Game Technology

Lecture 8 – 12.12.2017
Physics 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Dipl-Inform. Robert Konrad
Polona Caserman, M.Sc.

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

Overview

TECHNISCHE UNIVERSITÄT DARMSTADT

Today

- As easy as possible
- Build a simple demo with
  - Particle system
  - Colliding spheres
- Understand the basic principles

Next block

- Build upon what we have learned
- Look at more complicated case
- Apply the physics in a game-like application

Background

„Marbellous“

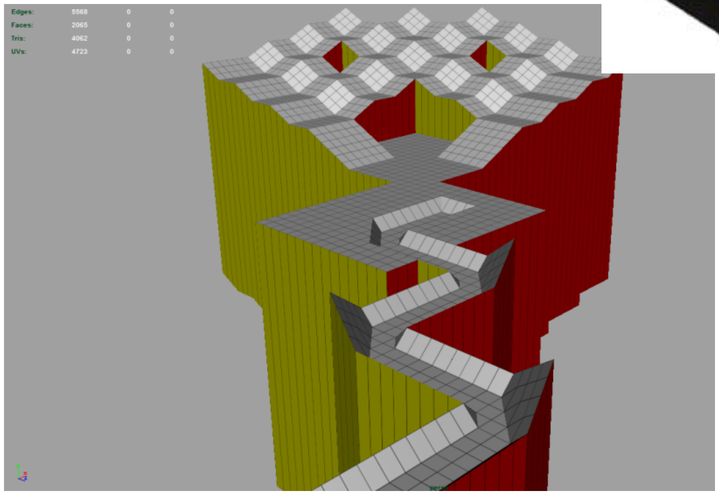
- Clone of „Marble Madness“ (1984)
- Roll a marble through a maze

Ball Physics

- Apply force based on key inputs
- Bounce off off the level geometry
- (Fall from too high)

Level

- Provided as a mesh
- „2D in 3D“



Tony Hawk's Pro Skater 5 (2015)

