

# Synthèse d'Image

# Introduction

Tamy Boubekour

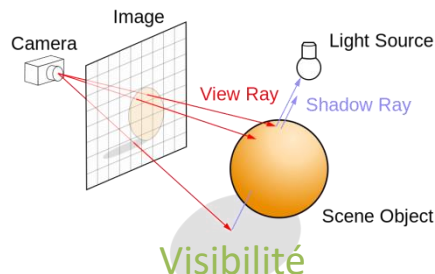
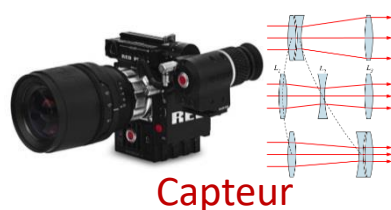
# **INTRODUCTION**

# Thèmes abordés

1. Scène et Géométrie
2. Caméra et Transformation
3. Visibilité
4. Apparence
5. Textures
6. Structures Spatiales
7. Ombres
8. Programmation GPU
9. Eclairage global
10. Effets Spéciaux
11. Rendu Expressif / NPR

# Synthèse d'Image

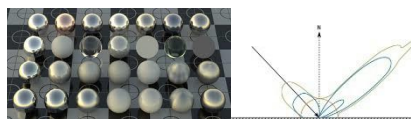
Modèles



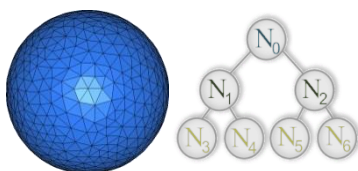
Algorithmes



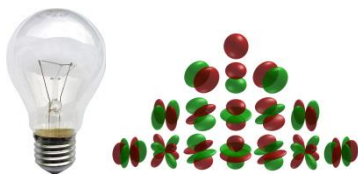
Mouvement



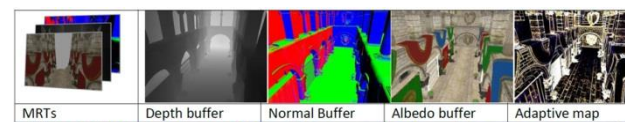
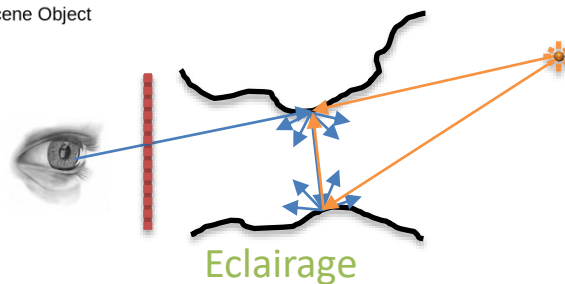
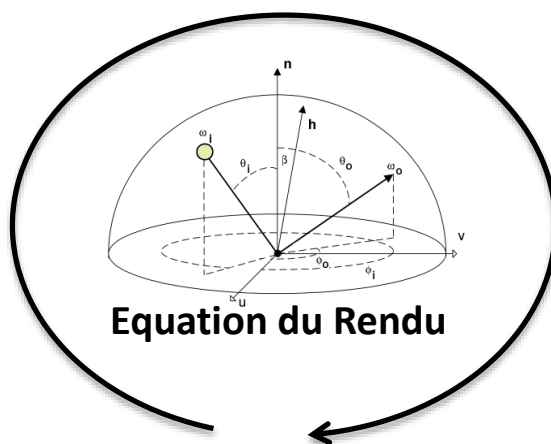
Apparence



