

Rendu avancé : Illumination, Shaders, Textures, VAO

Sébastien Beugnon

25 novembre 2021

Sébastien Beugnon

R&D Researcher

mail : sebastien.beugnon@emersya.com

Github : [@sbeugnon](#)



Emersya

Solutions d'e-commerce et de création collaborative

- ▶ Visionneuse 3D multi-plateforme
- ▶ Configurateur de produits
- ▶ Plateforme de gestion de modèles 3D

Sams^onite

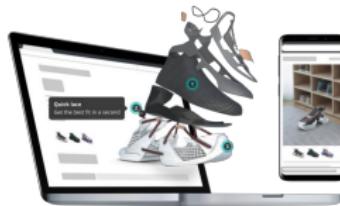
vitra.

swatch[®] 

Exploiter la 3D de l'idéation du produit à la vente et à la production



Conception produit



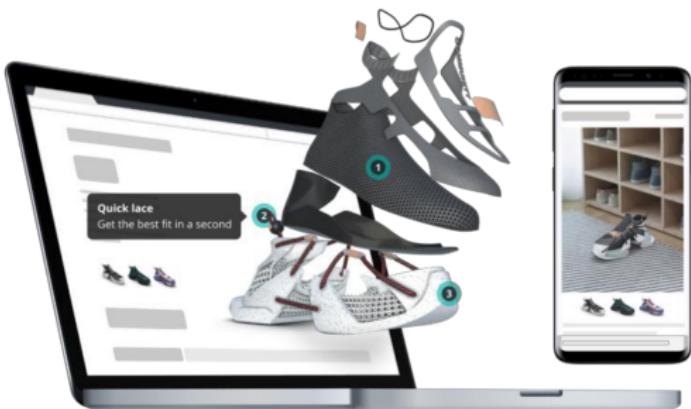
e-Commerce



Configuration de masse

Les usages de la 3D

- ▶ E-Commerce
- ▶ Industrie
- ▶ Jeu vidéo
- ▶ Cinéma
- ▶ Médical
- ▶ Géologie



Les usages de la 3D

- ▶ E-Commerce
- ▶ Industrie
- ▶ Jeu vidéo
- ▶ Cinéma
- ▶ Médical
- ▶ Géologie



Horizon Zero Dawn : Frozen wilds (2017).

Les usages de la 3D

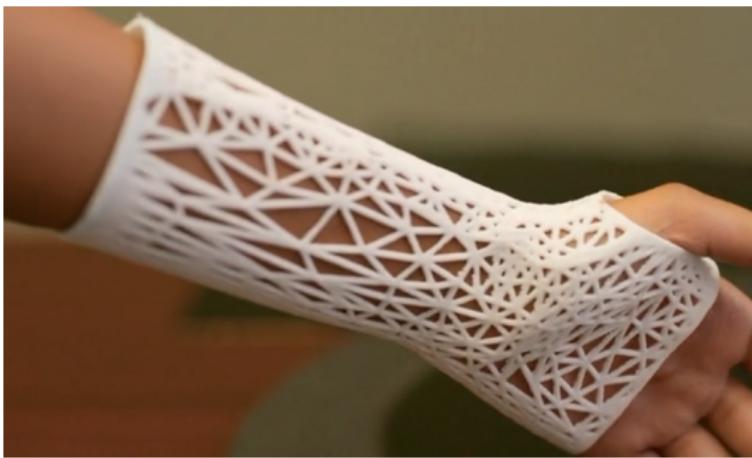
- ▶ E-Commerce
- ▶ Industrie
- ▶ Jeu vidéo
- ▶ Cinéma
- ▶ Médical
- ▶ Géologie



Le livre de la jungle (2016).

Les usages de la 3D

- ▶ E-Commerce
- ▶ Industrie
- ▶ Jeu vidéo
- ▶ Cinéma
- ▶ Médical
- ▶ Géologie



Impression 3D à usage médical.

2 types de rendu

Rendu pré-calculé (Baked)

- ▶ Images, animation, FX
- ▶ Requiert des minutes, des heures, de rendu pour une image



2 types de rendu

Rendu en temps réel (Realtime)

- ▶ Visionneuse, jeux vidéo, simulations
- ▶ Vitesse de rendu est cruciale (FPS)



Contexte
ooooo●

Théorie de la lumière
oooooooooo

Rendu
oooooooooo

Programmation
oooooooooooo

Conclusion
oooo

TP
oo

Sommaire

Contexte

Contexte

Théorie de la lumière

Rendu

Projection & Caméra

Programmation

API Graphique

OpenGL

Conclusion

TP

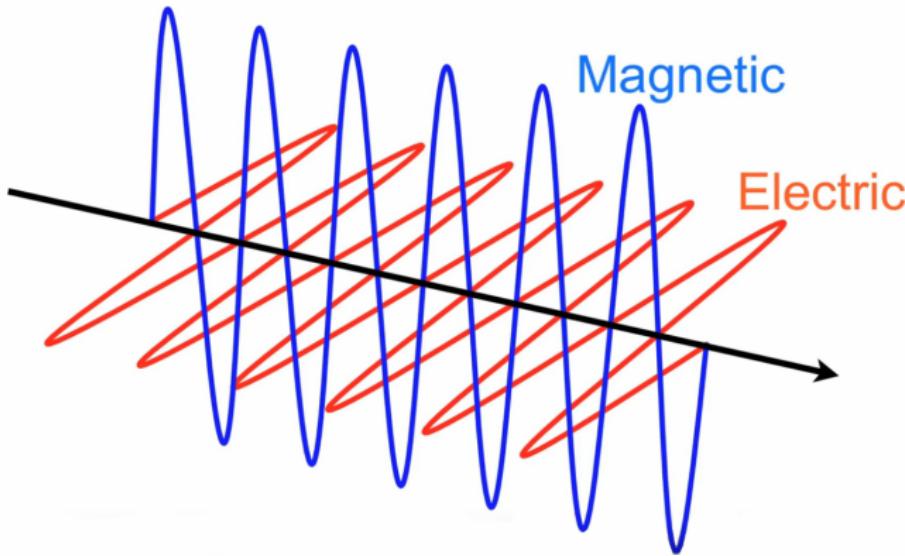
Théorie de la lumière

Physique de la lumière

- ▶ C'est quoi la lumière ?

