Rendu

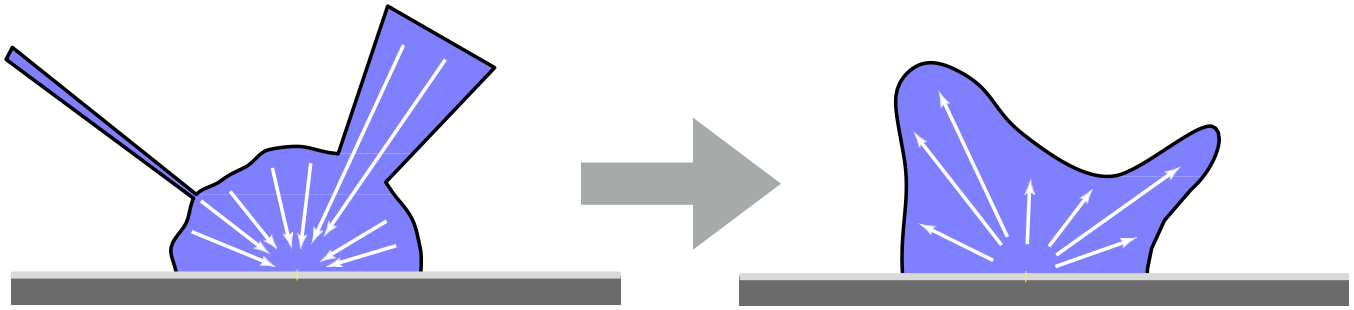
Rendu différé, ombrage

Source cours de Steve Marschner

Rendu différé

Réflexion de la lumière

- Toutes les type de réflexion reflètent tous les types d'illumination
  - Diffuse, brillante, miroir
  - Environnement, lumières ponctuelles ou étendues



distribution incidente
(fonction de direction)

Distribution réfléchie
(fonction de direction)

Catégories d'illumination

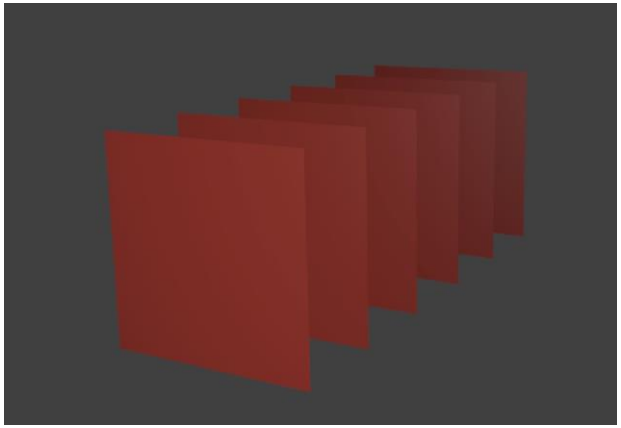
	Diffuse	Glossy	Mirror
indirect	Soft indirect shadows	blurry reflections of other objects	reflected images of other objects
environment	soft shadows	blurry reflection of environment	reflected image of environment
area	soft shadows	shaped specular highlight	reflected image of source
point/directional	hard shadows	simple specular highlight	point reflections

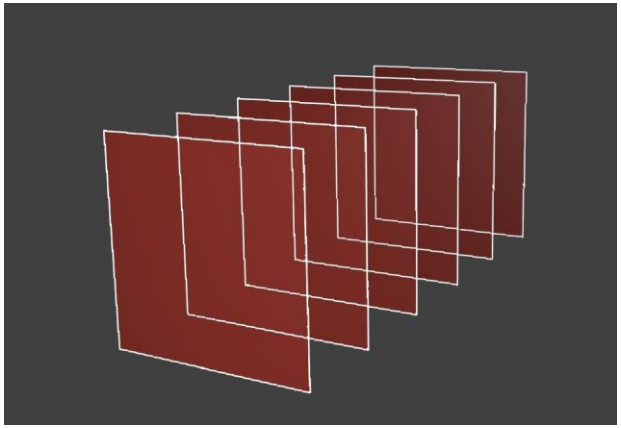
= easy to compute using standard shaders

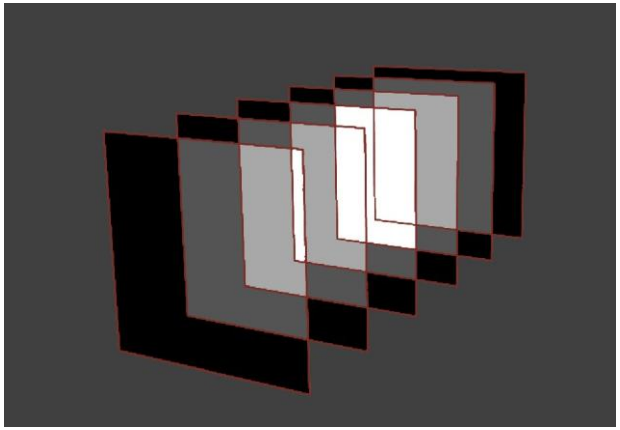
Rendu : forward shading (ombrage direct)

```
for each object in the scene
   for each triangle in this object
      for each fragment f in this triangle

         gl_FragColor = shade(f)
         if (depth of f < depthbuffer[x, y])
            framebuffer[x, y] = gl_FragColor
            depthbuffer[x, y] = depth of f
         end if

      end for
   end for
end for
```

Problème : Overdraw







Problème : complexité de la lumière

Just Cause 3 | Avalanche Studios

Traitements spatiaux complexes

Les fragments ne peuvent pas se parler

- Une contrainte fondamentale de performance GPU

Effets intéressants dépendent du voisinage et de la géométrie

- Bloom
- Occlusion ambiante
- Flou de mouvement
- Profondeur de champ
- Effets basés sur les arêtes

Rendu différé

Première passe de rendu

- Dessiner toute la géométrie
- Calculer les entrées (matériaux, géométrie) pour le modèle d ’éclairages
- Ne pas calculer l’ombrage
- Ecrire les entrées de l’ombrage dans un buffer intermédiaire

Deuxième passe

- Ne pas dessiner la géométrie
- Utiliser les entrées stockées pour calculer l’ombrage
- Écrire la sortie

Passe de post-traitement (optionnelle, peut aussi être utilisé en rendu direct

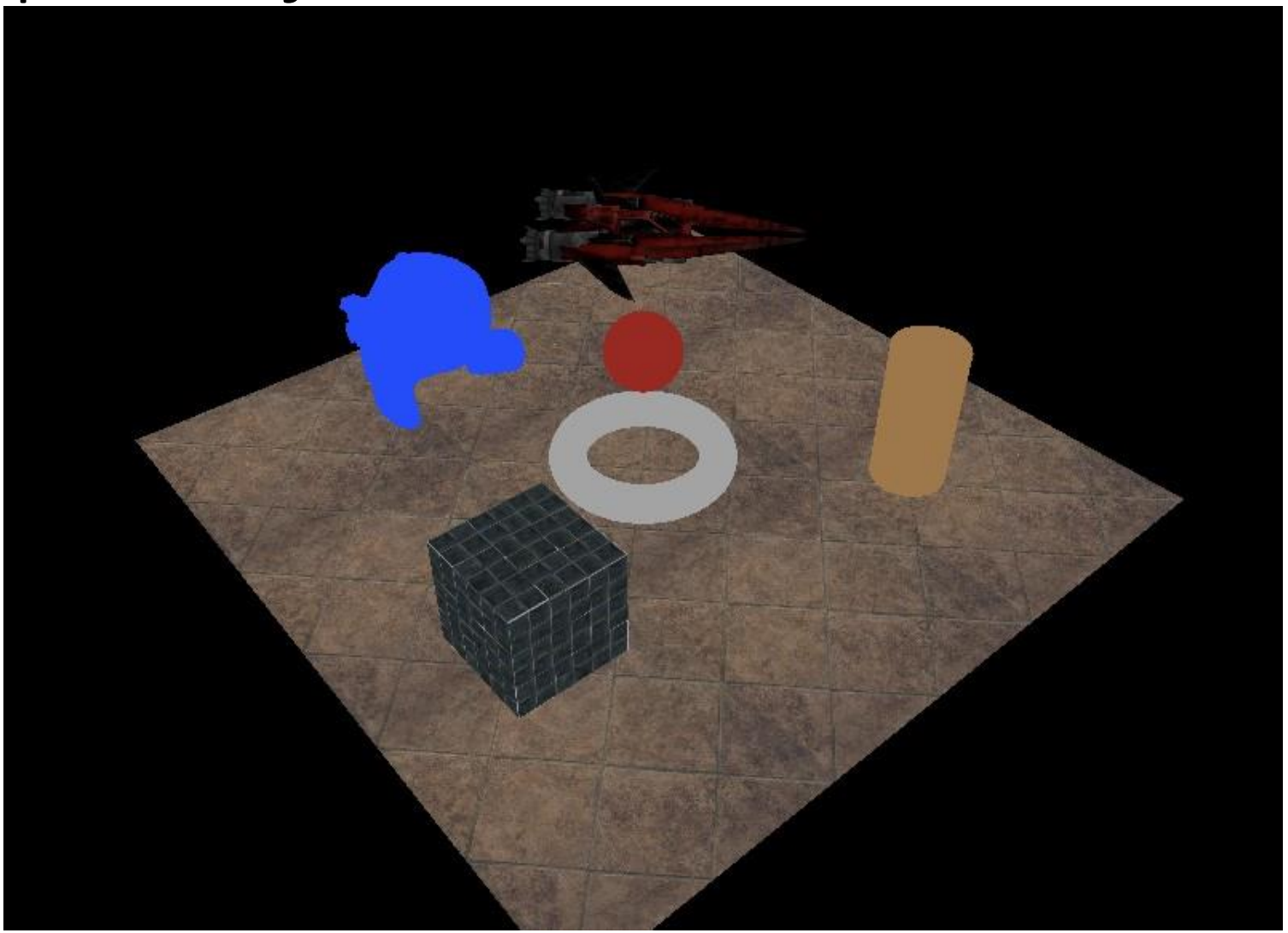
- Traiter l’image finale pour produire les pixels de sortie

Rendu différé : étape 1

```
for each object in the scene
   for each triangle in this object
      for each fragment f in this triangle

         gl_FragData[…] = material properties of f
         if (depth of f < depthbuffer[x, y])
            gbuffer[…][x, y] = gl_FragData[…]
            depthbuffer[x, y] = depth of f
         end if

      end for
   end for
end for
```

Utilisation de l'option "Multiple Render Targets" d'OpenGL dans laquelle gl_FragColor est remplacée par une liste de valeurs chacune écrites dans un buffer différent

1ère passe : juste les matériaux en sortie

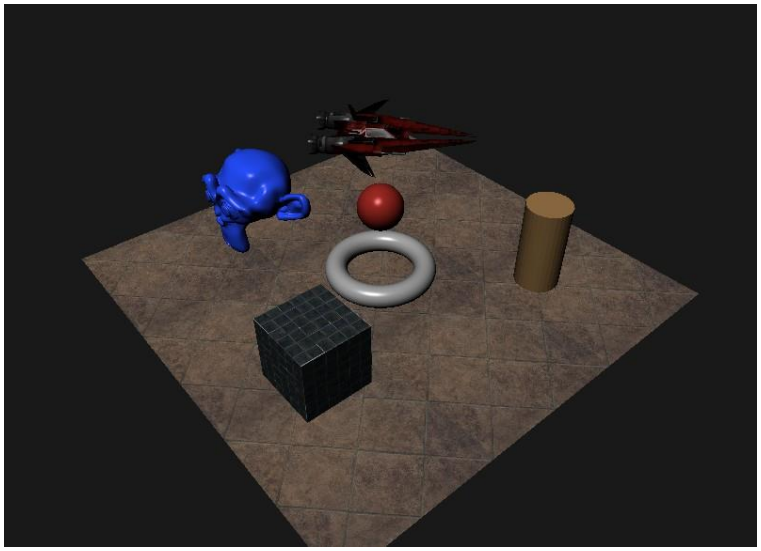


Rendu différé : étape 2

```
for each fragment f in the gbuffer
   framebuffer[x, y] = shade (f)
end for
```

Amélioration clé : shade (f) seulement exécutée pour les fragments visibles

La sortie est la même→

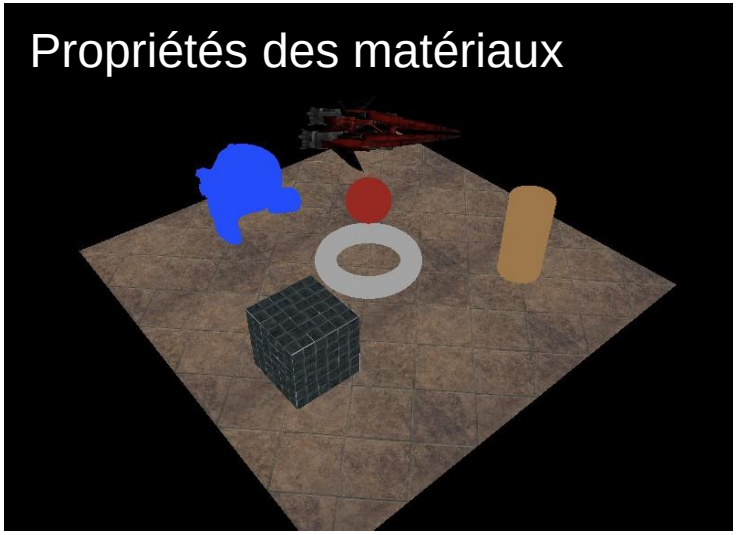


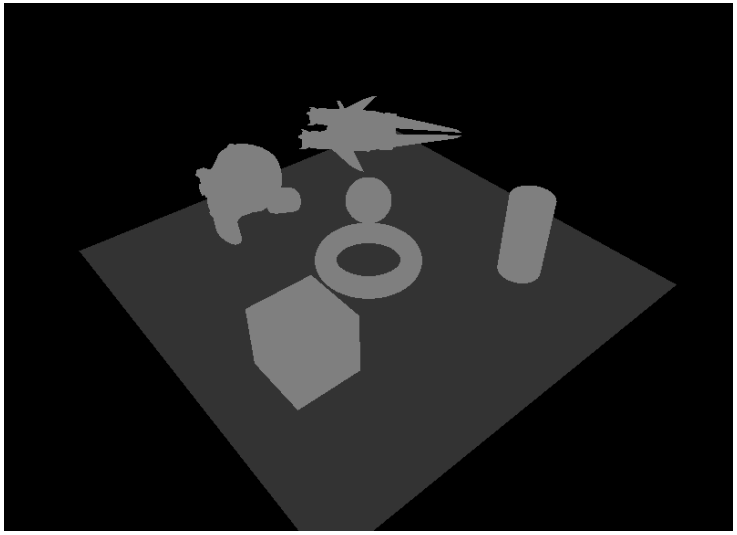
```
for each object in the scene
   for each triangle in this object
      for each fragment f in this triangle

         gl_FragData[…] = material properties of f
         if (depth of f < depthbuffer[x, y])
            gbuffer[…][x, y] = gl_FragData[…]
            depthbuffer[x, y] = depth of f
         end if

      end for
   end for
end for
```

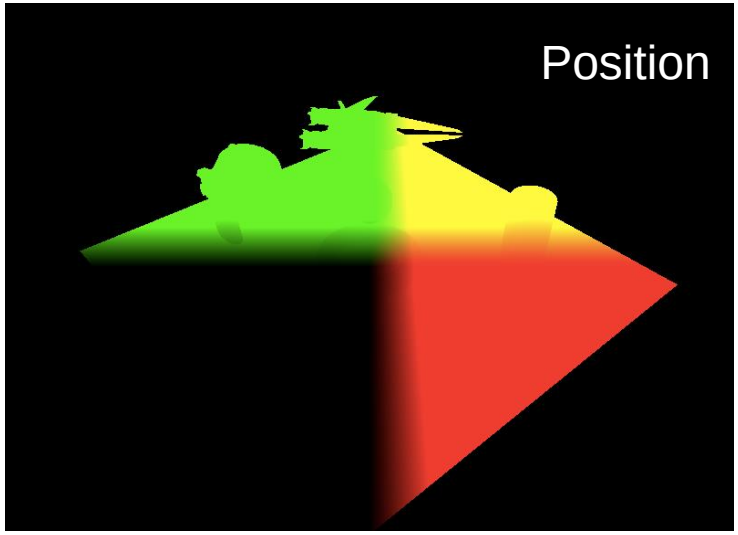
```
for each fragment f in the gbuffer
   framebuffer[x, y] = shade (f)
end for
```

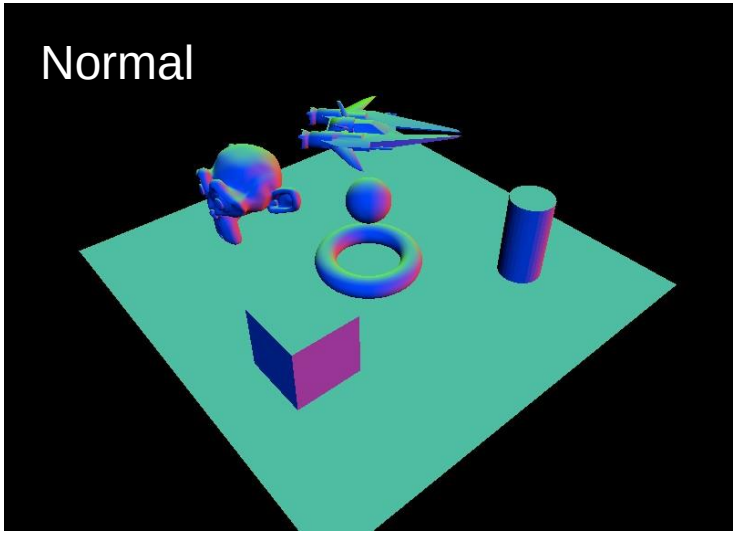
Utilisation de l'option "Multiple Render Targets" d'OpenGL dans laquelle gl_FragColor est remplacée par une liste de valeurs chacune écrites dans un buffer différent

G-buffer : textures multiples









Le super-shader

Shader calculant l'éclairage avec le g-buffer : code pour tous les modèles de matériaux/lumière en un seul grand shader.

```
shade (f) {
    result = 0;
    if (f is Lambertian) {
        for each light
            result += (n . l) * diffuse;
        end for
    } else if (f is Blinn-Phong) {
        ...
    } else if (f is ...) {
        ...
    }
    return result;
}
```

Entrées du super-shader

A besoin d'accéder à tous les paramètres de matériaux du fragment courant

- Blinn-Phong : kd, ks, n
- Microfacets : kd, ks, alpha
- Etc…

Aussi : position du fragment et normale à la surface

Solution : issus du shader de matériaux

```
{outputs}={f.material, f.position, f.normal}

if (depth of f < depthbuffer[x, y])

      gbuffer[x, y]       = {outputs}
      depthbuffer[x,  y] = depth of f
 end if
```

Rendu différé

Rendu en une passe : considérations de toutes les lumières pour chaque fragment

- Efforts gâchés car toutes ne contribuent pas
- Envoyé la géométrie en fonction de la lumière ne fonctionne pas

Rendu différé : fragments peuvent être visités en sous groupe

- Pour chaque lumière, définir des bornes de volumes (significativement) impactés par la lumière, calcul pour les fragments à l'intérieur (depth/stencil)

