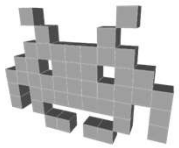
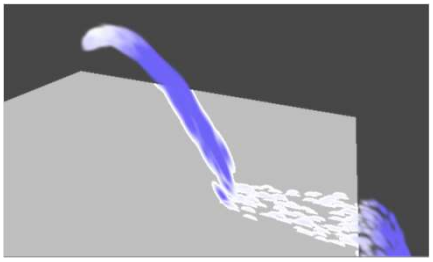
3D video games

Particle Systems

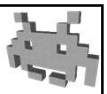


Marco Tarini



1

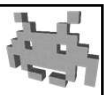
Course Plan



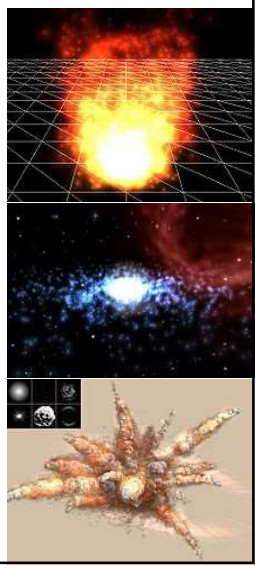
lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems** ←
lec. 6: Game **3D Models**
lec. 7: Game **Textures**
lec. 8: Game **3D Animations**
lec. 9: Game **3D Audio**
lec. 10: **Networking** for 3D Games
lec. 11: **Artificial Intelligence** for 3D Games
lec. 12: Game **3D Rendering Techniques**

2

Particle system (aka «particle effect», «particle FX»)

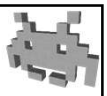


- Digital representations of 3D objects...
  - Not easily described by their surfaces
  - And/or: very dynamic (variable topology)
- ...such as:
  - clouds, dust clouds
  - flames, explosions
  - water sprays, waterfalls, spouts
  - rain, falling snow
  - wind (transporting dust / leaves / etc)
  - steam whiffle, walking dust-puffs
  - custom visual effects (e.g. for magic spells, etc)
  - swarms of flies
  - sparks, fireworks, electric sparks
  - gusts of smoke
  - *and so on*



3

Particle systems: just a bunch of particles

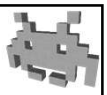


- one particle represents
  - a water drop, a flame spark, a rain drop, a smoke puff…
- state of a particle
  - Newtonian state: position, velocity
  - maybe also : orientation, angular velocity
  - lifespan («time (left) to live»)
  - custom variables: size, color , etc…
- Each particle is
  - dynamically emitted, aka “spawned” (from an «emitter»)
  - evolved (state changes)
  - and disposed (removed), after a brief line

} according to some predefined criteria

4

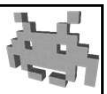
Particle systems: just a bunch of particles

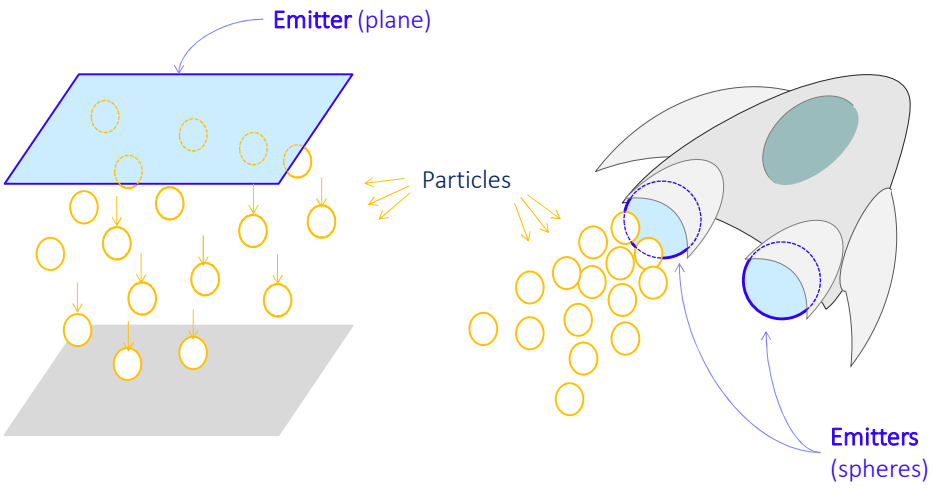


- Particles of a particle system are a simplified version of particles in a physics engine
  - with much simplified: dynamics, collision handling
  - individual particles are not important!
  - it's the collective behavior (e.g. 10^1 – 10^6 particles) that recreates the visual and the behavior of the recreated effect (flame, explosion)
  - the *entire* effect is often not that important either
    - cosmetics, not gameplay
- Note: particles systems are used in movies as well as videogames
  - We will discuss the videogame version

