CPSC 427
Video Game Programming

IO and the Observer Pattern

Helge Rhodin

Today

Recap: collisions and simulation

Communication between systems:

- ***The observer pattern***

If time permits, we will start with AI

Feature clarifications

- ***Particle effects (basic)***
  - Create particle locations and their motion on the CPU (smoke, fire, dirt…)
  - Render one Quad at every particle location
  - Create a shader (similar to light-up of the salmon that renders the particle in local object coordinates; can also be a texture)
  - ~~glDrawArraysInstanced~~ (old technique, no longer used)
- ***Advanced particle effects (counts as an additional feature)***
  - *Use the OpenGL point rendering function instead of quads*

Reminders:

- ***Be (better) prepared for face2face grading***
  - Have your laptop booted
  - Have the game compiled
  - Have the game running
  - Have the game at a point where you can demonstrate the feature

Submit a personal progress report

- Otherwise we will assume you did nothing/little
  - ***Do a late submission for M1 if still missing!***

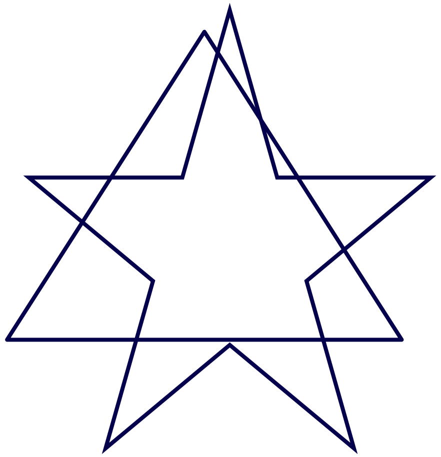
Decision trees – optional

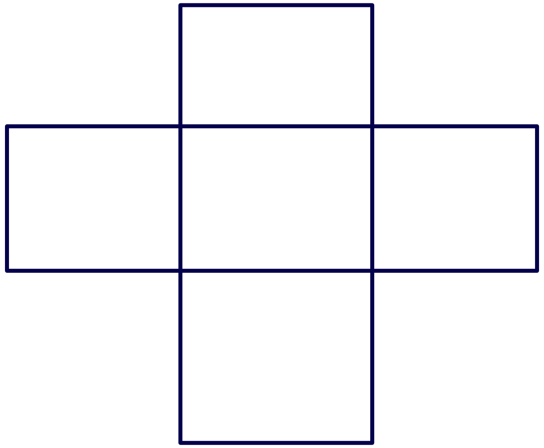
MTA – cross-play (ignore for now)

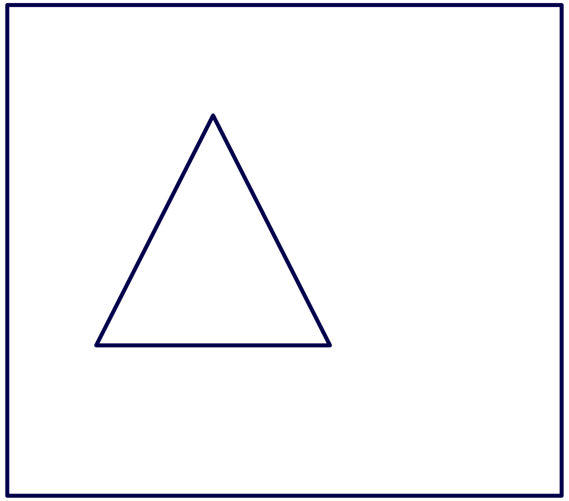
Collision Configurations?

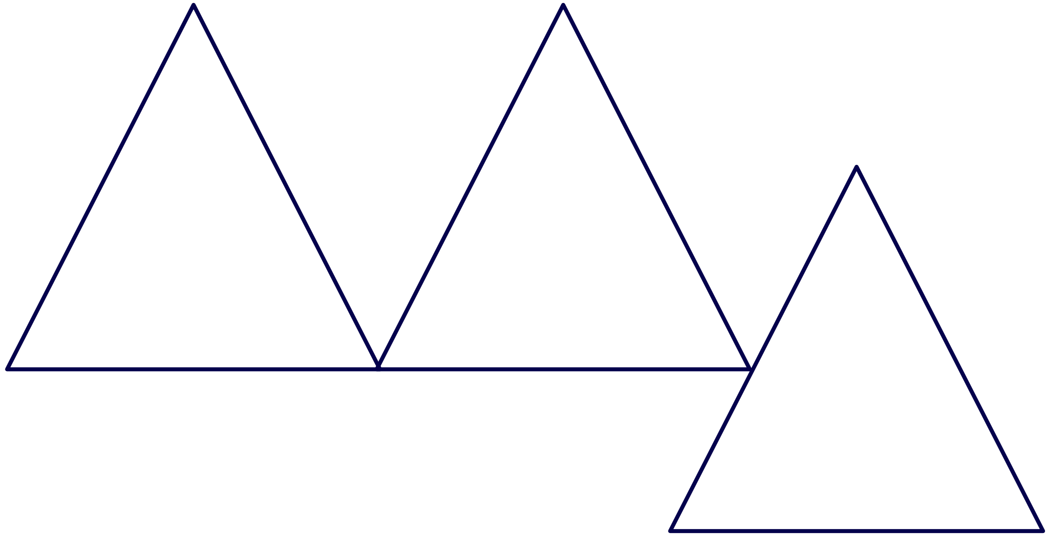
- Segment/Segment Intersection
  - *Point **on** Segment*

- Polygon inside polygon







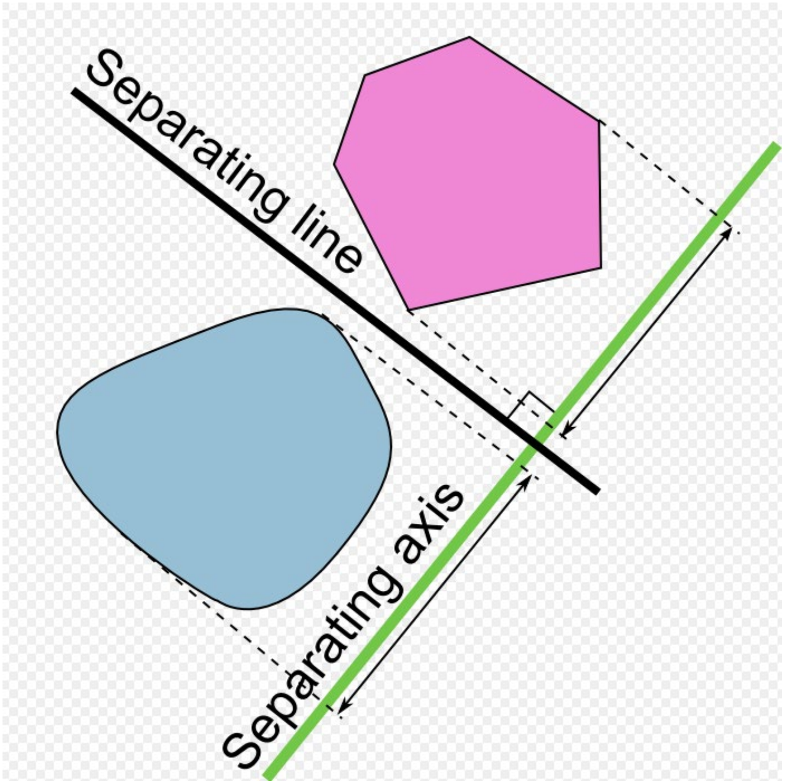


Separating Axis Theorem

Two convex shapes are not colliding if and only if there exists a line that separates the two

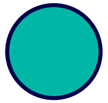
- In other words, if you can draw a line between two convex shapes without touching either, then the two shapes are not colliding.
- Otherwise, if no such line can be found, the shapes are definitely colliding
- In practice, only a few interesting lines need to be considered (such as edges)

More reading: https://en.wikipedia.org/wiki/Hyperplane_separation_theorem



Rigid Body Dynamics (rotational motion of objects?)

