UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Spring 2020

PROGRAMMING

LEVEL 1 – Overview of (2D) game engines

Maurizio Rigamonti

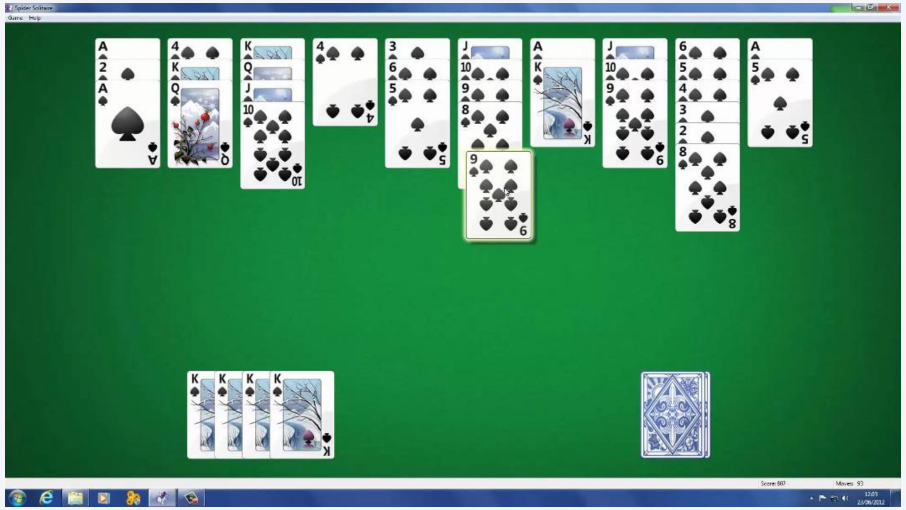
1. THE BASIC LOOP

THE BANNER SAGA

THE WITCHER 2

VIDEOGAMES

- “Static” world
  - It doesn’t update itself
  - Reactive to user interaction
  - Similar to “working” GUI



- “Living” world
  - It changes independently
  - User is an entity of the world
  - Similar to monitoring applications

TIME CONSTRAINTS IN GAMES

- Real-time software
  - Time critical nature
  - Time-constrained conditions

- Graphics
  - > 25 frames per second (fps)
- Music
  - continuous
- Interaction
  - <50 ms
- Loading time, world updating, and so on.

FUNCTIONALITIES OF AN ENGINE

- Load/Unload/Init resources
  - Engine
  - Game resources
  - Level resources

- Manage the **Game Loop**
  - Rendering
  - Network
  - Physics
  - World update

REAL-TIME LOOPS IN VG

- State of the world
  - Objects, non playing characters, and so on
- Interaction
  - Inputs
  - User: special entity in the world
- Aesthetics
  - Outputs: sound, music, graphics

SIMULTANEITY

- Simplification: 2 main parallel routines

1. **World update**
   - Constant speed on each machine
   - Gameplay afflicted by speed variations
   - 10-15 updates per second are often enough

2. **Rendering**
   - As often as possible
   - New PCs: smoother animation, better frame rate

- How to run them “simultaneously”?

A VALID SOLUTION

- Single threaded program
- Simulate threads with regular software loops and timers
- Key idea:
  - Execute update and render sequentially
  - Skip update calls -> synchronized with time
  - Render as often as possible

THE SIMPLE LOOP ALGORITHM

```
currentTime = Date.now();

while (!end){

    //check if time to render
    if (currentTime > lastRenderTime + renderTimeStamp){
          Render();
          lastRenderTime = currentTime;
    }

    //check if time to update
    if (currentTime > lastUpdateTime + updateTimeStamp){
          Update();
          lastUpdateTime = currentTime;
    }

    //store current time
    currentTime = Date.now();
}
```

THE GAME OBJECT

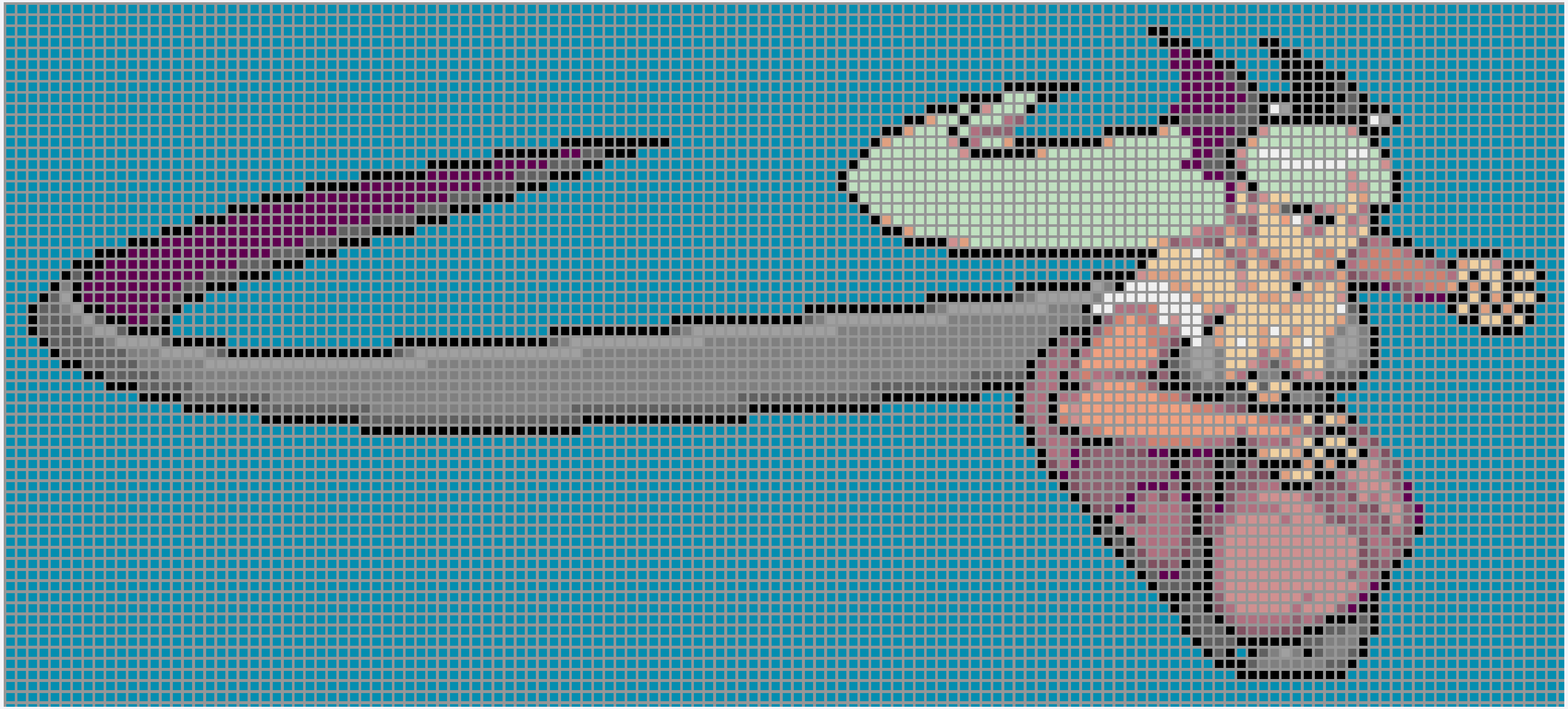
- Any possible resource in the game (avatar, NPC, scoreboards)

- Properties
  - Transform
  - Rendering
  - Physics (rigid body, colliders)
  - Specific properties

- Life cycle (similar to the game loop)
  - Init
  - Start
  - Update
  - Render
  - Destroy

