

# Procedural Animation

*Particles Trajectories*

# Particles Systems History

One of the first animated model in CG



*[ Particle systems - A Technique for Modeling a Class of Fuzzy Objects, William T. Reeves, Lucasfilm, 1983 (Star Trek II) ]*



*[ Karl Sims, Particle dreams, 1988 ]*

# Example of particle system

## Free fall of spheres under gravity

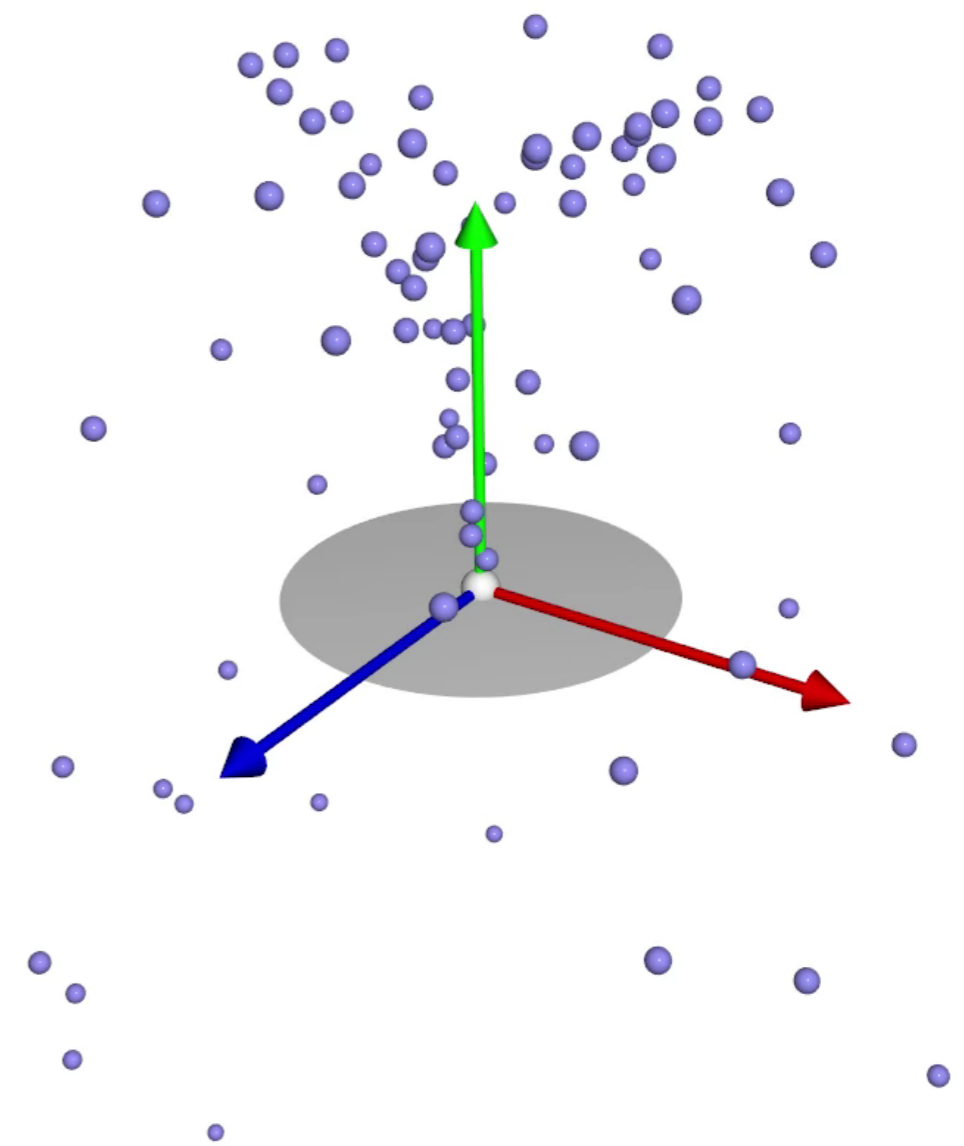
- Geometrical representation of each particle: sphere
- Equation of motion  $p(t) = \frac{1}{2}gt^2 + v_0t + p_0$
- Initial position and speed may be placed at random position
- Each particle may have a different life time

Q. What are the parameters used for  $p_0$  and  $v_0$  in this example?

$p_0 = \dots$

$v_0 = \dots$

Note: Coordinate system  $(x, y, z) = (\text{red}, \text{green}, \text{blue})$ .



# Bouncing spheres

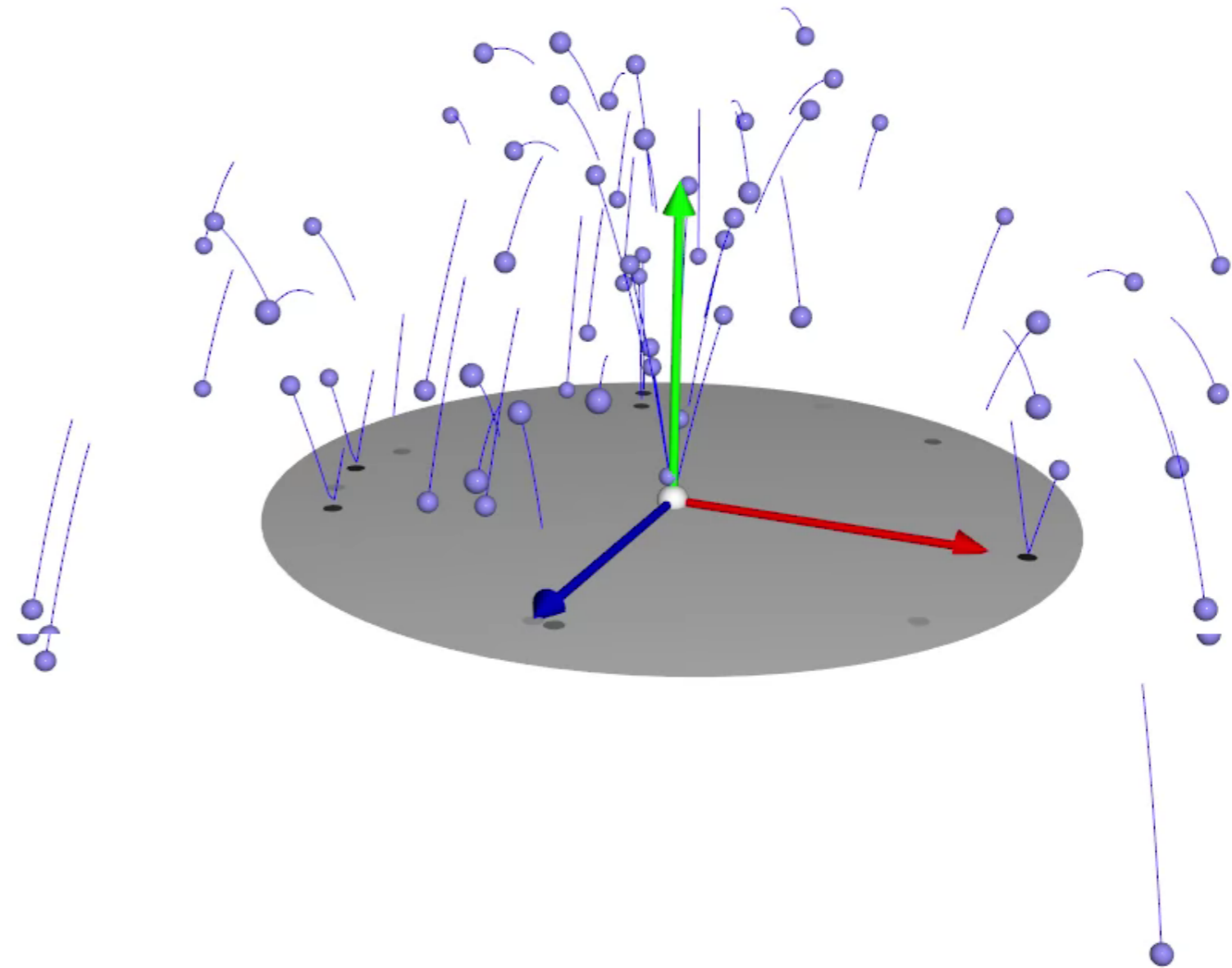
Q. What is the equation of motion ?  
(taking into account the bouncing)

$$p(t) = \dots$$

Help:

