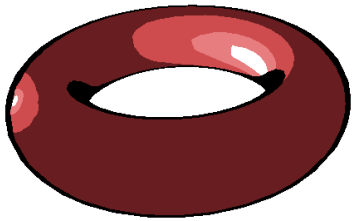
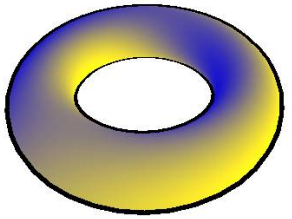
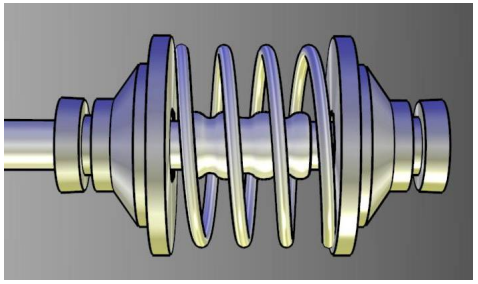
Quelques exemples de shaders



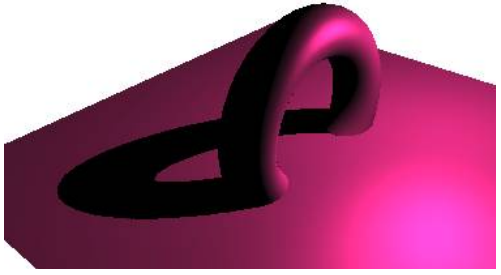
Toon shader avec silhouette



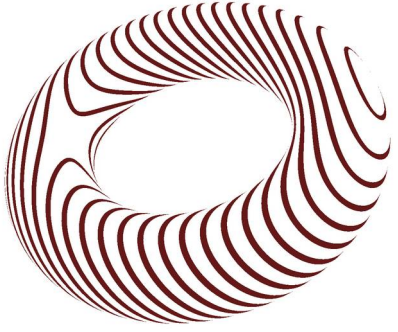


Le shader de « Gooch »

Environement map



Shadow map



Lignes de niveau

SILHOUETTE

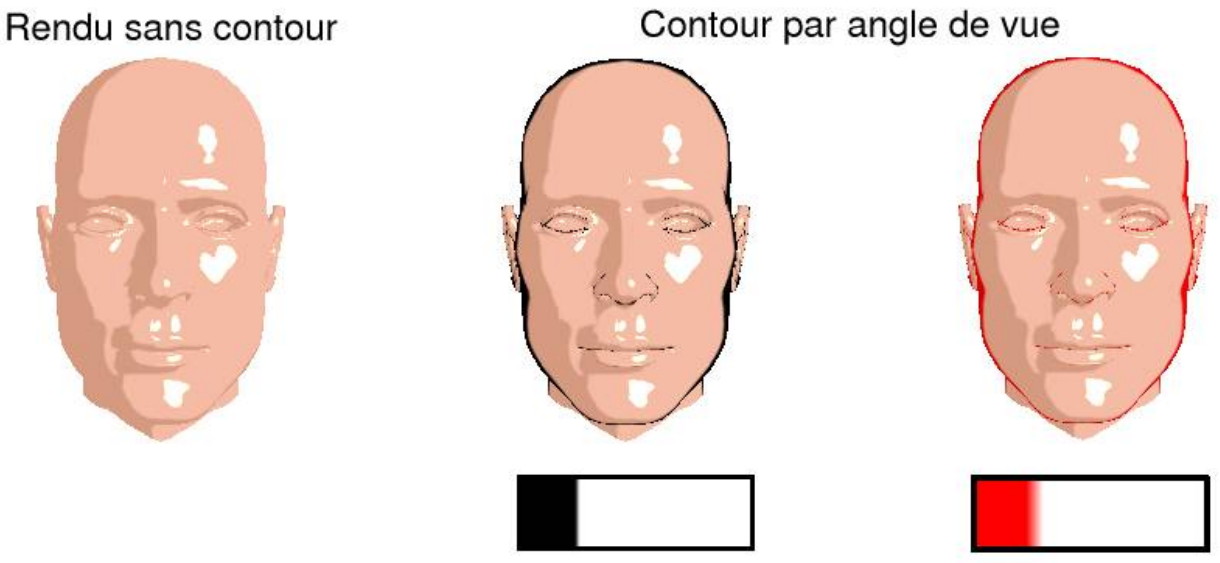
- ***But du rendu de silhouette***
- Il s'agit soit d'imiter un style *bande dessinée* en contourant les objets soit d'améliorer la *lisibilité* d'une figure:
  - en traçant le **contour** des objets,
  - en traçant les **frontières** entre des zones différentes d'un objet,
  - en soulignant le relief en traçant les **crêtes** et les **vallées**.

SILHOUETTE

Rendu de silhouette par angle de vue d'une surface

On *colorie* les zones pour lesquelles la *vue est rasante*
et donc susceptibles de correspondre à une *frontière entre une zone visible et une zone cachée*.

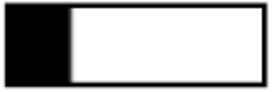
$$||\vec{N} \cdot \vec{V}|| < \epsilon$$



SILHOUETTE

- ***Calcul du rendu de silhouette par angle de vue d'une surface***
  - On calcule le ***produit scalaire du vecteur de vue et de la normale*** à la surface.
  - On accède à une *texture de seuillage* en fonction de la valeur de ce produit scalaire:

```
vec3 silhouette = texture(NdotV);
```



- On combine la couleur de bord avec la couleur de l'objet:

```
CouleurObjet *= silhouette;
```

- Ou on fait “à la main” (sans texture)

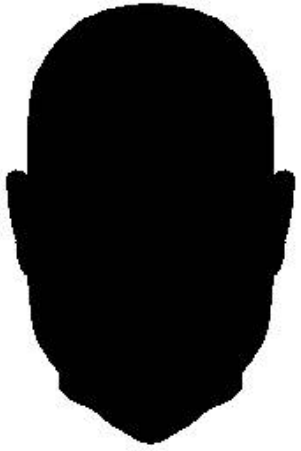
```
Si (N.V) < eps
coul ← bord
Sinon
coul ←Modele(V,L,N)          {modèle
diffus ou spéculaire}
Fin
```

SILOUETTE

Rendu de silhouette par halo

Diffus + speculaire

Silhouette dilatee



Contour par halo

Variantes :
1) Utiliser le stencil shader
2) Utiliser back-face culling

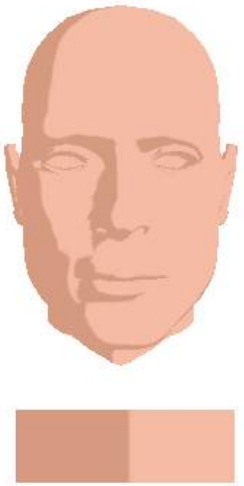
1) On dilate et on affiche en noir (déactiver le tampo de profondeur)
2) on revient à la taille normale et on affiche avec le rendu

TONE SHADING

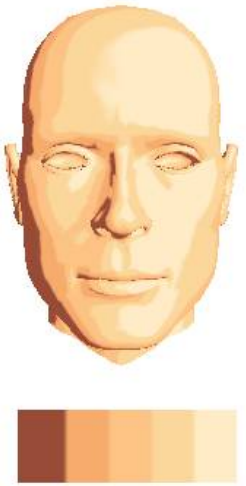
Rendu diffus par à-plats (Tone Shading)

