Introduction à la 3D

Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

Introduction à la 3D - Plan

- Affichage 3D
- Vidéo 3D
- Synthèse d'images 3D
- Agents Conversationnels

Affichage 3D

Introduction à la 3D

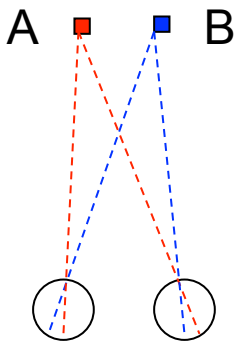
Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

Pourquoi l'affichage 3D?

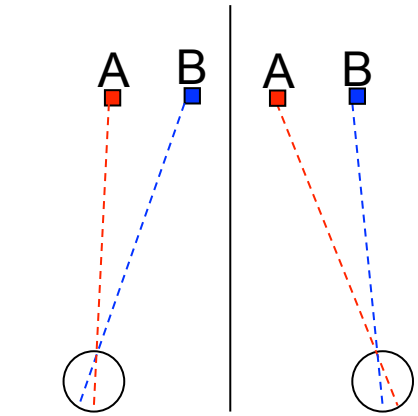
- **Vision naturelle de l'homme**
- **Besoin technique**
  - Création assistée par ordinateur
    - Aéronautique, automobile, …
  - Simulation physique réaliste
    - Médecine, aérospatiale, …
- **Besoin artistique**
  - Augmenter le réalisme multimédia
    - Jeux, Mondes Virtuels, …
  - Augmenter les capacités interactives d'une scène
    - Angles de vues, hot-spots, …

Stéréoscopie

- **« Tout procédé restituant une impression de relief à partir de deux images »**
  - Reconstruction du relief par le cerveau
- **« Inventée » en 1830-1840 (Sir Charles Wheatstone)**

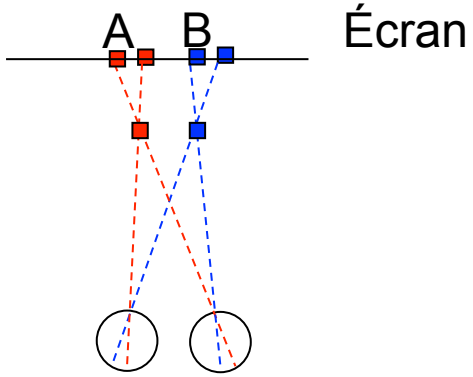


Vision Naturelle



Vue Gauche Vue Droite

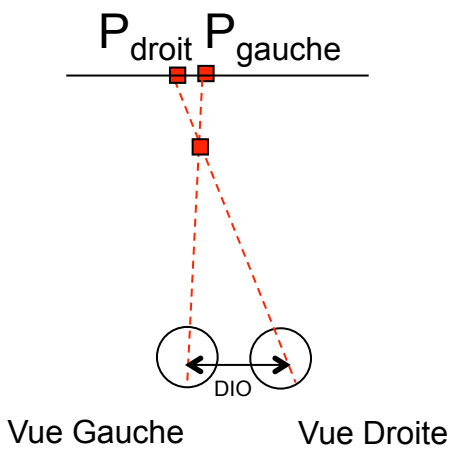
Stéréoscopie



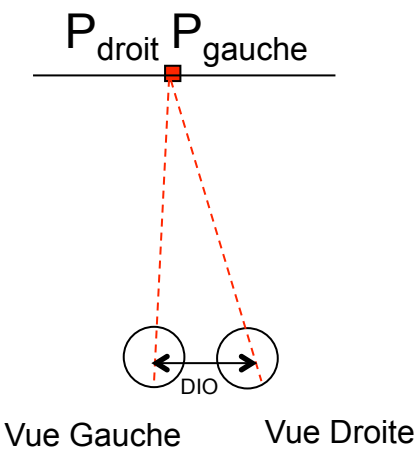
Vue Gauche Vue Droite

Stéréoscopie

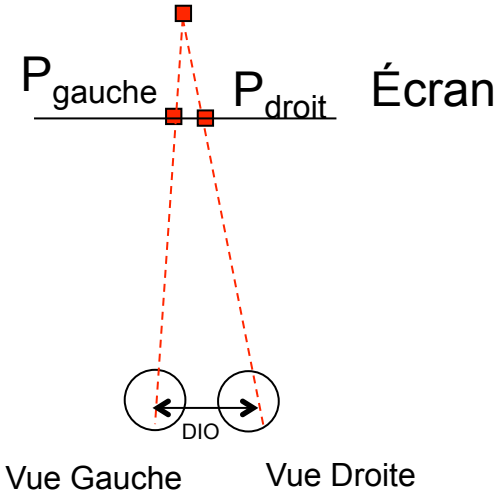
Perception profondeur



Point devant l'écran (jaillissement)



Point sur l'écran (2D)



Point derrière l'écran (profondeur)

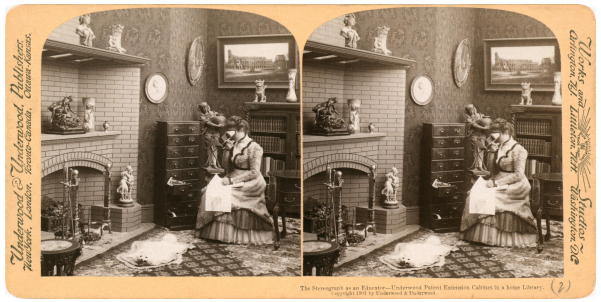
DIO: Distance interoculaire moyenne chez l'homme: ~6.4 cm

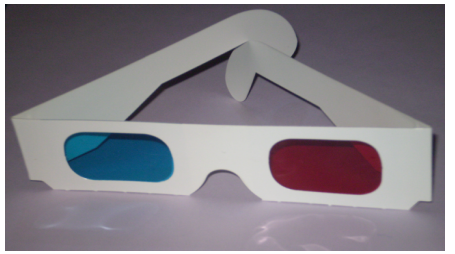
Stéréoscopie Parallèle - Exemple

Stéréoscopie Croisée – Exemple

Procédés

- **Anaglyphes**
  - Mélange de 2 vues stéréo
    - Recalage du sujet et déplacement gauche/ droite du fond
    - Filtrage composantes: Rouge(gauche) et Cyan (Droite)
    - Restitution via lunettes filtrantes





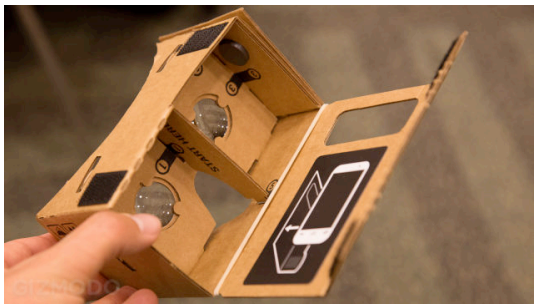
Procédés

- **Obturation**
  - Lunettes filtrantes (1 image gauche, 1 image droite)
  - Synchronisation source (écran, projecteur)
  - Très haute fréquence requise

- **Polarisation**
  - Projection simultanée des vues gauches et droite sur un écran avec polarisation 90°
  - Lunettes polarisée

Procédés

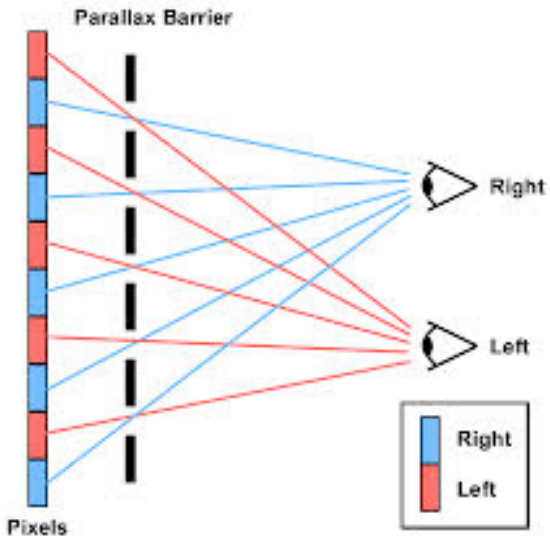
- **Casque VR**
  - Un écran partagé en 2 zone ou 2 écrans
    - Ou pico-projecteur
  - Option: Système optique
  - Option: Accéléromètre et gyroscope
- **Exemples**
  - Google Cardboard, Samsung Gear VR, Occulus Rift
    Lunettes 3D Vuzix

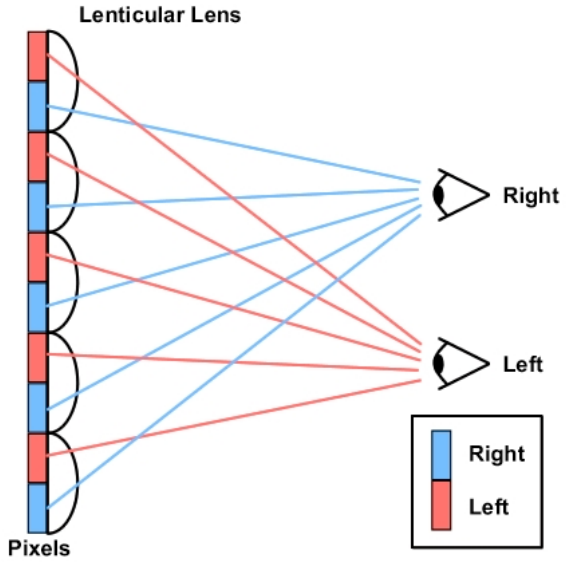


Procédés

Écrans LCD Stéréoscopiques

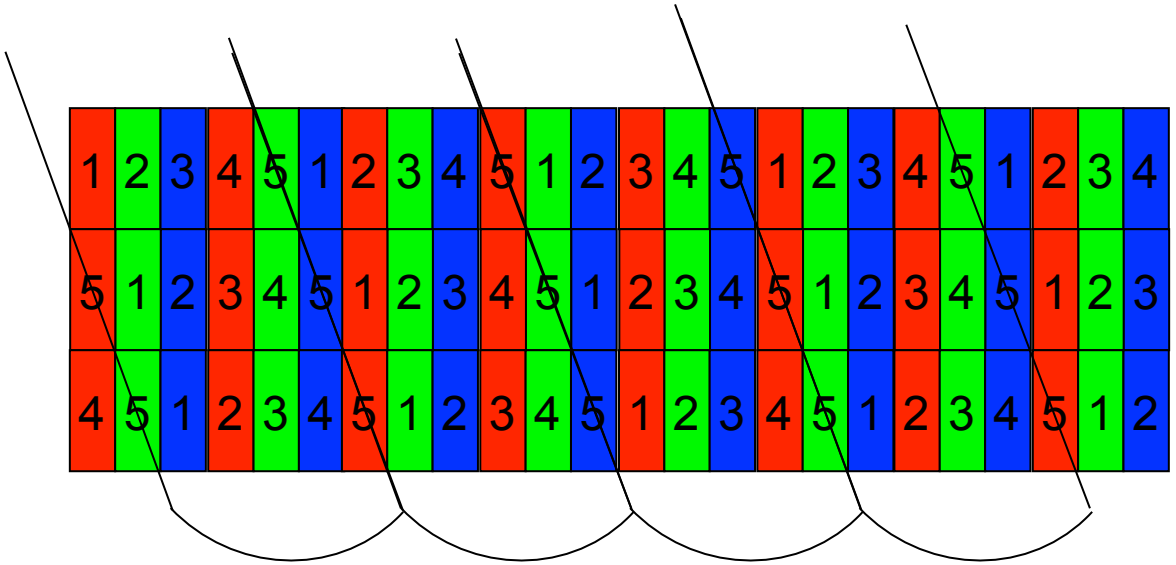
- Barrière parallaxe / réseau de lentille
- Réseau optique sur l'écran LCD
  - Avantage: Nombres de vues possibles >= 2
  - Inconvénient: Distance utilisateur-écran fixe