5

Emitters & Particles



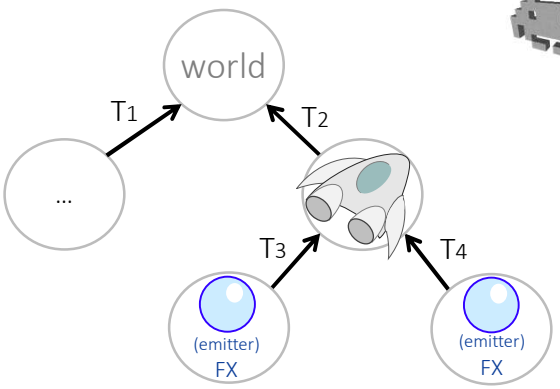


(example: falling snow)

(exampe: burst flame)

6

Emitters: in the scene graph!



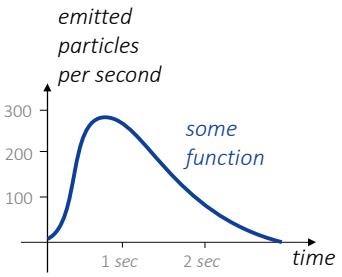
- Emitters reside in a scene graph node
  - as such: it is positioned/oriented in the scene
  - as such : it has some local/global transformation
  - as such : is has its own local & glob object space
  - to position/orientate the emitter means to position/orientate the particle system

The blaze, the explosion, the spray of water, etc …

7

Emitter: the producer of particles

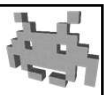
- emits particles according a designated criterion…
  - in pseudo-random way
    - with chosen probability distribution
  - at a designated *rate*
    - how many particles/sec
  - produces particle with an initial state
    - initial pos: randomly generated inside the emitter shape
    - initial vel, position, etc



- …for an established interval of time
  - e.g.: short (e.g. an explosion)
  - or medium (e.g. a blood gush from a wound)
  - or long (e.g. a column of smoke)
  - or undefined (e.g. water from tap, flame from torch…)

8

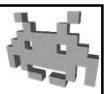
Emitter's «shape»



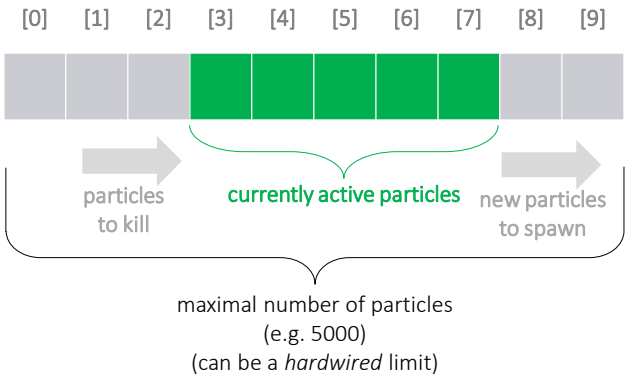
- Identifies the set of positions where new particles can be produced
- Just a 3D geometrical abstraction useful to guide particles creation
  - e.g. a sphere, cone, box, plane, point…
  - particle are created in a pseudo-random position inside this volume
  - Particle state:
    initialized with data expressed in world space
    or in object space (of the emitter)
    - e.g.: smoke: vel predominantly in Up dir. of *world* space
    - e.g.: rocket engine blaze: in Forward dir of *emitter* space

9

Internal data structure for a running particle system

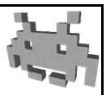


- A vector of particles
  - for each particle: its current status (position, velocity, time-to-live, …)
- "Circular" vector can be used



10

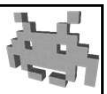
Internal data structure for a running particle system (pseudocode)



```
class Particle{
  vec3 pos;
  vec3 vel;
  float time_to_live; // how much longer?
  ...
}

class ParticleSystem{
  vector< Particle > particles; // circular vector

  // interval of active particles
  int first_active, last_active;

  function evolve( float dt );
  function render();
  function init();
}
```

11

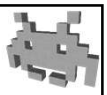
Particle system: GPU implementation?



- Running (i.e. playing) a particle system is extremely parallelizable
  - especially if the used dynamics is simplified
  - each particles “evolves” on its own
  - spawn a “new” particle? Just reinitialize an existing particle at the initial state
- GPU based implementations are relatively easy to do
  - GPU evolution
  - GPU rendering
  - particle data never leaves the GPU!