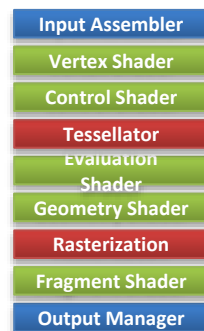
Forme



Lumière



Post-processing



Pipeline GPU



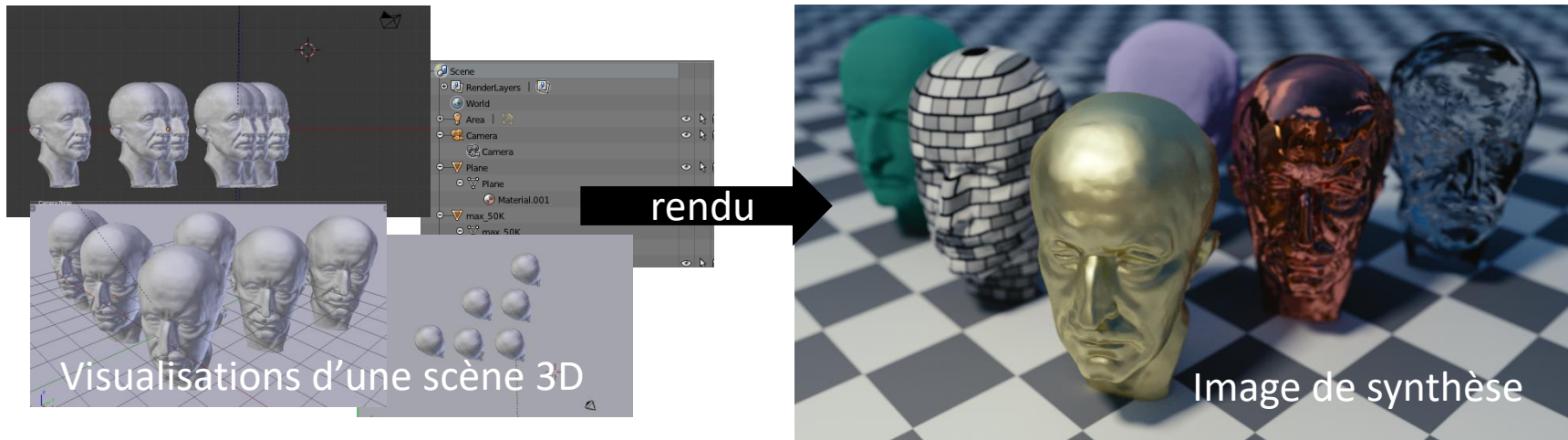
Rendu Temps-Réel

Implémentations

# Synthèse d'Image

- Informatique
  - Physique
  - Mathématiques Appliquées
  - Traitement du Signal
- 
- Une discipline de l'Informatique Graphique

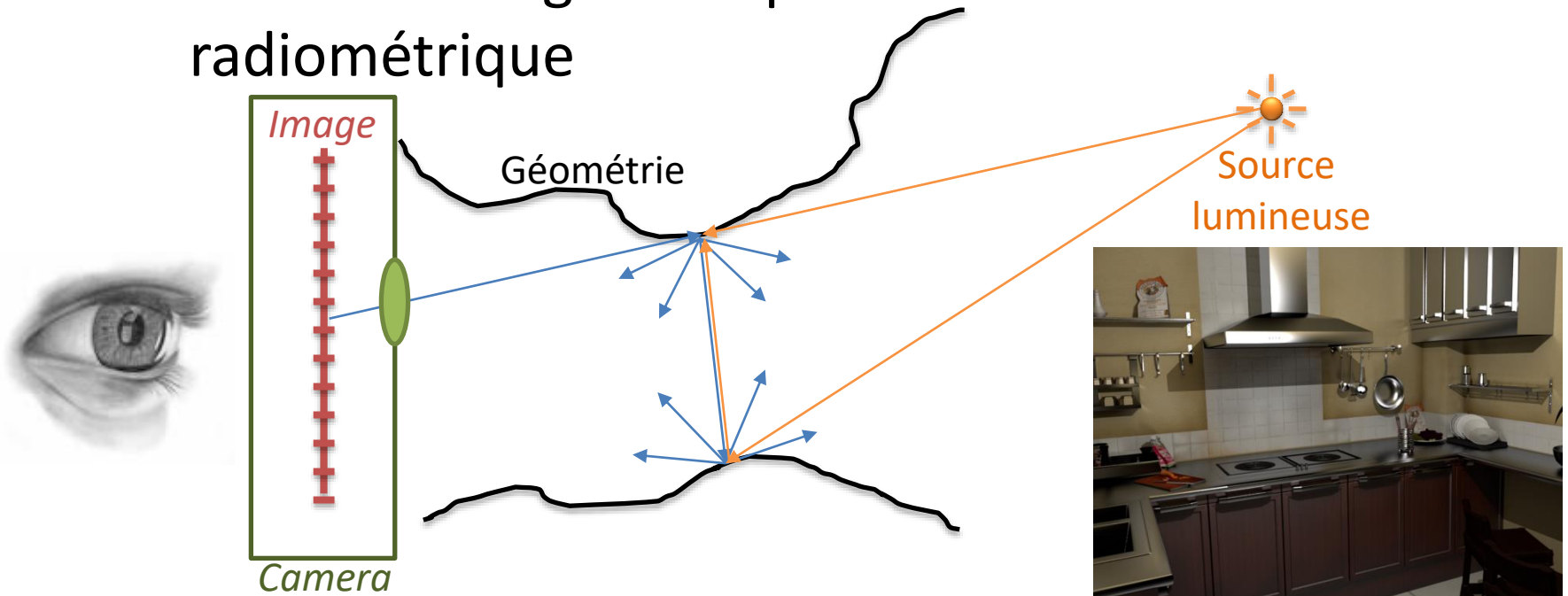
# Un processus de simulation



- Rendu = Synthèse d'Image
- Génération d'une image numérique à partir d'une scène 3D
- Réaliste (physiquement plausible) ou expressif