Procédés

- **Réseau Lenticulaire**
  - Entrelacement des pixels des différentes vues
  - Perte de résolution
    - Réseau oblique



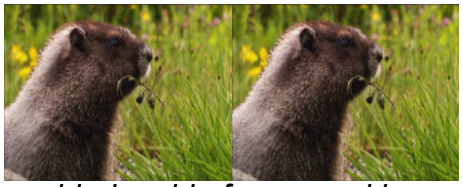
Vidéo 3D

Introduction à la 3D

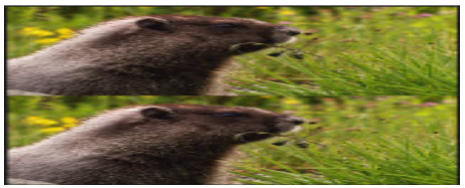
Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

Vidéo 3D

- **Vidéo Anaglyphes**
  - Une vidéo correspondant à l'entrelacement
  - Déformation des couleurs
    - Difficulté à compresser !
- **Vidéo stéréoscopique compatible décodage**
  - 2 vues regroupées dans une seule image

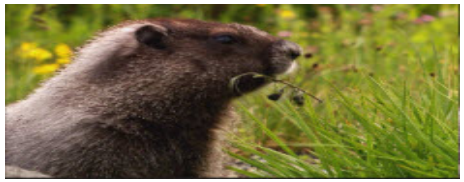


side-by-side frame packing



Top-to-bottom frame packing

  - Perte de résolution
- **Vidéo stéréo Blueray**
  - Une vidéo par vue
    - 2ème vue codée en « différentiel » de la vue principale



Vue #1: AVC\H.264

Vue #2: Vue #1 + MVC

Vidéo 3D

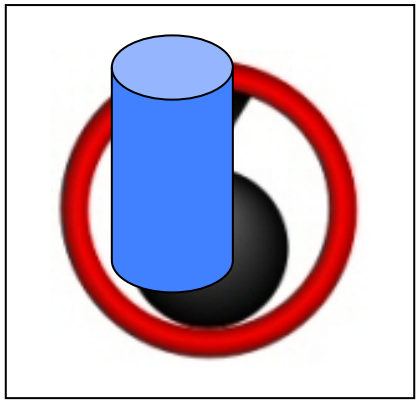
Vidéo Stéréo Cinéma

- Polarisation
  - Circulaire
    - Polarisation rectiligne sensible à l'angle des filtres !
  - Perte de luminosité
  - Quelques systèmes encore en obturation
- Deux projecteurs synchronisés
  - Real-3D: un seul projecteur
- Écran argentique
  - Garder la polarisation
- Projecteur à très haute fréquence (48 à 72 fps / vue)

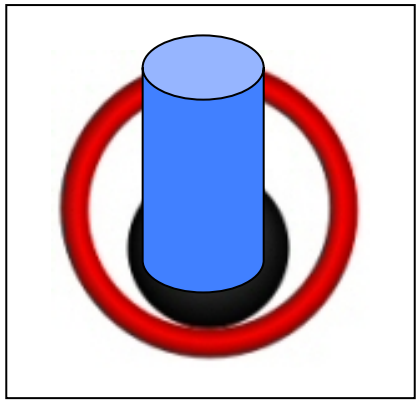
Vidéo 3D multi vue

- **Vidéo Multivue**
  - 1 vidéo par vue
  - Coûteux en bande passante
- **Vidéo + Z**
  - Une vidéo + une carte de profondeur
  - Reconstruction des vues à partir de la profondeur
  - Efficacité de codage
  - Problème des désocclusions

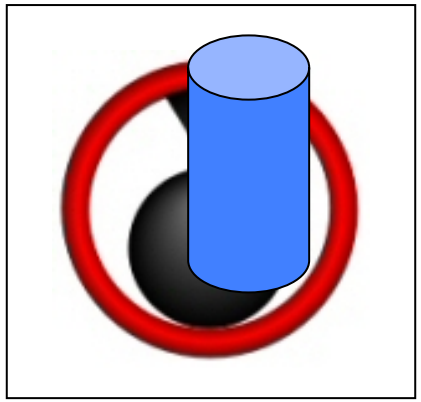
Vidéo 3D et désocclusions



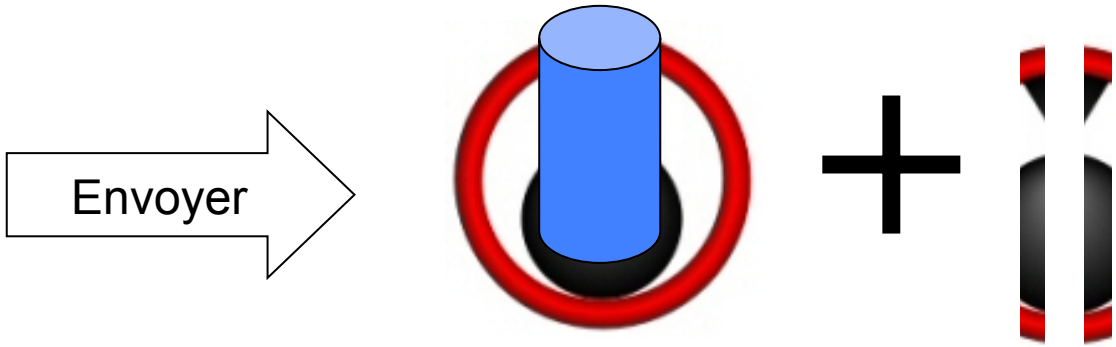
Vue 1



Vue 2

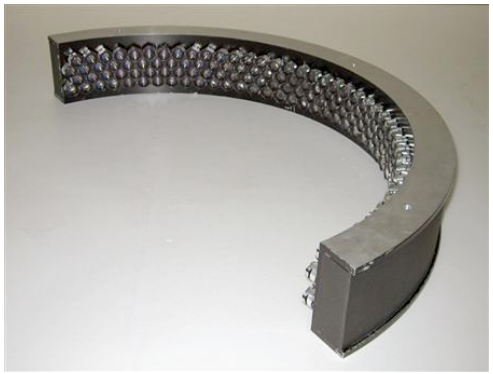


Vue 3



Acquisition 3D

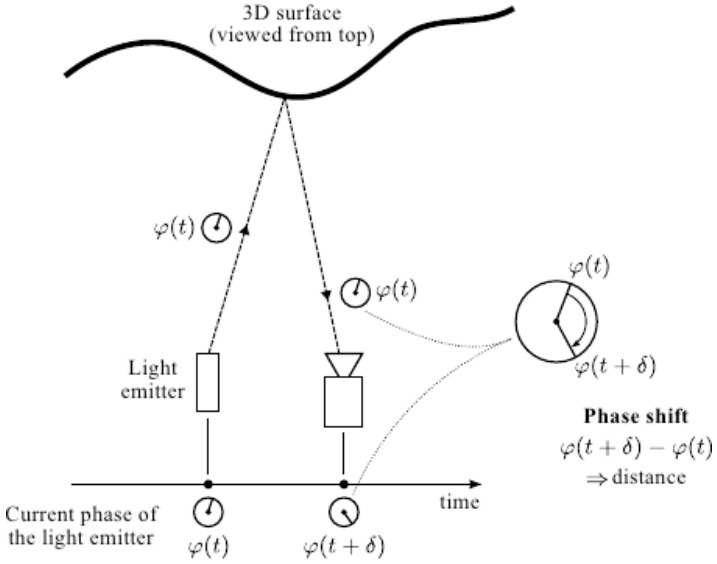
- **Caméra stéréo ou bancs de caméras**



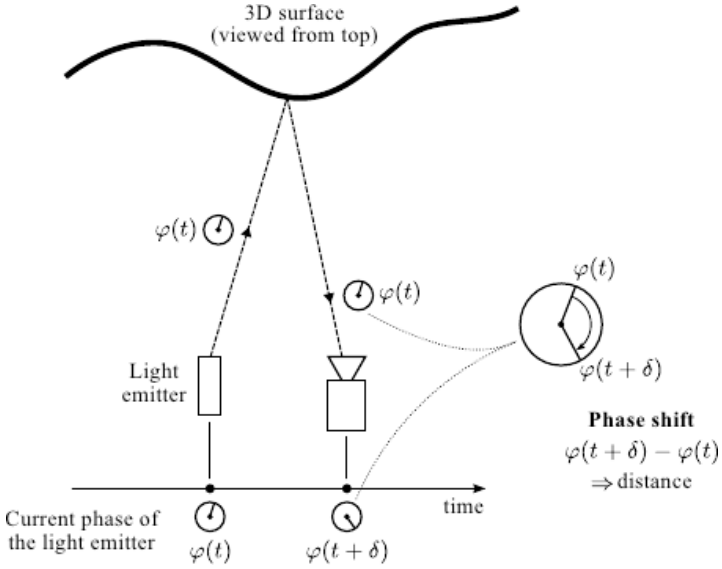
- Très précis
- Cout de stockage, de bande passante du système, etc

Acquisition 3D

- **LIDAR**
  - Décalage temporel
  - Précis
  - Point par point
    - Temps de capture long



- **Time of Flight**
  - Décalage de phase
  - Influence des matériaux
    - Moins Précis
  - Toute la scène d'un coup



Acquisition 3D

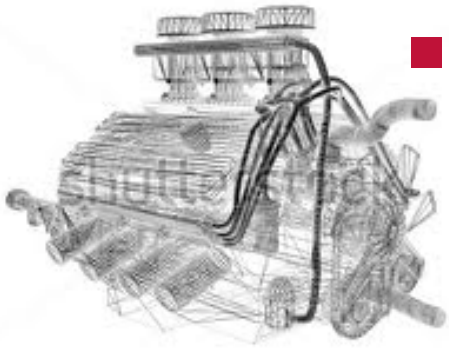
- **Kinect**
  - Projection IR d'un nuage de taches
  - Estimation des déformations
  - Sensible à la lumière naturelle (soleil)
  - Précision 0.4m (v1) / 0.8m (v2) ->5m
- **Kinect SDK**
  - Extraction des coordonnées du squelette
  - SDK2: Kinect dans IE 10 / HTML5
- **Avantage de la carte de profondeur**
  - Segmentation simplifiée
    - Isolation d'objets, de personnages
  - Reconstruction 3D simplifiée

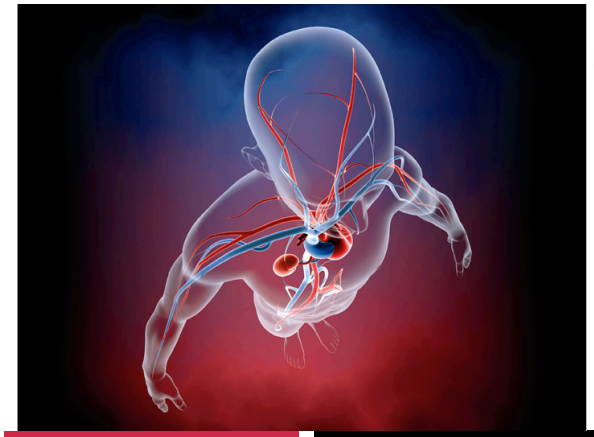
Synthèse 3D

Introduction à la 3D

Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

Pour qui ?