- Consider a particle emitted at time  $t = 0$
- At what time  $t_i$ , the particle touches the ground ?
- What is the new speed after impact ?
- Express the complete equation of trajectory  
(piecewise definition)

Bonus: How the "impact shadow" effect  
is computed ?

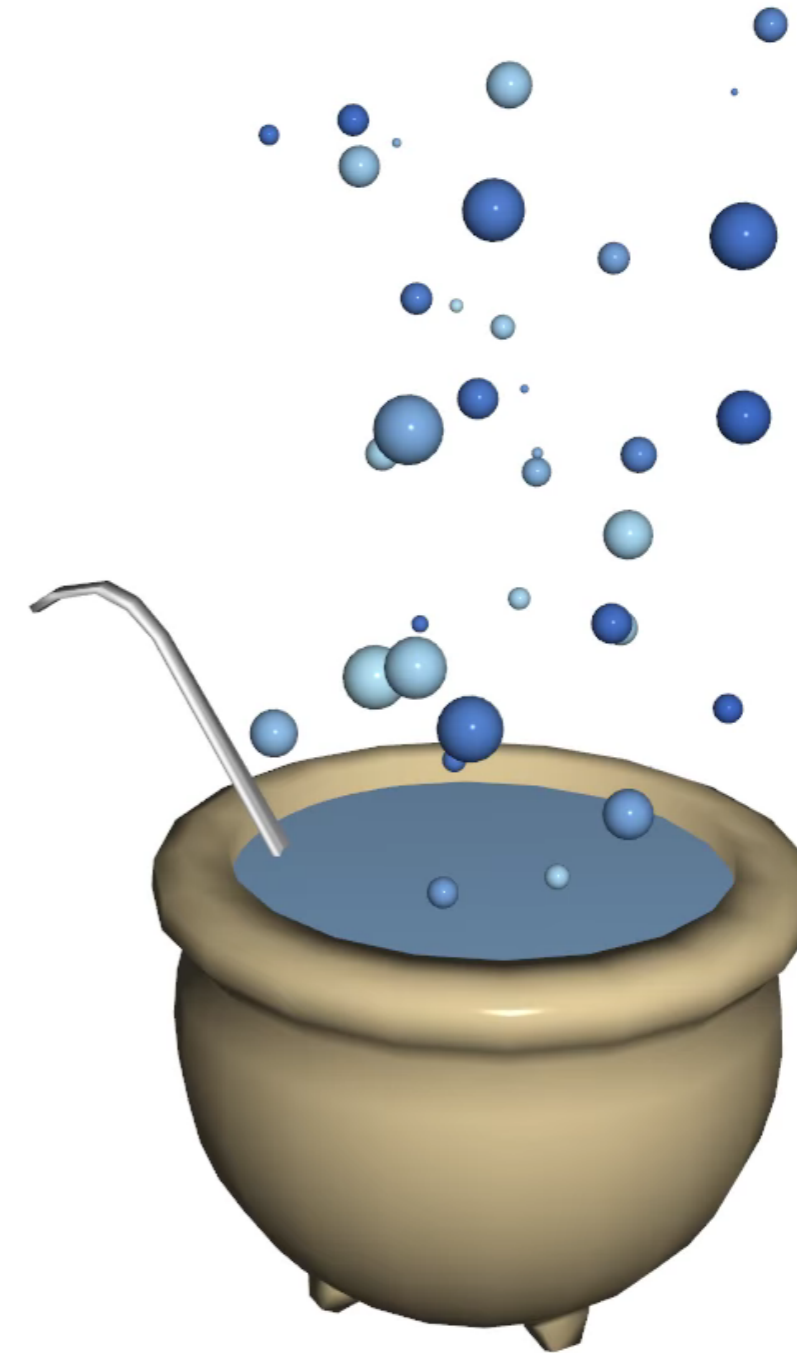


# Arbitrary motions

Trajectories are not restricted to physics-based equations

Q. *What are the parameters (and initial value) associated to each particle ?*

- position = ...
  - radius = rand([0,0.1])
  - ...
- *What is the equation of motion of a particle ?*
- $p(t) = \dots$



# Billboards, Impostors, Sprites

Particles can be displayed as small images/thumbnails (instead of simple sphere)