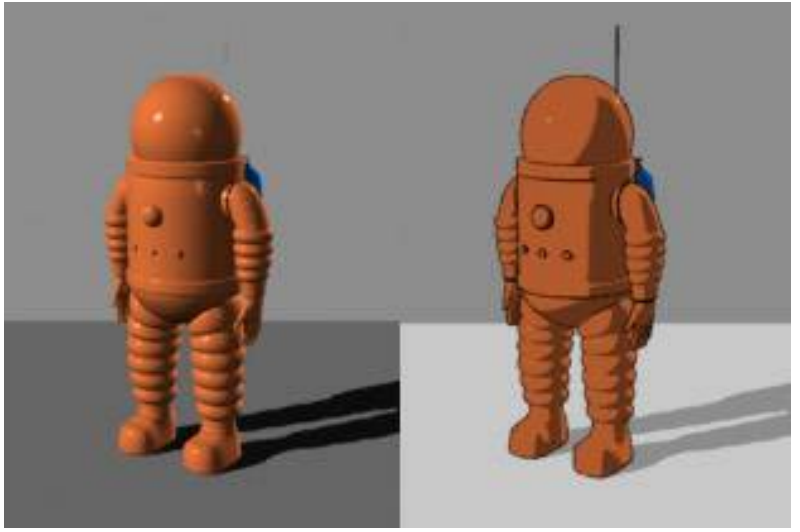
Rendu diffus Gouraud

Rendu 2 tons



Rendu 5 tons





$$||\vec{N}\cdot\vec{V}|| = \text{coord. texure 1D}$$

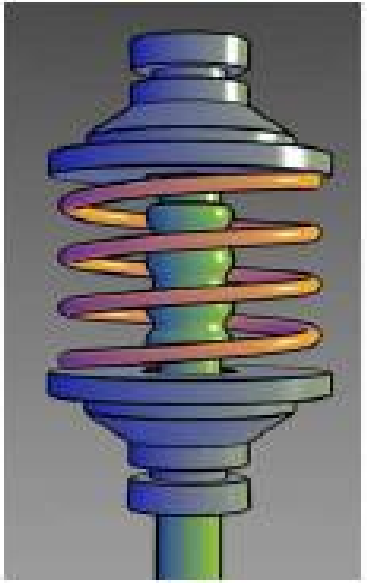
TONE SHADING = toon shader

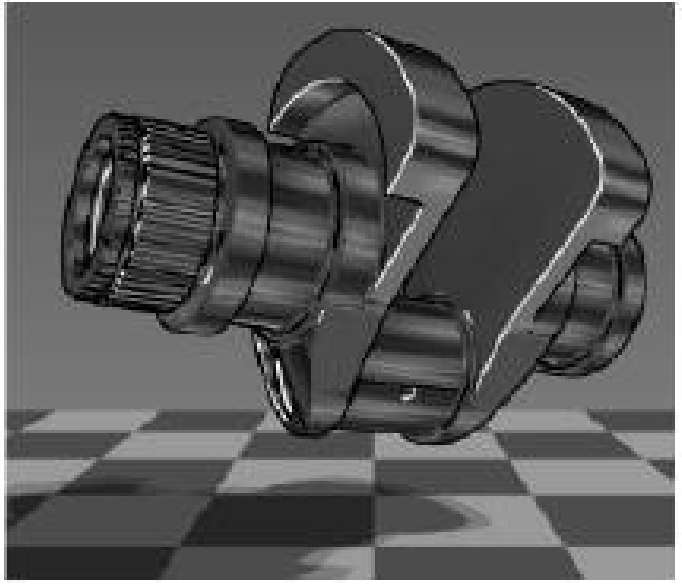
Rendu diffus 2 tons

Rendu diffus 2 tons
avec speculaire

Rendu diffus 2 tons
avec speculaire 2 tons

ILLUSTRATION TECHNIQUE
Éclairage de Gooch





les couleurs chaudes (rouge, l'orange, le jaune) sont perçues très différemment des couleurs froides (le bleu, le violet ou le vert).

$$I = \left(\frac{1 + \vec{l} \cdot \vec{n}}{2} \right) K_{cool} + \left(1 - \left(\frac{1 + \vec{l} \cdot \vec{n}}{2} \right) \right) K_{Warm}$$

Technique d'illumination non-photoréaliste avec modèle d'illumination nonstandard décrite par Amy Gooch, Bruce Gooch, Peter Shirley et Elaine Cohen (1998).

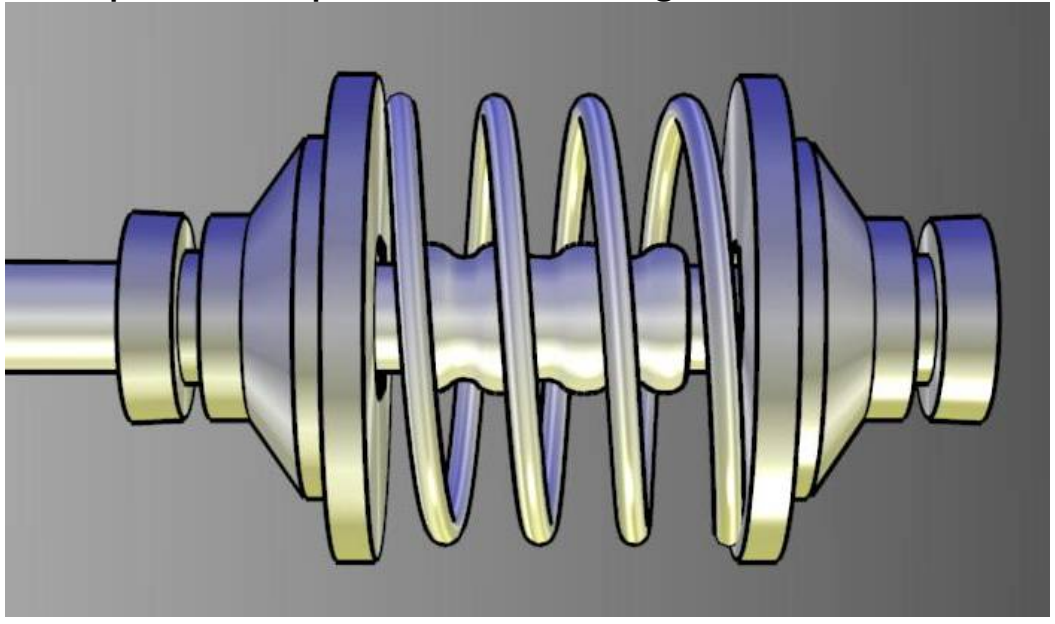
GOOCH

Éclairage de Gooch

Amy Gooch - Bruce Gooch - Peter Shirley - Elaine Cohen

A Non-Photorealistic Lighting Model For Automatic Technical Illustration.

SIGGRAPH 1998, Computer Graphics Proceedings



GOOCH

Méthode

1) On **remplace la lumière diffuse de l'objet** par des variations dans la luminosité et des variations sur le ton (*hue*)
2) **On combine la *couleur de l'objet***
   avec des couleurs prises dans une *palettes* de couleurs *chaudes* à *froides*
   en fonction de l'intensité de la lumière diffuse.