https://www.youtube.com/watch?v=JIXkRXYbKaI

Physics gone wrong…

TECHNISCHE UNIVERSITÄT DARMSTADT

Skyrim

https://www.youtube.com/watch?v=O2UDHkTITMk

Skate 3

https://www.youtube.com/watch?v=UaUR6u8nHoM

Assassin‘s Creed

https://www.youtube.com/watch?v=WyovOrA64B8

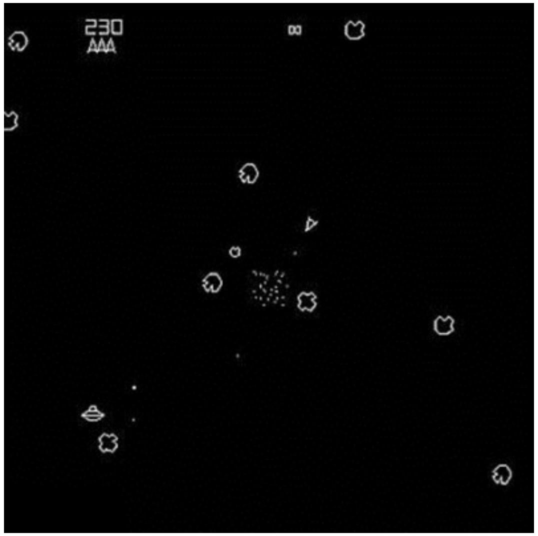
Physics History

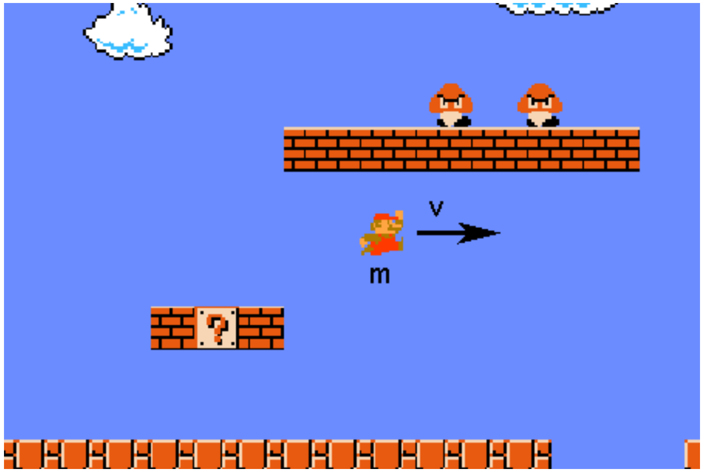
Special-Purpose Physics

- Like the games, built for one purpose/game
- E.g. Asteroids, Marble Madness, …

Built for enjoyment and good gameplay feeling

- Physical accuracy not important
- E.g. Mario’s momentum and friction





Physics History

TECHNISCHE UNIVERSITÄT DARMSTADT

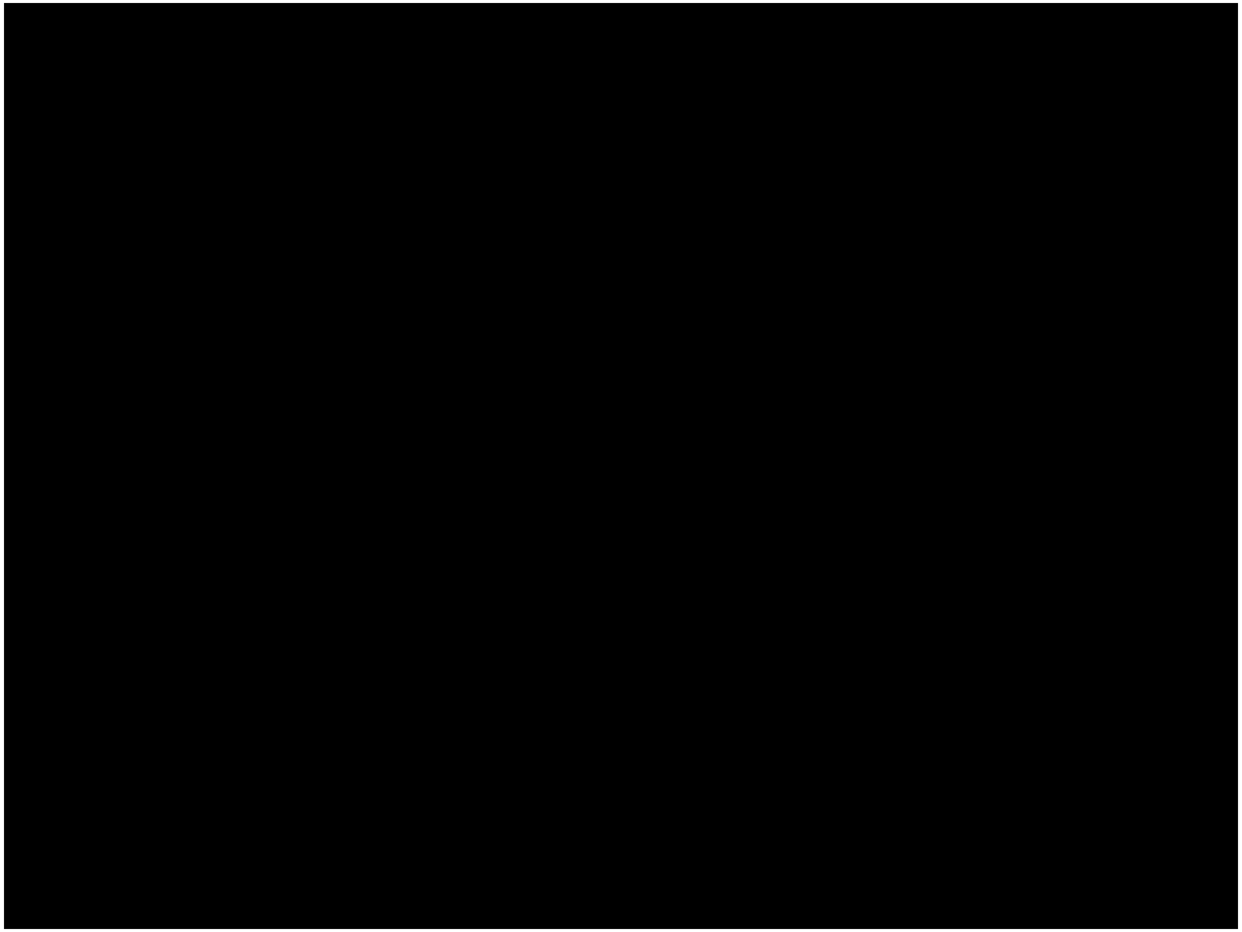
3D Physics

- Now more important to get realistic feel
- Started out with solutions developed in-house
- E.g. Trespasser (1998), own engine

Ragdoll Physics

- Physical Simulation for articulated bodies
- Previously only for unconcious characters
- Now mixed with forward kinematics

Trespasser (1998)



https://www.youtube.com/watch?v=i6cWEbkBeZQ

General-Purpose Physics

TECHNISCHE UNIVERSITÄT DARMSTADT

Libraries – Re-usable for different games

- Bullet
- Box2D
- …

Hardware Acceleration

- Nvidia Physx
  - Uses CUDA General-Purpose GPU Calculations
  - E.g. for particle systems
  - Source code available since 2015

Newtonian Physics

Isaac Newton
1643 - 1727

Newton’s three laws

I. Every object in a state of uniform motion tends to remain in that state of motion unless an external force is applied to it.

II. The relationship between an object's mass m, its acceleration a, and the applied force F is F = ma. Acceleration and force are vectors (as indicated by their symbols being displayed in slant bold font); in this law the direction of the force vector is the same as the direction of the acceleration vector.

III. For every action there is an equal and opposite reaction.

Law #1

Every object in a state of uniform motion tends to remain in that state of motion unless an external force is applied to it.

Examples of forces

- Gravity
- Drag
- Explosions
- …

→ **If we have an object that is just floating in space, simulation is very easy**

→ Just continue with the same velocity in the same direction

Law #2

The relationship between an object's mass m, its acceleration a, and the applied force F is F = ma. Acceleration and force are vectors (as indicated by their symbols being displayed in slant bold font); in this law the direction of the force vector is the same as the direction of the acceleration vector.

Mass m

- Measures the mass, not the weight
- The property that resists changes in linear or angular velocity

Acceleration a

- Measure of the change of velocity

Force F

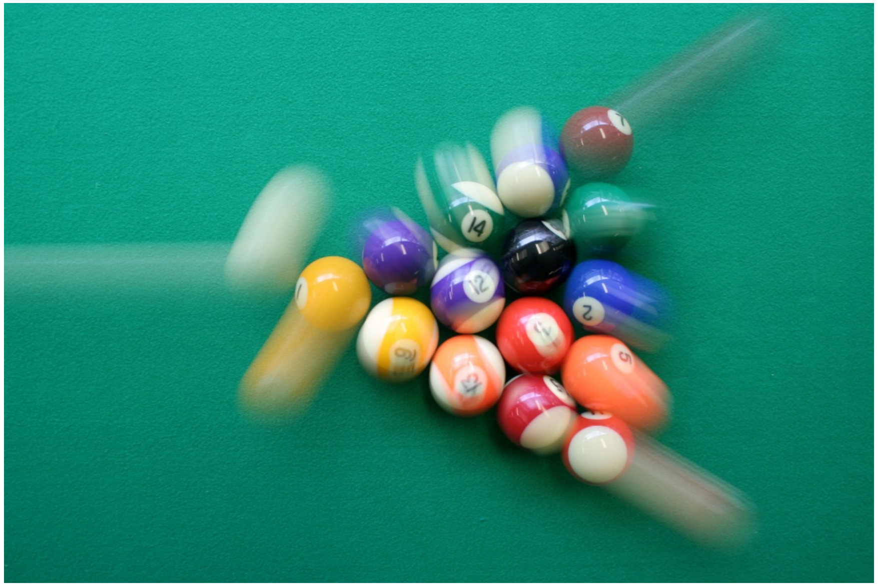
Law #3

For every action there is an equal and opposite reaction.

