Simulation & Animation

3VU

1. Introduction

Reinhold Preiner

SS 2025

Introduction

Reinhold Preiner

Postdoctoral Researcher at the
Institute of Visual Computing

reinhold.preiner@tugraz.at

- **Research:** Rendering, Geometry Processing, Surface-Reconstruction, Visualization
- **Teaching (Master):**
  - Simulation & Animation (SS, 3VU)
  - 3D Computer Graphics & Realism (WS, 3VU)
  - AK IVIS
  - Master-Project and Master-Thesis Topics

Overview

Definition

According to Google:

animate

verb
/ˈanɪmeɪt/

1. bring to life.
   "Prometheus stole fire from heaven to animate his clay men"

2. give (a film or character) the appearance of movement using animation techniques.
   "much-loved characters have been animated in this Franco-Canadian co-production"

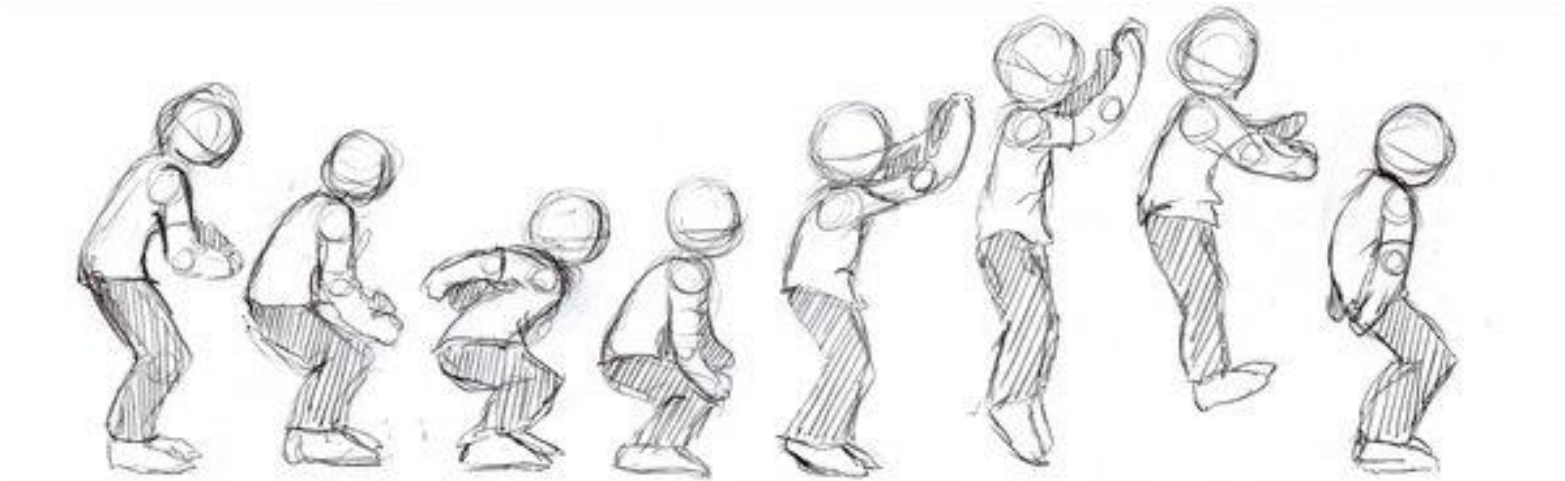
adjective
/ˈanɪmət/

1. alive or having life.
   "gods in a wide variety of forms, both animate and inanimate"
   synonyms: living, alive, live, breathing, sentient, conscious; More

Key goal in artistic representations!

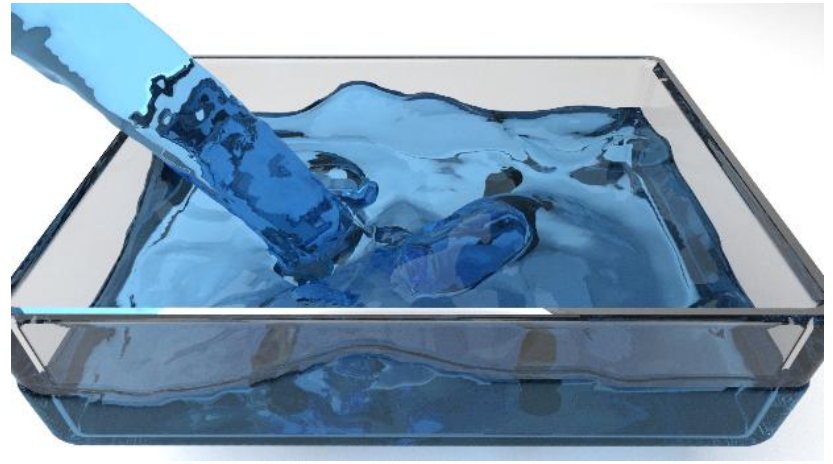
Simulation & Animation

- **Animation:** sequence of still images that give a sense of motion to a scene
- **Simulation:** specific approaches on how to compute these images
- Content of this lecture!



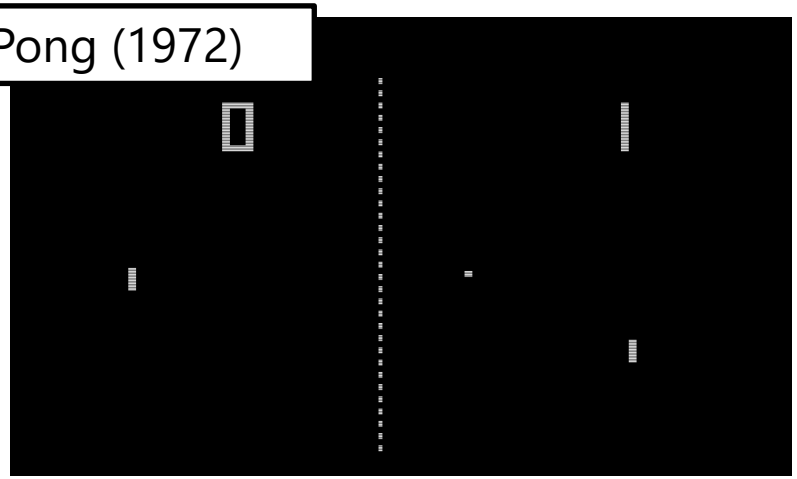
Motivation

- Films
- Computer games
- Commercials / TV
- Scientific simulations
- Training simulations
- Education
- User Interfaces
  (scrolling with inertia, spring-based animations, etc…)
- …



Animation in Video Games

- Animation and simulation are key elements in most video games.


Pong (1972)


Quake (1996)


Uncharted 4 (2016)


F1 2021

Physics Engines

- Simulation and Animation is an important aspect of computer games and CGI, therefore many physics engines have been developed to solve the underlying problems.

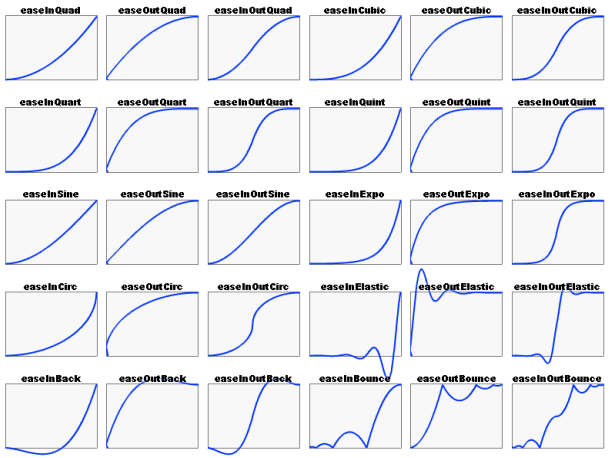
- 2D Physics
  - Box2D, Chipmunk2D, …

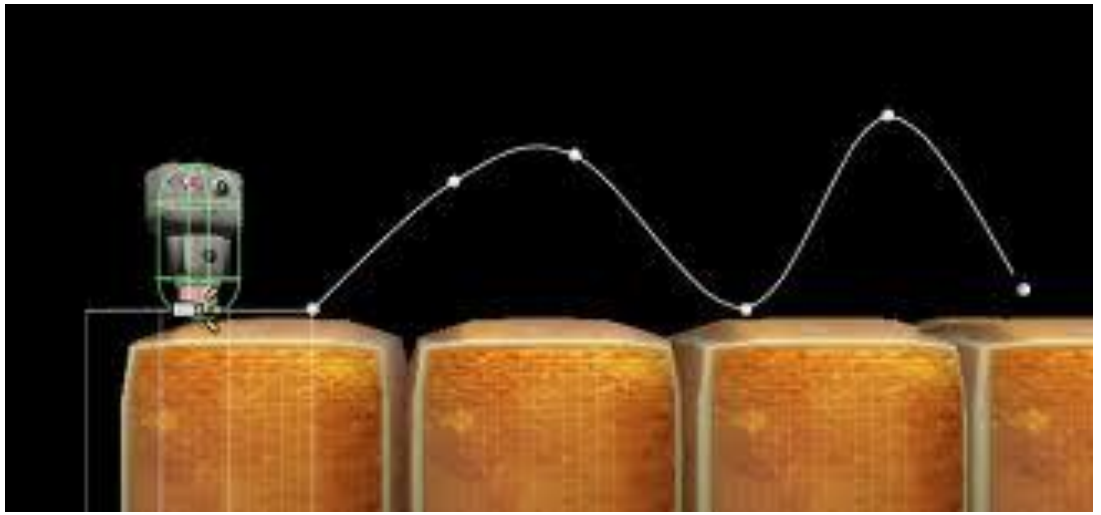
- 3D Physics:
  - NVIDIA PhysX
  - Bullet physics
  - Havok (Half Life 2)
  - …

Half Life 2

Challenges

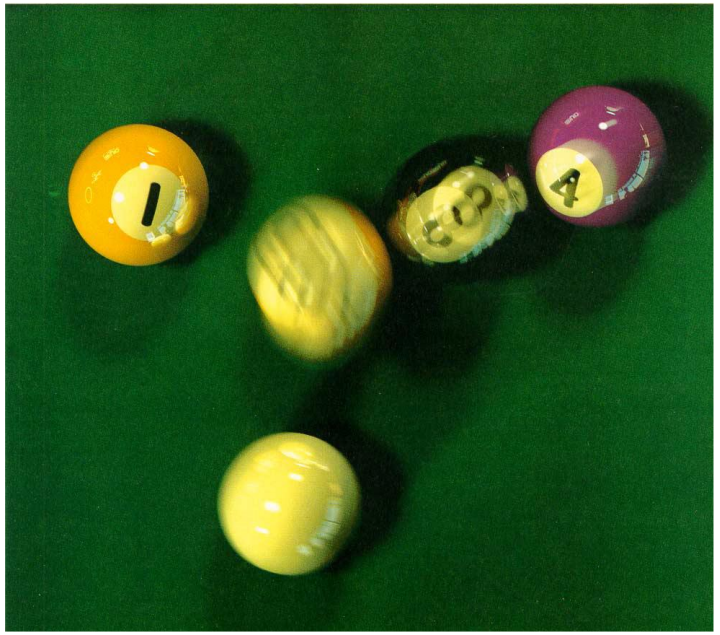
- ***Continuous Motion Paths & Speed Control***
  - → *Suitable Encoding of Transformations & Orientations*