2. RENDERING

SPRITE BASED CHARACTERS



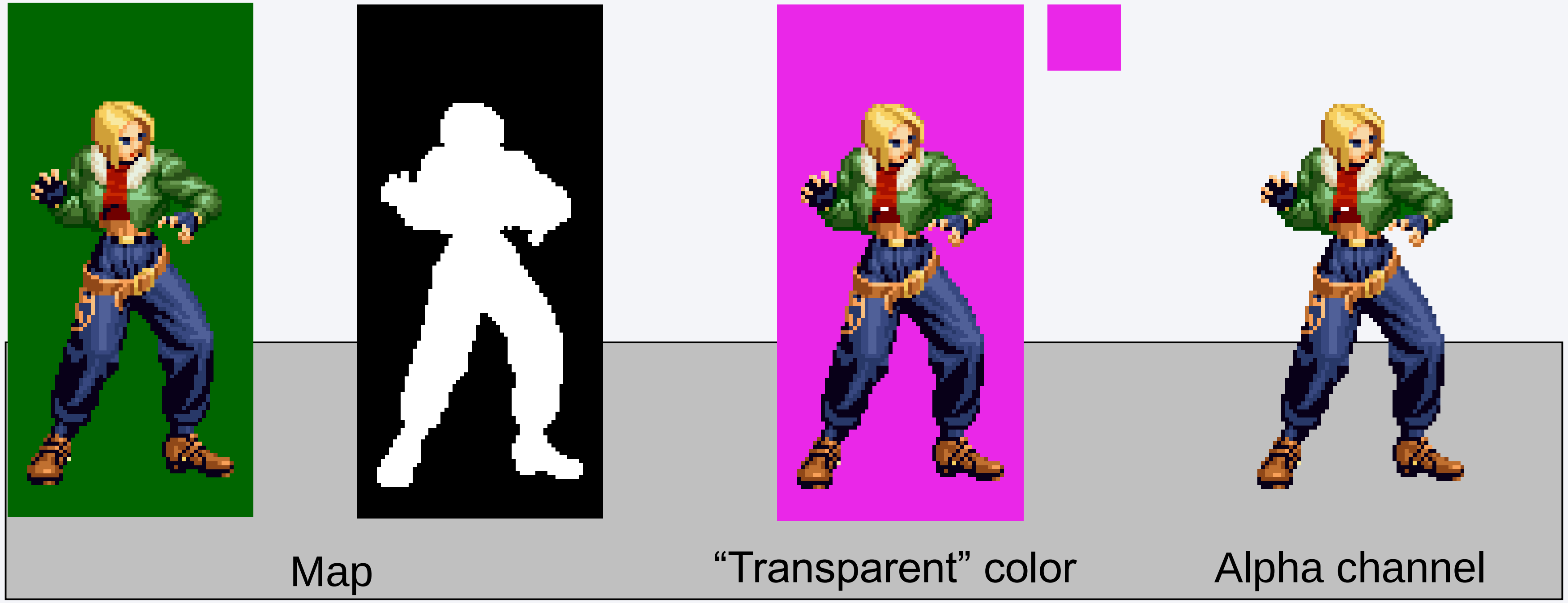
2D GAMES CHARACTERS

- Characters stored in a rectangular bitmap
    - Drawing surfaces is a very fast operation!
- The animation is obtained by alternating several images (frames)
- This structure is named a **sprite**
    - Reference to the first games, involving ghosts, sprites, knights, and so on.
- **Blitting**: operation of layering sprites onscreen
    - Blit is a contraction of “block image transfer”

SPRITE REPRESENTATION

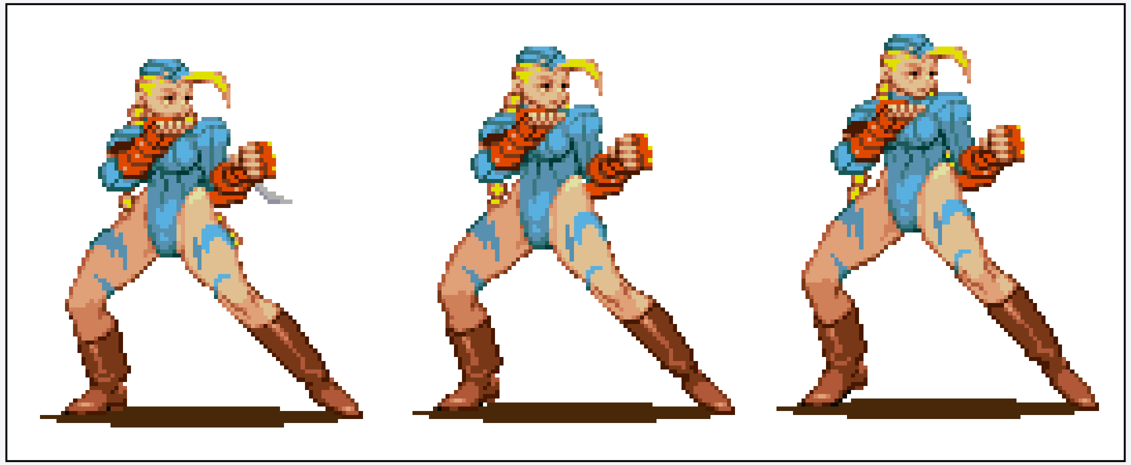
A sprite contains visible and non visible pixels

