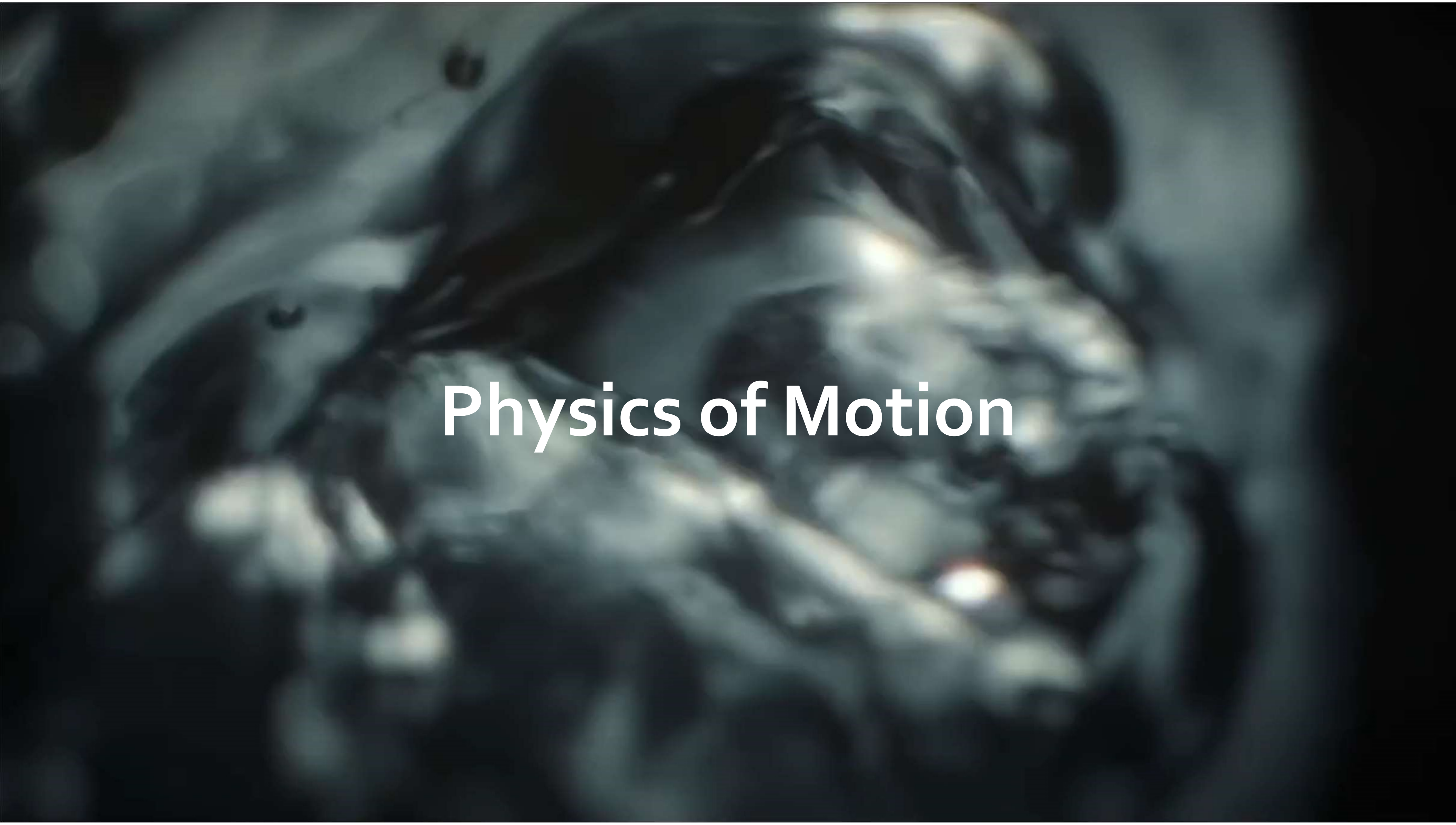
Physics of Motion



Location vs. Velocity vs. Acceleration

- Velocity = the rate of change of location $\left(\mathbf{v} = \frac{d\boldsymbol{r}}{dt}\right)$
- Acceleration = the rate of change of velocity $\left(\mathbf{a} = \frac{d\mathbf{v}}{dt}\right)$
- Acceleration affects velocity $\left(\mathbf{v} = \int \mathbf{a}dt\right)$
- Velocity affects location $\left(\boldsymbol{r} = \int \mathbf{v}dt\right)$
- Acceleration affects location $\left(\boldsymbol{r} = \iint \mathbf{a}dt\right)$
- If we find $\mathbf{a}$ we can calculate $\mathbf{v}$ and $\boldsymbol{r}$.