Physique de la lumière

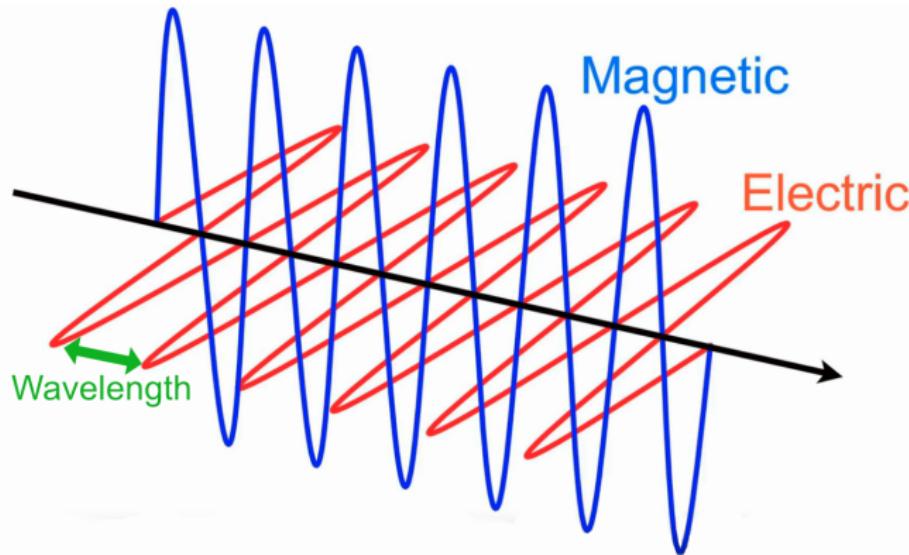
- ▶ C'est quoi la lumière ?
 - ▶ Une onde électro-magnétique



T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, S. Hillaire
Real-Time Rendering, Fourth Edition.
CRC Press, 2018

Physique de la lumière

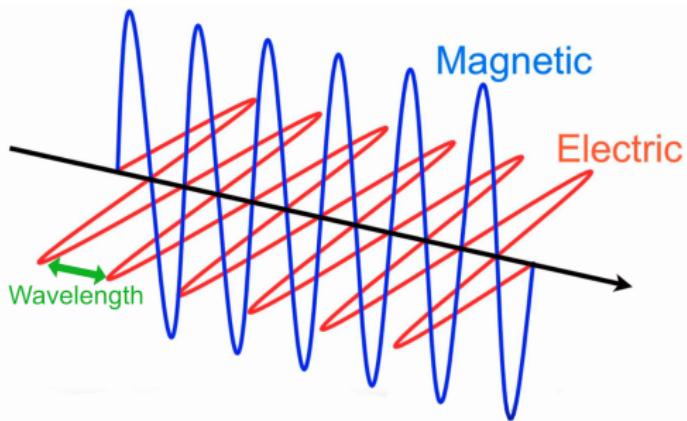
- ▶ C'est quoi la lumière ?
 - ▶ Une onde électro-magnétique



T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, S. Hillaire
Real-Time Rendering, Fourth Edition.
CRC Press, 2018

Physique de la lumière

- ▶ C'est quoi la lumière ?
 - ▶ Une onde électro-magnétique



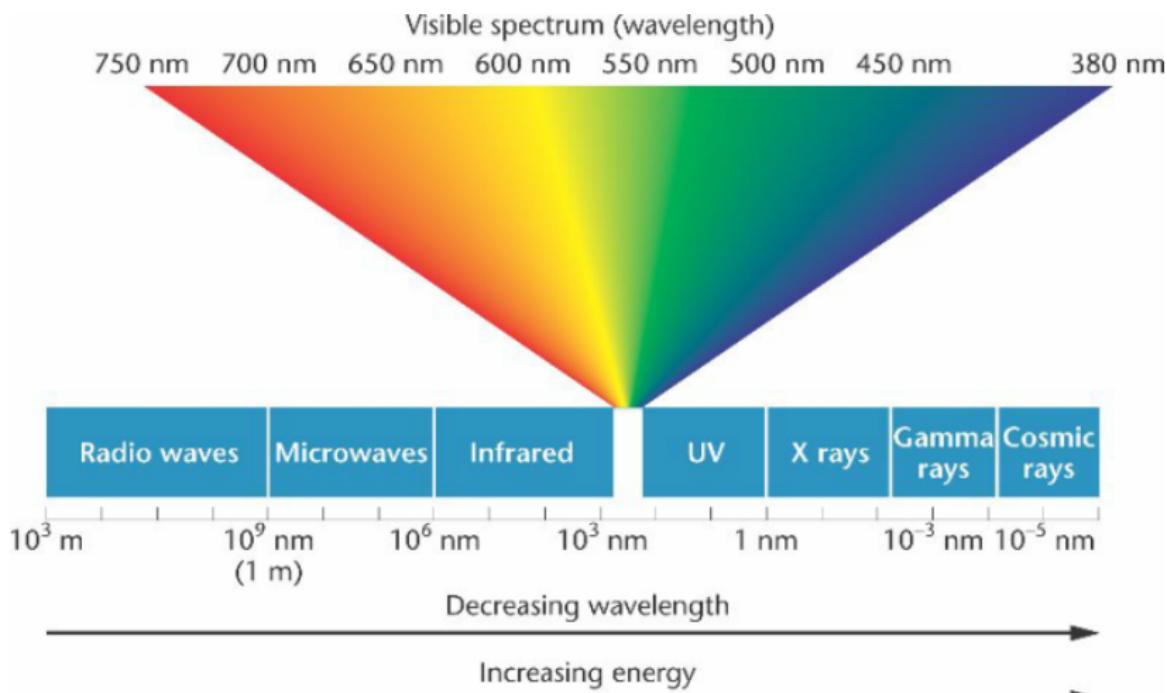
Définition

- ▶ Onde transversale
- ▶ Fréquence
- ▶ Longueur d'onde λ (nm)
- ▶ Irradiance (Energie)



T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, S. Hillaire
Real-Time Rendering, Fourth Edition.
CRC Press, 2018

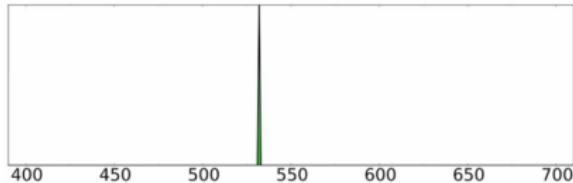
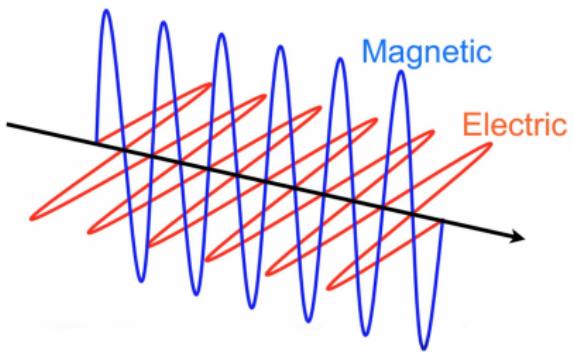
Physique de la lumière



Physique de la lumière

Laser

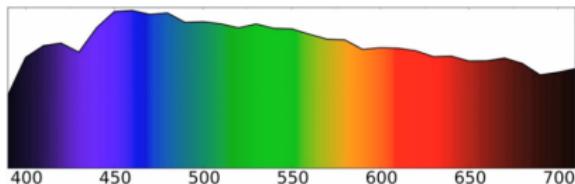
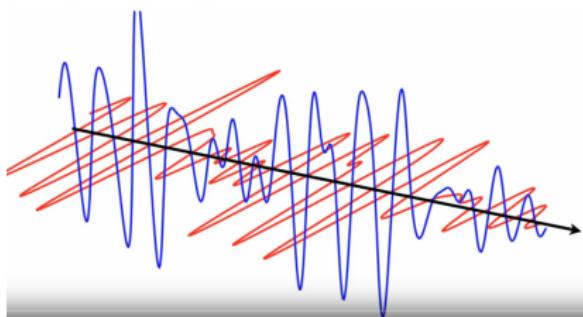
- ▶ Onde monochromatique = Belle sinusoïde
- ▶ Profile spectral discret (Discrete spectrum profile)



Physique de la lumière

Lumière blanche

- ▶ Onde bruitée
- ▶ Profile spectral continu (Continuous spectrum profile)



Physique de la lumière

- ▶ Comment la lumière est émise ?