Maps, "transparent color" or alpha channel

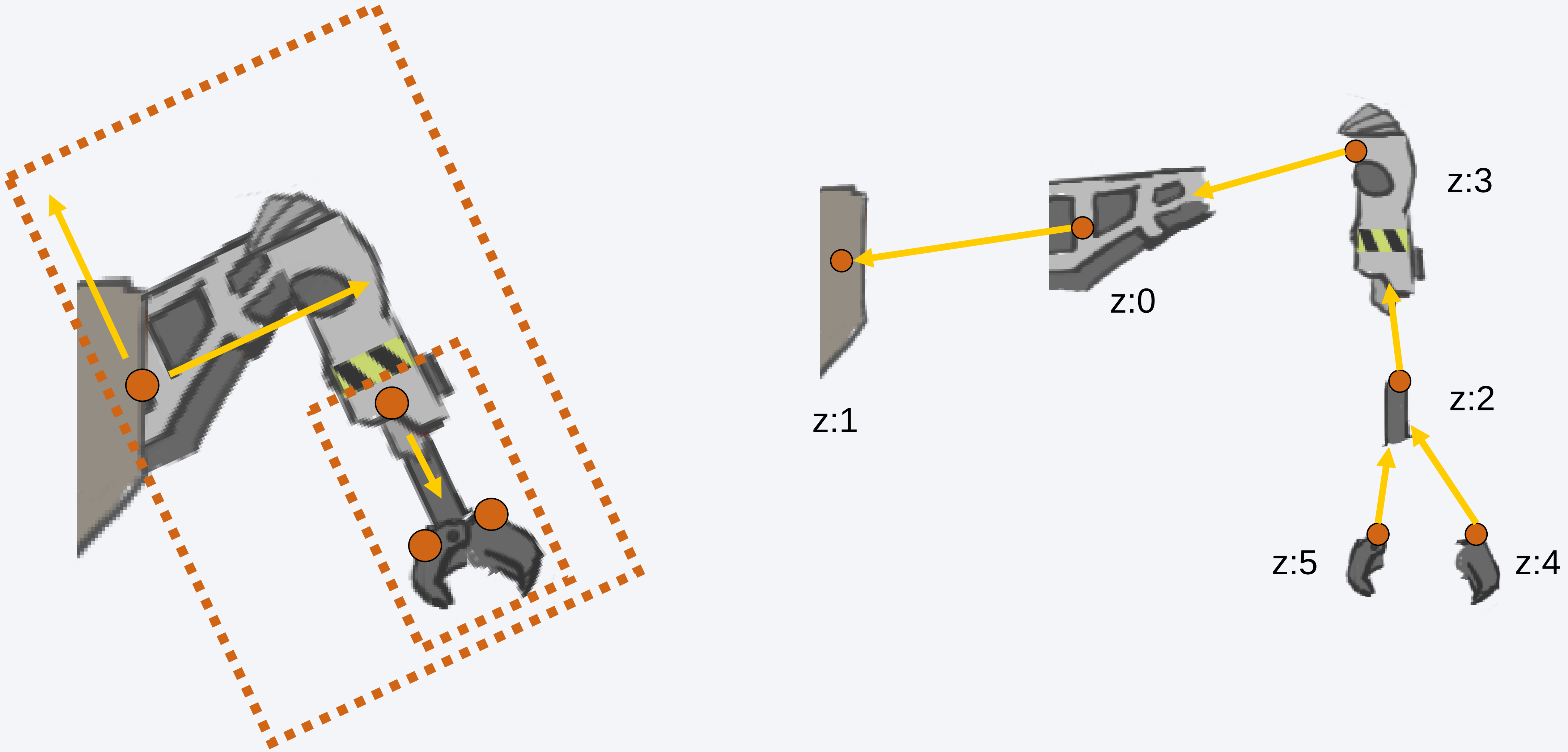


FRAMES REPRESENTATION FOR ANIMATION

Several images or a spritesheet



SKELETON



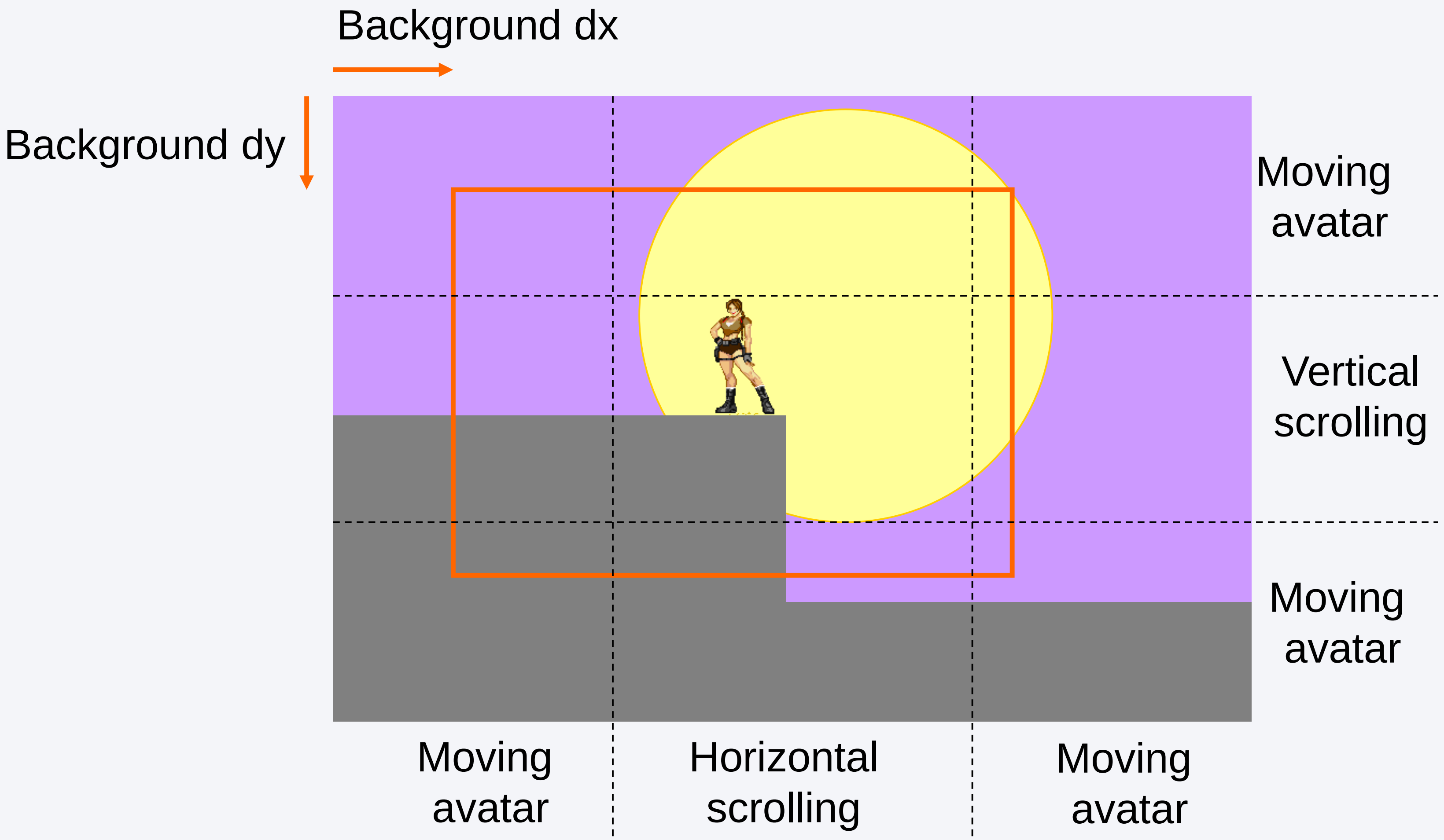
BASIC METHODS FOR BACKGROUND GRAPHICS

LEVEL GRAPHICS

- In general, levels use maps bigger than the screen
- The background can be
  - A big image
  - A composition of small images (tiles)
- Often, big images are also split in sub-images for optimization purposes

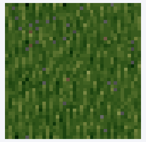
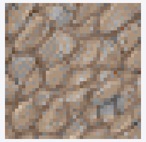
SCREENS

TWO- AND FOUR-WAY SCROLLING



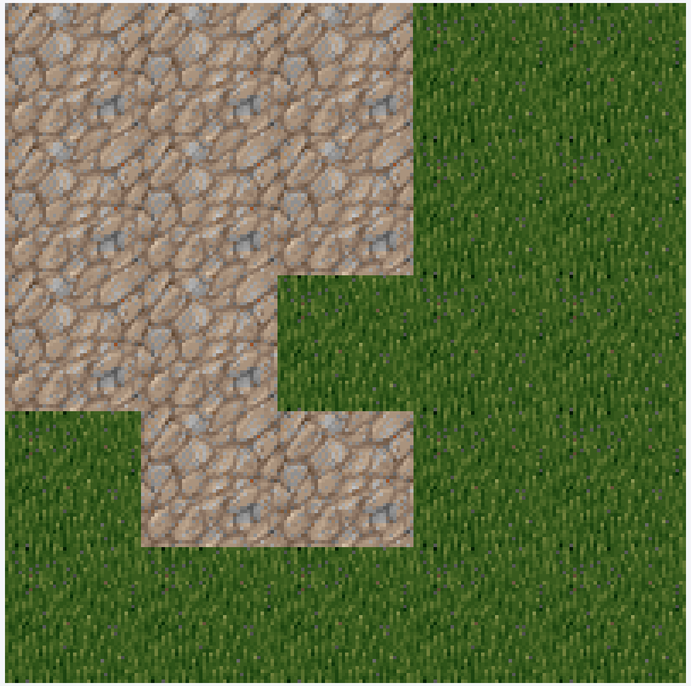
TILING

Tile table

0   1

Mapping matrix

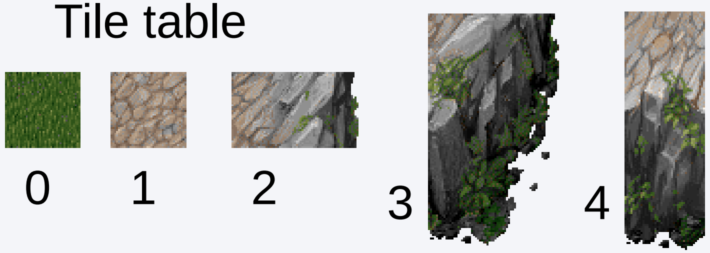
1	1	1	0	0
1	1	1	0	0
1	1	0	0	0
0	1	1	0	0
0	0	0	0	0



