Particules, imposteurs/billboarding

Example de système de particules

Chute libre de particules

- Représentation géométrique des particules: sphere
- Equation du mouvement $p(t) = \frac{1}{2}gt^2 + v_0 t + p_0$

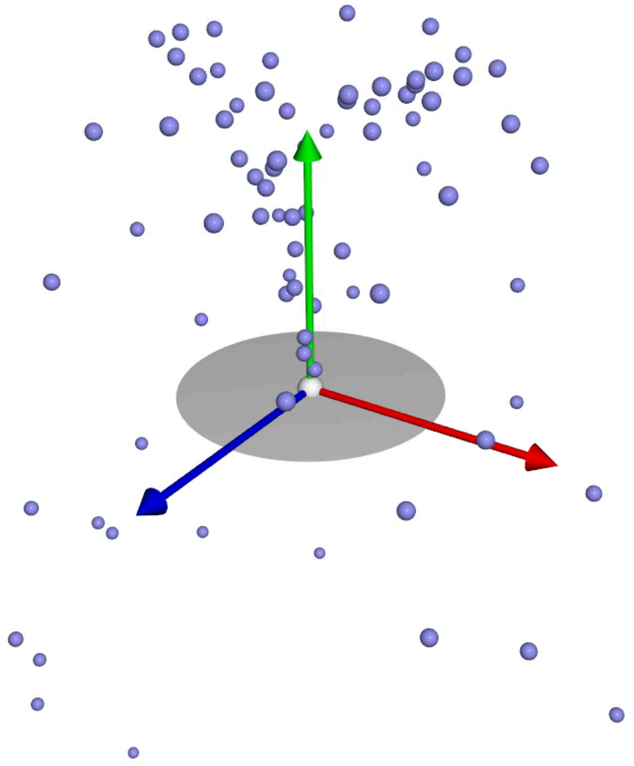
- Position initiale et vitesse peuvent avoir des composantes aléatoire
- Chaque particule peut avoir une durée de vie limitée

Q. *Quels sont les paramètres utilisés pour p_0 et v_0 dans cet example ?*

$p_0 = ...$

$v_0 = ...$

Note: Système de coordonnées (x, y, z) =(rouge, vert, bleu).



Rebond de sphères

Q. Quelle est l'équation du mouvement ?
(en prenant en compte un rebond)
$p(t) = ...$

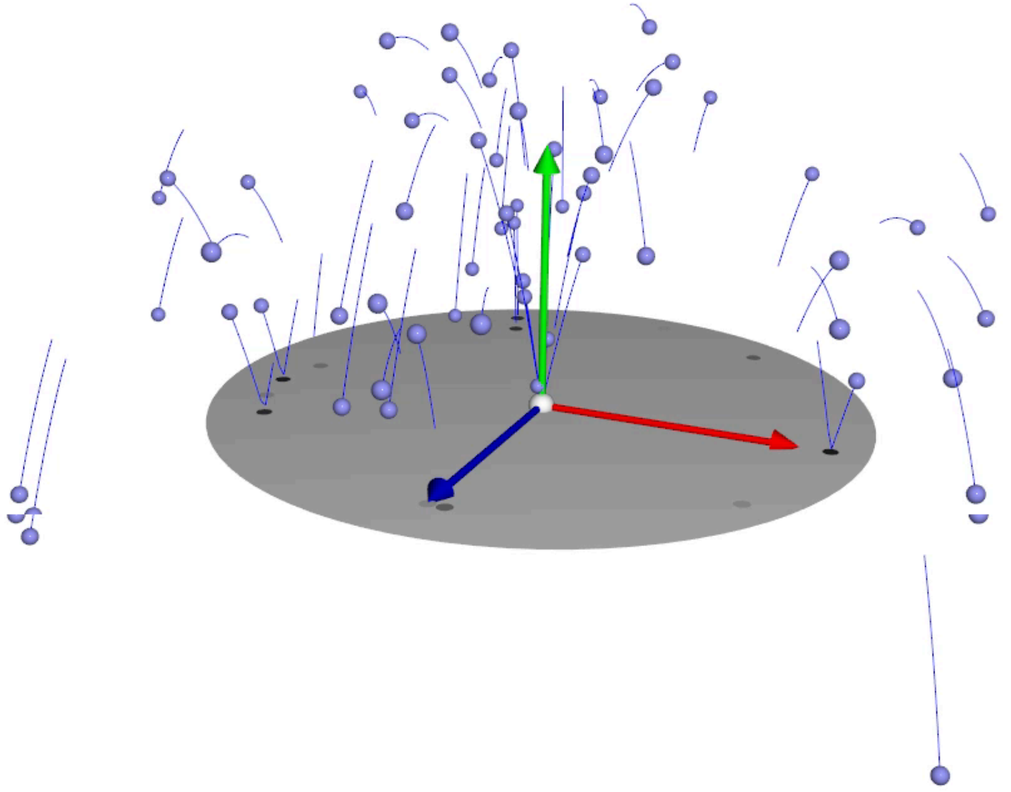
Aide:

- Considère une particule émise à $t = 0$

- A quel temps t_i, la particule touche le sol ?

- Quelle est la nouvelle vitesse après impact ?

- Exprimer l'équation du mouvement complète (définition par morceaux)



Arbitrary motions

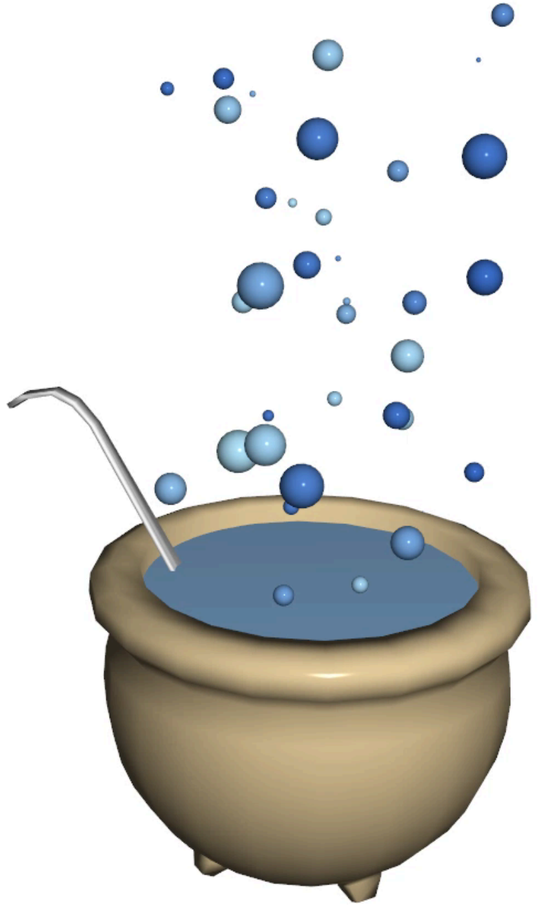
Trajectories are not restricted to physics-based equations

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 ?

- $p(t) = ...$



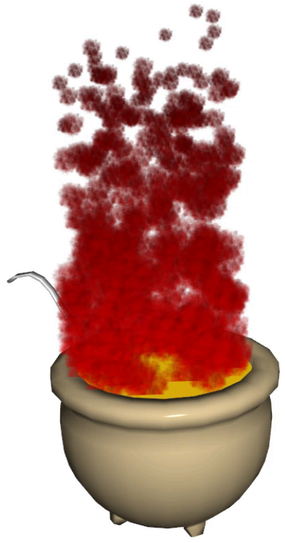
Billboards, Impostors, Sprites

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

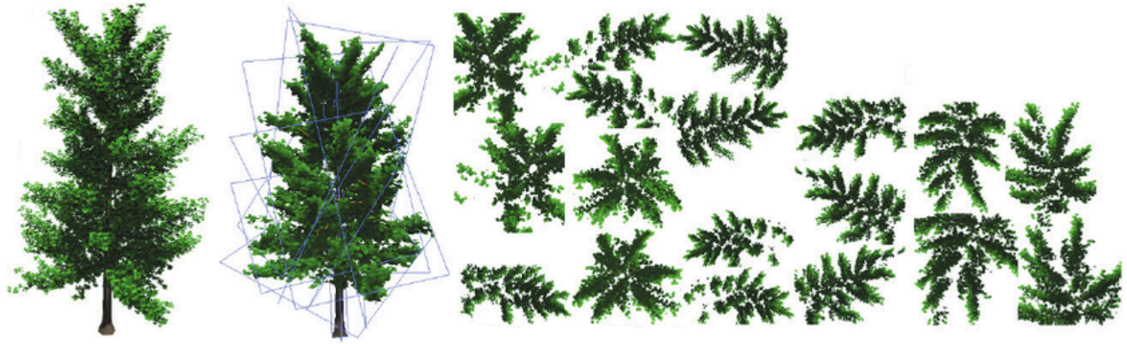
- The texture can contain transparency : quad geometry is invisible
  Fake a more complex geometry
- The quad can be facing the camera at all time (billboard)
- The texture can be animated (sprite)
- Texture can be adapted to the point of view (impostors)