In practice

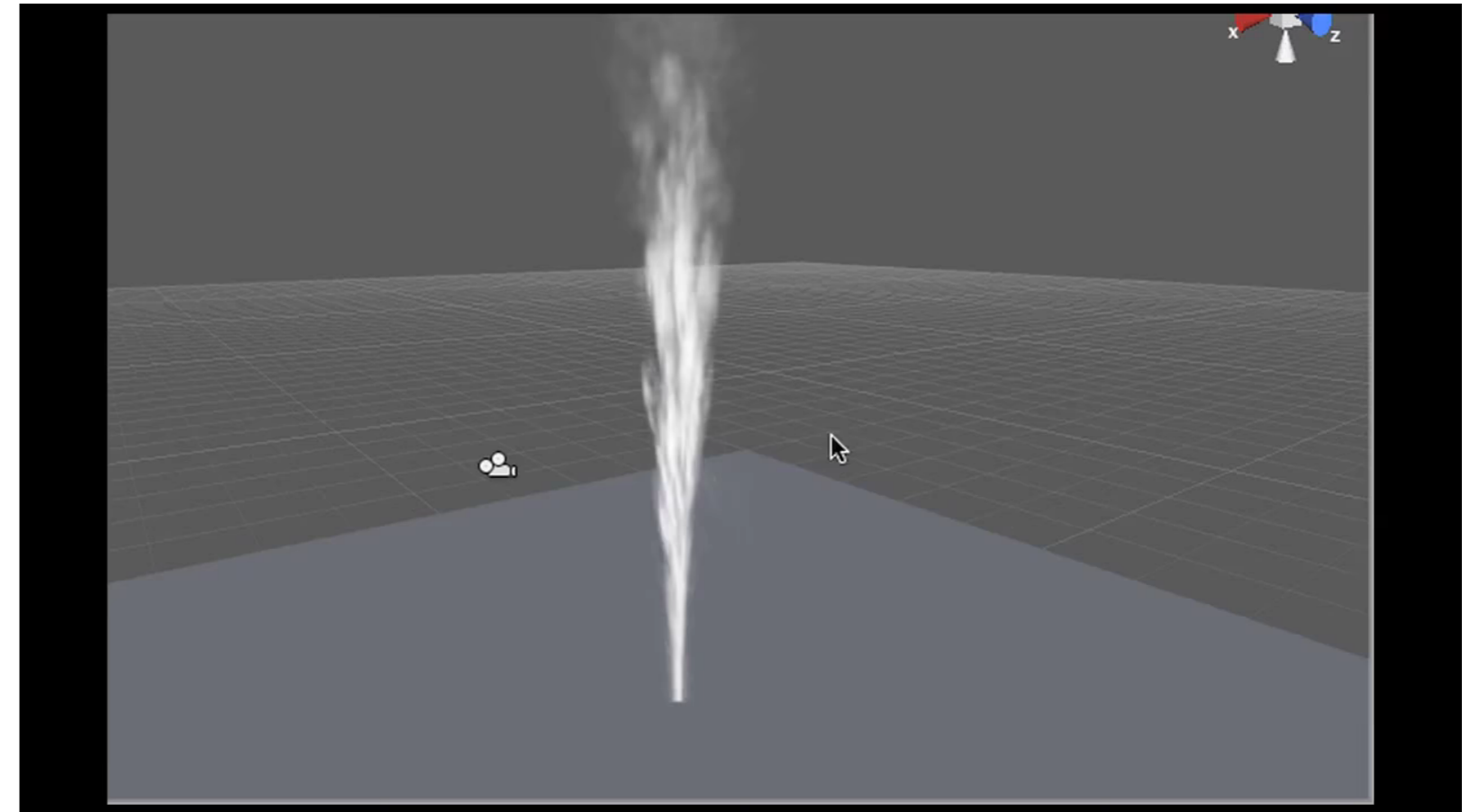
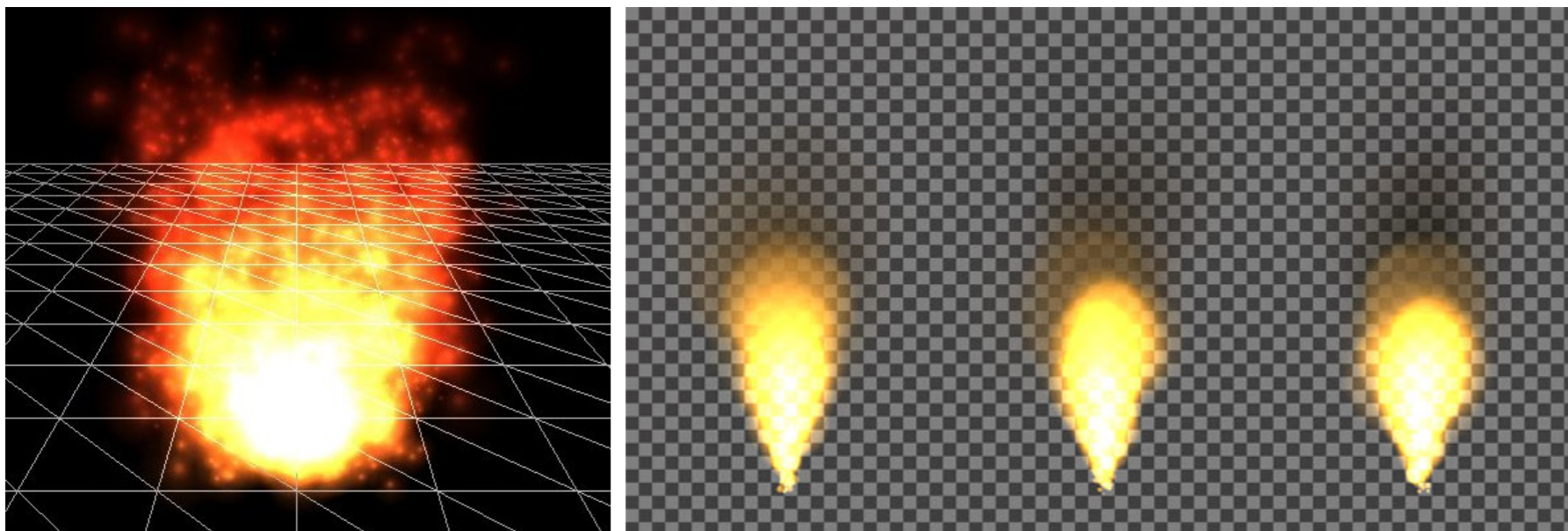
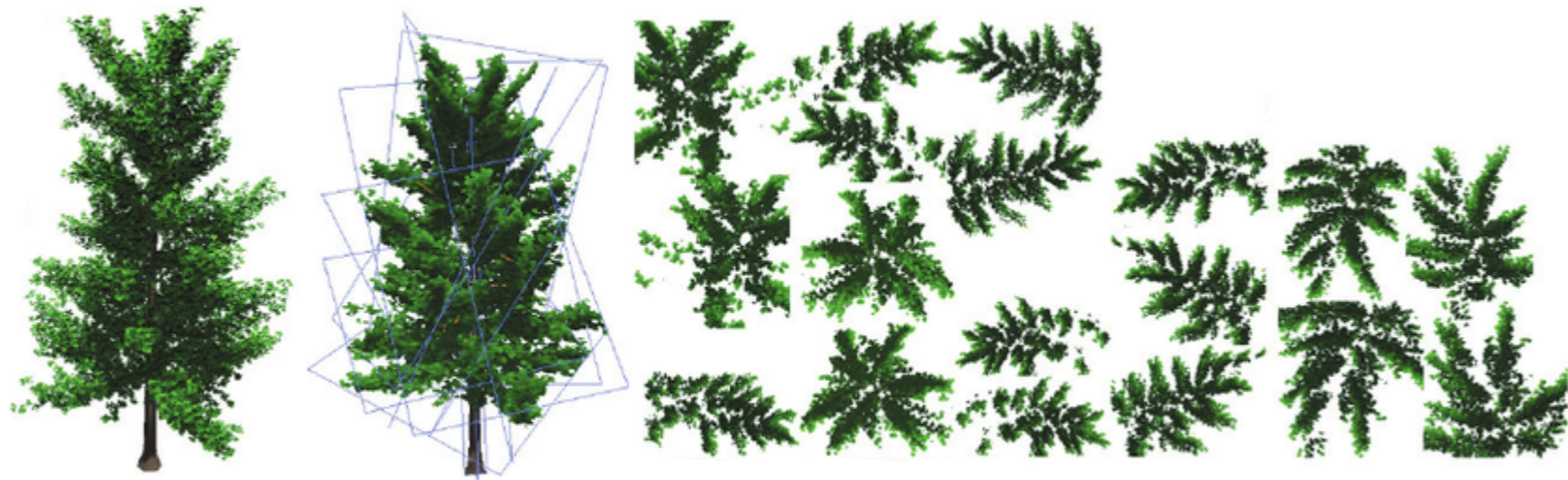
- Each particle is displayed as a quadrangle
- A texture is mapped on the quad
- The texture can contain transparency : quad geometry is invisible
  - Fake a more complex geometry
- The quad can be facing the camera at all time (billboard)
- The texture can be animated (sprite)
- Texture can be adapted to the point of view (impostors)