12

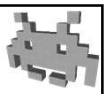
Particle system: randomness / noise



- The spawing and evolution of particles typically use a lot noise functions (pseudo randomness)
- Examples:
  - the initial position is randomly selected as any point inside the emitter
  - the initial color is selected as a random interpolation between two given color
  - the speed and accelerations have random comonents
- This creates differentiation and reflect the stochastic nature of the simulated phenomena
  - Flames, etc

13

Evolution of the particles: simplified dynamics



Note:
Can be computed in: emitter space, or world space, or interpolations

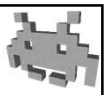
more procedural (in the sense of a simple procedure)

more physically-based (and expensive)

- Analytic evolution
  - state(t) ← f (t)
  - we can edit the trajectory of the particle f !
  - kinematic particles – no real dynamics
- Numeric evolution, no forces:
  - state($t + dt$) ← f (state(t) , dt)
  - e.g. with Verlet integration, or Euler…
  - but no forces: instead, vel is updated by a procedure.
  - e.g. puff of smoke accelerate upward, water droplets downward, air bubbles in water accelerate upward + random
- Numeric evolution, with forces:
  - give "mass" to particles
  - include (and cumulate) forces such as: cohesion between particles, repulsion between particles…

14

Evolution of the particles: simplified collision detection



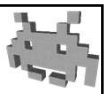
more procedural (in the sense of a simple procedure)

more physically-based (and expensive)

- No collisions!
  - e.g. smoke goes through walls (nobody cares)
  - easiest
- Collisions with only hardwired things
  - e.g. only with a plane, e.g. the ground
  - still very easy to parallelize
- Collisions with all static objects in the scene
  - can use spatial indexing structure.
  - only in necessary
- Collision with dynamic objects too
  - question to ask: is it really necessary?
- Collision with other particles too
  - luxury. Rare (in games)

15

Evolution of the particles: simplified collision response



more procedural (in the sense of a simple procedure)

more physically-based (and expensive)

Collision? Then…

- just kill particle
  - e.g. a spark hitting a wall just goes out
- stop particle: vel = 0
- ad hoc changes in state
  - e.g. a water droplet just stops on a surface for a while (looks wet) then disappears
  - e.g. a explosion particles just becomes a black stain, stays for a while, then disappears
- full impact computation, but one-way
  - elastic, static, or in between
  - particle is affected, object is not, even if dynamic
- full impact computation, and two-ways
  - impacted object is affected too (if dynamic)

16

Rendering a particle system: way 1 – render each particle

Each particle is individually rendered, as…

- one rendering primitive
  - a point ("point splatting") , a segment…
- or, one small 3D model
  - few (or one!) polygons, maybe textured
- or, one *impostor* , i.e.
  - a small quad centered at the particle
  - oriented towards the observer (usually)
  - with a texture (often, animated: frames)
    e.g. alpha maps + RGB maps
  - aka a "billboard"

most common case

Final look = superposition of all particles

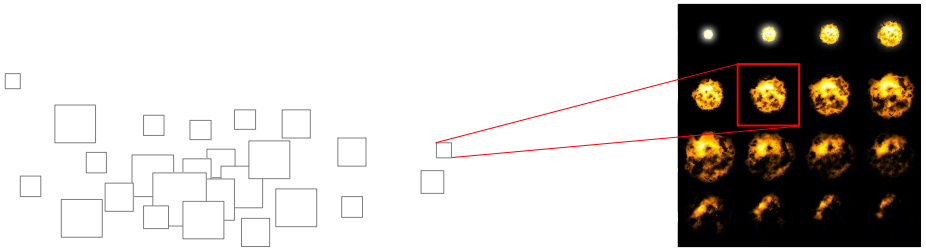
17

Rendering particles individually

- The aspect of individual particles is controllable in many ways
  - Size of impostor?
  - Color of the slat?
  - Transparency level the impostor?
  - Screen-space rotation of the impostor?
  - If animated impostor: frame to use?
  - etc…
- They can be parameters:
  - of time-to-live
    - e.g. for flame at start: red color; mid-life: yellow color; end: black color
    - E.g. for smoke:
      at beginning small and dense particles; at end: large and transparent
  - of speed
  - or others

18

Rendering particles as impostors
2D images (textures)

