

Informatique Graphique – 4ETI

Traitement et Synthèse d'Images

Synthèse d'images

Thibault Dupont (CPE)

Bureau B126A – thibault.dupont@cpe.fr

Informatique graphique à CPE

La suite avec moi (**en majeure IMI**)

Savoir théorique

- Modélisation 3D (S8)
- Animation et simulation (S9)
- Rendu (S9)

Savoir faire

- Traitement et synthèse d'image, synthèse (S7)
- C++ (S8)
- Programmation GPU (S9)
- Jeux vidéo (S9)

Informatique graphique

**Créer une image ou un ensemble d'image,
Calculer des données (variées) sur des objets 3D**

Pourquoi? analyse, traitement, rendu, visu. (image, vidéo, RV, RA)

Pour qui? médicale, artistique (films, jeux, patrimoine), CFAO (aérospatial, automobile, ...)

Comment? modélisation, création (procédurale, artistique, acquisition), géométrie algorithmique et discrète, reconstruction, rendu d'image, animation et simulation, visualisation et rendu

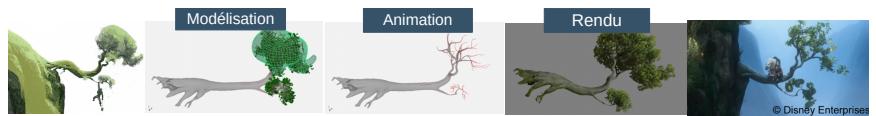


image : Frédéric Boudon
<https://team.inria.fr/virtualplants/files/2017/09/01-intro.pdf>
 ©Disney Enterprises

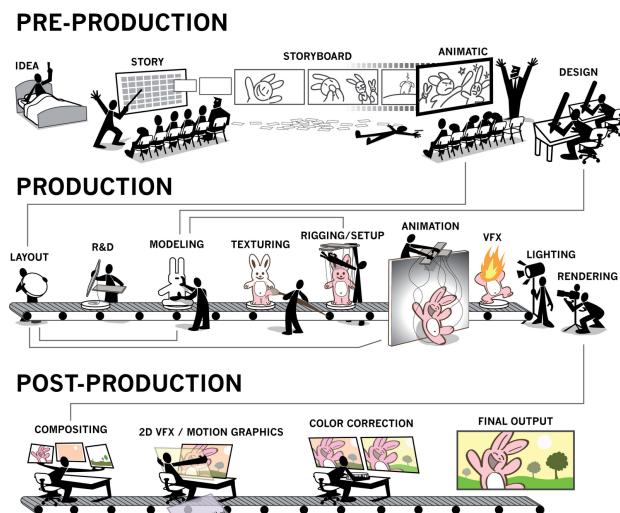
Thibault Dupont

Traitement et Synthèse d'Images - Synthèse d'images

2/33

Exemple pour un film d'animation

3D Production Pipeline



From 3D Animation Essentials, Andy Beane, John Wiley & Sons, 2012, image Brian Ludwig

Thibault Dupont

Traitement et Synthèse d'Images - Synthèse d'images

4/33

Historique

- 1950s** Oscilloscopes contrôlés par ordinateur et table traçante
- 1960s** SketchPad project (Ivan Sutherlands)
- 1970s** Première 3D, conf. Siggraph 1974, Star Wars ép. IV
- 1980s** Premières séquences 3D dans les films, Pixar, ... : super-calculateurs
- 1990s** Architectures dédiées, OpenGL 1.0 (1992), apparition des cartes graphiques
- 2000s** Programmation 3D sur PC
- 2010s** Pipeline programmable, GPGPU
- 2020s** C'est à vous ? RA, RV, Réalisme (illumination globale), ...

