s possible to use data directly.
- Examples:
  - Motion capture.
  - Retrieving live data from web servers.
    - FIFA updates player transfer values based on their real-life performance.



On simulation performance

- Slow games are often highly irritating.
- Optimization:
  - Use simpler models / reduce the accuracy of the simulations.
  - Parallelization / multi-threading.
  - Reduce data before downloading it (to optimize network performance).
  - Perform computations on the server instead of the client (or vice versa).

Sources

http://www.slideshare.net/becker/the-calm-and-the-storm

http://gafferongames.com/game-physics/integration-basics/

And my short blog series on making small simulations:

https://coil.com/p/whydoitweet/Small-Sims-Index/rlFDzMuZx