- $\rightarrow$ Only think about acceleration changing algorithms

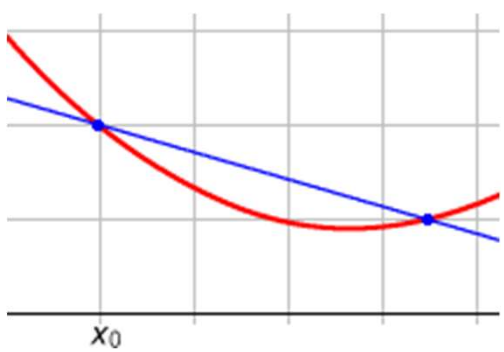
Program Structure

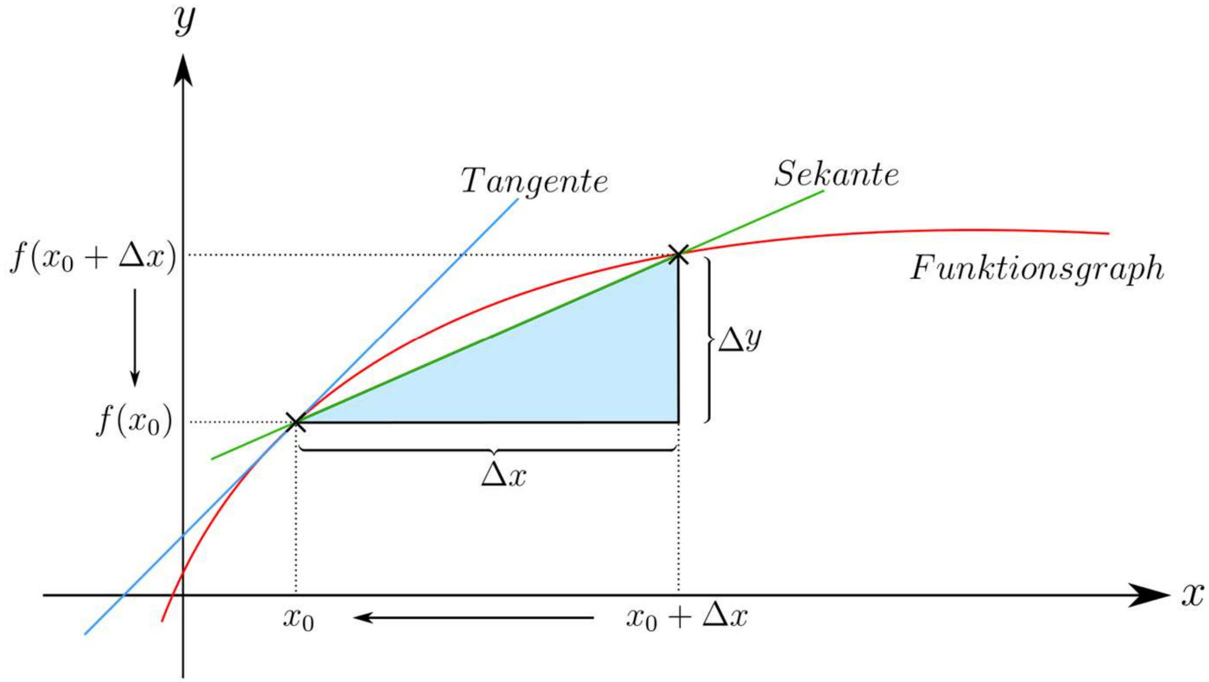
```
while (not finished)  // each frame once
{
  update(); // move everything a little bit
  render(); // draw everything
}
```

- Differentials and integrals $\left(\mathbf{v} = \frac{d\boldsymbol{r}}{dt}, \boldsymbol{r} = \iint \mathbf{a}\, dt\right)$ are continuous representations
- Code evaluates movement in discrete steps
- Discrete representations?