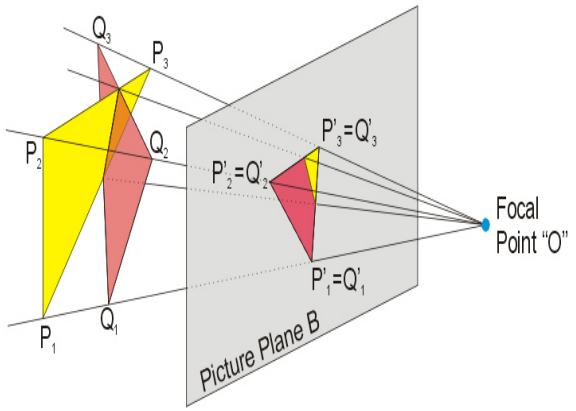


- **Divertissement**
  - Jeux: FPS, arcades, plateaux
  - Simulation (arts plastiques, musique, ..)
- **Industriel**
  - CAO
  - Tests et simulations
  - Impression 3D
- **Tertiaire**
  - Médical
  - Aérospatial
  - Architecture
  - …

Rendu 3D

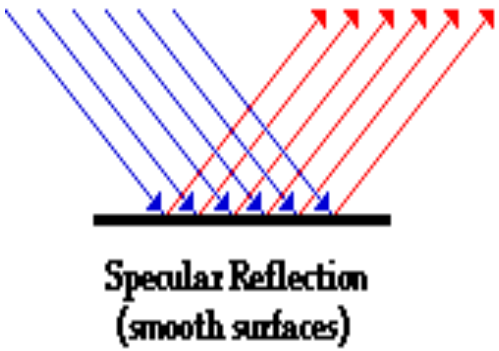
Géométrie

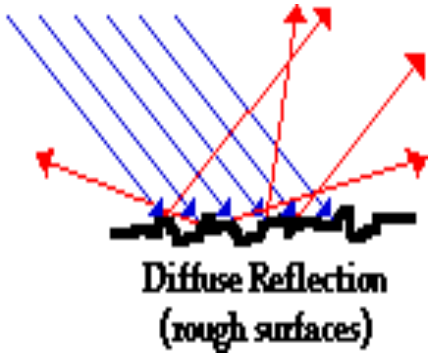
- Projection de données 3D sur un plan 2D
  - Projection perspective ou orthogonale
- Primitives: Triangles ou Carrés
- Élimination des objets
  - « Culling »
  - Éliminer les faces non visibles
    - Non présente dans la zone de vu
    - Masquée par une autre face

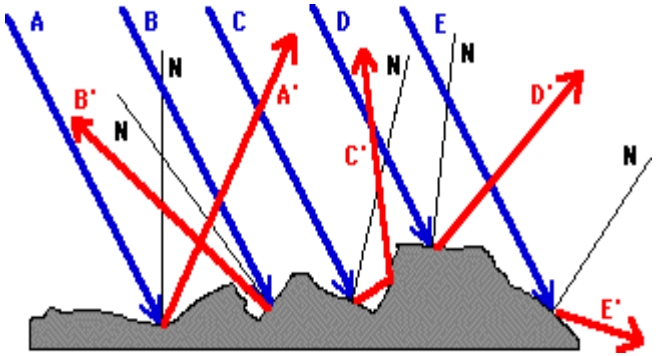


Synthèse 3D: Lumières

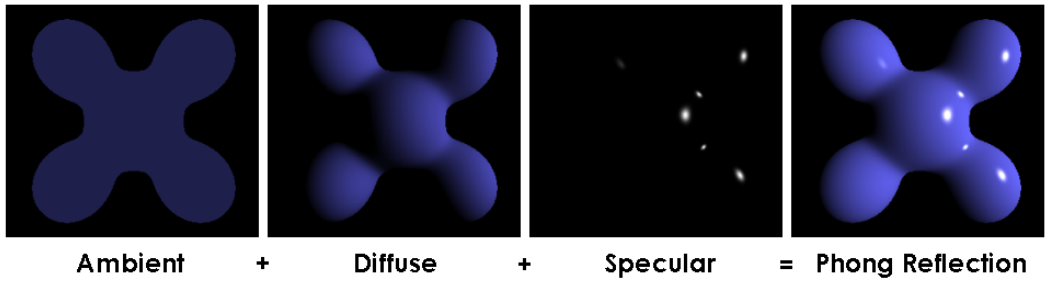
- **Illumination des objets**
  - Reflexion de la lumière







  - Modèles d'illumination
    - Modèle de Phong

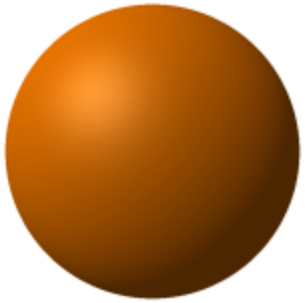
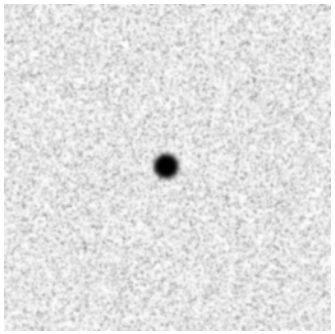


Synthèse 3D: Lumières

- **Modélisation des Types**
  - Ambiante: illumine toute la scène sans ombrage
  - Directionnelle: illumine toute la scène dans une direction donnée
  - Point: illumine la scène localement dans toute les direction
  - Spot: illumine la scène localement selon un cône de lumière
- **Distance**
  - Simule la puissance de la lumière
    - Atténuation nulle, linéaire ou quadratique

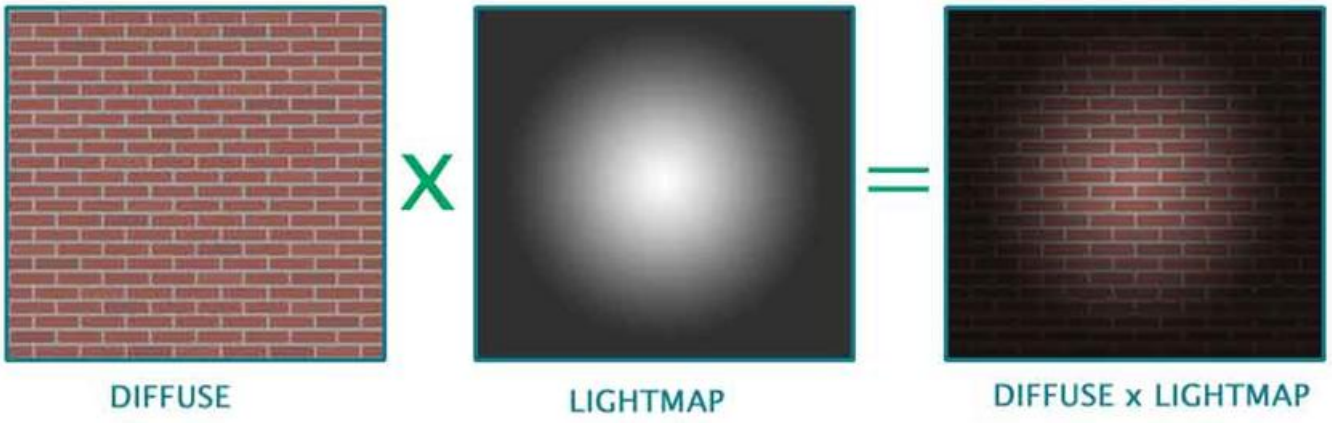
Synthèse 3D: Lumières

- **Ombrages des surfaces (« Shading »)**
  - Flat: couleur unie sur la face éclairée
  - Gouraud: interpolation linéaire des couleurs à partir des points de la face
  - Phong: Shading
  - Bump Mapping & Displacement Mapping:
    - Altération des normales

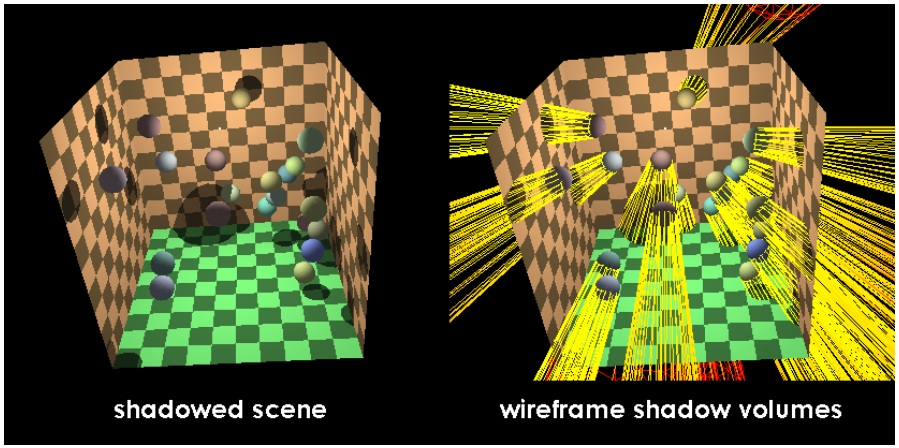

Synthèse 3D: Réalisme Avancé

- **Textures et relief**
  - Parallax Mapping
- **Multitextures**
  - Light Maps



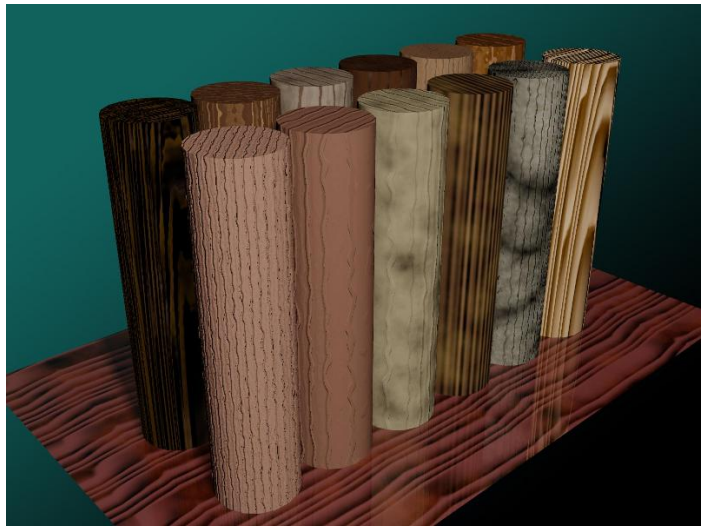
Synthèse 3D: Réalisme Avancé

- **Ombres**
  - Shadow Volume
  - Shadow Mapping
- **Reflexion Mapping**
  - Sphérique ou Cubique
- **Modèles Physique**



Synthèse 3D: Réalisme Avancé

- **Shaders**
  - Fonctions définies par l'utilisateur
  - Exécutées par le matériel
  - Manipulation de vertex et de pixel
- **Utilisation**
  - Illumination avancée
  - Systèmes de particules
  - Textures procédurales



Synthèse 3D: Vite ou beau ?