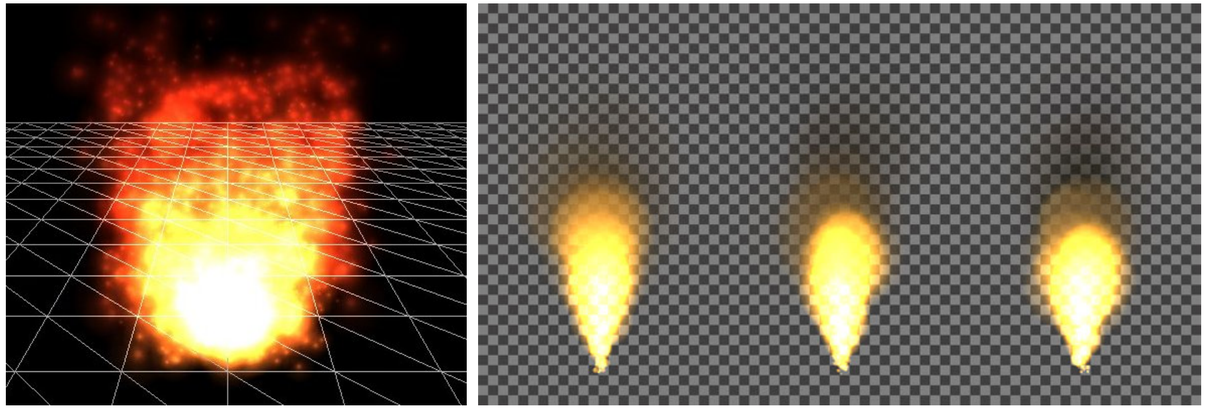


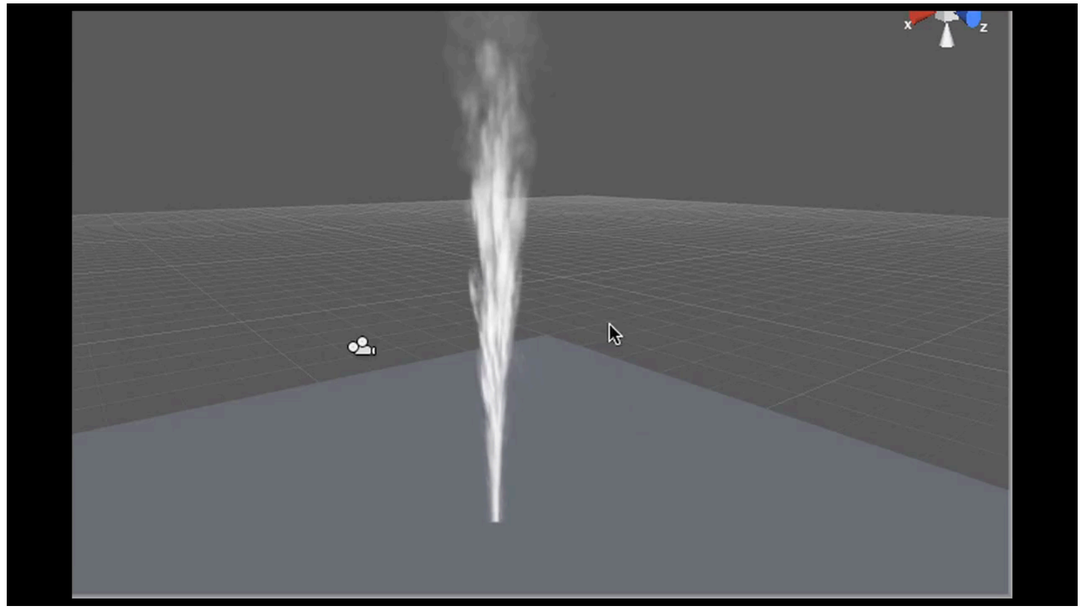
Usage of billboards

Large use of billboards for complex models

Vegetation, fire, smoke, etc.