Definition of Differential

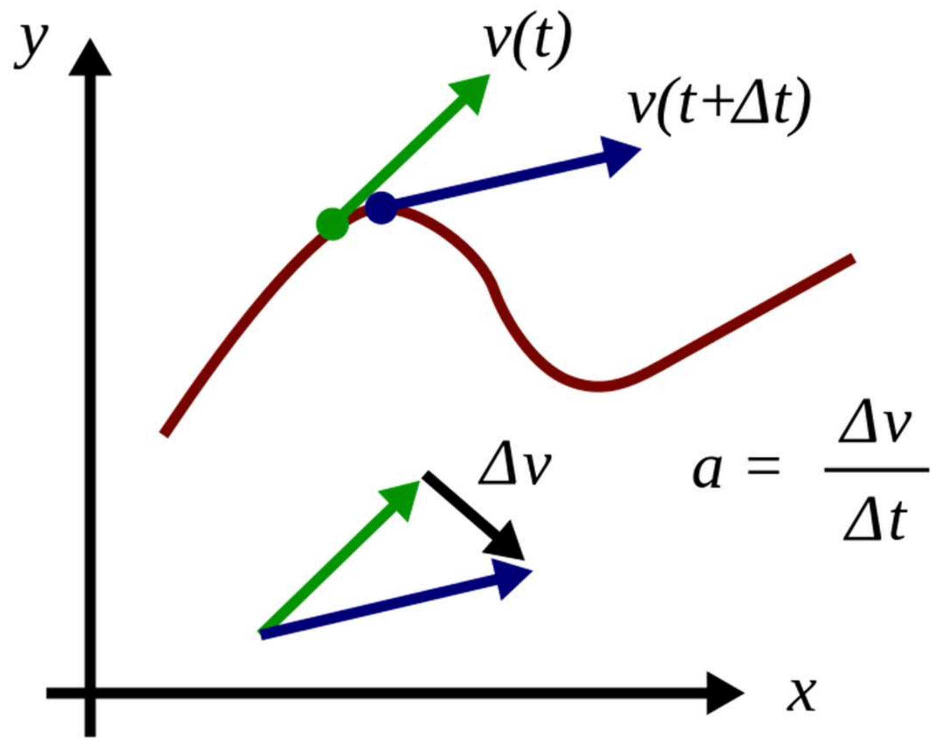
- $\frac{df}{dx} := \lim_{\Delta x \to 0} \frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$
- Tangent is approximated by secant $\frac{df}{dx} \approx \frac{\Delta f}{\Delta x}$ for small Δx





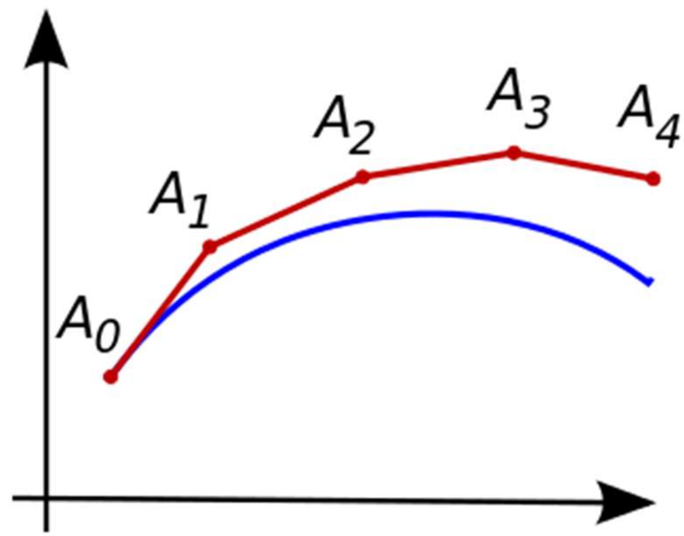
Location vs. Velocity vs. Acceleration (Discrete)

- $\mathbf{v}_{\Delta t} = \frac{\Delta \boldsymbol{r}}{\Delta t}$ $\mathbf{a}_{\Delta t} = \frac{\Delta \mathbf{v}}{\Delta t}$ will drop Δt subscript for visual simplicity
- How to integrate?



Forward Euler Method

- Numerical integration (solves first order differential equations)
- Idea: incrementally take small steps along tangent
- $f(t_0) = A_0$
- $f(t_{n+\Delta t}) \approx f(t_n) + \Delta t\, f'(t_n)$ for small Δt
- First order method – error proportional Δt
- Better integrators exist
  - More computationally expensive
  - Need to store/evaluate more data points



Location vs. Velocity vs. Acceleration (Discrete)

- $\mathbf{v} = \frac{\Delta \boldsymbol{r}}{\Delta t} \qquad \mathbf{a} = \frac{\Delta \mathbf{v}}{\Delta t}$
- Euler integration in two steps
- Acceleration affects velocity ($\mathbf{v} = \mathbf{v_0} + \mathbf{a}\,\Delta t$)
- Velocity affects location ($\boldsymbol{r} = \boldsymbol{r_0} + \boldsymbol{v}\,\Delta t$)

Program Structure

$\mathbf{a}cceleration = initialAcceleration$;
$\mathbf{v}elocity = initialVelocity$;
$\mathbf{p}osition = initialPosition$;

while (not finished) *// each frame once*

{

$\mathbf{a}cceleration = magicFunction(\mathbf{a}cceleration, \Delta t)$;

$\mathbf{v}elocity \mathrel{+}= \mathbf{a}cceleration * \Delta t$;

$\mathbf{p}osition \mathrel{+}= \mathbf{v}elocity * \Delta t$; *// move everything a little bit*

render(); *// draw everything*

}

Newtonian Physics – Laws of Motion

1. A body will remain at rest or continue to move in a straight line at a constant velocity unless acted upon by another force.
   - $\sum \mathbf{F} = 0 \Leftrightarrow \mathbf{0} = \frac{\Delta \mathbf{v}}{\Delta t}$ (So, Atari Breakout had realistic physics! ☺)
2. The acceleration of a body is proportional to the resultant force acting on the body and is in the same direction as the resultant force.