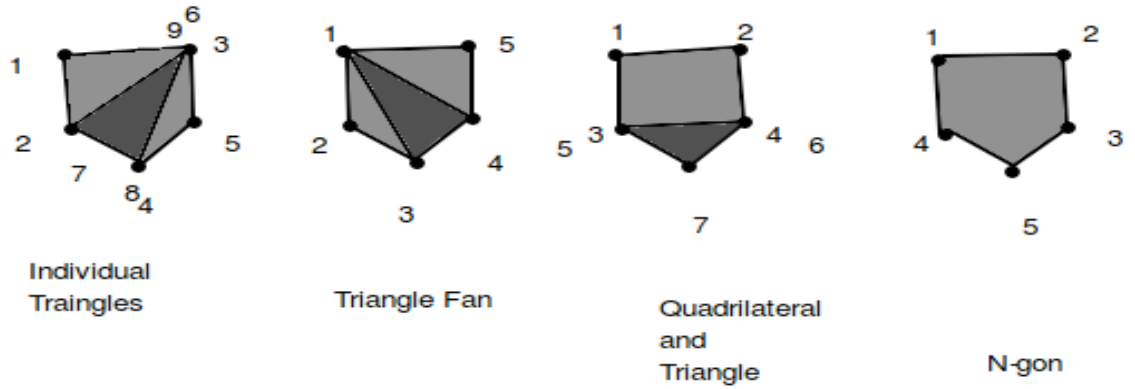
- **Applications temps réel:**
  - 12 -> 1000 Images / seconde
  - Moins fidèle à la réalité
  - API materielle: OpenGL ou DirectX
- **Applications Photo-Réaliste**
  - Pas de limite de temps
  - Ray-tracing, Ray-casting …
  - Pixar’s RenderMan

Déploiements 3D: Accélération Matérielle

- **GPU: Graphics Processing Unit**
  - Implémentation matérielle des opérations couteuses
    - Calcul vectoriel/matriciel
    - Gestion des textures
    - Interpolation des couleurs
    - Dessin des pixels, gestion de la profondeur (Z-buffer)
  - Version logicielle (via pilote de carte) pour le reste
    - Pas toujours de garantie sur ce qui est accéléré ou non
  - Traitement en parallèle des primitives
- **Qui ?**
  - Direct3D (Microsoft)
  - OpenGL (Khronos)
  - OpenGL-ES (Khronos)
    - WebGL: OpenGL-ES via JavaScript et navigateur

Principes de l'accélération matérielle 3D

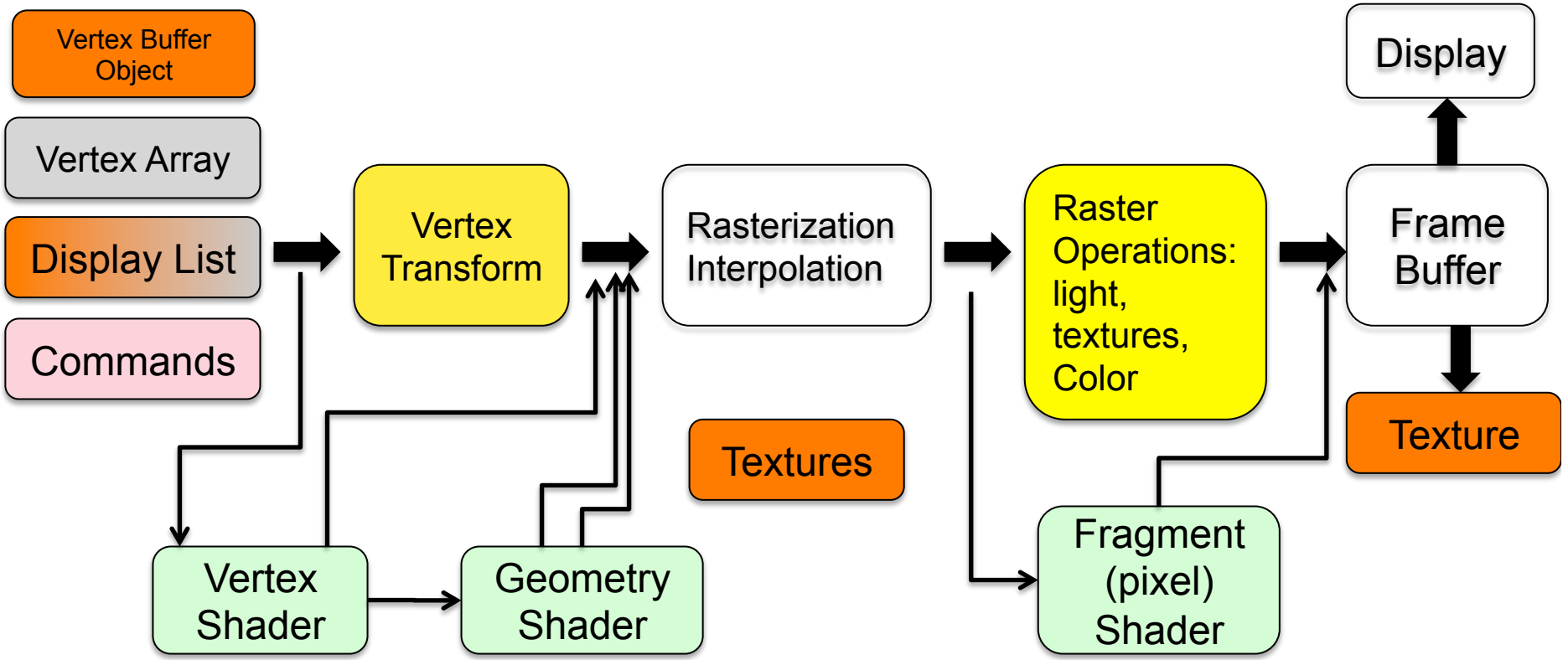
- **Suite d'instructions programmant le GPU**
  - Sortie directe sur la mémoire vidéo
    - Extensions pour le rendu hors écran
- **Primitives**
  - Simples: triangle, carré (quad), polygone convexe simple
  - Complexe: « soupe de triangle » + indexes



Principes de l'accélération matérielle 3D

- **Version « fixed pipeline » (old school)**
  - Variables d'état
    - Lumière, couleurs, paramètres de filtrages
    - Options de ciseaux
    - Transparence, brouillard, …
  - Piles de variables
    - Matrices de transformations (texture, modèles et perspectives) et états actifs
    - Push/pop
- **Version Shader**
  - Notion de programme (langage: GLSL)
  - Le programme décrit toutes les manipulations à faire:
    - Sur les vertex
    - Sur les pixels
  - Seule version disponible dans WebGL et GLES2

GPU Pipeline



CPU Mem | GPU Mem | Fixed Pipeline | Programable Pipeline

Programme simple typique

```
main()
{
//1- mise en place du programme

//2- mise en place OpenGL

while (!fin) {
        //3- interactions du programmes, mise à jour des variables (dont le temps
pour les animations), etc

        //4- identification des objets actifs pour la trame courante (liste
graphique), chargement de nouveaux objets / textures

        //5- rendu openGL – PAS DE MODIFICATIONS DES VARIABLES D'ETAT NON-OPENGL
SI PLUSIEURS PASSES NECESSAIRES (ex: stereo ou multivue)!


        //6- afficher la trame
        glSwapBuffer();
}

//nettoyage des ressources
}
```

Les Shaders

- **Vertex Shader**
  - Modifie les attributs d'un vertex
    - Position, couleurs, coordonnées de textures, etc
  - Application des matrices (transformations + projection)
    - Typiquement, les normales sont ajustées pour l'éclairage
- **Fragment Shader**
  - Modifie les attributs d'un pixel
    - Couleur/texture, transparence
  - Pixel écrit en mémoire vidéo si test profondeur OK
- **Autres**
  - Geometry Shader: permet de cloner des vertex à la volée (rendu multiple d'objet)
  - Tesselation Evaluation Shader
- **Programme**
  - 1! vertex shader + 1! fragment shader
  - En option, 1 geometry shader, 1 TES

Le Langage GLSL

- **But**
  - Interopérabilité de la programmation entre GPUs
  - Simple et proche du C
  - Permettre l'execution parallèle des instructions de rendus
    - Ex: 32 triangles en //
- **Problèmes**
  - Différentes versions du langage
    - Entre génération de cartes
    - Entre desktop (OpenGL) et Mobile (OpenGL ES)
- **Principes**
  - Lors de la compilation, définition
    - Variables locales et code
    - Données d'entrée (du précédent shader)
    - Données de sortie (pour shader suivant ou résultat final)
  - Avant de dessiner
    - Paramètres des shaders liés au programme ou « uniform »
      - Constant pour tout l'objet
        - Couleur, matrices, vecteurs, etc …
    - Attributs de l'objet (« attributes »)
      - Données changeantes par vertex

Exemples triviaux de shader (GLSL<3)

Vertex Shader

```
#version 110

void main()
{
    gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex;
}
```