# Usage of billboards

Large use of billboards for complex models

*Vegetation, fire, smoke, etc.*



# Use case in production

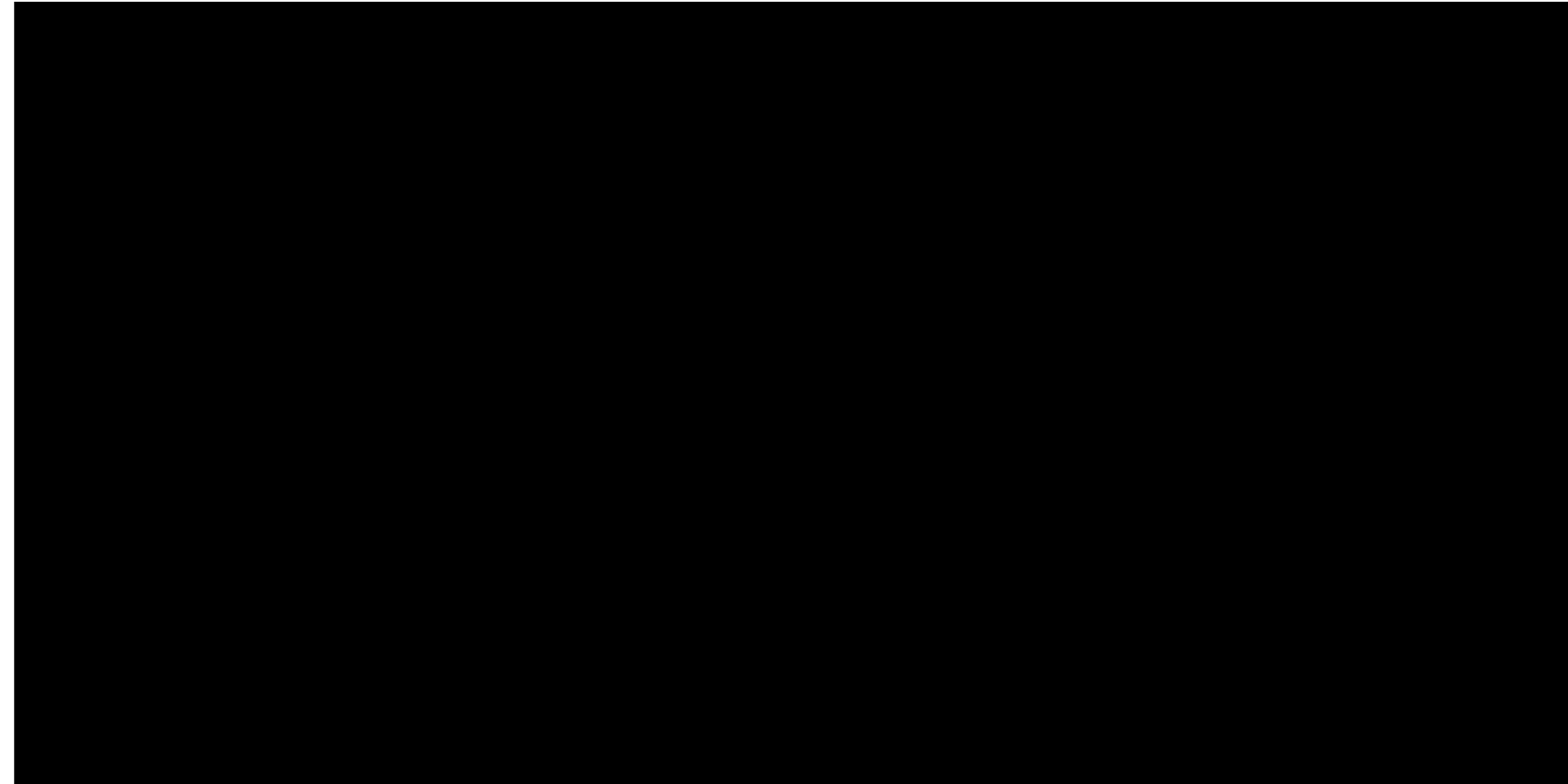
How to model the *horse heads made of water* ?



*The Lord of the Rings*

# Use case in production

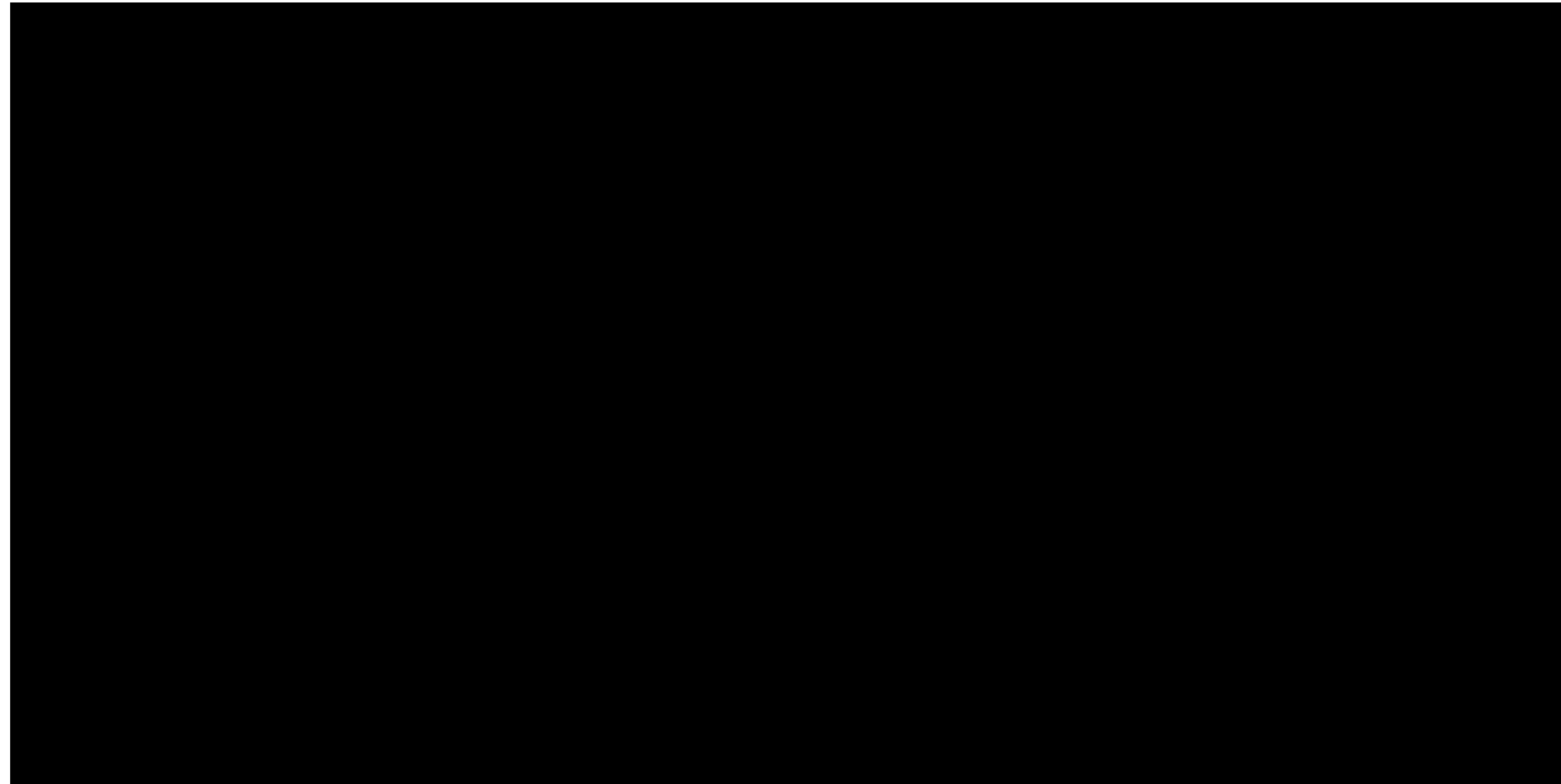
How to model the *horse heads made of water* ?



