

# PROGRAMMATION OPENGL

- OPENGL 4 - SUITE -

Jonathan Fabrizio

<http://jo.fabrizio.free.fr>

Version : Tue May 17 08:58:23 2022

# OpenGL : Pour aller plus loin

- ▶ *Off-screen rendering/Render to texture/MRT - Multiple Render Targets*
  - ▶ *Deferred Rendering*
  - ▶ Gestion des ombres
  - ▶ Flou
  - ▶ *Post processing*
  - ▶ *Picking*
  - ▶ ...
- ▶ Brouillard
- ▶ Moteur à particules
  - ▶ Feu
  - ▶ Pluie
  - ▶ ...
- ▶ *Billboard*
  - ▶ Feu
  - ▶ ...
- ▶ Rendu de l'eau

# OpenGL : *Off-screen rendering/Render to texture/MRT - Multiple Render Targets*

- ▶ Utilisation :
  - ▶ *Deferred Rendering*
  - ▶ Gestion des ombres
  - ▶ Blur
  - ▶ *Post processing*
  - ▶ *Picking*
  - ▶ ...

# OpenGL : *Off-screen rendering/Render to texture/MRT - Multiple Render Targets*

- ▶ *Frame buffer Object*
  - ▶ Stockage coté carte vidéo
- ▶ *Accumulation buffer*
- ▶ ...

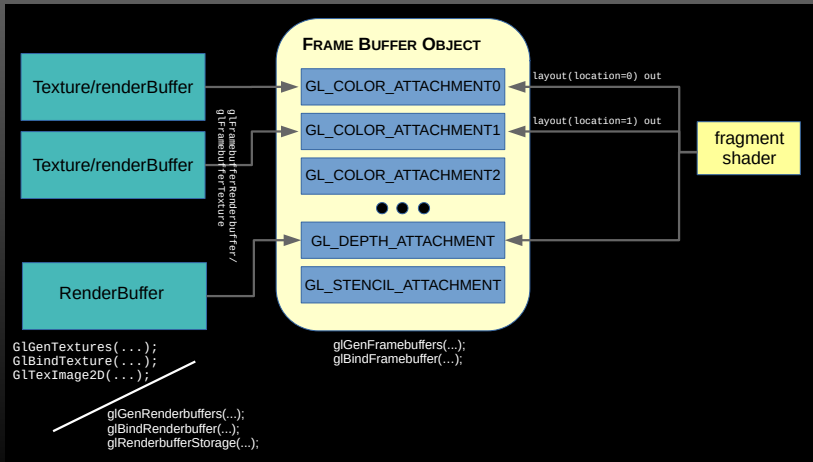
# OpenGL : *Off-screen rendering/Render to texture/MRT - Multiple Render Targets*

- ▶ *Frame buffer Object*
  - ▶ Stockage coté carte vidéo

- ▶ *Accumulation buffer*
- ▶ ...

# OpenGL : Off-screen rendering/Render to texture/MRT - Multiple Render Targets

## Frame buffer Object



# OpenGL : *Off-screen rendering/Render to texture/MRT - Multiple Render Targets*

## *Frame buffer Object*

```
glGenRenderbuffers(nb_render_buffers, render_buffers);

glBindRenderbuffer(GL_RENDERBUFFER, render_buffers[color_buffer]);
glRenderbufferStorage(GL_RENDERBUFFER, GL_RGBA, 512, 512);

glBindRenderbuffer(GL_RENDERBUFFER, render_buffers[depth_buffer]);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 512, 512);

glGenFramebuffers(1, &frame_buffer_object_id);
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, frame_buffer_object_id);

glFramebufferRenderbuffer(GL_DRAW_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
                           GL_RENDERBUFFER, render_buffers[color_buffer]);
glFramebufferRenderbuffer(GL_DRAW_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                           GL_RENDERBUFFER, render_buffers[depth_buffer]);
```