Fragment Shader

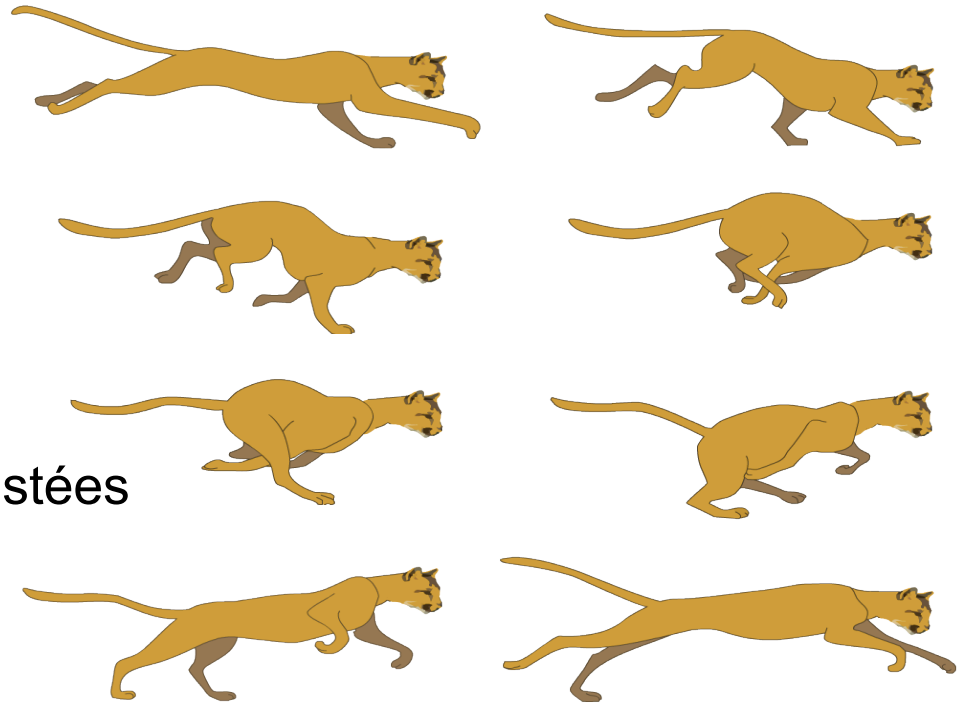
```
#version 110

void main()
{
     gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

WebGL

- **WebGL**
  - API Javascript de control du GPU
    - Portabilité entre OS
    - Support sur mobiles pas encore très bon
  - Utilisation de <canvas> de HTML5
    - Création d'un contexte OpenGL
    - Rendu via shaders GLSL
  - Animation
    - Via JS setInterval ou requestAnimationFrame
  - Textures
    - Via <img> and <video> tags
- **Pour aller plus loin**
  - http://learningwebgl.com/blog/
  - http://www.glge.org/

Animation de personnages par sprite

- **Texture**
  - Toutes les positions du personnage
- **Géometrie**
  - Carré toujours en face de la caméra (« billboard »)
  - Coordonnées de texture ajustées en fonction du temps
- **Simple et rapide**
  - Moins esthétique …



Animation par déformation

- **Définir**
  - une armature
    - Liste de segments de droite (« bone »)
    - Définie pour une pose A donnée
  - une enveloppe
    - Définie dans la pose A
    - Chaque vertex est associé à un ou plusieurs « bones » avec un poid P_i / bone
      - La somme des poids = 1
- **Animer**
  - Déformation de l'armatures via matrices de A vers la position souhaitée
  - Transformation du vertex = somme des transformations/bone du vertex, pondérées par le poids

Pour aller plus loin

- **Gestion des modèles**
  - Manipuler simplement des objets 3D complexes
  - MD2, MD5, Collada
  - Multiples librairies possibles (dont WebGL)
- **Gestion de la physique**
  - Simuler le monde réel de manière convaincante
    - Déplacement: Gravité, frottement, collisions et forces de réaction
    - Déformation: élasticité globale, déformation liquides, solides, systèmes particulaires (herbe, cheveux, …)
      - http://madebyevan.com/webgl-water/
      - http://threejs.org/examples/webgl_animation_cloth.html

Agents Conversationnels

Introduction à la 3D

Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

Agents Conversationnels Personnifiés

- **Personnages virtuels**
  - Interactifs, parlants, avec gestuelle, autonomes
    - Rendu réaliste
    - Synchro Audio, Visuel et stimuli environnements
  - Le plus souvent, anthropomorphique
- **Autonomie**
  - ACP planifie (quoi, quand) son discours et ses réponses (prendre part à la conversation)
- **Nouvelles interactions homme-machine**
  - Etude de la communication humain-humain

Défis pour les ACP

- “Non adapted emotional displays influence the user's evaluation of the agent negatively”
  (Walker et al., 1994, Becker et al., 2005, Prendinger and Ishizuka, 2001)
- “Appropriate emotional displays increase the agent’s believability” (Lim and Aylett, 2007)

→ importance du comportement émotionnel

→ socialement adapté

ACP

- **Simuler les capacité cognitives et expressive de l'humain**
- **Différents domaines d'études**
  - linguistique, phonétique, sciences cognitives, émotion, psychologie, sociologie
- **L'ajout d'émotions aux ACP renforce:**
  - La crédibilité de l'interaction
  - L'engagement dans l'interaction
  - La confiance

Agent Crédible

- **Definition - Loyall 1997: *« a character is considered to be believable if it allows the audience to suspend their disbelief »***
- **Cela comprend:**
  - Apparence physique
  - Emotions et personnalité
  - Capacités sociales
  - Actions et comportements cohérents
- **L'agent crédible:**
  - Peu agir sur le comportement de l'utilisateur