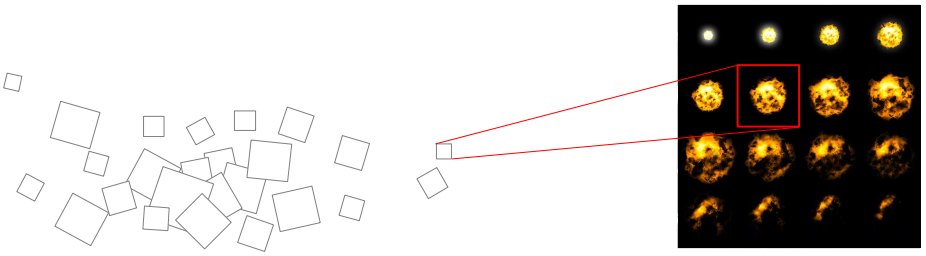


The image can change during time
(animation, sequence of frames)

The image is partially transparent or semitransparent
(it has “alpha” channel)

19

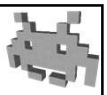
Rendering particles as impostors
2D images (textures)



can also be rotated in view space
(or, in 3D)

20

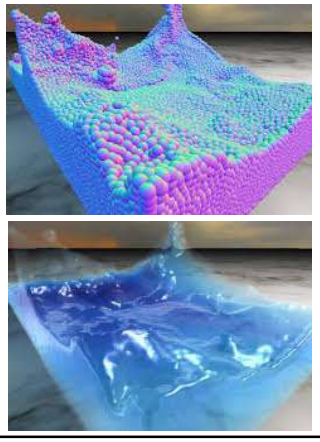
Rendering a particle system: way 2 – fuse particles in one 3D shape



- Usually too time consuming, for a game
- Can be approximated with screen-space techniques

see lecture on Rendering later

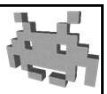
  - pass 1: splat a temporary "blob" for each particle in a offscreen buffer
  - pass 2: estimation of normals of "blobs" union in screen space
  - pass 3: rendering of the resulting surface
- Ideal for liquids!



this example by Simon Green (NVIDIA)

22

Authoring a particle system

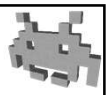


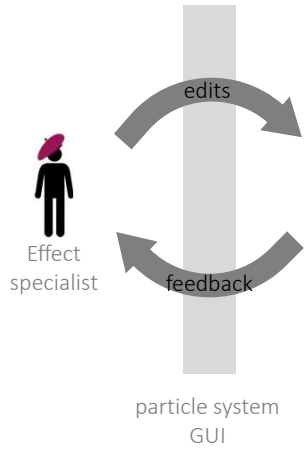
- Particle system = just another asset
- Authoring it = the task of the *Effects specialist*
  - Designing the behavior
    - Choose the emitter
    - specify how particles are created & evolved
    - How? by programming scripts for the task
    - and/or by specifying predefined set of parameters in a GUI (part of a given particle systems creation suite)
  - Designing the look
    - which image (texture) for impostor
    - which tiny 3D models ?
    - which splat parameters, etc.

digital artist

23

Authoring a particle system via a GUI





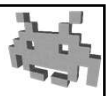
Particle System asset

- Spawning parameters
  - emitter shape
  - emission ratio (over time) →
  - initial state of particle (e.g. velocity)
  - initial time to live
- Evolution parameters
  - particle trajectory
  - changes in vel
  - forces, etc
- Rendering parameters
  - Rendering strategy
  - Colors
  - Sprites / textures →
  - Used 3D model, etc.

All that as a function of time,
or as a distribution random variables…

24

Many particle system framework / software exists



Example of specialized tools

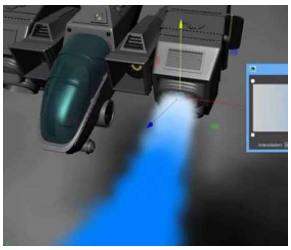
- Houdini (widely used for movies)
- Cascade (in Unreal)
- Particle Flows (in 3D studio Max)
- X-Particles (for Cinema4D)
- thinkingParticles (plug-in for different software)
- *…and many others*

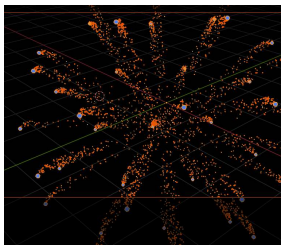
Many systems provide their own built-in editors

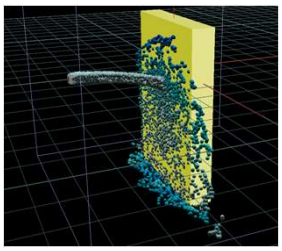
- Unity (“shuriken”) wysiwyg slider-based editor
- Blender
- Maya (“nParticles”)
- *…and many others*