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

5/33

Objectifs de ce cours

Informatique graphique

- Modélisation 3D : maillages
- Introduction au rendu : projection perspective
- Changement de repère : Modèle, Vue, Perspective, Fenêtre
- Principe du pipeline graphique OpenGL
- Interaction homme/machine, gestion du temps
- Animation, simulation : gestion des collisions
- Illumination : interpolation de Phong et Gouraud, modèle de Phong, texture

Programmation

- Programmation basique en C++
- Utilisation simple de l'API graphique OpenGL
- Programmation de shaders basique en GLSL
- Gestion des structures de données de maillages
- Implémentation d'un jeu 3D sans surcouche

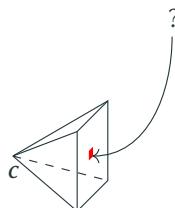
Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

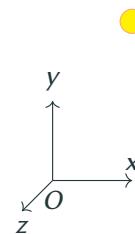
6/33

Affichage 3D d'une scène

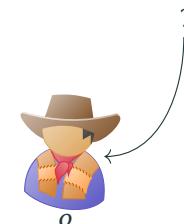
- Une caméra



- Des lumières



- Des objets



- Informations sur les objets ? les lumières
- Affichage : coloriage de pixel ?
 - o Regarder où les objets se projettent sur l'écran ←
 - o Regarder ce que chaque pixel de la caméra reçoit
- Temps pour affichage, interactivité ?

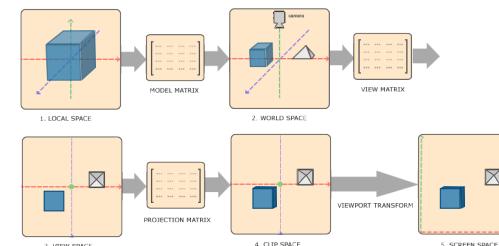
Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

7/33

Principes du rendu projectif :

- Créer un objet (ensemble de triangles) dans son repère.
- Placer l'objet dans le repère monde.
- Placer la caméra dans le repère monde (= monde dans rep. caméra)
- Projeter les obj. dans l'espace normalisé $[-1; 1]^3$ (NDC), dont profondeur
- Diviser les objets en fragment
- Calculer la position des fragments sur la fenêtre (dépend de la rés.)
- Colorer les fragments



<https://learnopengl.com/Getting-started/Coordinate-Systems>

Thibault Dupont

Beaucoup de calculs !
Traitements et Synthèse d'Images - Synthèse d'images

8/33

Rappel/début du langage C(++)

```
1 #include <iostream>
2 int var = 1; // variables globales
3
4 int f(int a, int* b)
{
5     int var = 2;
6     *b = *b + 1;
7     if(a < *b)
8         return *b;
9     return var + a;
10}
11
12 int main(int argc, char* argv[])
13 {
14     for (int a = 0; a < 3; a++)
15     {
16         std::cout << f(var, &a) << std::endl;
17         var += 1;
18     }
19     return 0;
20}
```

3

3

Thibault Dupont

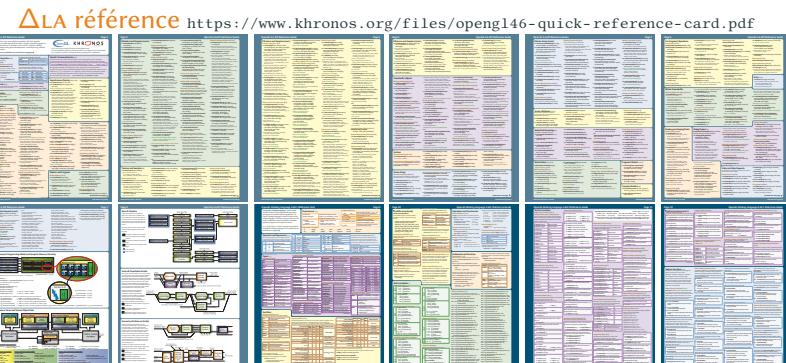
Traitements et Synthèse d'Images - Synthèse d'images

9/33

Documentation indispensables

https://www.khronos.org/opengl/wiki/Main_Page
<http://freeglut.sourceforge.net/docs/api.php>

(des différences pour mac)



Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

11/33

OpenGL

Code standard → CPU+RAM → API → GPU+VRAM → Écran
(Autres API : DirectX, Vulkan, Metal, ...)