    - Faire acheter un produit
  - Permet confiance, implication émotionnelle, compréhension:
    - Est conscient des problèmes de l'utilisateur
    - Interagit avec l'utilisateur, montre de l'empathie, de la compréhension
    - …

Merci !

Introduction à la 3D

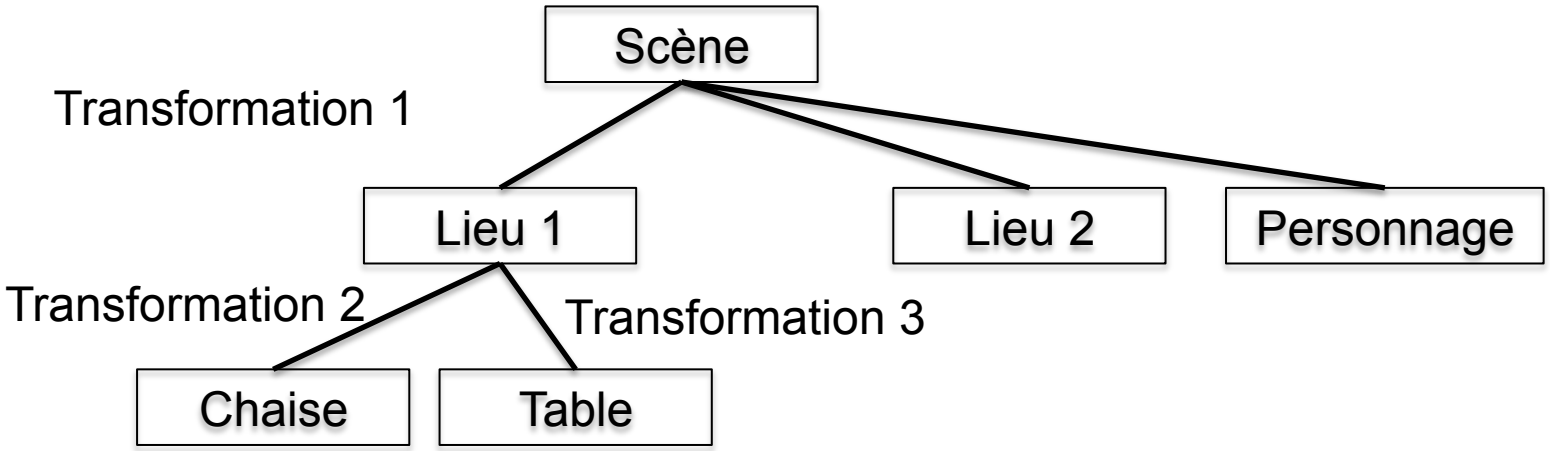
Jean Le Feuvre
jean.lefeuvre@telecom-paristech.fr

Annex 1
Bases de la synthèse 3D

Modéliser un environnement 3D

- **Objet: description mathématique 2D/3D**
  - Forme
    - Personnages
    - Décors
    - Terrain
  - Aspect
    - *Textures*, couleurs, modèle de lumière
- **Scène: ensemble des objets**
- **Caméra: point de vue dans la scène**
  - Unique pour tous les objets
- ***Modèle pour simulation physique***
  - *Poids, élasticité des matériaux, …*

Arbre de scène et Liste d'affichage



Liste Graphique, dans l'ordre d'affichage:

- Transformation 1 + Transformation 2 + Chaise
- Transformation 1 + Transformation 3 + Table
- Transformation 1 + Transformation X + Personnage

Dans la pratique, on utilise souvent un mélange des deux:

Arbre de scène pour décrire l'ensemble

Liste Graphique pour les objets à afficher à T

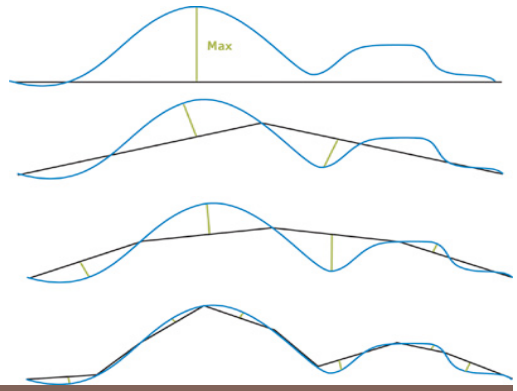
Les objets

- **Primitive: description mathématique 2D/3D**
  - Point: « Vertex »
  - Primitives simples
    - Point, Ligne, Triangles, Carrés
  - Primitives complexes:
    - courbes/surfaces de Bézier, d'ordre 2 ou 3

$$\mathbf{B}(t) = (1-t)^2\mathbf{P_0} + 2t(1-t)\mathbf{P_1} + t^2\mathbf{P_2} \ , \ t \in [0,1].$$

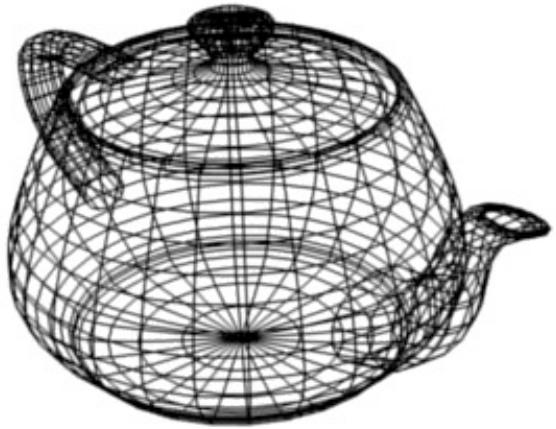
$$\mathbf{B}(t) = \mathbf{P_0}(1-t)^3 + 3\mathbf{P_1}t(1-t)^2 + 3\mathbf{P_2}t^2(1-t) + \mathbf{P_3}t^3 \ , \ t \in [0,1].$$

    - Puis *tesselation*
      - Approximation linéaire par primitive simples
      - Pour une « finesse » donnée
      - Accélération matérielle très récente …

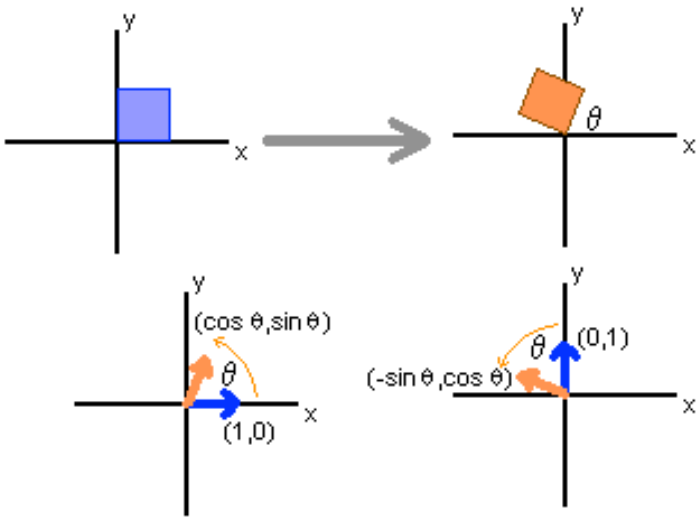


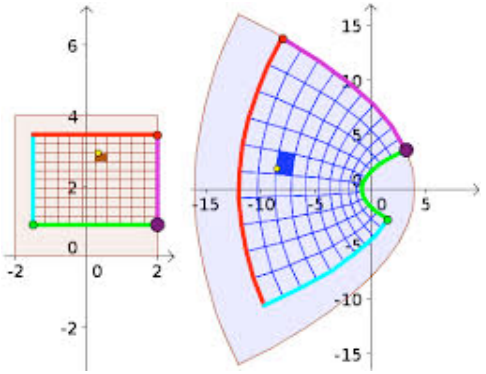
Les objets

- **Objet**
  - Assemblage de ces primitives
    - Notion de « groupe »
    - Non déformable: même aspect à chaque trame
      - Ex: Chaise, table
    - Déformable
      - Animation de personnage
      - CAO Industrielle
  - Via des transformation affines



Les Transformations

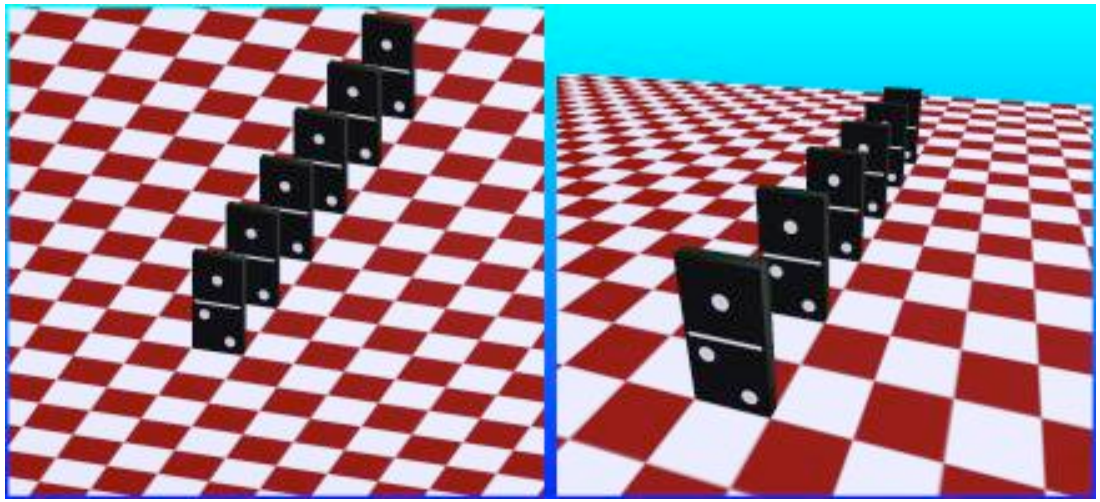




- **Déplacer les objets**
  - Rotation, translation, mise à l'échelle
    - Calculs simples sur l'ensemble des points
  - Transformations affines
    - 2D: x'= ax + by + c, y' = dx + ey + f
    - 3D: …
  - Transformations non linéaires
    - Modification point par point
- **Cumuler les transformations**
  - (x', y) = R(alpha)(x, y) + T(a, b)(x, y) + S(2, 1)(x,y) + …
  - Représentation Matricielle:
    - Tr = S * T * Ra
    - (x', y') = Tr * (x, y)

$$\begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix}$$

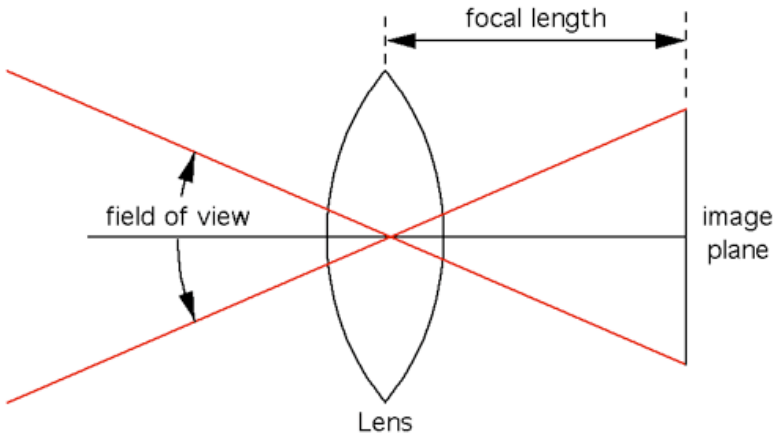
Orthogonal vs Perspective



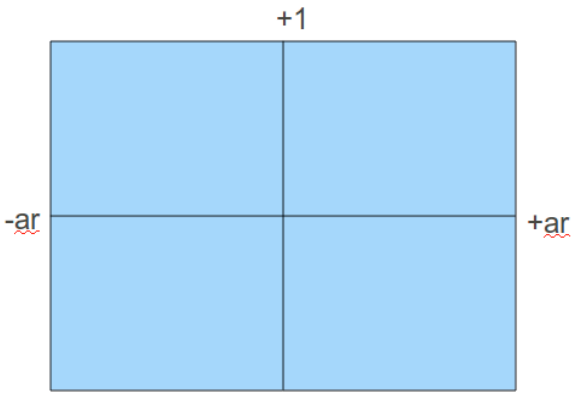
- **Projection Orthogonale**
  - Angles droits respectés
  - 2D, CAO 3D
- **Projection Perspective**
  - Point de fuite au centre de l'image
  - Objets lointains plus petits
  - Monde virtuels, jeux, …

Paramètres de perspective

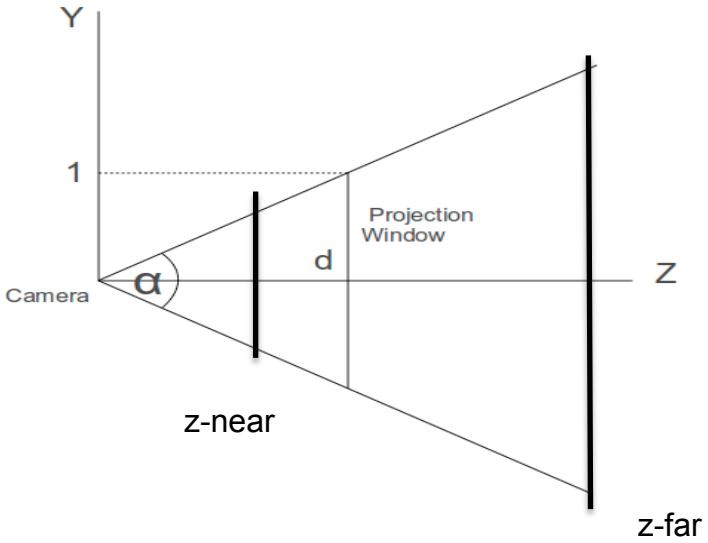
- **Z-far**
  - « far clipping plane »: les objets derrière ce plan ne sont pas dessinés
- **Z-near**
  - « near clipping plane »: les objets devant ce plan ne sont pas dessinés
- **Aspect Ratio**
  - Rapport largeur/hauteur
- **Notion d'ouverture**
  - En optique, notion de focale
  - En 3D, « field of view »