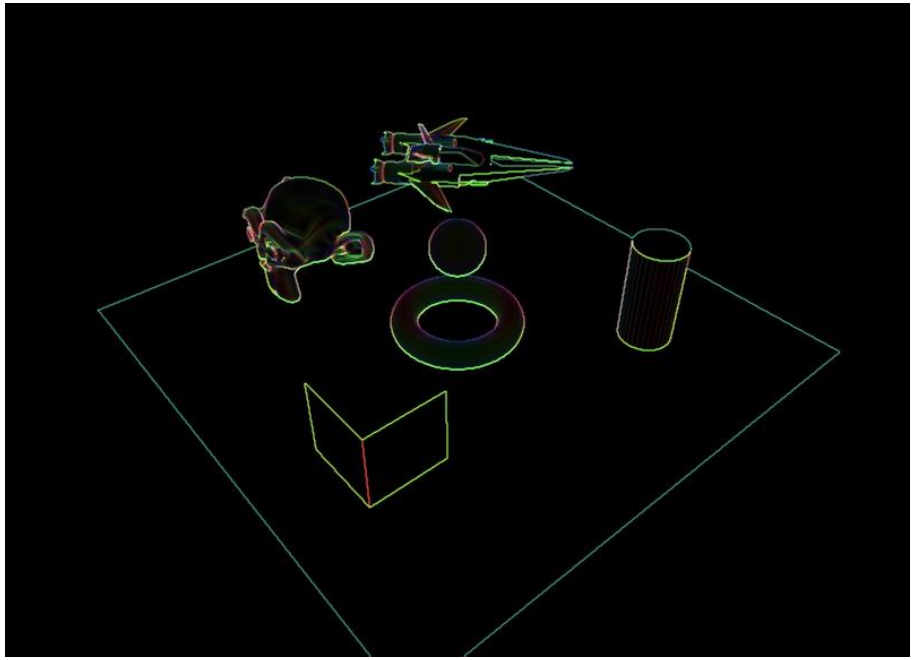
Puissance du rendu différé

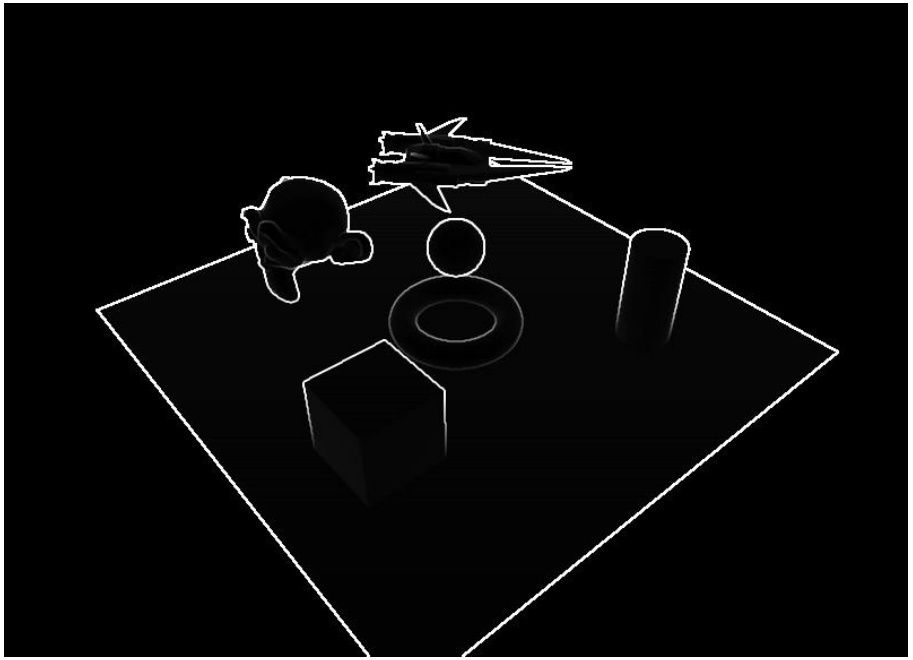
- Traitement d'images possible entre l'étape 1 et 2
  - étape 1 = remplir g-buffer
  - étape 2 = éclairer/ombrer
  - Ajouter de l'étapes 1,5 (filtrer g-buffer) possible

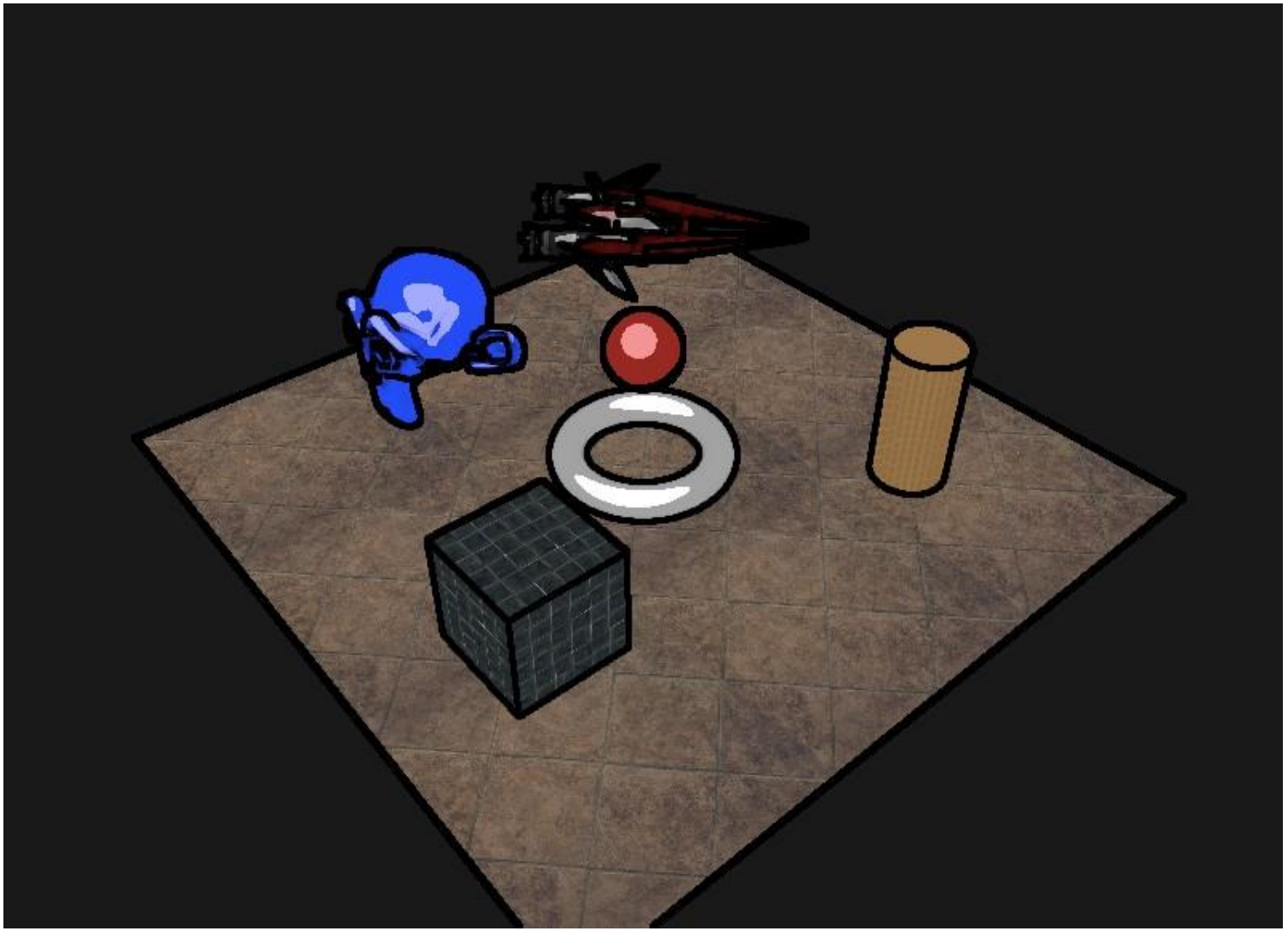
- Example
  - Détection de silhouette pour le rendu artistique
  - Occlusion ambiante dans l'espace écran
  - Débruitage et filtrage bilatéral en utilisant des informations géométriques

Détection de silhouettes

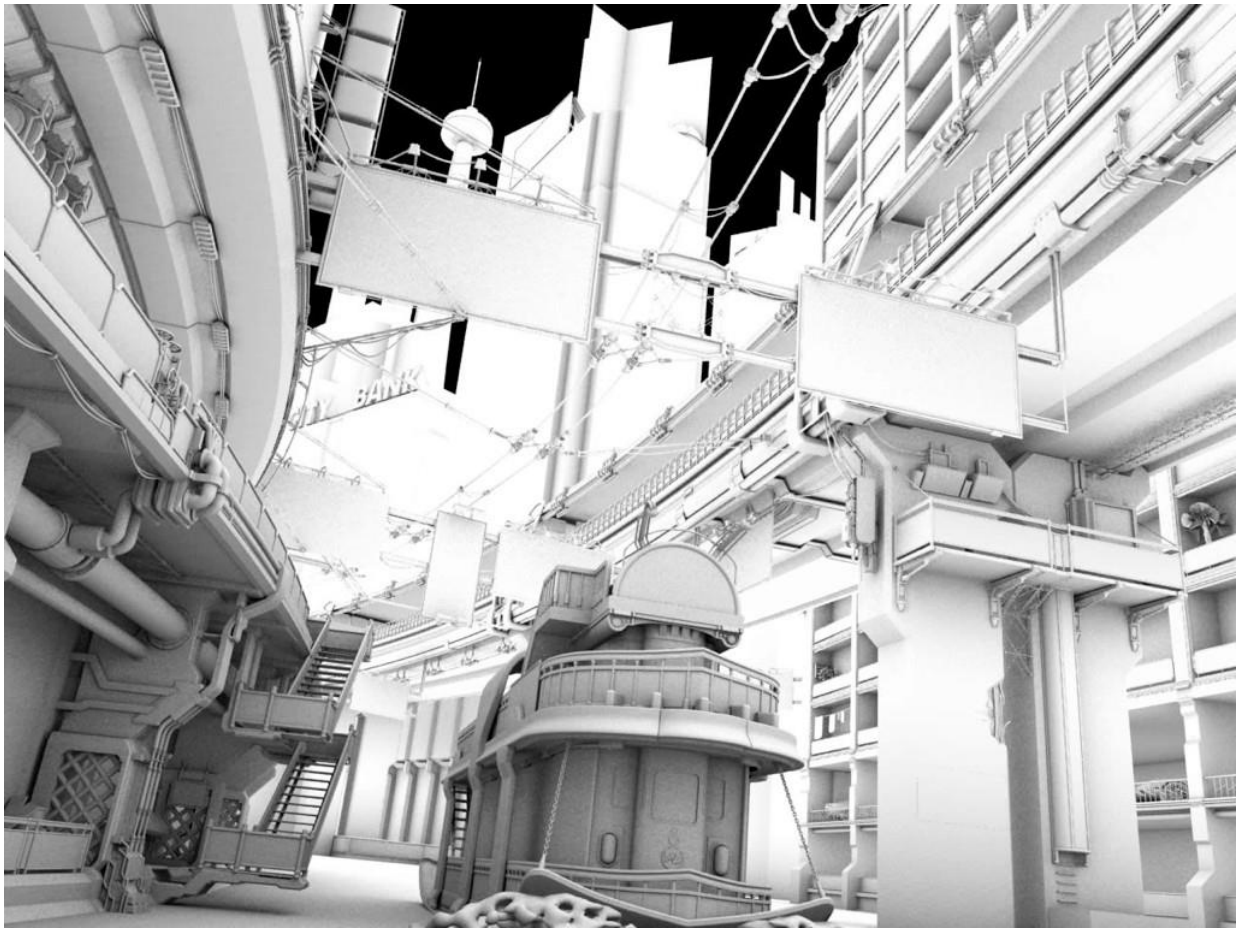
- En dérivant le depth buffer, on peut situer les silhouettes et pics







Occlusion ambiante



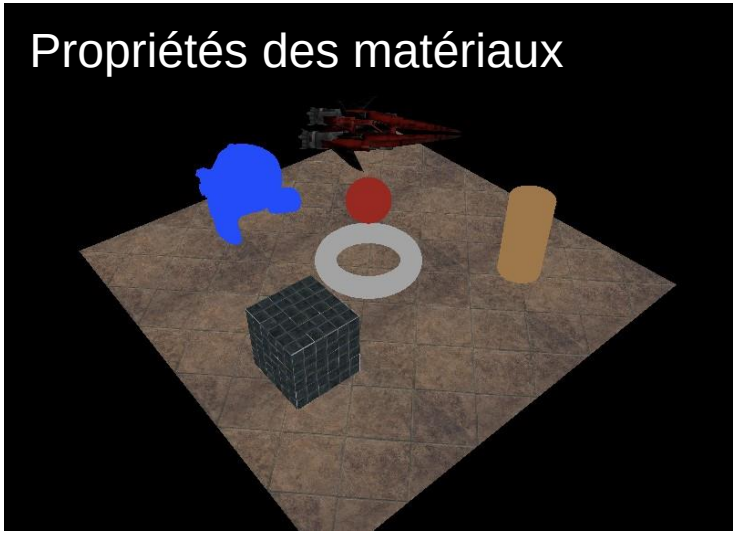
Débruitage

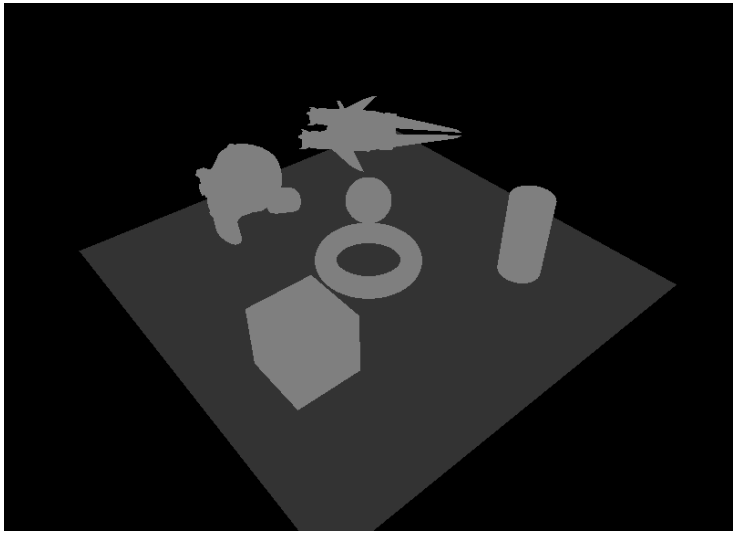
[Mara et al. HPG 2017]

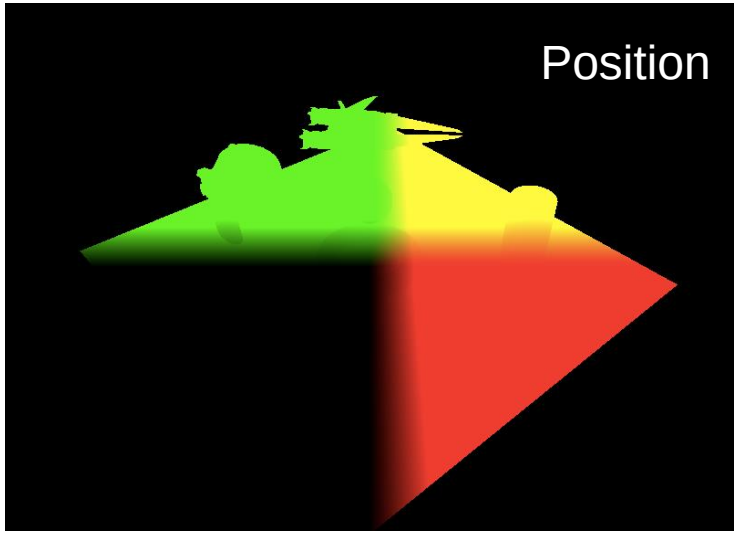
Débruitage

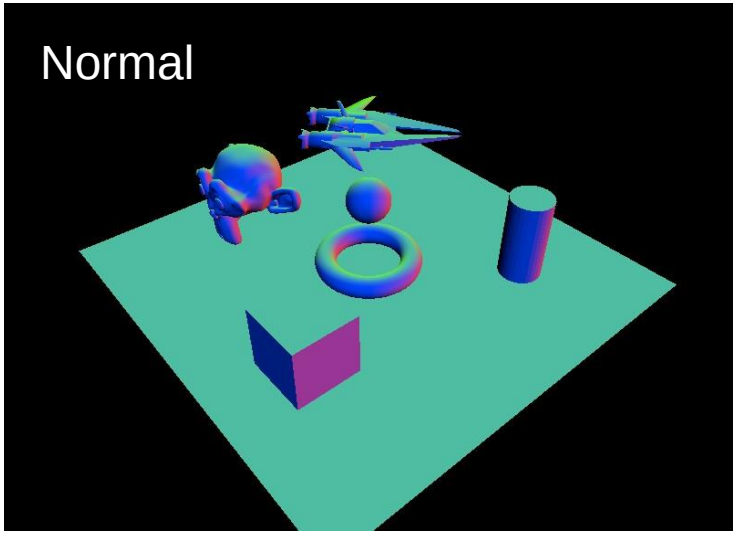
[Mara et al. HPG 2017]

Comment remplir les g-buffers ?