*The Lord of the Rings*

# Use case in production

Full Making-Of



*The Lord of the Rings*

# Implementation - transparency in OpenGL

Transparency is not straightforward using projection/rasterization rendering.

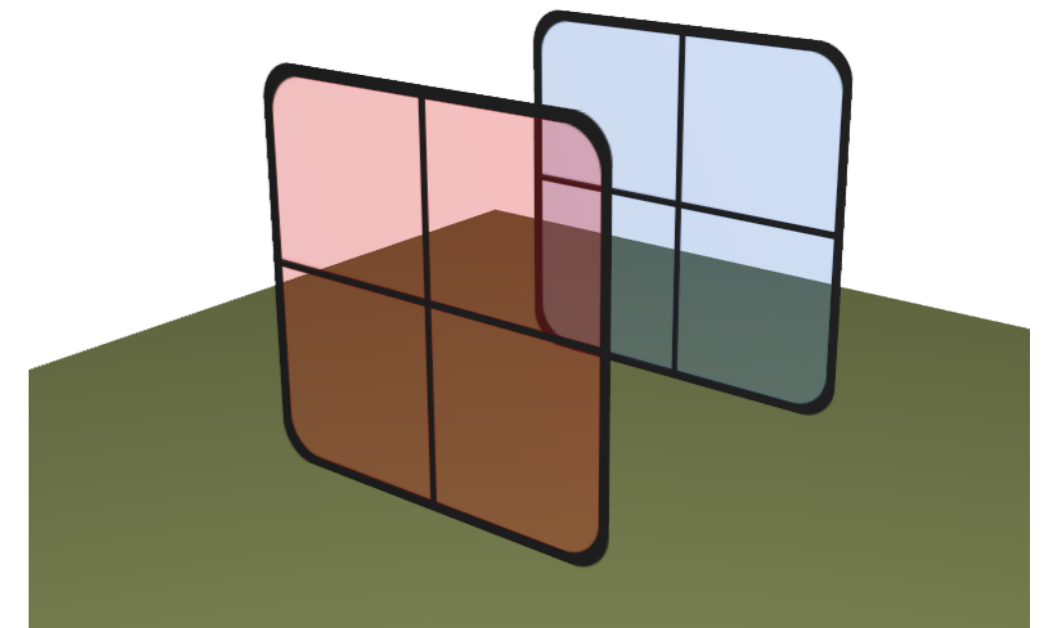
*No perfect solution*

⇒ Possibility to blend color for each pixel using  $\alpha$  channel (rgba)

Need to be explicitly activated in OpenGL

```
glEnable(GL_BLEND);  
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  
... Draw Calls ...  
glDisable(GL_BLEND);
```

- Various blending are possible for multiple effects (see [glBlendFunc](#))
- For transparency: (`GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_ALPHA`)  
⇒  $\text{newColor} = \alpha \text{ currentColor} + (1 - \alpha) \text{ previousColor}$



Important note:

The **order** of the draw calls is important (previousColor/currentColor)

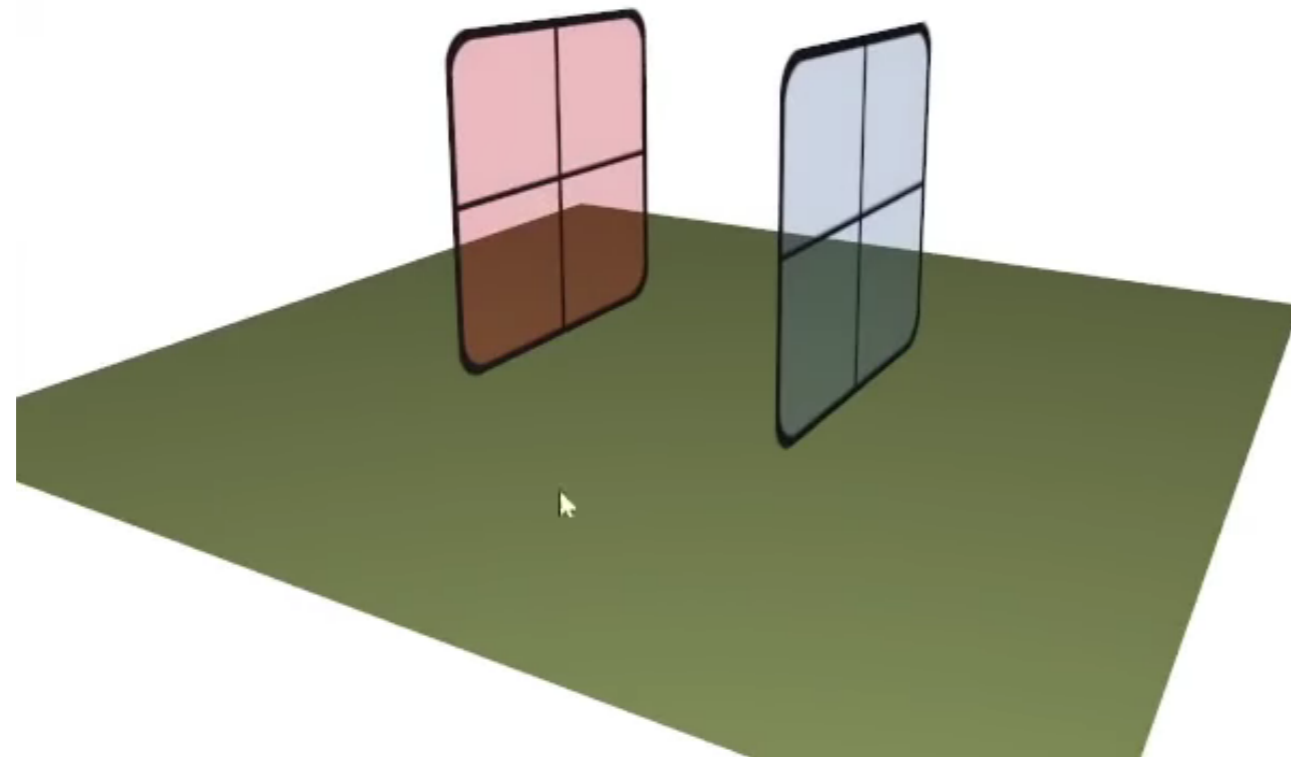
The depth-buffer cannot handle the notion of transparency