La matrice de perspective



Fenêtre virtuelle de projection



Coordonnées normalisées sur [-1,1]

$$x_p = \frac{x}{ar \cdot z \cdot \tan(\frac{\alpha}{2})}$$

Division par z pas pratique:

- z varie à chaque vertex
- ce n'est pas un produit matriciel !

$$y_p = \frac{y}{z \cdot \tan(\frac{\alpha}{2})}$$

La matrice de perspective

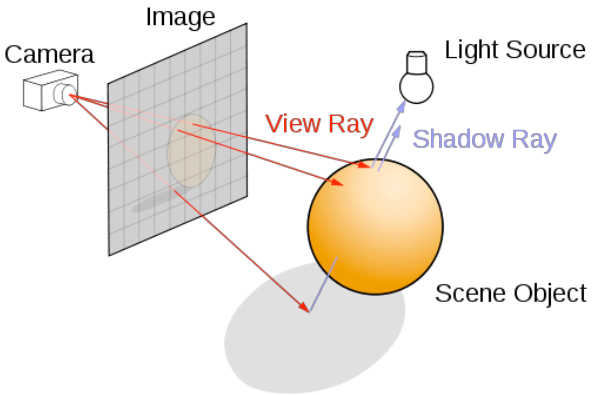
- **Division par Z gérée par le matériel**
  - « perspective division »
  - Pour un « z » normalisé entre « z-near » et « z-far »

$$\begin{pmatrix} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{-NearZ - FarZ}{NearZ - FarZ} & \frac{2 \cdot FarZ \cdot NearZ}{NearZ - FarZ} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- **Dans la pratique**
  - gluPerspective: Utilisation de FoV, z-near, z-far
  - glFrustum: Utilisation de haut/bas/gauche/droite (fenêtre de projection), z-near et z-far

Rendu des objets

- **Photo-Réalisme**
  - Propagation de la lumière dans la scène virtuelle
- **Techniques**
  - Ray Tracing
    - Pour chaque pixel de l'écran, trouver toutes les contribution au rayon de lumière arrivant à la camera
  - Ray Casting
    - Uniquement les contributions primaires
- **Inconvénients**
  - Très coûteux
  - Pas adapté au temps réel
  - Nécessite de connaître toute la scène au moment des calculs
    - Bande passante, stockage, etc …



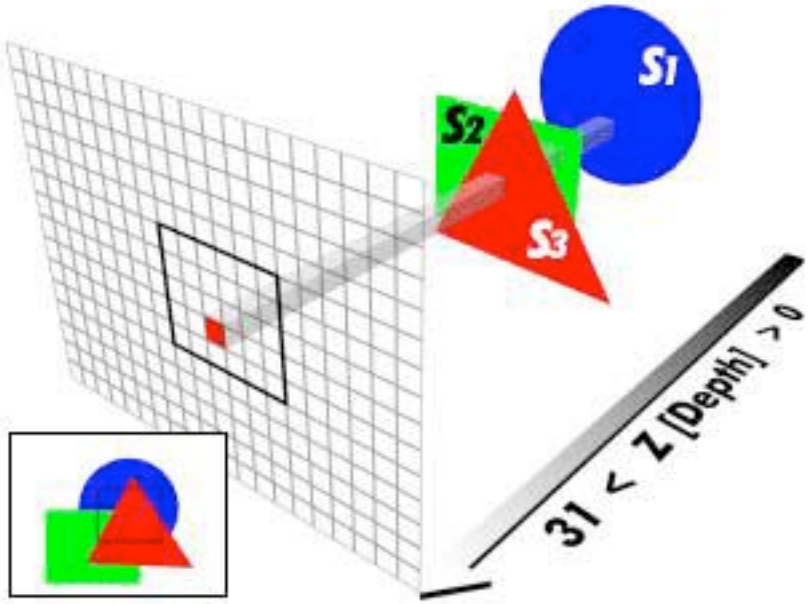
Rendu des objets

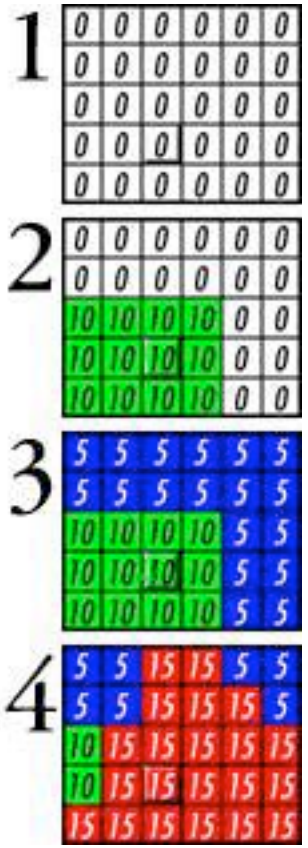
- **Via Carte Graphique (GPU):**
  - Dessin objet par objet
  - Primitives après projection: Point, lignes, triangles ou quadrilatères
  - Pour chaque point V dans la primitive correspondant à un pixel P,
    - Calcul de la couleur
    - Calcul de la profondeur Z
  - Si Z(V) > Z(P), pas de dessin
    - Autre objet plus proche
  - Si Z(V) <= Z(P), dessin
- **Z(P) est le z-buffer ou tampon de profondeur**
  - Stocke les valeurs de Z des pixels à l'écran
    - Non linéaire f(near, far, 1/z)
  - Précision finie: 8,16, 24 bits
  - Importance du choix de z-near et z-far

Une scène 3d simple

Le Tampon de profondeur

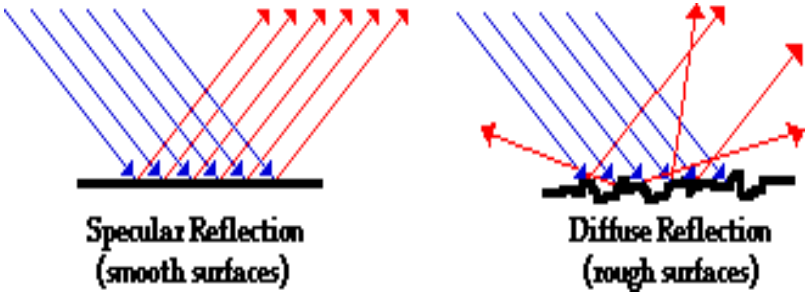
Principe du Z-buffer

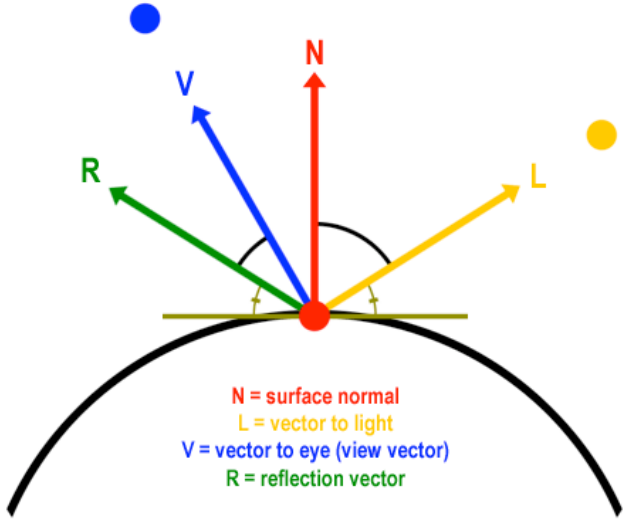




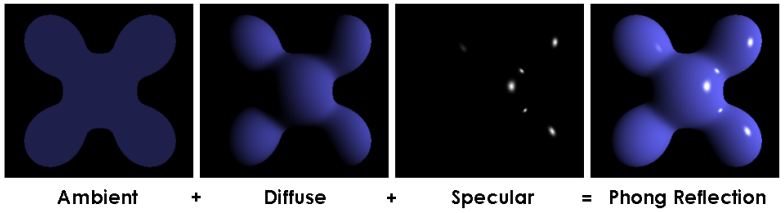
Modèles de lumières

- **Mélange entre « lumière diffuse » et « lumière spéculaire »**





- **Modèle de Phong**



$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{ lights}} (k_\mathrm{d}(\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s}(\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}})$$

- IMPORTANCE DES NORMALES

Modélisation des Lumières

- Ambiante: illumine toute la scène, sans « source » associée
- Point: illumine la scène localement dans toute les direction
- Directionnelle: illumine toute la scène dans une direction donnée
- Spot: illumine la scène localement selon un cône de lumière

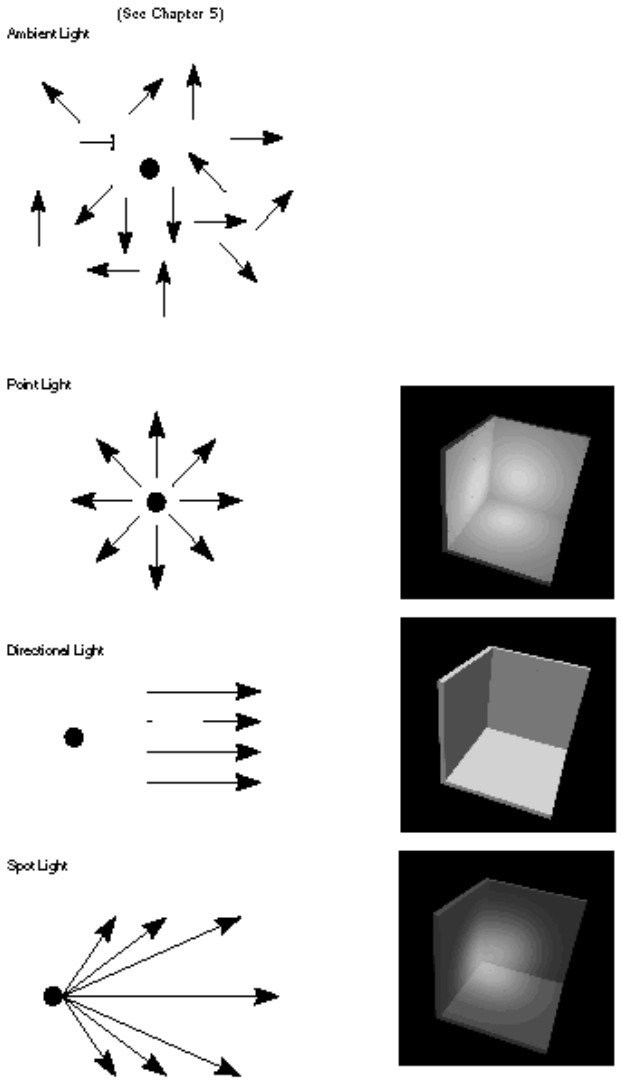
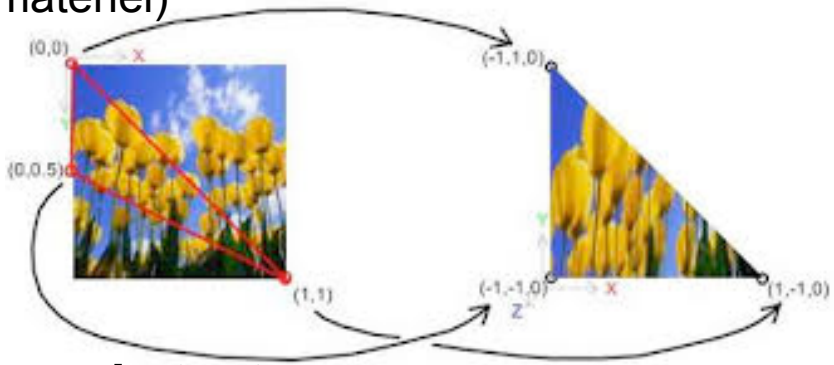


Image et Objets

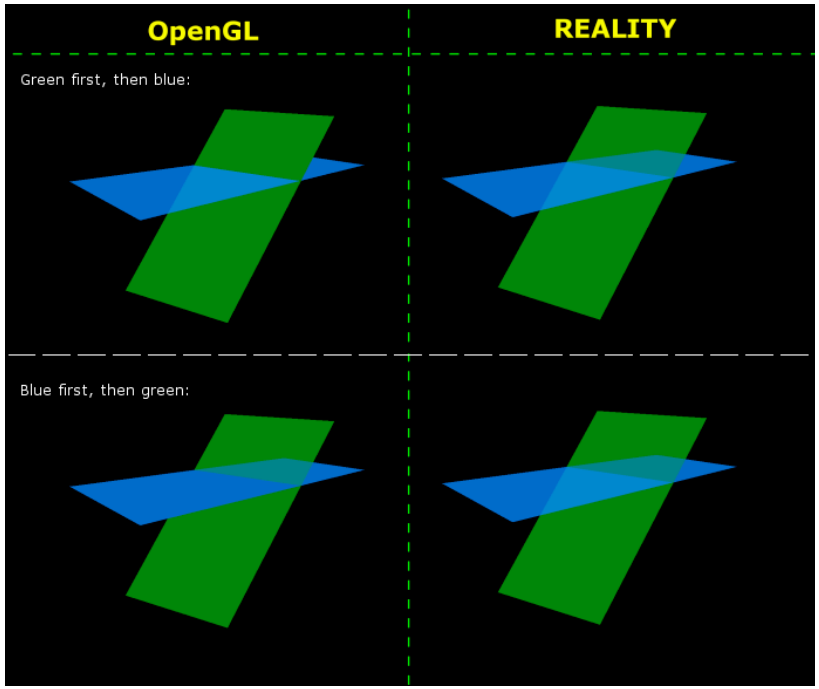
- **Associer une image à un objet**
  - Création d'une « texture », contenant une image RGB, RGB+Alpha, …
- **Associer une zone de l'image à une primitive**
  - Définition d'un espace de coordonnées pour les textures
    - U,V dans $[0.0,1.0]^2$
  - Définition de coordonnées de textures pour chaque point de la primitive
  - Interpolation pour les autres points de la face
    - Prise en compte de la perspective (via matériel)