Tiles: *2D Circle Graphic Archive*, Daniel Cook
http://www.lostgarden.com/

MULTILAYERED MAPS

Tile table

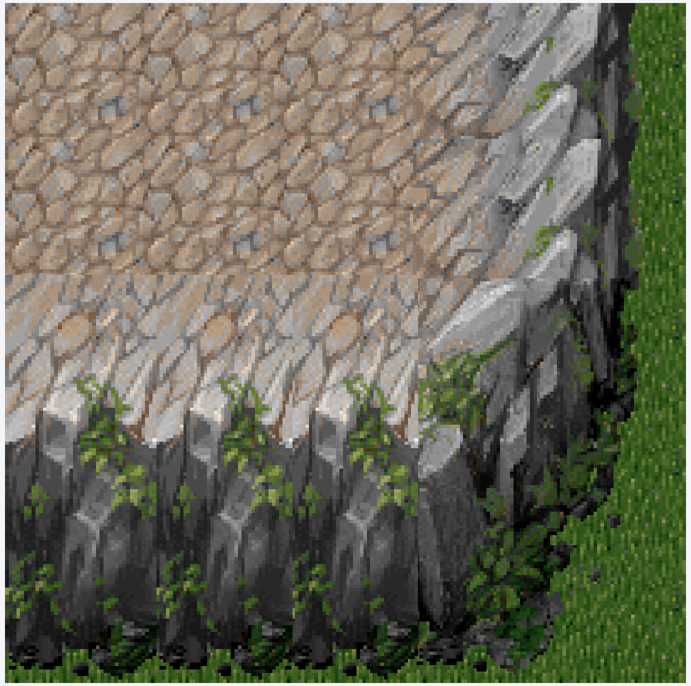


Matrix 0

1	1	1	0	0
1	1	1	0	0
1	1	0	0	0
0	1	1	0	0
0	0	0	0	0

Matrix 1

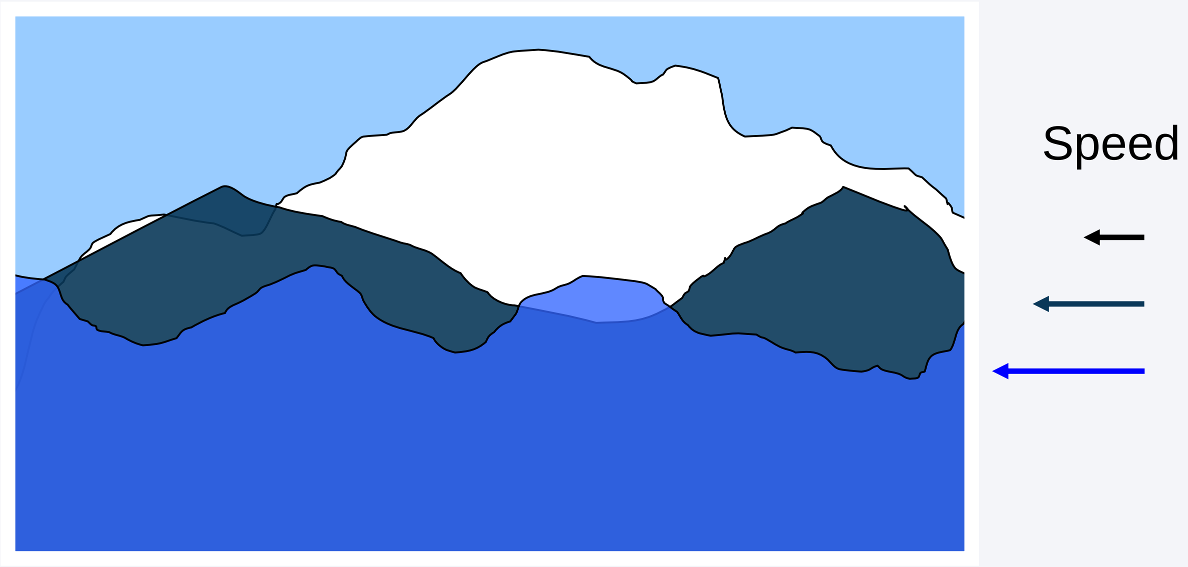
			2	
			2	
4	4	4	3	



Tiles: *2D Circle Graphic Archive*, Daniel Cook
http://www.lostgarden.com/

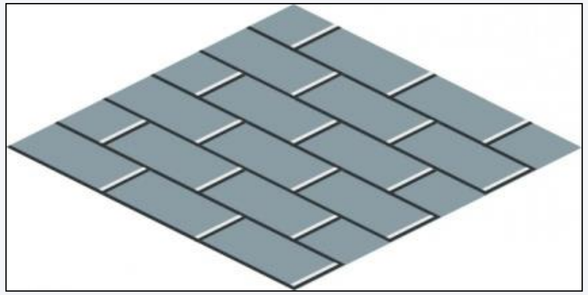
PARALLAX SCROLLING

- Optical phenomenon
  - Apparent displacement of a distant object, viewed from 2 different positions
- Foreground objects seem to move faster than background
- Semi 3D effect, adding sense of depth

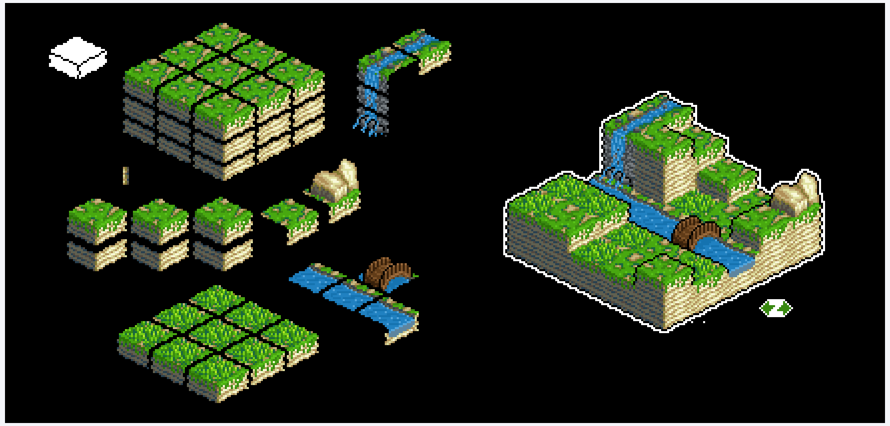


ISOMETRIC TILES

- Simulation of 3D scenario
  - Invented by architects and industrial designers
- Representation of objects and levels as if they were rotated of 45 degrees
  - Parallel perspective
  - Lines do not converge in a conic perspective
  - No distortions



http://forums.create.msdn.com/forums/t/45209.aspx



http://www.miniwizardstudios.com/iso-tiles.asp

PAGE-SWAP OR BIG MAPS

- Scrolling background, without being limited to a set of tiles
- Draw the level complete image and divide it into sectors
  - Dynamic loading during the game
  - Active sectors in main memory, the rest on secondary devices (DVD, hard drive, etc.)
  - Smaller sectors = faster loading

3. UPDATING THE WORLD

THE UPDATE

1. Refresh inputs

2. Update the game objects
   - Usually specific behaviors are integrated at this stage

3. Update the physical engine
   - e.g. rigid body, particles

4. Collision detection
   - Usually 2 stages: broad + narrow

5. Collision resolution

REFRESH INPUTS

- 1 and only 1 peripheral
  - Related to gameplay and game experience!
  - Difficult to integrate multiple peripherals. Sometimes the gameplay also changes