# Implementation - transparency in OpenGL

## General Process:

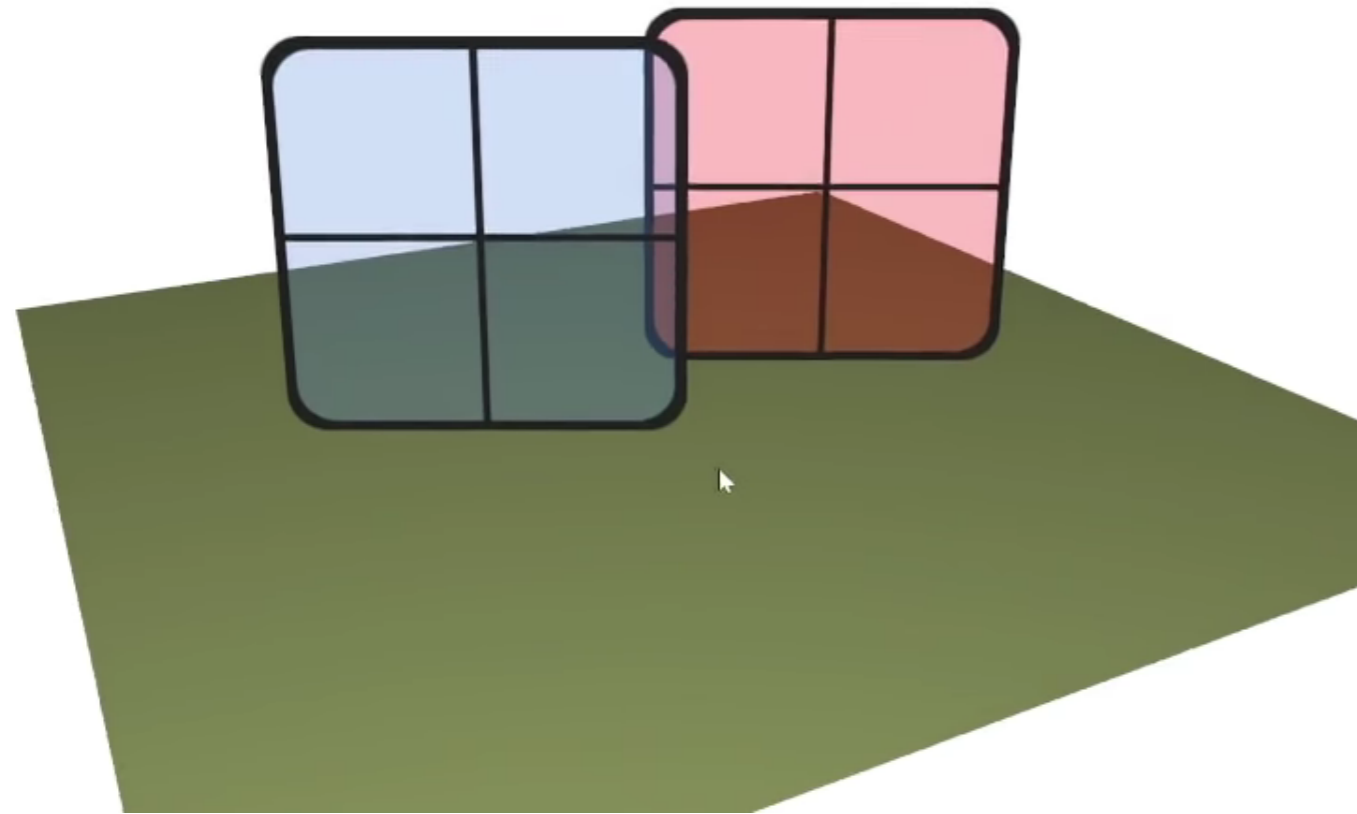
- 1. Draw all **non-transparent** objects
  - 2.a Activate color blending, deactivate depth buffer write
  - 2.b **Sort** transparent objects by depth to the camera
- $$p_{proj} = Proj \times ModelView \times p$$
- $$z_{depth} = p_{proj}.z / p_{proj}.w$$
- 3. **Draw transparent objects** from furthest to closest

```
... Draw all non-transparent objects ...
glEnable(GL_BLEND); // Color Blending
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDepthMask(false); // do not write on depth buffer
... Sort transparent objects by depth ...
... Draw all transparent objects from furthest to closest ...
glDisable(GL_BLEND);
// do not forget to re-activate depth buffer write
glDepthMask(true);
```



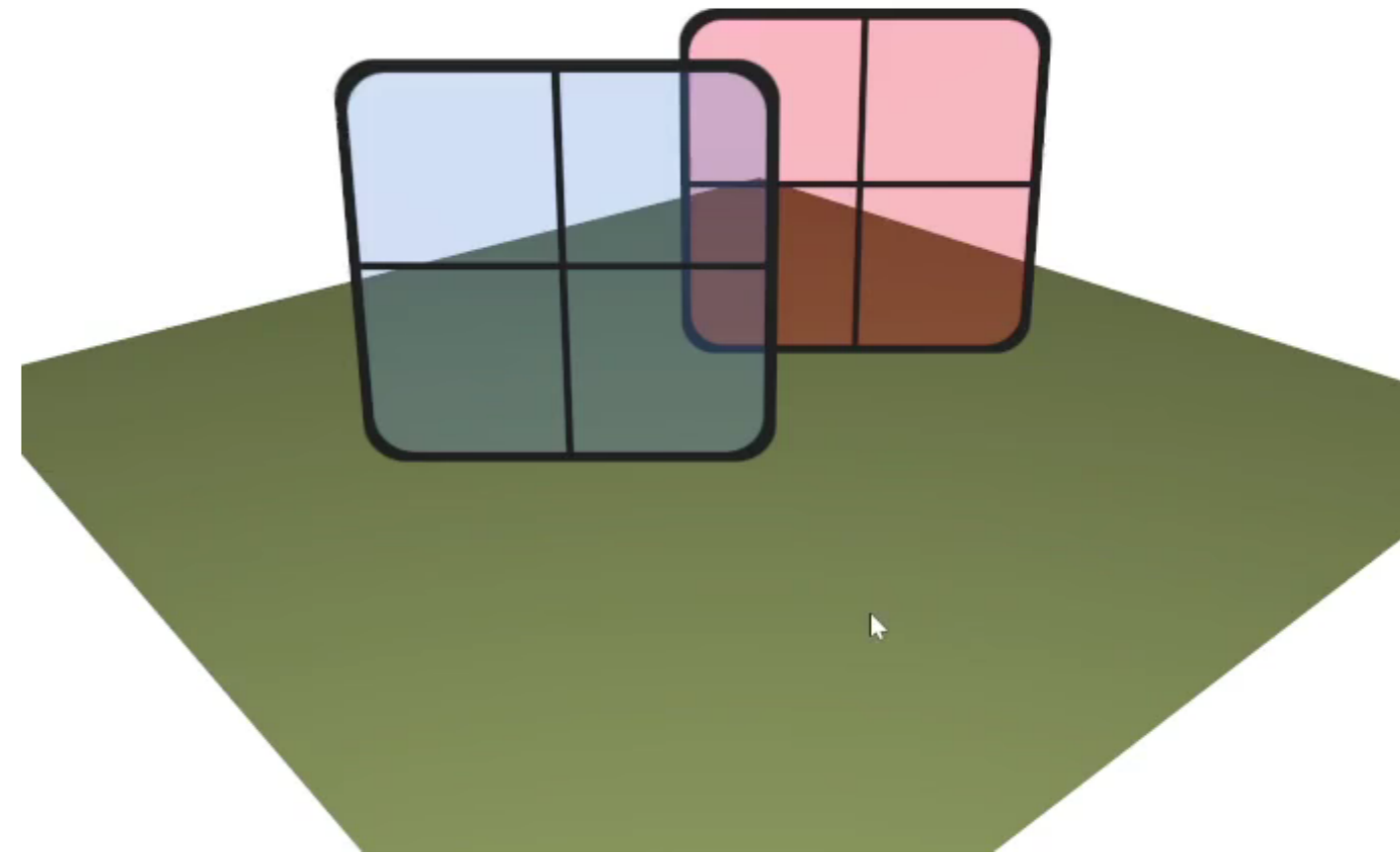
# Image Blending - common errors

Keeping depth-buffer write activated  
(without sorting)



*Strong artifact, cannot be used*

No sorting, depth buffer write de-activated



*Note:*

Can be OK for blurry sprites (like smokes)  
where element order is hard to distinguish.  
Avoids the cost of the sorting.