Physique de la lumière

► Comment la lumière est émise ?

Évènement physique

Une onde lumineuse (*lightwave*) est émise lorsque les charges électriques d'un objet oscillent.

► Comment les charges oscillent ?

Physique de la lumière

► Comment la lumière est émise ?

Évènement physique

Une onde lumineuse (*lightwave*) est émise lorsque les charges électriques d'un objet oscillent.

► Comment les charges oscillent ?

Évènement physique : Émission

Conversion d'une énergie (chaleur, électrique, chimique)

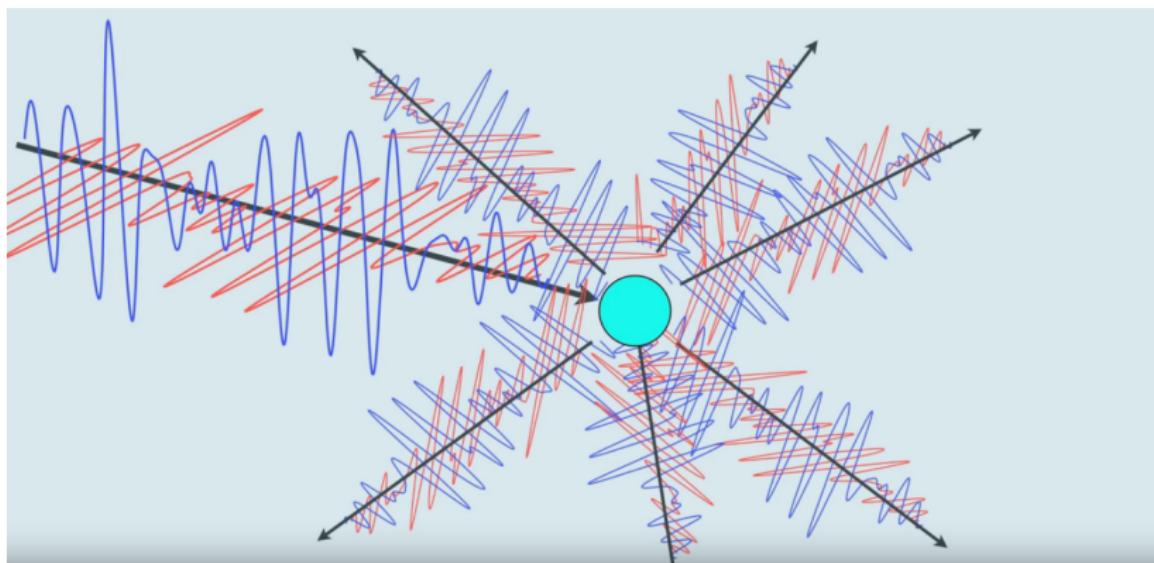
Une partie est convertie en énergie lumineuse et rayonne de l'objet

► Un objet *émissif* est considéré comme une source lumineuse (*light source*) dans le domaine du rendu.

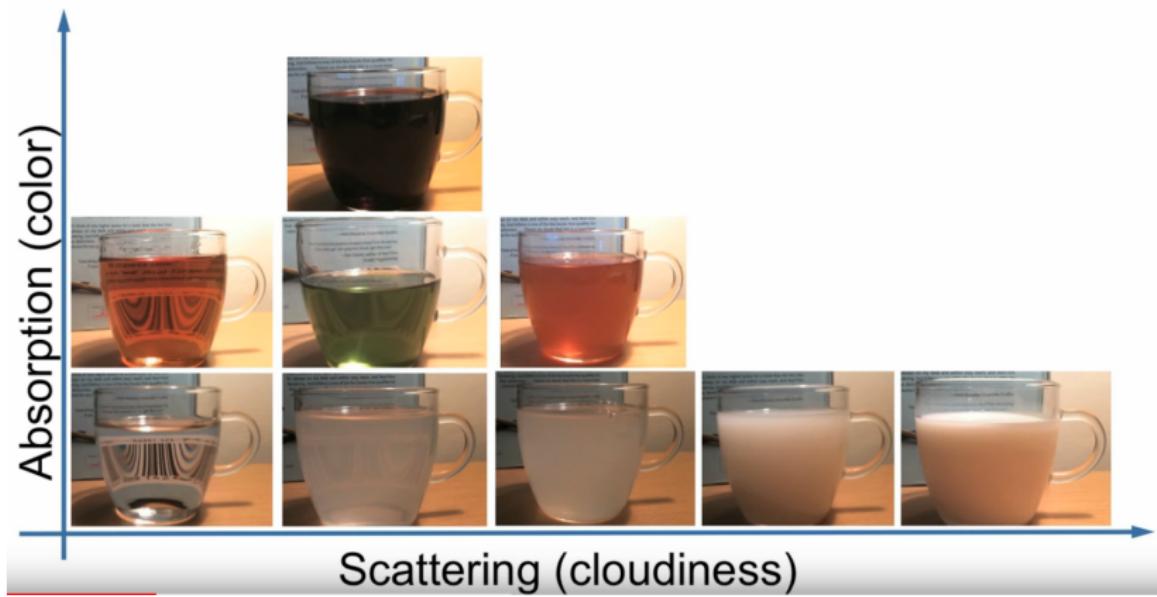
Particule

Scattering (Diffusion, Dispersion)

► Interaction matière-lumière



Support



Support

Medium homogène

Volume rempli de manière uniformément espacée de molécules identiques

- ▶ Liquide (eau)
- ▶ Solide (cristal)