# OpenGL : *Off-screen rendering*

## *Frame buffer Object*

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, frame_buffer_object_id);  
...  
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);  
...  
glutSwapBuffers();
```



# Exemples

# Deferred Rendering

Problème : beaucoup de calculs sont faits pour des points qui ne seront finalement pas visibles !

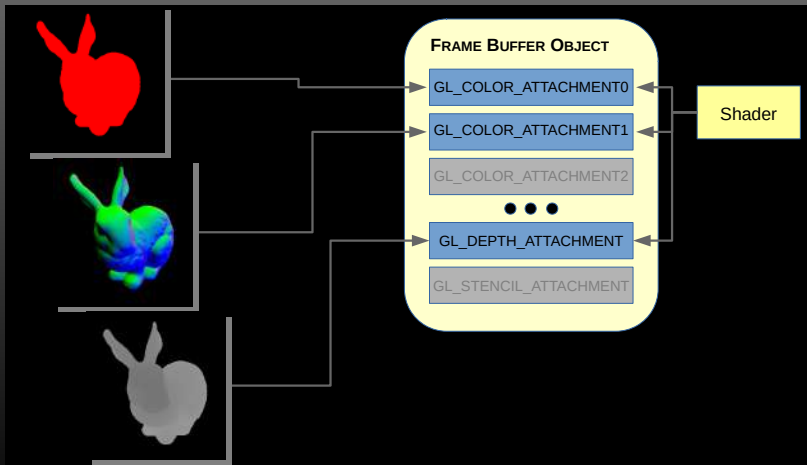
# Deferred Rendering

Problème : beaucoup de calculs sont faits pour des points qui ne seront finalement pas visibles !

- ▶ Solution : retarder le calcul final (illumination). Calcul final fait au dernier moment.
- ▶ Avantage : ce calcul n'est pas fait pour les points qui ne sont pas visibles

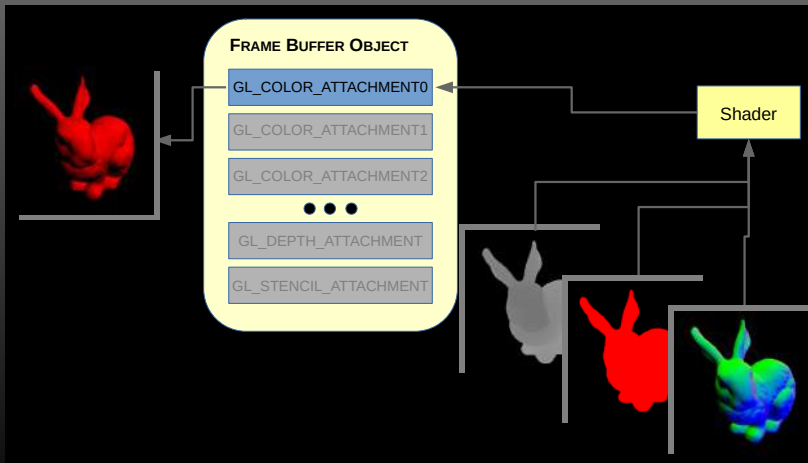
# Deferred Rendering

## Etape 1 (*offscreen*)



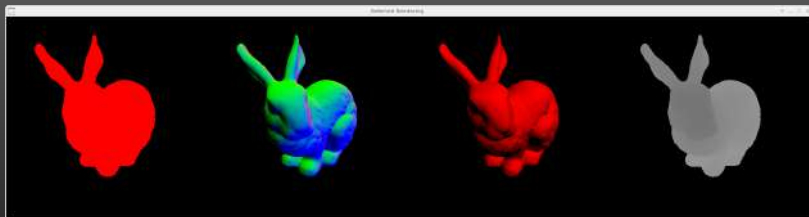
# Deferred Rendering

## Etape 2



# Deferred Rendering

Résultat



Couleurs

Normales

Combinaison

(z-buffer)

# Deferred Rendering

## *MRT : Multiple-render target*

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, frame_buffer_object_id);  
GLenum bufs[2] = {GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1};  
glDrawBuffers(2, bufs);
```

---

```
layout(location=0) out vec4 output_color;  
layout(location=1) out vec4 output_normal;  
output_color = vec4(color, 1.0);  
output_normal = vec4(normal, 1.0);
```

- ▶ *depth shadow map*
- ▶ *second depth shadow map*



► *depth shadow map*

# OpenGL : depth shadow map

# OpenGL : depth shadow map

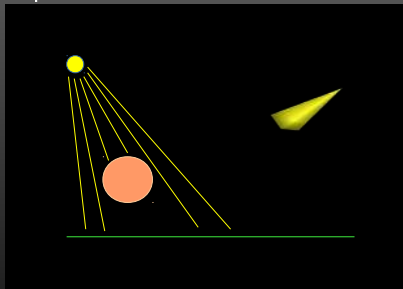
- ▶ Faire le rendu depuis la lumière pour évaluer ce qui est visible depuis la source lumineuse.

# OpenGL : depth shadow map

- ▶ Faire le rendu depuis la lumière pour évaluer ce qui est visible depuis la source lumineuse.
- ▶ Faire le rendu en tenant compte de parties visibles ou pas depuis la source lumineuse.

# OpenGL : depth shadow map

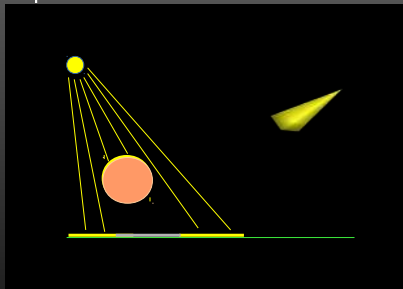
1. Faire le rendu depuis la lumière pour évaluer ce qui est visible depuis la source lumineuse.



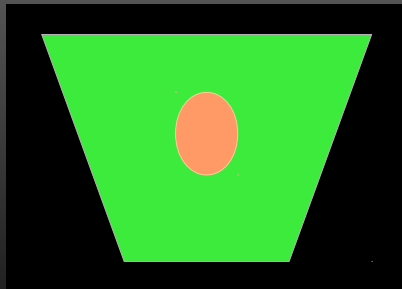
Scene

# OpenGL : depth shadow map

1. Faire le rendu depuis la lumière pour évaluer ce qui est visible depuis la source lumineuse.



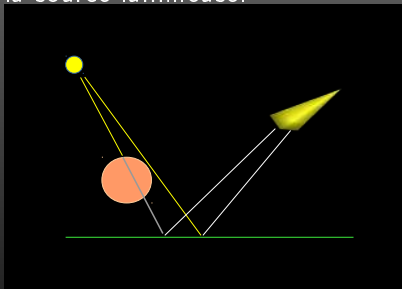
Scene



Rendu depuis la source lumineuse

# OpenGL : depth shadow map

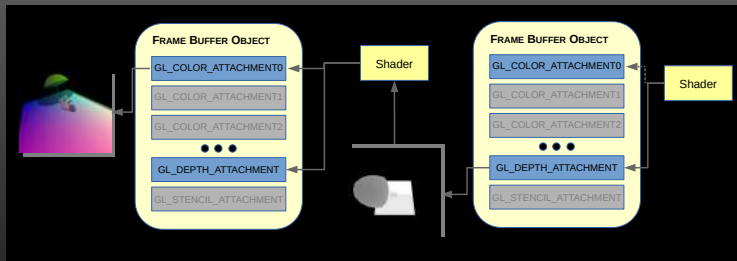
2. Faire le rendu en tenant compte de parties visibles ou pas depuis la source lumineuse.