Limitations du rendu différé

Chaque pixel dans le g-buffer stocke les informations pour une seule surface

- Transparence (blending) est difficile
- Anti-alisasing est plus compliqué

Pour la transparence, méthode « hybride »

- Rendu différé pour les objets opaque et direct pour les translucides
- Objets transparents peuvent récupérer les informations des obj opaques derrière eux

Pour l'anti-aliasing flou intelligent

- Utilisation du g-buffer pour flouter les informations le long des arêtes (pas au travers)

Rendu hybride

Anti-aliasing

Un seul élément d’ombrage par pixel

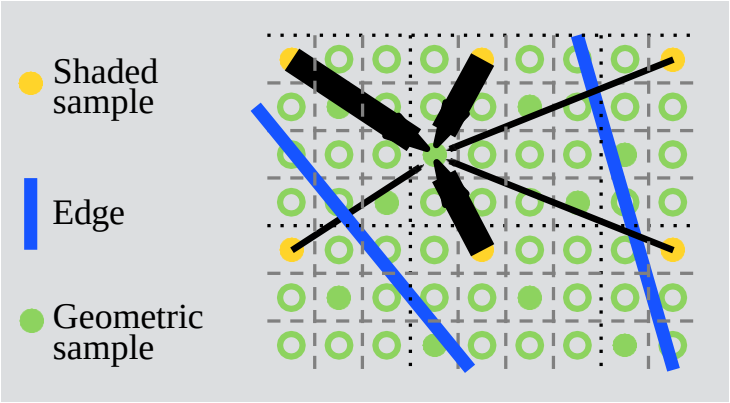
Reconstruction en mélangeant avec les éléments voisins

- Sélection en cherchant les arêtes (Morphological AA [Reshetov 09])
- Détecter les arêtes en utilisant la profondeur multi-échantillons (Subpixel Reconstruction AA [Chajdas et al. 11])

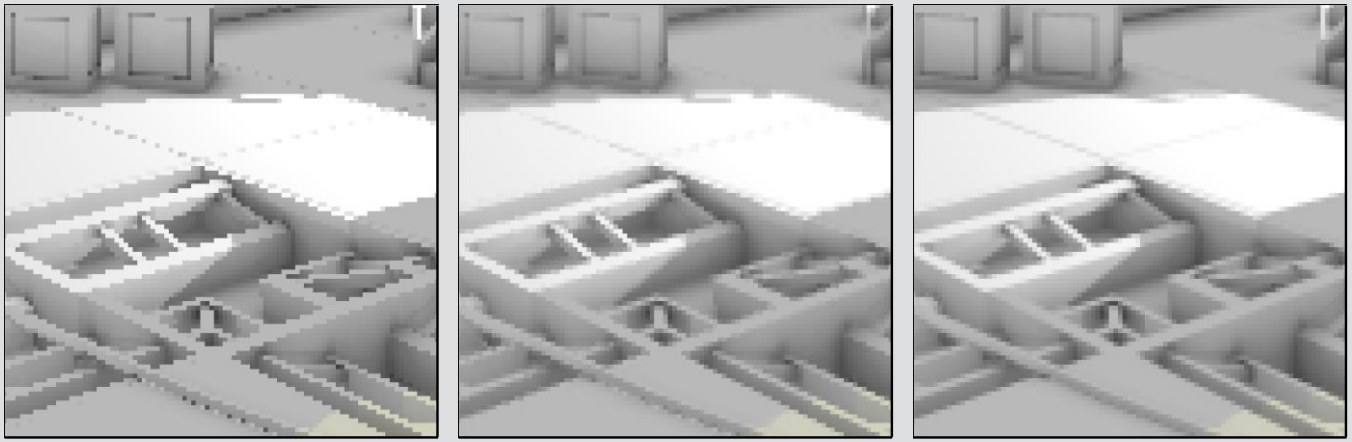
MLAA

[Reshetov 09]

SRAA



Similar Time ! *Similar Quality !*



(a) 1⇥ Shading + Box (poor, fast) (b) **NEW: 1⇥ Shading + SRAA (good, fast)** (c) 16⇥ Shading + Box (good, slow)

[Chajdas et al. 11]

Résumé : rendu différé

Pour

- Stockez tout ce qui est nécessaire à la passe 1
  - Normales, diffus, spéculaire, positions,…
  - G-buffer
- Après le z-buffer, on calcule l'ombrage seulement pour ce qui est visible

Contre

- Transparence (seulement un fragment pas pixel)
- Anti-aliasing (le AA multi-échantillons pas facile à adapter)

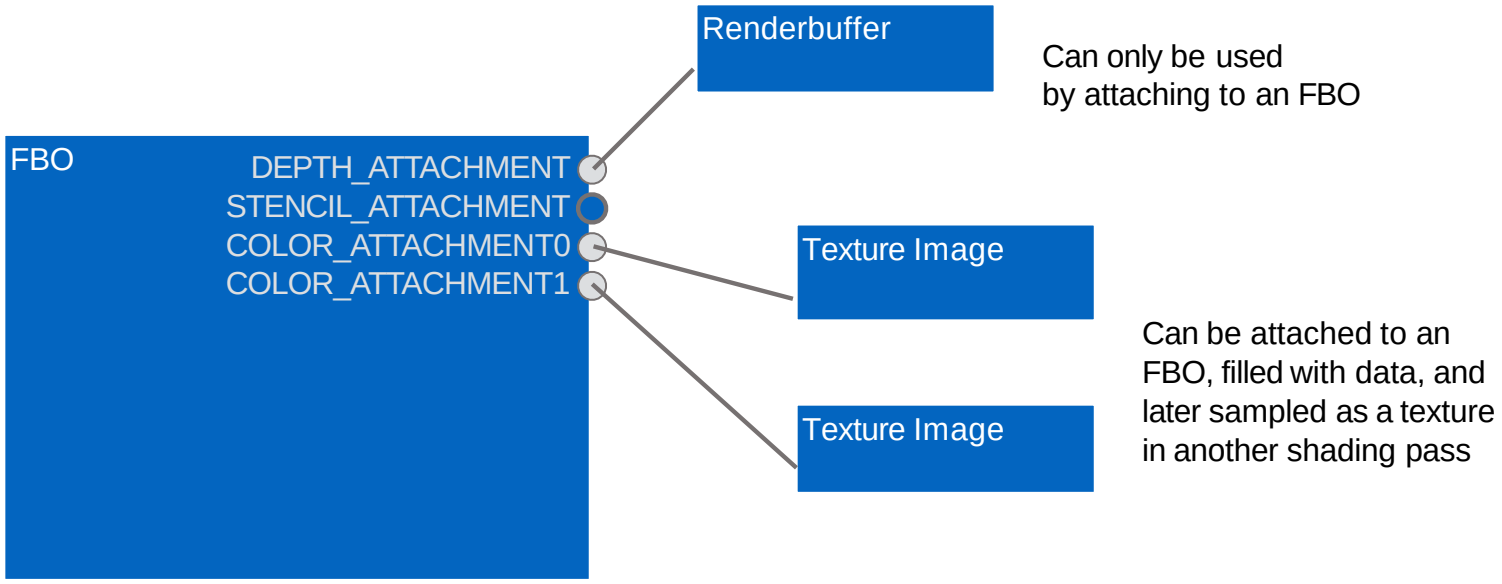
Les moteurs de jeux standards proposent le rendu direct ET différé

En OpenGL

- Par défaut en OpenGL, tous les fragments de sortie sont écrits dans le framebuffer affiché à l'écran
  - Buffer par défaut
  - Peut contenir plusieurs buffers : front and back pour le double buffering ; gauche et droite pour les appareils stereo/HMD.
  - Redirection de la sortie : glDrawBuffer()

- Rendu différé ou multi-pass, création d'un Framebuffer Object (FBO)
  - Attacher une image au FBO pour recevoir les la sortie du shader
  - La couleur (nombre variable) reçoit la donnée de couleur de gl_FragData […] (gl_FragColor alias de gl_FragData[0])
  - Profondeur requise pour le z-buffering des fonctions ;

Framebuffer Object



Framebuffer Object : C++

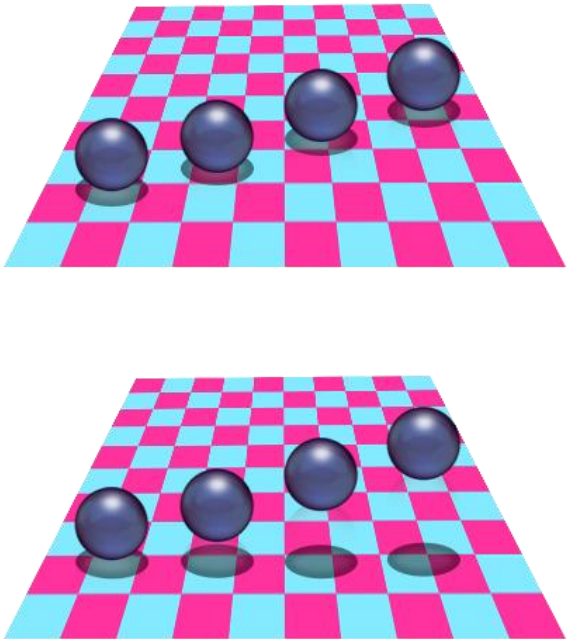
```
unsigned int gBuffer;
glGenFramebuffers(1, &gBuffer);
glBindFramebuffer(GL_FRAMEBUFFER, gBuffer);
unsigned int gPosition, gNormal, gColorSpec;

// - position color buffer
glGenTextures(1, &gPosition);
glBindTexture(GL_TEXTURE_2D, gPosition);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB16F, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGB, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, gPosition, 0);

// - normal color buffer
glGenTextures(1, &gNormal);
glBindTexture(GL_TEXTURE_2D, gNormal);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB16F, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGB, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT1, GL_TEXTURE_2D, gNormal, 0);

// - color + specular color buffer
glGenTextures(1, &gAlbedoSpec);
glBindTexture(GL_TEXTURE_2D, gAlbedoSpec);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT2, GL_TEXTURE_2D, gAlbedoSpec, 0);

// - tell OpenGL which color attachments we'll use (of this framebuffer) for rendering
unsigned int attachments[3] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1, GL_COLOR_ATTACHMENT2 };
glDrawBuffers(3, attachments);

// then also add render buffer object as depth buffer and check for completeness.
[...]
```

```
layout (location = 0) out vec3 gPosition;
layout (location = 1) out vec3 gNormal;
layout (location = 2) out vec4 gAlbedoSpec;
```

Ombres

https://www.realtimeshadows.com/sites/default/files/sig2013-course-softshadows.pdf

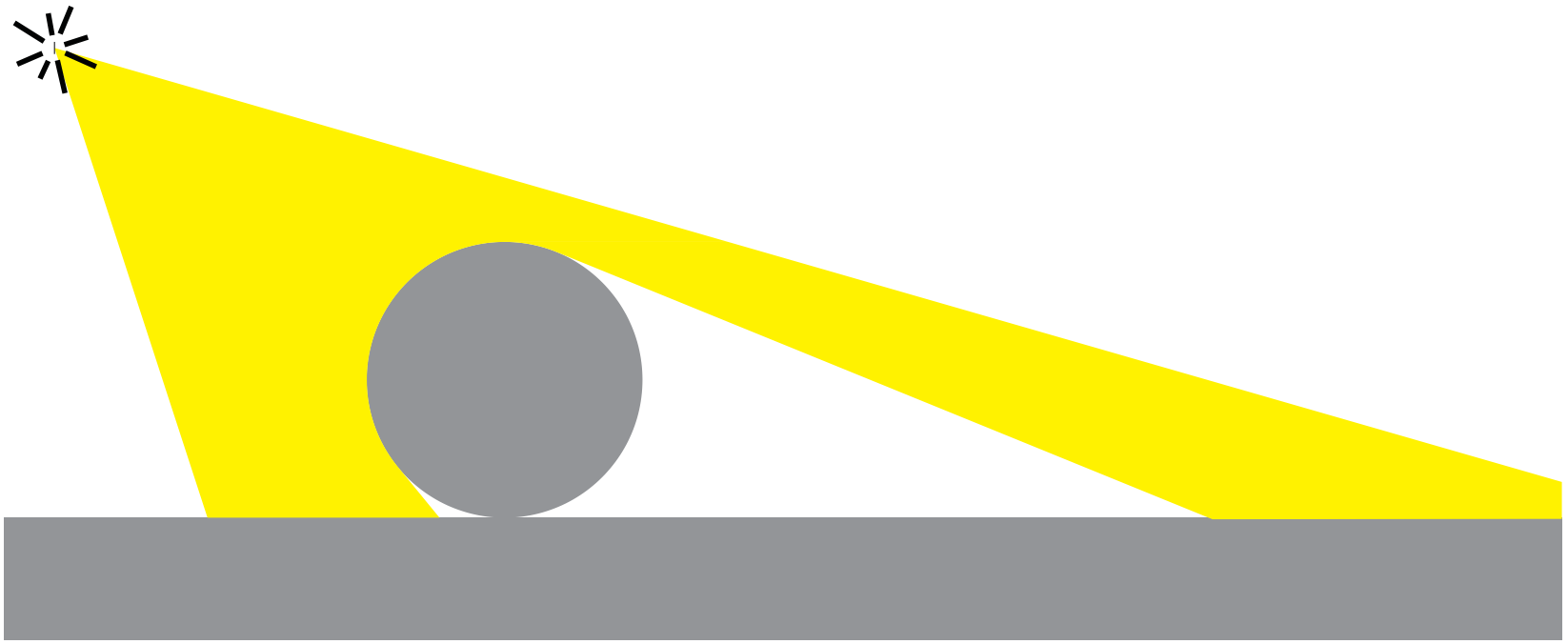
Les ombres, un indice de profondeur



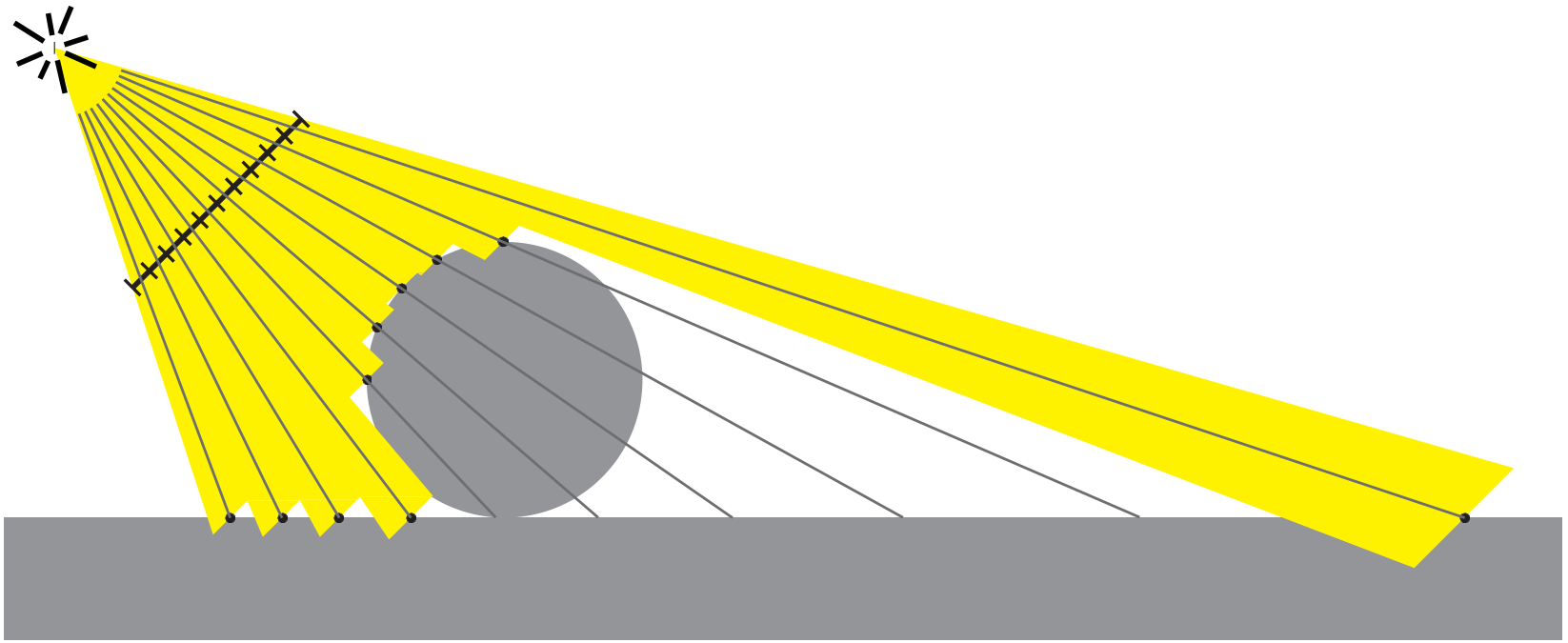
[tricks-and-illusions.com]