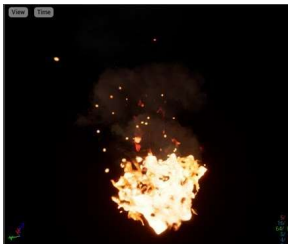
25

Particle systems in…

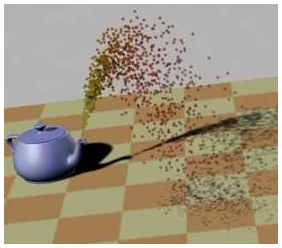

nParticles (Maya)


Blender


Houdini


Cascade (Unreal)

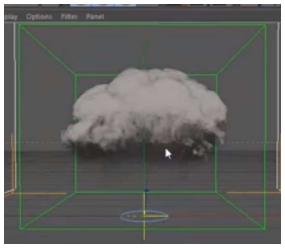

Shuriken (Unity)


RenderMan 20

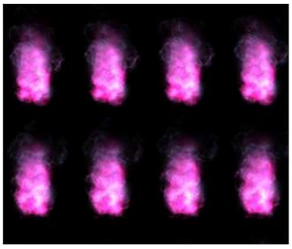
26

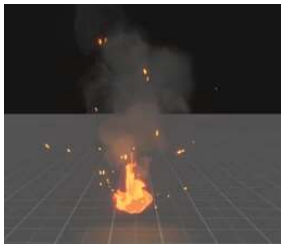
Particle systems in…


Particle Flow (3D max)


X-Particles (Cinema4D)


Thinking Particles

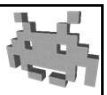

TimeLineFx (RigzSoft)


PocCornFX

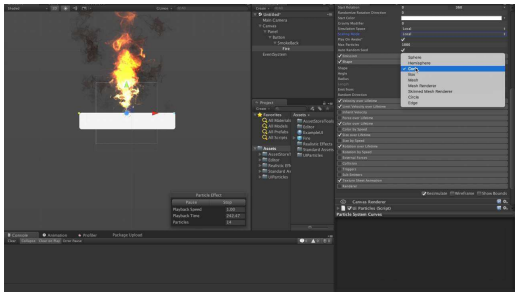

Particle Illusions (Boris FX)

27

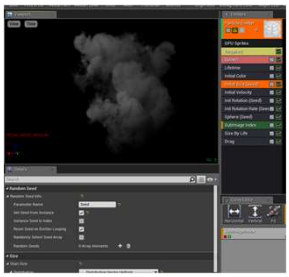
Just two notable examples



- Unity built-in editor for “shuriken” particle systems

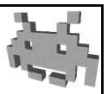


- Unreal built-in editor for “cascade” particle system



28

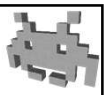
Lack of established formats for particle system assets



- Each software suit uses its own:
  - set of parameters, tricks, degrees of customizability
  - interface to let a FX specialist author the particle system
- ...and file formats to store the particle system asset. Examples:
  - Unity: saved as .prefabs
  - Unreal: “cascade” file format
  - Maya: .pdb .pda
  - Renderman: .ptc
  - Houdini: .geo .bgeo

29

Lack of established formats for particle system assets



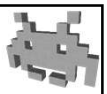
- Problems:
  - hard to run a particle system in a game engine unless that particle system was authored in that engine/system
  - hard to reuse or off-source particle systems across different systems / engines
- To solve this, a few "Esperanto" format have been proposed for particle systems:
  - Not very much established yet

Partio (by Disney)

ALEMBIC (by Sony)

30

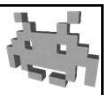
Particle systems: cosmetics or gameplay?



- Usually, only graphic coating
  - increases visual realism
  - communicates what's going on to the player (e.g. splashes = you are walking on water. metal sparkles = you have been it)
  - no effect of individual particles on gameplay
- This justifies many approximations:
- Remarkable exceptions exist
  - of particles used in gameplay

31

Digression: particle systems outside videogames

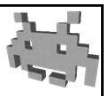


- Particle effects are used in movies too
  - the techniques are the same
  - naturally, there is less need for simplification
  - because dynamics + collisions + rendering are off-line not real time
  - A few of the tools listed above are specialized for this scenario

- Additional use of particle systems in movies: fur / hair / grass.
  - Imagine the trajectory of each particle as shape of an individual hair instead of the position as a function of time

32

Practical (and fun) exercises



- Improvise as a FX specialist
  - use any of the above software (e.g. unity or unreal)
  - use its interface to create a particle system to simulate … something (an explosion)
  - you can for example follow a tutorial
- Observe some existing particle effect
  - download them from repository / asset stores
  - see how they are implemented
- Reminder: this course is does not cover about digital artist skills, but experimenting helps understanding better what's behind the scenes

33