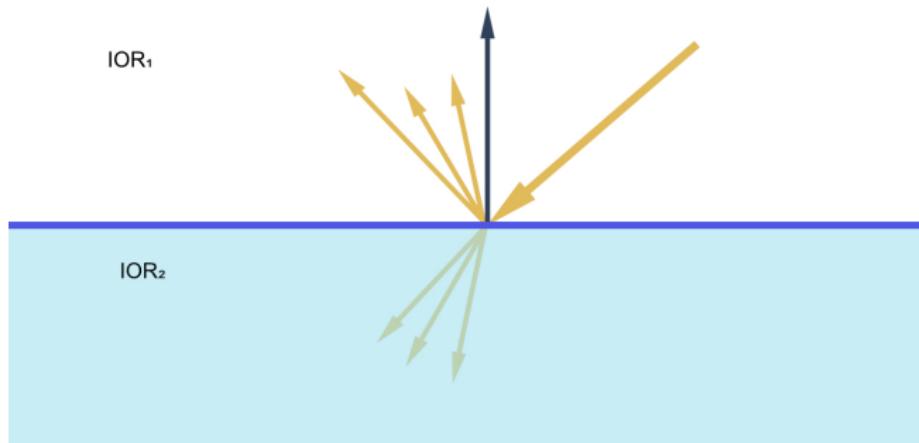
Propriété optique : Indice de réfraction

- ▶ IOR ou (*refractive index n*)
- ▶ Abstraction du détails de la lumière au niveau moléculaire pour considérer le médium comme un volume continu et homogène.
- ▶ Certains médias sont **absorbants**

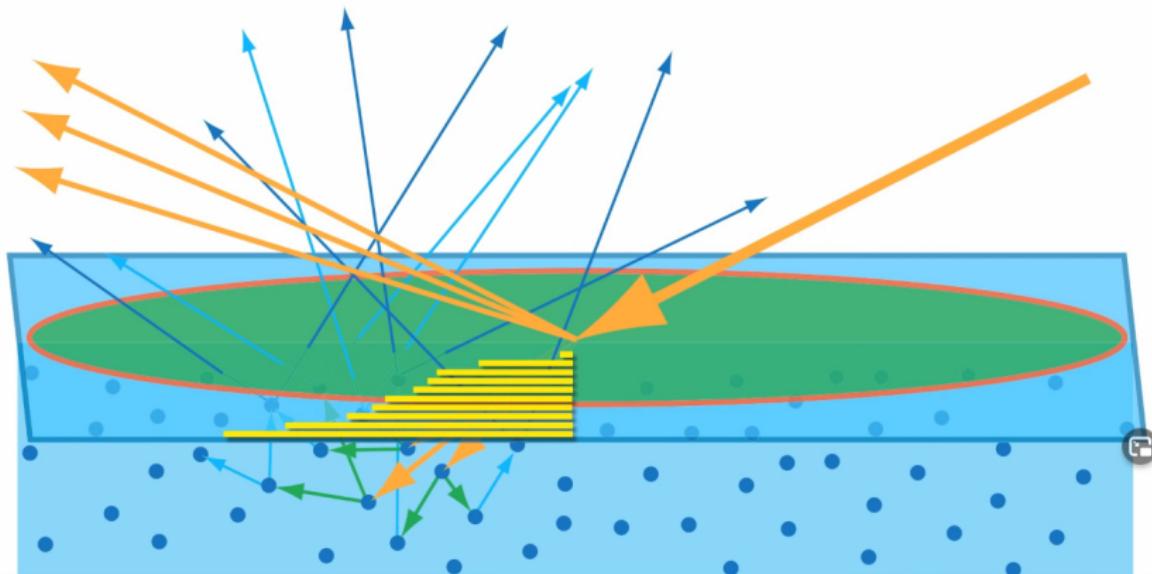
Surface

2 composantes de lumière

- ▶ Réflexion (*Reflection*) : le changement de direction de la lumière retournant dans le milieu initial
- ▶ Réfraction (*Refraction*) : le changement de direction de la lumière se déplaçant au sein du nouveau milieu



Difficulté de modélisation

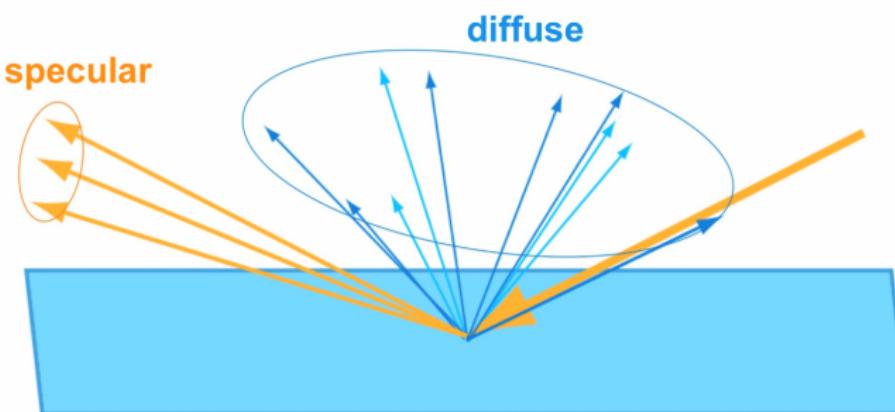


Rendu

Simplification du modèle

2 composantes

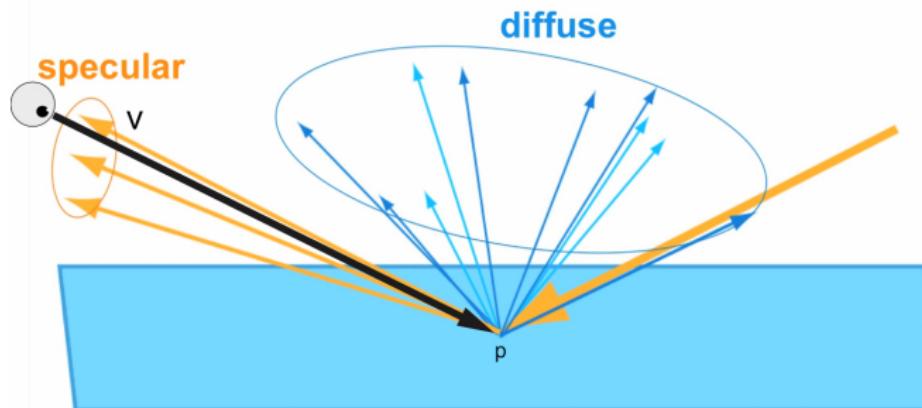
- ▶ Diffusion (Diffuse)
- ▶ Spéculaire (Specular)



Simplification du modèle

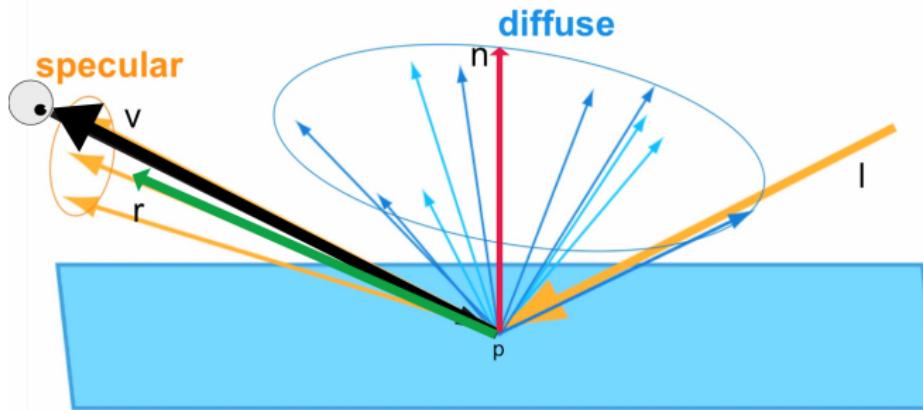
2 composantes

- ▶ Diffusion (Diffuse)
- ▶ Spéculaire (Specular)