# Rendu basé physique

- Simulation du transport de la lumière
  - De la source au capteur
  - Dans une scène virtuelle
  - Pixel de l'image du capteur = mesure radiométrique



# Rendu temps-réel

- Pour les systèmes interactifs
- Approximation géométrique et radiométrique de la scène
- Calcul parallèle (GPU)



# Applications

Effets spéciaux et animation



Jeux vidéo et application interactives



CAO, architecture et la prévisualisation



# Bref Historique

Invention de l'écran CRT

1959 : Invention du TX-2 par Ivan Sutherland (MIT)

- Premier système graphique informatique interactif

1960 : William Fetter, Boeing Aircraft

- Études ergonomiques sur le corps humain
- Terme « Computer Graphics »

1963 : Logiciel **SketchPad**

- Thèse de Sutherland
- Sur TX-2
- Turing Award en 1988

1969 : SIGGRAPH

- *ACM Special Interest Group on Graphics*

Années 70 : bases en modélisation et rendu

- Placage de textures
- Z-Buffer
- Surfaces de Subdivision

Années 80 : machines personnelles (Commodore, Atari)

- Le film Tron
- Les premiers jeux en 3D
- Systèmes Interactif & CAO



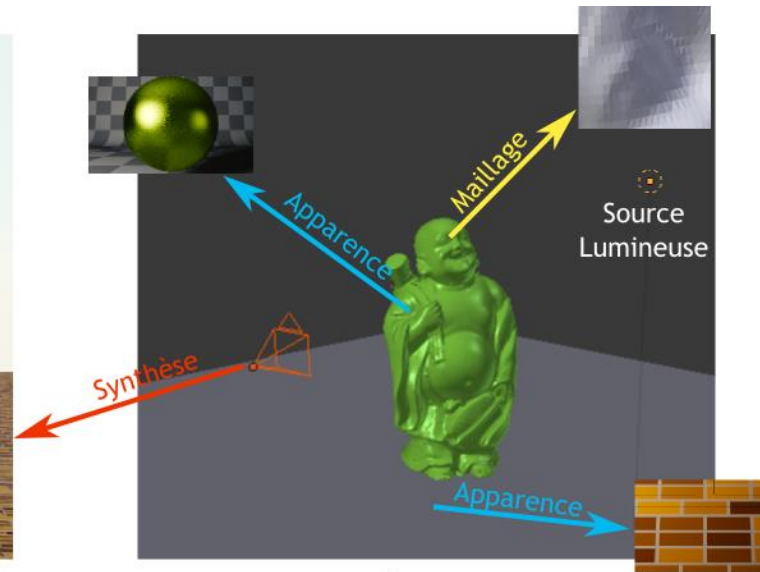
# **SCÈNE ET GÉOMÉTRIE**



# Modèles de Scène 3D



Image Numérique

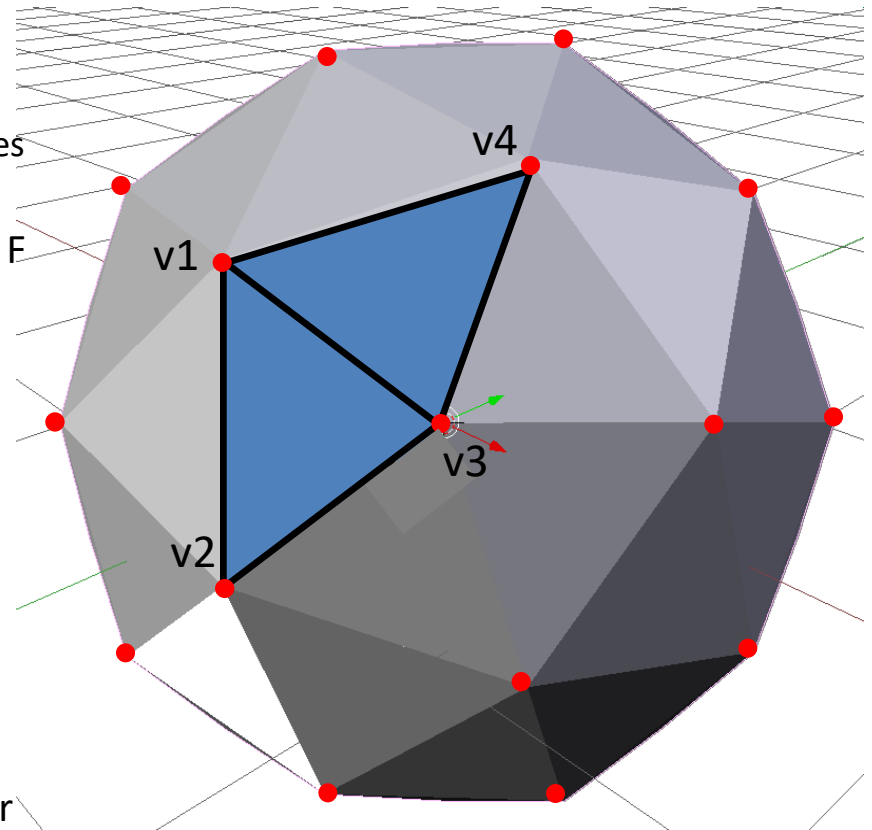


Scène 3D

- Une collection de modèles :