# Implementation - transparency in OpenGL

For impostors shape with fully opaque/transparent fragment:

**discard** can be used on transparent fragments

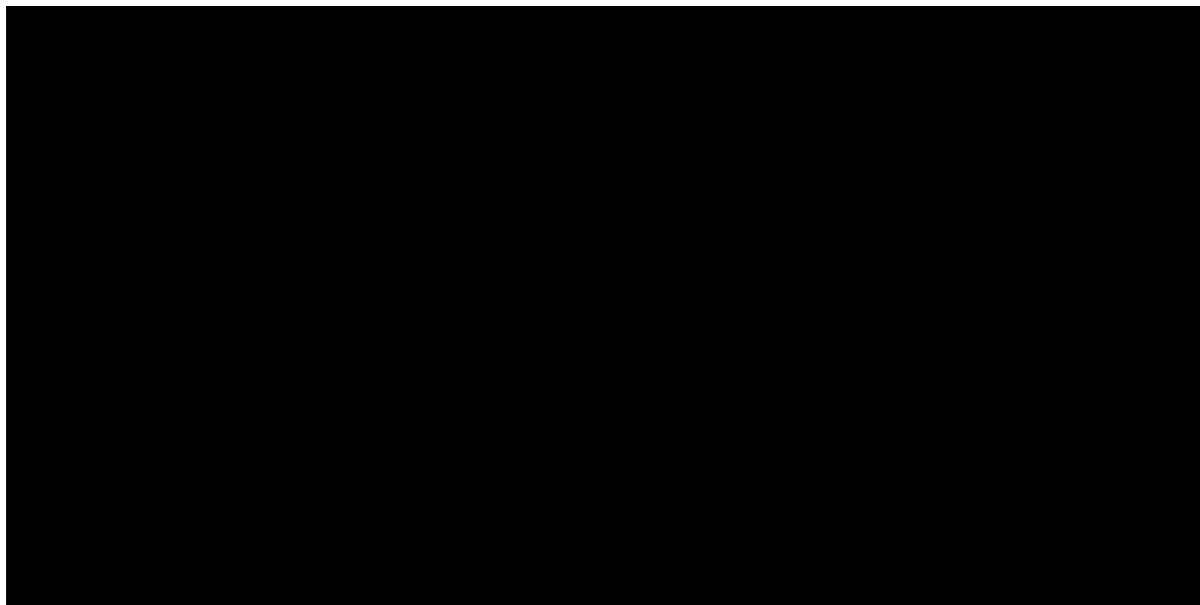
*discard = Stop fragment shader execution (nothing is written)*

(+) Generic, standard display

*no need to sort shapes, depth-buffer works*

(-) Possible aliasing artifacts

(-) Cannot be used on semi-transparent objects



*In fragment shader:*

```
float limit = 0.6;
if( texture.a < limit) {
    discard;
}
```

# Instancing

In particle system we need to display lots of similar shapes

Multiple draw calls ( $N > 1000$ ) can be costly

*each draw call requires synchronization b/w CPU-GPU*

```
// Slow for large N
for(int k=0; k<N; ++k)
    someOperator(k)
    draw(shape) // ... glDrawElements(...)
```

## Solution: Use Instancing

`glDrawArrays` → `glDrawArraysInstanced`

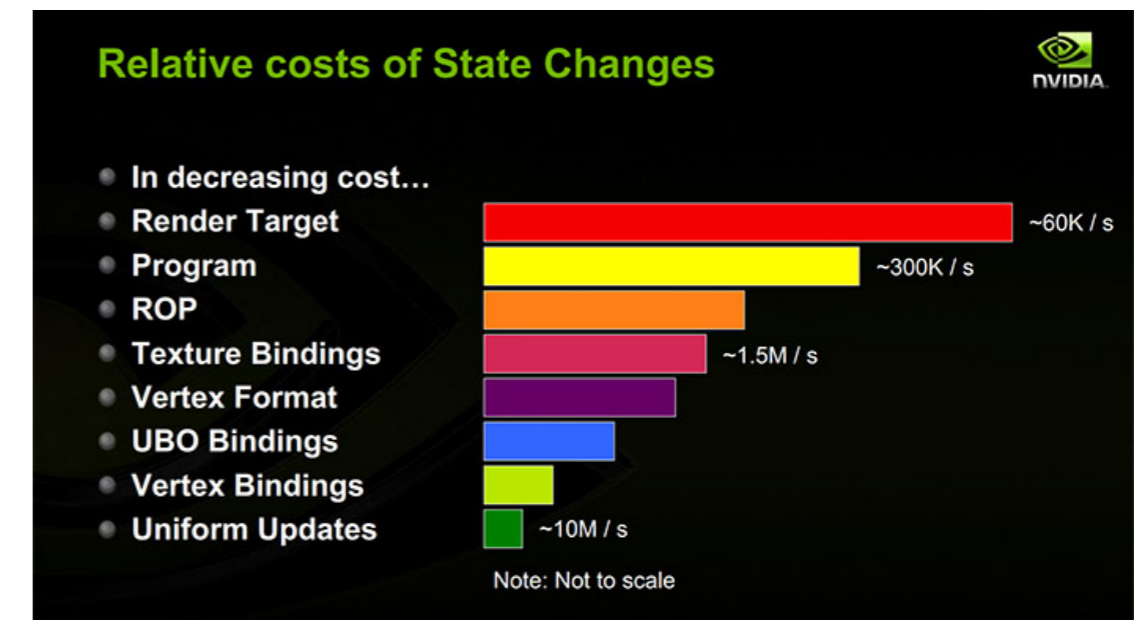
`glDrawElements` → `glDrawElementsInstanced`

*In the C++ code*

```
// 1 unique call, N is a parameter
drawInstanced(shape, N) // ... glDrawElementsInstanced(...,N)
// No more explicit loop
```