Vecteurs essentiels

- ▶ p , le point sur la surface
- ▶ n , la normale à la surface
- ▶ v , le vecteur de la surface à la caméra
- ▶ l , le vecteur de l'origine de la lumière à la surface
- ▶ r , le vecteur direction de la lumière reflétée $r = \text{reflect}(-l, n)$



Modèle de Phong

Phong illumination or Phong reflection

Modèle empirique sur l'illumination locale d'une surface

3 composantes

- ▶ Ambiante (Ambient)
- ▶ Diffusion (Diffuse)
- ▶ Spéculaire (Specular)



James F. Blinn

MODELS OF LIGHT REFLECTION FOR COMPUTER SYNTHESIZED PICTURES.
Proc. 4th annual conference on computer graphics and interactive techniques, 1977

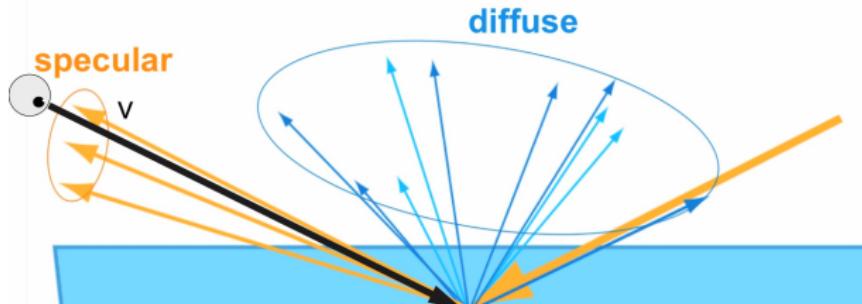
Modèle de Phong

Phong illumination or Phong reflection

Modèle empirique sur l'illumination locale d'une surface

3 composantes

- ▶ Ambiante (Ambient)
- ▶ Diffusion (Diffuse)
- ▶ Spéculaire (Specular)



La magie du rendu

Équation de la réflectance

$$L_o(v) = \int_{\Omega} f(l, v) \times L_i(l)(n.l) d\omega_i$$

où $L_o(v)$ est la radiance perçue dans la direction v , $L_i(l)$ est la radiance perçue dans la direction l pour la i -ème lumière ;

En terme de code

Une intégrale se traduit comment ?

La magie du rendu

Équation de la réflectance

$$L_o(v) = \int_{\Omega} f(l, v) \times L_i(l)(n.l) d\omega_i$$

où $L_o(v)$ est la radiance perçue dans la direction v , $L_i(l)$ est la radiance perçue dans la direction l pour la i -ème lumière ;

En terme de code

Une intégrale se traduit comment ?

- ▶ Approximation du résultat (Approche statistique, Monte-Carlo)
- ▶ Somme des radiances perçues de chaque lumière.

Modèle de Phong

► Matière

- K_a , coefficient de réflexion ambiante
- K_d , coefficient de réflexion diffuse
- K_s , coefficient de réflexion spéculaire
- s , brillance (*Shininess*)

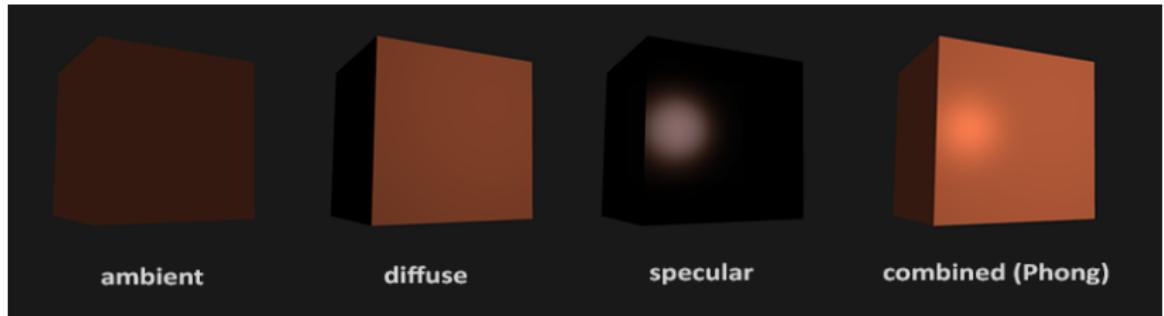
► Lumière

- C_a , couleur ambiante
- $C_{i,\{d,s\}}$, couleurs de la lumière (diffuse, spéculaire)

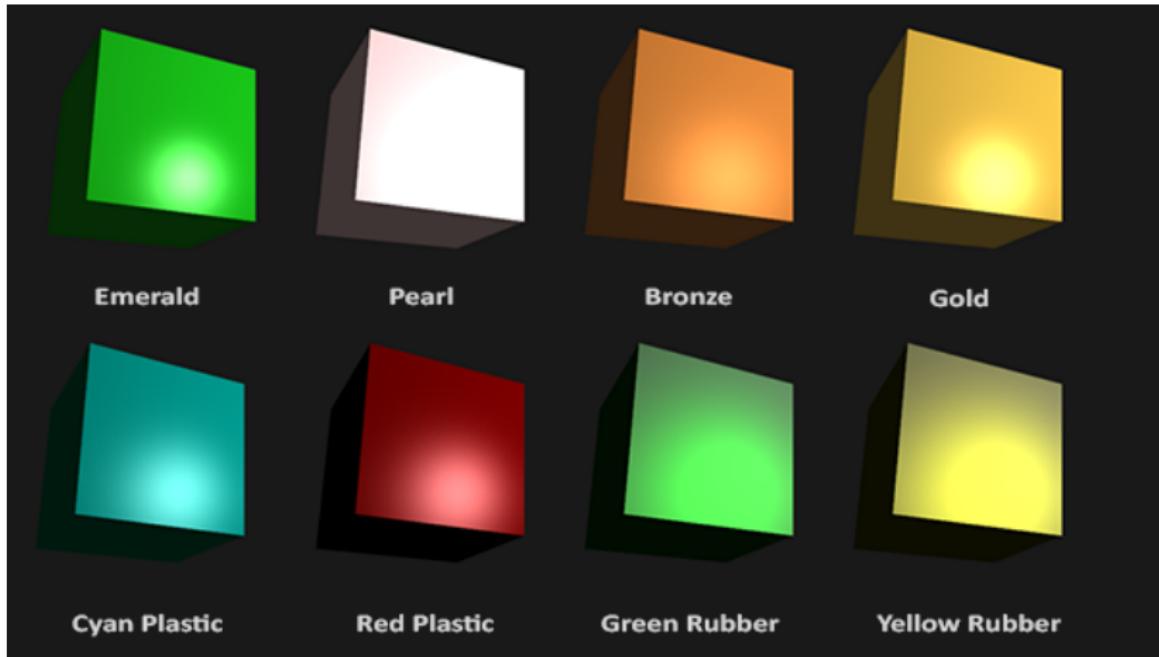
Phong lighting

$$\begin{aligned} L_o = K_a \times C_a &+ \sum_{i \in lights} K_d \times C_{i,d} \times \max(0.0, n \cdot l_i) \\ &+ K_s \times C_{i,s} \times (r_i \cdot v)^s \end{aligned}$$