Scene

Il faut exprimer les points visibles par la caméra dans le repère de la lumière pour pouvoir comparer la profondeur

# OpenGL : depth shadow map





# OpenGL : depth shadow map

## Initialise *FBO*

```
g|GenTextures(1, &(render_buffers[output_buffer]));
g|BindTexture(GL_TEXTURE_2D, render_buffers[output_buffer]);
g|TexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1024, 1024, 0, GL_RGB, GL_UNSIGNED_BYTE,
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

g|GenTextures(1, &depth_texture);
g|BindTexture(GL_TEXTURE_2D, depth_texture);
g|TexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32, 1024, 1024, 0, GL_DEPTH_C
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
g|TexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

g|BindTexture(GL_TEXTURE_2D, 0);

g|GenFramebuffers(1, &depth_frame_buffer_object_id);
g|BindFramebuffer(GL_FRAMEBUFFER, depth_frame_buffer_object_id);

g|FramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, render_buffers[output
g|FramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, depth_texture, 0);
g|DrawBuffer(GL_COLOR_ATTACHMENT0);

if (g|CheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
```

# OpenGL : depth shadow map

## Rendu depuis la source lumineuse

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, depth_frame_buffer_object_id);
glViewport(0, 0, 1024, 1024);
glDrawBuffer(GL_COLOR_ATTACHMENT0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glUseProgram(light_program);
glBindVertexArray(bunny->vao_id);
glDrawArrays(GL_TRIANGLES, 0, bunny->vertex.size());
glFinish();
```

# OpenGL : depth shadow map

## Rendu depuis la camera

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glUseProgram(program);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, depth_texture);
glGenerateMipmap(GL_TEXTURE_2D);
glBindVertexArray(bunny->vao_id);
glDrawArrays(GL_TRIANGLES, 0, bunny->vertex.size());
glFinish();
```

# OpenGL : depth shadow map

## Vertex shader

```
#version 450

layout(location = 1) in vec3 vPosition;
layout(location = 2) in vec3 vNormal;

uniform mat4 model_view_matrix;
uniform mat4 projection_matrix;

uniform vec3 light_position;
uniform mat4 light_model_view_matrix;
uniform mat4 light_projection_matrix;

const mat4 scale_bias_matrix = mat4(vec4(0.5f, 0.0f, 0.0f, 0.0f),
                                     vec4(0.0f, 0.5f, 0.0f, 0.0f),
                                     vec4(0.0f, 0.0f, 0.5f, 0.0f),
                                     vec4(0.5f, 0.5f, 0.5f, 1.0f));

(...)

void main() {
    gl_Position = projection_matrix * model_view_matrix * vec4(vPosition, 1.0f);
    point_in_light = light_projection_matrix * light_model_view_matrix * vec4(vPosition, 1.0f);
    light_dir = normalize(light_position - vPosition);

    point_in_light = point_in_light / point_in_light.w;
    point_in_light = scale_bias_matrix * point_in_light;
    (...)
}
```

# OpenGL : depth shadow map

## *Fragment shader*

```
#version 450

in vec3 color;
in vec3 normal;
in vec3 light_dir;
in vec4 point_in_light;

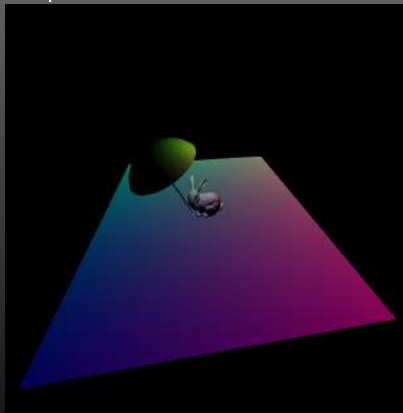
layout(location=0) out vec4 output_comb;

uniform sampler2D depth_texture;

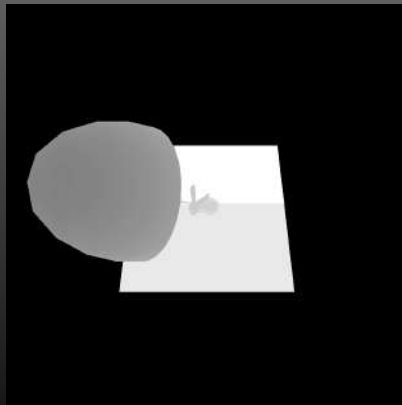
void main()
{
    vec2 textCoor = vec2(
        clamp(point_in_light.x, 0.0f, 1.0f),
        clamp(point_in_light.y, 0.0f, 1.0f)
    );
    float shadow = texture(depth_texture, textCoor).r;
    float shadow1;
    if ((point_in_light.z) > shadow)
        shadow1 = 0.5f;
    else
        shadow1 = 1.0f;
    output_comb = vec4(color, 1.0) * shadow1 * clamp(dot(light_dir, normal), 0.0,
```