3. For every action, there is an equal and opposite reaction.

- More recent physics show laws break down when trying to describe universe (Einstein), but still good enough for computer games

Newtons 2nd Law of motion

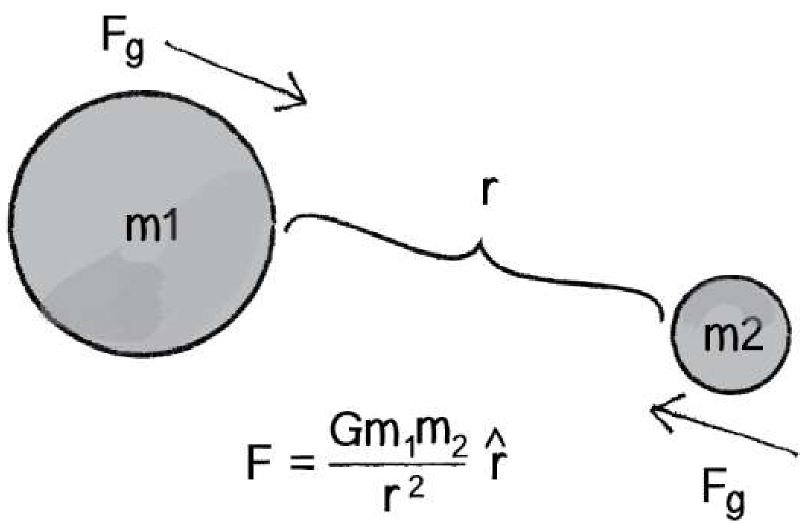
- $\mathbf{F} = m\mathbf{a}$
- Force = mass * acceleration … for a given body
- Force and accelerations are vectors
- We want acceleration
  - $\mathbf{a} = \frac{\mathbf{F}}{m} \ldots$ acceleration is indirect proportional to force applied to body
- More than one force?
  - $\mathbf{a} \mathrel{+}= \frac{\mathbf{F}_i}{m}$ … force accumulation

What is $\boldsymbol{F}$?

- Usual forces that hang around (with direction and magnitude)
  - Gravity
  - Friction
  - Resistance
  - Wind
  - Drag
  - Magnetism
  - …
  - Invent your force
    - Jedi
    - …

Force Example – Gravitational Attraction

- G …*universal gravitational constant* (6.67428×10^{-11})
- Direction?
  - Towards each other



Particle Systems

Particle Systems

- Break up complex phenomena into component parts (particles)
- For fuzzily defined phenomena
- Highly complex motion
- No animation of each part by hand
- Provide overall rules for animation
- Dust, sparks, fireworks, leaves, flocks, water spray, fluids (water, mud, ...), fire, explosions, hair, fur, grass, clothing, ...

What is a Particle?