*In the Shader:*

```
// ...
someOperator(gl_InstanceID)
// ...
```



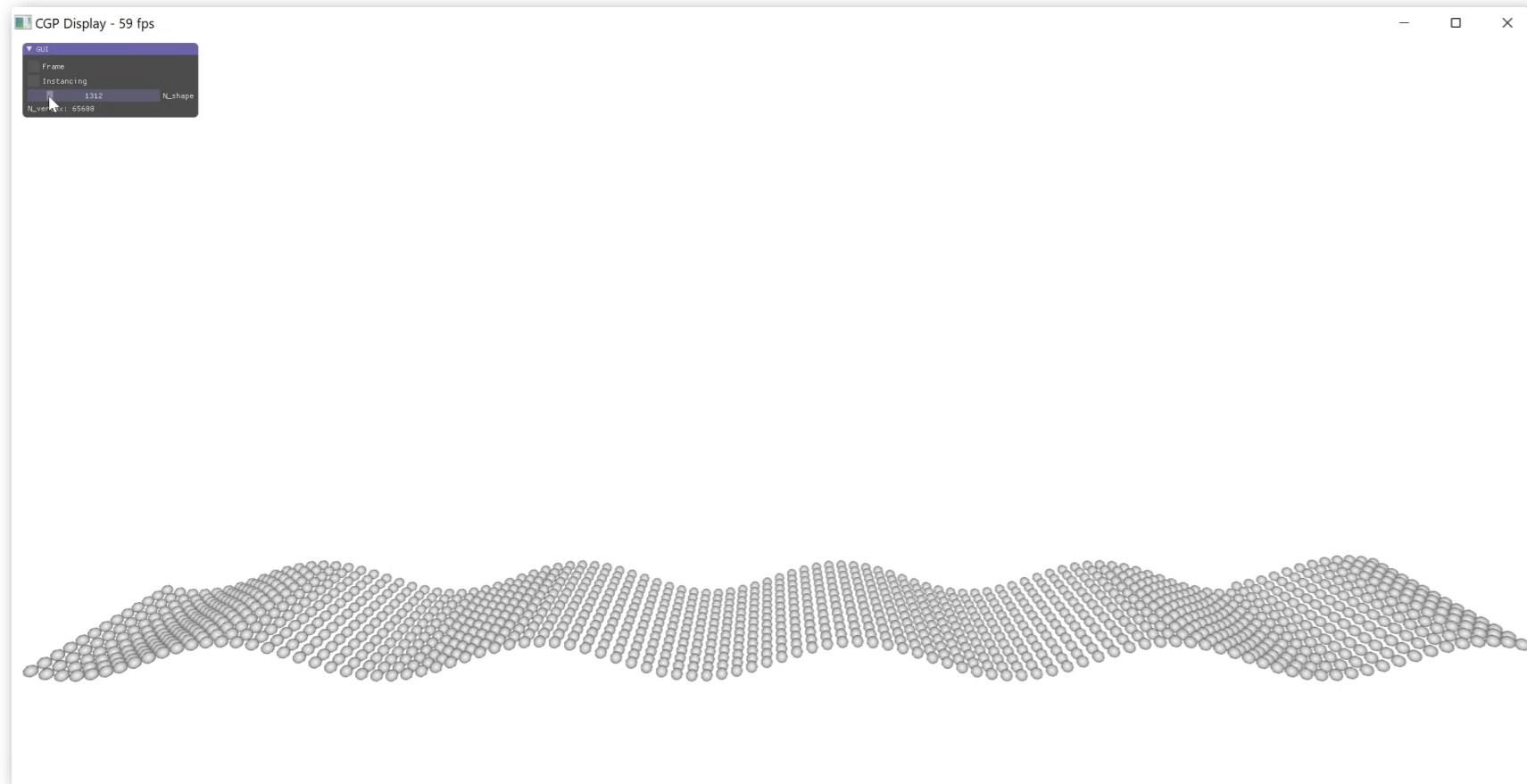
[ NVIDIA ]

(+) Full capability of your GPU

(-) Require a specific shader

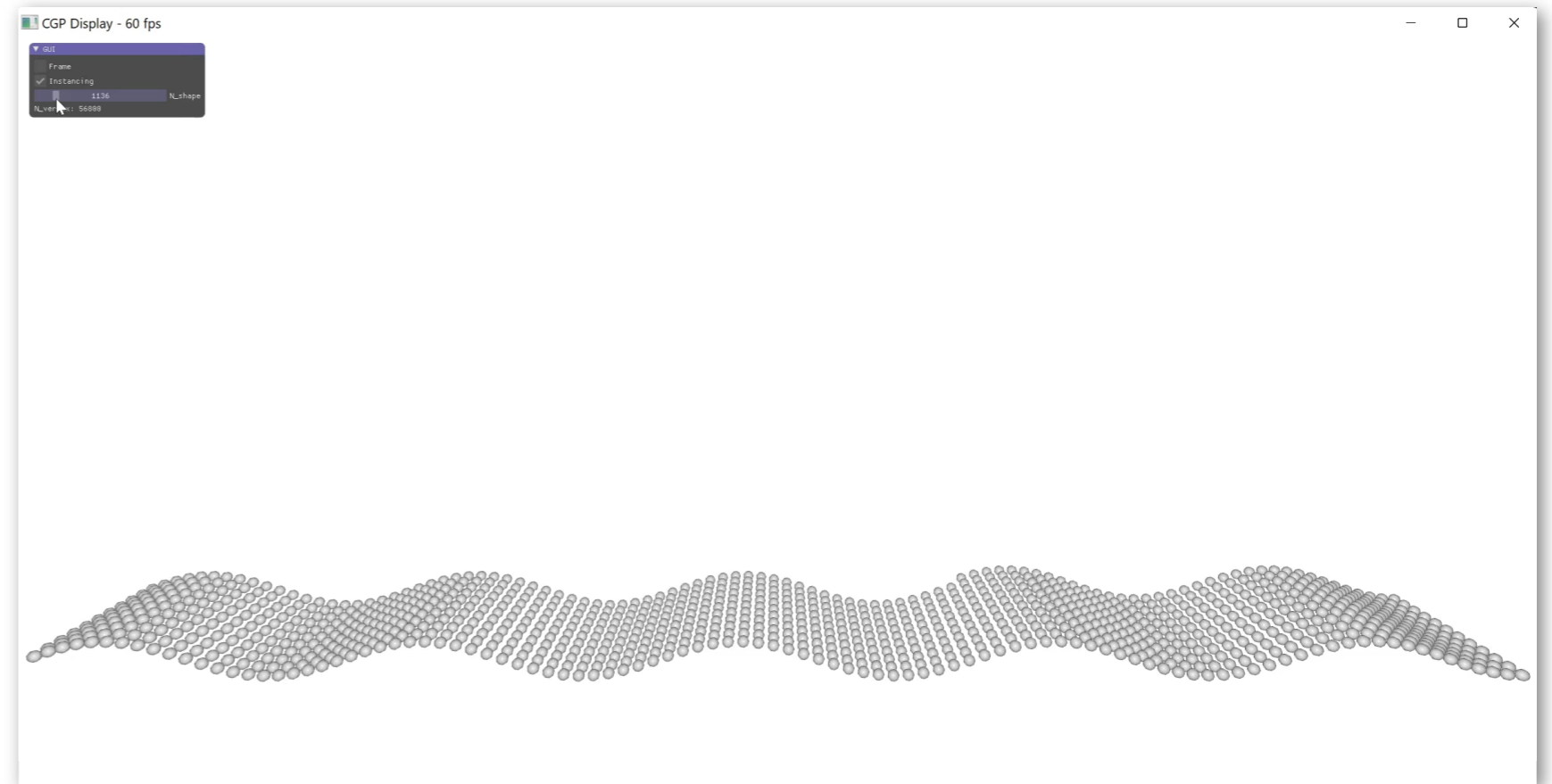
# Instancing Demo

## Without Instancing



Slow for 15,000 particles  
#vertices=750,000

## With Instancing



Can display 450,000 particles at 60fps (depends on GPU)  
#vertices=22,000,000