Résultats Phong



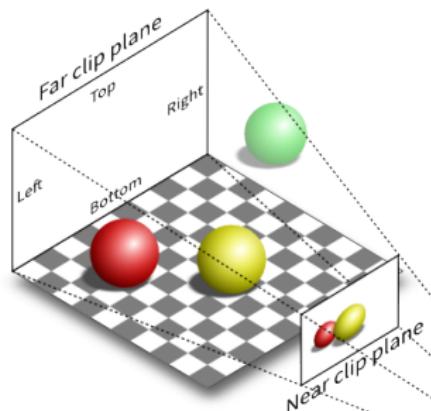
Résultats Phong



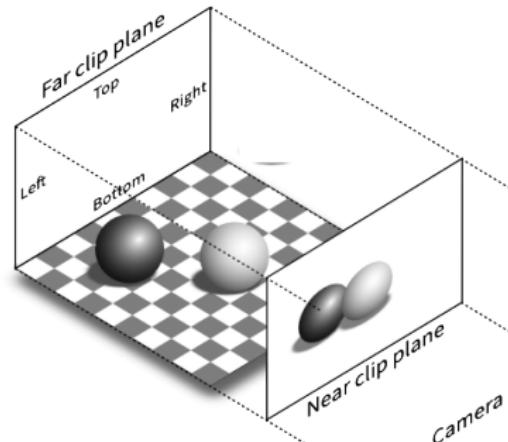
Projection

Deux types de projection

- ▶ Perspective (`gluPerspective`)
- ▶ Orthographique (`glOrtho`)



Perspective projection (P)



Orthographic projection (O)

Projection

Deux types de projection

- ▶ Perspective (gluPerspective)
- ▶ Orthographique (glOrtho)

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2f \times n}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Où FOV est l'angle en radian du champs de vision, ar le ratio d'aspect d'écran, n et f respectivement la profondeur du plan proche et du plan éloignée.

Projection

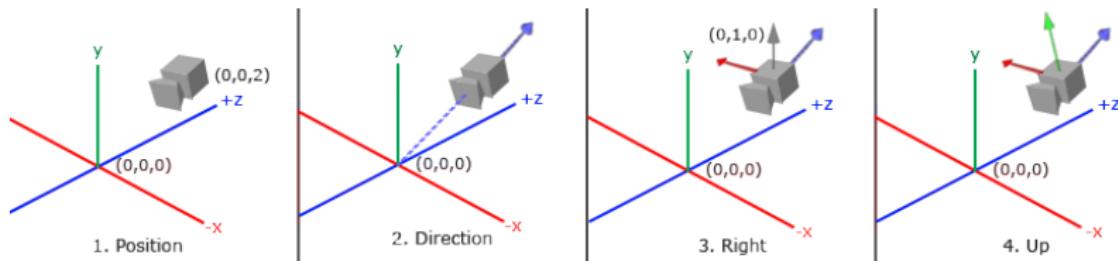
Deux types de projection

- ▶ Perspective (gluPerspective)
- ▶ Orthographique (glOrtho)

$$P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Où l , r , t et b sont des coordonnées dans le repère écran, n et f sont respectivement la profondeur du plan proche et du plan éloigné.

Caméra (*View*)



Contrôles d'une caméra classique

- ▶ A partir d'une position P
- ▶ Observation dans une direction D
- ▶ Avec des vecteurs donnant le haut de l'image U et la droite de l'image R

Caméra (*View*)

Contrôles d'une caméra classique

- ▶ A partir d'une position P
- ▶ Observation dans une direction D
- ▶ Avec des vecteurs donnant le haut de l'image U et la droite de l'image R

Représentée sous forme de matrice 4x4 :

$$V = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Combinaison de matrices

Modèle-Vue-Projection (*Projection-View-Model*)

Ordre spécifique d'enchaînement des matrices pour transformer dans le repère

$$v_{screen} = P \times V \times M \times (v_x, v_y, v_z, 1)$$

Où P est la matrice de projection, V la matrice vue et M la matrice de transformation du modèle.

Autre matrice utile : Matrice normale

$$N = (M^{-1})^t$$

Programmation

API Graphique

API Legacy

Microsoft®
DirectX
1992



1995

API Web



2011, 2017

API Graphique

API Legacy

Microsoft®
DirectX

1992



1995

API Moderne



2014



2016

API Web



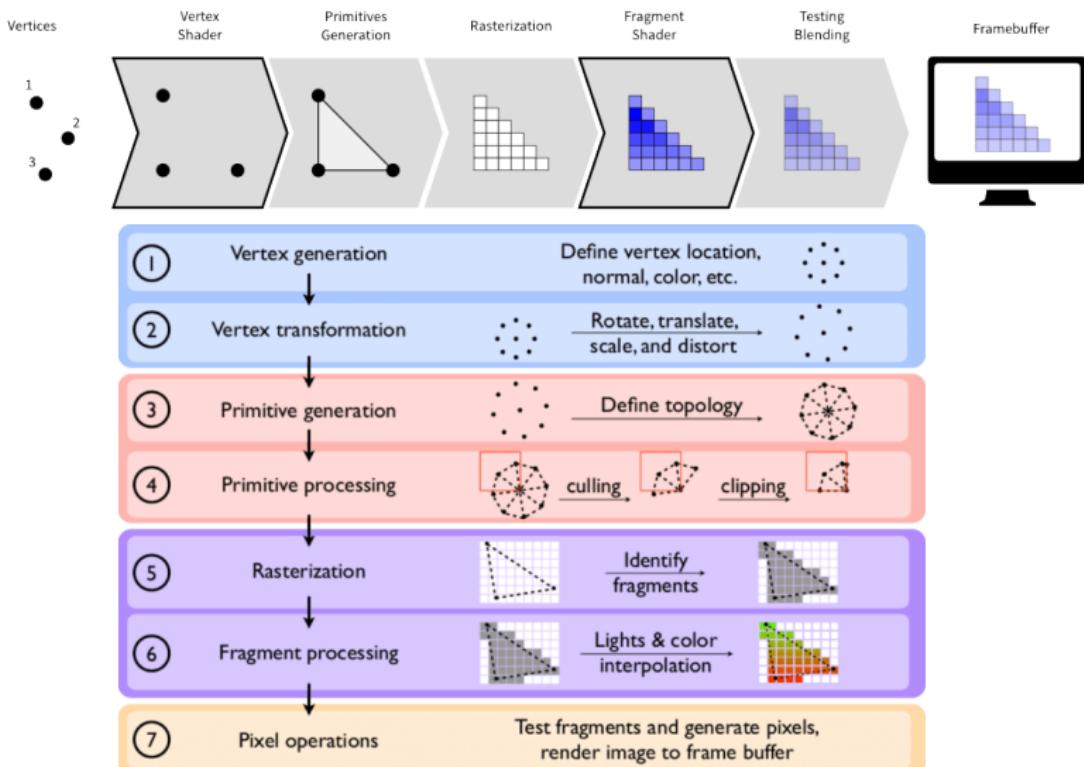
2011, 2017



WebGPU

202X

Pipeline de rendu (temps réel)