- Crée pour du rendu projectif (surtout des triangles)
- Spécifications, implémenté dans les drivers
- Communication CPU/GPU
- Pas juste sur PC (console, web, téléphone ...)
- Fonctionnement en machine d'état : Indiquer ce que l'on souhaite faire, puis demander de le faire
 - OpenGL 1.0, 1992 : Mode direct - `glBegin()`, `glEnd()` - ⇒ DÉSUET
 - OpenGL 2.0, 2004 : Pipeline programmable (fragment et vertex)
 - OpenGL 3.0, 2008 : Modern OpenGL (Déprécié bcp de chose + TF + FBO)
 - OpenGL 3.2, 2009 : Shader de géométrie
 - OpenGL 4.0, 2010 : Shader de tessellation
 - OpenGL 4.3, 2012 : Shader de calcul
 - OpenGL 4.6, 2017 : Actuellement

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

10/33

Utilisation OpenGL

- Création d'une fenêtre
- Création du contexte (machine d'état)
- Initialisation
 - Création des programmes GPU (shaders) + envoi GPU
 - Création des données + envoi GPU
- Boucle d'affichage
 - Indiquer les shaders + données
 - Affichage ($n \times /s$)
 - » Shaders du pipeline graphique

(freeglut, glut, SDL, GLFW, Qt, GTK, ...)

(+ lib., glew, glad, glLoadGen, glee, ...)

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

12/33

Shader

Shader

- Programme sur GPU → en parallèle (SIMD)
- Inclus dans un ensemble ordonné appelé pipeline
- Ensemble des shaders du pipeline stockés dans un programme

Initialisation

1. Créer un programme GLuint glCreateProgram()
2. Créer les shaders GLuint glCreateShader(GLenum type)
3. Donner le code source aux shaders void glShaderSource(...)
4. Compiler les shaders void glCompileShader(GLuint shader)
5. Attacher les shaders au programme ... void glAttachShader(GLuint prog, GLuint shader)
6. Lier l'ensemble void glLinkProgram(GLuint prog)
7. (Optionnel) Supprimer les shaders void glDeleteShader(GLuint shader)

Affichage

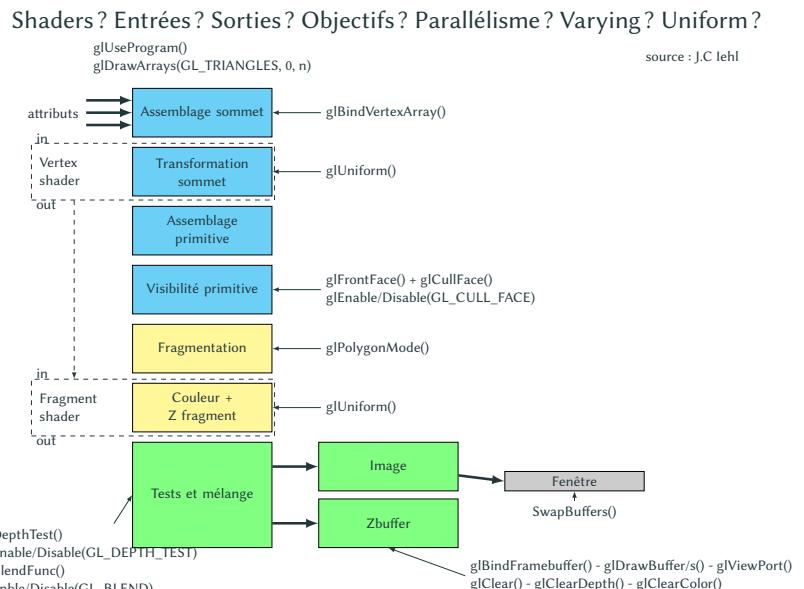
1. Indiquer le programme à utiliser void glUseProgram(GLuint prog)
2. Demander l'affichage void glDrawArrays(GLenum mode, GLint first, GLsizei count)