- An abstract model
  - The abstract controller is mapped to each device
  - The player is that way enabled to choose the preferred device
  - Zero impact on the game engine

INPUT: SYNCHRONOUS MODEL

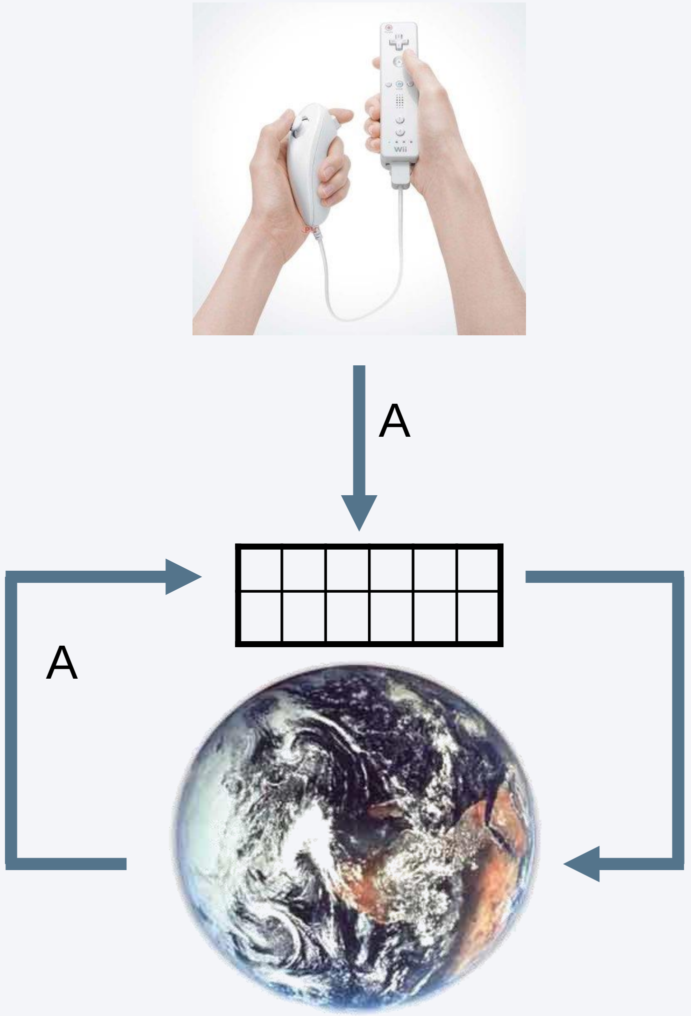
1. The world is waiting
2. User interactions create events
3. The world reacts to the event
    - E.g. Java event model

INPUT: ASYNCHRONOUS MODEL

1. The world is running

2. At each loop, it consults the input devices or a **table**

3. The world reacts to the states of the devices



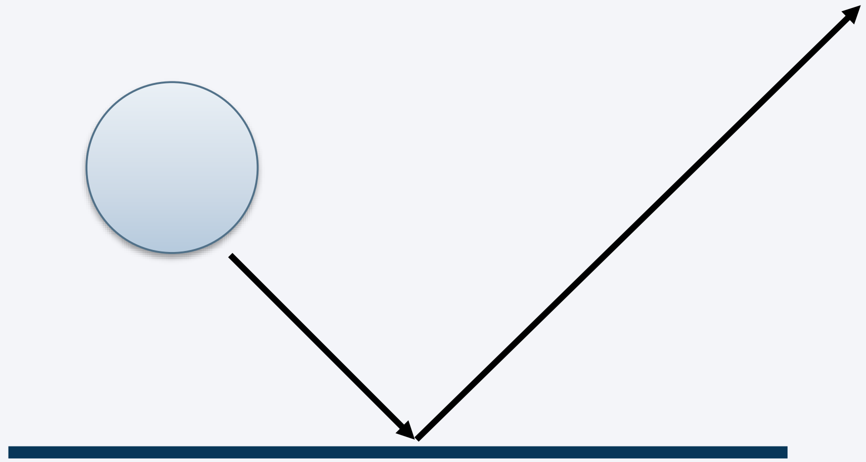
ASYNCHRONOUS TABLE

- The table contains “boolean” values mapped to the different entries

- Better than consulting each possible entry
  - Consistence of data: use values recorded at the same time

- Physics engine
  - Simulation of bodies
  - Possible to define parameters
  - Based on Newton’s mechanics

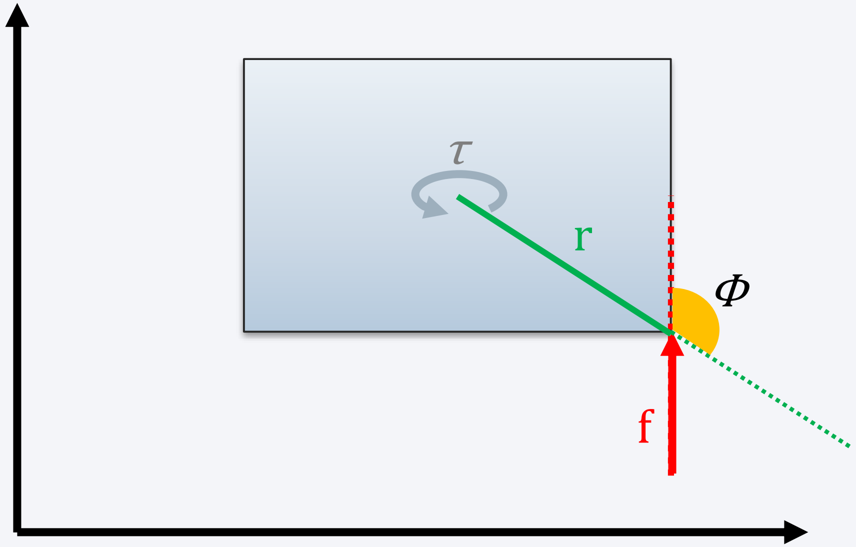
- Newton’s three laws of motion:
  - Inertia
  - Force, Mass, and Acceleration
  - Action and Reaction



- Solid that don’t deform
  - Doesn’t exist in the real world!
- Simplify dynamics of solids
- Very useful for games
- Properties
  - Position, mass, velocity
  - Shape (it can be rotated!)
  - Sounds
  - Etc.

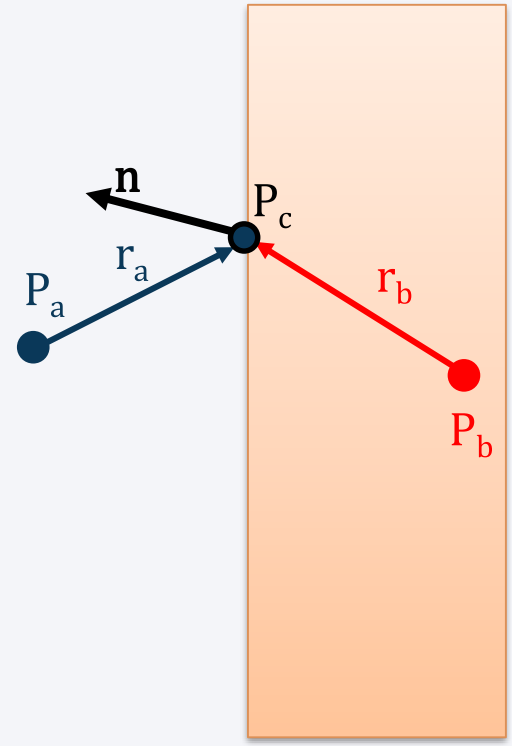
- A mass
- A position vector
- A velocity vector