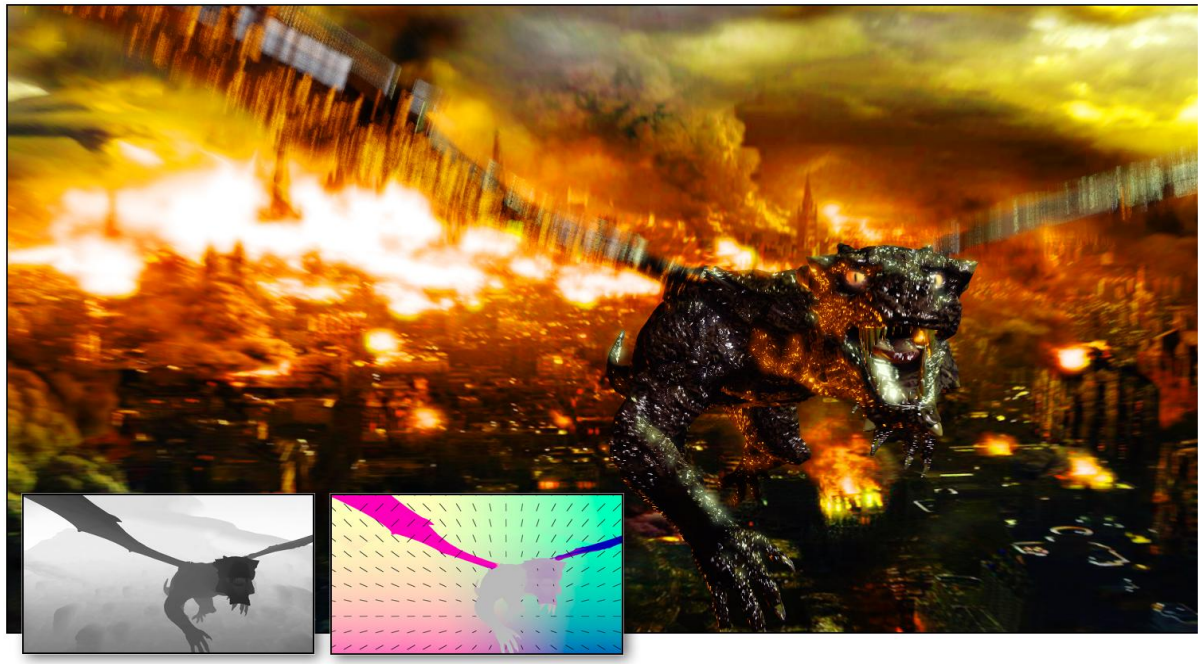




Challenges

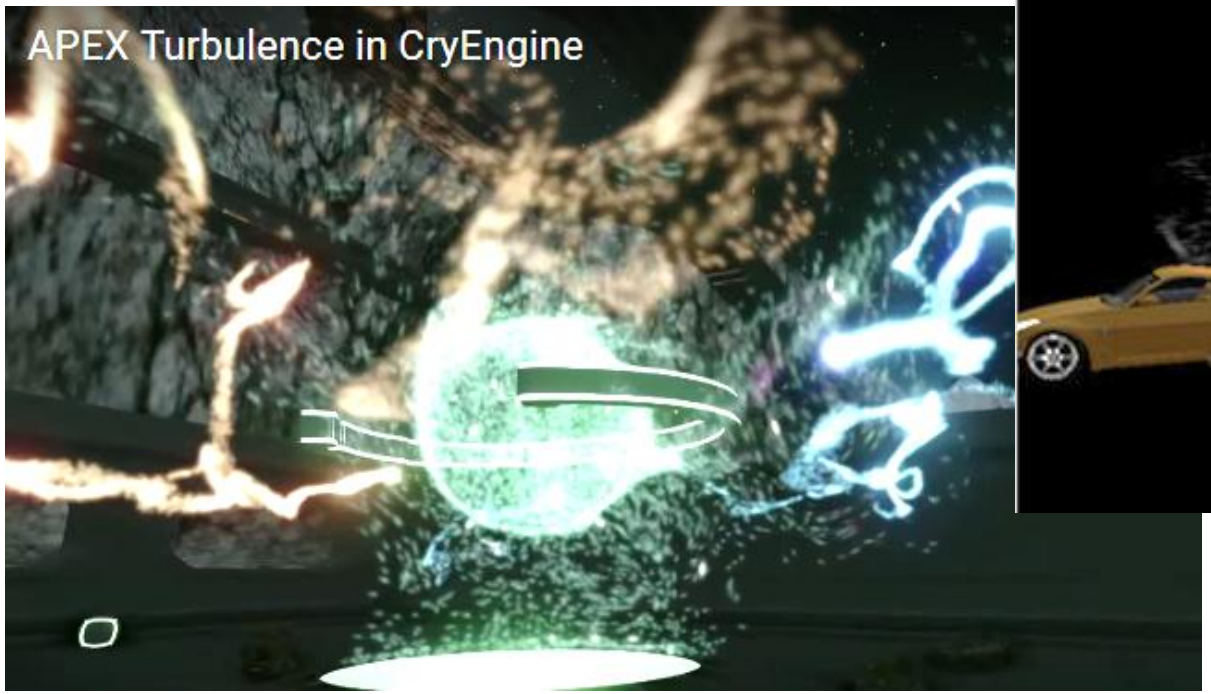
- *Aliasing in Animation / Motion Blur*





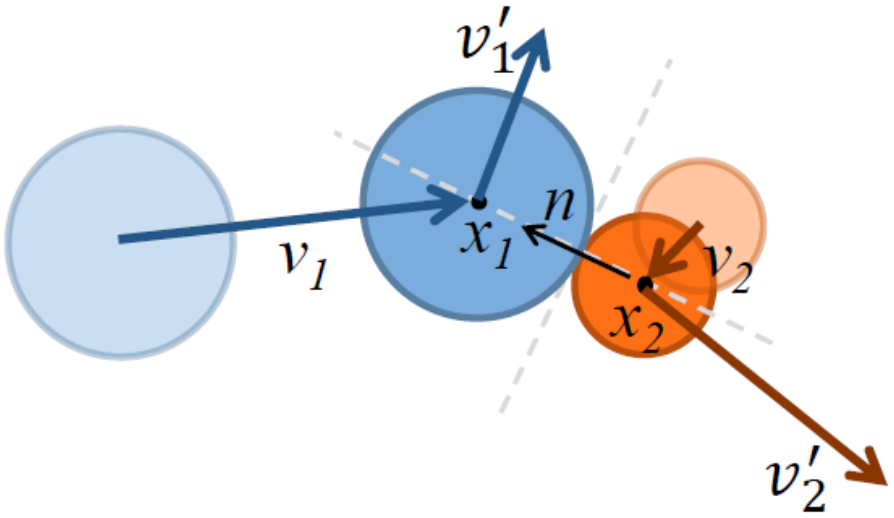
Challenges

- *Physical-based Motion Integration*
- *Particle Dynamics*



Challenges

- ***Collision Detection & Response***
- ***Rigid-body Dynamics***



- ***Rigid bodies*** → objects **do not deform**, lengths and angles are preserved