Thibault Dupont

Traitement et Synthèse d'Images - Synthèse d'images

13/33

Pipeline graphique pour le cours de synthèse

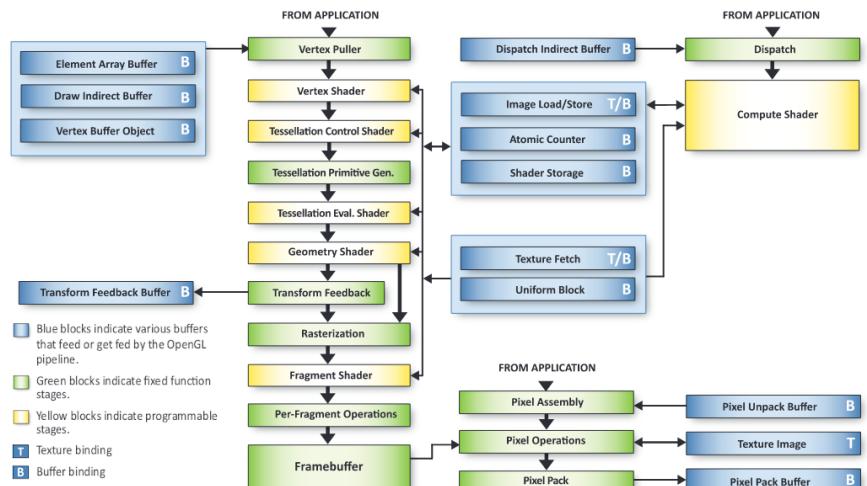


Thibault Dupont

Traitement et Synthèse d'Images - Synthèse d'images

15/33

Pipeline graphique OpenGL 4.6



khronos.com

Thibault Dupont

Traitement et Synthèse d'Images - Synthèse d'images

14/33

Vertex shader

- Permet de définir où afficher la géométrie (sans la modifier)
- Connaissance d'un unique point (+ uniforme, texture)
- En parallèle, un par sommet
- En général utilisation matrice MVP

Entrée Attribut d'un vertex + uniform + textures**Sortie** Vertex (position modifié (dans NDC) (`gl_Position`) + données interp.

Ne modifie pas les buffers d'entrées

Thibault Dupont

Traitement et Synthèse d'Images - Synthèse d'images

16/33

Fragment shader

- Calcul de la couleur (et profondeur -automatique pour nous-) pour un unique pixel \in triangle
- Après la rasterisation/fragmentation
- En parallèle, un par fragment de la primitive

Entrée Éléments interpolés + uniform + textures + build-in

S sortie Couleur + profondeur

Shaders en pratique

Vertex shader

Entrée : 2 données/vertex, 1 var. commune. **S sortie** 1 var. varying/vertex

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aCol;

uniform vec3 t;

out vec3 vColor;

void main()
{
    vColor = aCol;
    gl_Position = vec4(0.5*aPos + t, 1.0);
}
```

Fragment shader

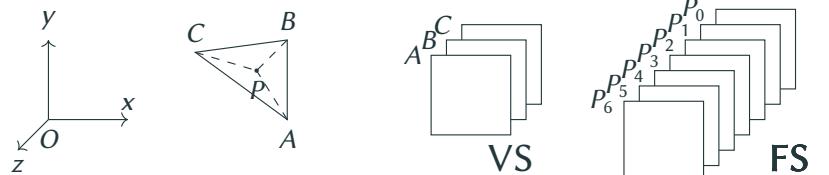
Entrée : 1 var. interp./frag. (default. barycentrique). **S sortie** couleur (et prof.)/frag.