Langage de *shading*

- ▶ OSL (Open Shading Language, Sony Pictures Imageworks)
- ▶ GLSL (GL Shading Language, Khronos)
- ▶ Spir-V (Standard Portable Intermediate Representation, Khronos)
- ▶ WSL (WebGPU Shading Language, WebGPU)
- ▶ HLSL (High-Level Shading Language, Microsoft)

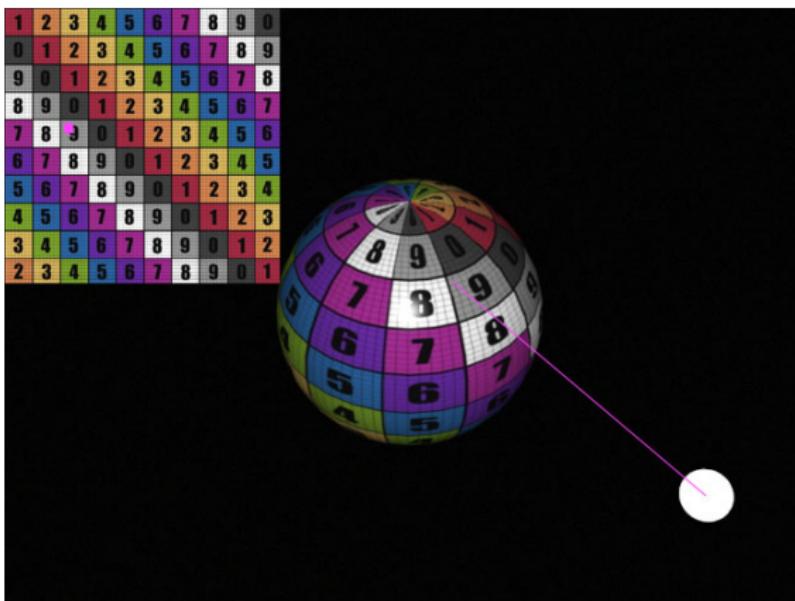
Langage de *shading*

- ▶ OSL (Open Shading Language, Sony Pictures Imageworks)
- ▶ GLSL (GL Shading Language, Khronos)
- ▶ Spir-V (Standard Portable Intermediate Representation, Khronos)
- ▶ WSL (WebGPU Shading Language, WebGPU)
- ▶ HLSL (High-Level Shading Language, Microsoft)

Placage de texture

Texture Mapping

Le fait d'appliquer une image 1D, 2D, 3D sur des primitives géométriques. Un pixel d'une texture est appelé *texel*.



OpenGL : Texture

▶ Création

```
unsigned int texture;
 glGenTextures(1, &texture);
 glBindTexture(GL_TEXTURE_2D, texture);
 // set the texture wrapping
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
 // set/filtering options (on the currently bound texture object)
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
 // load and generate the texture
 int width, height, nrChannels;
 unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0)
 ;
 if (data) {
     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
                 GL_UNSIGNED_BYTE, data);
     glGenerateMipmap(GL_TEXTURE_2D);
 } else {
     std::cout << "Failed to load texture" << std::endl;
 }
 stbi_image_free(data);
```

▶ Suppression

```
glDeleteTexture(1, &texture);
```

Exemple de shader GLSL

Vertex shader

```
// Vertex attributes
attribute vec3 a_position;
attribute vec3 a_normal; // deactivate because
    unused
// texture

// Uniforms
// Matrices, Material, etc
uniform mat4 u_projection, u_modelview;
// Textures
uniform sampler2D texture1;
// Interpolated values to transmit to fragment
// shaders
varying vec3 v_worldVertexPosition;
// Main program
void main(){
    vec4 vertPos4 = u_modelview * vec4(a_position,
        1.0);
    v_worldVertexPosition = vec3(vertPos4) /
        vertPos4.w;
//    vec3 rgb = texture(ourTexture, vertPos4.xy)
//    ;
    gl_Position = u_Projection * vertPos4;
}
```

Fragment shader

```
precision mediump float;
// Recover interpolated values
varying vec3 v_worldVertexPosition;
// Uniforms
uniform float u_intensity;

void main() {
    gl_FragColor = u_intensity * vec4(
        v_worldVertexPosition, 1.0);
}
```

OpenGL : Vertex Buffer Object

► Création (Attributs de sommet)

```
GLuint vboId;
float* vertices = new float[vCount*3]; // create vertex array
GLuint dataSize = sizeof(float) * vCount * 3;
glGenBuffers(1, &vboId);
 glBindBuffer(GL_ARRAY_BUFFER, vboId);
 glBufferData(GL_ARRAY_BUFFER, dataSize, vertices, GL_STATIC_DRAW);

// it is safe to delete after copying data to VBO
delete[] vertices;
```

► Création (Indices)

```
GLuint vboId2;
int* indices = new int[vCount*3]; // create vertex array
// ... populates array
GLuint dataSize = sizeof(int) * vCount * 3;
glGenBuffers(1, &vboId2);
 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId2);
 glBufferData(GL_ELEMENT_ARRAY_BUFFER, dataSize, indices, GL_STATIC_DRAW);
```

OpenGL : Vertex Buffer Object

► Utilisation

```
// bind VBOs for vertex array and index array
glBindBuffer(GL_ARRAY_BUFFER, vboId1);           // for vertex coordinates
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId2);    // for indices
// OpenGL 2.0 with custom shader
 glEnableVertexAttribArray(attribVertex);          // activate vertex position
     array (explanation of attribVertex later)
// set vertex arrays with generic API
 glVertexAttribPointer(attribVertex, 3, GL_FLOAT, false, 0, 0);
// draw 6 faces using offset of index array
 glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
// bind with 0, so, switch back to normal pointer operation
 glBindBuffer(GL_ARRAY_BUFFER, 0);
 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

OpenGL : Shader

▶ Création

```
GLuint fragmentshader = glCreateShader(GL_FRAGMENT_SHADER);

glShaderSource(fragmentshader, 1, (const GLchar**)&fragmentsource, 0);

glCompileShader(fragmentshader);

glGetShaderiv(fragmentshader, GL_COMPILE_STATUS, &IsCompiled_FS);
if (IsCompiled_FS == FALSE) {
    glGetShaderiv(fragmentshader, GL_INFO_LOG_LENGTH, &maxLength);