We need to take care of this when we are simulating collisions

- Collision Detection
- Collision Response

→ This is where the fun begins ;-)

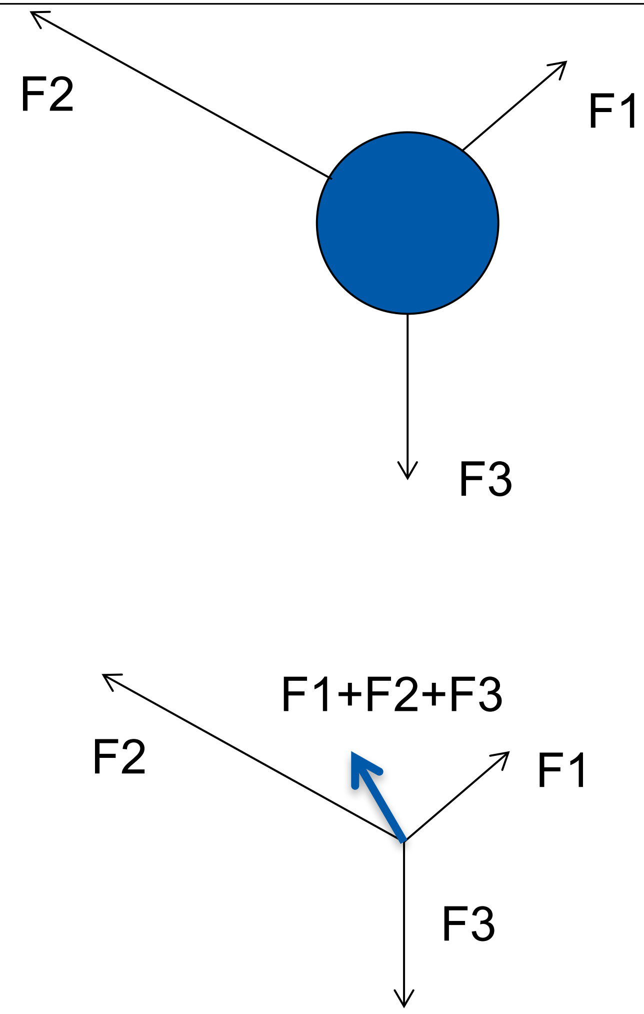


D‘Alambert‘s Prinicple

Forces being applied to an object add up (Vector sums)

Will save us computational time and make code more readable

- Calculating the effect of each force individually
  - Vs
- Accumulating forces and calculating the effect of the sum of the forces



Particle Systems

Particle

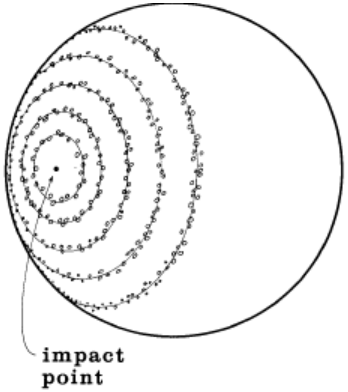
- Infinitisemally small object

→ No need to calculate rotations, forces off-center

→ No volume

Origins

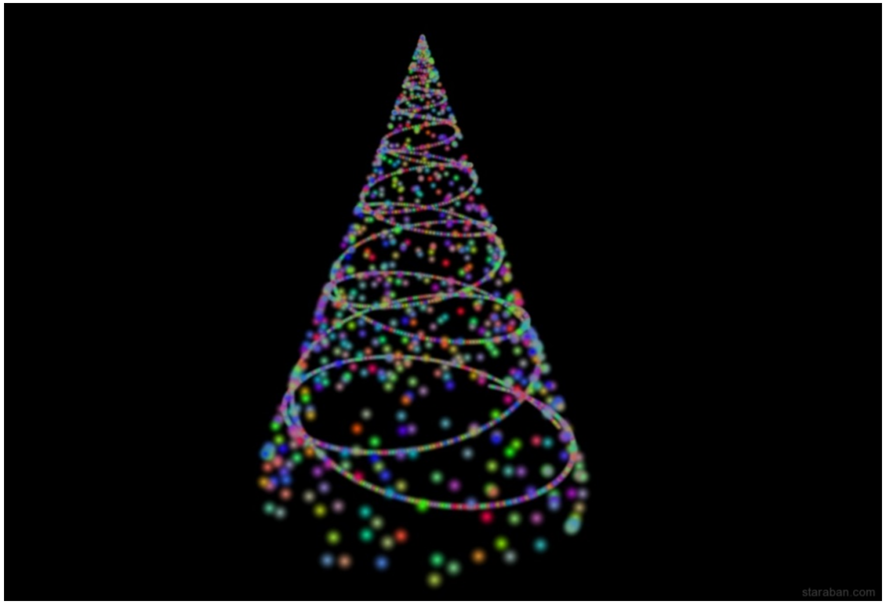
- William T. Reeves: „Particle Systems A Technique for Modeling a Class of Fuzzy Objects”, 1983
- Worked on “Star Trek 2 – The Wrath of Khan”



Particle Systems

Use in Games today

- Gaseous effects
  - Fire
  - Smoke
  - Gasses
- Explosions
- Atmospheric effects



Basis for advanced techniques

- Cloth simulation
- Hair simulation
- Fluid simulation

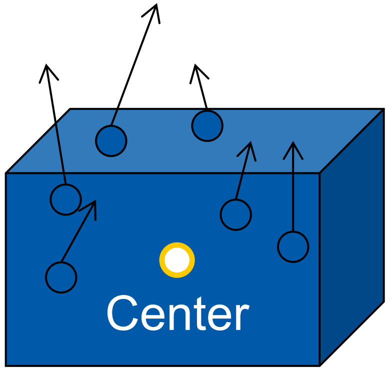
Particle Systems

Emitter

- Geometric shape in which the particles are spawned
- Spheres
- Boxes
- Complex polyhedra (meshes)
- Planes
- …

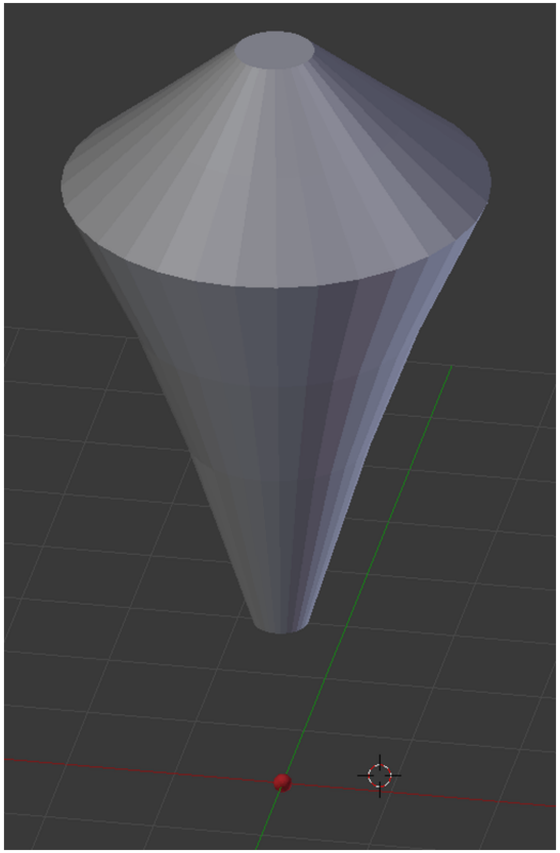
Emission Control

- Position (on faces, vertices, edges, inside the volume, …)
- Random positioning of the emitted particles
- Number of particles
- Initial velocity
- Other particle properties