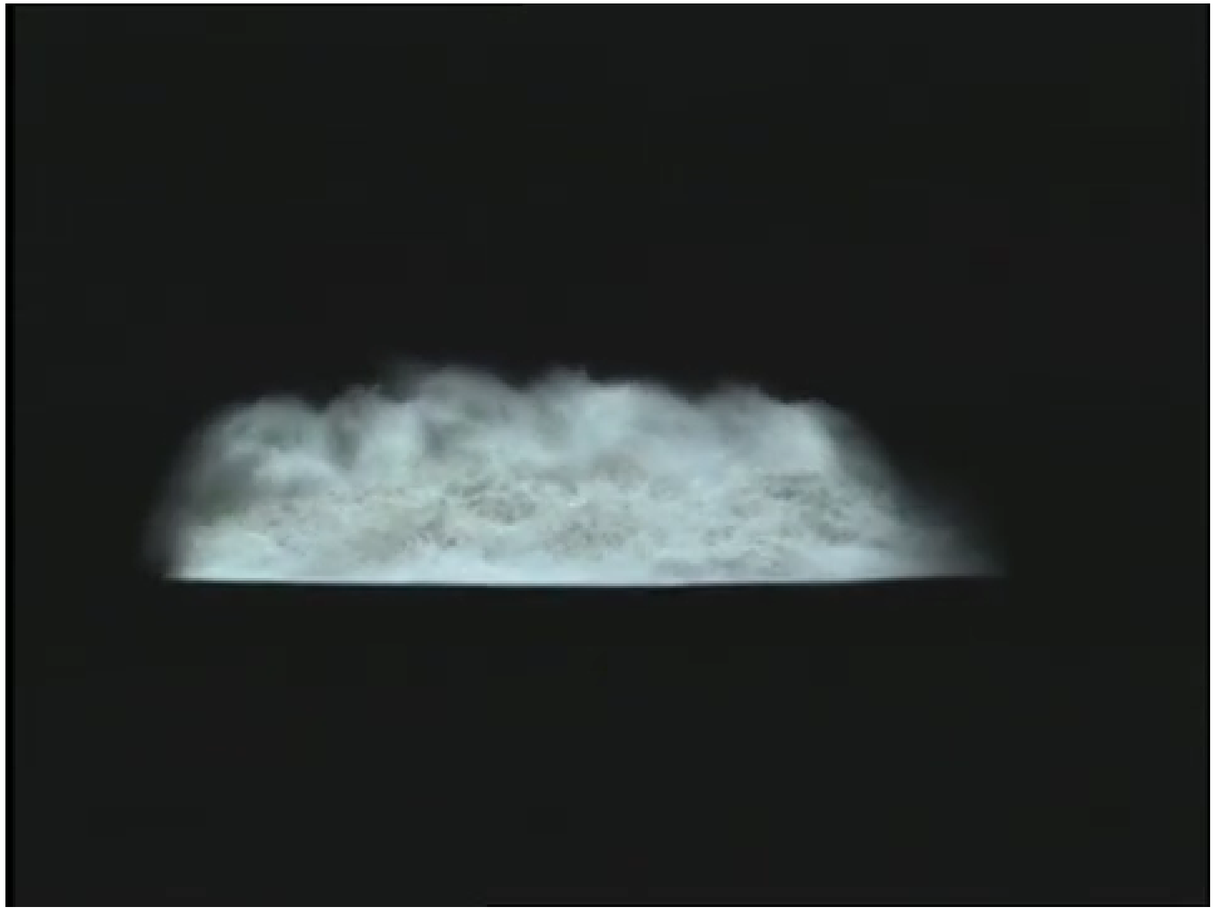
Use case in production

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

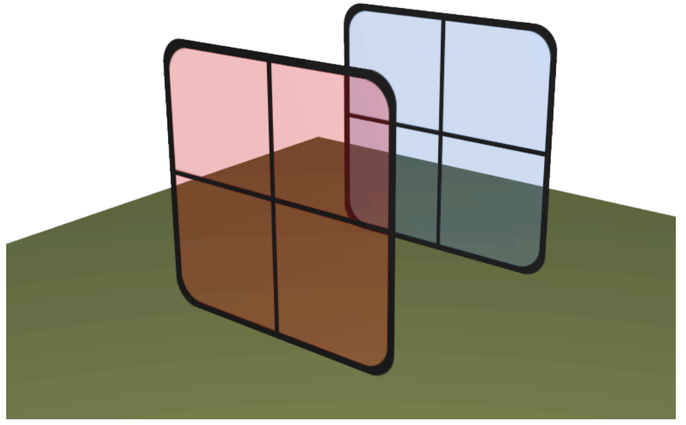
$\Rightarrow$ Possibility to blend color for each pixel using α channel (rgba)