Les ombres, une ancre

Les ombres, une ancre

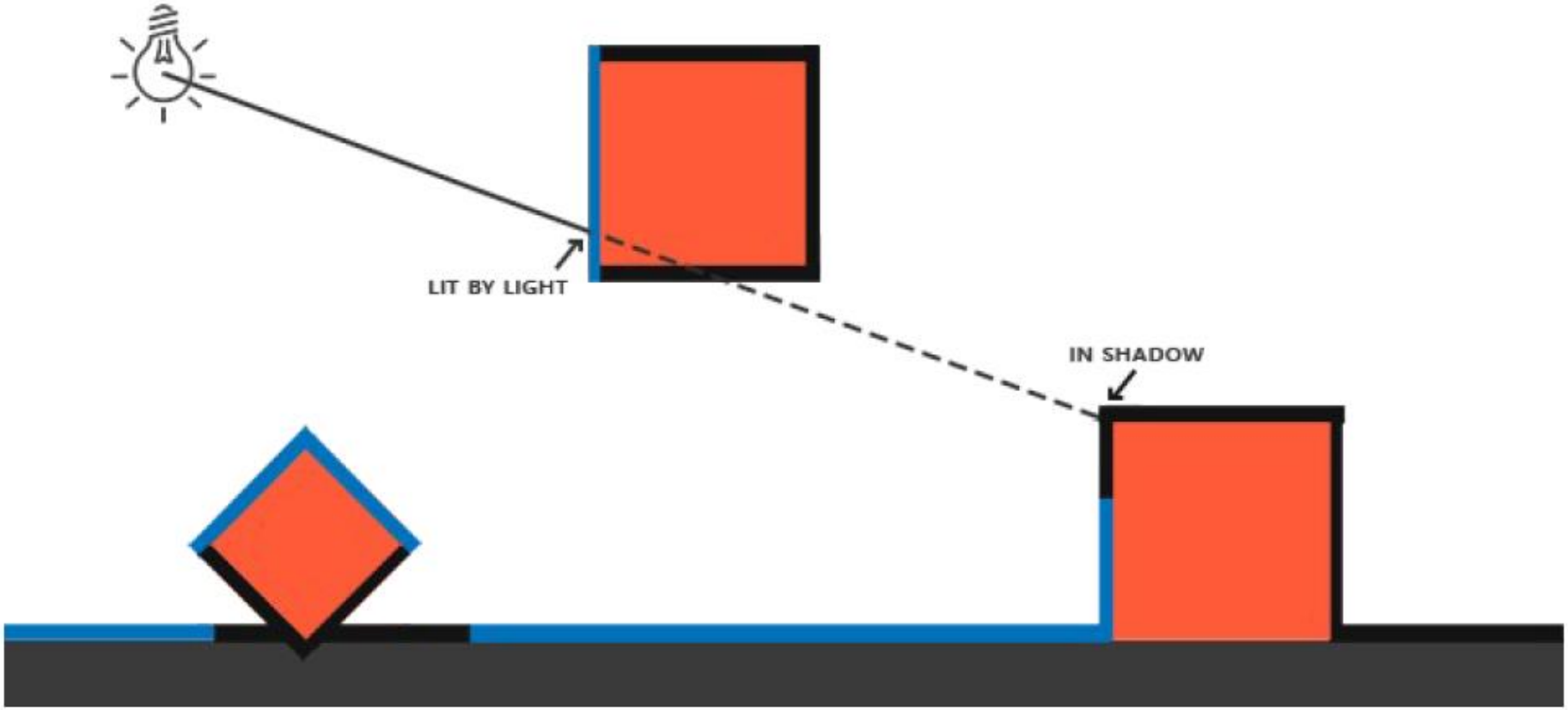


Shadow mapping

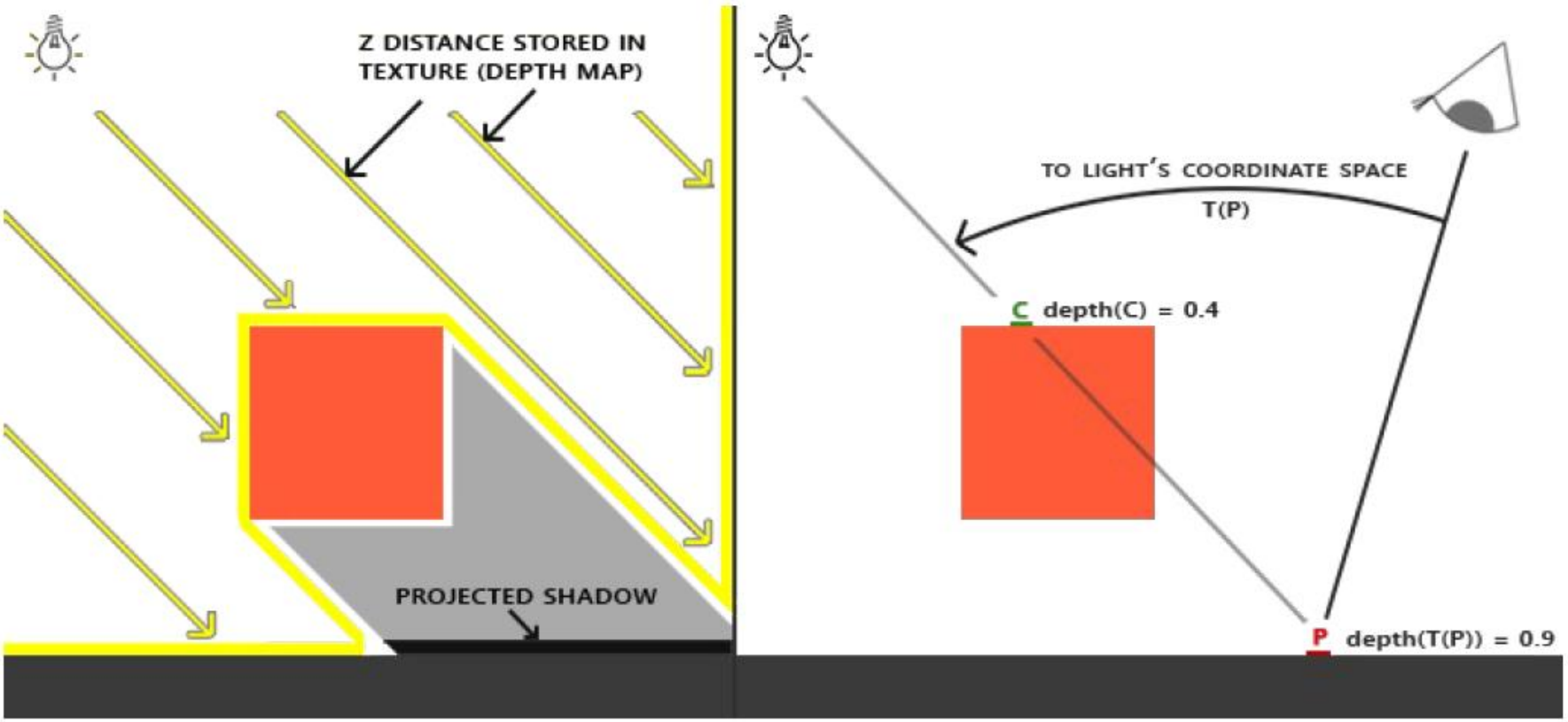


Shadow mapping

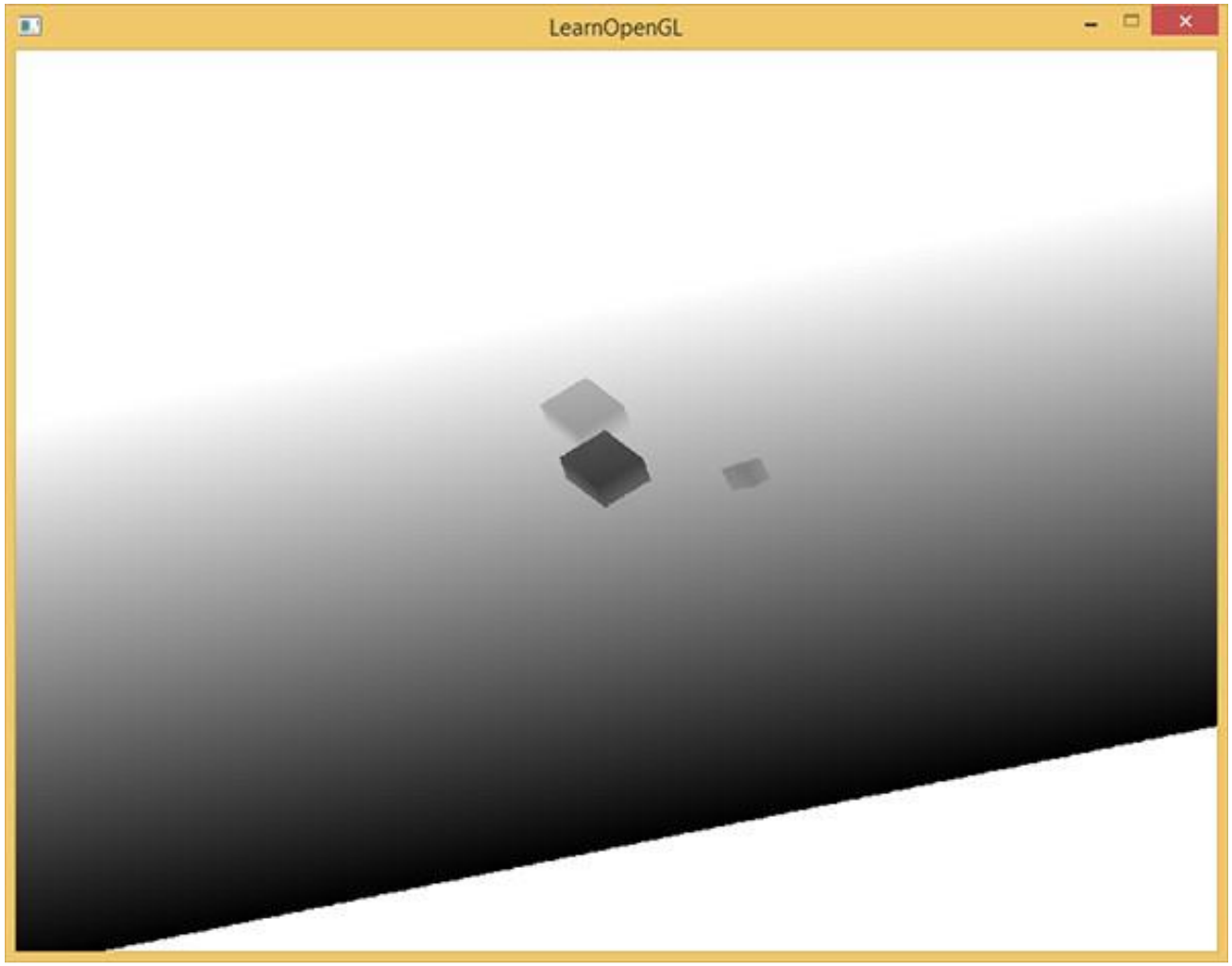
Shadow mapping

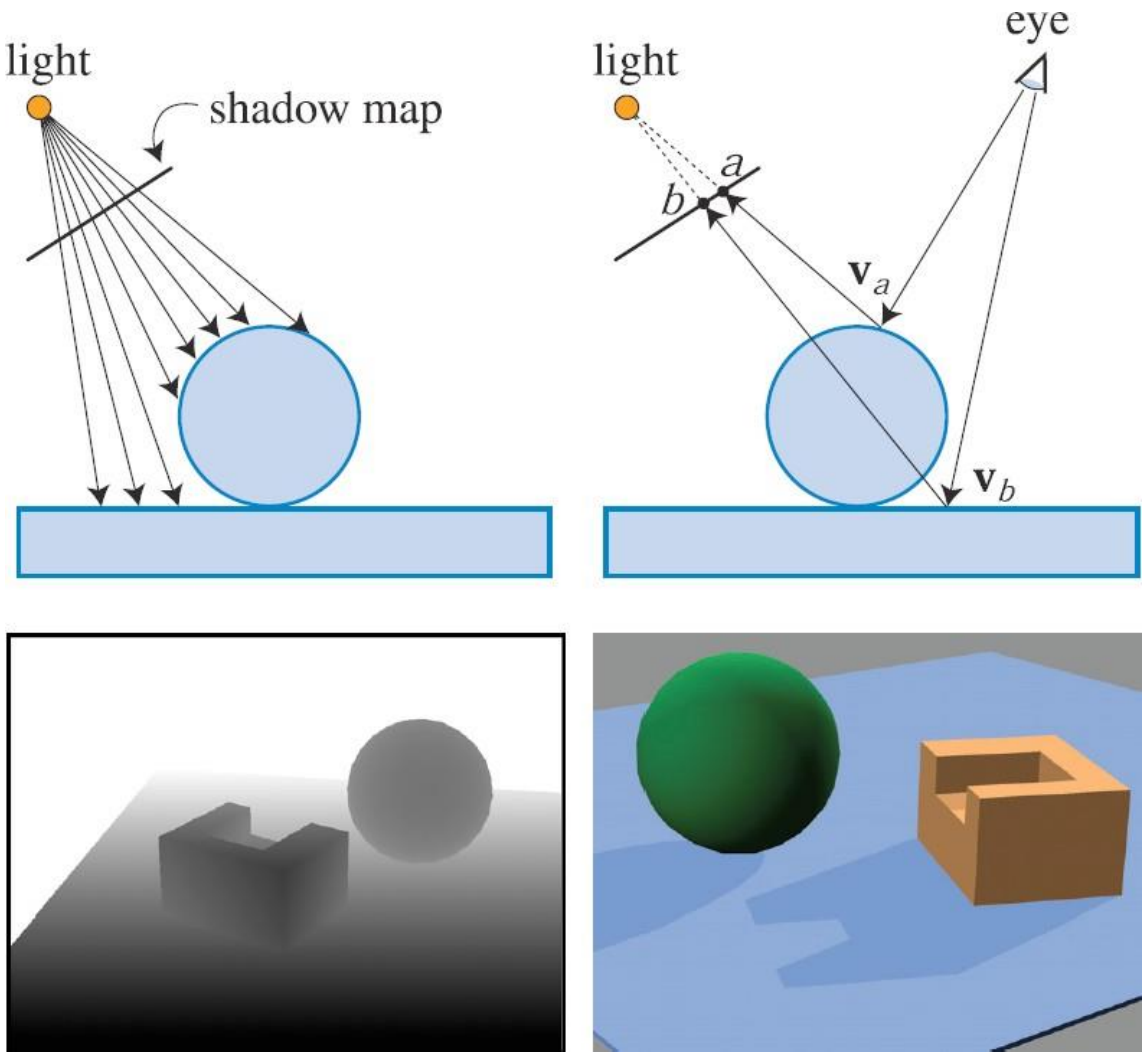


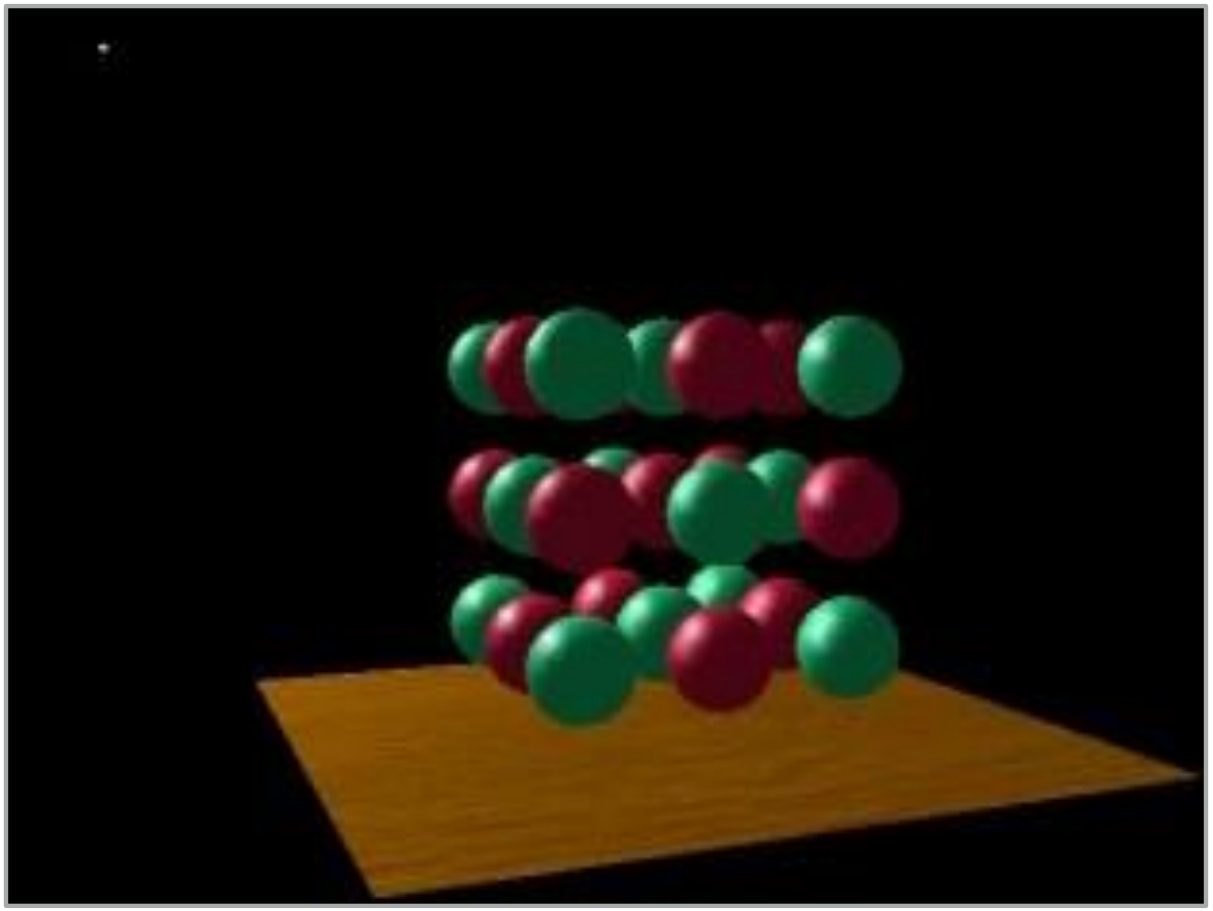
Shadow mapping



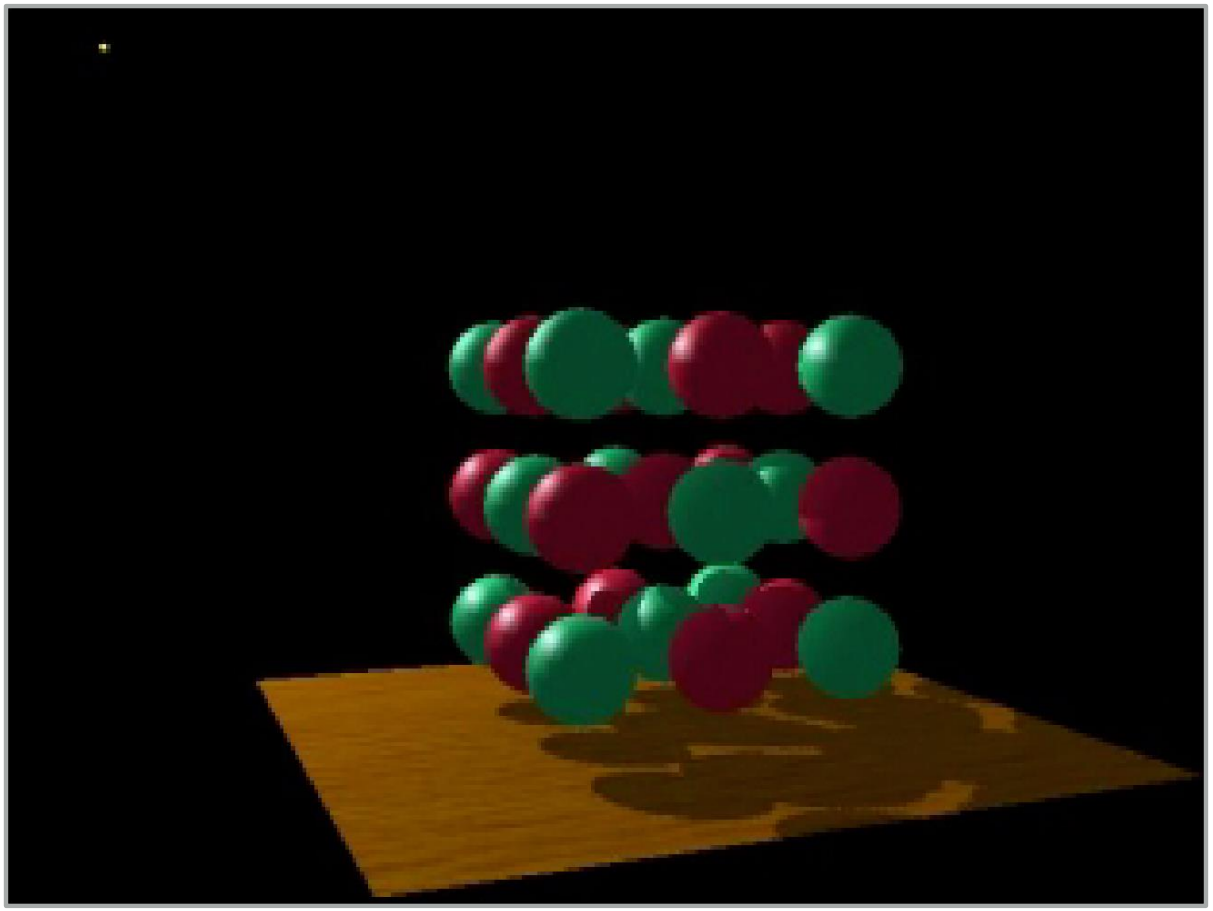
Shadow map : profondeur vue de la position de la lumière