Challenges

- ***Fracture / Destruction***

NVIDIA PhysX

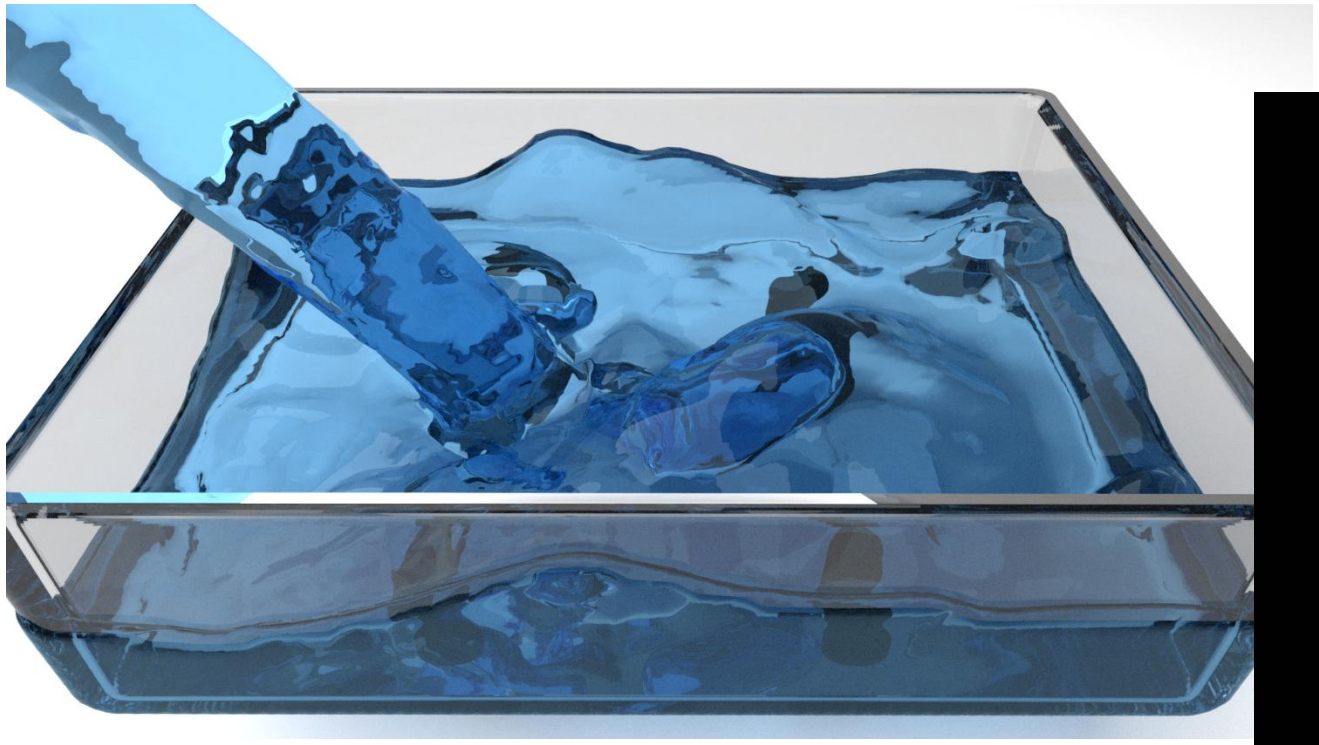
Challenges

- ***Character animation & motion capture***

Source: screencrush.com/motion-capture-movies

Challenges

- ***Fluid Simulation:*** *Liquids and Gases*



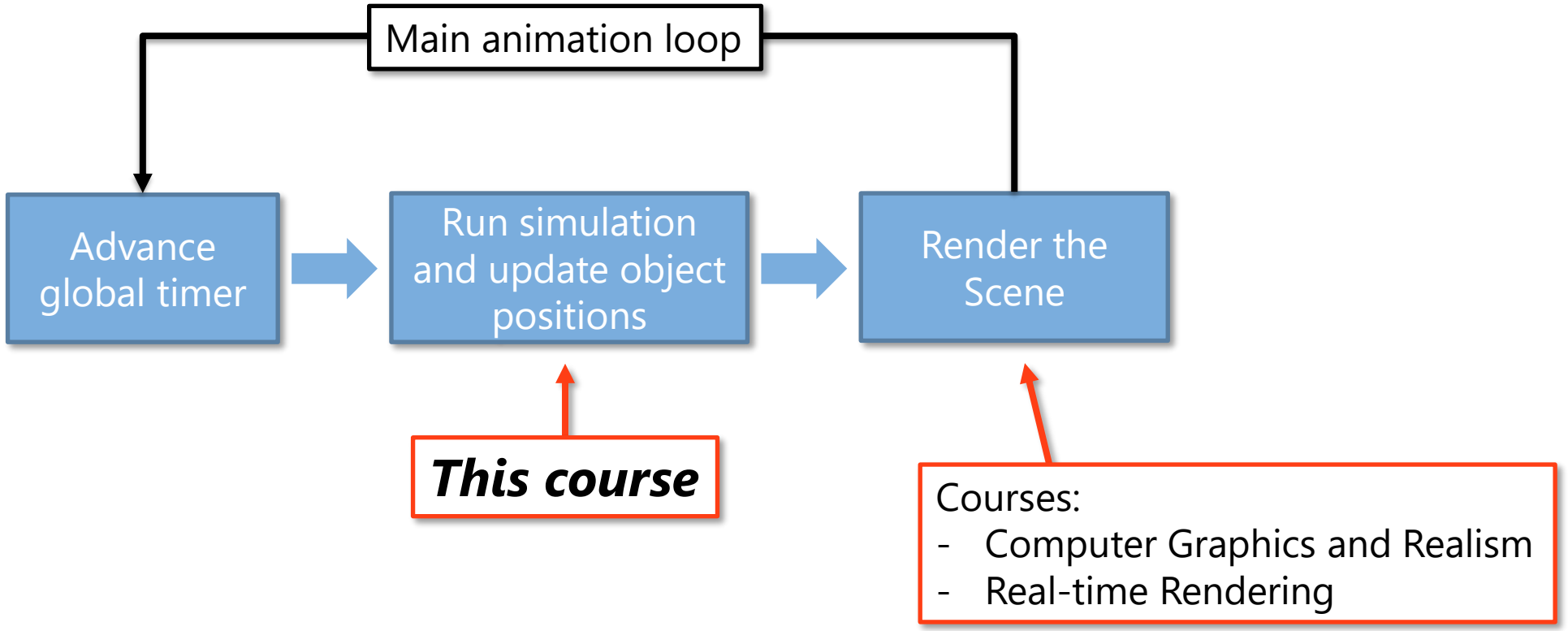
Challenges

- ***Fluid Simulation:*** *Liquids and Gases*

Challenges

- ***Realistic Crowds***

Computer Animation Pipeline



Framerate-Independent Animation Speed

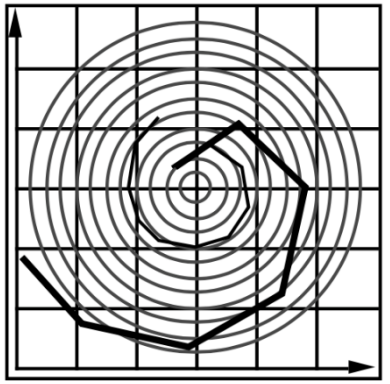
```
function animate(time_now) {
    // time delta since last animation update in seconds
    dt = time_now - global.lastAnimationTime
    global.lastAnimationTime = time_now
    // use time delta to scale animation update
    for each gameObject
        gameObject.pos += gameObject.velocityVector * dt
}


function updateFrame(time_now) {
    animate(time_now)    // make sure animation state is up to date before rendering
    renderFrame()        // actually render the scene state
}
```

Scaling by dt is important to ensure constant animation speed at non-constant framerates!

Decoupled Animation Update Rate

```
dt = time_now – global.lastAnimationTime
```



Updating animation at frame rate frequency can lead to inaccurate or instable animation.

→ Perform animation updates at higher temporal resolution

```
function updateFrame(time_now) {
    animation_dt = desired_time_step  // custom animation delta (anim_dt << dt)
    for (t = global.lastAnimationTime + anim_dt; t < time_now; t += anim_dt)
       animate(t)   // compute animation update at intermediate time steps t
    renderFrame()   // actually render the scene state
}
```

Classes of Animation Techniques

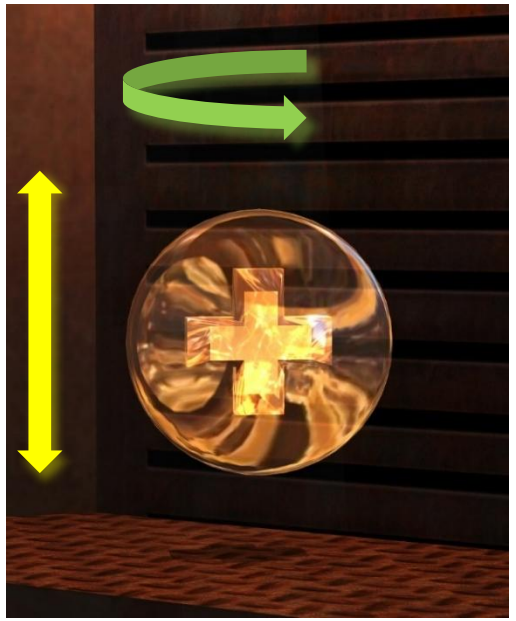
1. Procedural animation
2. Keyframe animation
3. Physically based
4. Synthetic / Data-driven
5. AI-based (Evolved / Trained / Engineered)

1. Procedural Animation

- Animation parameters are controlled by a simple function of time
- **Example:** *position.y* = 10 + **cos**(*time*)

  → y coordinate of the object oscillates between 9 and 11

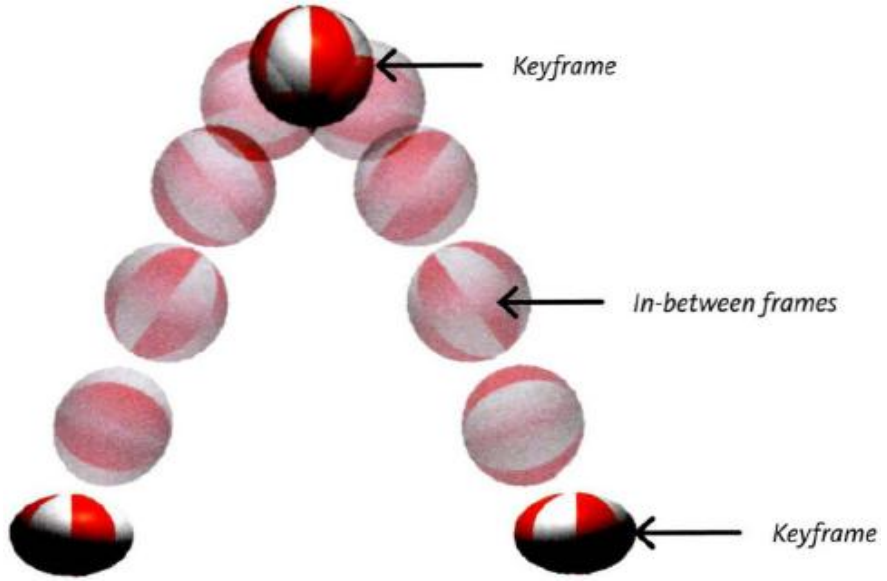
- **Pros:**
  - Explicit artistic control
  - Fast performance for simple functions
- **Cons:**
  - Not physically-based results
  - Some animations cannot be easily expressed as analytic functions



2. Keyframe Animation

- Animation parameters artistically controlled by setting **“key frames”**.
- Intermediate states of the animation are produced by **interpolating** the keyframe values.

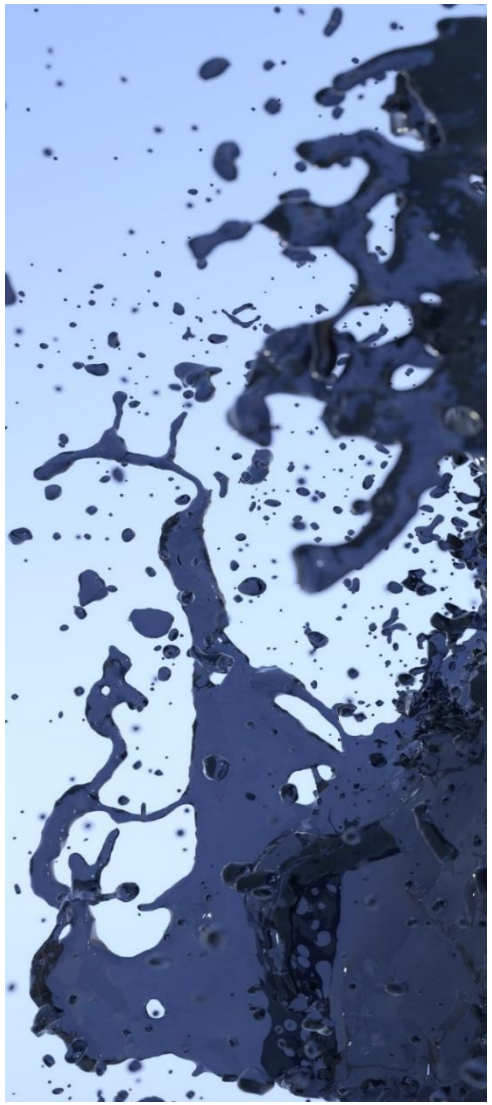
- **Pros:**
  - Explicit artistic control
  - Very fast performance
- **Cons:**
  - Tedious, many key frames often required
  - Not physically-based results
- Various interpolation methods



3. Physically based Animation

- Animation is produced by simulating the laws of physics
- **Examples: particle systems, rigid-body dynamics, fluid simulation, …**

- **Pros:**
  - Very realistic, physically-based results
- **Cons:**
  - No direct artistic control
  - Computationally expensive



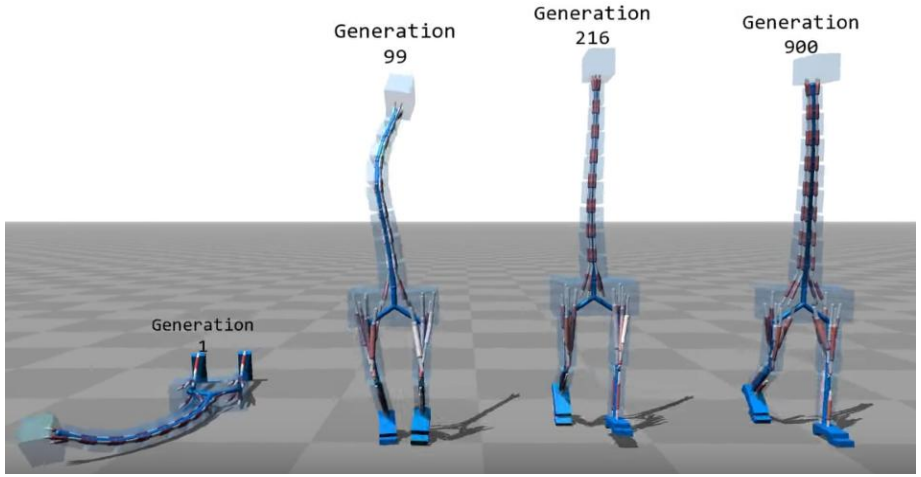
Synthetic / Data-driven Animation

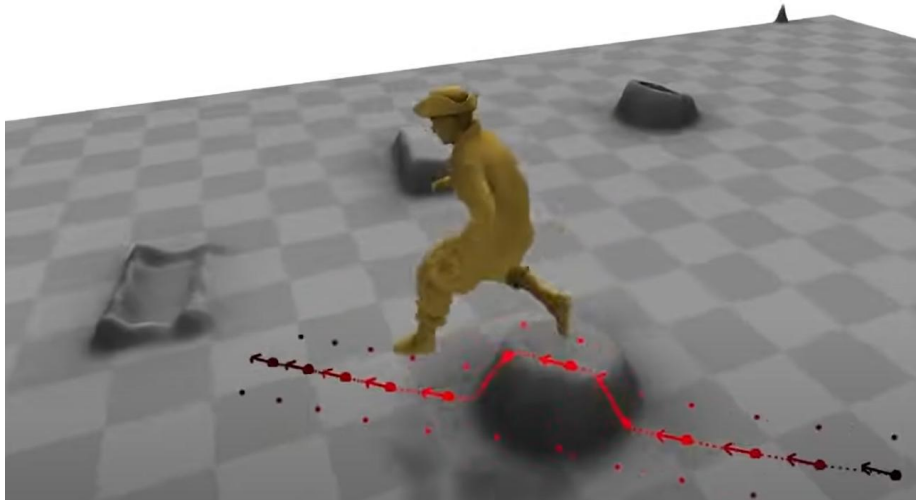
- Animation is produced by capturing and retargeting the motion from real objects to virtual ones.
- Data-Driven Animation
- **Examples:** **Motion Capture (Mocap)**, facial performance capture