Example – Particle systems shaping objects

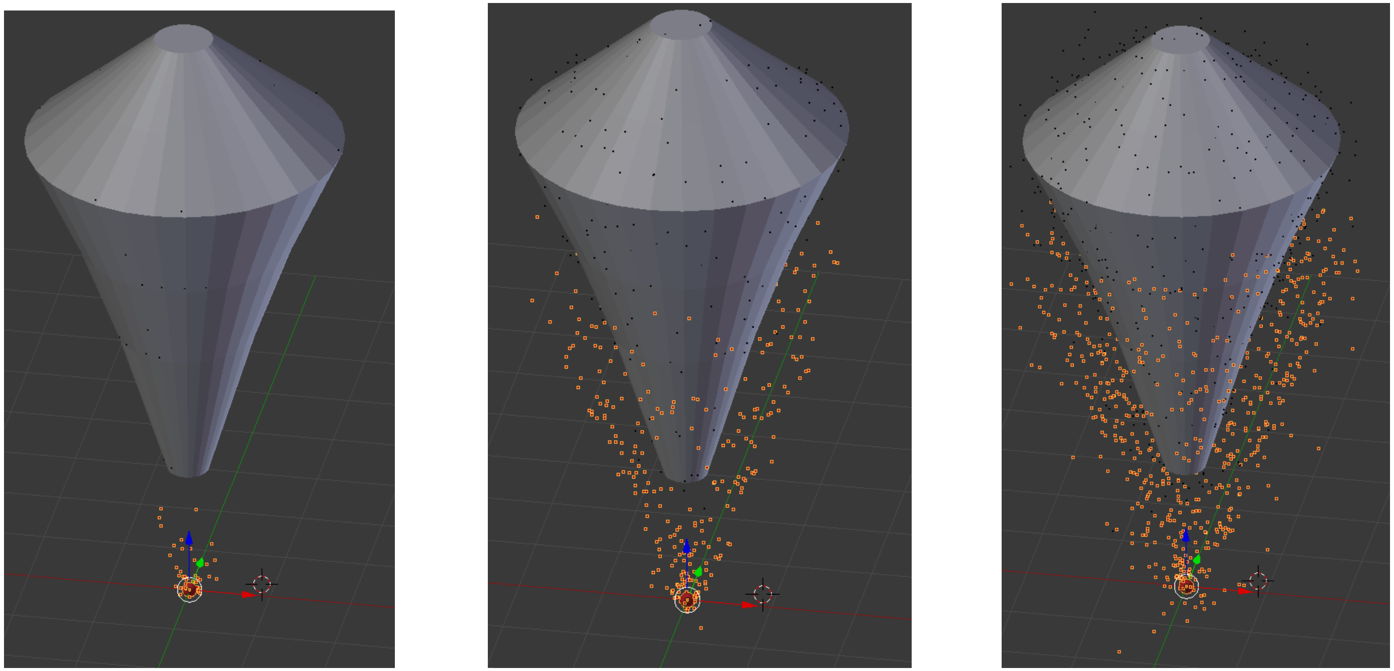
Goal: Render an amorphous/gaseous "alien"



Two particle systems

- One emits particles that are rendered, no gravity
- One emits invisible particles
  - From the shape of the mesh
  - No velocity, no gravity
  - Brownian motion
  - Act as attractors for the other particles

Example – Particle setup in Blender



Example – Rendering to 2D

Particle system control parameters

Initial position

- Jittering – amount of randomness

Spawn rate

- The rate itself
- Changes over time
- All at once, over a certain time, continuuously, maximal number of particles, …

Initial direction & velocity

- Direction (straight up, sidewards, …)
- Velocity

Gravity

Particle system control parameters

Other forces

- Wind
- Player interaction
- …

Time to live

2nd and further levels

- Spawn new particles at the end of the life cycle
- E.g. used for fireworks

Animation

- Control shape, size, transparency, sprite or any other parameter over time

Rendering Particles

Billboards

- Textures with (alpha) textures
- Simple geometry (can be instanced)

Rotating the particles to the camera

- Use the inverse of the view matrix
- View matrix is usually Translation and Rotation
- We only care about the rotation part
  - → Orthogonal matrix, can be inverted by transposing

Depth-Sorting the particles

- Use the transformed z-value of the particle
- Sometimes not necessary – can be a performance setting

Trails

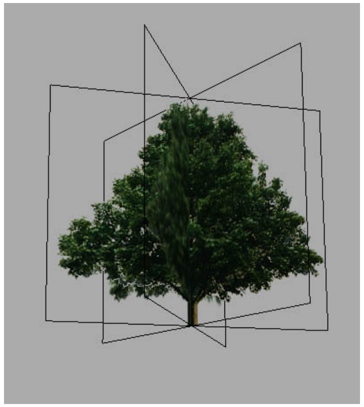
Types of Billboards

Quads

- Oriented towards the camera
  - In all directions (e.g. particles)
  - Only in one axis (e.g. vertical objects such as trees)

Several quads

- Placed around central axes
- E.g. for trees, bushes (vertical)
- Or beams (along the central axis)
- Not oriented towards the camera → one side always visible to a certain degree



Example: Fire

Gravity: Little to none (fire moves upward)

Lifetime: Such that the flames do not rise unrealistically high

Emission: Continuously

Texture: Simple texture with alpha (to get round look)

Tint

- Control parameter that can be animated over the lifetime of the particle
- Color value
  - Simple case: Color 1 at birth, Color 2 at death
  - More complicated cases: Provide intermediate key colors
- Supply to shader via a uniform
  - Write the tint-color as rgb and use alpha from the texture