  - Capteur (caméra)
  - Géométries
    - Maillages, particules, iso-surfaces, etc
  - Apparence
    - Matériaux, textures
  - Lumières
  - Animation
    - Évolution temporelles des paramètres
- des autres modèles
  - Physique solide, fluides, corps déformables
  - Interactivité et actuators
- Une structure entre ces modèles
  - Appartenance et hiérarchie
  - Données et instances

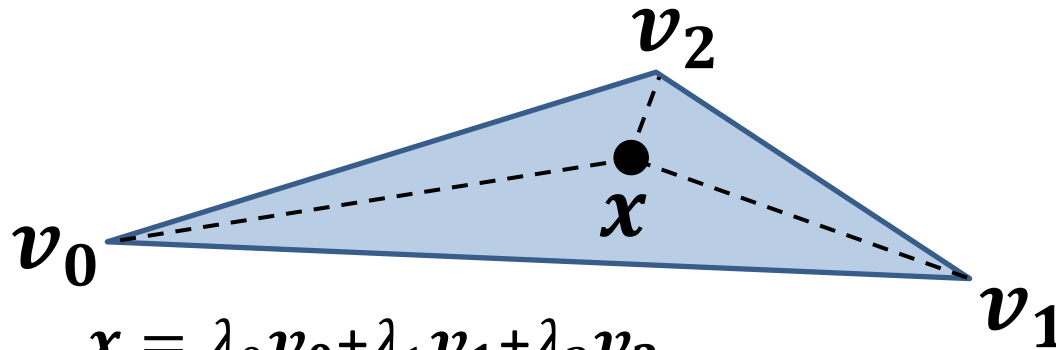
# Surface Maillée

- Maillage:
  - modèle géométrique dominant en rendu
  - Génération possible à partir des autres modèles
    - Cf cours de Modélisation Géométrique
- Définition: un ensemble de faces polygonales  $F$  indexant un ensemble de sommets  $V$ .
- $V$ : Ensemble de sommets (géométrie)
  - $v1(x, y, z)$
  - $v2(x, y, z)$
  - $v3(x, y, z)$
  - $v4(x, y, z)$
- $F$ : Ensemble de faces (topologie)
  - $(v1, v2, v3)$
  - $(v1, v3, v4)$
- Outre la position, chaque sommet peut porter d'autres attributs:
  - vecteur *normales*, critiques en rendu.
  - *Couleur par sommet*
  - *Coordonnées de textures (UV)*



# Coordonnées barycentrique

- Coordonnée d'un point dans l'espace d'un polygone
- Simple pour un triangle
- Permet d'interpoler linéairement tout attribut de sommet

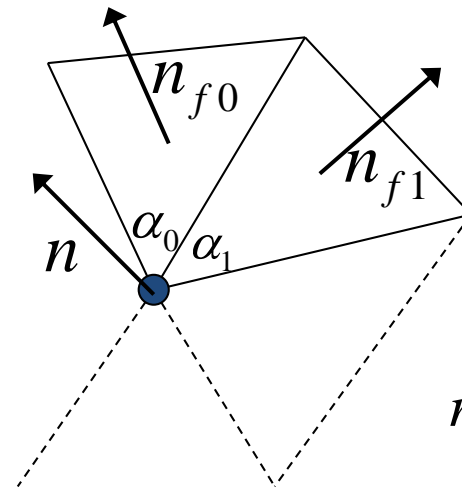
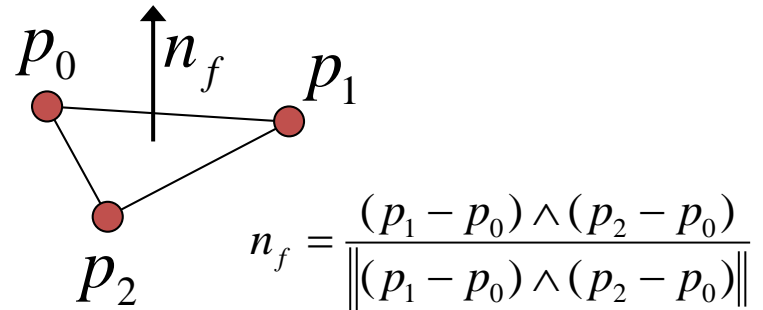