- **From particles to rigid bodies…**



Particle

$$state = \begin{cases} \vec{x}\ position \\ \vec{v}\ velocity \end{cases}$$

$\mathbb{R}^4$ in 2D

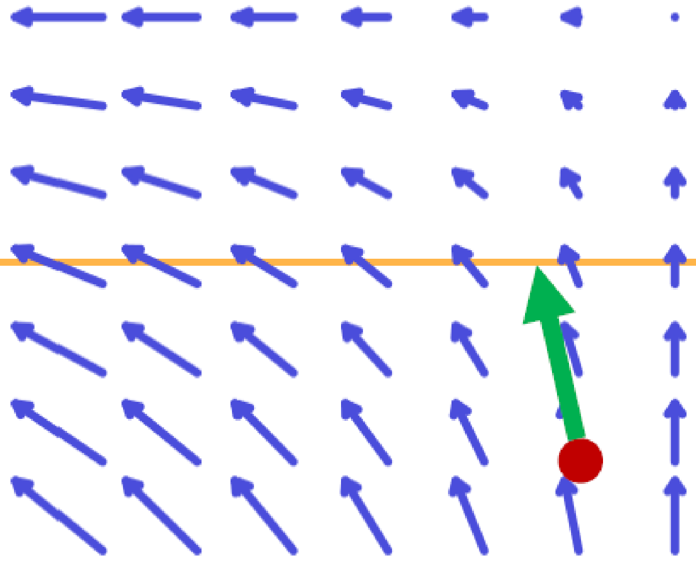
$\mathbb{R}^6$ in 3D

Rigid body

$$state = \begin{cases} \vec{x}\ position \\ \vec{v}\ velocity \\ R\ rotation\ matrix\ 3x3 \\ \vec{w}\ angular\ velocity \end{cases}$$

$\mathbb{R}^{12}$ in 3D

Recap: Force, impulse, vel…



Our goal: position and velocity

Think of:

- ***Force as an invisible string that pulls the object***
  - *changing in magnitude and direction over time and space*
  - *without a force, the object moves in a straight line*
- ***Impulse as a change in velocity***
  (dependent on the object mass)
  - *Force applied over one timestep*
    (can be continuous or instantaneous at some point during the step)

Simulation ingredients

- ***Plain forces (gravity, springs, …)***

$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_i)/m)d_t \qquad F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

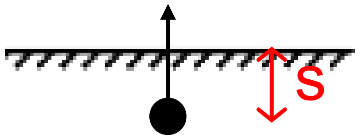
- ***Impulses (collision, player input, …)***

$$\vec{v}_{i+1} = \vec{v}_i + \vec{j}/m$$

← no d_t !

- ***Positional constraints (penetration)***

$$\vec{v}_{i+1} = \vec{v}_i + \beta * s \quad \text{or} \quad \vec{p}_{i+1} = \vec{p}_i + \beta * s$$

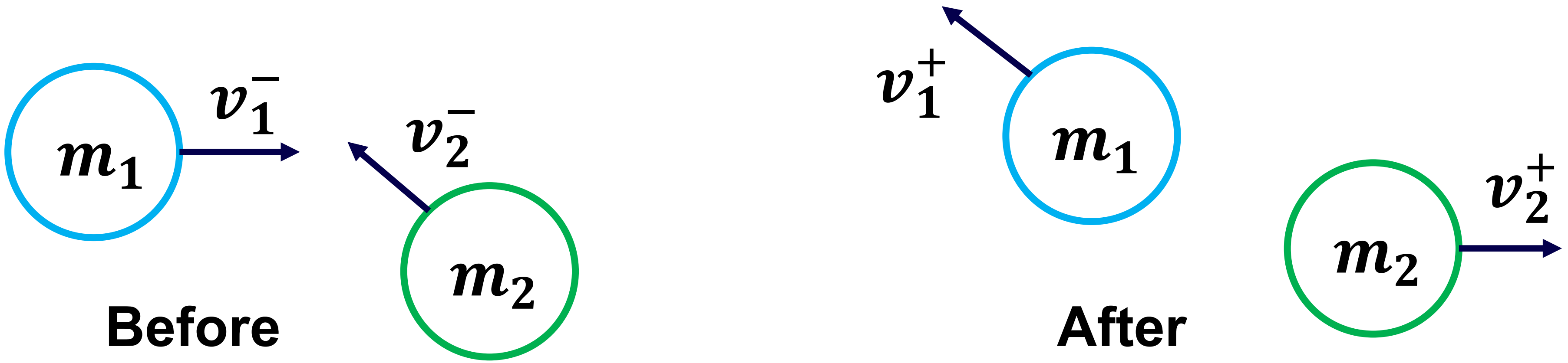


May lead to overshooting

Instead: fix position directly! (hacky but effective)

Particle-Particle Collisions (radius=0)

- Particle-particle **frictionless elastic impulse response**



- Momentum is **preserved**

$$m_1 v_1^- + m_2 v_2^- = m_1 v_1^+ + m_2 v_2^+$$

- Kinetic energy is **preserved**

$$½\, m_1 {v_1^-}^2 + ½\, m_2 {v_2^-}^2 = ½\, m_1 {v_1^+}^2 + ½\, m_2 {v_2^+}^2$$

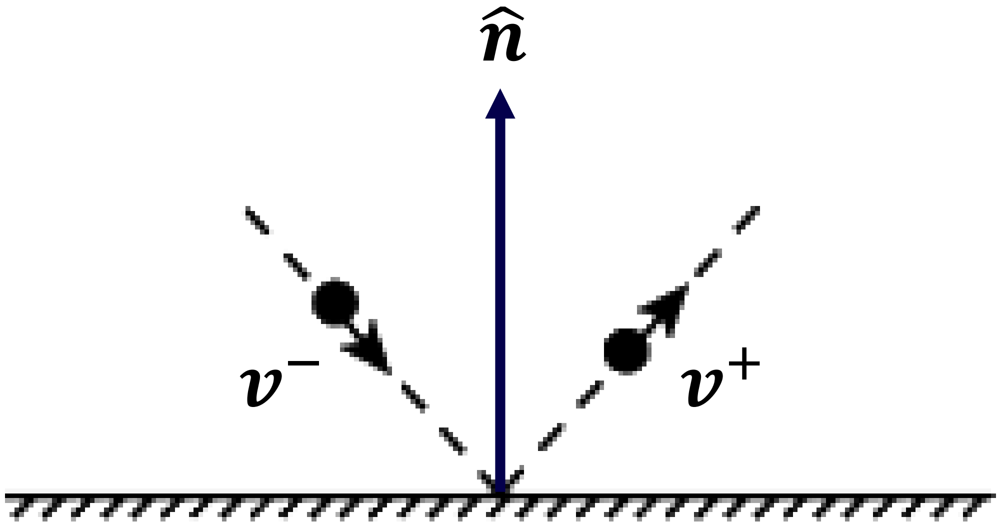
- Velocity is **preserved in tangential direction**

$$t \circ v_1^- = t \circ v_1^+ , \qquad t \circ v_2^- = t \circ v_2^+$$

Particle-Plane Collisions

- ***Apply an 'impulse' of magnitude j***
  - Inversely proportional to mass of particle
- ***In direction of normal***