Integration for particles

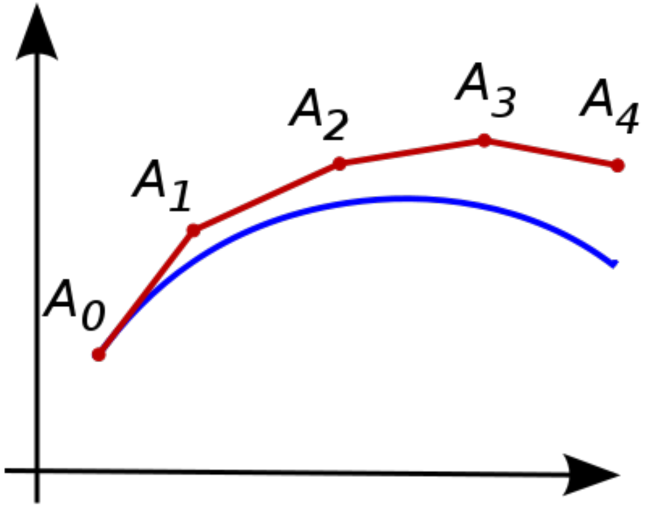
TECHNISCHE UNIVERSITÄT DARMSTADT

We need to simulate the effect of forces on the particles

Closed solution not tractable for real-time interaction and especially player interactions

Numerical integration

- Simplest approach: Euler integration



Apply Newton‘s second law

Newton‘s second law:

$$F = m \cdot a$$
$$a = \dot{v}$$
$$v = \dot{x}$$

$\boldsymbol{F}$: force
$\boldsymbol{m}$:mass
$\boldsymbol{a}$: acceleration
$\boldsymbol{v}$: velocity
$\boldsymbol{x}$: position

By transforming, this can be rephrased as a differential equation for the second derivative of the position, depending on the mass (assumed to be constant) and the force(s) acting on the object at time t.

$$\ddot{\boldsymbol{x}} = \frac{\boldsymbol{F}}{\boldsymbol{m}}$$

Solve the differential equation

Usually done numerically

$\boldsymbol{t}$: *Previous time*
$\boldsymbol{\Delta_t}$: *Timestep*
$\mathbf{t} + \boldsymbol{\Delta_t}$: *Current time*

Easiest algorithm: Euler method

First step: Velocity

$$\ddot{x} = \dot{v} = \frac{F}{m}$$

$$v_{t+\Delta_t} = v_t + \frac{F}{m}\Delta_t$$

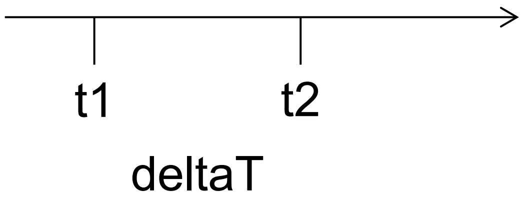
Second step: Position

$$x_{t+\Delta_t} = x_t + v_t\Delta_t$$

Time

With game‘s frame rate

- Update each frame
- Keep track of the last frame time
- Only an approximation of the next frame‘s duration
- Watch out for paused game (e.g. tabbed out of the window)



Independent

- Simulate independently of frame rate
- Sub-frame calculations → more exact
- Can adapt
  - If nothing happens, use large time step
  - Go to important moments (collisions)

Time Source

High Precision Event Timer (HPET)

- Found in chipsets starting in 2005
- 64 bit counter
- Counts up with a frequency of at least 10 MHz
- OS sets up an interrupt with a certain frequency

Getting the time

- Divide the counter value by the frequency
- Watch out for large values (e.g. PC in standby over weeks)

Rigid Bodies

Solid bodies that do not deform

Added properties

- Center of mass
- Rotation
- Angular velocity
- Angular acceleration
- Moment of inertia

Basic Terms

TECHNISCHE UNIVERSITÄT DARMSTADT

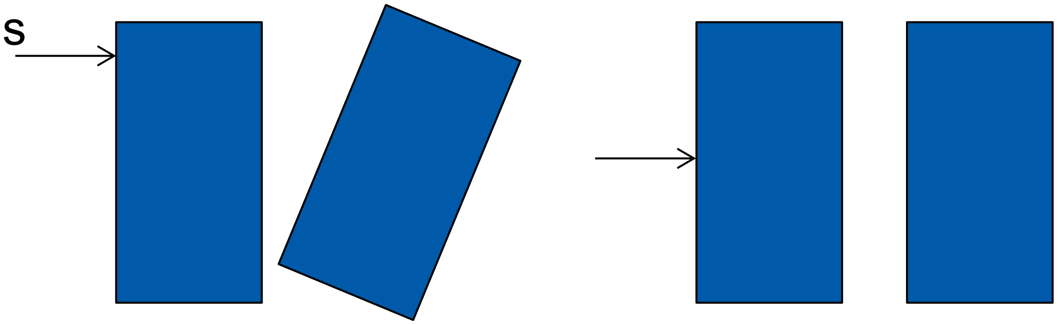
Mass: The property that resists changes in velocity

Center of mass

- Manually: Defined by artist
- Automatically: Assume uniform distribution
  - Integrate over the volume of the body

Force applied in line with the center of mass change only linear velocity

- Easiest way to handle collisions
- But not very realistic



Calculating the center of mass

Could sample the body at regular intervals

- x_i position of element i
- m_i mass of element i

$$C = \sum_{i=1}^{N} x_i m_i$$

Formula exists for polyhedra (assuming they are uniformly dense)

- V Volume, n_i normal of face i, a_i, b_i, c_i vertices

$$C = \frac{1}{2V} \sum_{i=1}^{N} \frac{1}{24} n_i([a_i+b_i]^2 + [b_i + c_i]^2 + [c_i + a_i]^2)$$

http://wwwf.imperial.ac.uk/~rn/centroid.pdf

Benefitial to save the object so that object space origin = center of mass

Moment of Inertia

Captures the way in which a body resists changes to angular velocity

Think of non-uniform objects

- Pushing at different points leads to different results

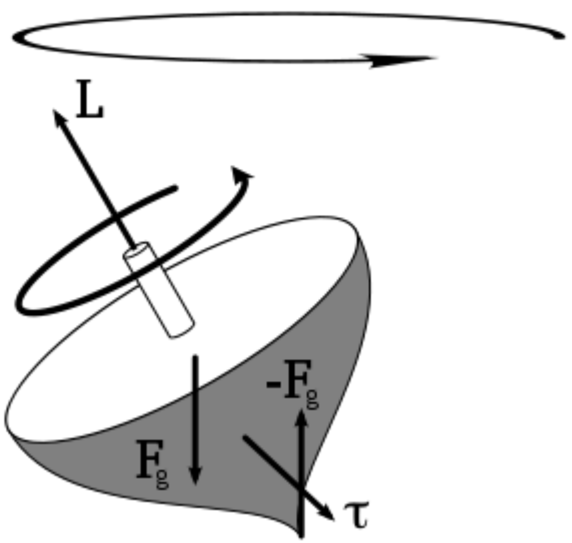
More in the next lecture

Torque

~ „Angular acceleration“

Forces that act off the center of balance

More info in next lecture



Collision Detection

TECHNISCHE UNIVERSITÄT DARMSTADT

Information we need to calculate a response

- Was there a collision?
- What was the collision normal?
- How far are the objects interpenetrating?