- Angular properties
  - In 2D, rigid bodies can only rotate on z axis
- Angular velocity
  - Scalar for radians per second, represented as ω (omega)
- A rotational force



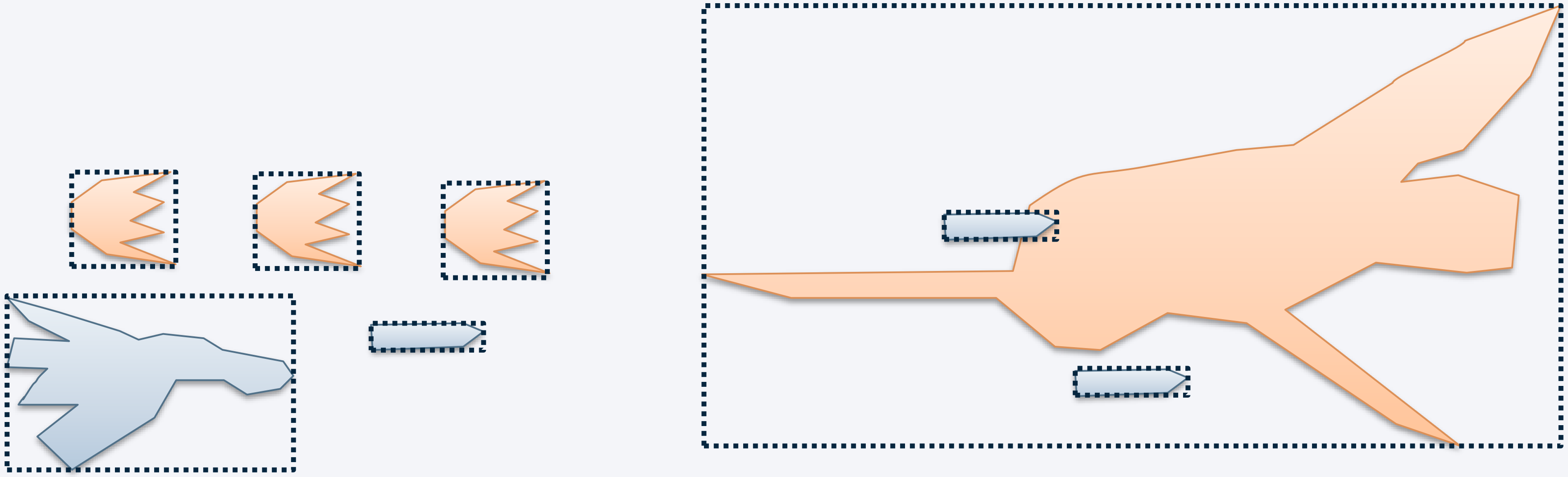
COLLISIONS

- Collision
  - Two shapes intersect
  - Distance smaller than a tolerance
- Detection is computationally expensive: O (n^2)
- Solution to improve calculation: 2 phases
  1. Broad phase
  2. Narrow phase



BROAD PHASE OVERVIEW

- Find pairs of shapes **potentially** colliding and exclude non colliding pairs
- Usual strategy: *space partitioning* coupled with *bounding boxes* (or volumes)



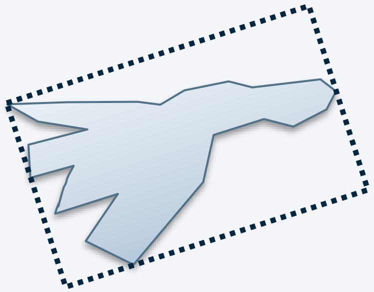
Not colliding

Potentially colliding

BOUNDING VOLUMES

- Shapes can be complex
- Use simpler shapes to accelerate process: **bounding volumes**