- **Pros:**
  - Very realistic
- **Cons:**
  - Often requires expensive hardware setup

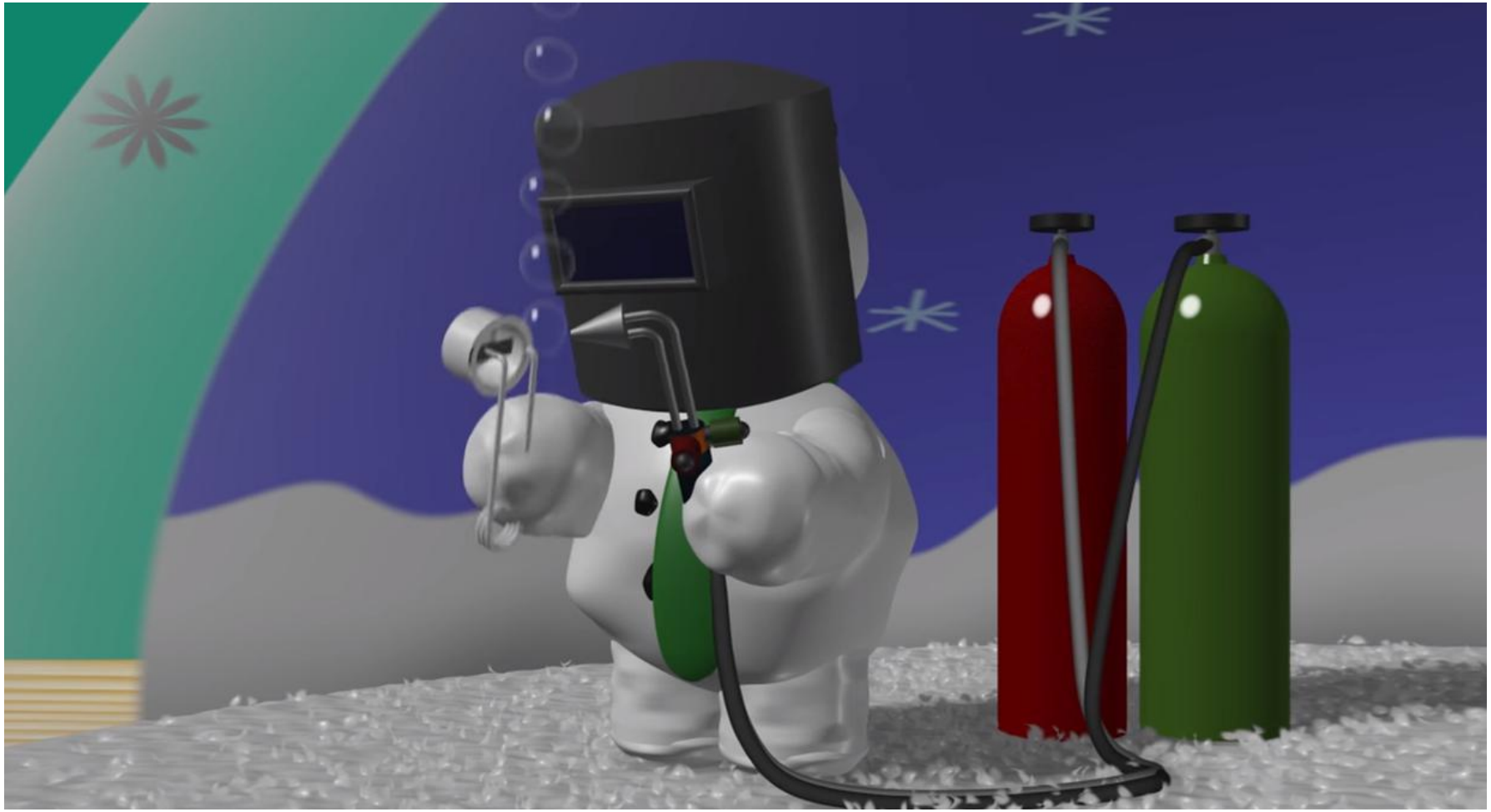
Evolved / Trained / Engineered AI

- Animation is produces by a complex decision system (AI) that reacts on sensory input
- **Evolutionary Approach:**
  Define goals and constraints of an animation system and let an evolutionary optimization process find the optimal animation parameters.
- **Trained Approaches:**
  Train a neural network with appropriate motion and poses for given environmental conditions
- **Engineered:** Rule based, Decision Trees …





Course Organization



Course Organization

- ~10 lectures
- programming assignment **(60 pts)**
  - 2D computer game
  - techniques from simulation and animation
  - groups of 2-3 students
  - Final submission interview on your code
- written exam **(40 pts)**
  - questions from the lecture material
- You have to score at least 50% on both the assignments and the exam to pass the course!

Assignment – 2D Computer Game

- groups of 2 – 3 students*
- type/content: up to your own creativity!
- has to feature **2 techniques** from the lecture **per student:**

1a. Path interpolation
1b. Physically based particle dynamics
1c. Mass-spring systems ***(*required for groups of 3)***
1d. Rigid-body dynamics

At least one of the techs 1a - 1d per student

2. **Voronoi Fracture**
3. **Motion blur**
4. **Hierarchical transformations**

Details on the techniques on later slides …

Programming Assignment – Specs

- Webbased (JS) or Standalone (C++) Executable
- Has to run on the following reference platform
  - Standalone: Win >= 10, Visual Studio 2022, not dependent on specific video card
  - Web: Firefox or Chrome
- Web: HTML5, Canvas, PixiJS
- Standalone: C++ & OpenGL, GLFW, SDL2, SFML, raylib, …
- If we cannot run your submission out of the box (missing DLLs, Linux executable submitted, etc.), it will result in a reduction of points.
- Frameworks/Libs may be used for asset import, controls, graphics
- **No use of game engines,** simulation code (position and state updates, physics, etc.) **has to be implemented yourself!**
- **Framerate-independent** animations!
  → update of animation state decoupled from rendering update

Practical Assignment – 1. Submission

- Game Concept Specifications as PDF (~3 pages), containing
  - Game title
  - Name, matr. number, and email address of the group members
  - Description of the vision/concept of the game:
    - Content/Objects/Items, Aim of the Game, Controls
    - if given, reference to games that inspire your game
  - Specification of platform (web: which JS lib; standalone: which language)
  - at least one schematic mockup (hand-drawn is sufficient)
  - **list of techniques** you intend to implement and
  - description of how they add to the game (game mechanics, or just decoration)
- Uploaded to TeachCenter using the provided Latex Template
- Deadline: **13.03.**
- **Note: By uploading the game concept you will definitely receive a grade.**

Practical Assignment – 2. Submission

- Running Pre-Release, featuring
  - running display pipeline and animation loop (framerate-independent)
  - working input controls (keyboard, mouse)
  - basic objects/primitives as game assets
  - at least one featured technique per group member
  - short PDF commenting your submission (controls, technique, tech code description)
- Executable game files and source uploaded as zip to TeachCenter
- Deadline: **30.04.**

Practical Assignment – 3. Submission

- Final Release featuring all specified techniques
- Executable game files and source uploaded as zip to TeachCenter
- Including a PDF containing:
  - one representative high-res in-game screenshot on page 1
  - User-Doc: Game-Manual (game elements, gameplay, controls)
  - Tech-Doc: how techs are realized and how they can be visualized
- Game Video,
  - Length/ Format: 30 – 60 seconds, .mp4
  - Should show the Game Name on the lower right corner
  - No narration, no group member names, matriculation numbers, or year!
- Deadline: **30.05.**

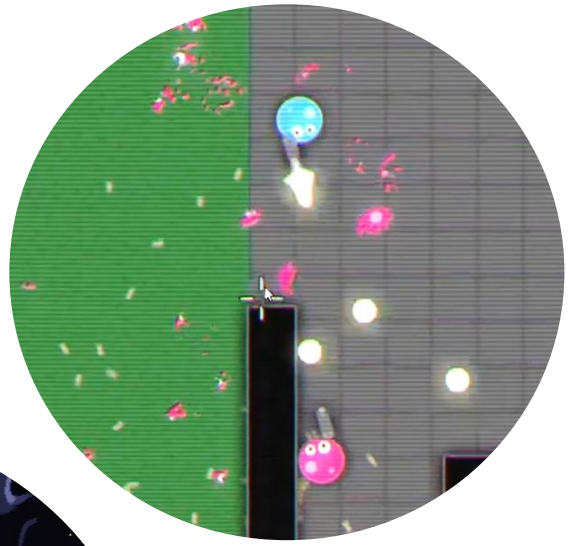
Submission Interviews

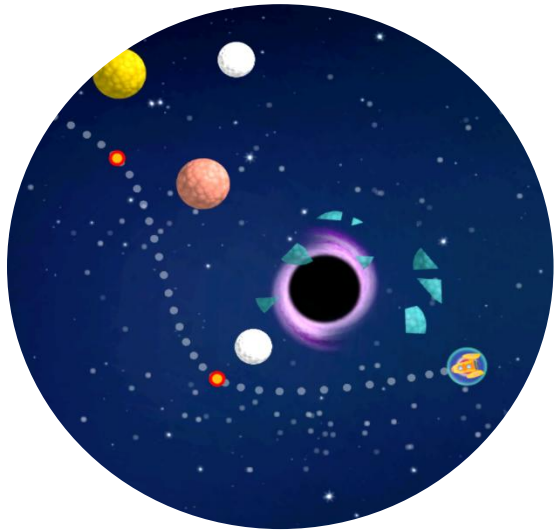
- ~30 min slots per group, registration via Doodle
- Interview on your implementation, live demo, questions about the code