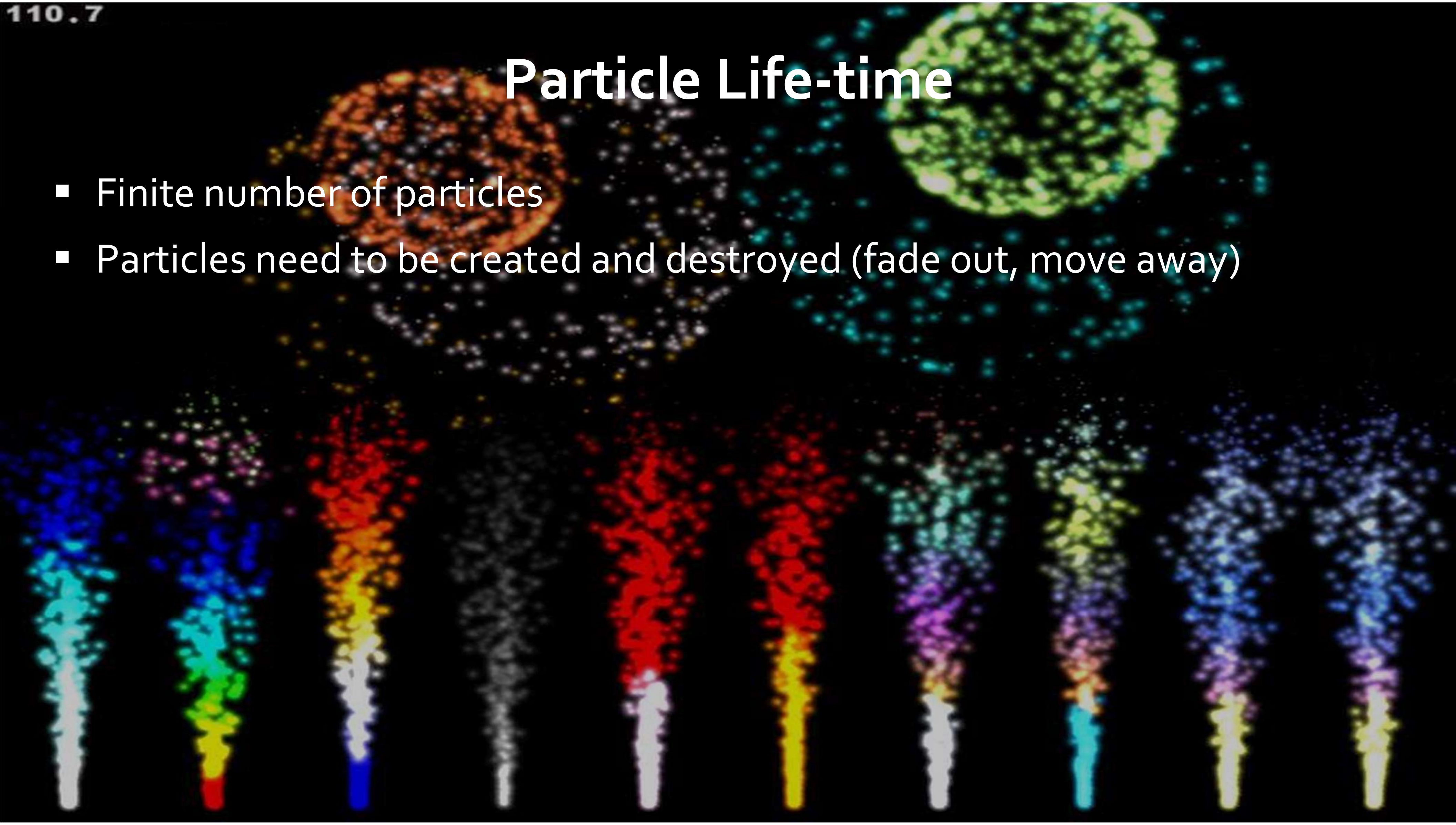
- Particles have a position
- Usually add other attributes you need for simulation and/or rendering
  - Velocity, mass, acceleration, age, color, radius, orientation, temperature, type, texture, …

Particle Systems – Considerations

- When and where particles start/end?
- Rules that govern motion (attached variables, e.g. color)
- How to render the particles?