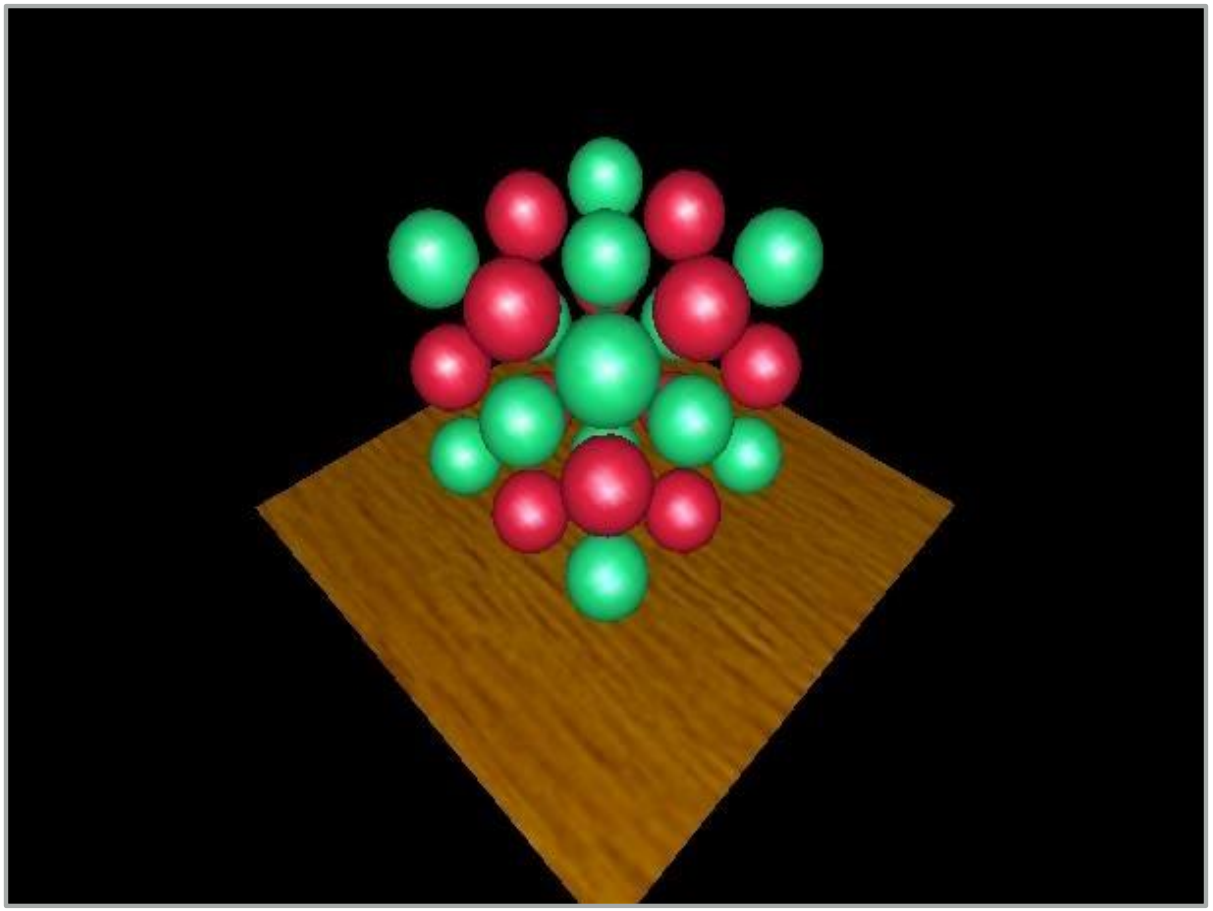

light
shadow map
light
eye
a
b
$\mathbf{v}_a$
$\mathbf{v}_b$



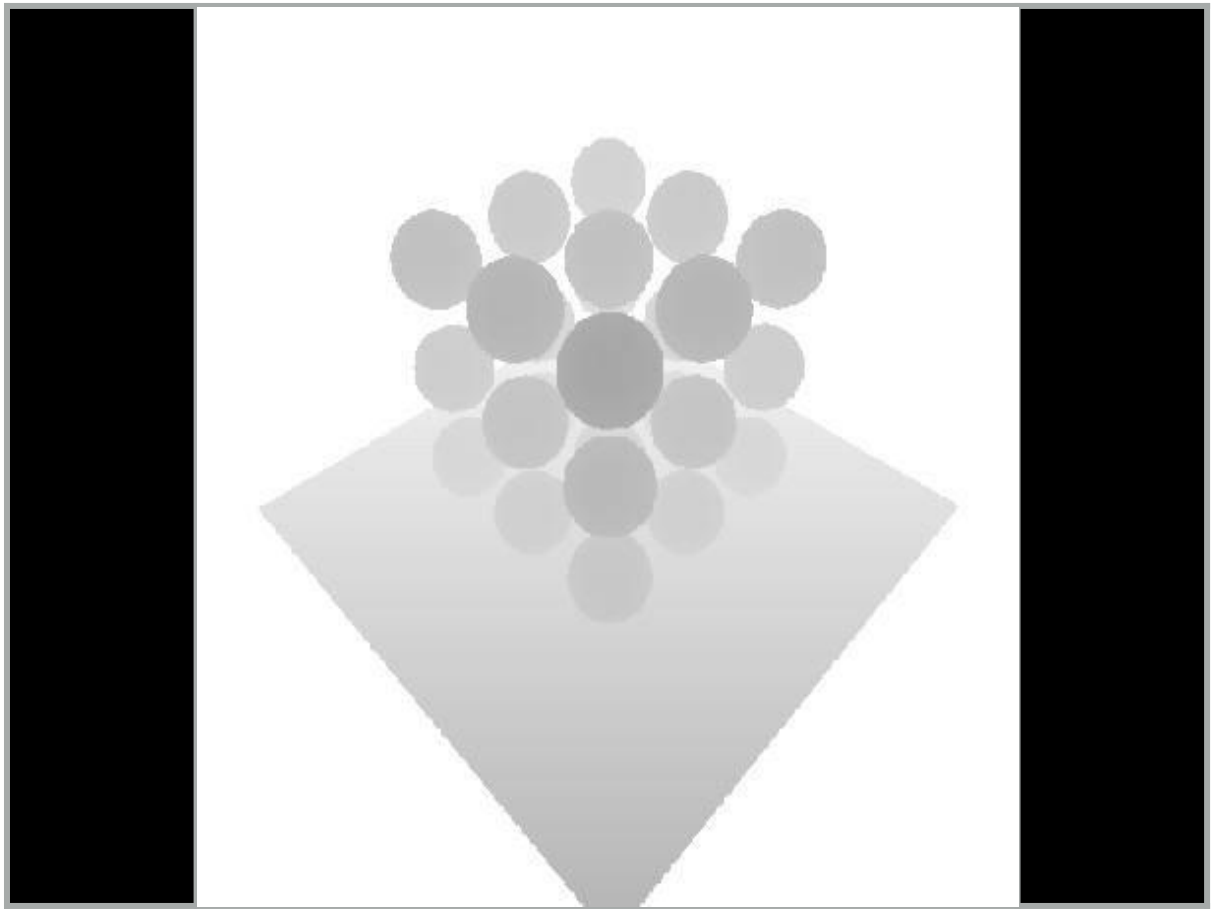
Mark Kilgard



Mark Kilgard



Mark Kilgard



Mark Kilgard

FBO pour shadow map

```
struct FBO_ShadowMap {
    GLuint depthMapFBO;
    GLuint depthMapTexture;
    unsigned int depthMapTextureWidth;
    unsigned int depthMapTextureHeight;

    bool allocate( unsigned int width = 1024 , unsigned int height = 768 ) {
        glGenFramebuffers(1, &depthMapFBO);
        glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);

        depthMapTextureWidth = width;
        depthMapTextureHeight = height;

        // Depth texture. Slower than a depth buffer, but you can sample it later in your shader
        glGenTextures(1, &depthMapTexture);
        glBindTexture(GL_TEXTURE_2D, depthMapTexture);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT16, width, height, 0, GL_DEPTH_COMPONENT, GL_FLOAT, 0);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, depthMapTexture, 0);

        glDrawBuffer(GL_NONE); // No color buffer is drawn to.

        if(glCheckFramebufferStatus(GL_FRAMEBUFFER) == GL_FRAMEBUFFER_COMPLETE) {
            std::cout << "FBO successfully created" << std::endl;
            glBindFramebuffer(GL_FRAMEBUFFER, 0);
            return true;
        }
        else{
            std::cout << "PROBLEM IN FBO FBO_ShadowMap::allocate() : FBO NOT successfully created" << std::endl;
            glBindFramebuffer(GL_FRAMEBUFFER, 0);
            return false;
        }
    }
    void free() {
        glDeleteFramebuffers(1, &depthMapFBO);
    }
```

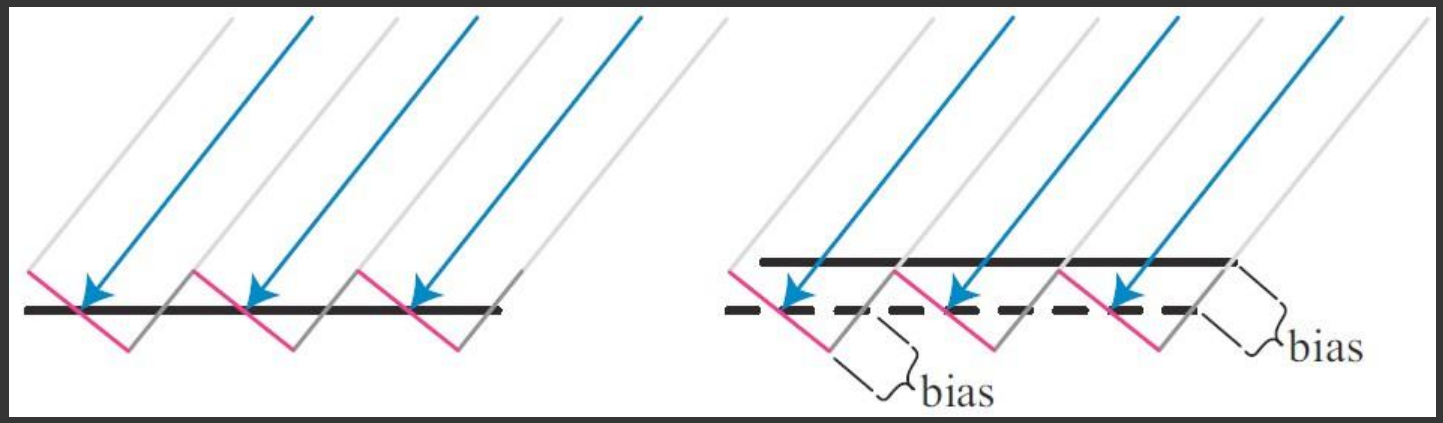
Comparer les profondeurs

- Dans le fragment shader

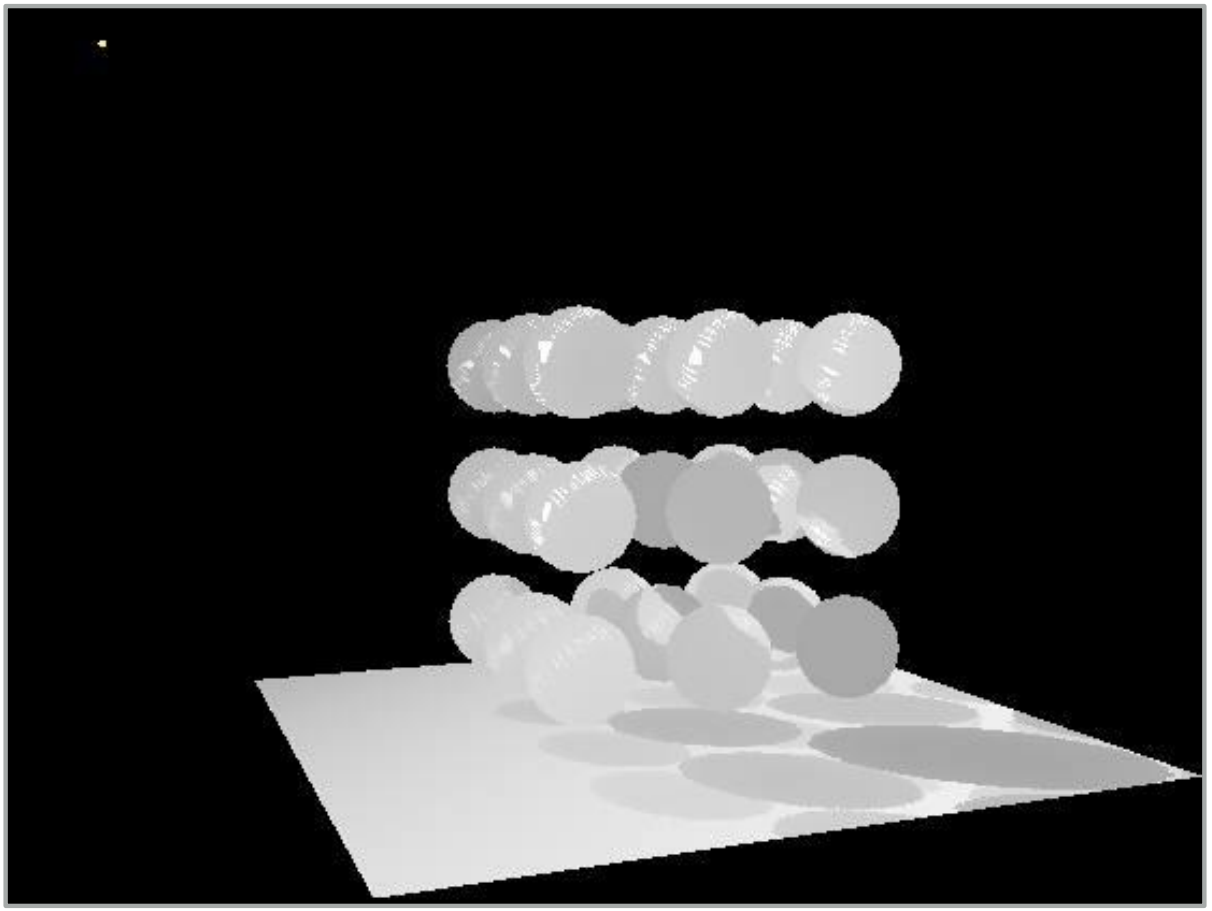
```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // perform perspective divide
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // transform to [0,1] range
    projCoords = projCoords * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragPosLight as coords)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    // check whether current frag pos is in shadow
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;

    return shadow;
}
```

Shadow Map : problème

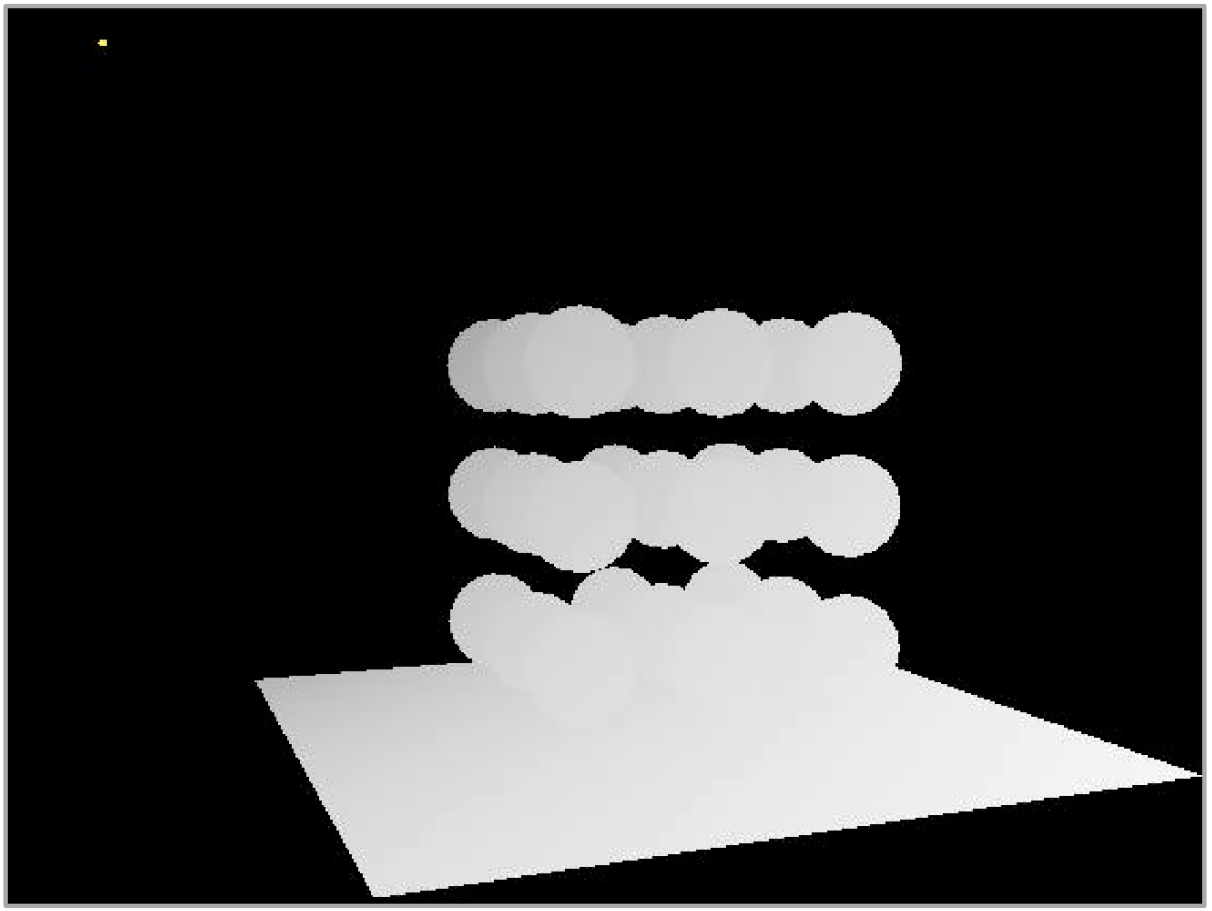
- Si A et B sont presque égaux?
- Speckling



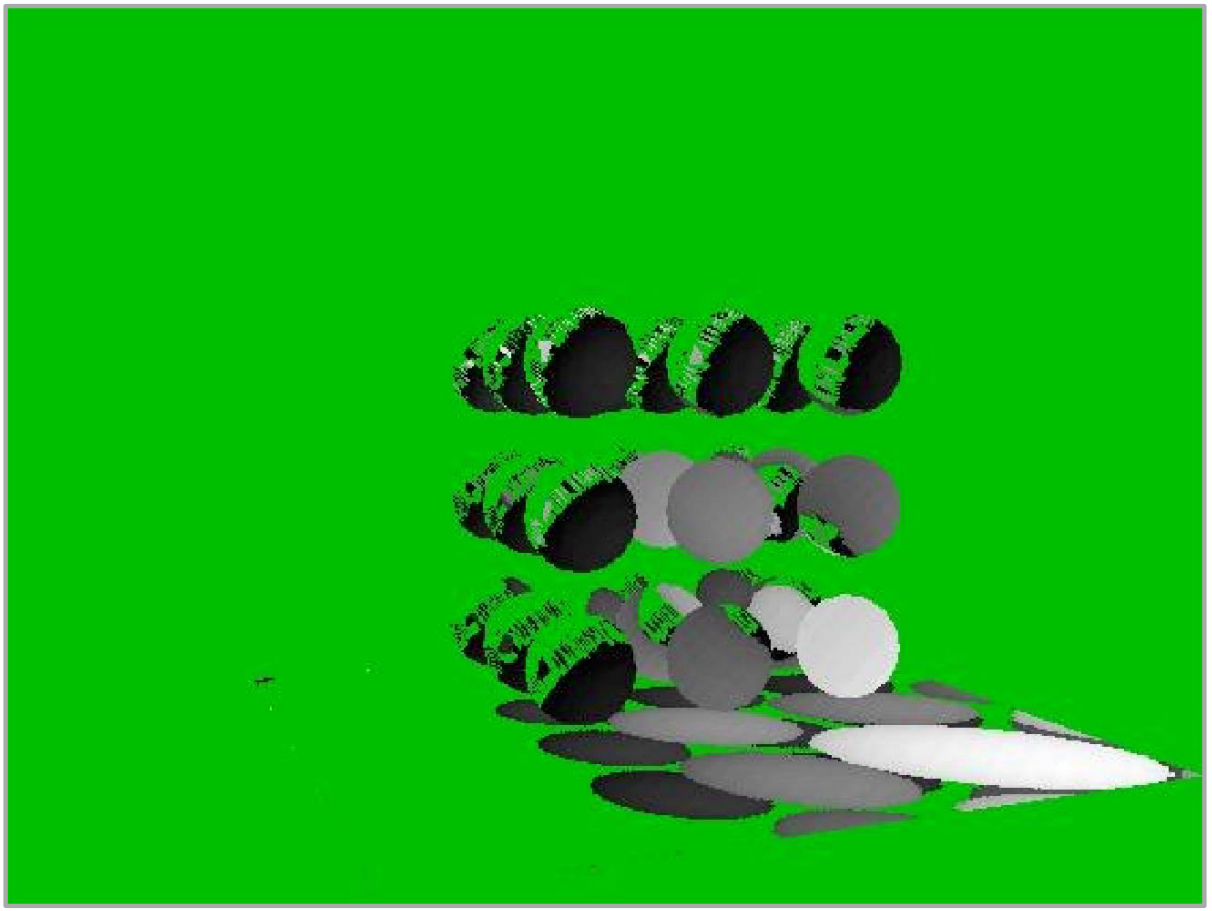
slide courtesy of Kavita Bala, Cornell University