```
#version 330 core
in vec3 vcolor;
out vec4 color;

void main()
{
    color = vec4(vcolor, 1.0f);
}
```

Variables interpolées

Transmettre des informations (couleur, profondeur, coord. texture, ...) d'une étape du pipeline à la suivante.



$$\begin{cases} \alpha = \frac{A_{PBC}}{A_{ABC}} \\ \beta = \frac{A_{APC}}{A_{ABC}} \\ \gamma = \frac{A_{ABP}}{A_{ABC}} \end{cases} \text{ avec } \begin{cases} \alpha + \beta + \gamma = 1 \\ 0 \leq \alpha, \beta, \gamma \leq 1 \end{cases}$$

$$p = \alpha a + \beta b + \gamma c$$

Initialisation du programme

```
#include <stdio.h>
#include <GL/glew.h>
#include <GL/glut.h>

static void init(){}
static void display_callback(){}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800,800);
    glutCreateWindow("Ma fenêtre");
    glutDisplayFunc(display_callback);
    glewInit();

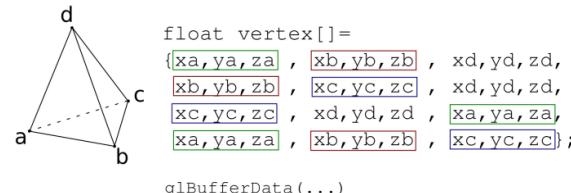
    init();
    glutMainLoop();
    return 0;
}
```

glClear() : Fonction OpenGL, efface écran + profondeur

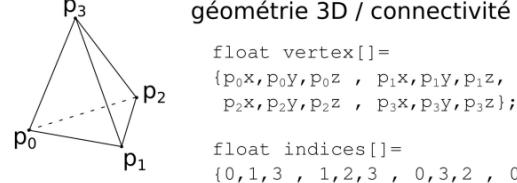
glutMainLoop() : Fonction glut, lance la boucle d'affichage

glewInit() : Initialise les fonctions OpenGL utilisable par le programme

Maillages



Séparation:
géométrie 3D / connectivité



Cours de synthèse d'images, Damien Rohmer, 2015

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

21/33

Gestion des buffers d'entrée

```
float geometrie[] = {0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f};
```

```
unsigned int index[] = {0, 1, 2};
```

```
unsigned int vao, vbo, vboi;
```

```
//Création VAO (stocke le fonctionnement/état des buffers)
```

```
glGenVertexArrays(1, &vao);
```

```
//Utilisation VAO
```

```
glBindVertexArray(vao);
```

```
//Création VBO
```

```
glGenBuffers(1, &vbo);
```

```
//Utilisation VBO
```

```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

```
//Remplissage VBO
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(geometrie), geometrie, GL_STATIC_DRAW);
```

```
//Ajout d'un attribut à l'indice 0 pour le VBO
```

```
glEnableVertexAttribArray(0);
```

```
//Guide de lecture attribut du VBO
```

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), 0);
```

```
//Création EBO = VBOI (connectivité)
```

```
glGenBuffers(1, &vboi);
```

```
//Utilisation EBO
```

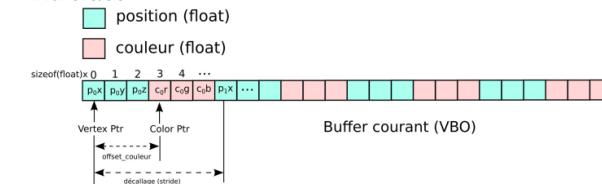
```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboi);
```

```
//Remplissage EBO
```

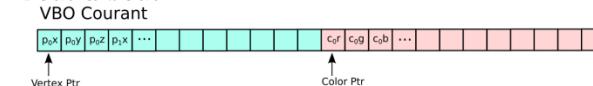
```
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(index), index, GL_STATIC_DRAW);
```

Gestion des buffers

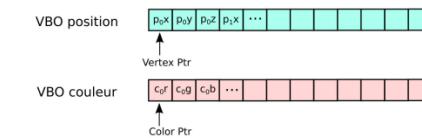
Entrelacé



Bout-à-bout



Separé



Cours de synthèse d'images, Damien Rohmer, 2015

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

22/33

Gestion des variables uniformes