Deux exemples de palettes de tons froids -> chauds

Construction de la palette et utilisation

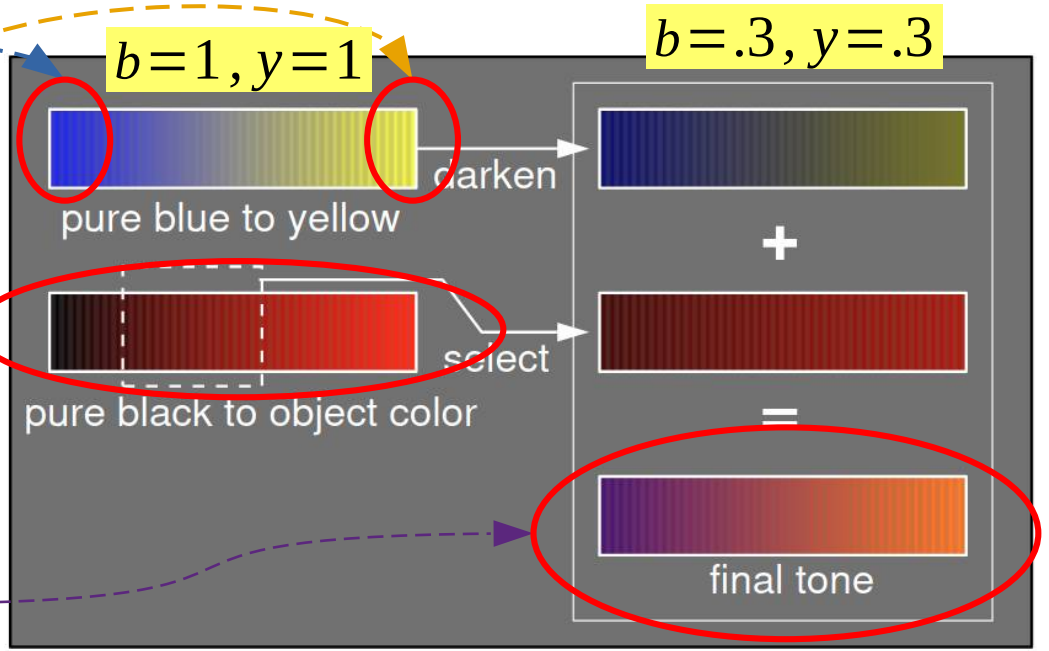
$$K_{blue} = (0,0,b), b \in [0,1]$$
$$K_{yellow} = (y,y,0), y \in [0,1]$$

$$k_{cool} = k_{blue} + \alpha \; k_d$$
$$k_{warm} = k_{yellow} + \beta \; k_d$$

k_d = couleur diffuse de l'objet (r,v,b)

$$I = \left(\frac{1+\vec{l}\cdot\vec{n}}{2}\right) K_{cool} + \left(1 - \left(\frac{1+\vec{l}\cdot\vec{n}}{2}\right)\right) K_{Warm}$$

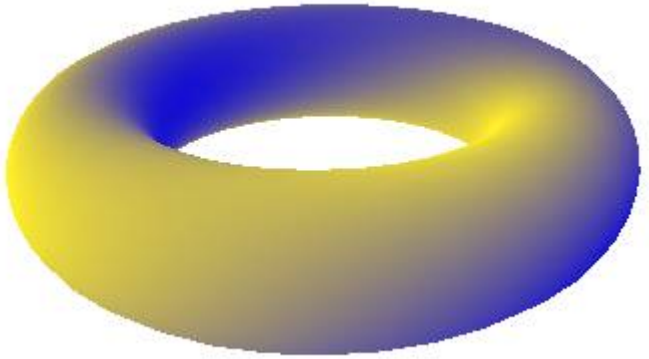
$$\vec{l}\cdot\vec{n} \in [-1,1] \Rightarrow \frac{1+\vec{l}\cdot\vec{n}}{2} \in [0,1]$$
$$I \in [k_{cool}, k_{warm}]$$



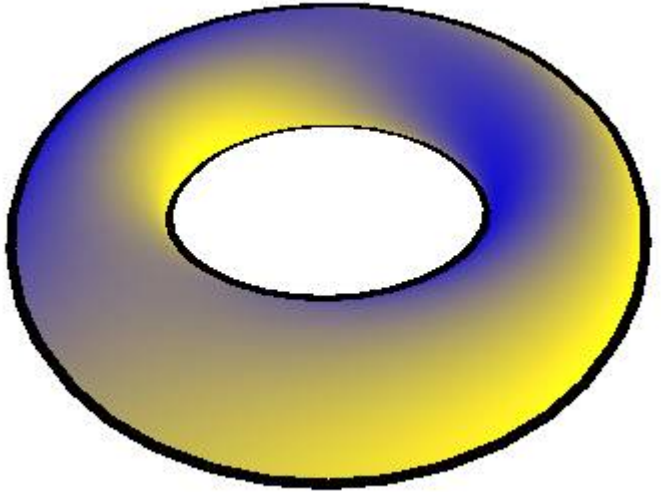
b, y, α, β : paramètres du modèle à définir

$\vec{l}$ = vecteur unitaire, direction de la lumière
$\vec{n}$ = vecteur normal unitaire

Exemple



Sans contour



Avec contour

Exmples de composition

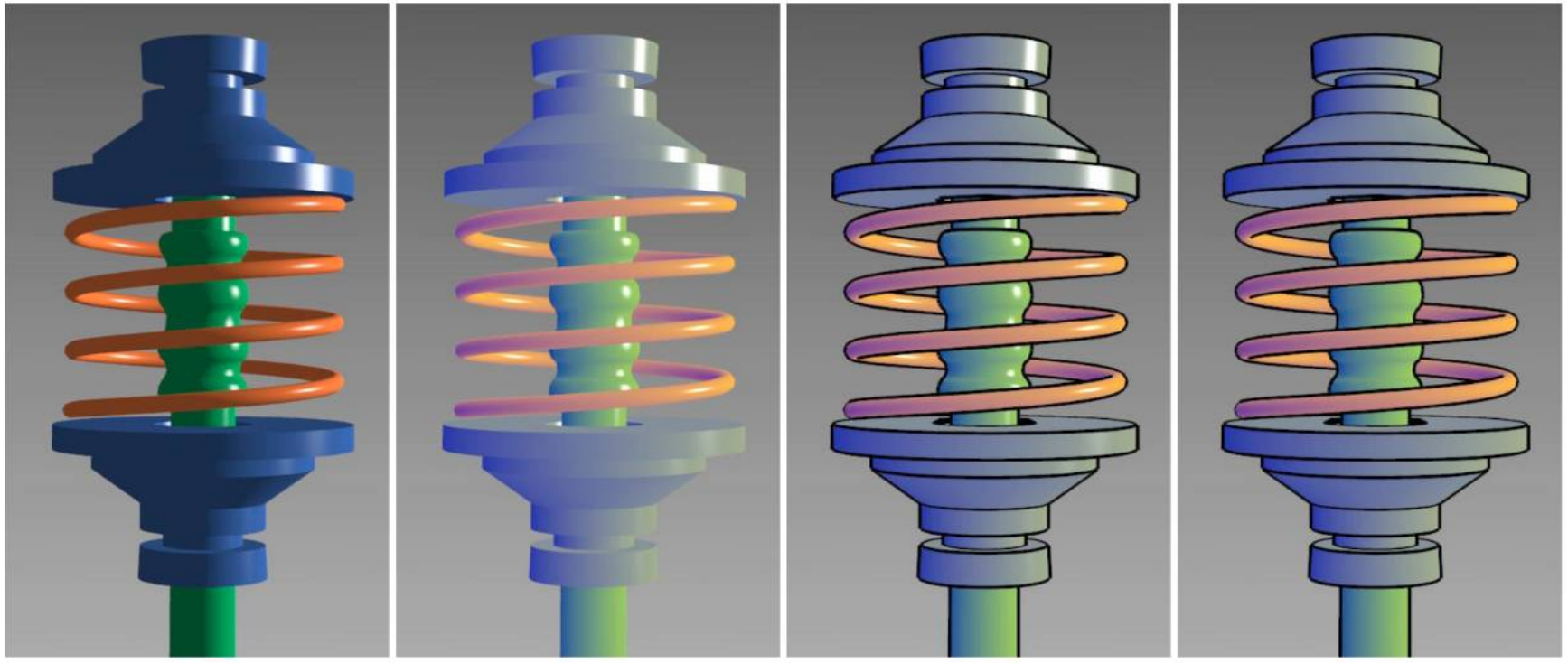
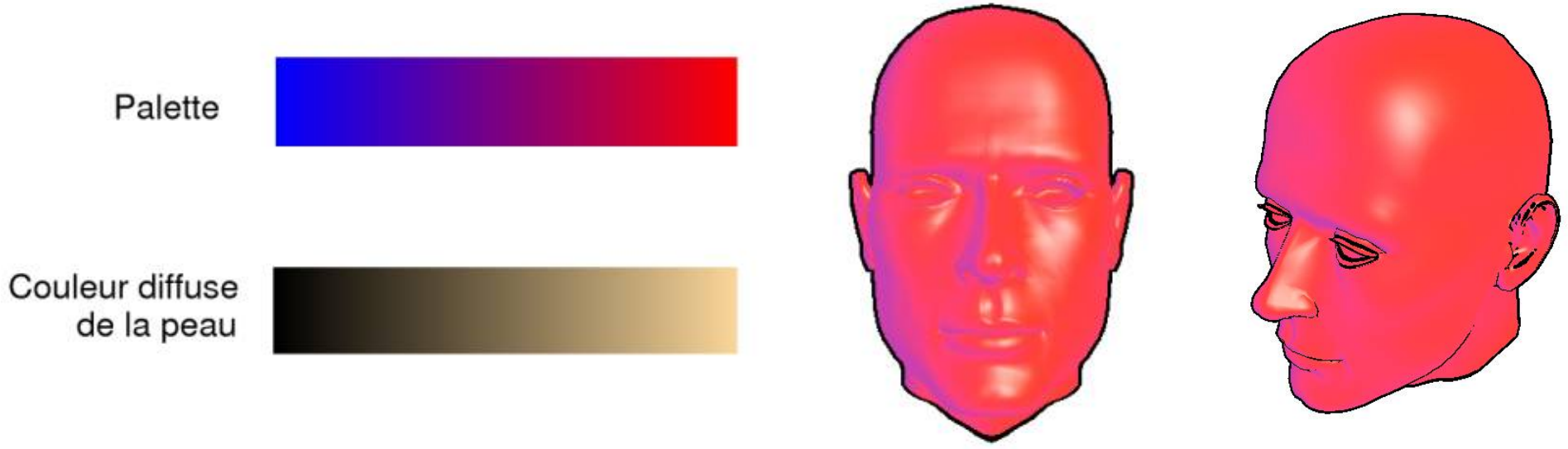