# OpenGL : depth shadow map

## Etape 1



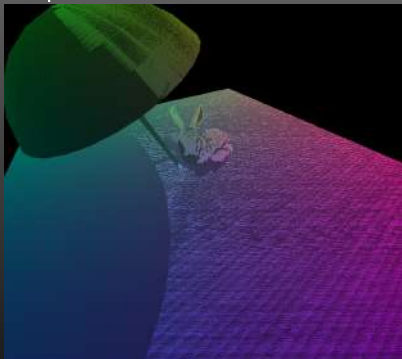
Scene



Rendu depuis la source lumineuse

# OpenGL : depth shadow map

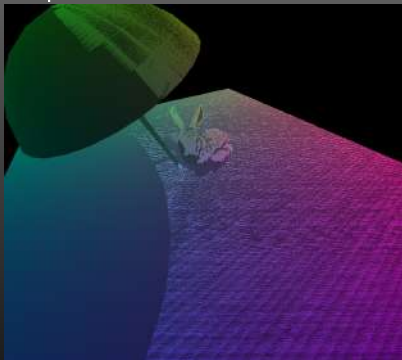
Etape 2



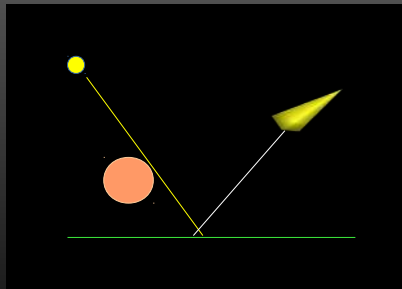
Résultat

# OpenGL : depth shadow map

Etape 2



Résultat





# OpenGL : depth shadow map

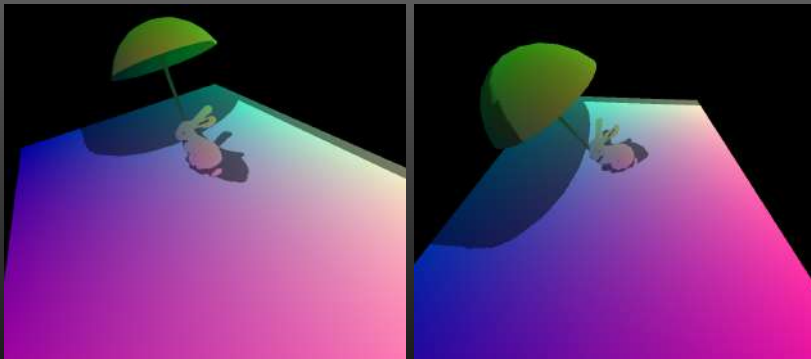
Ajout d'un biais :

```
glUseProgram(light_program);  
glBindVertexArray(bunny->vao_id);  
glEnable(GL_POLYGON_OFFSET_FILL);  
glPolygonOffset(1.0f, 1.0f);  
glDrawArrays(GL_TRIANGLES, 0, bunny->vertex.size());  
glDisable(GL_POLYGON_OFFSET_FILL);
```

Quelle valeur pour l'offset ?

# OpenGL : depth shadow map

Résultat

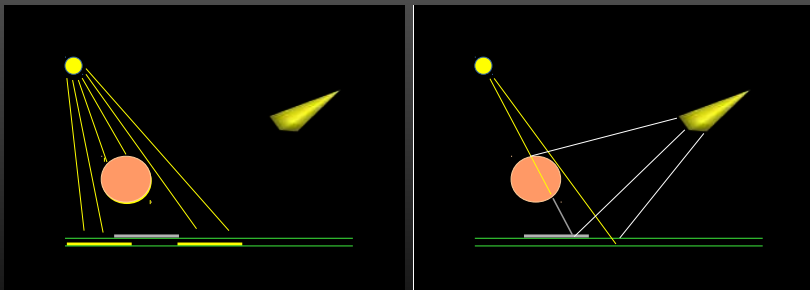


► *second depth shadow map*

# OpenGL : les ombres

## ► *second depth shadow map*

Idée : Faire le rendu de la scène depuis la lumière en regardant les faces arrières des objets

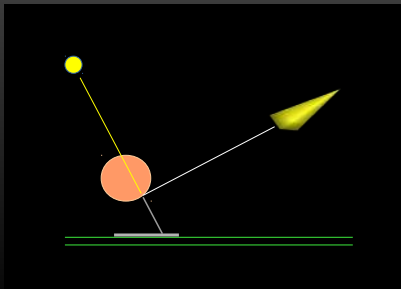


# OpenGL : les ombres

## ► *second depth shadow map*

Idée : Faire le rendu de la scène depuis la lumière en regardant les faces arrières des objets. Le problème du biais n'est plus un problème :

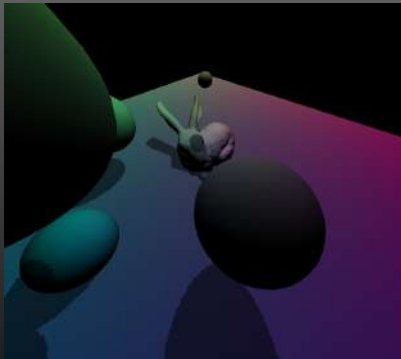
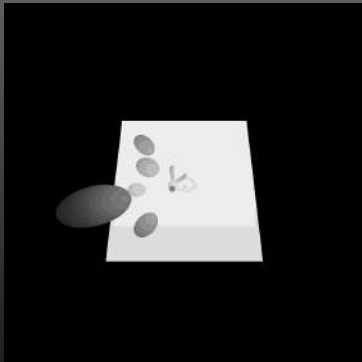
- Il est plus facile à fixer
- Il est même inutile si on considère la face arrière comme non éclairée



# OpenGL : les ombres

```
glEnable(GL_CULL_FACE);  
glFrontFace(GL_CCW);  
//Rendering from light source point of view  
...  
glFrontFace(GL_CW);  
//Rendering from camera point of view  
...
```

# OpenGL : les ombres



# OpenGL : les ombres

Pour aller plus loin :

- ▶ `sampler2DShadow/textureProj()` ;
- ▶ soft shadow map
- ▶ ...



# OpenGL : flou de bougé

# OpenGL : flou de bougé

- ▶ *Post processing*
- ▶ *Frame buffer Object*
- ▶ ...

# OpenGL : flou de bougé

- ▶ *Post processing*
  - ▶ *Frame buffer Object*
  - ▶ ...
- 
- ▶ *Accumulation buffer*

# OpenGL : flou de bougé

- ▶ *Post processing*
- ▶ *Frame buffer Object*
- ▶ ...