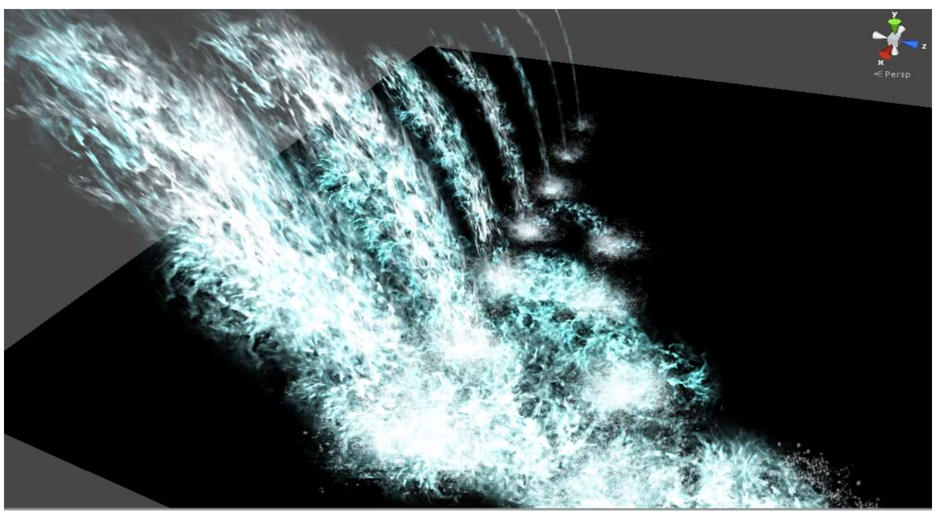
Particle Life-time

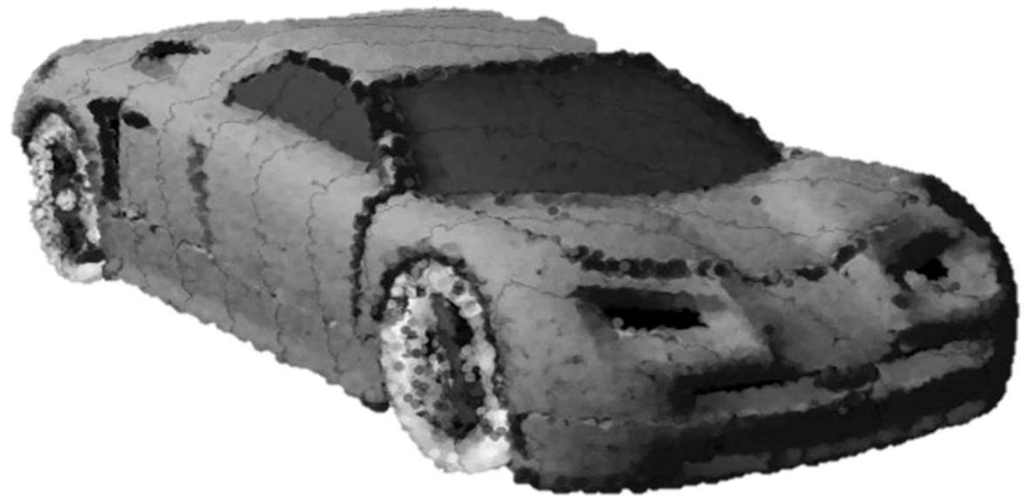
- Finite number of particles
- Particles need to be created and destroyed (fade out, move away)