Impulse in physics: Integral of F over time
In games: an instantaneous step change (not physically possible), i.e., the force applied over one time step of the simulation



$$j = (1 + \epsilon)(v^- \cdot \hat{n})m$$

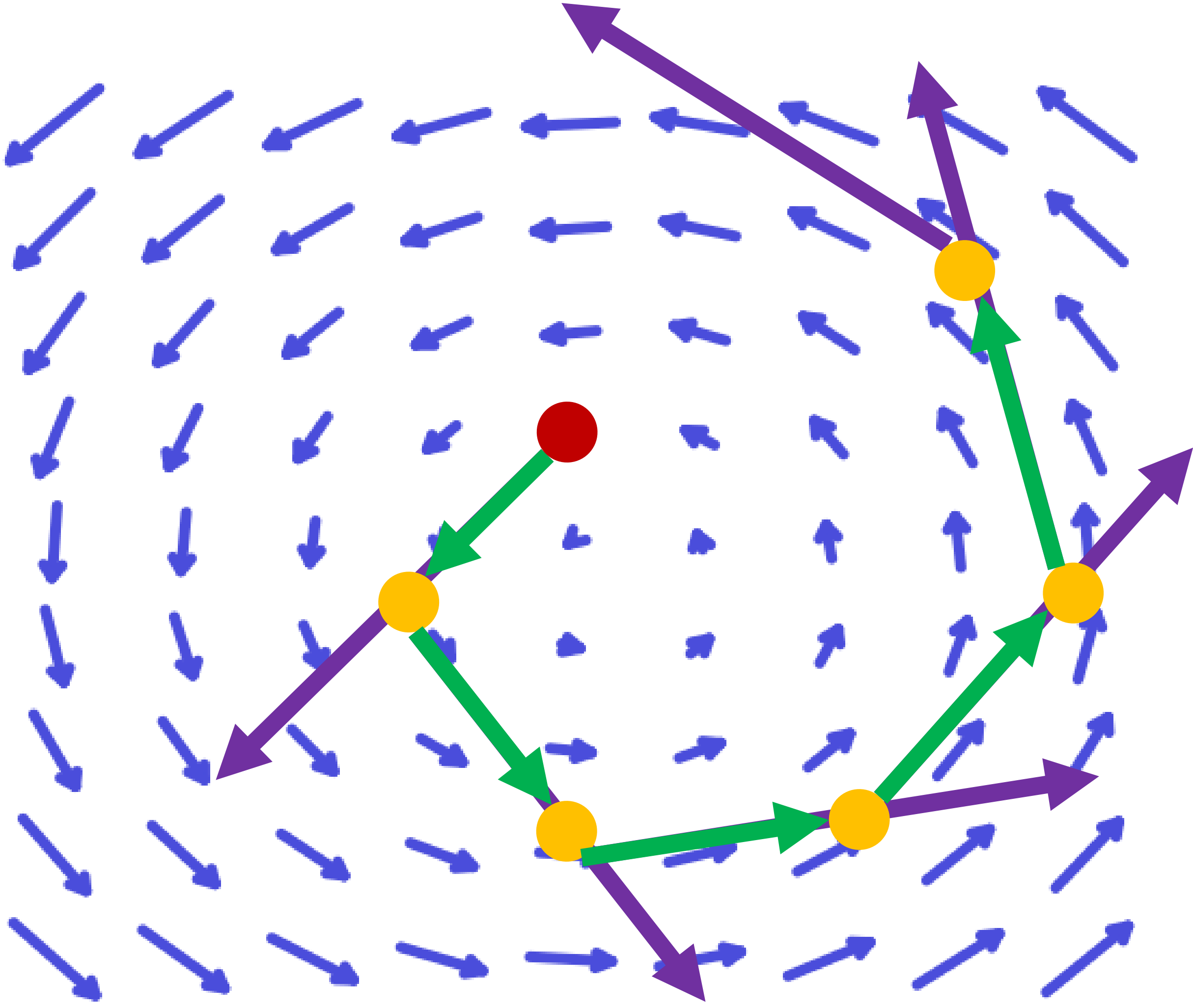
$$\vec{j} = j\,\hat{n}$$

$$v^+ = \frac{\vec{j}}{m} + v^-$$

What is the effect of ϵ ?

Explicit Euler Problems

- Solution spirals out
  - *Even with small time steps*
  - *Although smaller time steps are still better*