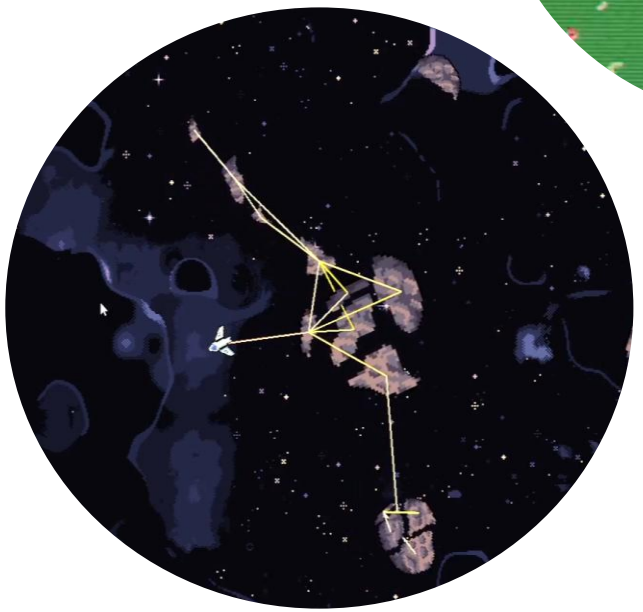
- Dates: **16.06. – 20.06.**

Written Exam + Game Event

- **Written Exam** (~45 min) – registration via TUG Online
- Final closing event - **Game Event**
  - ~10 min Live Presentations of selected games
- Date: **23.06.**







Passing the Course

- For a **positive grade**, you have to
  - **Pass the submission interview** (know/understand your code)
  - **score ≥50% (20 pts) on the exam**
  - **score ≥50% (30 pts) on the assignment**

  } ≥50/100 pts in total

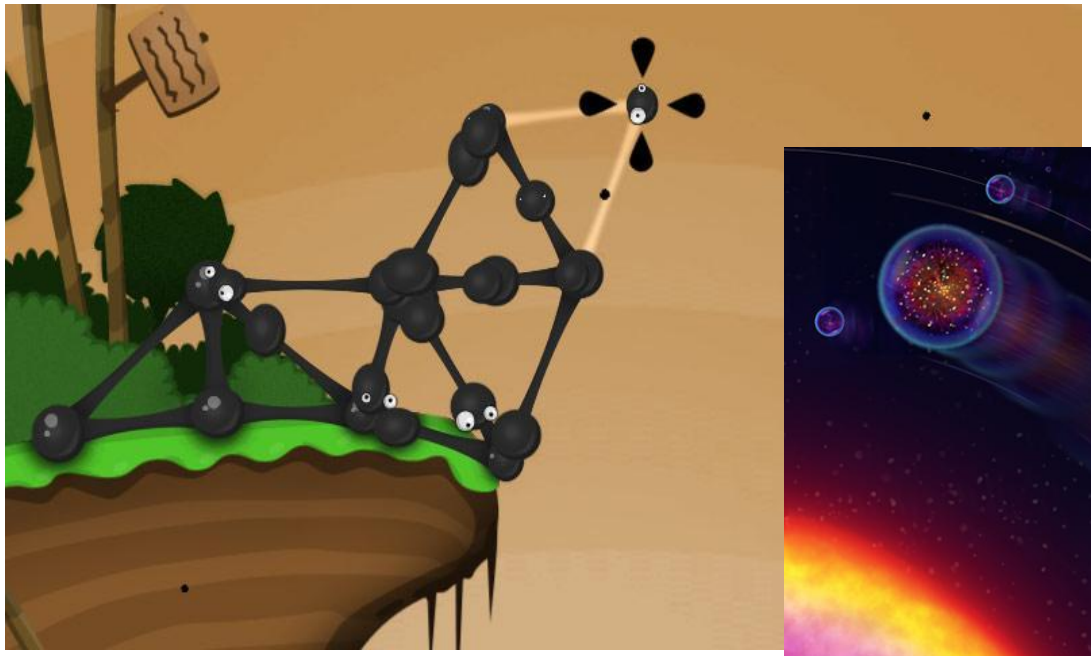
- To improve on a negative grade, repetition of the oral exam is possible within 4 weeks after the end of the course.
- Individual requests for a second exam per email to reinhold.preiner#tugraz.at

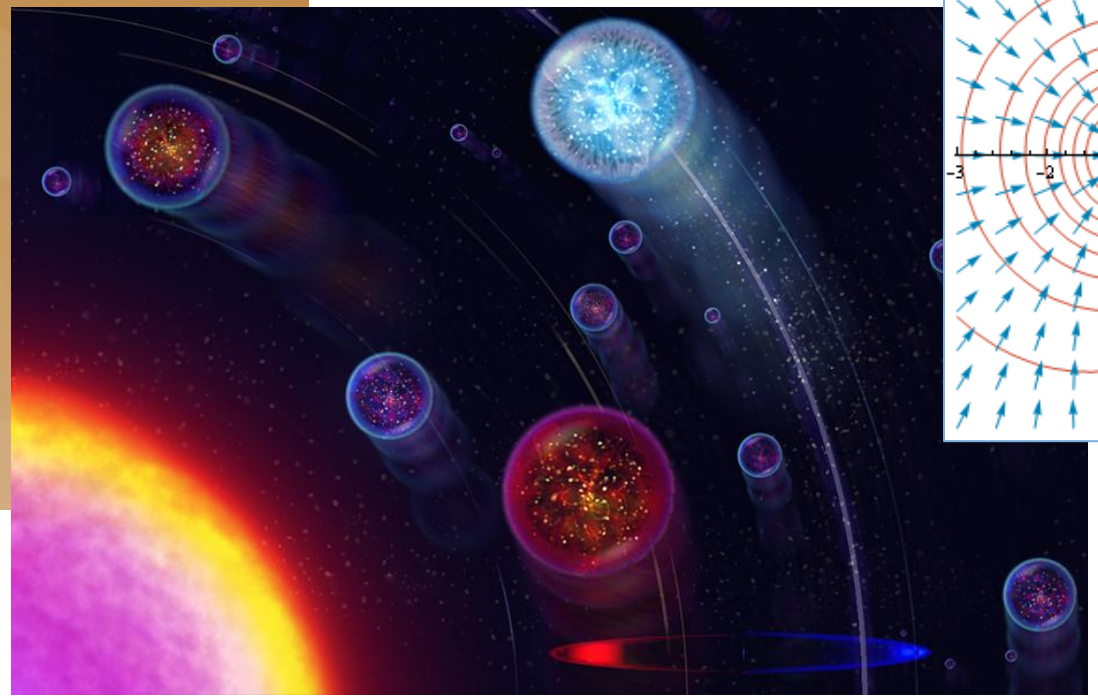
Plagiarism

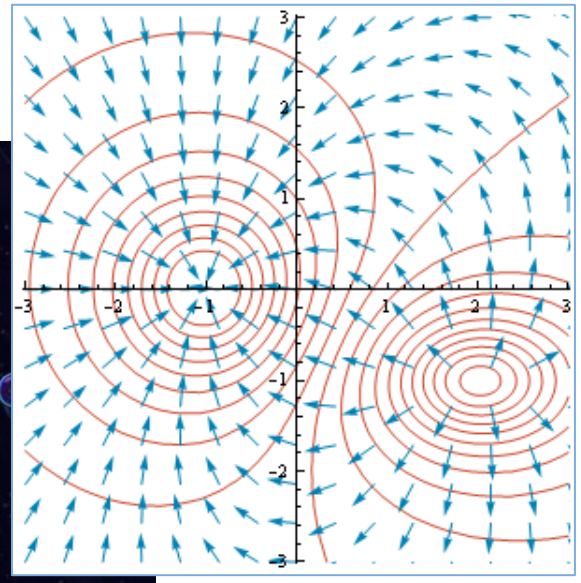
- Discussion of ideas (TC forum) is encouraged, sharing code is prohibited.
- Any external resources (code samples, tutorials) used have to be referenced in your documentation.
- But: **The final submitted tech code has to be your own original work!**

- Submitting code that is not your own or failing to cite external sources is considered plagiarism, and graded accordingly.

- Plagiarism checks will be applied on submitted code.







Assignment Techniques

1a. Path Interpolation

- Your game contains the movement of one or more objects along a Catmull-Rom spline at **controlled speeds** (at least one non-constant motion, e.g., ease-in/ease-out).
- Splines should have **at least 3 segments** forming a curve
- Spline evaluation is implemented by yourself
- Control points of the spline may be dynamic, manipulated by the user, …

Flight Control (Firemint)

1a. Path Interpolation (2)

- Required Visualizations (switchable)
  - *Spline curve*
  - *Control points and arc-length table samples shown as points on curve*
- Required adjustable controls
  - *Traversal Speed*
  - *Animation Update Rate*
- *Practical learning targets:*
  - *Spline-Interpolation*
  - *Arc-Length Parametrization*
  - *Speed control*

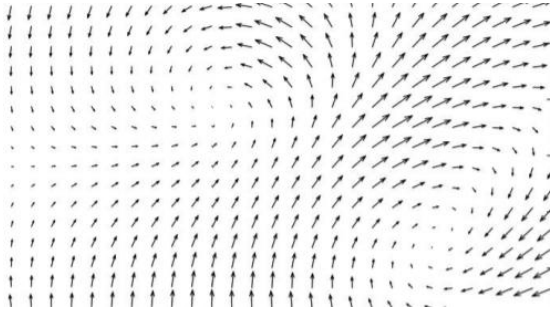
  (→ Lecture 3. Interpolation)

Filename: ***PathInterpol****.js/h/cpp*
- Curve Datastructure
- Arc-Length Table Preprocessing/Sampling
- Arc-Length Parametrized Curve Evaluation
- Visualizations

Flight Control (Firemint)

1b. Physically based Particle Dynamics

- Game contains a few particle-like objects that move according to (Newtonian) forces
  - Objects should have a sufficient size (and life time) to visually follow and asses its path
  - „Particle-like“: objects are treated as singular points (only mass + position, orientation and size disregarded for simulation)
- Implement an apropriate ODE solver that moves the particles according to the forces using **4th-order Runge-Kutta Integration**.
- Use at least three different **non-constant** force sources acting at particles at the same time:
  - Radial (gravity of multiple planets or mouse pointer)
  - Pre-defined force vector field (grid-based or analytic)



Osmos (Hemisphere Games)

1b. Physically based Particle Dynamics (2)

- Required Visualizations (switchable)
  - *Force field (vector grid)*
  - *Object trajectories of the last few seconds*
- Required adjustable controls
  - *Switch between RK-4 and simple Euler integration*
  - *Animation update rate / step size h*
- *Practical learning targets:*
  - *Force-based Particle Dynamics*
  - *RK-4 Integration*

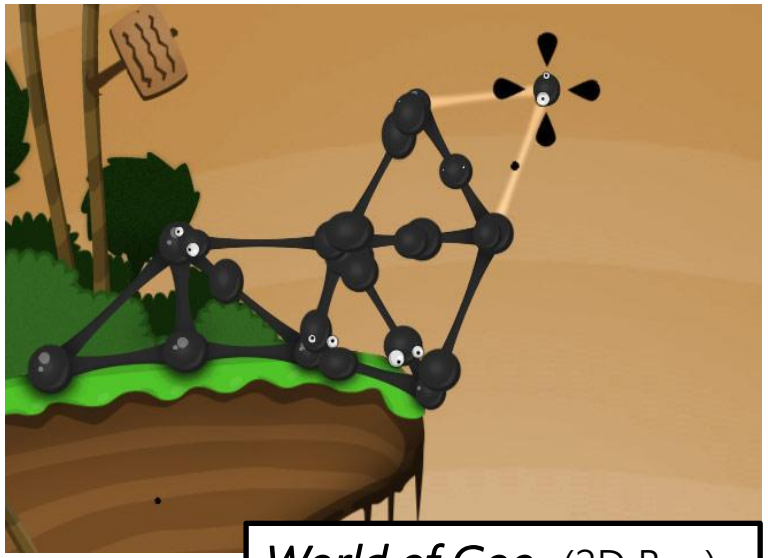
  (→ Lecture 5. Physically based Animation)