Figure 11: Left to Right: a) Phong model for colored object. b) New shading model with highlights, cool-to-warm hue shift, and without edge lines. c) New model using edge lines, highlights, and cool-to-warm hue shift. d) Approximation using conventional Phong shading, two colored lights, and edge lines.

GOOCH

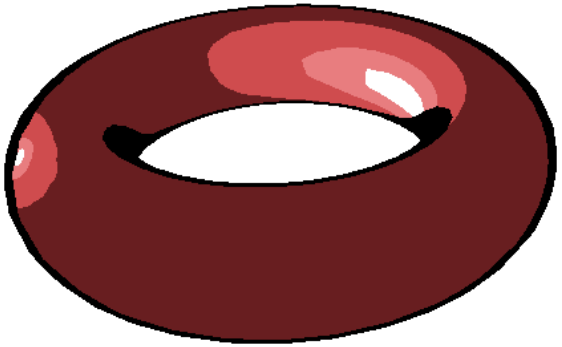
Exemple



Traitement d'image pour l'augmentation de la morphologie

Environment Map

- Exemple



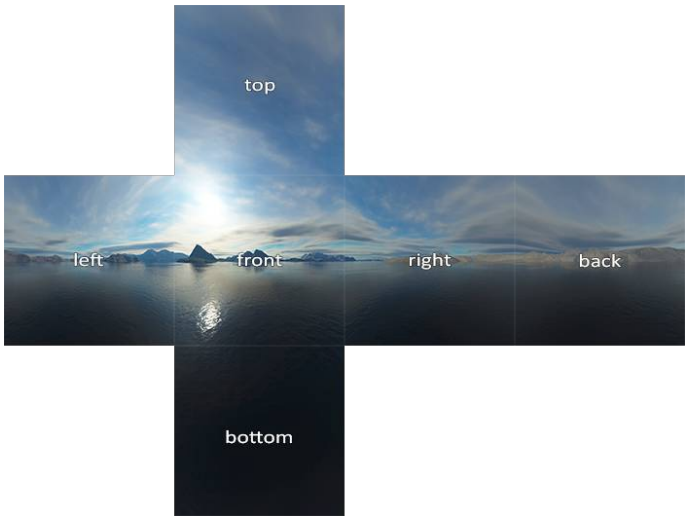
toon shader

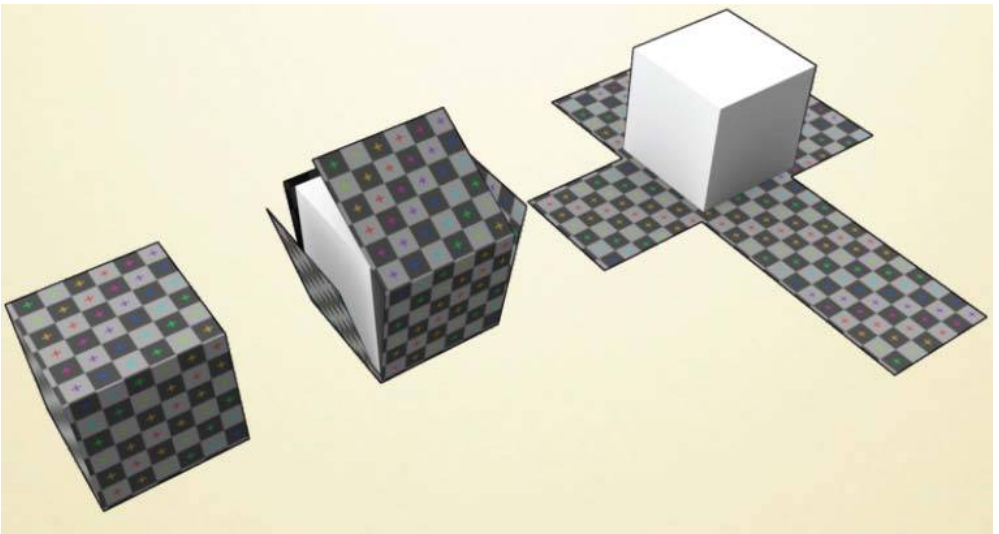
Environment map

Mixte des deux

En partie tiré du cours de Emmanuel Agu

Environment map : principe





L’environnement est enregistré dans 6 images = skybox
skybox = 6 images prises suivant des directions perpendiculaires

Environment map : principe

- Le centre de la skybox = repère de la scène

Pour chaque sommet :