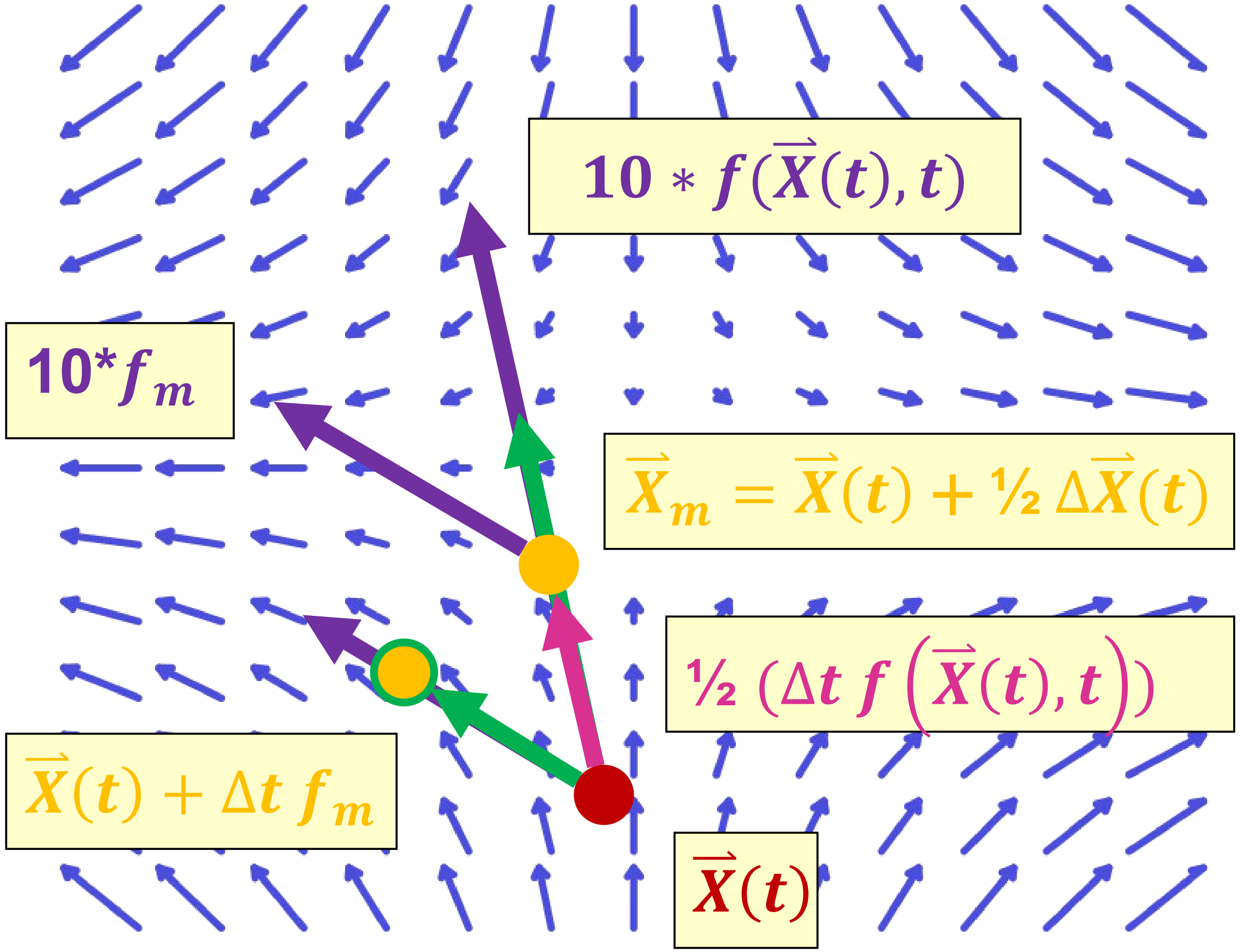


Definition: Explicit

- *Closed-form/analytic solution*
- no iterative solve required

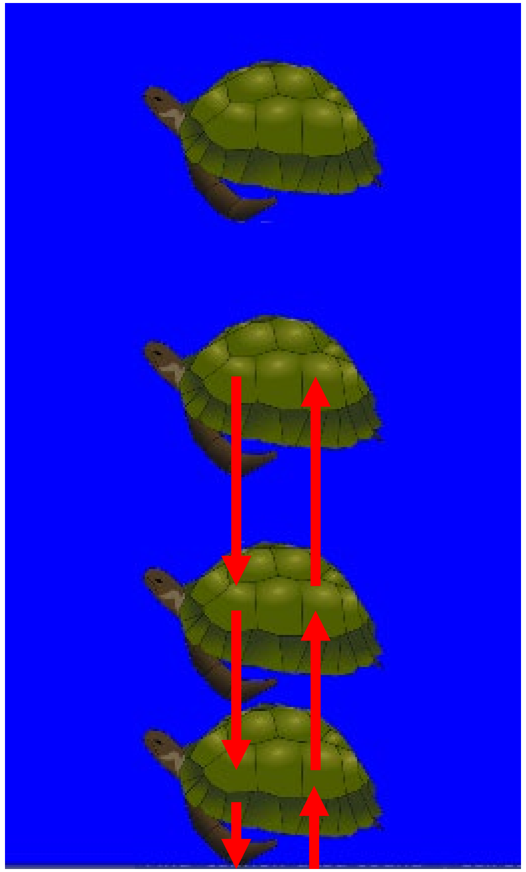
Midpoint Method

1. ½ Euler step
2. evaluate f_m at $\vec{X}_m$
3. full step using f_m



Issues:

- ***Complex relations***
  - *Multiple entities*



Self study: Sequential impulse updates

Idea:

- ***Apply each constraint (e.g, collision between two bodies) one-by-one***
- ***Resolve inaccuracies iteratively***
  - *An inner loop of ~10 iterations*
- ***Compute v+ at pt***

Excellent resource:

https://box2d.org/files/ErinCatto_UnderstandingConstraints_GDC2014.pdf

https://box2d.org/files/ErinCatto_ModelingAndSolvingConstraints_GDC2009.pdf

Self study: Sequential impulse updates

Step 1: Forces acting on individual objects

- Gravity, air resistance, wind…
- Compute forces, then update velocity

Step 2: Pairwise forces (or group-wise)

- Detect collisions, compute penetration and restitution (bouncing) forces, update velocity of the involved entities right after the force computation (no accumulation!)
- Iterate by computing impulses and updating velocities (repeat K=~10 times, until corrective impulses are small)

Step 3: Update positions

- Use velocities from the previous step

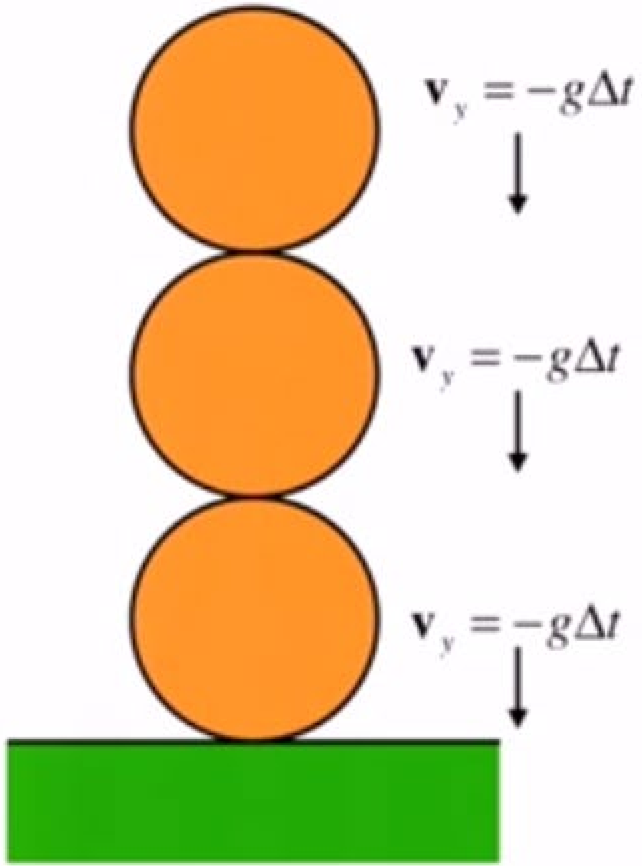
Step 4: Apply positional constraints (to mitigate drift)

Self study: Sequential impulse updates

Pitfalls:

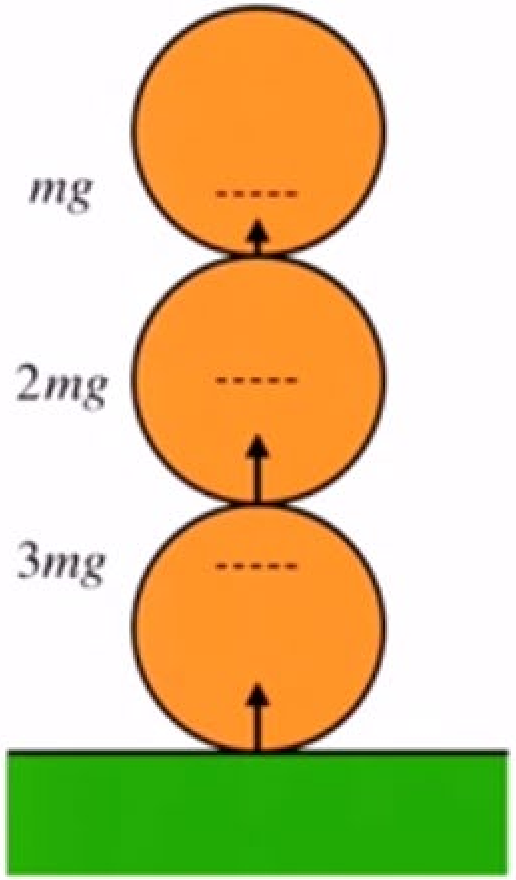
- ***Important to update velocity right after computing constraint/forces***
- ***Important to update the velocity of both objects at the same time for a collision event***
- ***Restitution (bouncing) is complex***
  - *The outgoing velocities depend on the relative masses of objects*
    - What if multiple objects are stacked?
    - The ones below influence the one above
    - Inaccurate with sequential updates, requires block optimization (optimization of multiple constraints at once; system of equations)