    /* The maxLength includes the NULL character */
    fragmentInfoLog = (char *)malloc(maxLength);

    glGetShaderInfoLog(fragmentshader, maxLength, &maxLength, fragmentInfoLog);

    /* Handle the error in an appropriate way such as displaying a message or
       writing to a log file. */
    /* In this simple program, we'll just leave */
    free(fragmentInfoLog);
    return;
}

/* If we reached this point it means the vertex and fragment shaders compiled and
   are syntax error free. */
```

OpenGL : Programme de *shading*

▶ Création

```
// Create a shader program
GLuint shaderprogram = glCreateProgram();
/* Attach our shaders to our program */
glAttachShader(shaderprogram, vertexshader);
glAttachShader(shaderprogram, fragmentshader);
// Link programs (should show error of shaders)
glLinkProgram(shaderprogram);
// Show link status
glGetProgramiv(shaderprogram, GL_LINK_STATUS, (int *)&IsLinked);
if (IsLinked == FALSE) {
    GLint maxLength = 0;
    glGetProgramiv(shaderprogram, GL_INFO_LOG_LENGTH, &maxLength);
    char * spInfoLog = new char[maxLength];
    glGetProgramInfoLog(shaderprogram, maxLength, &maxLength, spInfoLog);
    delete[] shaderProgramInfoLog;
    return;
}
// Now those shaders can be deleted after linking
glDeleteShader(vertexshader);
glDeleteShader(fragmentshader);
```

OpenGL : Programme de *shading*

► Utilisation

```
// Usage of program
glUseProgram(shaderprogram);
// .. use program to render something
glUseProgram(0); // Reset to classic opengl
```

► Attributs

```
// get location of attribute "a_Position in shader program"
GLint vertexAttrib = glGetAttributeLocation(shaderProgram, "a_position");
if (vertexAttrib != -1) {
    glBindBuffer(GL_ARRAY_BUFFER, &vboId1);
    // activate vertex position array
    glEnableVertexAttribArray(attribVertex);
    // bind attribute with vbo
    glVertexAttribPointer(attribVertex, 3, GL_FLOAT, false, 0, 0);
}
```

► Uniformes

```
// Get location of attribute "a_Position in shader program"
GLint modelViewLocation = glGetUniformLocation(shaderProgram, "u_modelView");
if (modelViewLocation != -1) {
    // ... float modelViewMatrix[16];
    glUniformMatrix4fv(modelViewLocation, 1, GL_FALSE, modelViewMatrix);
}
glUniform1f(glGetUniformLocation(shaderProgram, "u_intensity"), 1.0f);
```

OpenGL : Utilisation des textures dans les shaders

► Assigner une texture dans un shader

```
// activate usage of 2D textures
glEnable(GL_TEXTURE_2D);
// activate texture (limited to 32 or less depending on machine)
int indexActiveTexture = 0;// 0 to 31
glActiveTexture(GL_TEXTURE0 + indexTexture);
// bind texture as Texture 0
glBindTexture(GL_TEXTURE_2D, texture);
// set used active texture
glUniform1i(glGetUniformLocation(shaderProgram, "texture"), indexActiveTexture);
```

OpenGL : Vertex Array Object

▶ Création

```
// create VAO
glGenVertexArrays(1, &vaoID[0]);
// bind VAO
glBindVertexArray(vaoID[0]);
// ... int vboIDS[3]
glGenBuffers(3, &vboIDS[0]);
glBindBuffer(GL_ARRAY_BUFFER, vboIDS[0]);
glBufferData(GL_ARRAY_BUFFER, nbVertices * sizeof(GLfloat), vertices,
    GL_STATIC_DRAW);
glVertexAttribPointer((GLuint)0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glBindBuffer(GL_ARRAY_BUFFER, vboIDS[1]);
glBufferData(GL_ARRAY_BUFFER, nbVertices * sizeof(GLfloat), normals,
    GL_STATIC_DRAW);
glVertexAttribPointer((GLuint)1, 3, GL_FLOAT, GL_FALSE, 0, 0);
glBindBuffer(GL_ARRAY_BUFFER, vboIDS[2]);
glBufferData(GL_ARRAY_BUFFER, nbVertices * sizeof(GLfloat), colors,
    GL_STATIC_DRAW);
glVertexAttribPointer((GLuint)2, 3, GL_FLOAT, GL_FALSE, 0, 0);
// Deactivate VAO
glEnableVertexAttribArray(0);
glBindVertexArray(0);
```

▶ Utilisation

```
glBindVertexArray(vaoID[0]);
// glDrawArrays or glDrawElements ...
```

Contexte
oooooo

Théorie de la lumière
oooooooooooo

Rendu
oooooooooooo

Programmation
ooooooooooooooo

Conclusion
●oooo

TP
oo

Conclusion

Attentes dans l'industrie

Compétences attendues

- ▶ Programmation & Modélisation 3D
- ▶ Traitement d'images (*Image Processing*)
- ▶ Traitement de maillages (*Mesh Processing*)

Compétences avancées

- ▶ Compression (2D, 3D)
- ▶ Interactivité (*UI, UX*)
- ▶ Intelligence Artificielle (*Deep/Machine Learning*)
- ▶ Infrastructure Web (*Transfer optimization*)
- ▶ Modèles de lumières et de matières (*shaders, PBR*)

Références

-  **Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, Bruno Levy**
Polygon Mesh Processing.
CRC Press, 2010
-  **Matt Pharr, Wenzel Jakob, and Greg Humphreys**
Physically Based Rendering : From Theory to Implementation. (Third Edition)
Morgan Kaufmann Publishers, 2016
-  **T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, S. Hillaire**
Real-Time Rendering, Fourth Edition.
CRC Press, 2018
-  **Wolfgang Engel (Editeur)**
GPU Zen : Advanced Rendering Techniques. (1, 2)
Wolfgang Engel, 2017-2019
-  **Eric Lengyel**
Foundations of Game Engine Development, Volume 1 : Mathematics.
Terathon Software LLC, 2016
-  **Eric Lengyel**
Foundations of Game Engine Development, Volume 2 : Rendering.
Terathon Software LLC, 2019

Rendu avancé
Sondage



Contact : sebastien.beugnon@emersya.com

Contexte
oooooo

Théorie de la lumière
oooooooooooo

Rendu
oooooooooooo

Programmation
ooooooooooooooo

Conclusion
oooo

TP
●○

TP

Travaux pratiques

* Vous pouvez utiliser votre propre moteur si vous le souhaitez

Objectifs

- ▶ Rajouter dans vos objets 3D, des coordonnées de texture (UV)
- ▶ Utiliser des VBO (et/ou des VAO) pour gérer vos données
- ▶ Développer un premier *shader* reproduisant le modèle de Phong
- ▶ Intégrer des textures dans votre *shader* servant à définir la couleur d'un objet