▶ *Accumulation buffer*

- ▶ Post processing

# OpenGL : flou de bougé

- ▶ Post processing
  - ▶ Rendu dans une texture
  - ▶ Affichage et traitement de la texture

- ▶ Post processing
  - ▶ Rendu dans une texture
  - ▶ Affichage et traitement de la texture
  - ▶ Problèmes des objets en mouvement/pas en mouvement

# OpenGL : flou de bougé

- ▶ Rendu d'une position dans une texture/un render buffer
- ▶ Accumulation dans une texture (Type *float* sinon les valeurs sont *clampées*)

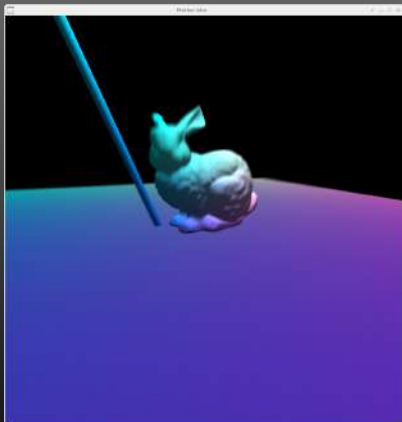
```
glBlendEquationSeparate(GL_FUNC_ADD, GL_FUNC_ADD);  
glBlendFunc(GL_ONE, GL_ONE);  
glDisable(GL_DITHER);
```

- ▶ Copies dans l'image finale (rendu ou glBlitFramebuffer)

On ne peut pas faire le rendu direct dans l'accumulateur à cause du *blending*.



# OpenGL : flou de bougé



# OpenGL : *Post processing*

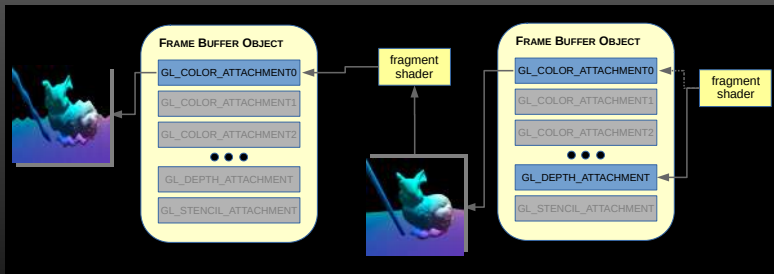
- ▶ Rendu dans une texture

# OpenGL : *Post processing*

- ▶ Rendu dans une texture
- ▶ Affichage et traitement de la texture

# OpenGL : *Post processing*

- ▶ Rendu dans une texture
- ▶ Affichage et traitement de la texture



# OpenGL : *Post processing*

## Rendu dans une texture

```
glGenTextures(1, &(render_buffers[output_buffer]));
glGenRenderbuffers(1, &(render_buffers[depth_buffer]));

glBindTexture(GL_TEXTURE_2D, render_buffers[output_buffer]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1024, 1024, 0, GL_RGB, GL_UNSIGNED_BYTE);
// Attention à l'interpolation
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glBindTexture(GL_TEXTURE_2D, 0);

glBindRenderbuffer(GL_RENDERBUFFER, render_buffers[depth_buffer]);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 1024, 1024);

glGenFramebuffers(1, &temp_frame_buffer_object_id);
glBindFramebuffer(GL_FRAMEBUFFER, temp_frame_buffer_object_id);

glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, render_buffers[output_buffer], 0);
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, render_buffers[depth_buffer]);
glDrawBuffer(GL_COLOR_ATTACHMENT0);
```

## Rendu dans une texture

```
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, temp_frame_buffer_object_id);  
//draw the scene  
...  
  
//Link the texture  
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, render_buffers[output_buffer]);  
glGenerateMipmap(GL_TEXTURE_2D);  
texID = glGetUniformLocation(quad_programID->program_id, "renderedTexture");  
glUniform1i(texID, 0);
```

# OpenGL : *Post processing*

## *Fragment shader*

```
#version 400

in vec2 UV;
uniform sampler2D renderedTexture;

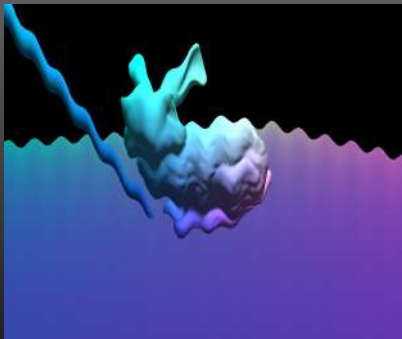
out vec4 color;

//Wave deformation
float f(float x, float freq) {
    return 0.01*cos(freq*x);
}

void main(){
    color = texture(renderedTexture, vec2(UV.x, UV.y+f(UV.x,100.0)));
}
```



# OpenGL : *Post processing*



# OpenGL : *Object Picking*

# OpenGL : le brouillard

- ▶ Définition des paramètres :
  - ▶ `glFogi(GL_FOG_MODE, GL_LINEAR);`
  - ▶ `glFogf(GL_FOG_START, 0.1);`
  - ▶ `glFogf(GL_FOG_END, 2000);`
  - ▶ `glFogfv(GL_FOG_COLOR, fog_color);`
- ▶ Activation
  - ▶ `glEnable(GL_FOG);`

# OpenGL : le brouillard

## ► Définition des paramètres :

- `glFogi(GL_FOG_MODE, GL_LINEAR);`
- `glFogf(GL_FOG_START, 0.1);`
- `glFogf(GL_FOG_END, 2000);`
- `glFogfv(GL_FOG_COLOR, fog_color);`

## ► Activation

- `glEnable(GL_FOG);`

# OpenGL : le brouillard

## ► Définition des paramètres :

- `glFogi(GL_FOG_MODE, GL_LINEAR);`
- `glFogf(GL_FOG_START, 0.1);`
- `glFogf(GL_FOG_END, 2000);`
- `glFogfv(GL_FOG_COLOR, fog_color);`

## ► Activation

- `glEnable(GL_FOG);`

# OpenGL : le brouillard

- ▶ Objectif : modifier la couleur en fonction de la distance.
  - ▶ Choisir la fonction...