```
GLint loc_rotation = glGetUniformLocation(shader_program_id, "rotation");
if (loc_rotation == -1) std::cerr << "Pas de variable uniforme : rotation" << std::endl;
glUniformMatrix4fv(loc_rotation, 1, false, pointeur(rotation));
```

```
GLint loc_translation = glGetUniformLocation(shader_program_id, "translation");
if (loc_translation == -1) std::cerr << "Pas de variable uniforme : translation" << std::endl;
glUniform4f(loc_translation, translation_x, translation_y, translation_z, 0.0f);
```

- Propre à un programme (le dernier utilisé)
- Recherche de l'identifiant de la variable pour ce programme
- Mise à jour de la variable
- Accessible dans tout le pipeline graphique en lecture seule

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

23/33

Thibault Dupont

Traitements et Synthèse d'Images - Synthèse d'images

24/33

Demande d'affichage

- `glUseProgram()` : Comment ?
- `glBindVertexArray()` : Quoi ? (+ uniformes/textures)
- `glDrawArrays()` || `glDrawElements()`,

Projection

Orthogonale • Projeté orthogonalement à un plan

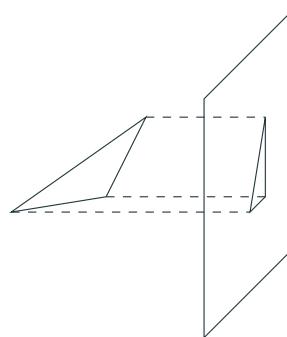
• Pas de point de fuite

• ex. $x=x$, $y=y$, $z=0$

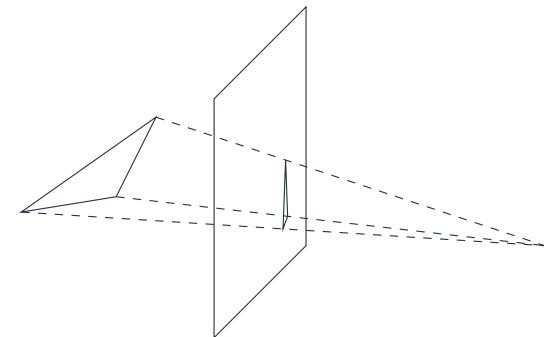
Perspective • Projeté sur un point (on s'arrête sur le plan)

• Les éléments lointain apparaissent plus petits

• Étudié en majeure Image (matrice donnée en TP)



Thibault Dupont



Traitements et Synthèse d'Images - Synthèse d'images

26/33

Gestion de la profondeur

par default Affiche triangle dans l'ordre donné : algo. du peintre

zbuffer Image de profondeur (la plus proche)