Sequential Impulses local solver

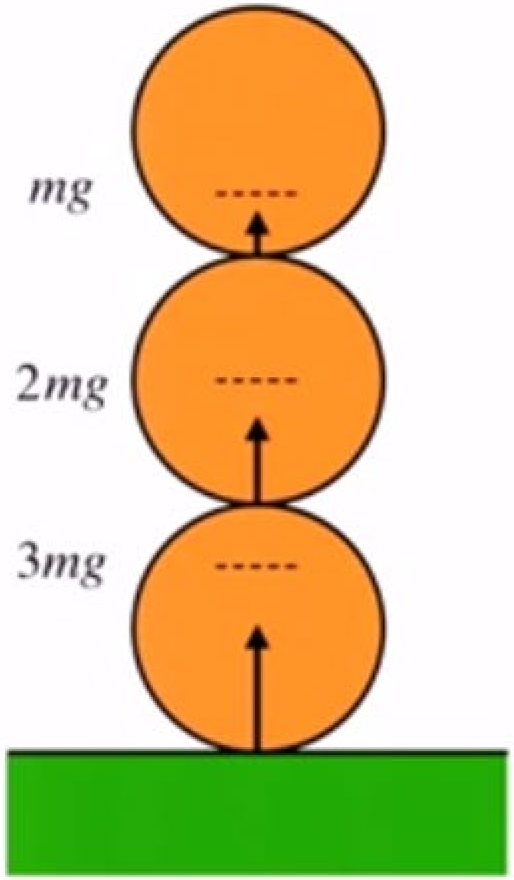


Apply gravity

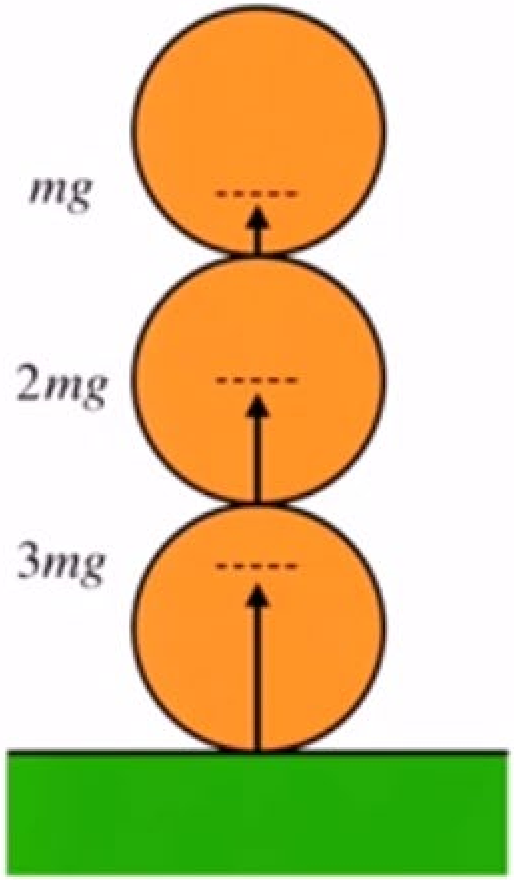
Iteration 1



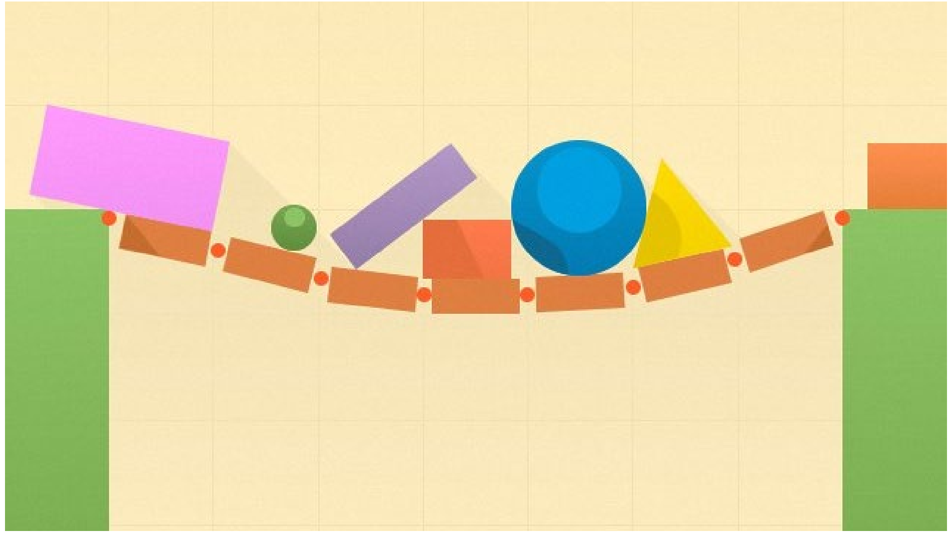
Iteration 2



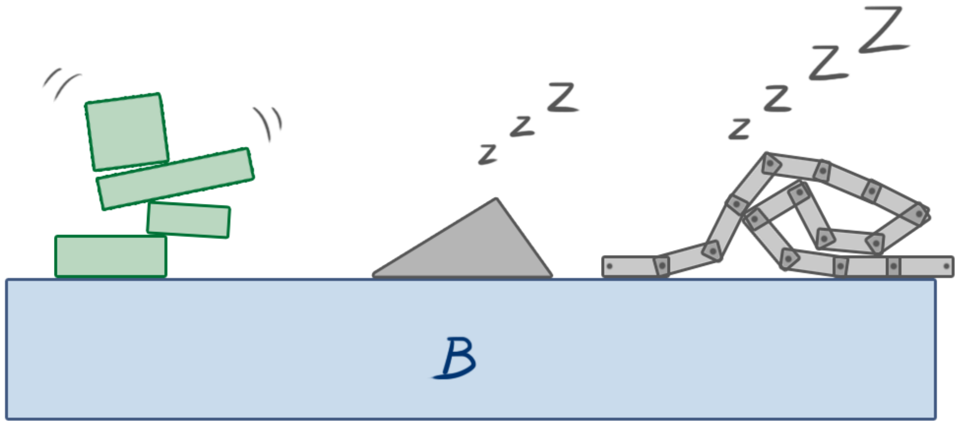
Iteration 3



Self-study: Constrained physics



By Nilson Souto
https://www.toptal.com/game/video-game-physics-part-iii-constrained-rigid-body-simulation



Questions

Which solver to use? For a space simulator (with accurate orbits, e.g., satellites)

1: Forward Euler

2: Backwards Euler

3: Midpoint

4: Trapezoid

5: Seq. Impulses

Questions

Which solver to use? For a jump & run

1: Forward Euler

2: Backwards Euler

3: Midpoint

4: Trapezoid

5: Seq. Impulses

Questions

Which solver to use? For a billiard game (with many balls that can stack)