# OpenGL : le brouillard

Exemple :

```
#version 450
```

```
in vec3 color;  
in vec3 normal;  
in vec3 light_dir;
```

```
vec4 fog_color = vec4(1.0,1.0,1.0,1.0);  
layout(location=0) out vec4 output_comb;
```

```
float get_fog_factor(float d) {  
    const float FOG_MIN = 0.8;  
    const float FOG_MAX = 1.0;  
  
    if (d >= FOG_MAX) return 1.0;  
    if (d <= FOG_MIN) return 0.0;  
  
    return 1 - (FOG_MAX - d) / (FOG_MAX - FOG_MIN);  
}
```

```
void main()  
{  
    float fog = get_fog_factor(gl_FragCoord.z);  
    output_comb = mix(vec4(color, 1.0)*(clamp(dot(light_dir, normal), 0.0, 1.0)),  
                      fog_color, fog);  
}
```



## Systèmes à base de particules

- ▶ Permet de modéliser des éléments difficiles à modéliser avec des solides classiques ou des surfaces
  - ▶ Feu,
  - ▶ fumée/poussière,
  - ▶ spark, pluie
  - ▶ ...



# Moteur de particules

## Fonctionnement

- ▶ Caractéristiques
  - ▶ position
  - ▶ couleur
  - ▶ taille
  - ▶ forme
  - ▶ ...
- ▶ Lois
  - ▶ création
  - ▶ destruction
- ▶ Règles (évolution)
  - ▶ Modifications des caractéristiques

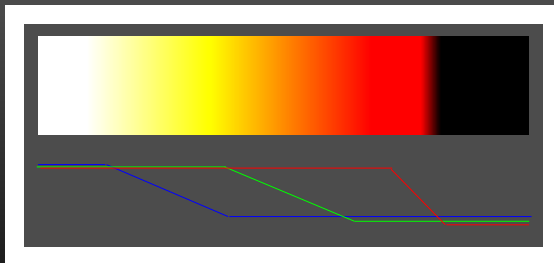
## Utilisation de particules

- ▶ Création
  - ▶ Plusieurs centaines
  - ▶ Apparaissent dans une zone précise
  - ▶ Avec une couleur proche du blanc
- ▶ Forme
  - ▶ Point
  - ▶ Sphere
- ▶ Evolution
  - ▶ Changement de couleur
  - ▶ Déplacement vers le haut avec perturbations
- ▶ Destruction
  - ▶ Atteint la couleur noir

# Le feu

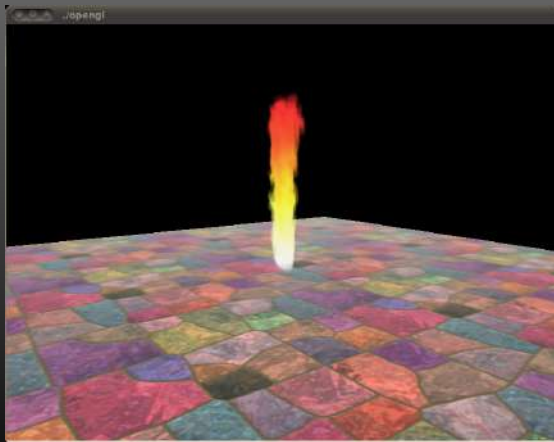
## Utilisation de particules

- ▶ Evolution
- ▶ Changement de couleur



# Le feu

## Résultat



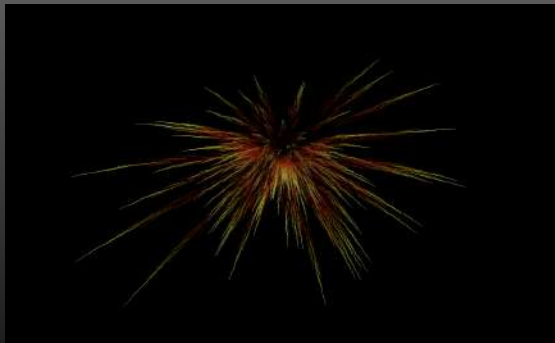
Flou de bougé

Utilisation d'un Billboard.  
Affichage d'un feu en 2D.

# étincèles/feu d'artifice/explosion

# étincèles/feu d'artifice/explosion

Résultat



# OpenGL : Moteur de particules

## Point sprite

```
glEnable(GL_PROGRAM_POINT_SIZE);
```

## *Vertex shader*

```
vec4 pos = projection_matrix * model_view_matrix * vPosition;  
gl_PointSize = size * (1.0 - pos.z / pos.w) * 32.0;  
gl_Position = pos;
```

## *Fragment shader*

```
vec2 temp = gl_PointCoord - vec2(0.5);  
float f = dot(temp, temp);  
if (f > 0.25)  
    discard;  
output_color = color;
```



# OpenGL : Moteur de particules

Point sprite

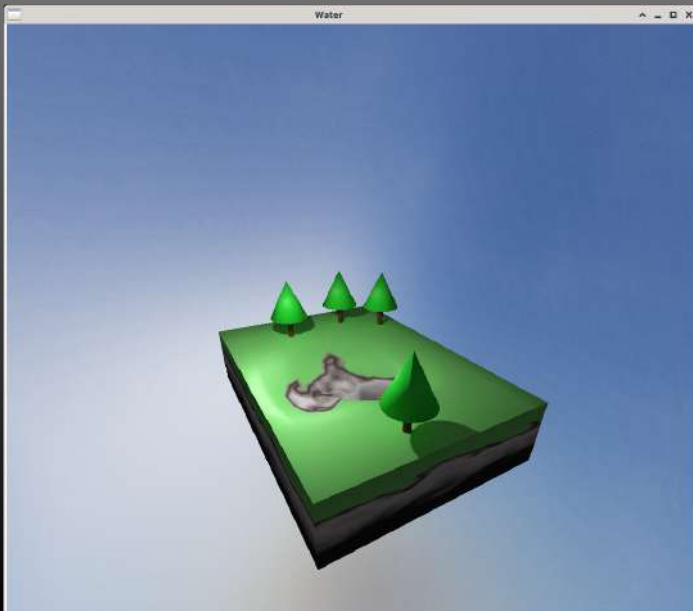


Utilisation d'un Billboard :