Coordonnée  
barycentriques de  $x$   
dans le triangle

$$\begin{cases} x = \lambda_0 v_0 + \lambda_1 v_1 + \lambda_2 v_2 \\ \lambda_0 = \frac{[(v_2 - v_1) \otimes (x - v_1)] \cdot a}{(v_1 - v_0) \otimes (v_2 - v_0)} \\ \lambda_1 = \frac{[(v_0 - v_2) \otimes (x - v_2)] \cdot a}{(v_1 - v_0) \otimes (v_2 - v_0)} \\ \lambda_2 = \frac{[(v_1 - v_0) \otimes (x - v_0)] \cdot a}{(v_1 - v_0) \otimes (v_2 - v_0)} \end{cases}$$
$$a = \frac{(v_1 - v_0) \otimes (v_2 - v_0)}{\|(v_1 - v_0) \otimes (v_2 - v_0)\|^2}$$

# Normales

- **Essentielles pour le rendu**
  - Alignement de la BRDF
- Stockage aux sommets ou par cartes (normal maps)
- Calculs possibles:
  - Moyennes des normales des faces incidentes
  - Moyennes pondérée par les angles des arêtes incidentes
    - Plus robustes pour les distributions de triangles non uniformes
  - Moyenne pondérée par l'aire de l'intersection du triangle et de la cellule de voronoi du sommet.



$$\dot{n} = \frac{\sum_i \alpha_i n_i}{\sum_i \alpha_i}$$
$$n = \frac{\dot{n}}{\|\dot{n}\|}$$

# Interpolation de Normales

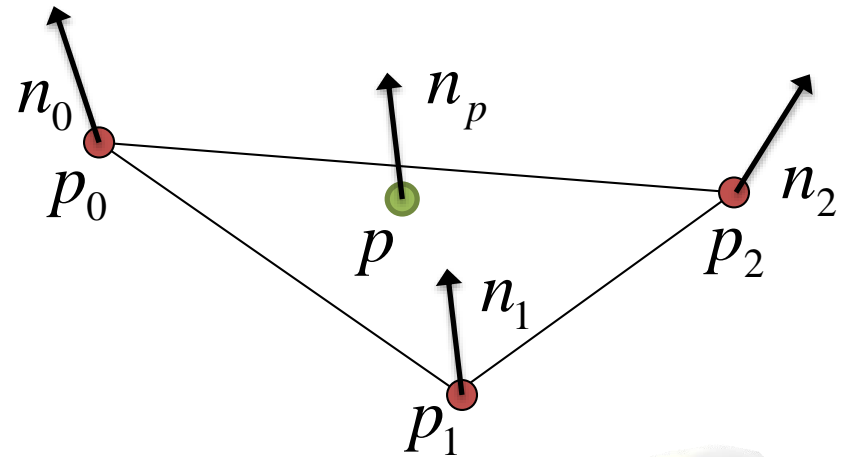
- Une normale en chaque point d'un maillage à partir des normales de ses sommets [Phong 75].
- Soit un point  $p$  sur un triangle  $t$  tel que :

$$p = \lambda_0 p_0 + \lambda_1 p_1 + \lambda_2 p_2$$

Avec  $(\lambda_0, \lambda_1, \lambda_2)$  les **coordonnées barycentrique** de  $p$  dans  $t$ .

Alors on définit la normale interpolée de Phong en  $p$  comme :

$$n_p = \frac{\lambda_0 n_0 + \lambda_1 n_1 + \lambda_2 n_2}{\|\lambda_0 n_0 + \lambda_1 n_1 + \lambda_2 n_2\|}$$



Normale de face



Normale de Phong

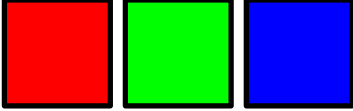


**IMAGERIE**

# Image Numérique

- Grille de pixels couleurs
- Pixel:
  - Photographie numérique : correspond à un point une petite région sur le capteur photosensible
  - Image de synthèse :
    - correspond à un point le capteur virtuel
    - correspond à un ensemble de chemins lumineux
    - correspond à un point 3D de la scène
    - correspond à une collection d'attributs interpolés

# Couleur en image de synthèse

- Espace RGB 
- Précision variable
  - 8bits/composante : standard des APIs, des GPU, du rendu et des écrans
  - 16/32bits par composante : imagerie à haute dynamique (HDR)
    - Nécessite une conversion avant affichage (Tone Mapping)
- **RGBA** : transparence *alpha* par pixel
  - Pose un problème d'ordonnancement
    - Pas directement compatible avec les algorithmes de rendu par projection (Z-Buffer)