1: Forward Euler

2: Backwards Euler

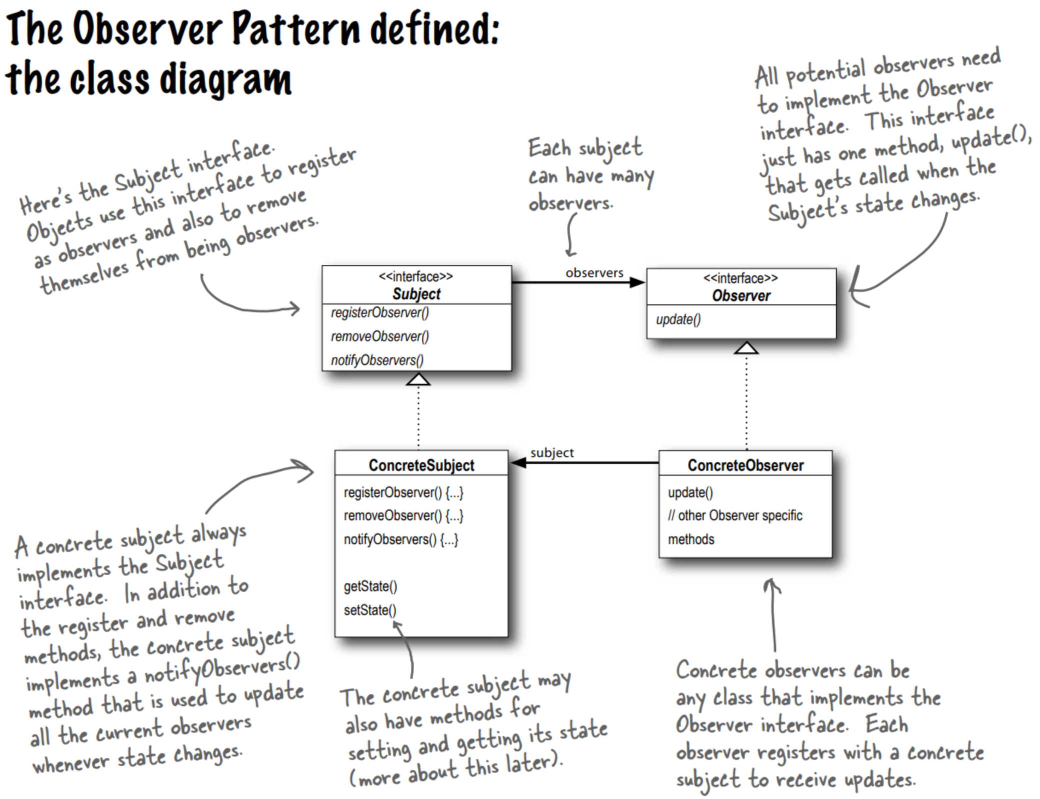
3: Midpoint

4: Trapezoid

5: Seq. Impulses

CPSC 427
Video Game Programming

IO and the Observer Pattern

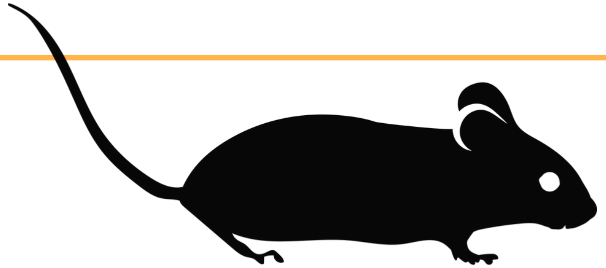


Helge Rhodin

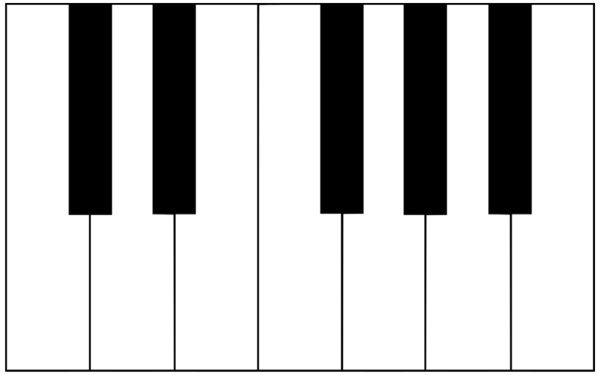
Mainloop

```
int main(int argc, char* argv[]) {

    ...

    2. Mainloop:

    while (!world.is_over()) {
```

Event Processing

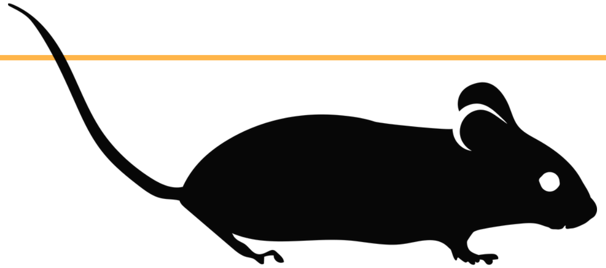


Mouse event,
Keyboard event,
etc.

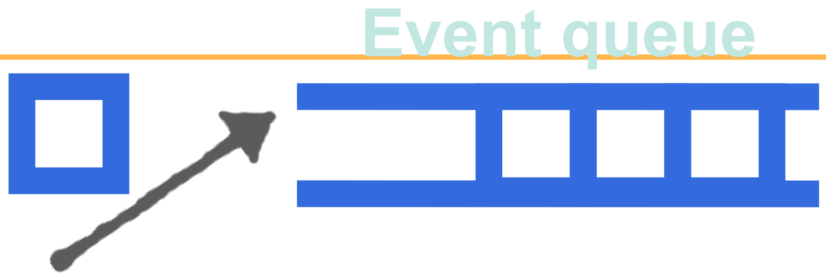


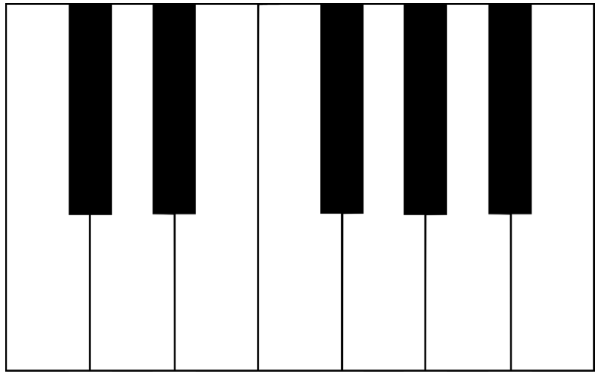
Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

Event Processing: Event Queuing



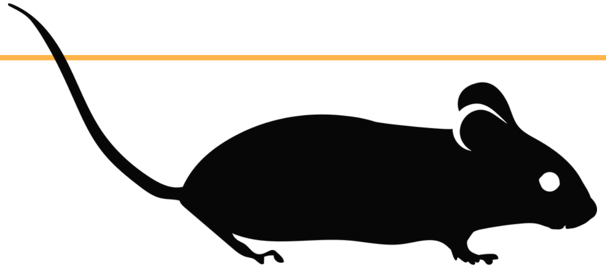
Mouse event,
Keyboard event,
etc.



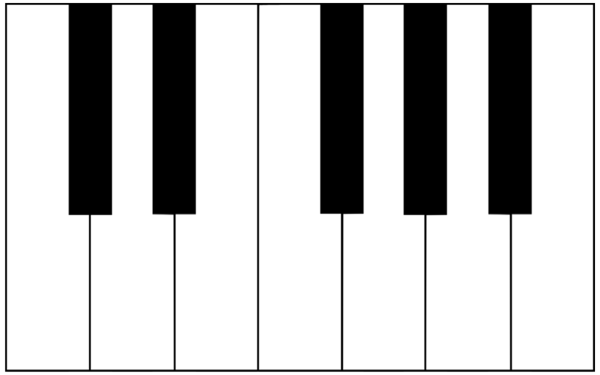


Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

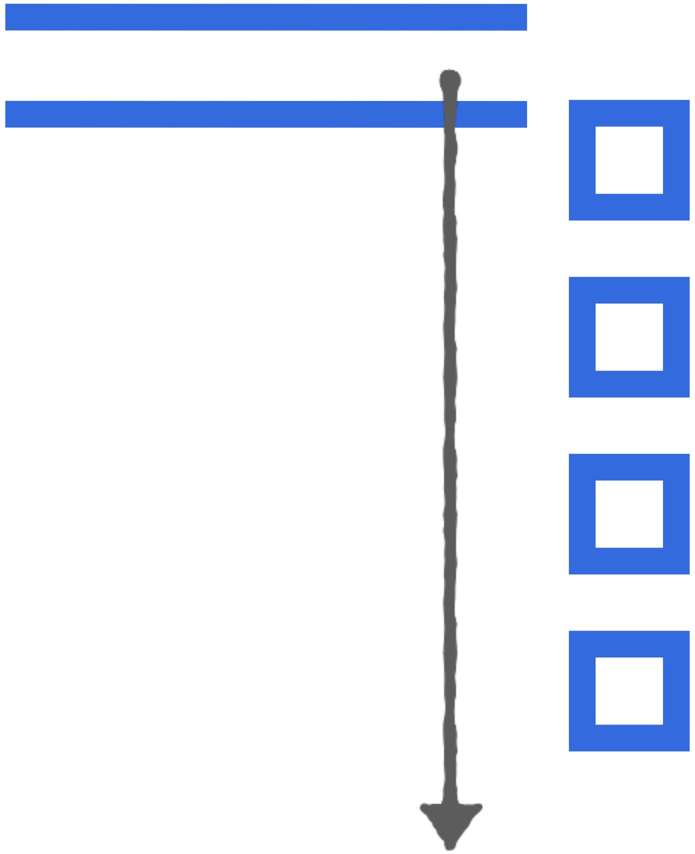
Event Processing: Event Polling



Mouse event,
Keyboard event,
etc.