Intersection Sphere - Sphere

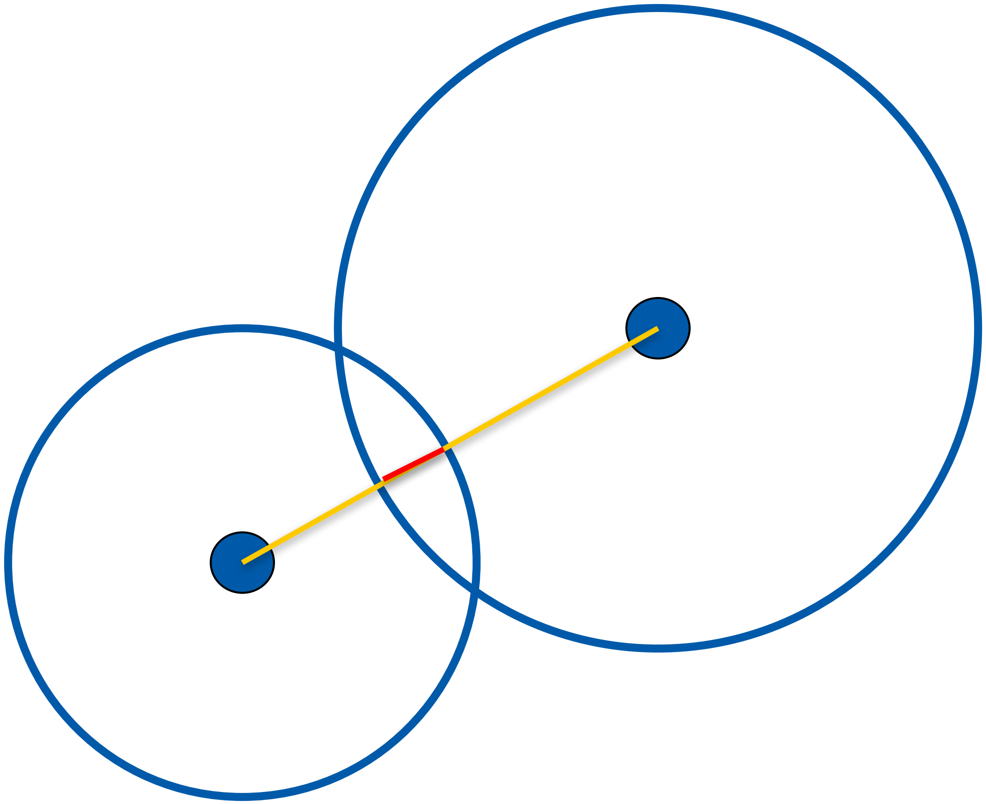
Easiest intersection

The spheres intersect if the distance of the centers is less than the sum of the radii

Collision normal can be found as the direction of one sphere's center to the other

Penetration depth is the difference between the sum of the radii and the distance of the center's positions from each other

Intersection Sphere - Sphere



Intersection Plane - Sphere

Describe the plane as

- Normal n
- Distance along this normal D from the origin

Then, for every point on the surface of the plane, the following equation holds:

$$\boldsymbol{x} \cdot \boldsymbol{n} = \boldsymbol{D}$$

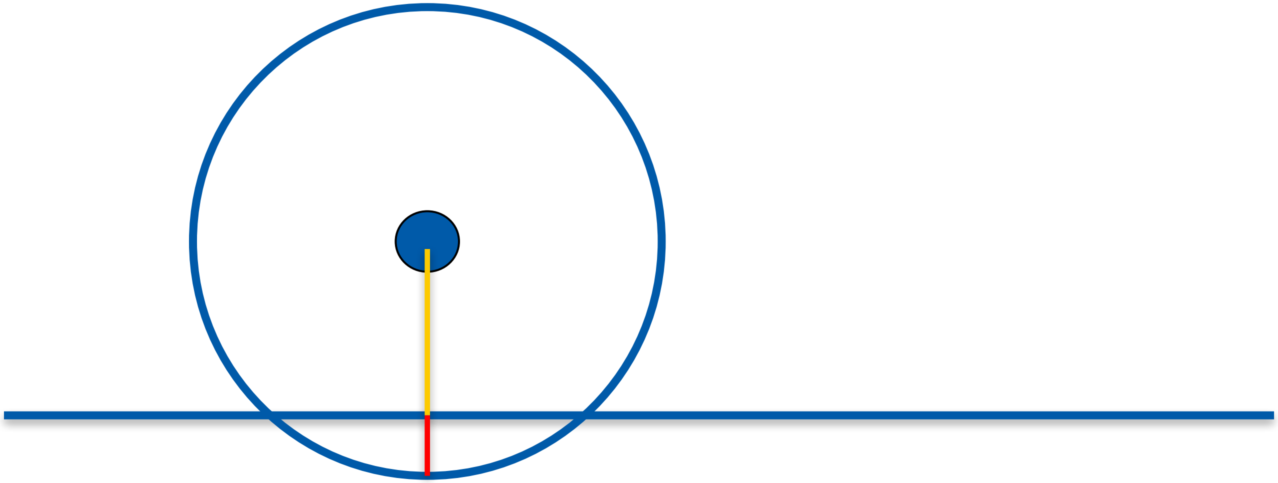
We can use the following formula to get a signed distance from the plane:

$$\boldsymbol{d} = \boldsymbol{x} \cdot \boldsymbol{n} - \boldsymbol{D}$$

Implicit formula

- Gives us a signed distance
- = 0 everywhere on the surface of the plane
- Distance to the plane everywhere else
- Sign indicates direction (with normal, in the opposite direction)