Mark Kilgard

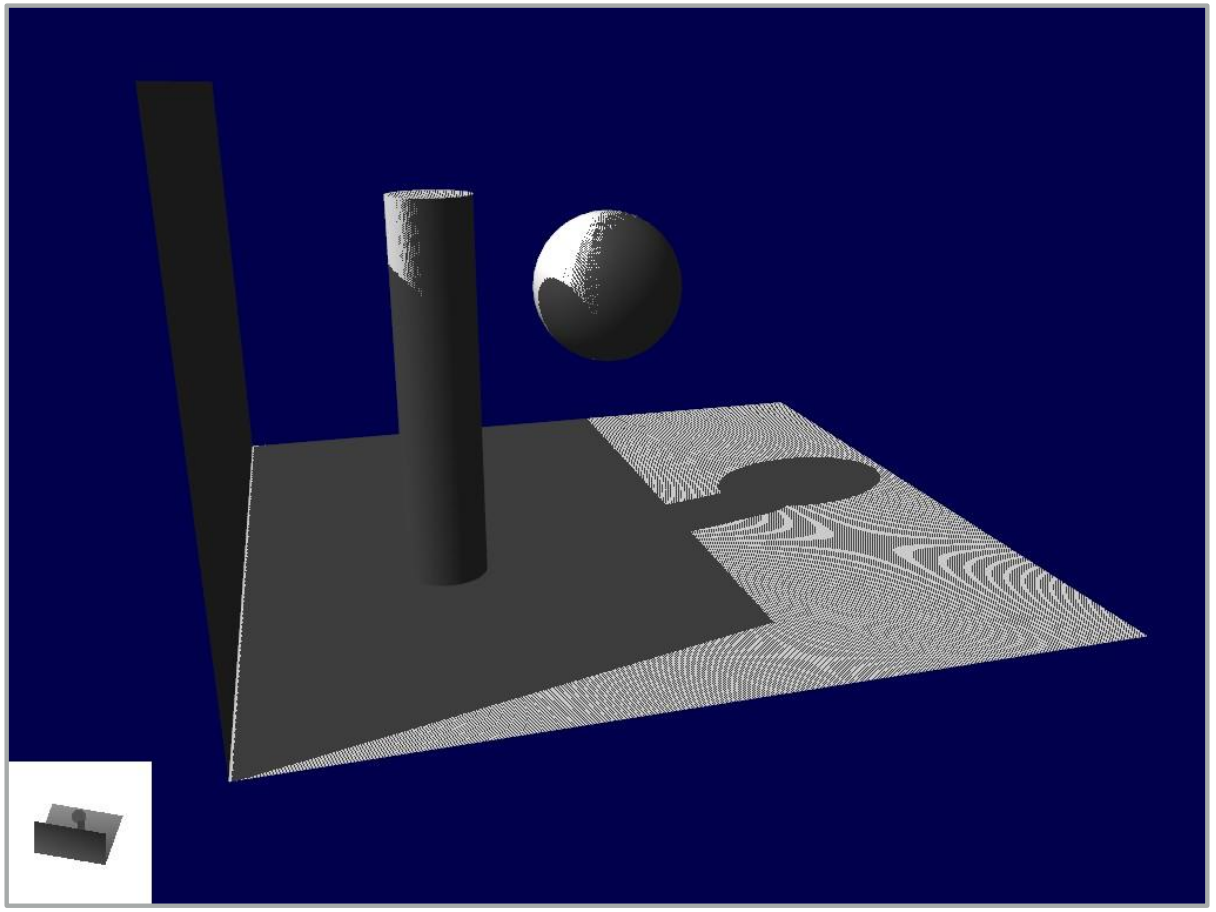


Mark Kilgard

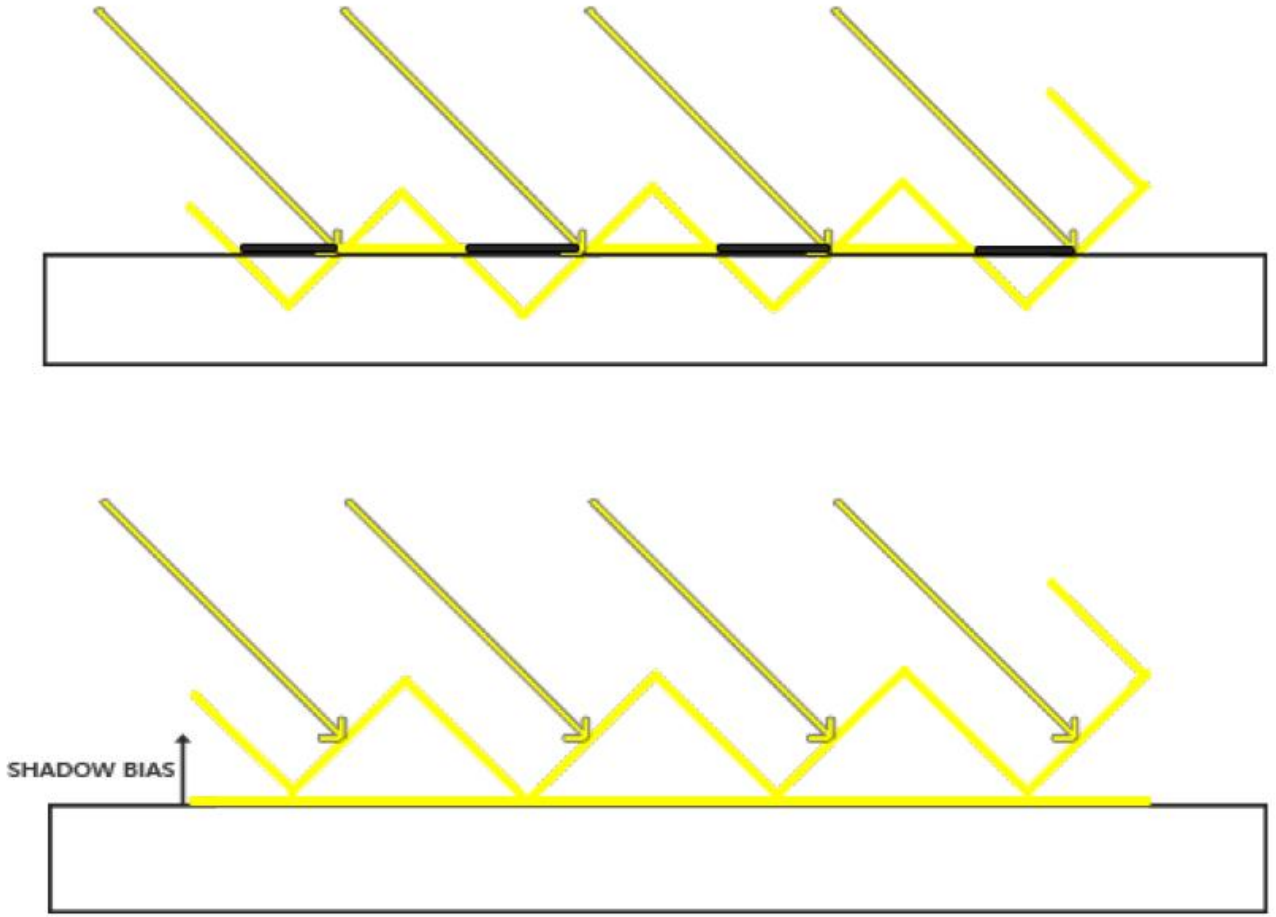


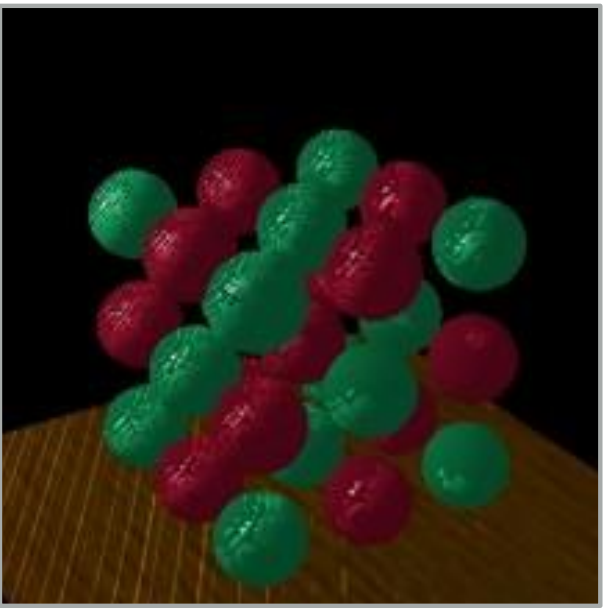
Mark Kilgard

1er essai de shadow mapping

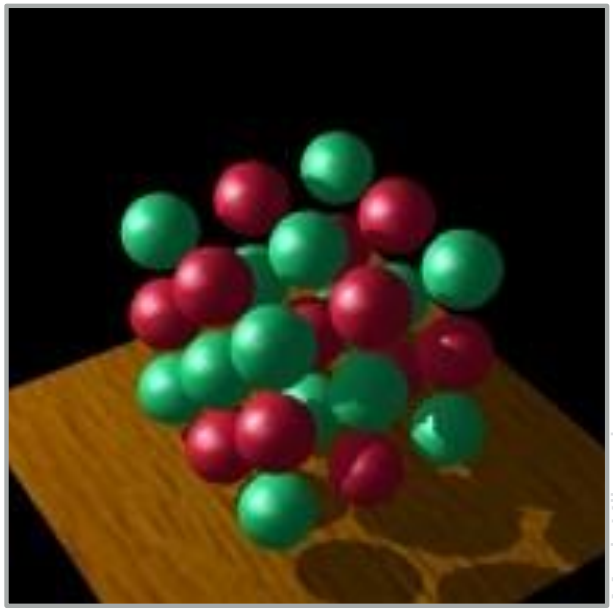


Solution 1 : ajouter un offset

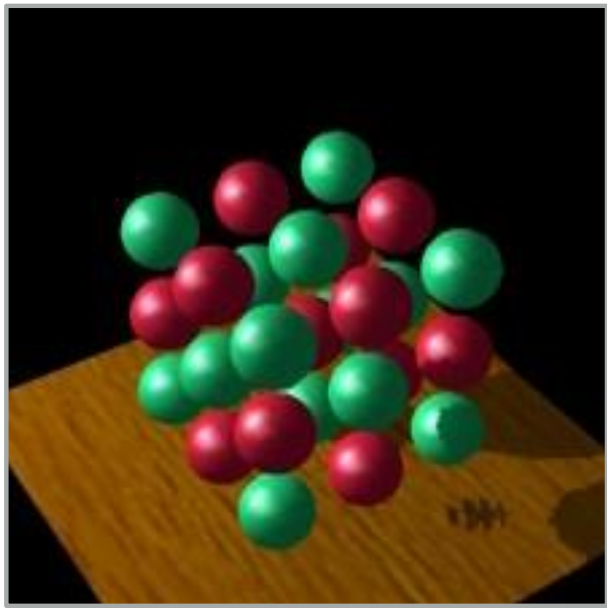




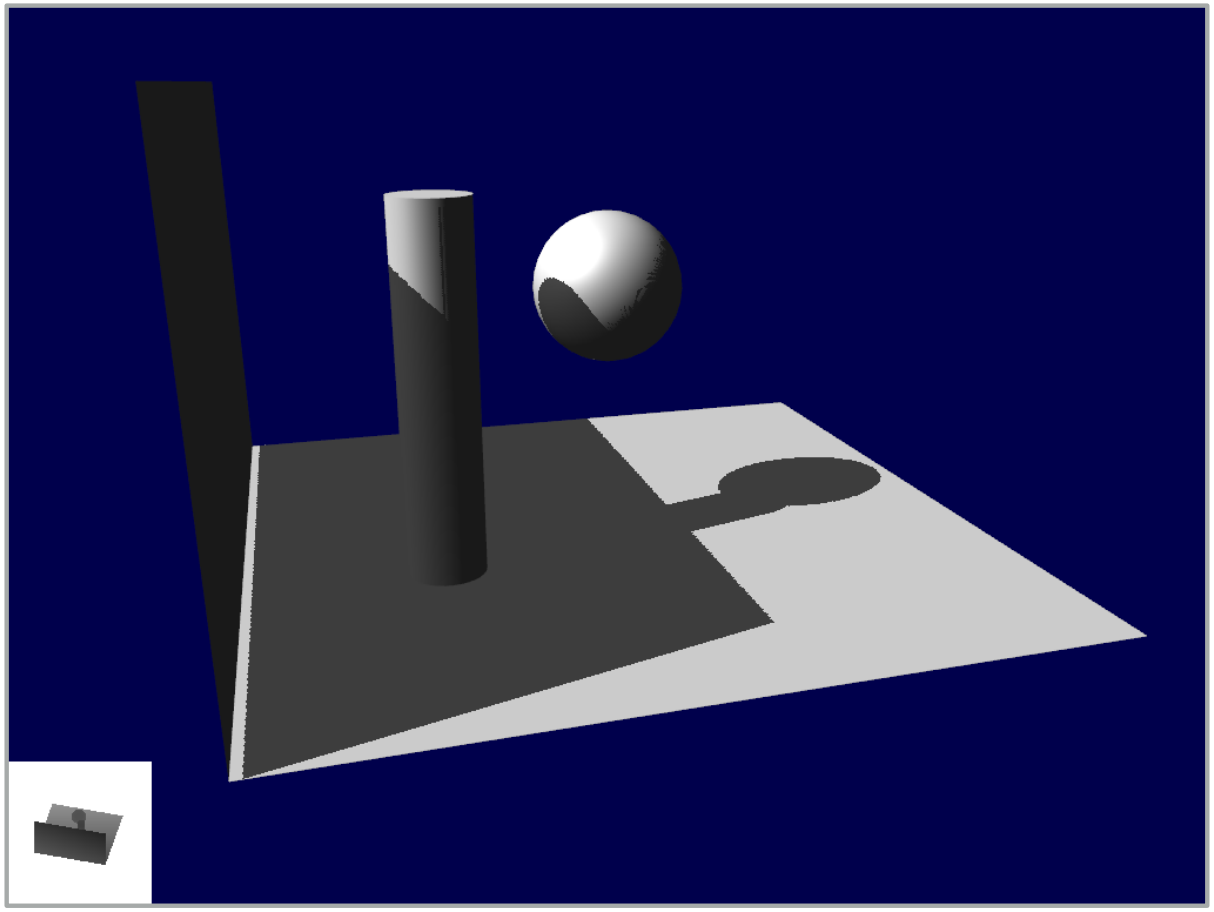
not enough shadow bias



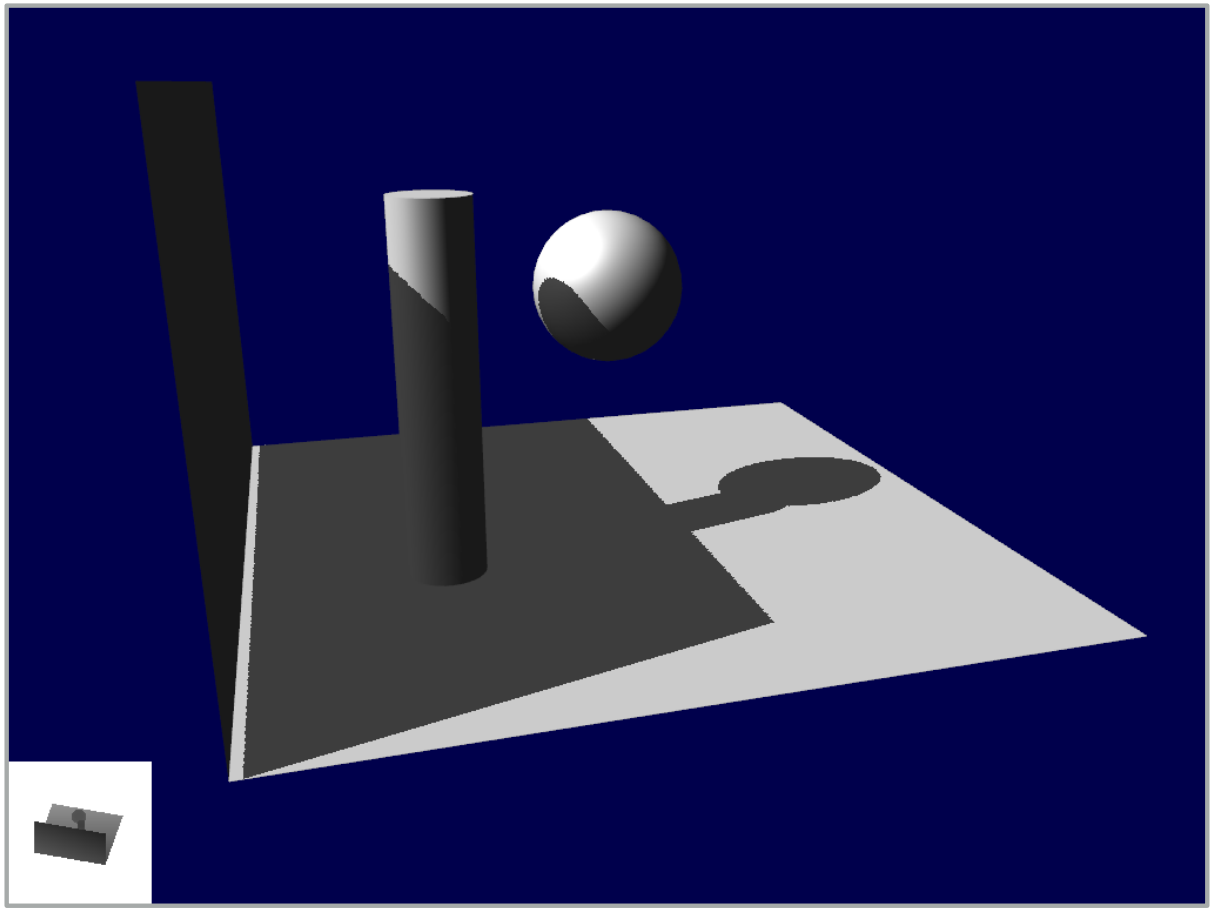
good shadow bias



too much shadow bias

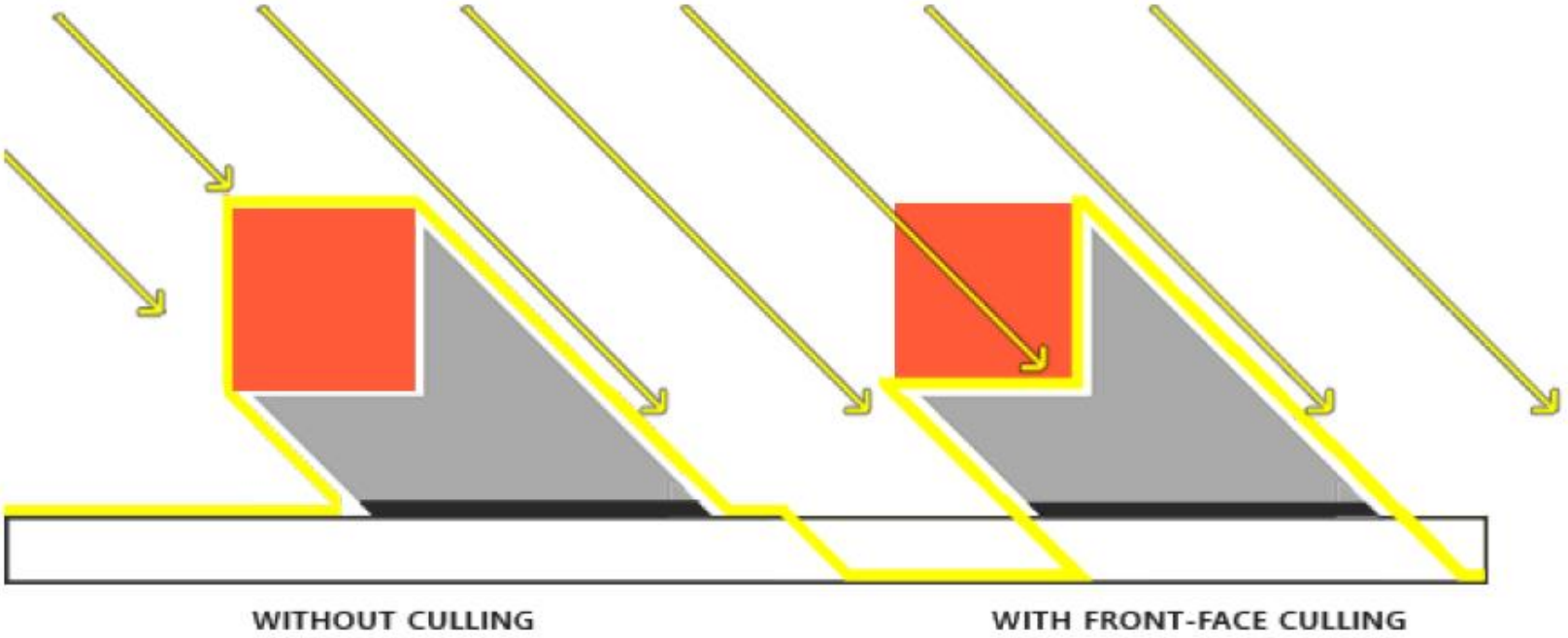


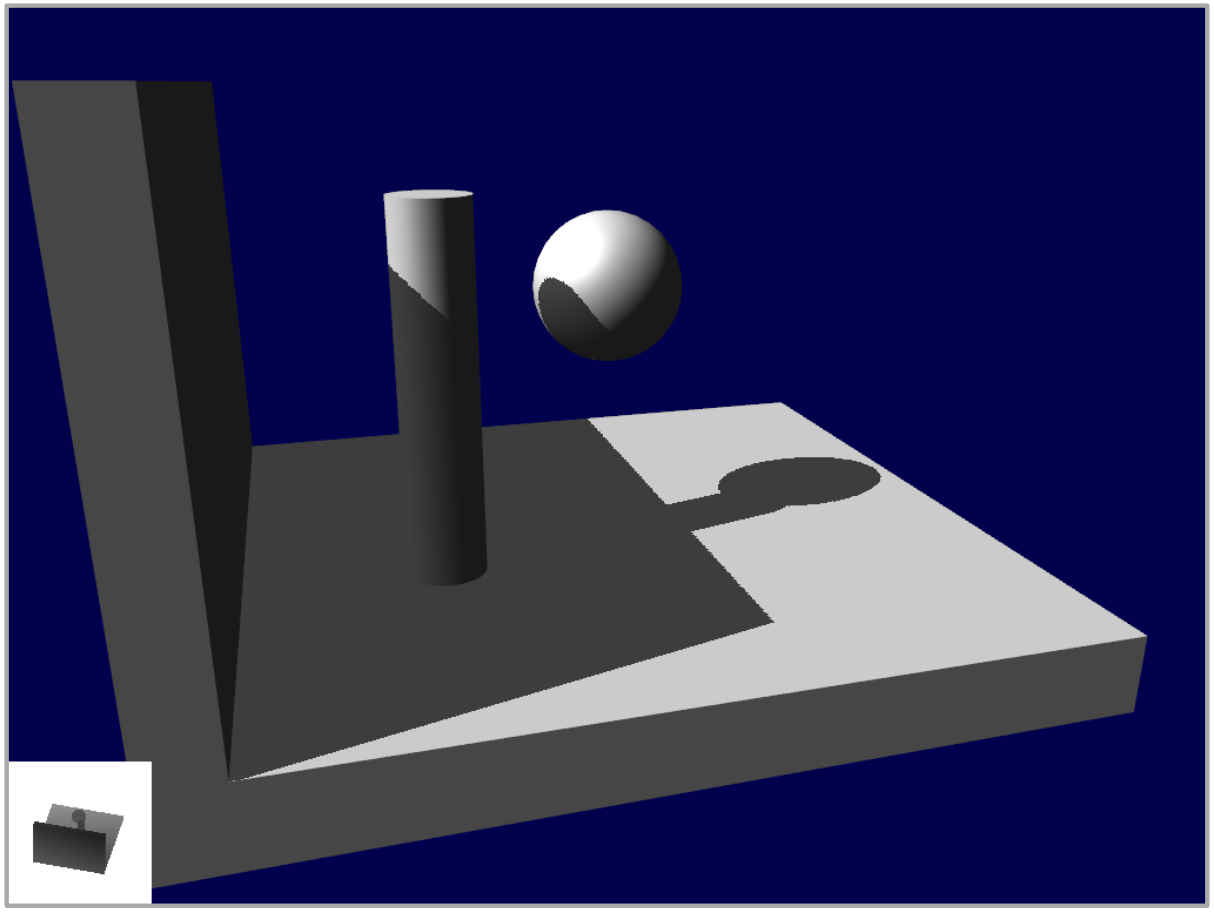
shadow mapping with constant bias



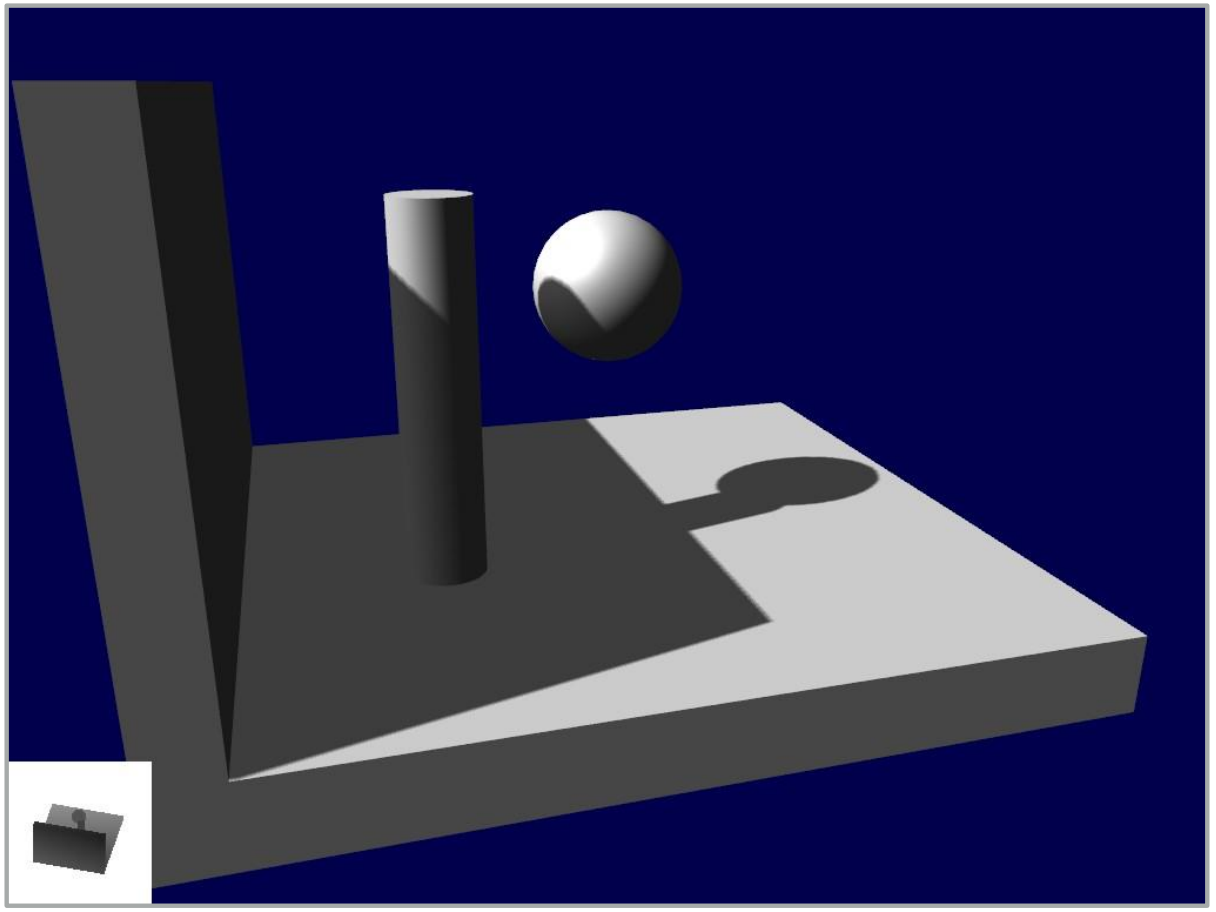
shadow mapping with slope-dependent bias

Solution 2 pour les objets solides et bien orientés





closed surfaces and slope-dependent bias



adding percentage-closer filtering

Cascaded shadow maps (parallel-split)

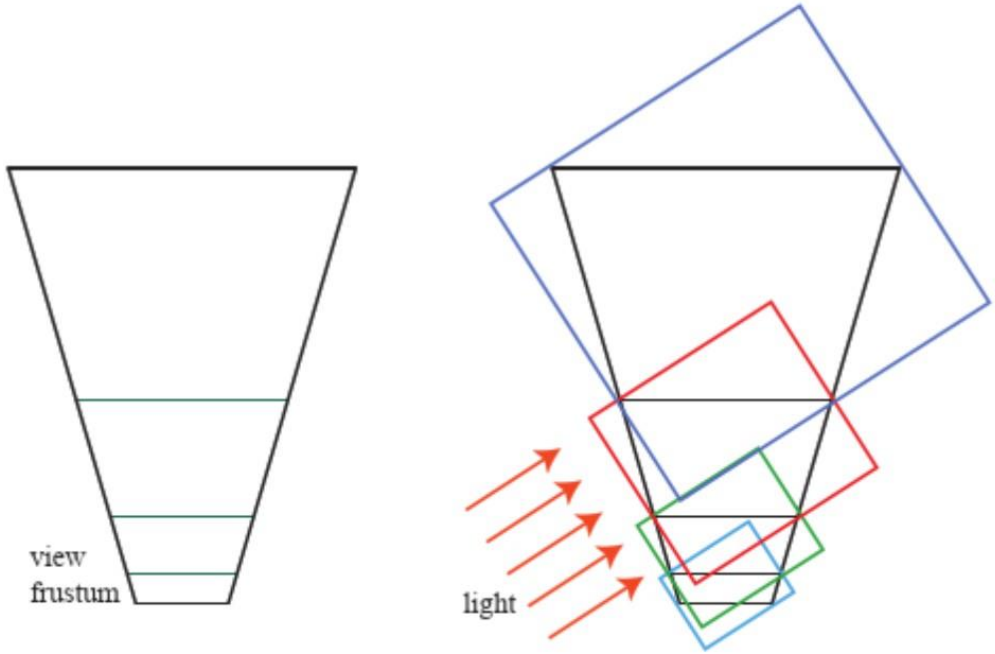


Figure 7.18. On the left, the view frustum from the eye is split into four volumes. On the right, bounding boxes are created for the volumes, which determine the volume rendered by each of the four shadow maps for the directional light. *(After Engel [430].)*