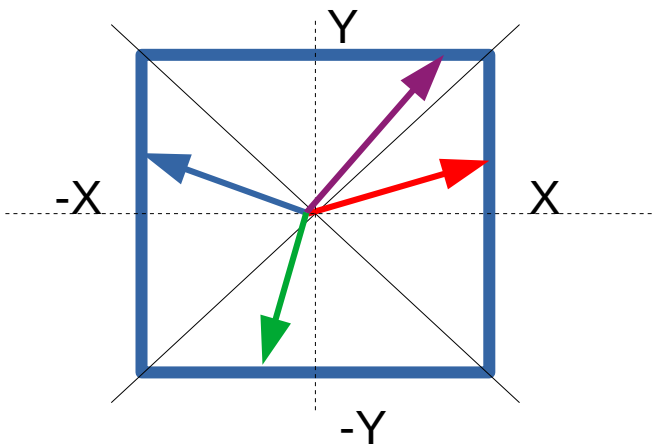
- on calcule le rayon réfléchi
- on calcule les coordonnées (u,v) de la texture

Environment map : en détail

Calcul du rayon réfléchi $\vec{R} = 2(\vec{N} \cdot \vec{V})\vec{N} - \vec{N}$

La plus grande composante de ***R*** détermine la face du cube intersecté

Exemple en 2D



|X| > |Y| et X > 0

|X| > |Y| et X < 0

|Y| > |X| et Y > 0

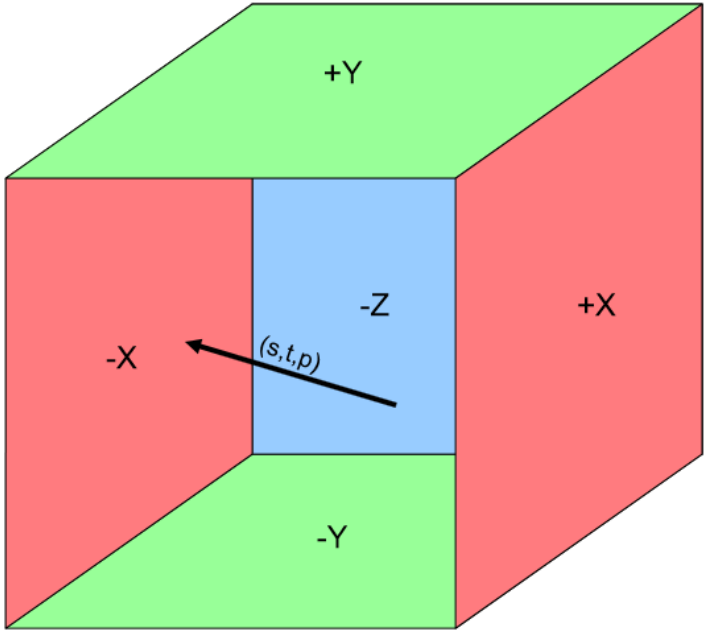
|Y| > |X| et Y < 0

Les autres composantes donnent les coordonnées de texture

Pour l'exemple en rouge t = |Y|/|X|

Environment map : en détail

Cube Map Texture Lookup:
Given an (s,t,p) direction vector , what (r,g,b) does that correspond to?



- Let L be the texture coordinate of (s, t, and p) with the largest magnitude
- L determines which of the 6 2D texture "walls" is being hit by the vector (-X in this case)
- The texture coordinates in that texture are the remaining two texture coordinates divided by L: (a/L,b/L)

Built-in GLSL functions

```
vec3 ReflectVector = reflect( vec3 eyeDir, vec3 normal );

vec3 RefractVector = refract( vec3 eyeDir, vec3 normal, float Eta );
```

Environment map : en détail

- R = (-4, 3, -1)
- Même direction que R = (**-1**, 0.75, -0.25)

=> il faut utiliser la texture de la face x = -1

- Les coordonnées de texture sont (u=) y = 0.75, (v=) z = -0.25
- Mais attention :
  - le cube est défini par x, y, z = ± 1
  - Les coordonnées de texture sont définies dans [0, 1]

=> Il faut réajuster [-1 1] vers [0,1] => f(new) = 1/2 old + 1/2

u = ½ + ½ y, u = ½ + ½ z

=> u =0.875, v = 0.375

Mais en GLSL c'est bien plus simple

Environment map : implémentation openGL

- Coté CPU
  - Il faut définir une texture de type « CUBE MAP »
  - L’envoyer au GPU
- Coté shader
  - Calculer le rayon réfléchi
  - En déduire les coordonnées de texture

Environment map : implémentation openGL - Coté CPU

- Créer un objet texture

```
glGenTextures(1, &tex);
glActiveTexture(GL_TEXTURE1); // ici on choisit la texture unit 1
glBindTexture(GL_TEXTURE_CUBE_MAP, tex);
```

- Charger les 6 images (à faire pour chaque image)

```
front  = glmReadPPM("./texture/front.ppm", &iwidth, &iheight);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z ,0,3,iwidth,iheight,0,GL_RGB
,GL_UNSIGNED_BYTE,  front );
```

Environment map : implémentation openGL - Coté CPU

Pour les autres images avec la correspondance suivante

- GL_TEXTURE_CUBE_MAP_POSITIVE_X = right
- GL_TEXTURE_CUBE_MAP_NEGATIVE_X = left
- GL_TEXTURE_CUBE_MAP_POSITIVE_Y = top
- GL_TEXTURE_CUBE_MAP_NEGATIVE_Y = bottom
- GL_TEXTURE_CUBE_MAP_POSITIVE_Z = front
- GL_TEXTURE_CUBE_MAP_NEGATIVE_Z = back

Environment map : implémentation openGL - Coté CPU

- Définir les paramètres de la texture comme d'hab (et après l'avoir bindé)

```
glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_MAG_FILTER,GL_NEAREST );
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
```

Environment map : implémentation openGL - Coté CPU

- Lier la texture à la **variable** du fragment shader

```
GLuint texMapLocation;
texMapLocation = glGetUniformLocation(programID, "texMap");

glUseProgram(programID);
// le paramètre « 1 » doit correspondre à la unit texture choisie (cf cours sur texture)
glUniform1i(texMapLocation, 1);
```

Environment map : implémentation openGL - Coté GPU

- Vertex shader : on calcule le rayon réfléchi

```
out vec3 R ;
...
// normal = normal à l'objet dans le repère de l'objet
// rayonIncident = (position de la caméra – position vertex ) dans le repère de la scène
NormaleTransf = normalize(transpose(inverse(mat3(MODEL)))*normal) :
R = reflect(rayonIncident, NormaleTransf);
```

Environment map : implémentation openGL - Coté GPU

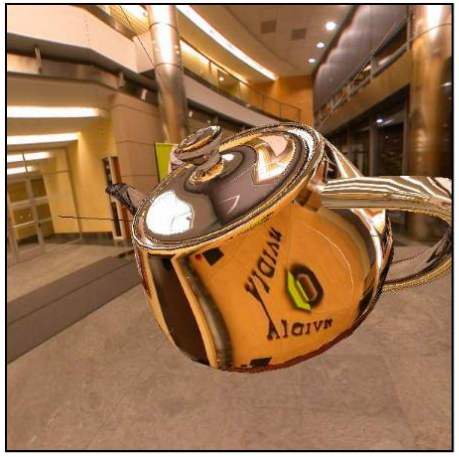
- Fragment shader :

```
//récupérer la texture Cube map
uniform samplerCube texMap;
//récupérer du rayon réfléchi pour le fragment
in vec3 R ;
...
// on récupère la valeur de la texture à l'aide du rayon réfléchi
vec4 texColor = textureCube(texMap, R);
finalColor = texColor; // on affecte la couleur en sortie
FinalColor =mix(texColor,vec4(colorRes,1.),mixCoeff);   // ou on mélange avec un autre rendu
```