Need to be explicitely 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)*

  $\Rightarrow$ newColor = α currentColor + $(1 - \alpha)$ 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, desactivate 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);
```

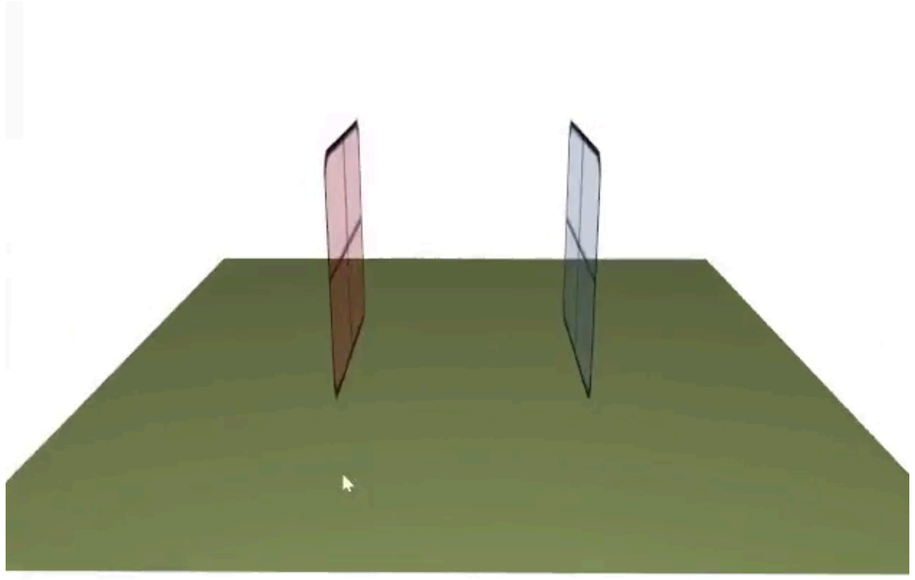
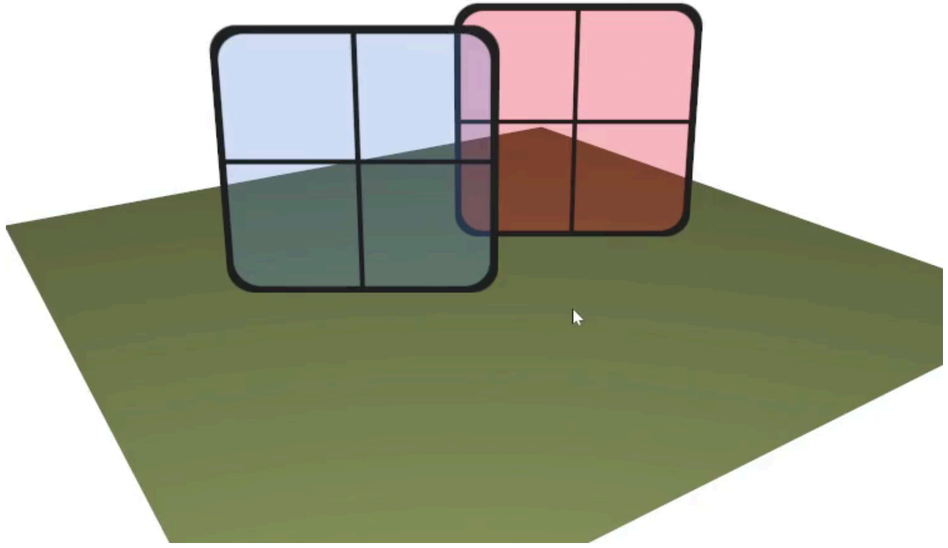


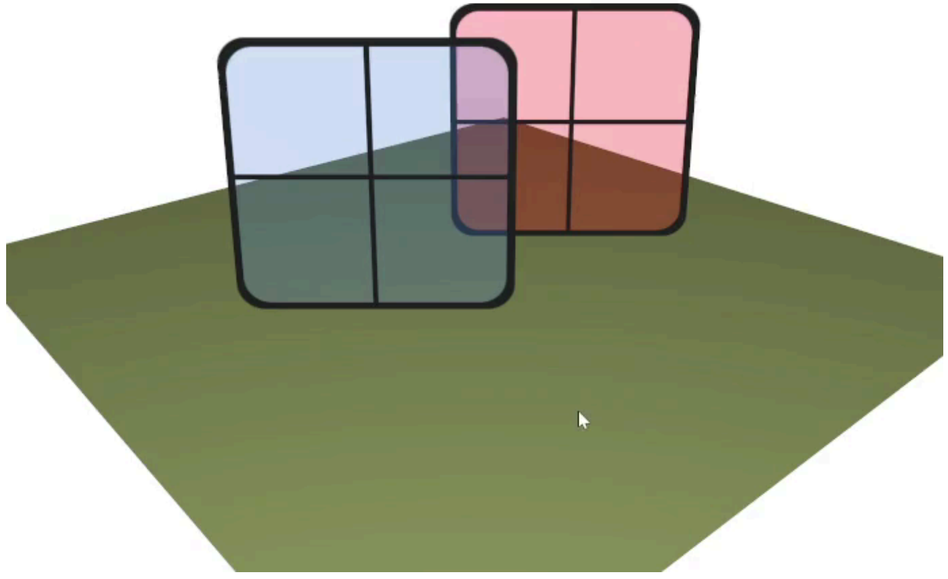
Image Blending - common errors

Keeping depth-buffer write activated (without sorting)



Strong artifact, cannot be used

No sorting, depth buffer write de-activated



Note:

Can be OK for blury 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

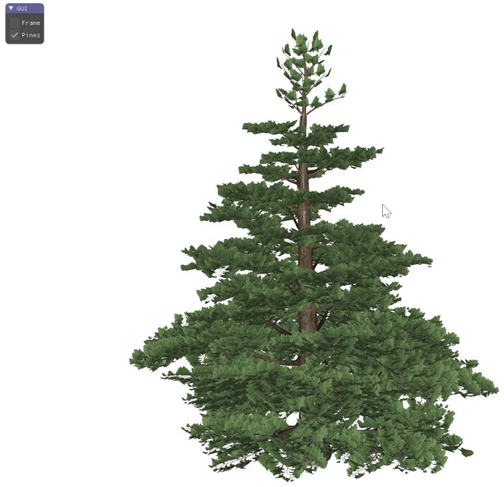
discard = Stop fragment shader execution (nothing is written)

(+) 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

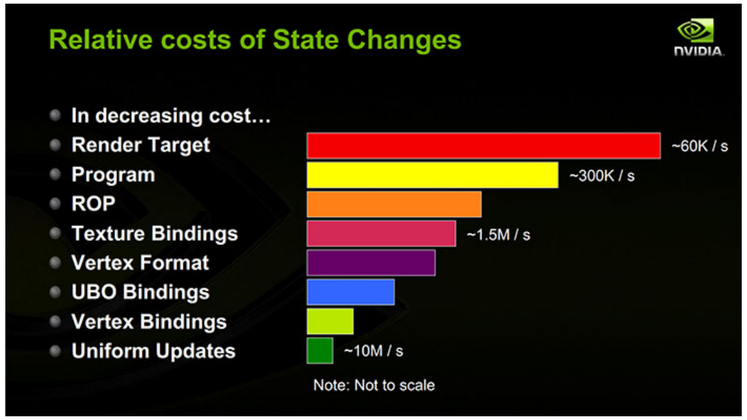
glDrawElements → glDrawElementsInstanced

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)
// ...
```