Event queue



```
while (!world.is_over()) {

        glfwPollEvents();

}
```

Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

Event Processing: Event Callback

GLFW calls corresponding callbacks:

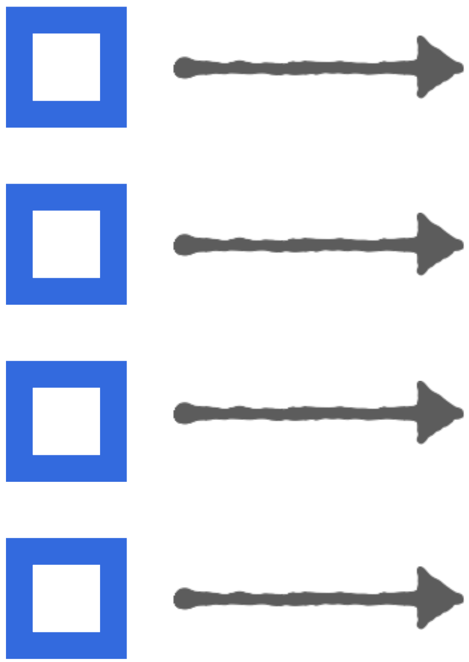
- void World::on_key(GLFWwindow*, int key, int, int action, int mod)

  —-> You need to set salmon motion here.

- void World::on_mouse_move(GLFWwindow* window, double xpos, double ypos)

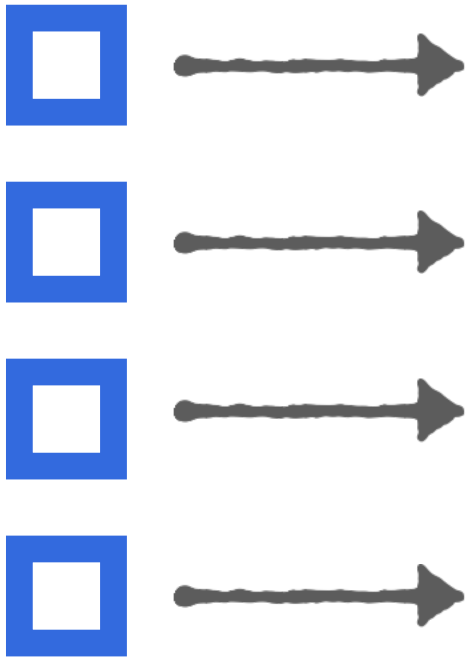
  —-> You need to fill this function to set salmon rotation.

Event Processing: Event Callback



How does GLFW know which callback to call?

Event Processing: Event Callback



How does GLFW know which callback to call?

—-> Registered in initialization:

`world.init(...)`

```
glfwSetKeyCallback
glfwSetCursorPosCallback
```

Mainloop

```
int main(int argc, char* argv[]) {
…
2. Mainloop:
while (!world.is_over()) {
   2.1 Event processing
   2.2 Game state update
   2.3 Rendering a frame
}
…
}
```

The Observer Pattern

- ***Gang of Four (GoF)***
  - *Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides*
  - ***Design Patterns: Elements of Reusable Object-Oriented Software*** (1994)

- ***A pattern described by the GoF***
- event-driven
  - *clients register for an event*

Good ref (object oriented): https://gameprogrammingpatterns.com/observer.html

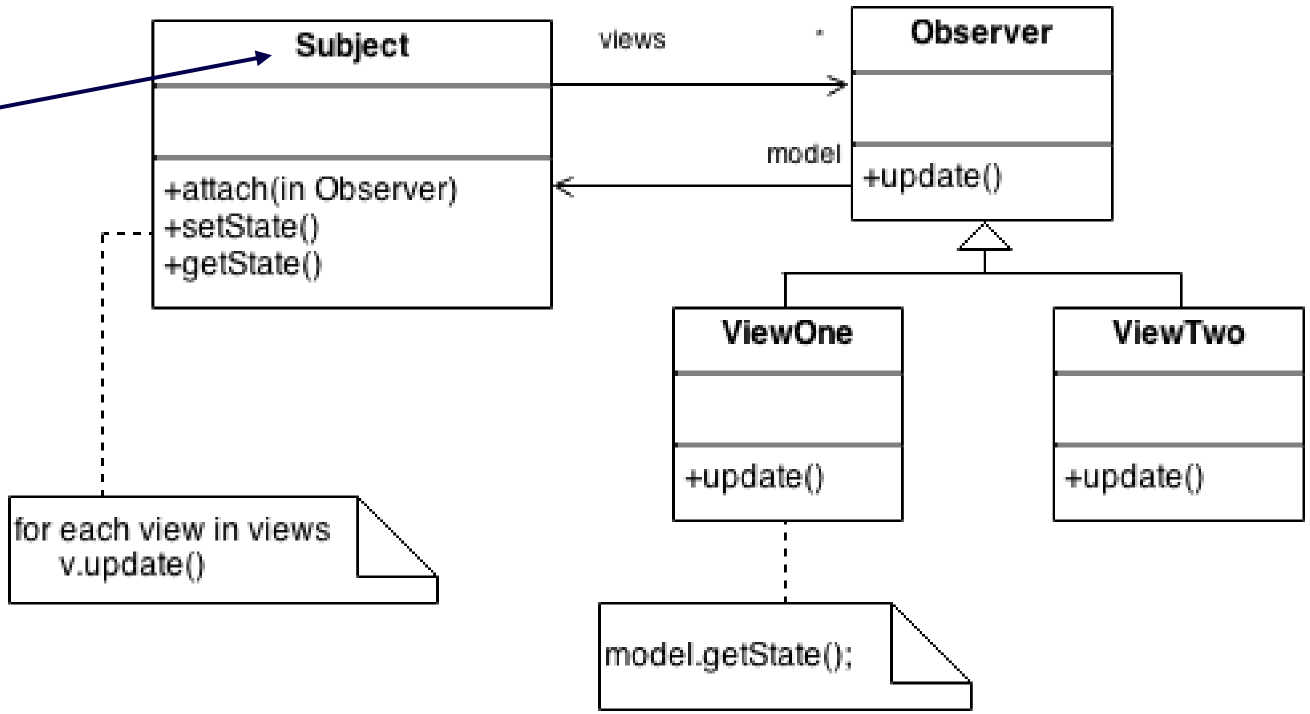
Use Cases

- ***Rewards***
- ***Communication between systems (in ECS)***
- ***User input***
- ***Have you encountered this problem yet ?***