Environment map : réfraction

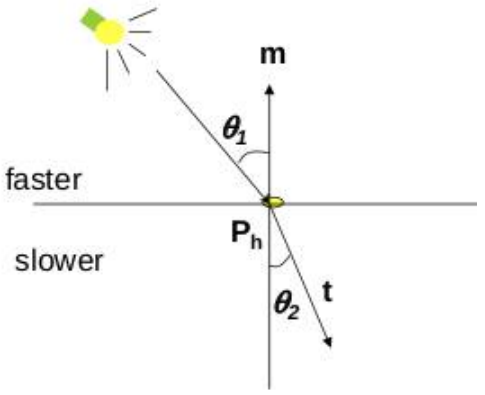
Peut être utilisé pour la réfraction (transparence)

Réflexion



Réfraction

Il suffit d'utiliser le rayon réfracté
=> loie de Snell



En GLSL (vertex shader)
T = refract(rayonIncident, NormaleTransf, rapportIndiceReffract);

Exemple

réflexion

Réfraction avec un ratio d’indice = 0.7

Réfraction avec un ratio d’indice = 1

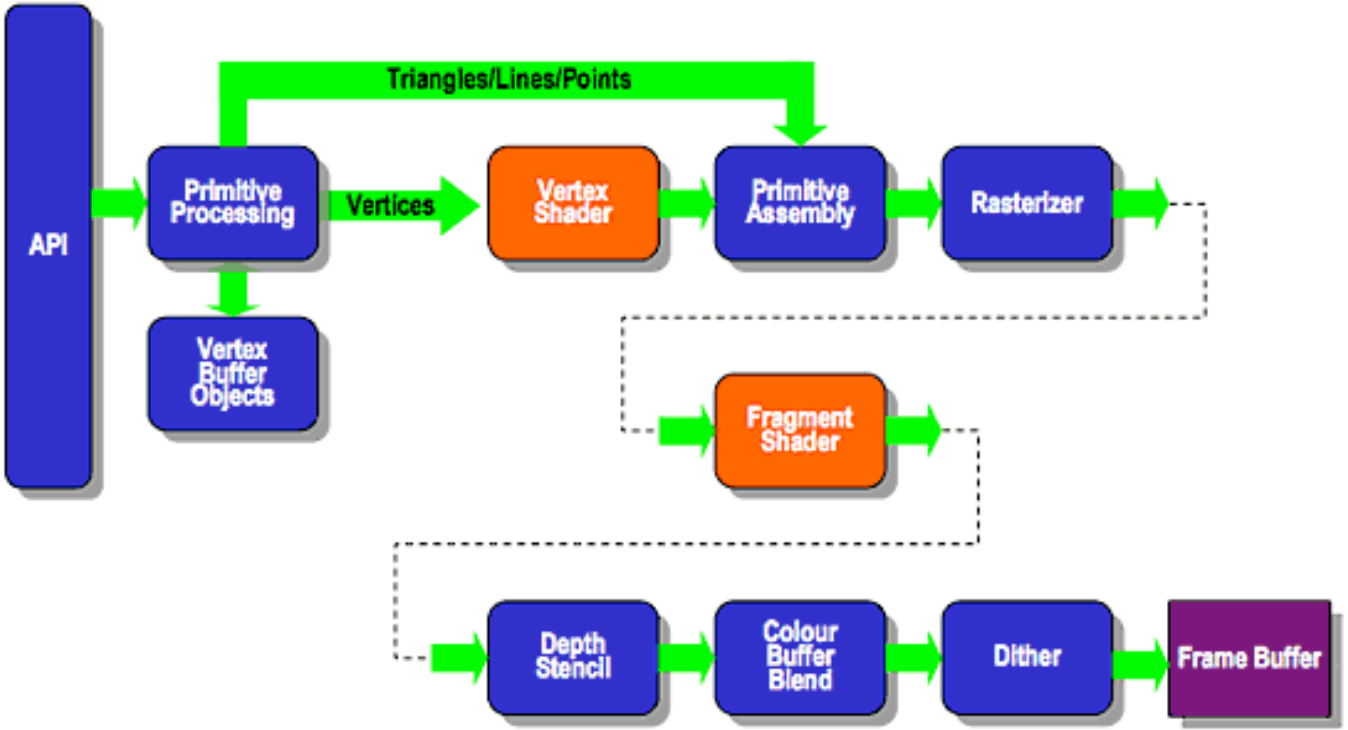
Question / Exercice

- Quels sont les limites de cette approche ?
- Afficher la « skybox ».

Deux mots sur les frameBuffers

- Ils sont utilisés pour faire des rendus en plusieurs passes
- On calcule un premier rendu dans une frame buffer et il est stocké dans une texure
- On calcule un deuxième rendu utilisant le résultat du 1er.
- Exemples d’application
  - Ombres portées
  - miroir

- Le pipeline dynamique

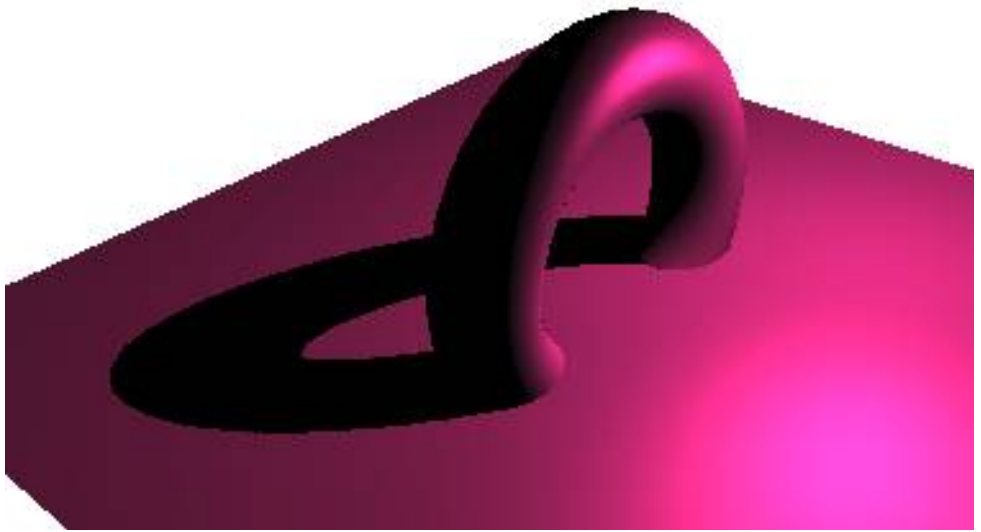


https://alexandre-laurent.developpez.com/tutoriels/OpenGL/OpenGL-GLSL/?page=page_1#L1-A-1