- **Déformations possibles via des matrices de texture**

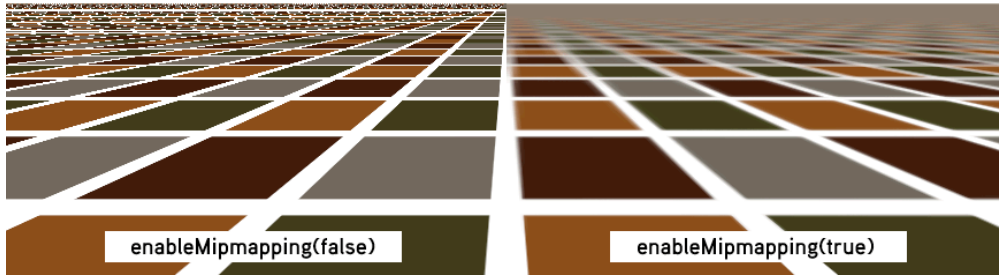
Objets et transparence

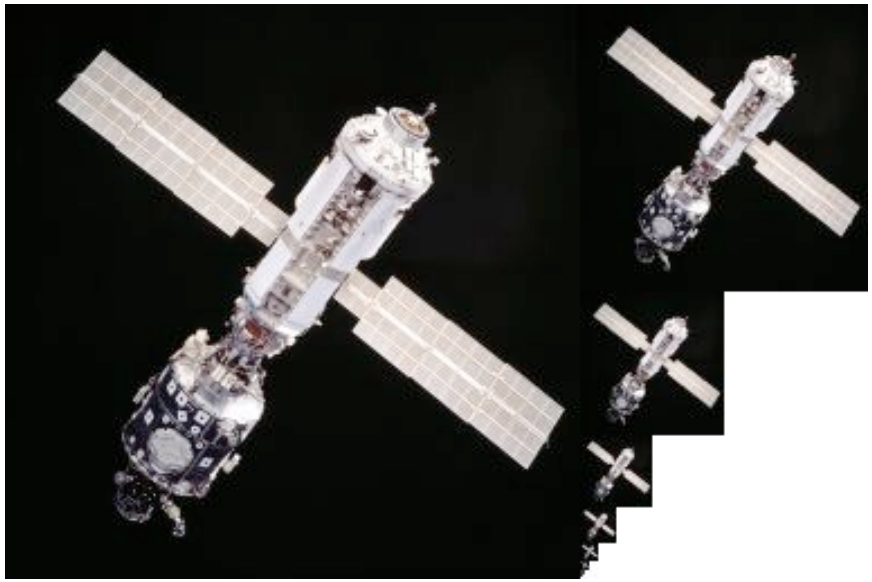
- **Problématique**
  - La carte graphique dessine les objets
    - Les uns après les autres
    - dans l'ordre demandé par l'utilisateur
    - Sans garder la couleur de chaque objet
- **Solutions**
  - Trier les objets
  - Si intersections:
    - Re-couper les objets
      - Lourd
      - Tri pas toujours possible
    - Dessiner en plusieurs passes
      - Pas simple …



Textures et niveau de détail

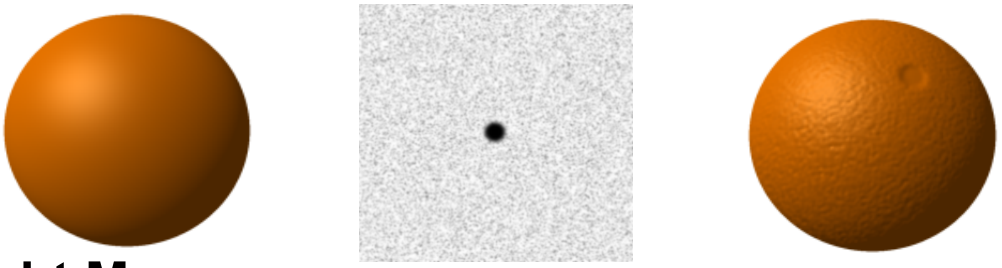
- **Taille= 2^n x 2^k**
- **Ex: Texture 512x512**
  - Si la face fait 3x3 , 50x50 pixel après projection ?
  - Pixelisation/crénelage
- **Mip-Mapping**
  - Versions de la texture à différentes résolutions
  - Améliore le rendu des objets lointains



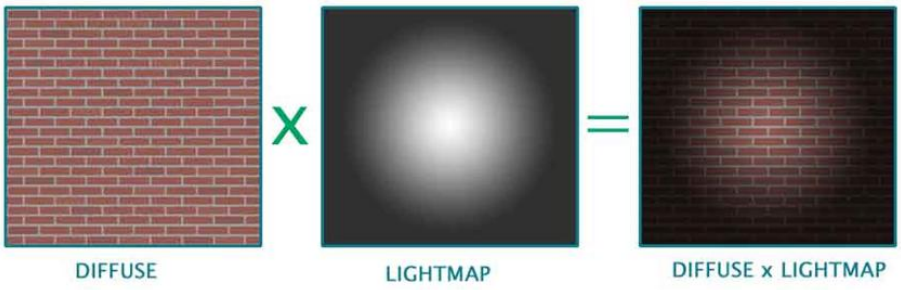


Textures avancées

- **Possibilité d'avoir plusieurs textures / objet**
  - Plusieurs coordonnées de textures par vertex
- **Bump Mapping & Displacement Mapping**
  - Altération des normales
  - Altération des vertex

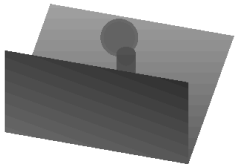


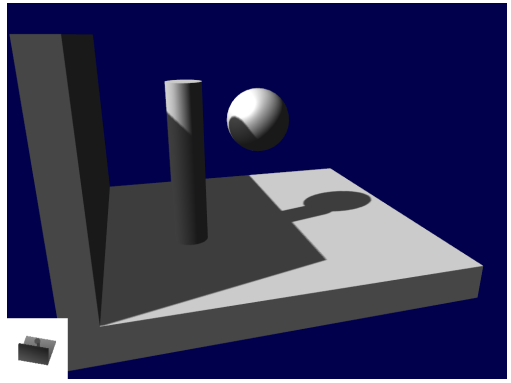
- **Light Maps**
  - Lumière pré-calculée et stockée en texture



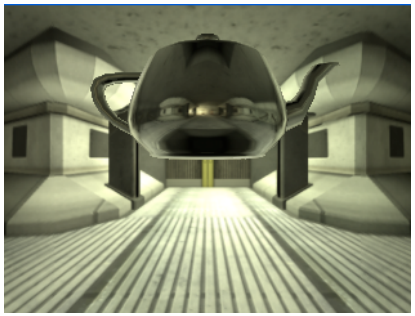
Synthèse 3D: Réalisme Avancé

- **Ombres**
  - Shadow Mapping
    - Scène vue depuis la lumière
      - « distance » pixel<->source
    - Deuxième passe
      - Distance-> ombre
  - Shadow Volume
    - Très complexe mais plus précise
  - Et bien d'autre encore
    - Selon les besoins…





- **Reflexion / Environment Mapping**
  - Monde vu depuis l'objet
  - Cube map



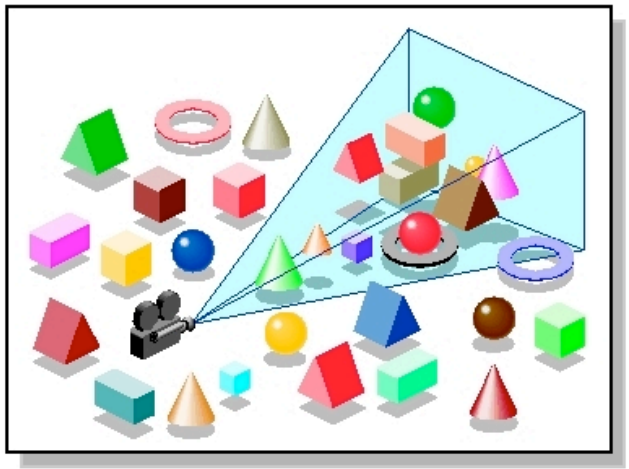
Gestion des données

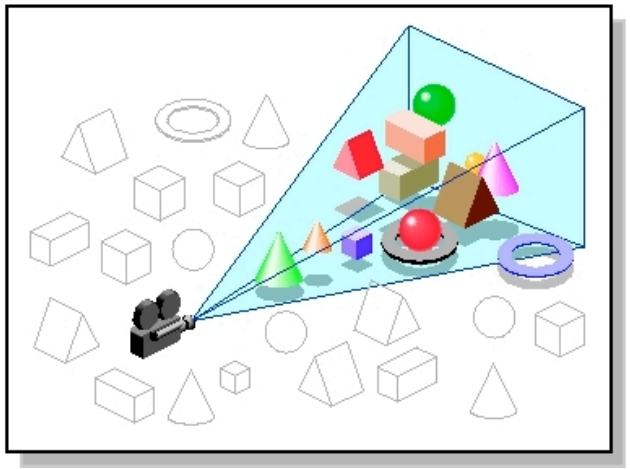
- **Données lues par le programme (mémoire système)**
  - Vertex info: position, normale, coordonnées de textures
  - Textures
- **Mémoire GPU**
  - != mémoire système
  - Limite de taille et de bande passante CPU->GPU
- **Transmettre uniquement les objets utiles**
  - « liste graphique » pour un trame donnée
  - Libérer les ressources matérielles si non utilisées
    - Ex: changement de niveau, changement d'accessoire d'un personnage, etc …

Gestion des données

Elimination d'objets

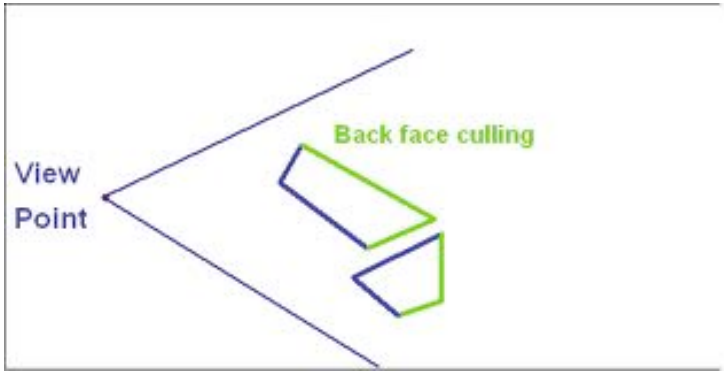
- « Culling »
- Si objet hors de la zone de dessin, pas de calcul
  - Notion de Frustum (volume de visualisation
- Pas effectué par le matériel

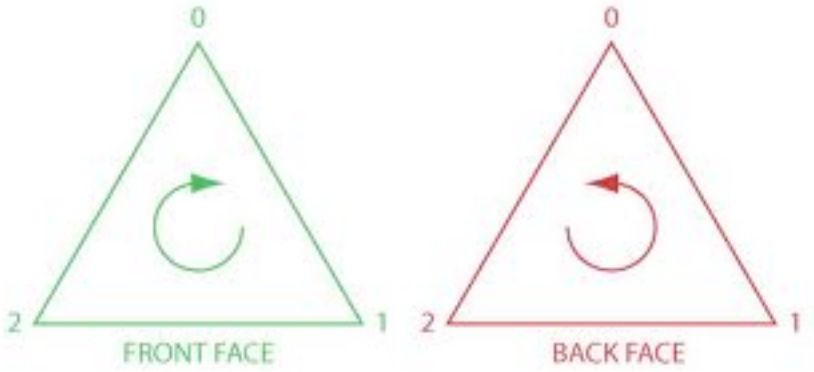


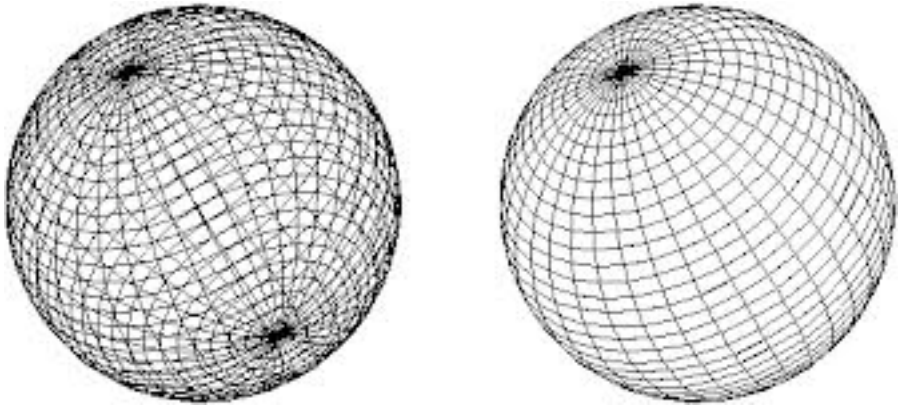


Gestion des données

- **Éliminer les faces non visibles**
  - Faces arrières des objets
    - Masquées par une autre face
    - Objets fermés et opaques
  - Peut être effectué par le matériel
    - « back-face culling »
  - Test Vue.Normal
    - !! FACES ORIENTEES







Gestion des données: animations

- **Plusieurs trames de rendu pour un même objet**
  - Envoyer N fois les mêmes objets au GPU n'est pas efficace !
- **DisplayList**
  - Enregistre une série de commandes envoyées au GPU
  - « Relecture » de cette série
    - Plus efficace si la série est stockée en mémoire GPU
  - Non paramétrable
    - Pas possible de ré invoquer en ne changeant qu'un paramètre (ex couleur du vertex K)
- **VertexBufferObject**
  - Attributs de vertex stockés sur le GPU
  - Ré-utilisation du VBO ne nécessite pas de retransmettre les données
  - Utilisé pour position, normale, coordonnées de textures, couleur