Intersection Plane - Sphere



Elastic Collisions

https://www.youtube.com/watch?v=sVTJNv3-mWk

Collision Response

Separate objects

- In reality: Elastic collision → energy is absorbed
- Approximate using coeffiction of restitution (COR)
  - Float between 0 and 1 → indicates the amount of speed retained after the collision
  - COR = 1 → No energy lost

Immovable Objects

- Infinite mass
- Save as inverse mass
  - Needed for calculations this way already
  - Infinite mass → Inverse mass = 0

Collision between two objects

Calculate the collision normal

- Direction along which the two objects are colliding
- Plane-Sphere: Use the plane’s normal
- Sphere-Sphere (for now): Use the vector from one sphere’s center to the other’s center

Calculate the separating velocity

- Velocity with which the objects are moving apart plus direction
- Sum of velocities projected onto the collision normal
- Careful with signs
- Velocity < 0: Colliding
- Velocity = 0: Resting/Sliding
- Velocity = > 0: Separating (Nothing to do, yay ☺)

Collision between two objects

Calculate a new separating velocity

- Using the coefficients of restitution and mass of the involved objects

Calculate an impulse that changes the velocity accordingly

- Instantaneous change in velocity
- In reality: Forces acting over very small times

Solve the interpenetration

- Move the objects so that they are not colliding any more
- Along the collision normal
- With the aspect ratio of the weights involved
- Immovable object (e.g. ground): Movable object has to move

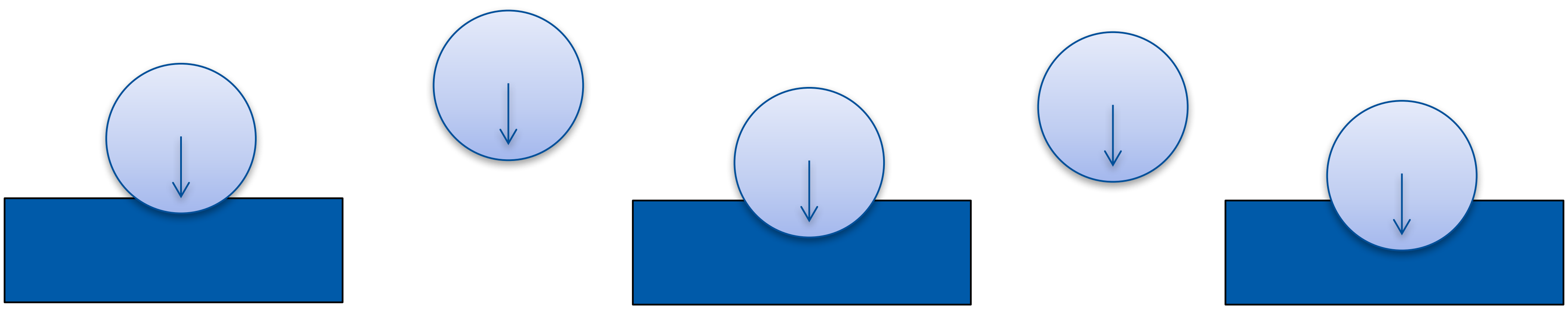
Apply the impulses

- = Adding to the current velocity

Problems – Interpenetration

Ignoring interpenetration

- Just calculate separating velocity
- Objects „hammer“ themselves into the ground
- On each collision, the object settles a bit more into the ground
- → Move the object out of the ground



Problems – Bouncing, Resting

Reality – Objects do not interpenetrate

- Deformation
- Energy shifted between the materials

Resting Contact

- Ground supports the resting object
- → Force that counters gravity

Ways to reduce/eliminate bouncing

- Add an additional impulse to counter the effect of gravity in the next frame
- Put objects to sleep when their energy goes low enough

Sleeping

In many games, most objects will be resting most of the time

- They only move when a script or a player action causes them to move

Identify when objects do not need to be simulated any more

- Start in a stable position (level design) and sleep initially
- Recognize that the energy is low enough

Wake up again

- Whenever the object takes part in the physical simulation
- Identify „Islands“
  - Groups of objects that should wake up together
  - E.g. the billard balls in the start configuration

Summary

- Particle Systems
  - Emitters
  - Billboarding
  - Control parameters

- Numerical integration
  - Euler integrator

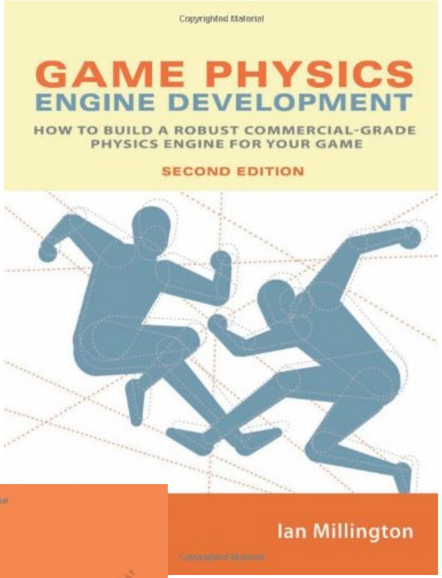
- Collision detection and Collision resolution
  - Collision between spheres
  - Collision sphere-plane