Rendu dans une texture

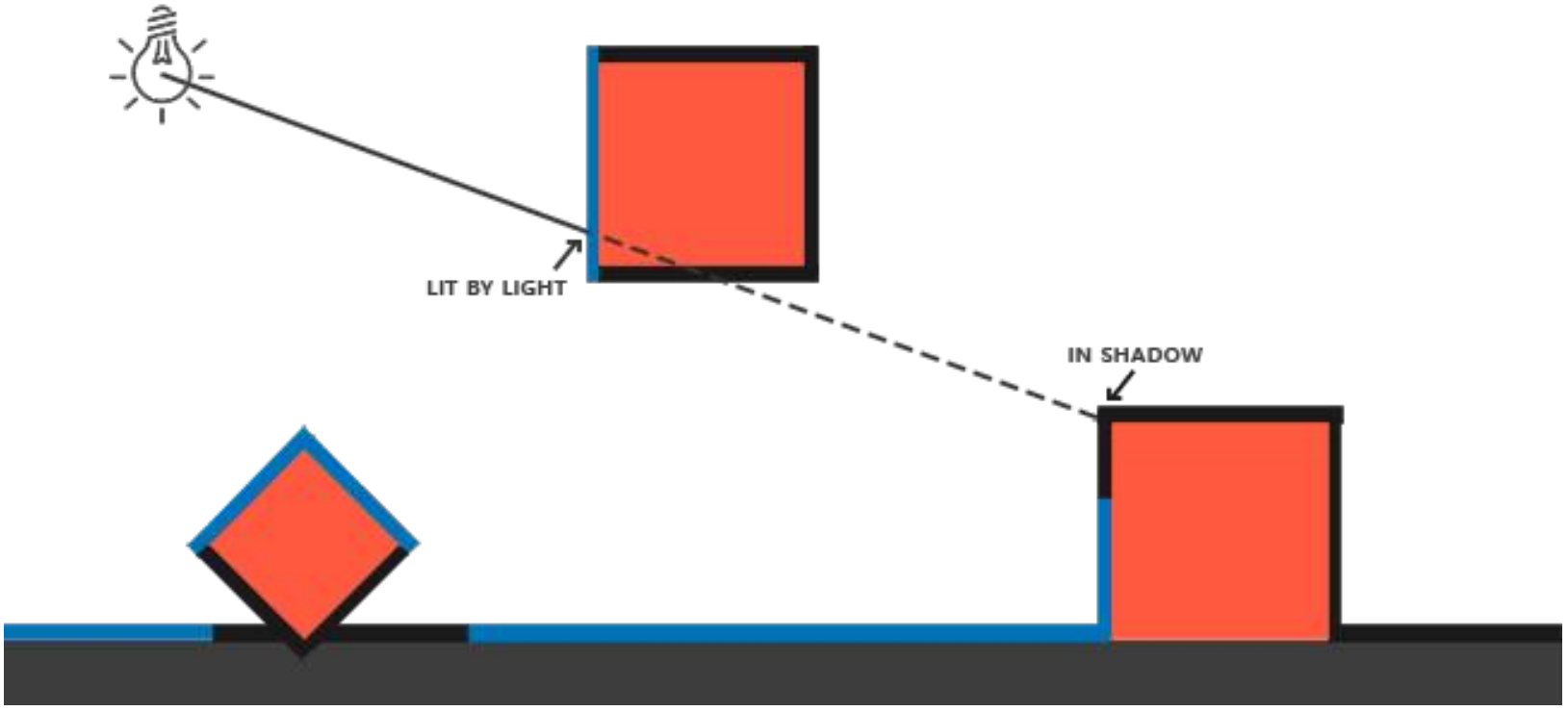
- Principe
  - Au lieu de faire le rendu à l'écran (écriture dans le FrameBuffer par défaut)
  - Il est fait dans une texture = texture attachée au « FrameBuffer »
- FrameBuffer =
  - Conteneur pour les textures
  - Avec éventuellement un Z- buffer
- Application
  - Ombres portées = Shadow Mapping
  - miroir

Shadow Mapping



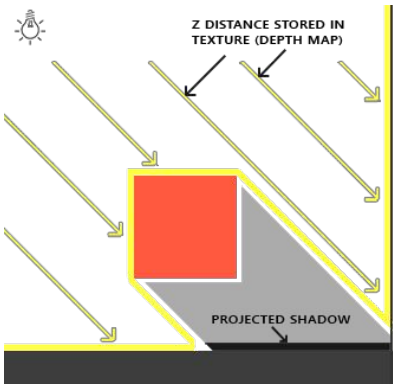
D’après https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Shadow Mapping : idée



https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Shadow Mapping : principe



Rendu en deux passes

Un shader (vertex et fragment) par passe

- Passe 1 :
  - On calcule l'image de profondeur du point de vue de la lumière
  - On stocke le résultat dans une texture « **de profondeur** »
- Passe 2 :
  - On fait le rendu du point de vue de la caméra
  - Pour chaque fragment on calcule ses coord. dans le repère de la lumière
  - Si la coord en **Z (profondeur) > texture de profondeur** => **dans l'ombre**

https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Shadow Mapping : passe 1

Rendu dans une texture

- On crée un framebuffer pour le rendu

```
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

- On crée une texture pour stocker le rendu :

```
GLuint depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
             SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
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);
```

Nb de composantes
On a besoin
que de la
profondeur

Format
=
profondeur

Ces paramètres
seront utiles
pour lire la texture

Shadow Mapping : passe 1

Rendu dans une texture

- On attache la texture au framebuffer (fct **`GlFramebufferTexture2D`**)

```
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
GlFramebufferTexture2D( GL_FRAMEBUFFER,
                        GL_DEPTH_ATTACHMENT,
                        GL_TEXTURE_2D, depthMap, 0);
glDrawBuffer(GL_NONE);

glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

On attache que le tampon profondeur

On précise qu'on a pas besoin de faire Le rendu des couleurs

Ainsi le rendu fait dans le framebuffer sera automatiquement stocké dans la texture

Shadow Mapping : passe 1

Faire le rendu

```
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glClear(GL_DEPTH_BUFFER_BIT);
//    Configurer les shaders et matrices
...
//  lancer le rendu
...
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Shadow Mapping : passe 1

Pour faire le rendu :

configurer les matrices pour le shader de la depth Map

- On fait simple = projection orthogonale dans la direction des la lumière

On essaie d'englober tous les objets de la scène

```
glm::mat4 lightProjection = glm::ortho(-10., 10., -10., 10.,
                                       near_plane, far_plane);

glm::mat4 lightView = glm::lookAt(LightLocation,
                                  glm::vec3( 0.0f, 0.0f,  0.0f),
                                  glm::vec3( 0.0f, 1.0f,  0.0f));

glm::mat4 lightSpaceMatrix = lightProjection * lightView;
```

Rq : on vise le centre de la scène

Shadow Mapping : passe 1

Enfin les shaders

- Vertex shader

  on calcule la position du sommet dans le repère de la lumière

```
#version 450
layout (location = 0) in vec3 aPos;

uniform mat4 lightSpaceMatrix;
uniform mat4 model;

void main()
{
    gl_Position = lightSpaceMatrix * model * vec4(aPos, 1.0);
}
```

Avant, dans le pg Opengl, il ne faudra pas oublier d'envoyer les matrices au shader

Shadow Mapping : passe 1

Enfin les shaders

- Fragment shader

  encore plus simple, i.e. rien à faire : on sort la valeur z du fragment

```
#Version 450

void main()
{
    // gl_FragDepth = gl_FragCoord.z;
}
```

Est fait par défaut
donc même inutile

Shadow Mapping : passe 2

Le rendu

```
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Configurer les shaders et matrices (Comme d'hab)
...
glBindTexture(GL_TEXTURE_2D, depthMap);
// lancer le rendu (Comme d'hab)
...
```

Shadow Mapping : passe 2

Le vertex shader :

```
#version 450
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 FragPos;
out vec3 Normal;
out vec4 FragPosLightSpace;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;
uniform mat4 lightSpaceMatrix;

void main() {
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = transpose(inverse(mat3(model))) * aNormal;
    FragPosLightSpace = lightSpaceMatrix * vec4(FragPos, 1.0);
    gl_Position = projection * view * vec4(vs_out.FragPos, 1.0);
}
```

Comme d'hab', mais en plus, on calcule la position du sommet dans le repère de la lumière

Shadow Mapping : passe 2

Le fragment shader :

Avant,
il faut faire le nécessaire pour récupérer la texture de profondeur
Les variables uniformes utiles