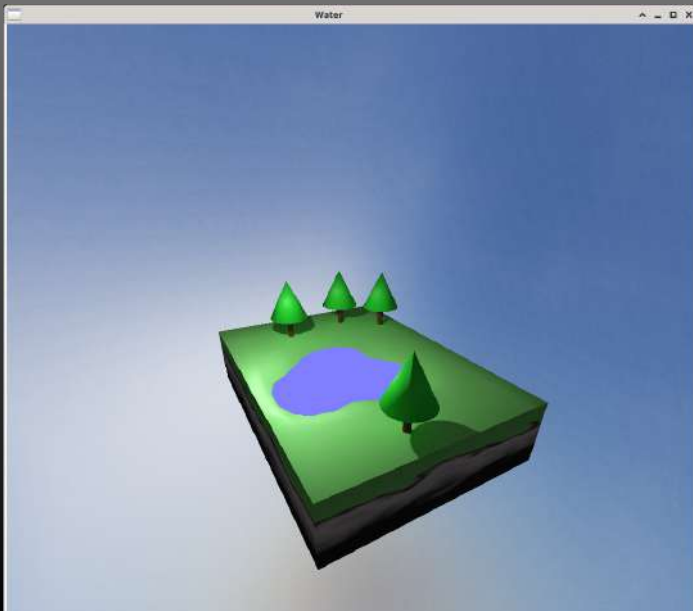
- ▶ Affichage d'un feu en 2D.
- ▶ fumée
- ▶ ...