FileName: ***ParticleDynamics****.js/h/cpp*
- Particle Data Structure
- Definition and application of forces
- 4th-Order Runge-Kutta-Integration
- Visualizations

Osmos (Hemisphere Games)

1c. Mass-Spring-Systems*

- Your game contains a mass-spring system that interacts in some way with the user and behaves according to physical forces.
- You implement an ODE solver that moves the mass-spring object according to the given forces using **Velocity-Verlet Integration**
- Your engine uses **Hookean forces** and at least **one external force** (e.g., gravity, wind, …)
- ** tech has to be features in groups of 3 students!*



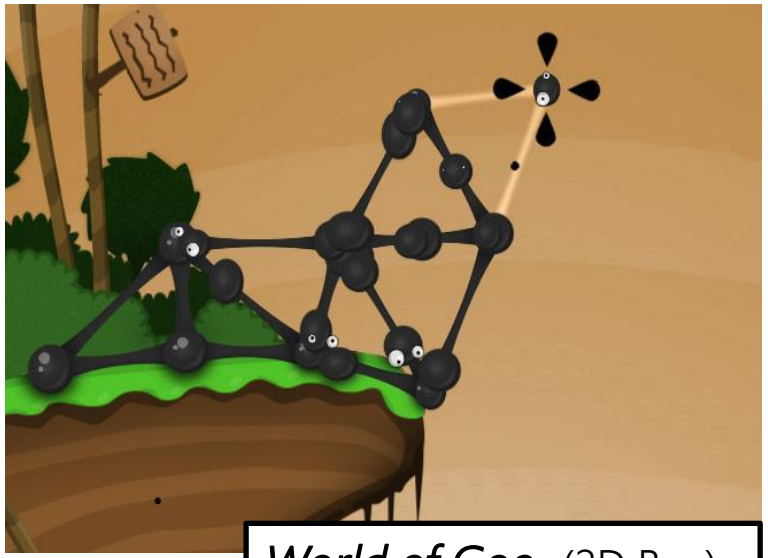
World of Goo (2D Boy)

1c. Mass-Spring-Systems (2)

- Required Visualizations (switchable)
  - *Graph edges color-coded by spring strains*
- Required adjustable controls
  - *Spring Stiffness k*
  - *Animation update rate / step size h*
- *Practical learning targets:*
  - *Mass-Spring System Definition*
  - *Hookean Forces*
  - *Velocity-Verlet Integration*

  (→ Lecture 5. Physically based Animation)

FileName: ***MassSpring****.js/h/cpp*
- Data Structures
- Hookean Force Calculation
- Velocity-Verlet Integration
- Visualizations



World of Goo (2D Boy)

1d. Rigid-Body Dynamics

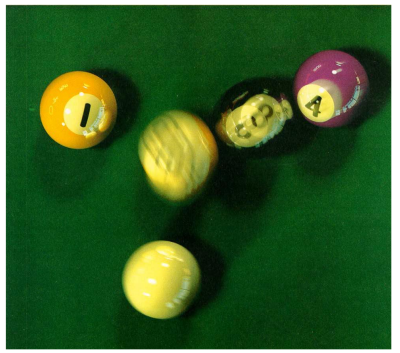
FileName: ***RigidBody****.js/h/cpp*
- Data Structures
- Collision Detection
- Collision Resolve (Momentum Updates)
- Velocity-Verlet Integration
- Visualizations

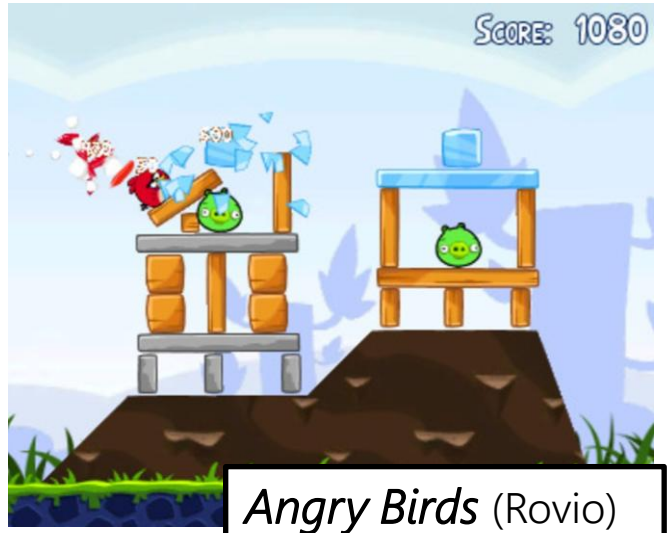
- Game objects have **linear and angular momentum.**
- Your game **detects collisions** between these objects using **cuboid** (required) and spherical (optional) **bounding proxies**.
- Objects **react to elastic collisions** by changing their momentums based on physical forces (bouncing off, adjusting rotation, etc.) using **Velocity-Verlet Integration.**

- Required Visualizations (switchable)
  - *Current momentum vectors*
  - *Show Collider Primitive (Box/Sphere)*

- *Practical learning targets:*
  - *Handling elastic collisions*
  - *Linear and angular momentums*

(→ Lecture 5. Physically based Animation)

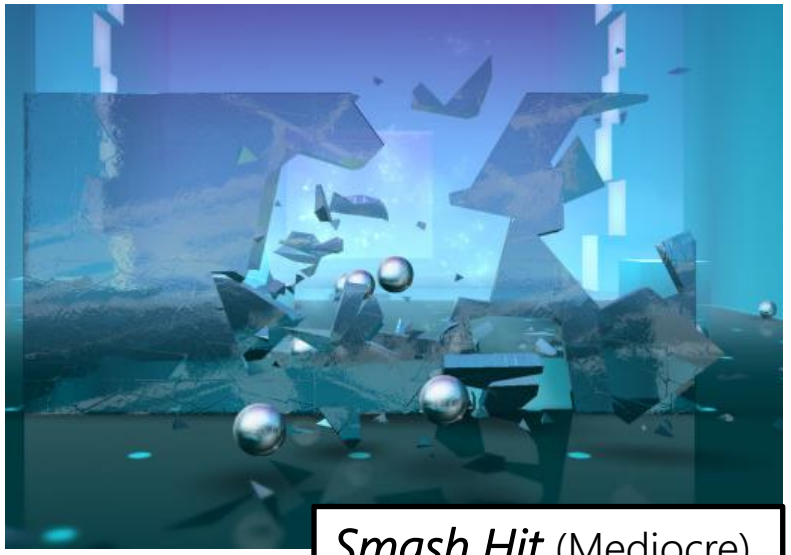




Angry Birds (Rovio)

2. Voronoi Fracture

- Your game contains 2D objects that are dynamically fractured into a number of small pieces using **pixel-based *Voronoi Fracture.***
- 2D objects (images) have a complex, **non-rectangular silhouette.**
- Voronoi seeds of the fracture pattern are computed **dynamically** at runtime **based on given causalities** (bullet impact, punch location).
- Voronoi cells are computed based on a true pixel/grid-based distance field, which must be additionally **noised.**
- Examples:
  - Shattered glass
  - Fragged enemies



Smash Hit (Mediocre)

2. Voronoi Fracture (2)

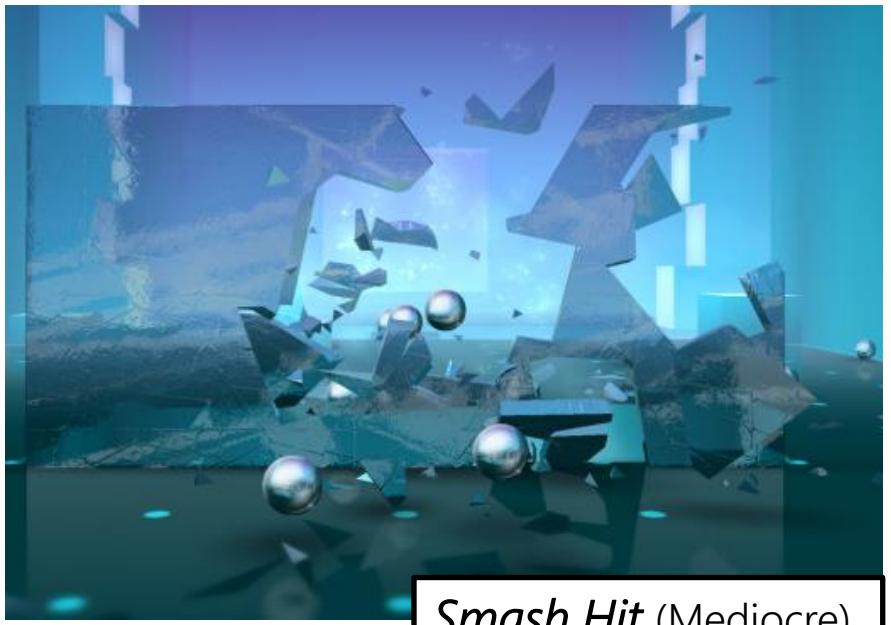
- Required Visualizations (switchable)
  - *Voronoi Seed Points*
  - *(Noised) Distance fields to cell boundary (color gradient)*

FileName: ***VoronoiFracture****.js/h/cpp*
- Data Structures
- Fracture Logic
- Visualizations

- Required adjustable controls
  - *Additional Noise-Overlay on/off*

- *Practical learning targets:*
  - *Plausible Fracture Patterns*
  - *Implicit Shapes*
  - *Voronoi-Cells*
  - *CSG Operations*

(→ Lecture 6. Voronoi Fracture)



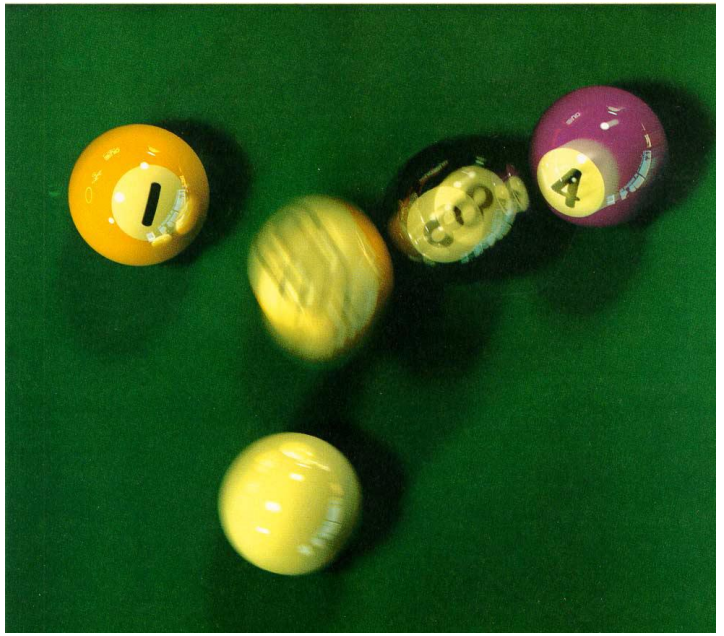
Smash Hit (Mediocre)

3. Motion Blur

- Your game contains **high-speed object movement** that might not be properly resolved at slow animation update.
- You implement **two** motion blur strategies to counter temporal aliasing
  1. **stochastic motion blur**

     OR

     post-process motion blur
  2. **temporal supersampling**
- allow to dynamically switch the mode at runtime in your game (differences should be visible)