# **INTRODUCTION À C++**

# C++

- Langage de programmation dominant en informatique graphique
- Abstraction de haut niveau
  - Structure élégante des programmes
- Accès bas niveau
  - Programmes efficaces

# Du C au C++

- **Objets** et héritage multiple
- Patrons de classes (templates)
- Surcharge et **redéfinition** des opérateurs
- Gestion des exceptions
- STL : bibliothèque standard (vector, list, IO, etc)
- Langage très (trop) permissif
  - Se donner des règles et les respecter
  - Dans ce cours :
    - pas besoin d'héritage, ni de template complexes, ni d'itérateurs, etc
    - se limiter à des classes simples
    - ne pas hésiter à implémenter des fonctions isolées type C pour le traitement
    - exploiter au maximum la STL

# Classes et Opérateurs

```
#include <iostream>
```

```
class Vec2 {
```

```
public:
```

```
    Vec2 (float x = 0.0, float y = 0.0) : _p[0] (x), _p[1] (y) {} // Constructeur
```

```
    ~Vec2 () {} // Destructeur
```

```
    inline float length () const { return sqrt (_p[0]*_p[0]+_p[1]*_p[1]); } // Méthode
```

```
    inline float & operator[] (int i) { return _p[i]; } // Redéfinition de l'opérateur crochet
```

```
    inline const float & operator[] (int i) const { return _p[i]; } // Surchage pour instances constantes
```

```
private:
```

```
    float _p[2]; // Attribut
```

```
};
```

```
int main (int argc, const char * argv[]) {
```

```
    Vec2 x (2.0, 3.0); // Construction
```

```
    std::cout << x.length () << std::endl; // Affiche 3.6055...
```

```
    return 0; // Retour sans erreur
```

```
}
```

# Mémoire et pointeurs

```
#include <iostream>
```

```
[...]
```

```
int main (int argc, const char * argv[]) {  
    Vec2 * x = new Vec2 (2.0, 3.0); // Appelle du constructeur  
    std::cout << x->length () << std::endl;  
    delete x; // Détruit la valeur pointée à l'aide du destructeur du type  
    return 0;  
}
```

*L'allocation dynamique est couteuse en temps : à réserver au « gros » objets.*



# Références

```
#include <iostream>
```

```
void doNothing (Vec2 v) {
```

```
    v[0] = 5.0;
```

```
}
```

```
void doSomething (Vec2 & v) {
```

```
    v[0] = 5.0;
```

```
}
```

```
int main (int argc, const char * argv[]) {
```

```
    Vec2 x (2.0, 3.0);
```

```
    doNothing (x);
```

```
    std::cout << x[0] << std::endl; // Affiche 2.0
```

```
    doSomething (x);
```

```
    std::cout << x[0] << std::endl; // Affiche 5.0
```

```
    return 0;
```

```
}
```

# STL

```
#include <iostream>
```

```
#include <vector>
```

```
int main (int argc, const char * argv[]) {
```

```
    std::vector<Vec2> polygon;
```

```
    polygon.push_back (Vec2 (0.0, 0.0));
```

```
    polygon.push_back (Vec2 (1.0, 0.0));
```

```
    polygon.push_back (Vec2 (0.0, 1.0));
```

```
    if (polygon.size () == 3)
```

```
        std::cout << « C'est un triangle » << std::endl;
```

```
    return 0;
```

```
}
```

# Une classe complète

- La class Vec3
- Base des TPs de ce cours

[Listing code]

# Références sur C++

- The C++ Programming Language, Special Edition, Bjarne Stroustrup, Addison-Wesley, 2009
- [www.cplusplus.com](http://www.cplusplus.com)
- <https://www.sgi.com/tech/stl/>

# **INTRODUCTION À OPENGL**

# OpenGL

- API graphique générique
  - Mac/PC/Linux/iOS/Android/html5/etc
- Plusieurs version
  - OpenGL Classique (v1.2)
  - OpenGL programmable (v2.0)
  - OpenGL moderne (v3/v4)