Observer Pattern – OOP

- *Define a common interface*
- *All observers inherit from that interface*

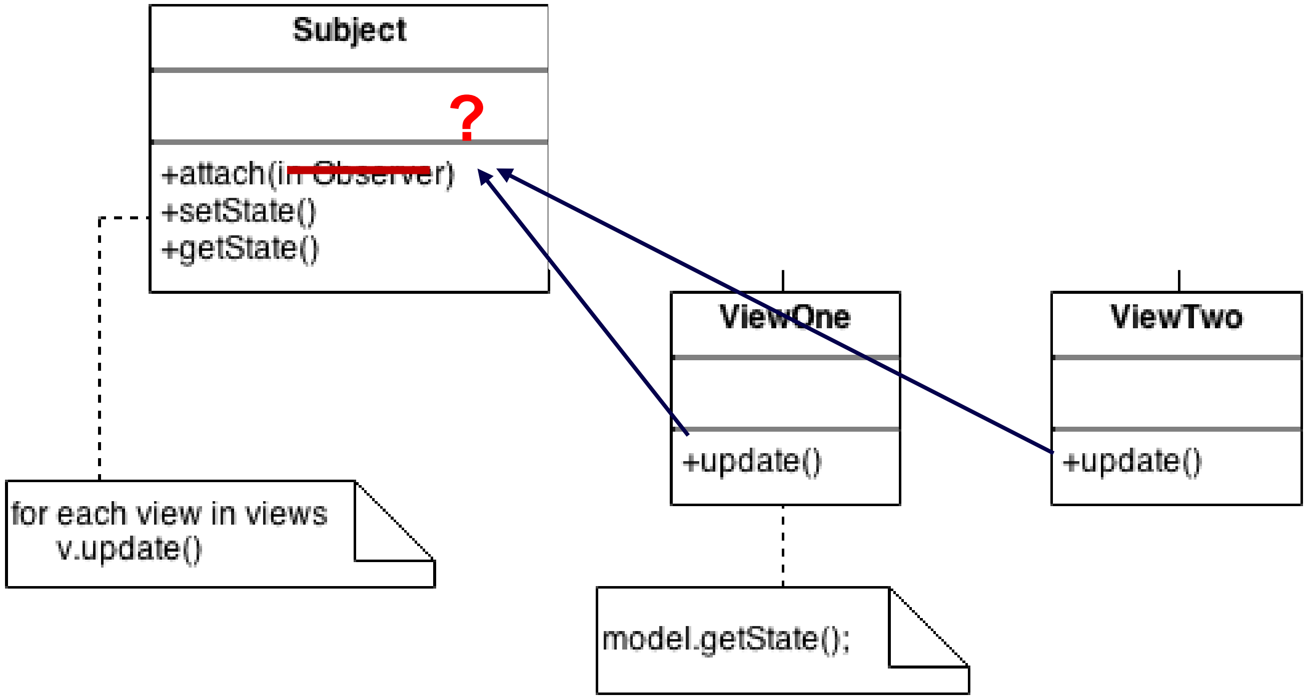
Called Subject by GoF



Do we want inheritance?

Observer Pattern – With Functions

- ***function with matching signature instead of class***



A function that accepts a function

- ***Using std::function***

```
#include<functional>

void LambdaTest (const std::function <void (int)>& f)
{
    ...
}
```

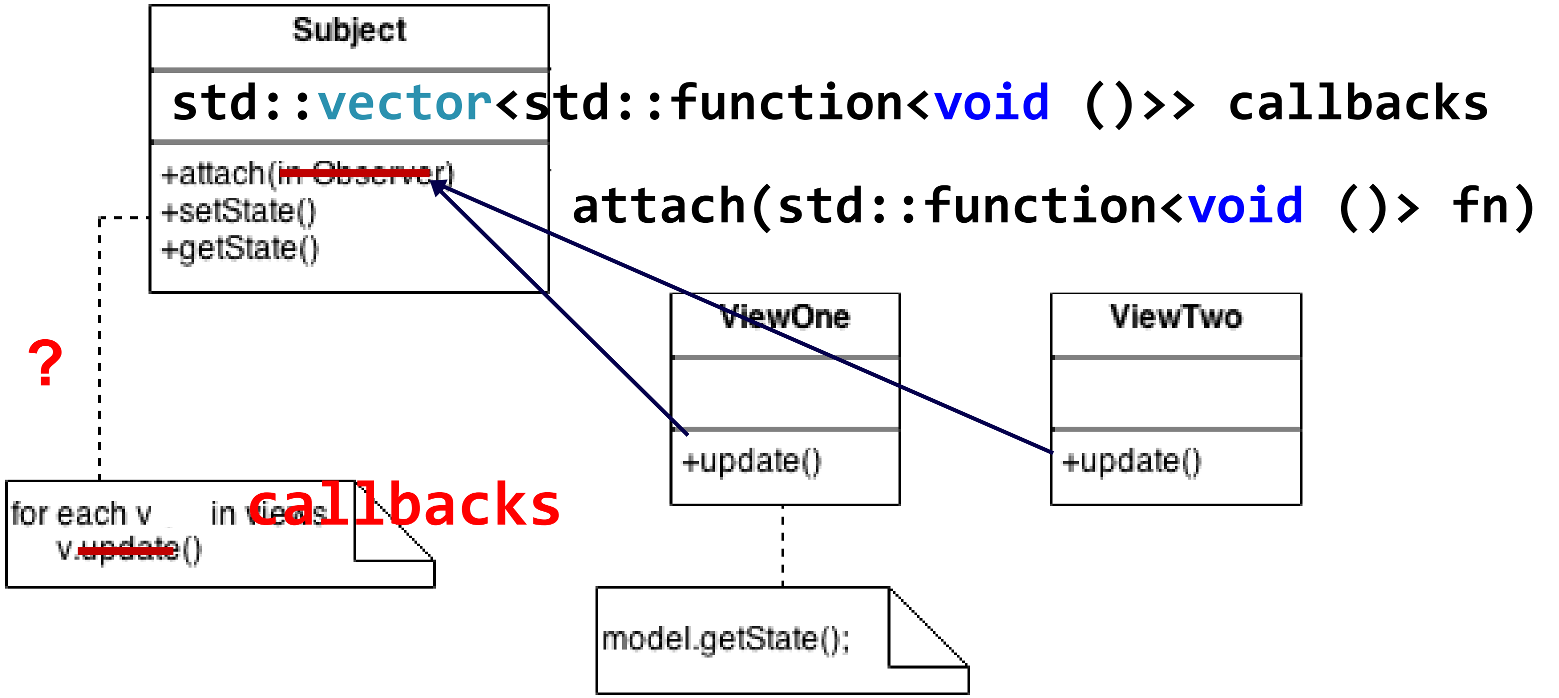
- ***Using templates***

```
template<typename Func>
void LambdaTest(Func f) {
    f(10);
}
```

  - *use templates to accept any argument with an operator()*

Observer Pattern – With Functions

- *function with matching signature instead of class*



Issues with passing member functions?

- *You may have to **std::bind** the **this** pointer*

- *Or use lambda functions as a wrapper (C++ 11)*

- *Make sure that the object is not moved*
  - *E.g., components within the ECS system can be moved around*
    - Don't create a callback to components!

Lambda Functions

Definition:

- auto y = [] (int first, int second) { return first + second; };

Call: int z = y(1+3);

- Infers return type for simple functions (single return statement)
  - otherwise

```
auto y = [] (int first, int second) -> int { return first + second; };
```

- Can capture variables from the surrounding scope.

```
int scale;
auto y = [] (int first, int second) -> int { return scale*first + second; };

auto y = [&] (int first, int second) -> int { return scale*first + second; };
```

Performance?

- ***Isn't this slow?***

- ***Is it dangerous?***