Axis-aligned
(or not-oriented)
bounding box

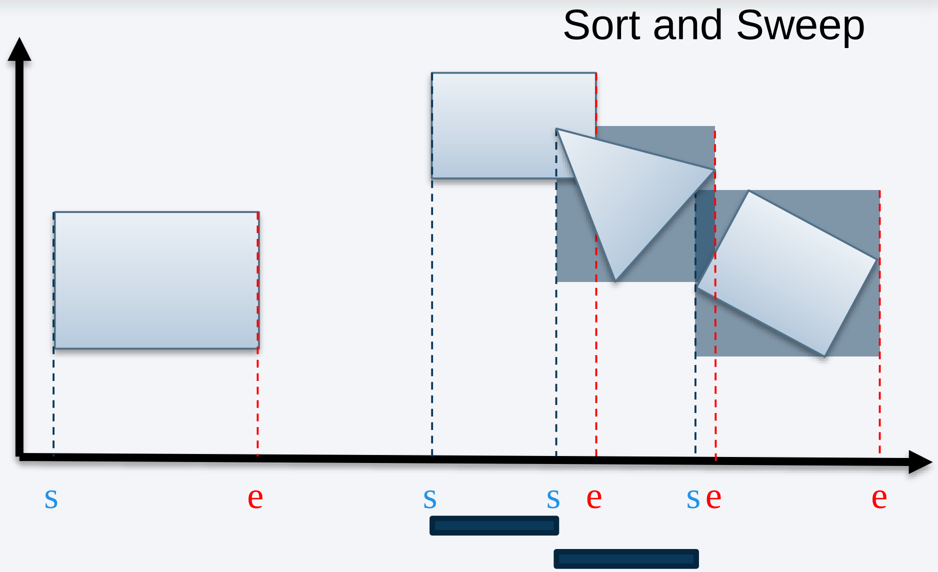


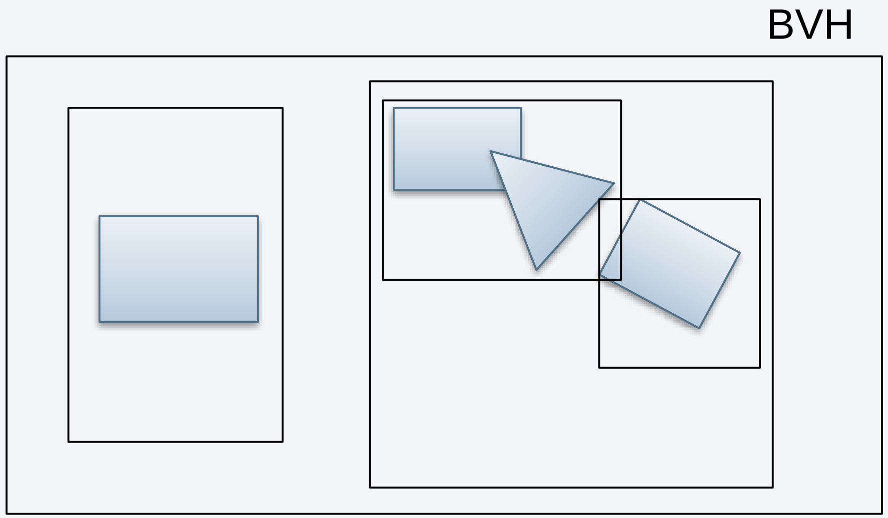
Oriented
bounding box

Bounding
circle

SPACE PARTITIONING

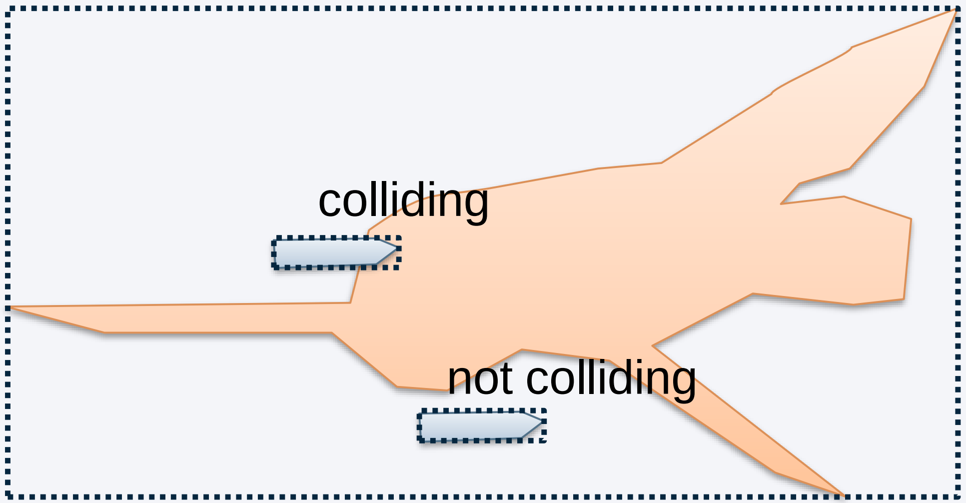
- AABB is cheap…
- … but expensive for each pair: O (n^2)
- Space partitioning to reduce amount of AABB tests!
  - Uniform grids
  - Quadtrees (2D) and Octrees (3D)
  - Spatial Hashing
  - **Sort and Sweep**
  - **Bounding volumes hierarchy**





NARROW PHASE OVERVIEW

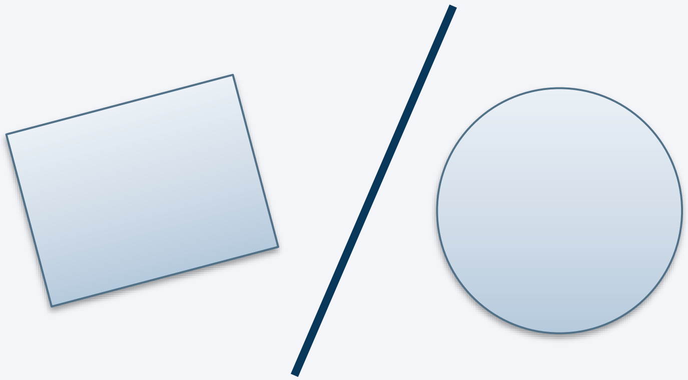
- Analyze potential colliding pairs found in the broad phase
- Detect collision really happening

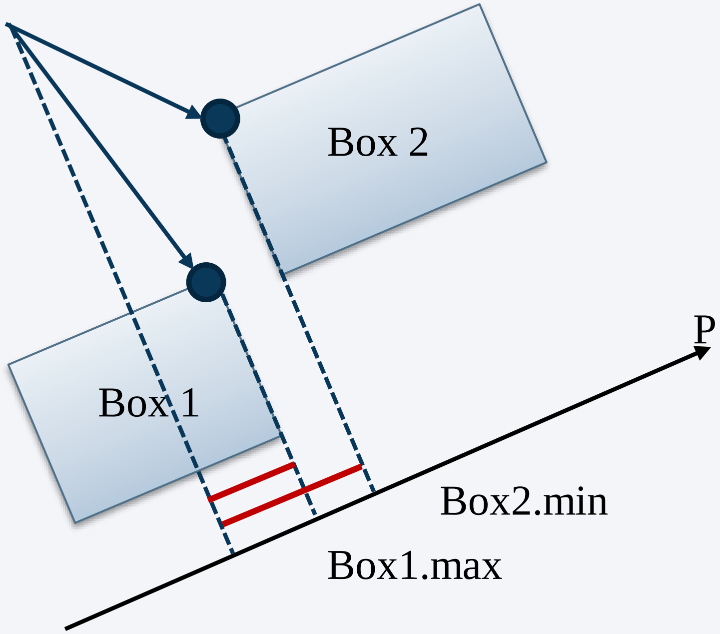


STRATEGIES

- Process convex shapes only
  - E.g. use convex hull for concave shapes
- Test intersections
  - Use different techniques for different colliders
  - E.g. Separating Axis Theorem (SAT) for convex complex shapes
  - Distance between polygons (cf. Erin Catto)
- Continuous VS discrete collision detection

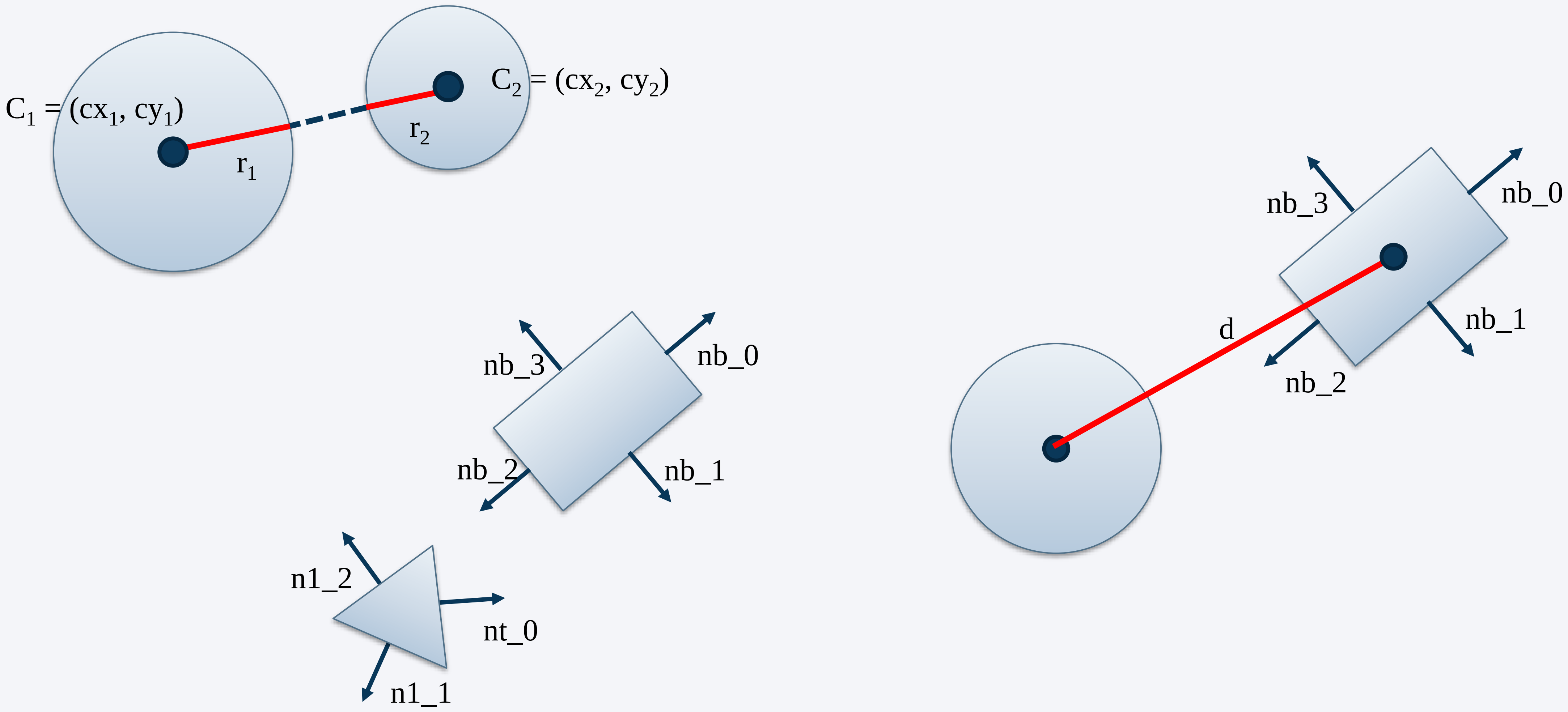
THE SEPARATING AXIS THEOREM





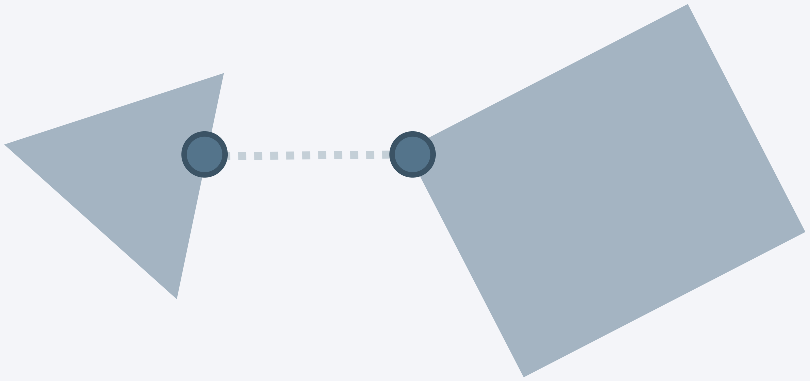
- Collision between polygons
- Idea: if a line can be drawn between two polygons, they don’t collide
- Visually easy to do, but… how to calculate it?