*Programmation avancée en OpenGL traitée plus tard*

# HelloWorldGL.cpp

```
#include <GL/glut.h> // Inclue GL.h automatiquement
// Pour compiler: g++ -o HelloWorldGL -lGL -lglut HelloWorldGL.cpp
void display () {
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Efface les composantes couleurs et profondeur (zbuffer) du framebuffer
    glBegin (GL_TRIANGLES); // Commence une liste de primitive de type triangle
    glColor3f (1.0, 0.0, 0.0); // Spécifie la couleur RGB de tous les sommets à partir de maintenant
    glVertex3f (0.0, 0.0, 0.0); // Emet un sommet de coordonnées x=0, y=0, et z=0
    glVertex3f (1.0, 0.0, 1.0);
    glVertex3f (0.0, 1.0, 0.0); // Trois sommets ont été émis > un triangle, rouge ici, a été dessiné
    glColor3f (0.0, 1.0, 0.0); // La couleur passe au vert désormais
    glVertex3f (1.0, 0.0, 0.0);
    glVertex3f (1.0, 1.0, 0.0);
    glVertex3f (0.0, 0.0, 1.0); // Un triangle vert est dessiné
    glEnd (); // Termine la liste de primitive à dessiner
    glutSwapBuffers ();
    // Tout a été dessiné dans un buffer arrière, pour éviter un dessin progressif à l'écran. On échange le buffer arrière avec le buffer avant
    // (celui dessiné à tout instant à l'écran) avec glutSwapBuffers.
}

int main (int argc, char ** argv) {
    glutInit (&argc, argv); // Initialise le gestionnaire de fenêtre glut
    glutInitDisplayMode (GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE); // Les buffers seront en RGBA. Un buffer arrière est présent.
    glutInitWindowSize (1280, 800); // La résolution de la fenêtre et de ses buffers
    glutCreateWindow («HelloWorldGL.cpp»); // Création de la fenêtre à l'écran
    glEnable (GL_DEPTH_TEST); // Active le zbuffer
    glutDisplayFunc (display); // Place le pointeur vers la fonction de dessin OpenGL
    glutMainLoop (); // Commence un boucle de dessin infinie.
    return 0;
}
```

# Ressources OpenGL

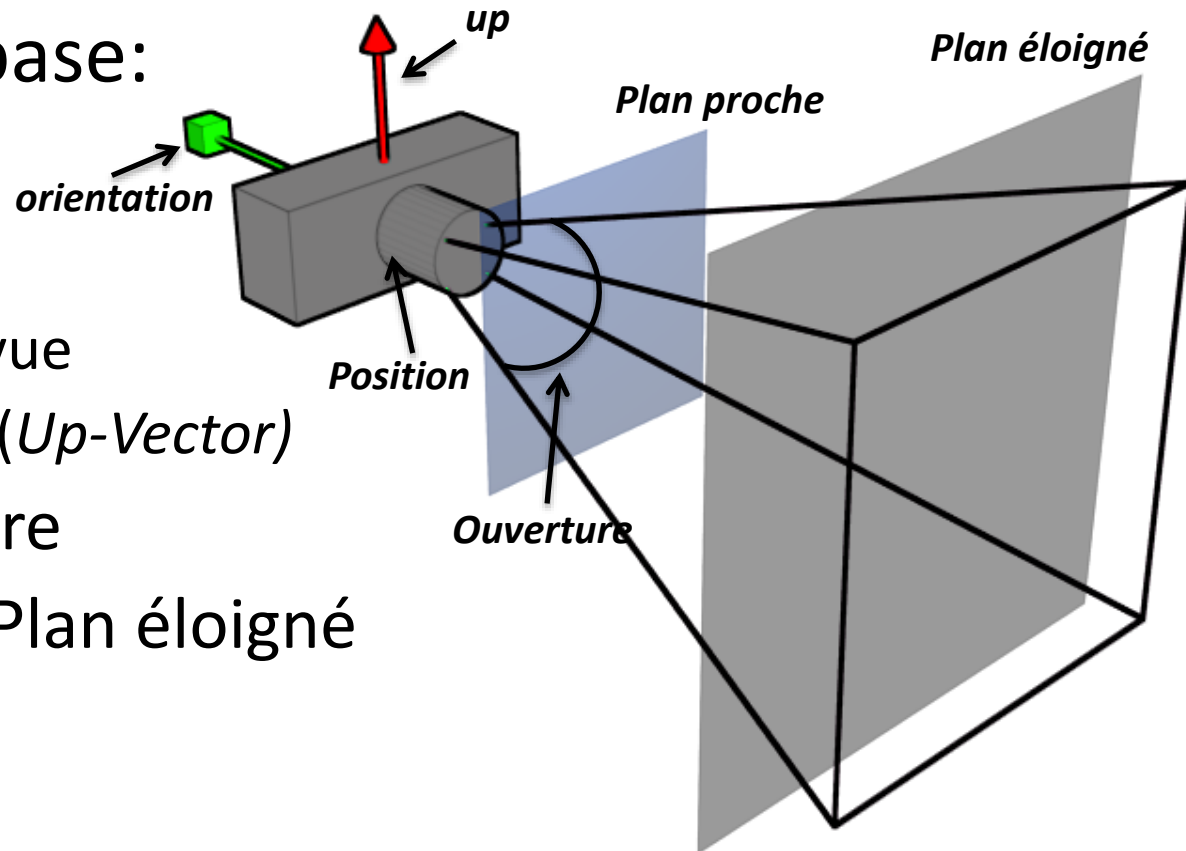
- Une vaste communauté en ligne
- Une référence : [www.opengl.org](http://www.opengl.org)
- Quelques programme d'exemples sur mon site
  - Notamment gmini :  
<https://github.com/superboubek/gmini>