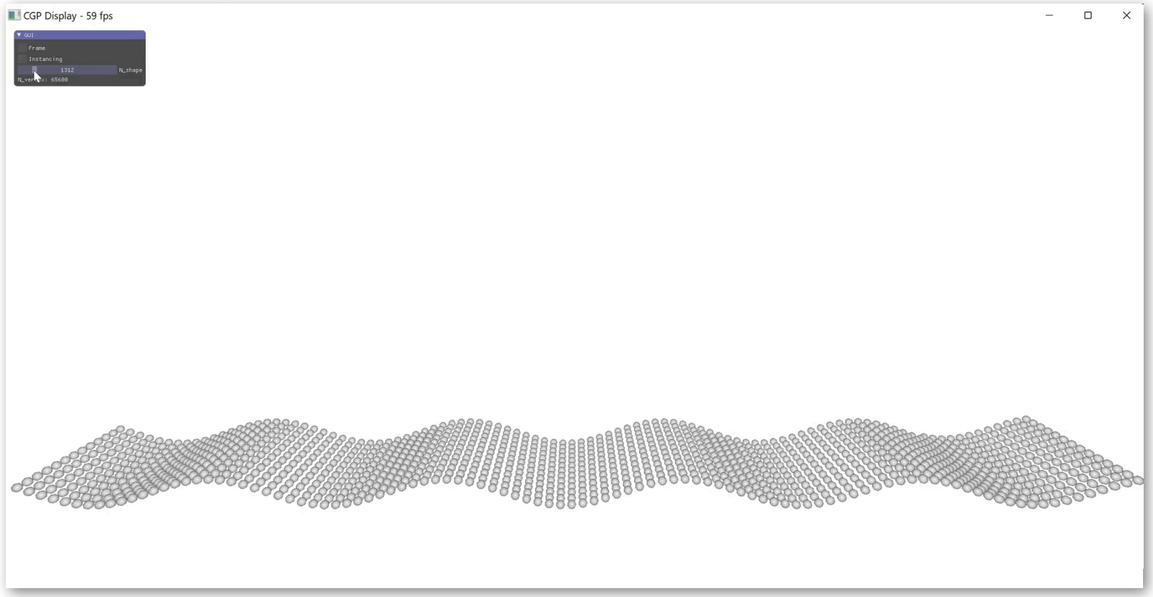
[NVIDIA]