Literature

"Game Physics Engine Development", Ian Millington

"Real-Time Collision Detection", Christer Ericson

Box2D blog http://box2d.org/

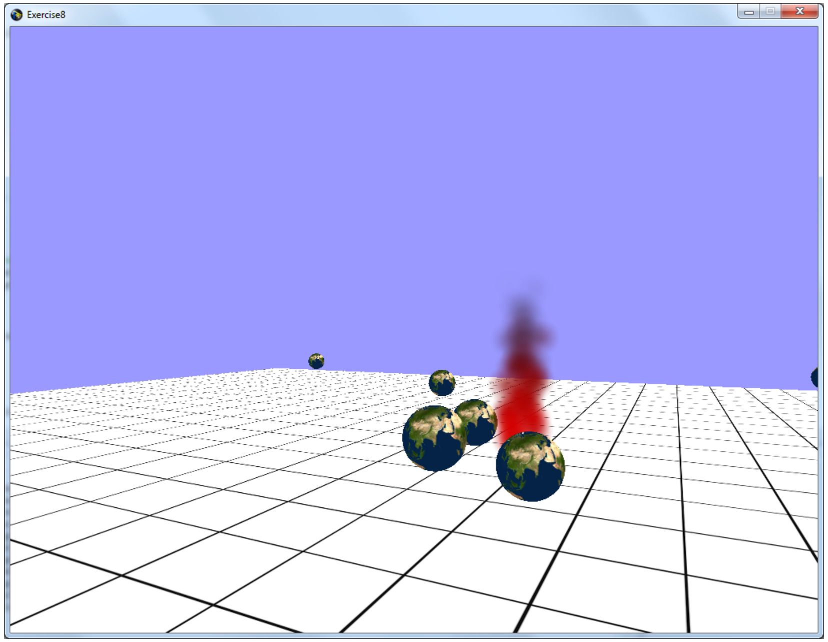


Exercise

Will be up after the lecture

Particle System

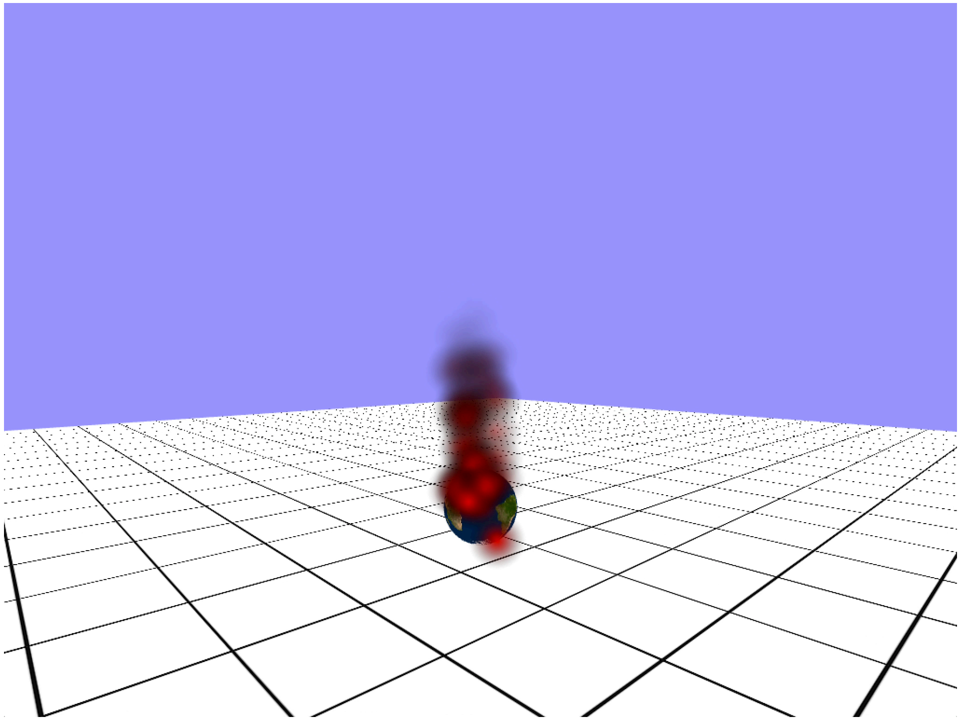
- Orient billboards to the camera
- Implement one new control parameter
  - Free choice of effect
    - Gas
    - Explosion
    - Rain
    - …
  - If you can't think of anything, use the fire example



Exercise

Physical Simulation

- Spheres are shot from the camera using Spacebar
- Very simple solution



Geometry Shaders

TECHNISCHE UNIVERSITÄT DARMSTADT

Input

- One primitive (point, line or triangle)

Output

- Fixed number of primitives

Use cases:

- One primitive (point, line or triangle)
  - Use instanced rendering instead
- Render to cube maps
- Transform feedback to multiple buffers

Instanced Rendering

TECHNISCHE UNIVERSITÄT DARMSTADT

Set multiple vertex buffers

- First one contains some geometry
- Second one contains for example transformation matrices
  - Plus a step rate

DrawInstanced(int instances)

- Draws the geometry *instances* times and increases the transform index by *step rate* each time

Domain/Control and Hull/Evaluation Shaders

Domain/Control

- Transform points
- Set inner and outer tessellation levels (kind of a separate step)

Hull/Evaluation

- Calculate final positions

Compute Shaders

Run on user-defined 1D/2D/3D arrays

- Bind input/output buffers
- Read/write data buffers, index by invocation id

Modern Graphics APIs

Vulkan

Direct3D 12

(Metal)

XXX

YYY

Pipeline States

Previously

- Set fragment shader
- Set depth mode
- Set blending mode
- …

Now

- At program start
  - Create pipeline
  - pipeline.fragmentShader = …
  - pipeline.depthMode = …
  - pipeline.compile()
- Laters
  - Set pipeline

Pipeline States

Previously

- Set fragment shader
- Set depth mode ← can trigger shader recompilation
- Set blending mode ← can trigger shader recompilation
- …

Now

- At program start
  - Create pipeline
  - pipeline.fragmentShader = …
  - pipeline.depthMode = …
  - pipeline.compile() ← shaders should always be compiled here
- Laters
  - Set pipeline

Command Lists

Previously

- setIndexBuffer
- setVertexBuffer
- drawWhatever

Now

- createCommandList
- commandList.setIndexBuffer
- commandList.setVertexBuffer
- commandList.drawWhatever
- commandList.close
- queue.submit(commandList)

Command Lists

Previously

- setIndexBuffer ← calls have to be converted to actual GPU commands
- setVertexBuffer ← you can do this from only one thread
- drawWhatever

Now

- createCommandList
- commandList.setIndexBuffer
- commandList.setVertexBuffer
- commandList.drawWhatever
- commandList.close ← convert commands here, do this on any thread you like
- queue.submit(commandList)

Buffers

Previously

- createVertexBuffer

Now

- allocateMemory(some fancy options)
- createVertexBuffer(memory)

Buffers

TECHNISCHE UNIVERSITÄT DARMSTADT

Previously

- createVertexBuffer

Now

- allocateMemory(some fancy options) ← cpu or gpu mem, cache options,…
- createVertexBuffer(memory)

Buffers

Previously

- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething
- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething

Now

- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething
- Wait for the GPU
- Hope for the best
- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething

Buffers

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Previously

- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething
- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething

Now

- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething
- Wait for the GPU ← This is very slow
- Hope for the best ← Easy to get wrong because cache coherency,…
- vertexBuffer.lock; memcpy; vertexBuffer.unlock;
- drawSomething