activation `glEnable/Disable(GL_DEPTH_TEST)`

Interaction

Le déplacement des personnages, la gestion du temps et de la caméra se fait dans le programme principal.

Pour déplacer les personnes deux grandes méthodes existent :

Mesh based

Modification de la position de l'objet avant l'envoi des données

⚠ On doit renvoyer le maillage dès que celui-ci bouge

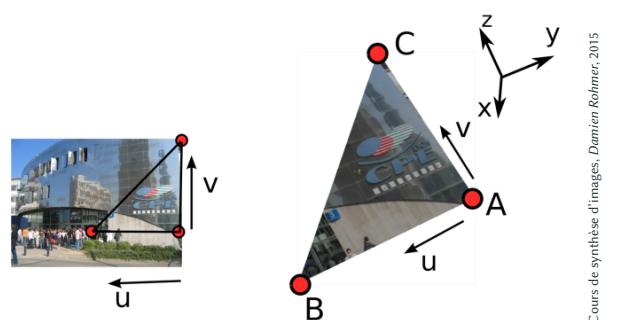
Shader based

Modification de la position de l'objet dans le vertex shader

⚠ Attention pour les intersections, dans le code principal, le maillage ne se déplace pas !

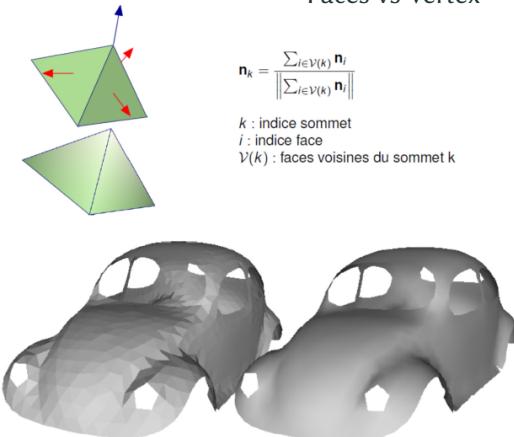
Pour la caméra et le temps, si besoin, on peut envoyer les paramètres en uniforme dans les shaders.

Texture

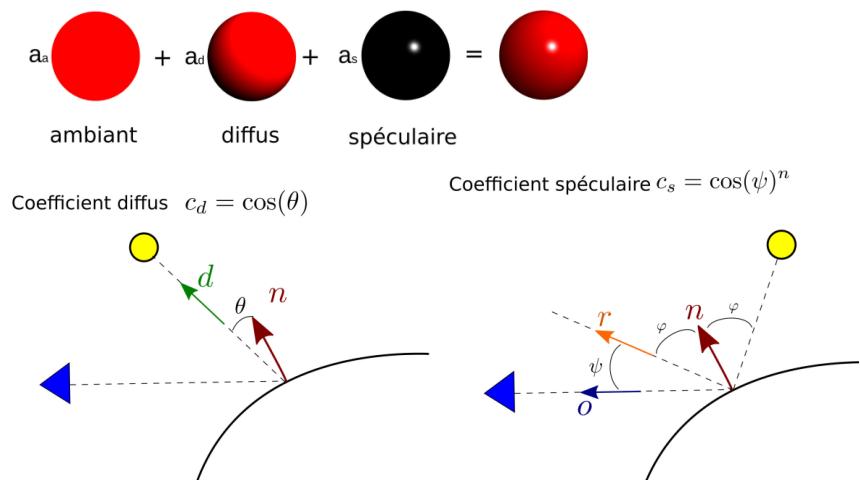


Normales

Faces vs Vertex



Modèle d'illumination de Phong



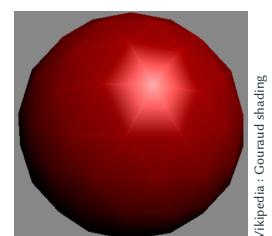
Interpolation : Gouraud vs Phong

Gouraud

Calculer l'illumination pour chaque sommet puis interpoler les couleurs

Phong

Interpoler les normales/couleurs pour chaque fragment puis calculer l'illumination



Artefact interp. de Gouraud

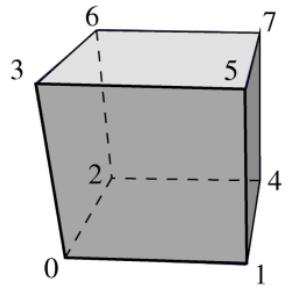
Format fichiers 3D

```
OFF  
8 6 12  
0 0 0  
1 0 0  
0 1 0  
0 0 1  
1 1 0  
1 0 1  
0 1 1  
1 1 1  
4 0 1 4 2  
4 1 5 7 4  
4 3 6 7 5  
4 2 6 3 0  
4 2 4 7 6  
4 0 3 5 1
```

.off

```
v 0 0 0  
v 1 0 0  
v 0 1 0  
v 0 0 1  
v 1 1 0  
v 1 0 1  
v 0 1 1  
v 1 1 1  
f 1 2 5 3  
f 2 6 8 5  
f 4 7 8 6  
f 3 7 4 1  
f 3 5 8 7  
f 1 4 6 2
```

.obj



Cours de synthèse d'images, Damien Rohmer, 2015