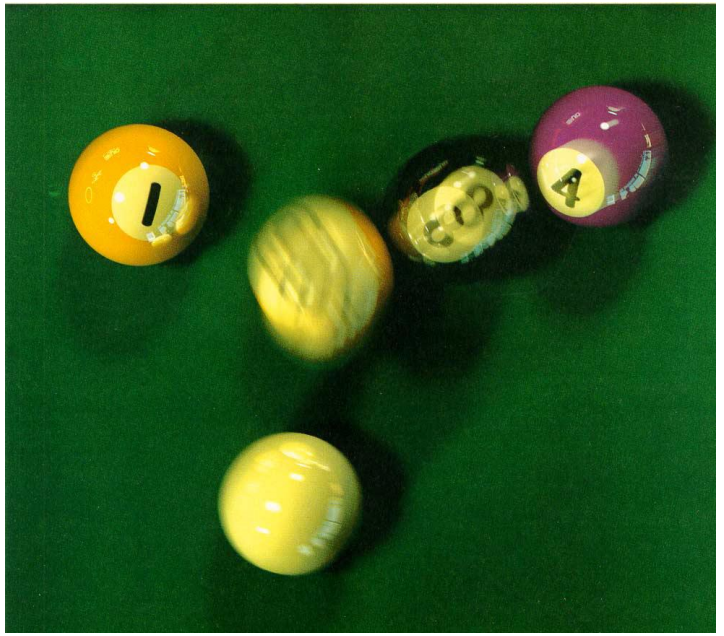
3. Motion Blur (2)

- Required adjustable controls
  - *Switch between the two motion blur modes (differences should be visible)*
  - *Game pausable to see freezed motion blur*

- *Practical learning targets:*
  - *Temporal anti-aliasing*
  - *Motion blurred animations*

  (→ Lecture 4. Motion Blur)

FileName: ***MotionBlur****.js/h/cpp*
- Stochastic or Postprocess Motion Blur Logic



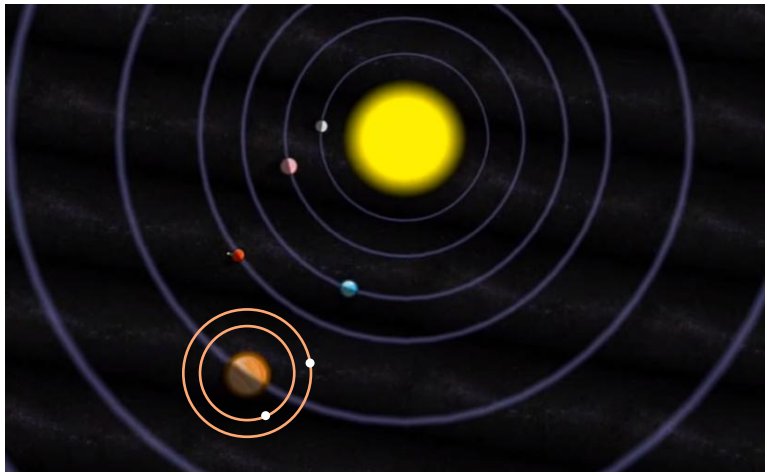
4. Hierarchical Transformations

- Game objects move relative to other parent objects → hierarchy of reference systems
- Your game manages a hierarchy of transformations **(translations and rotations, min. 3 levels after root, multiple objects per level)** and propagates parent transformations down to its children.

FileName: ***SceneHierarchy****.js/h/cpp*
- Hierarchical scene graph data structure
- Hierarchical transformation logic

- Objects and their trafos might be controllable by the player.
- Required Visualizations (switchable)
  - *Object trajectories of the last few seconds*
- *Practical learning targets:*
  - *Hierarchical Scene Graphs*
  - *Transformation Representations*

  (→ Lecture 1 and 2)



Compatible and conflicting Techniques

- Procedural and physically-based techniques cannot be easily combined:
  An object can either move due to
  - Procedural motion [**Path interpolation (1a)** or **hierarchical transformations (4)**] ***OR***
  - be treated as a physical **particle-like object (1b)**, ***OR***
  - be treated as a physical **mass-spring system (1c)**, ***OR***
  - be treated as physical **rigid bodies (1d)**
- But: Combinations are possible to a certain extent:
  - a hierarchical trafo system (4) can move along an interpolated path (1a)
  - a path curve (1a) can be part of a hierarchical system (4)
  - amass spring system (1c) can be moving along a path (1a) or a hierarchical system ()
- Different objects in the game can be modeled due to different techs.

General Technical and Formal Requirements

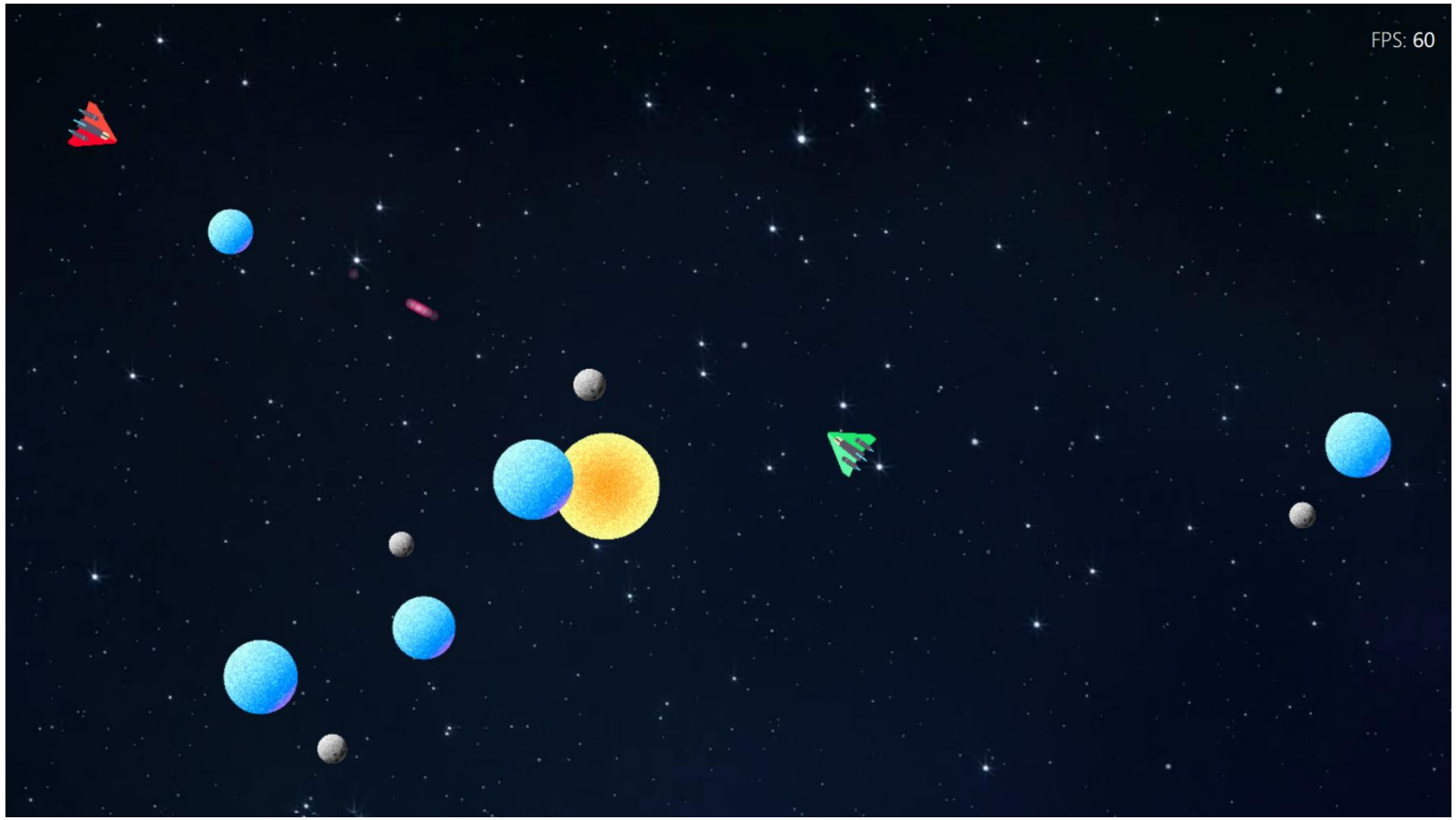
- Fully implemented gameplay (game-over, scores, etc.) not required for full assignment points, will however bring bonus points.
- Frame rate and simulation/animation update rate should be dynamically adjustable.
- Game pause (animation and screen) should be possible at any time (showing current screen).
- Each individual tech code has to be found in a single file with the specified file name (see tech specification in previous slides).
- Submission zip-file content: structure and naming convention:
  - bin/ game executable
  - res/ any game resources, images, textures, shaders, models
  - src/ game source
  - doc.pdf documentation created using the provided LaTeX template
  - video.mp4 game video
- **Sources** of all assests (images, models, sprites, sounds) **have to be documented**, whether self-made, or third-party (**only FREE ones allowed**).

Failing these requirements will lead to a reduction in points!

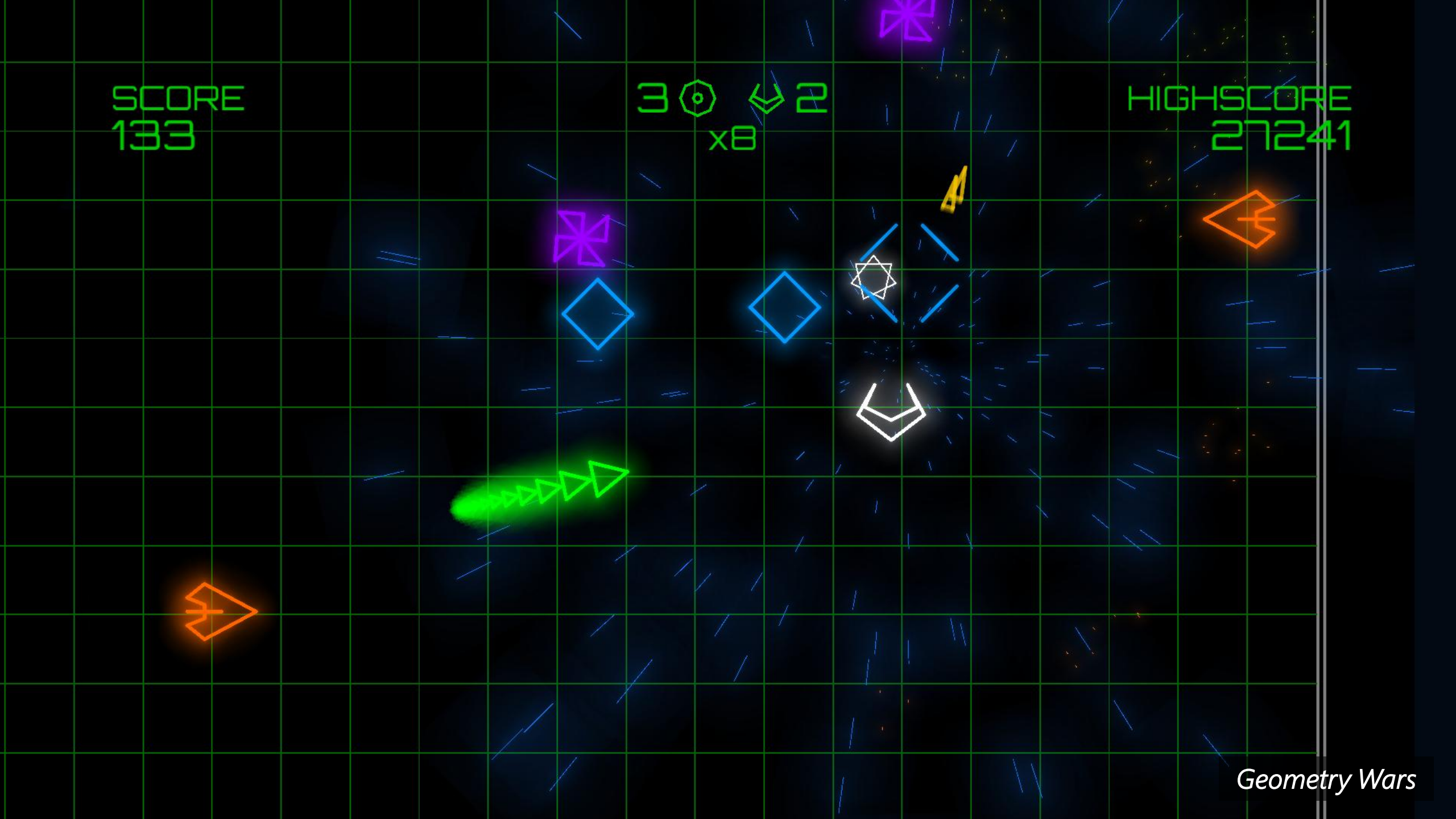
Notes and Hints

- Visuals are Secondary!
  - Focus on game mechanics and S&A techniques first, then graphics
  - Start with primitive mockup shapes, replace later
  - Or use simplistic visual style altogether
- Sounds are appreciated, but not mandatory
- Webbased Apps are probably easier to start with
- Standalone apps allow more flexibility