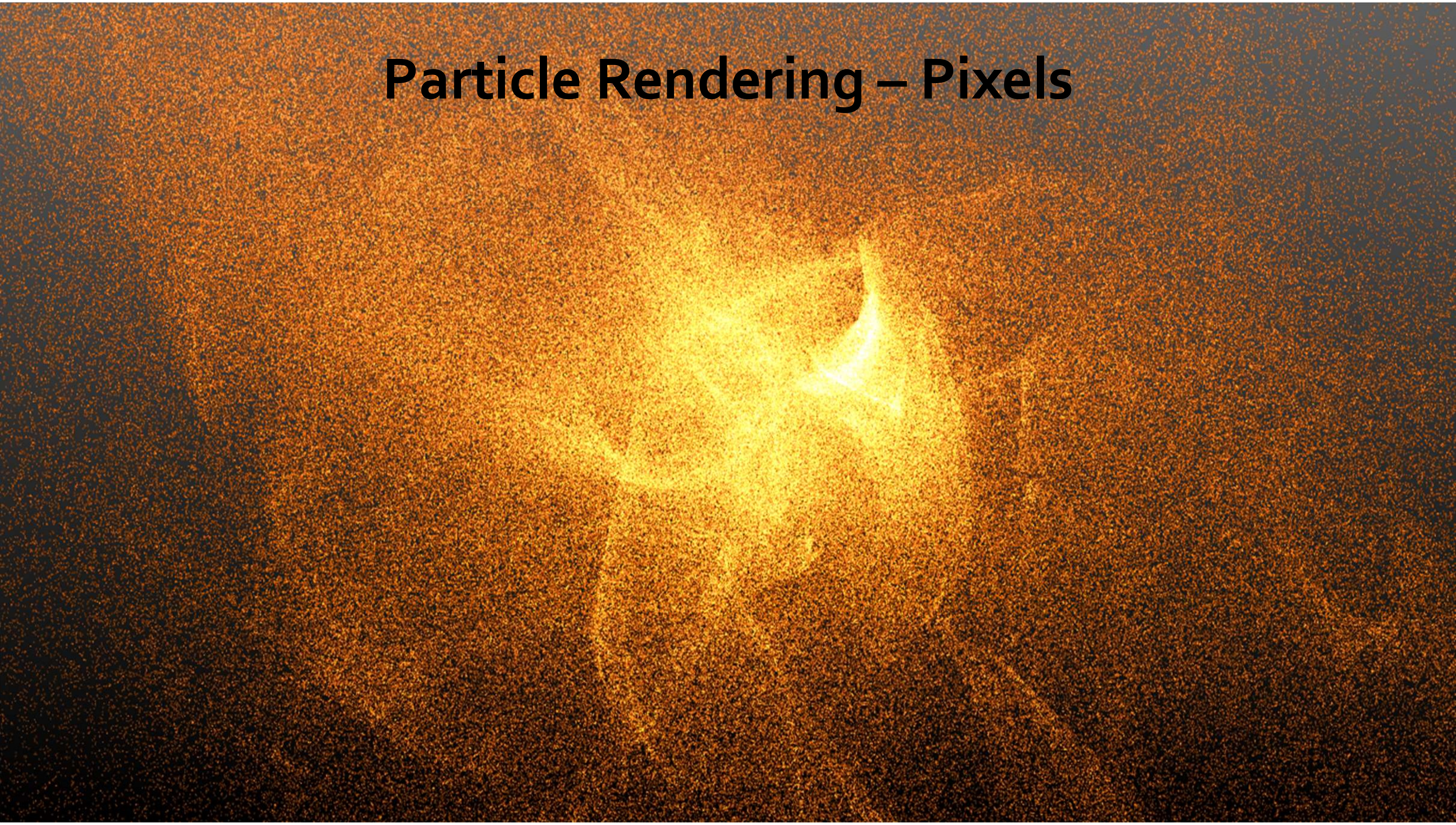
Particle Seeding

- Randomly within a shaped volume or on a surface
- At a source (waterfall, ...)
- Where there aren't many particles currently
- Several per frame