# L'eau

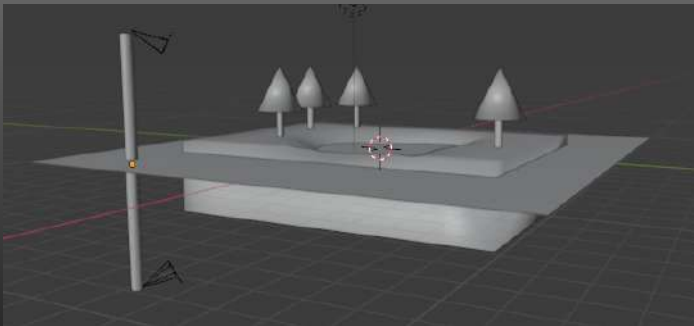
# L'eau



# L'eau

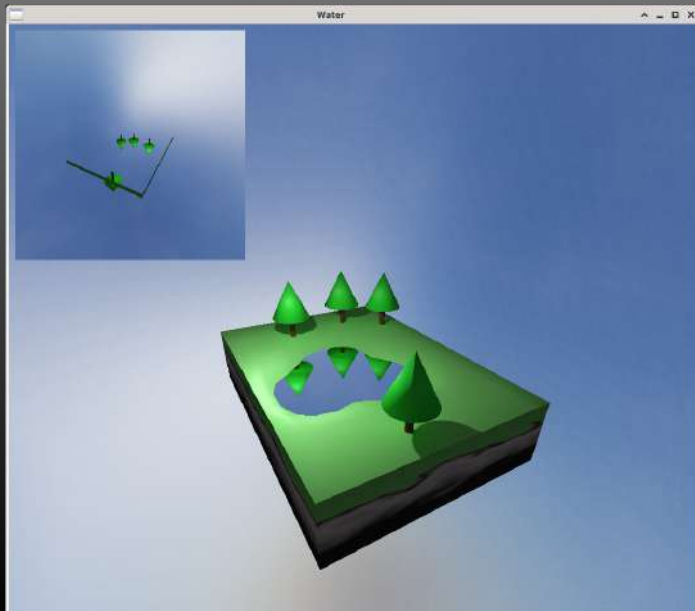


# L'eau

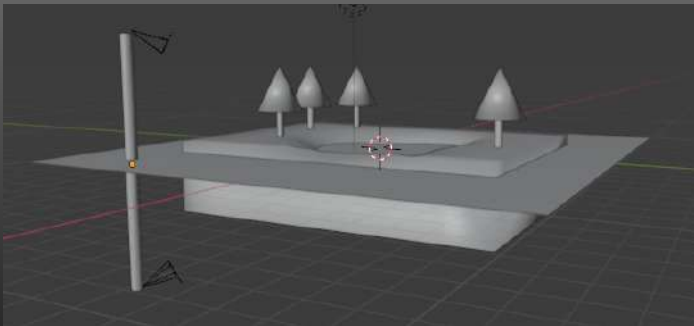


- ▶ `glEnable(GL_CLIP_PLANE0);`
- ▶ `gl_ClipDistance[0] = position.y;`

# L'eau



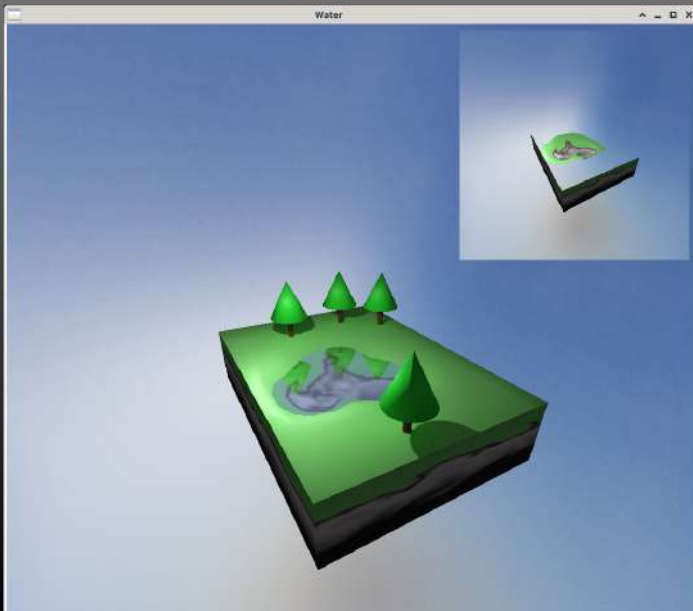
# L'eau



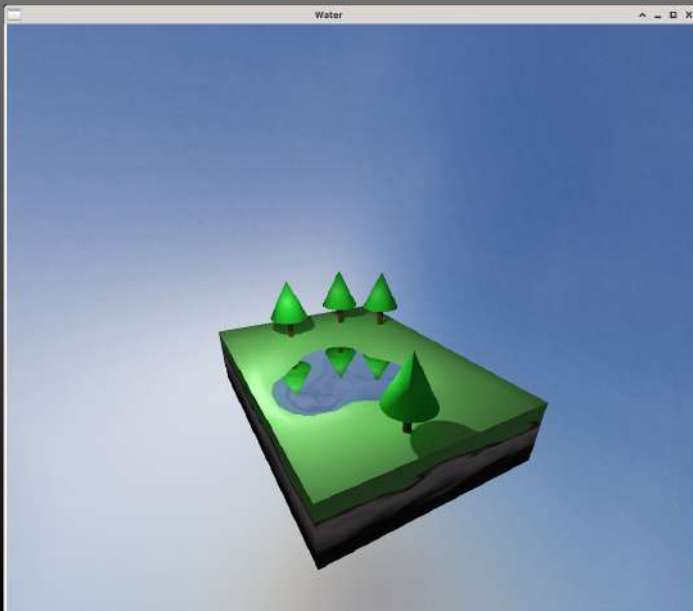
- ▶ `glEnable(GL_CLIP_PLANE1);`
- ▶ `gl_ClipDistance[1] = -position.y+0.4;`



# L'eau



# L'eau



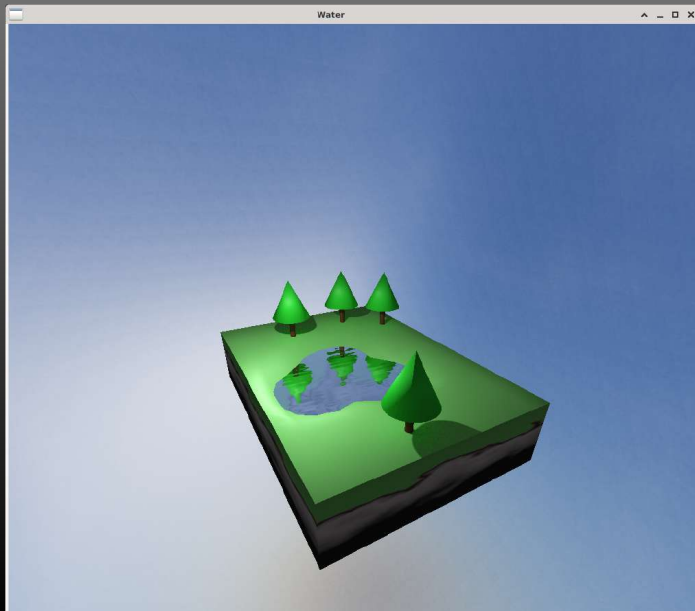
# L'eau



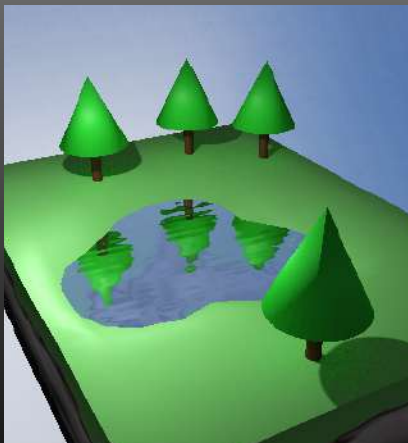
## Fresnel

- ▶ Schlick approximation
- ▶  $\text{dot}(\text{normal}, \text{view})^\alpha$

# L'eau

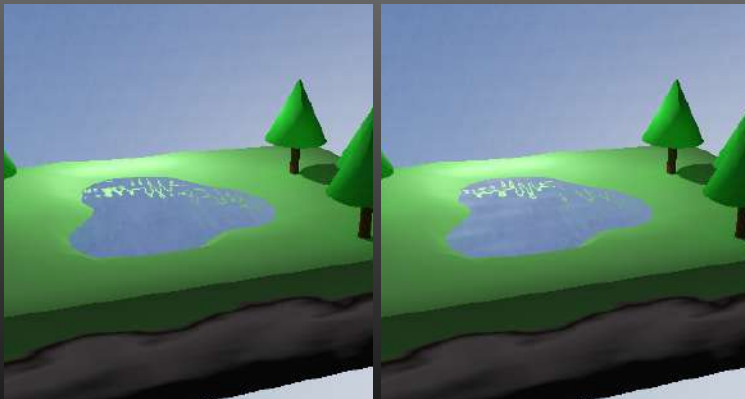


# L'eau



Utilisation de Fourier : combinaison de 6 sinusoides

# L'eau



Ajout de reflets de la lumière

## Améliorations

- ▶ Utiliser la profondeur pour assombrir l'eau
- ▶ Utiliser un rendu pour le rendu de la refraction et du rendu final (comme dans Far Cry) et donc gagner du temps

# Conclusion