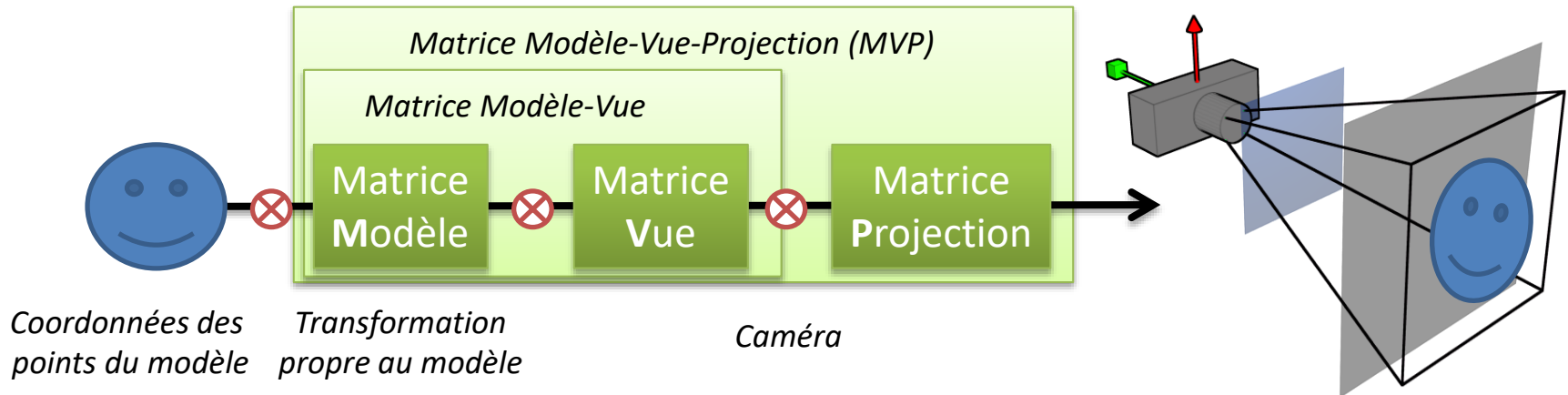
# **CAMÉRA ET TRANSFORMATIONS**

# Modèle de Caméra

- En général: *pinhole* camera
- Projection perspective
- Paramètre de base:
  - Position
  - Orientation
    - Direction de vue
    - Vecteur haut (*Up-Vector*)
  - Angle Ouverture
  - Plane proche/Plan éloigné



# Transformation et Projections

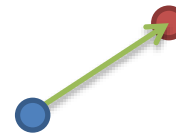


- Représentation par une matrice 4x4
- Transformation rigide
  - Translation
  - Rotation
  - Echelle
- Utilisation: changement de repère pour le placement des géométries dans le repère de la caméra et leur projection

# Transformation Affine

## Translation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



## Rotation

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix} \quad R_Y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix} \quad R_Z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

## Scaling

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

*Transformations appliquées pour :*

- *Déplacer les sommets des points 3D (matrice modèle)*
- *Les placer dans le repère de la caméra (matrice vue)*

# Projection

- Projeter les sommets des polygones (transformés) dans le plan de l'image.
- 2 types de projections:



- Encore une fois exprimable à l'aide d'une matrice 4x4 : la **matrice de projection**

# Matrice de projection en perspective

- Définie par les paramètres intrinsèque de la camera:
  - fov = ouverture
  - h = hauteur de l'image
  - w = largeur de l'image
  - n = distance au plan proche
  - f = distance au plan éloigné
- Forme de la matrice :
  - Pour un volume de vue symétrique
  - Avec

- $a = w/h$
- $t = \tan (fov/2)$

$1/at$	0	0	0
0	$1/t$	0	0
0	0	$-(f+n)/(f-n)$	0
0	0	-1	$-2fn/(f-n)$

# Géométrie Projective

- Raisonner dans l'espace des droites
- Projection en perspective

Point  $xyz > xyzw$  : **coordonnées homogènes**

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ w=1 \end{pmatrix} \rightarrow \dots \textit{transformations} \dots \rightarrow \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} \rightarrow \begin{pmatrix} x' / w' \\ y' / w' \\ z' / w' \end{pmatrix}$$

***Projection  
Homogène***

# Livres

- **Fundamentals of Computer Graphics**, by Peter Shirley, Steve Marschner, et alia, A.K. Peters, July 2009.
- **Realistic Ray Tracing**, Shirley & Morley
- **Real-Time Rendering**, Akenine-Möller, Haines, Hoffman
- Liste d'ouvrages:  
<http://www.realtimerendering.com/books.html>