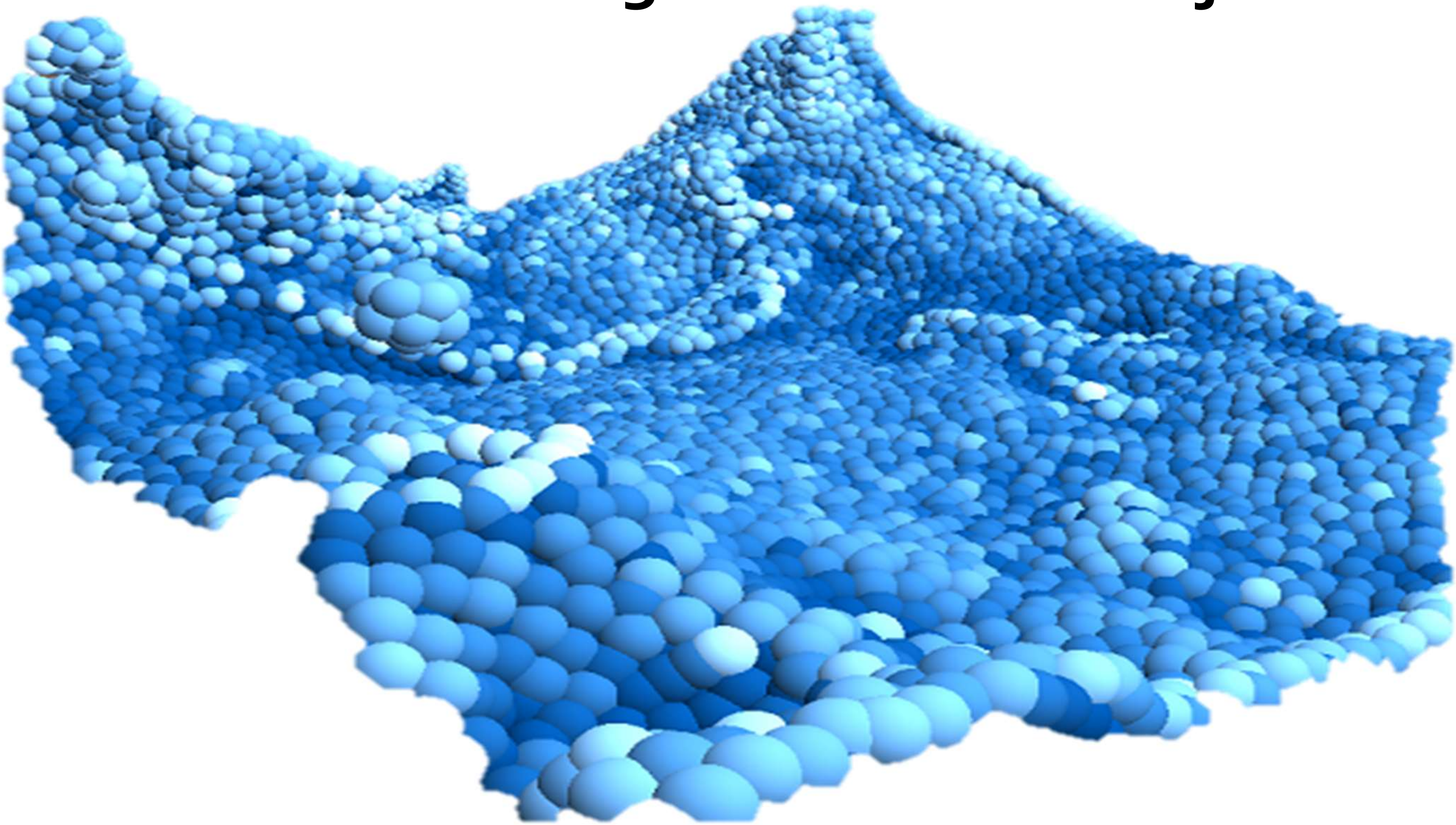


Particle Rendering – Pixels

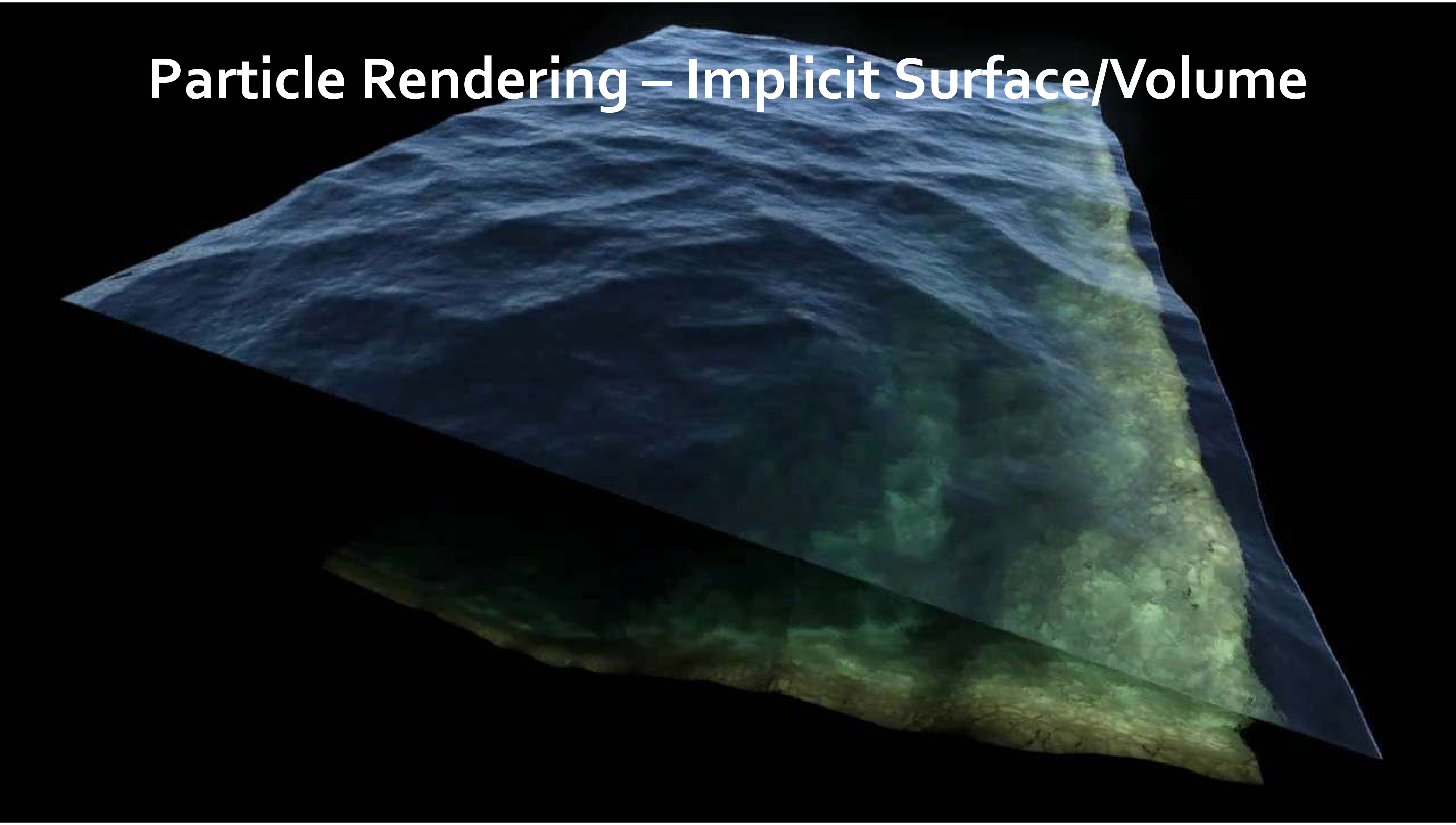


Particle Rendering – Point Sprites

Particle Rendering – Geometrical Objects



Particle Rendering – Implicit Surface/Volume



Particle Rendering – Overlap

- No special handling (zBuffer)
  - Works only for opaque particles
  - Hard borders
- Blending add
  - Works without depth sorting
  - Models each point emitting (but not absorbing) light
  - Not real lighting
  - Works for sparks, fire, …
- Compute depth order, do alpha blending, worry about lighting, shadow…