(+) Full capability of your GPU

(-) Require a specific shader

Instancing Demo

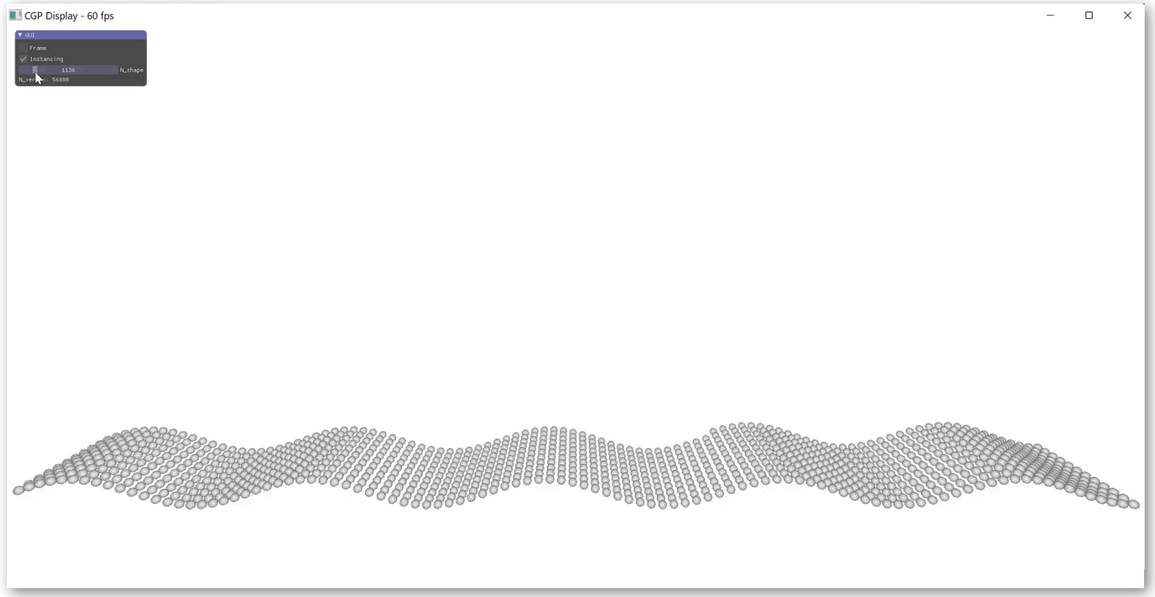
Without Instancing



Slow for 15,000 particles

#vertices=750,000

With Instancing



Can display 450,000 particles at 60fps (depends on GPU)

#vertices=22,000,000