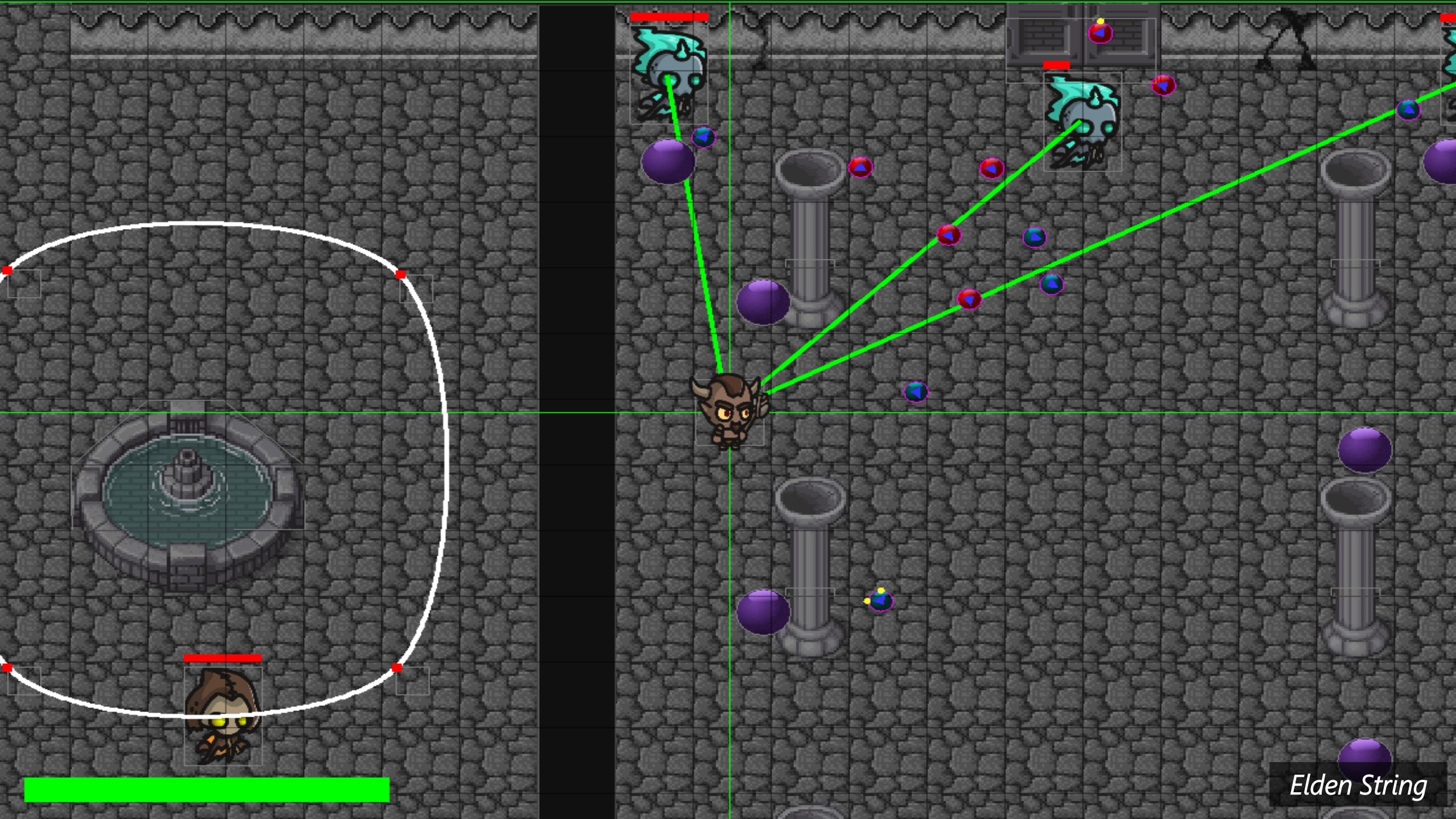
Badland (Frogmind)



Impressions from previous years



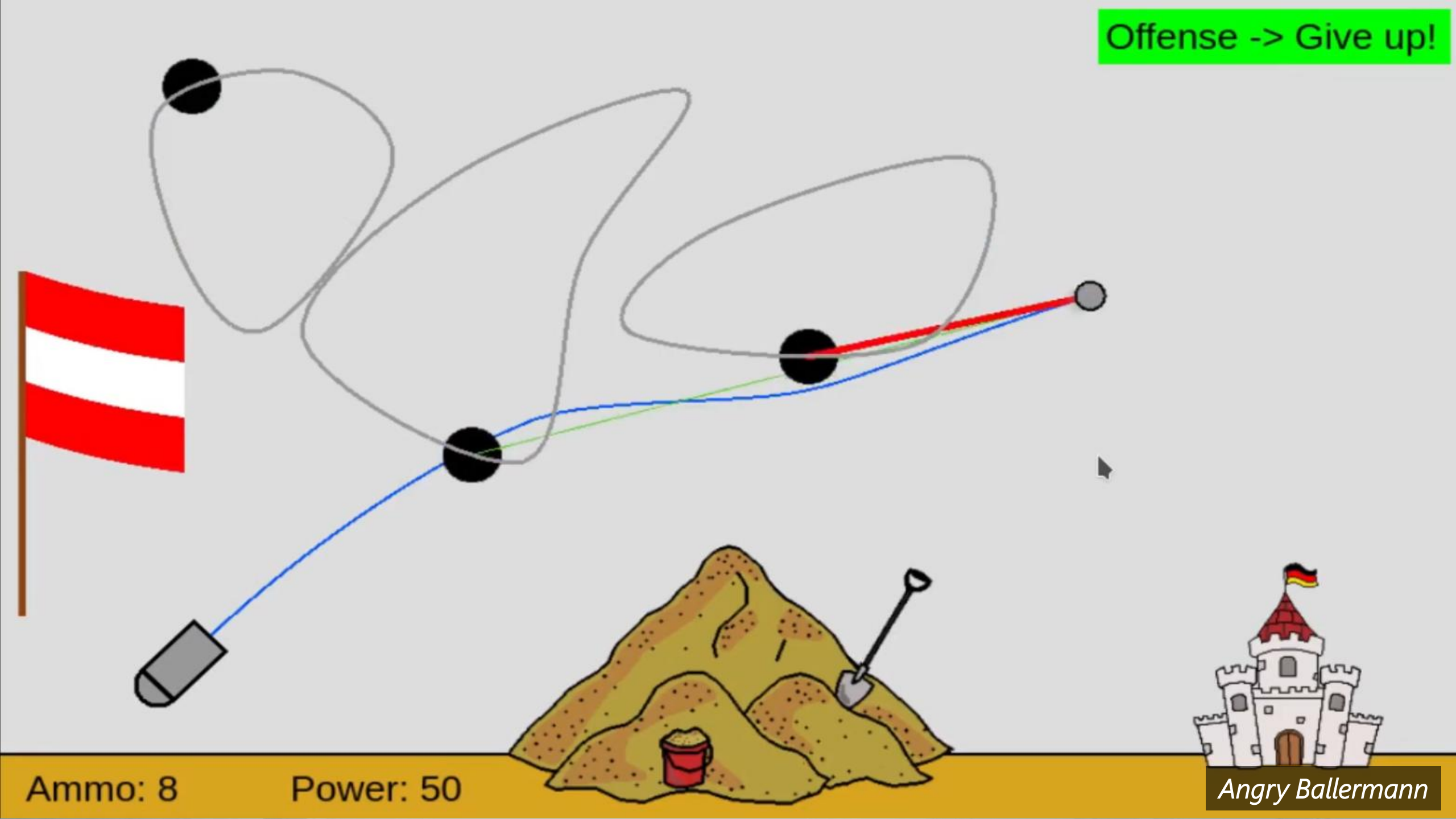
Geometry Wars


Elden String


0:33:13
Fury Road

Planet Pool


Under Construction
Weapon: [Bazooka: ∞] [Grenade: 10] [Missile: 1]
[Skip] [Fire!]
Team: Yaw Dawgs
Under Construction


Offense -> Give up!
Ammo: 8
Power: 50
Angry Ballermann

Intelligent Space Missiles


Animation Tick Interval: 5 ms
Renderer FPS: 67
Score: 10
Balls of Destiny


Cybergunk 2007


Z/Y
X
C
HEALTH: 4
SCORE: 0000002133
ShootyLooty

Links

WebGL

- HTML5/Canvas:
  - www.w3schools.com/html/html5_canvas.asp
- PixiJS www.pixijs.com

C++/OpenGL

- GLFW www.glfw.org
- SFML www.sfml-dev.org/tutorials
- SDL www.libsdl.org
- GLM glm.g-truc.net
- Eigen eigen.tuxfamily.org
- Raylib www.raylib.com

After this lecture

1. Team grouping!
   - Find colleagues outside lecture hall or post to TC forum
2. Think about a cool game concept that features the requested techs (2 techs per group member)
3. Submit your game concept until 13.03.

Tentative Timeline 2025

- 3.3. Introduction
- 10.3. Transformations + Orientation Representations
- 13.3. 1. Submission Deadline – Game Specs
- 17.3. Path Interpolation
- 24.3. Motion Blur
- 31.3. Physically based Animation I
- 7.4. Physically based Animation II
- **14.4-27.4. Easter Holidays**
- 28.4. Fracture
- 30.4. 2. Intermediate Deadline
- 5.5. Character Animation

Tentative Timeline 2025

- 12.5. Fluid Dynamics
- 19.5. Crowd Simulation
- 26.5. Buffer Lecture
- 30.5. 3. Submission Deadline – Final Game
- 2.6. Buffer Lecture
- **9.6. Pentecost/Pfingsten**
- 16.6. Q&A Recap
- 16.-20.6. Submission Interviews
- 23.6. Exam + Game Event