[Möller et al. *RTR*]

Single shadow map, 2048x2048

Four 1024x1024 shadow maps (equal memory)

Filtrer les shadow maps

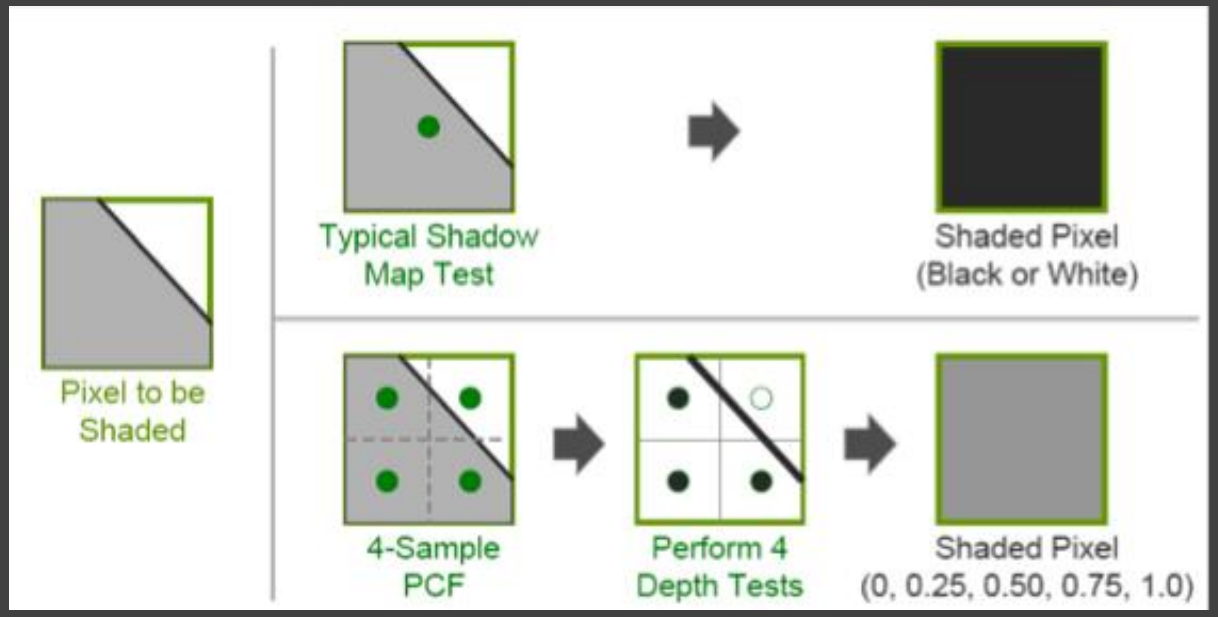
- Shadow map "lookups" causent de l'aliasing

- Les pixels sont des fonctions non-linéaires de la profondeur d'ombrage
  - Appliquer un filtrage linéaire de la profondeur est faux

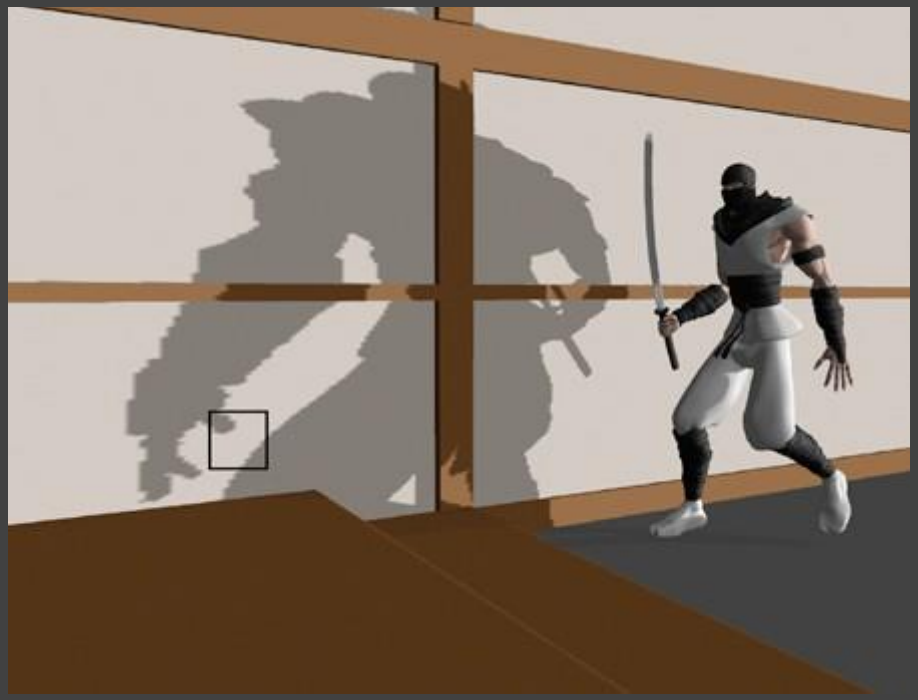
- On veut filtrer la sortie, pas l'entrée, du test d'ombrage
  - Quelle fraction des échantillons passent le test
  - Les échantillons passent le test si ils sont plus proches que la profondeur de la Shadow Map
  - « percentage closer filtering » ou PCF

Percentage Closer Filtering

- Soften the shadow to decrease aliasing
  - Reeves, Salesin, Cook 87
  - GPU Gems, Chapter 11

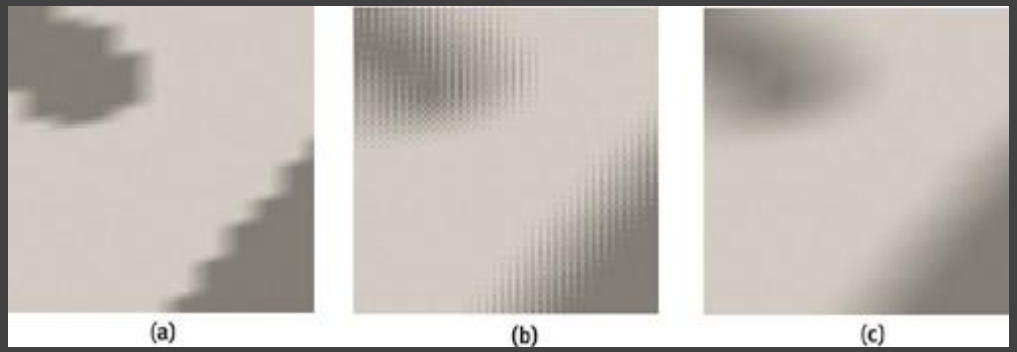


1 sample SM



4 sample PCF

16 sample PCF


(a)
(b)
(c)

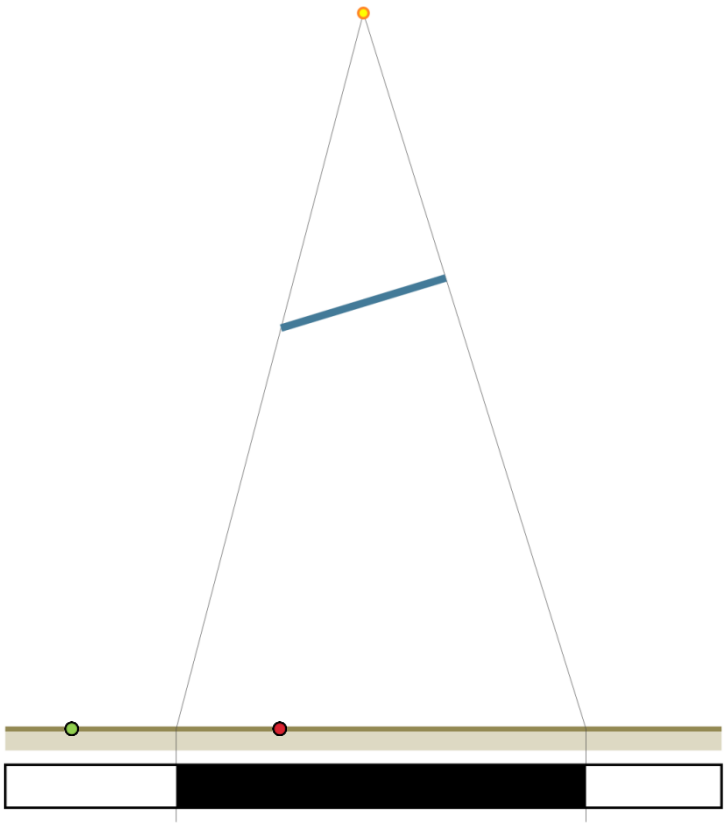
Ombres douces (petites sources)