```
...
uniform sampler2D shadowMap;
uniform vec3 lightPos;
uniform vec3 viewPos;
void main()
{
// calcul de l'ombrage
    // passage des coord. Homogènes en coord. cartésiennes
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // transform de [-1,1] depth map vers [0,1] texture
    projCoords = projCoords * 0.5 +  vec3(0.5,0.5,0.5);
    // récup de la profondeur la + proche à partir des coord. Xy du fragment
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    // on détermine si le fragment est dans l'ombre
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;
```

La fct « Texture » retourne un gvec4
On récupère l'unique 1er composant (« r ») contenant la profondeur

Shadow Mapping : passe 2

Le fragment shader (suite) :

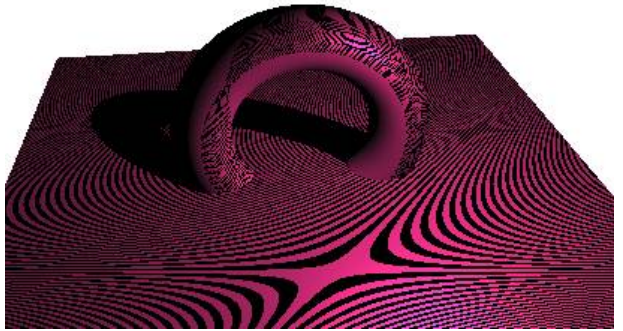
```
...

    // calculate shadow
    float shadow = ShadowCalculation(fs_in.FragPosLightSpace);
    vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color;

    FragColor = vec4(lighting, 1.0);
}
```

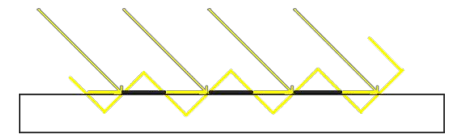
On intègre l'ombre dans le calcule d'éclairement.
shadow = 1 => ambient uniquement
Shadow =0 => comme d'hab'

Shadow Mapping : passe 2

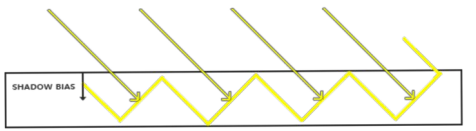
Le fragment shader : Petit pb => petite correction

Effet de Moiré





La discrétisation
=> un coup au dessus
Un coup en dessous



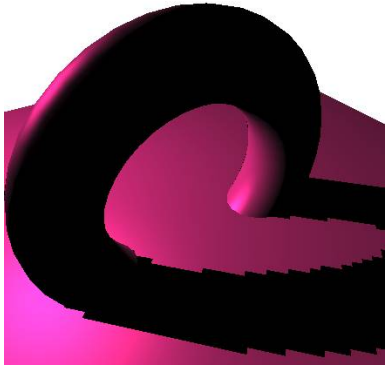
On tient compte du biais

```
// de façon basique biais = constante
    float bias = 0.005;
// ou encore mieux en tenant compte de l'angle de la lumière avec la facette
    float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);

float shadow = currentDepth - bias > closestDepth  ? 1.0 : 0.0;
```

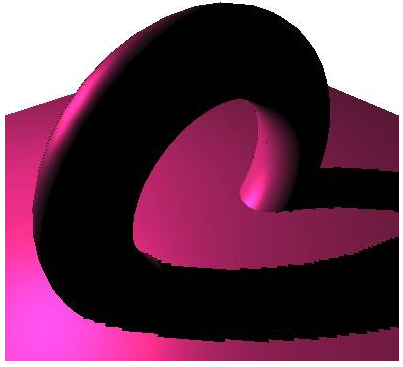
Shadow Mapping : amélioratio,

Résolution de la texture de profondeur = > Alliasing d'ombrage

Depth map de 128 x 128



Depth map de 256 x 256



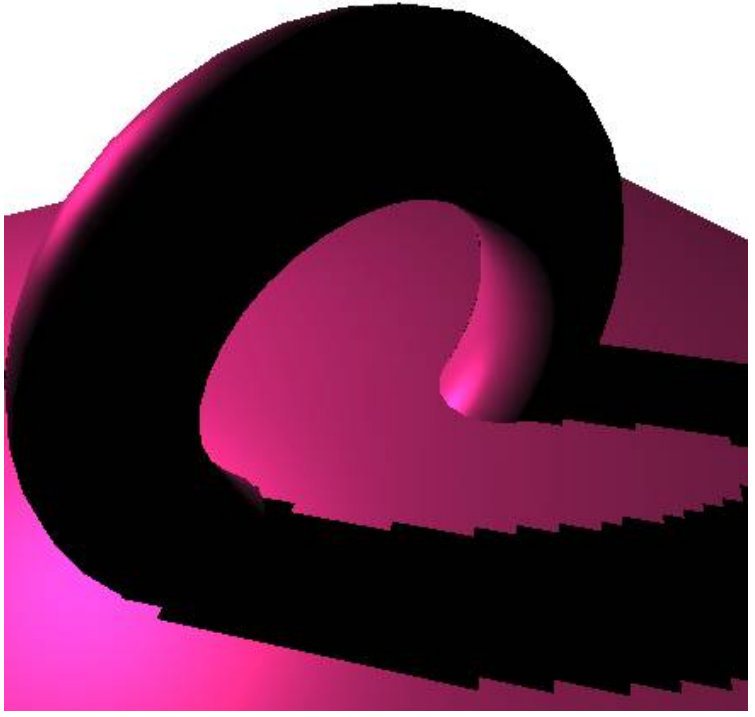
Peut être atténué par un filtre PCF (Percentage closer filtering)

```
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
// on compte le nb de textels du 1 voisinage plus loin
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; // on fait la moyenne
```

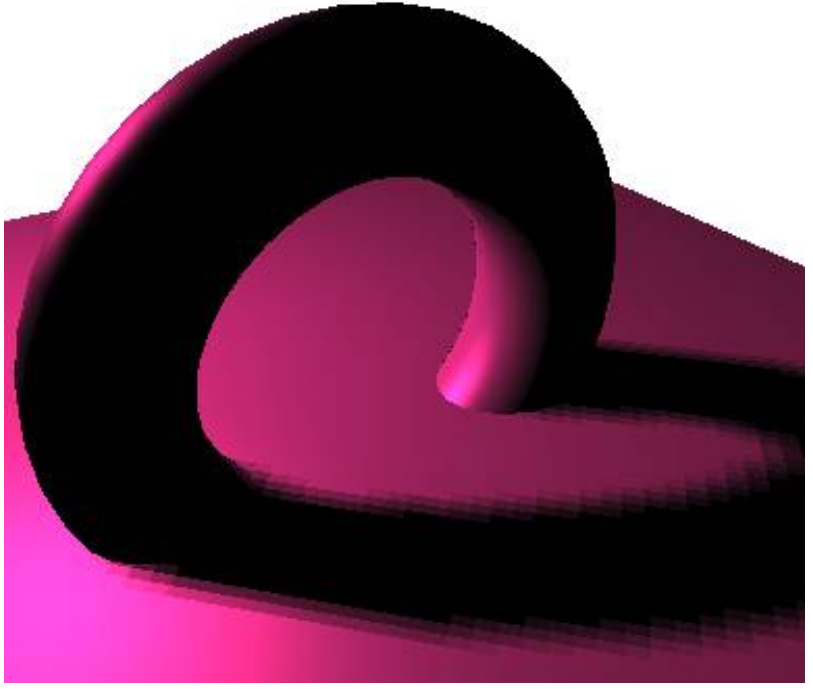
Taille d'un texel

Shadow Mapping : amélioratio,

Résolution de la texture de profondeur = > Alliasing d’ombrage



Depth map de 128 x 128
Sans PCF



Depth map de 128 x 128
avec PCF