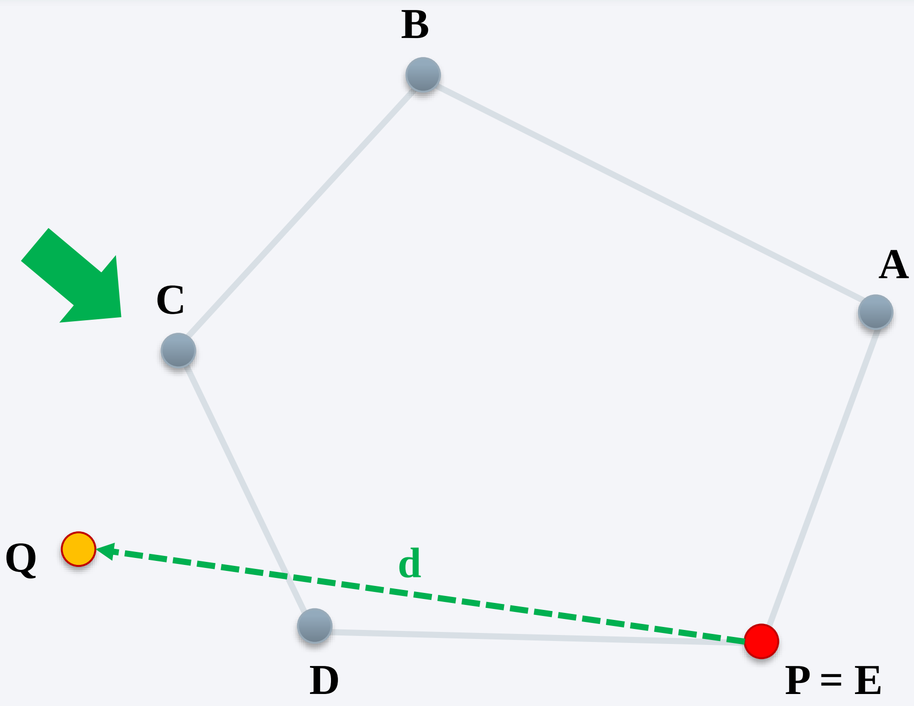
SAT: CASES



$C_1 = (cx_1, cy_1)$
r_1
$C_2 = (cx_2, cy_2)$
r_2
nb_3
nb_0
nb_2
nb_1
n1_2
nt_0
n1_1
nb_3
nb_0
d
nb_1
nb_2

DISTANCE BETWEEN POLYGONS





- The closest points give us the distance between convex polygons
- If the distance is zero, there is an overlapping
- **Useful to calculate compenetrating**

COLLISION RESOLUTION

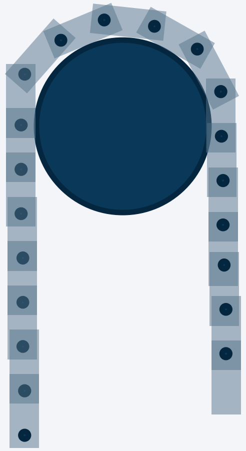
- Update game objects properties when collisions are detected

- Based on constraints
  - Non-penetration constraints
  - Hinge joints
  - Ball Joints
  - …

- Force or Impulse based approaches

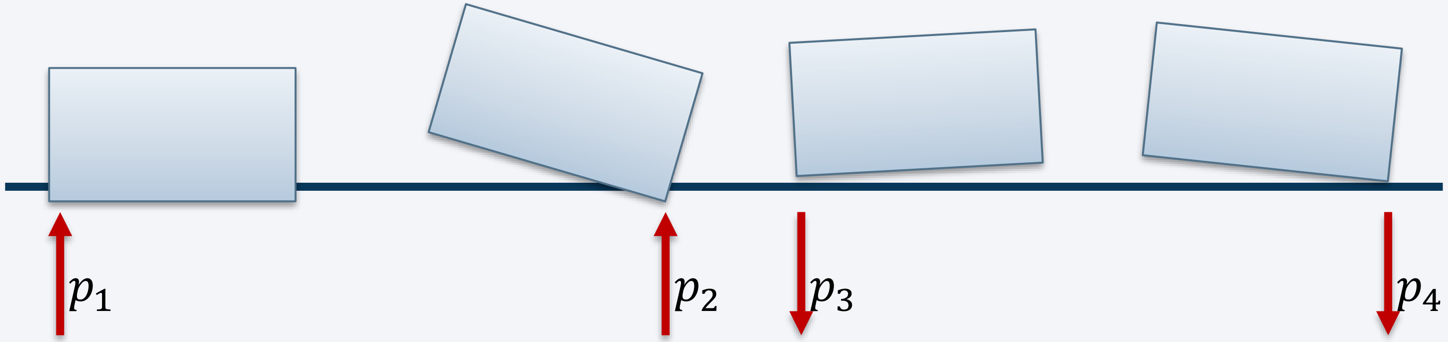
- Optimization
  - E.g. islands technique

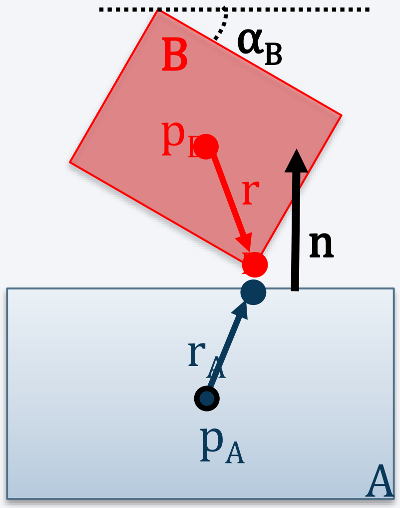
CONSTRAINTS, FORCES AND IMPULSES



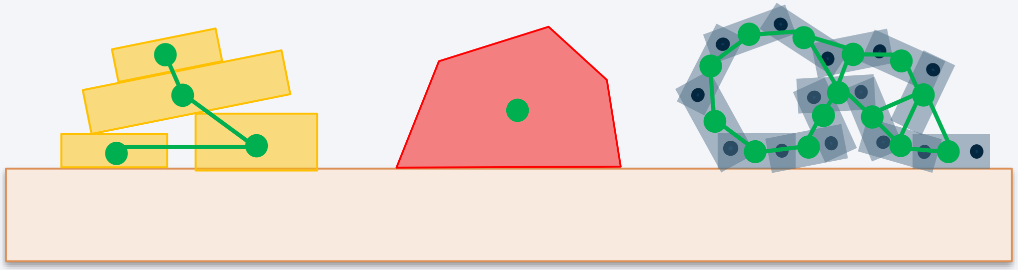
constraints

impulses





islands



forces

QUESTIONS?