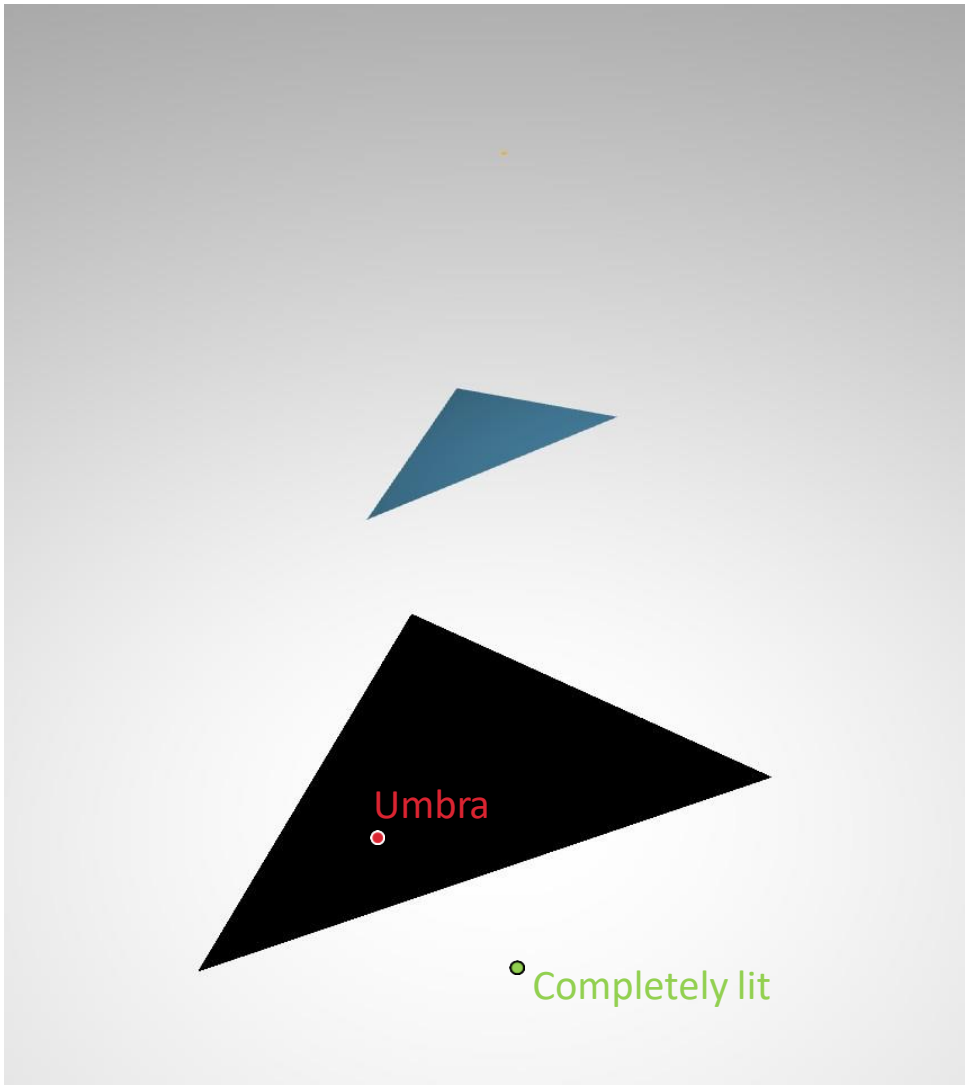
- Effet : flouter les frontières
  - PCF
  - … quelle largeur de filtre ?

- Les ombres réelles dépendent de l'aire de la lumière visible sur la surface
  - Varie de façon complexe
  - Ex : lumière à travers les feuilles d'un arbre

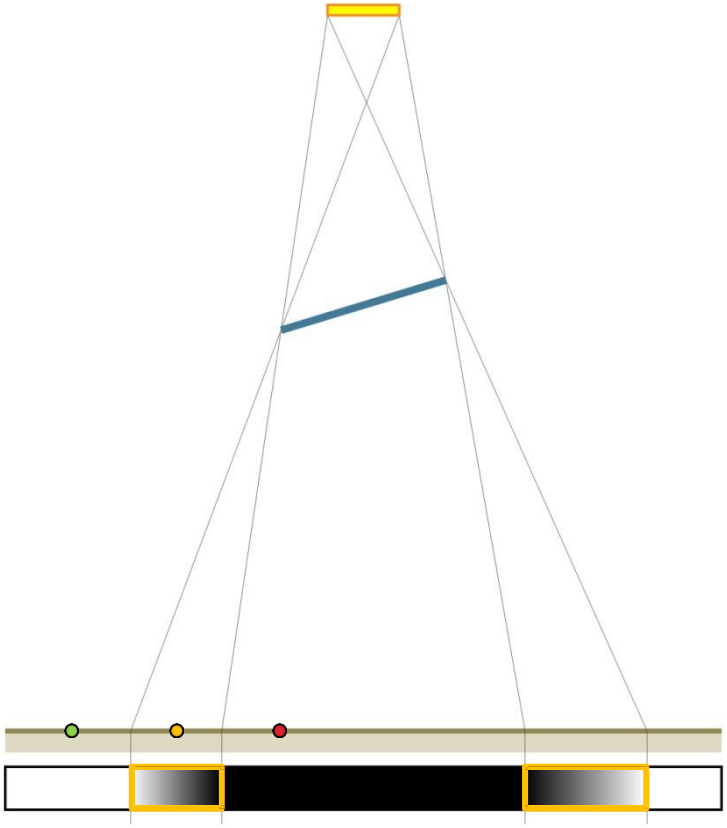
- Approximation : convolution
  - Les ombres sont convoluées quand les bloqueurs et la source sont parallèles et planaires
  - Fusion des occulteurs : approximation de la géométrie par un bloqueur planaire

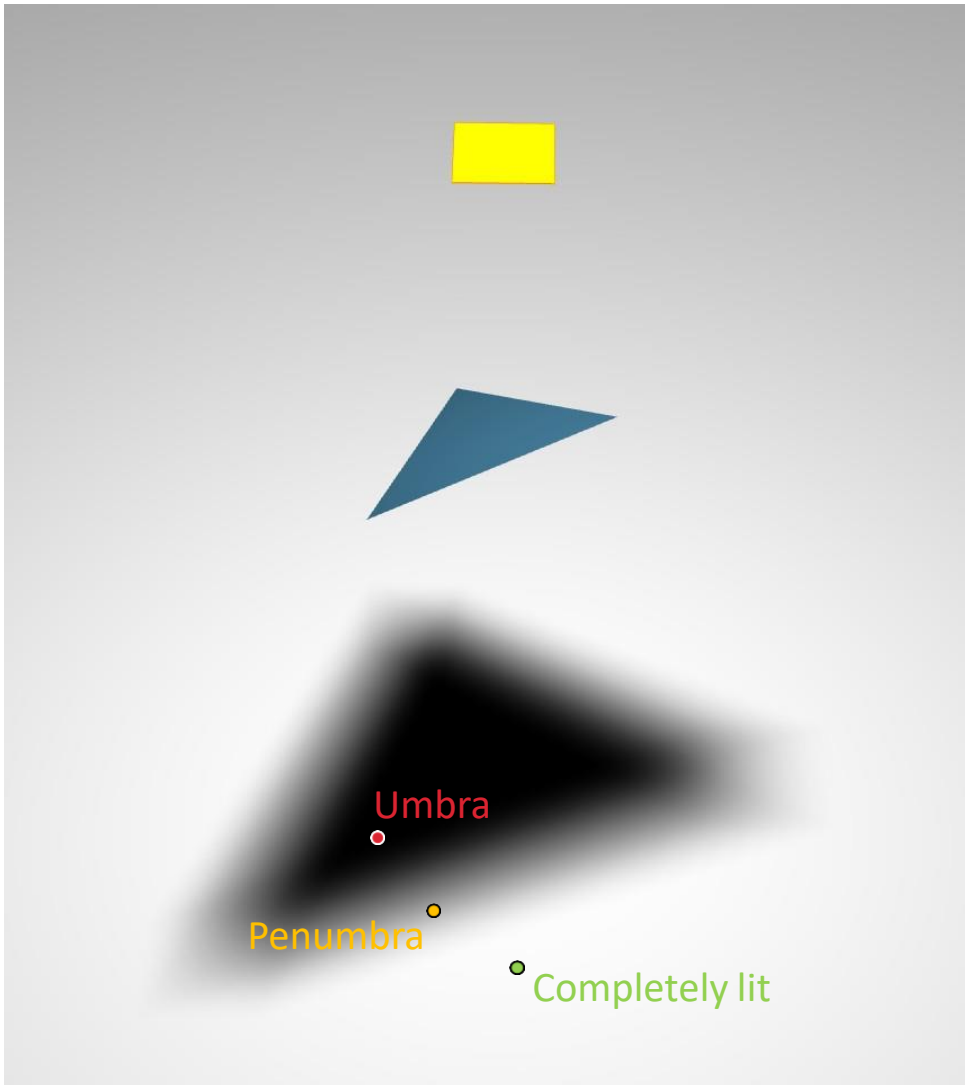
Hard Shadows



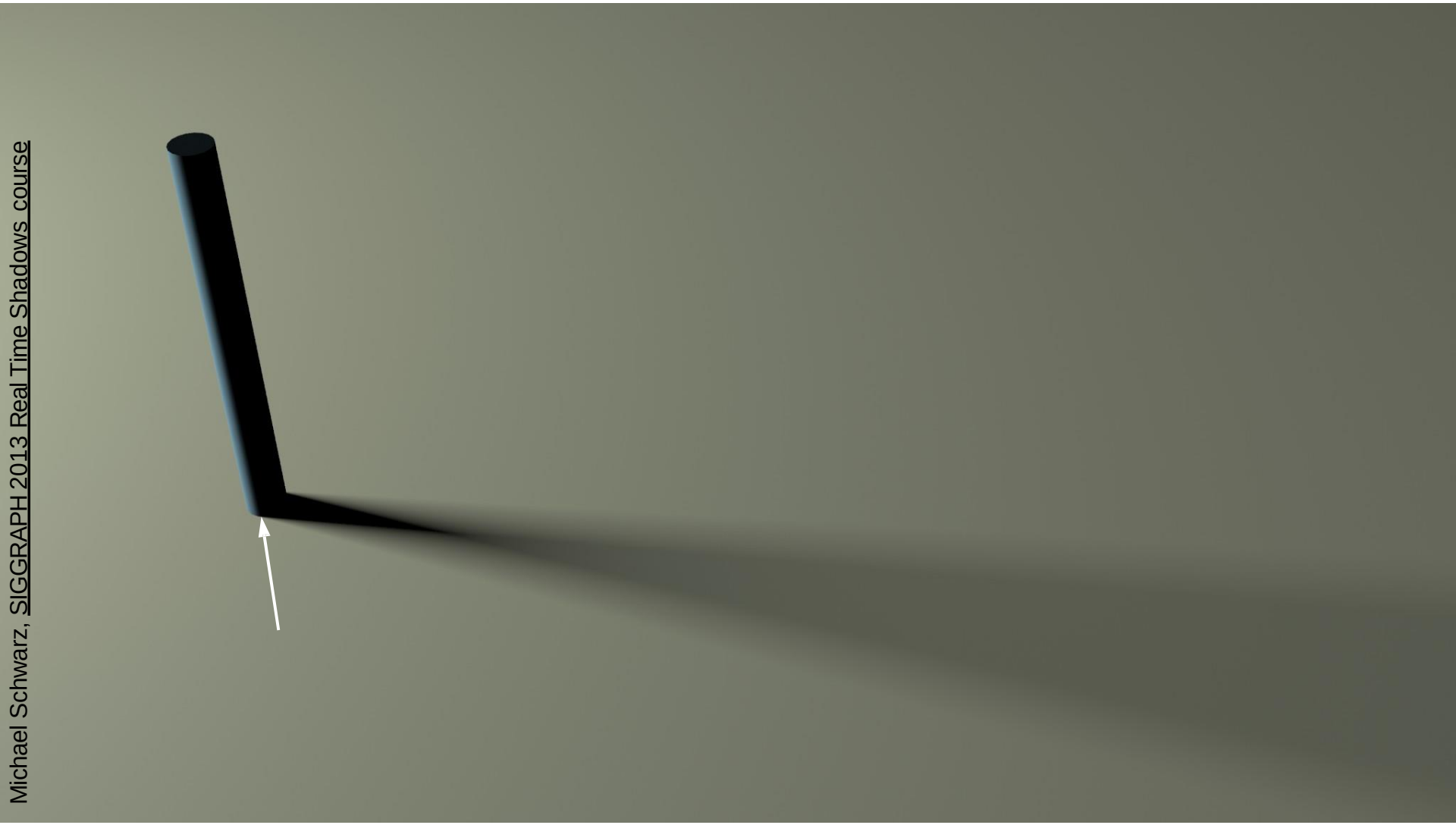


Soft Shadows





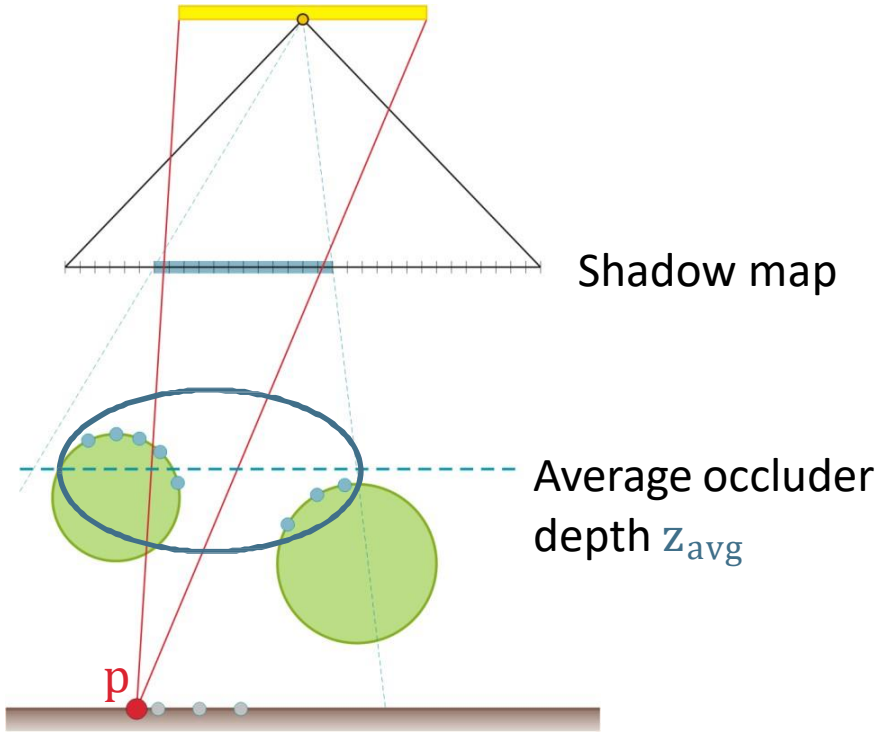
Shadow Hardening on Contact



Michael Schwarz, SIGGRAPH 2013 Real Time Shadows course

Percentage-Closer Soft Shadows

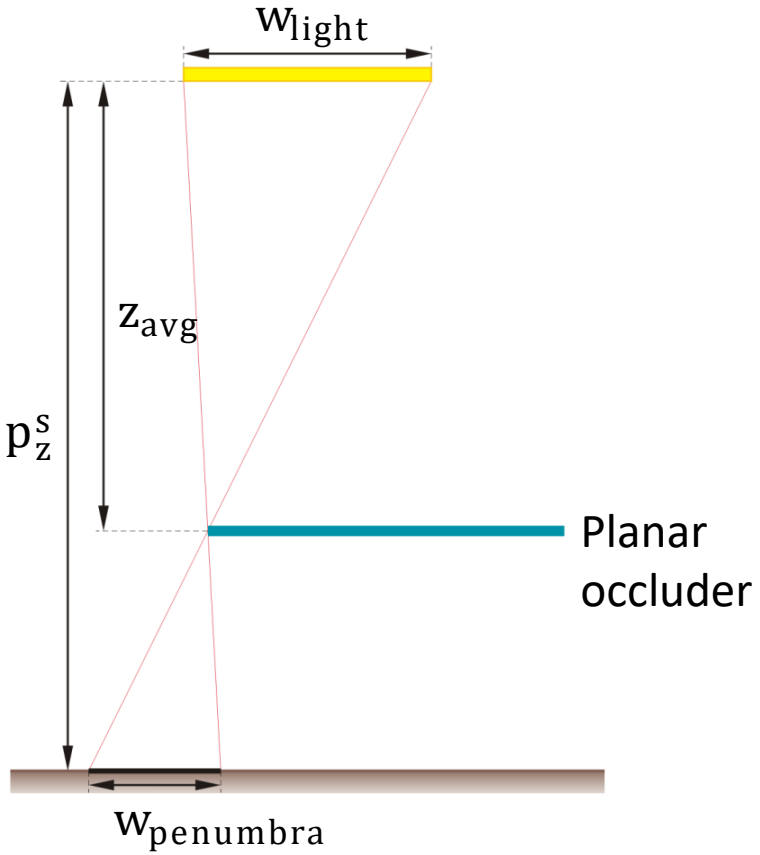
1. Blocker search



Percentage-Closer Soft Shadows

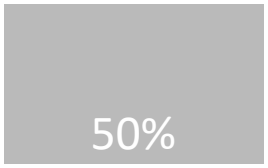
1. Blocker search
2. Penumbra width estimation

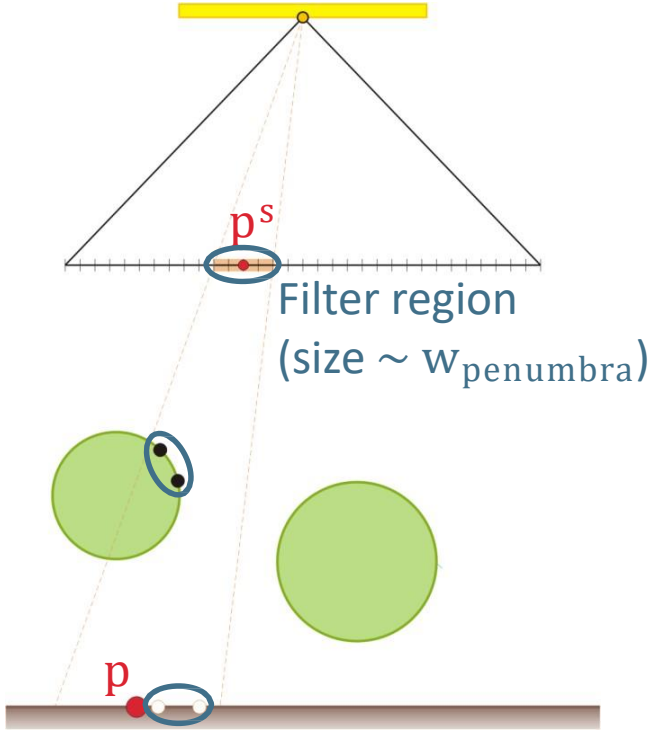
$$w_{penumbra} = \frac{p_z^s - z_{avg}}{z_{avg}} w_{light}$$



Percentage-Closer Soft Shadows

1. Blocker search
2. Penumbra width estimation
3. Filtering





PCF – ombres douces

Fernando, NVidia whitepaper ~2005

```
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
for(int x = -1; x <= 1; ++x)
{
    for(int y = -1; y <= 1; ++y)
    {
        float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r;
        shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
    }
}
shadow /= 9.0;
```