Cours 1 - Premiers pas en 3D

Xavier Décoret - INF583 – École Polytechnique

Disclaimer

- Attention, c'est le cours
  - le plus difficile
  - le plus important
- Il y a beaucoup de "on verra plus tard"
- C'est le "canvas" sur lequel ancrer les connaissances

Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

Overview

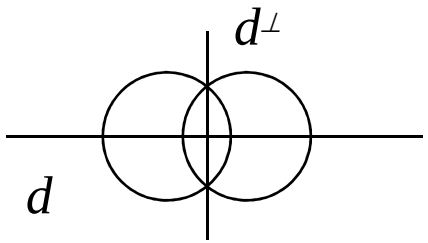
- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

Représentation du monde

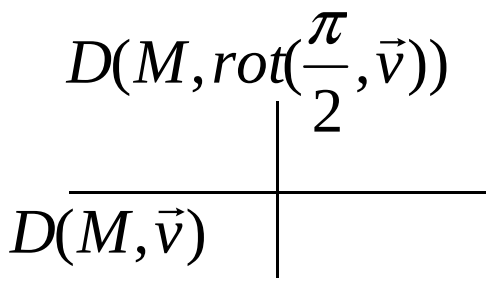
- Notion de modèle
  - représenter un objet à manipuler
    - modèles réduits, maquettes
    - visualisation et édition
  - abstraire la réalité
    - simplifier/idéaliser le problème
    - réduire/identifier les paramètres
- Types de modèles?
  - modèles mathématiques
    - classes d'objets représentables
  - modèles numériques
    - valeurs décrivant un objet particulier

Spécificité de l'informatique

- Le modèle doit être constructif
  - ex: la suite des nombres premiers
    - en math
      - définition par des propriétés
      - manipulation par des théorèmes
    - en info
      - on veut leurs valeurs!
  - ex: droite perpendiculaire à une autre
    - en math
      - règle et compas
    - en info
      - coordonnées 2D



modèle du compas et de la règle



Application au graphisme (1/2)

- Que faut-il modéliser?
  - la forme des objets
  - l'apparence des objets
    - matériaux
    - ombres
    - plus généralement, les interaction lumineuses
  - la "physique"
    - mouvements
    - collisions
    - déformations

Application au graphisme (2/2)

- Quelles sont les contraintes?
  - modèle **puissant** permettant:
    - de représenter une large classe d'objets
    - de créer des objets complexes à partir d'objets simples
  - modèle **intuitif** permettant
    - d'éditer le modèle
    - d'animer des parties du modèle
  - modèle **efficace**
    - peu coûteux en mémoire
    - rapide à afficher / à traiter

Conclusion

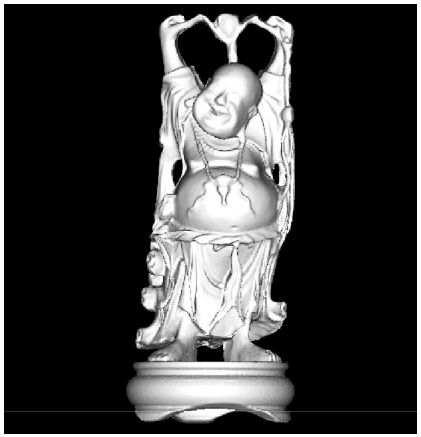
- Il n'y a pas un mais des modèles!
- On verra principalement les maillages
  - PGCD des différents modèles
  - Adapté au cartes accélératrices 3D
  - Plutôt intuitif
  - Massivement utilisé
- On en verra d'autres plus tard
  - *Modélisation à base de point*
  - *Modélisation à base d'image*
  - *Modélisation procédurale*
  - *Surfaces paramétrées*

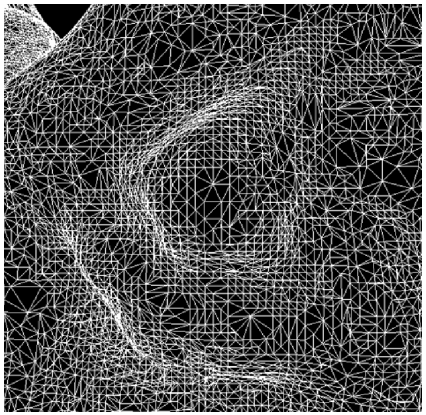
Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

Maillages

- Ensemble de sommets et de faces
  - sommet (*vertex/vertices*)
    - ensemble d'attributs dont une position 3D
  - faces (*face, facets*)
    - relie les points
    - a deux "côtés"
    - l'orientation compte
    - doit être convexe
    - peut se ramener aux triangles

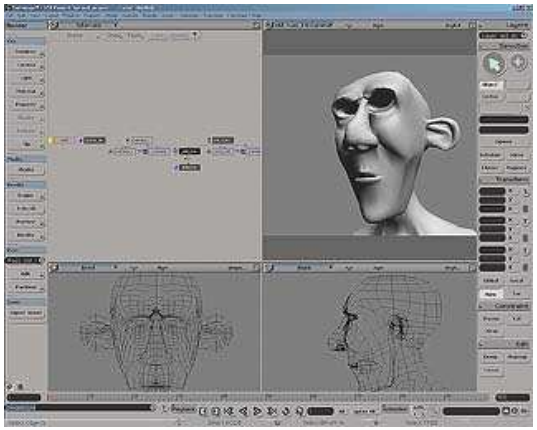




Authoring

- Avec un système de scan 3D

- Avec un outil de DCC (*Digital Content Creation*)



Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

Coordonnées homogènes

- Espace projectif IP^3
  - Espace quotient sur IR^4 pour la relation d'équivalence

$$(x,y,z,w) \equiv (x',y',z',w') \Leftrightarrow \begin{cases} \left(\frac{x}{w},\frac{y}{w},\frac{z}{w}\right) = \left(\frac{x'}{w'},\frac{y'}{w'},\frac{z'}{w'}\right) \text{si } w \neq 0 \\ (x,y,z) = (x',y',z') \text{si } w = 0 \end{cases}$$

  - Contient IR^3 plus les points à l'infini

- À quoi ça sert?
  - Unifier les calculs géometriques
  - Permettre des interpolations

} Cf. plus loin dans le cours

Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

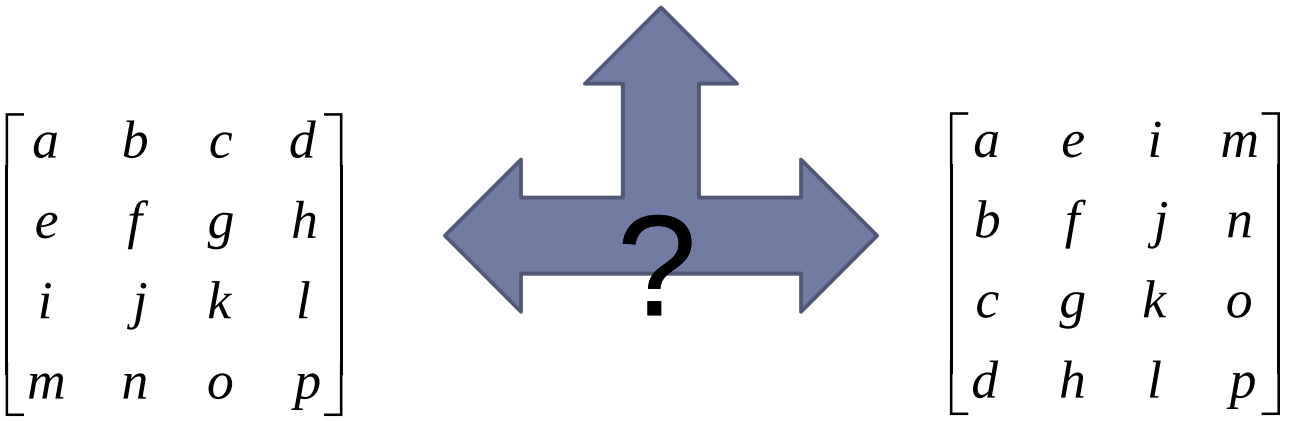
Transformation

- Pourquoi?
  - Utilisation de repères locaux
  - Instanciation
- Lesquelles?
  - Rigide: translation & rotation
  - Autres: mise à l'échelle, *shear*
  - Et plus dans un instant…
- Quelle représentation?
  - Matrice 4x4

Représentation par matrice 4x4

- Attention à "l'orientation"
  - *Row order* vs. *column order*

```
float M[16] = { a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p };
float M[4][4] = { {a,b,c,d },
                  {e,f,g,h },
                  {i,j,k,l },
                  {m,n,o,p } };
```



Row order

Column order

Représentation par matrice 4x4

- Attention à "l'orientation"
  - *Row order* vs. *column order*
  - Anglo saxons et Français

t une transformation de matrice

$$\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

v un point de coordonnées

(x, y, z, w)

coordonnées de $t(v)$?

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \quad ? \quad [x \;\; y \;\; z \;\; w] \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

Représentation par matrice 4x4

- Attention à "l'orientation"
  - Voici la convention/cohérence choisie

```
float M[16] = { a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p };
float M[4][4] = { {a,b,c,d },
                  {e,f,g,h },
                  {i,j,k,l },
                  {m,n,o,p } };
```

$$M = \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

coordonnées de $t(v)$

$$\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

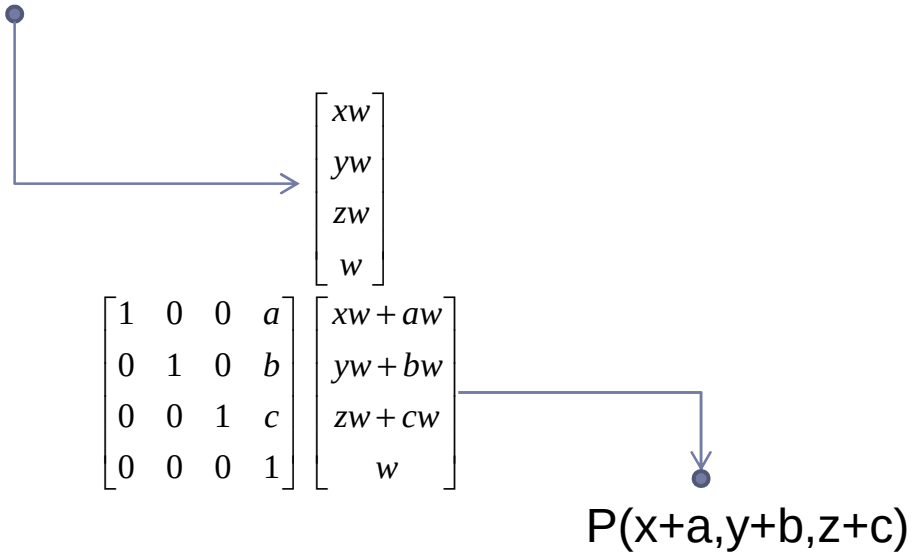
Représentation par matrice 4x4

- Permet les transformations précédentes

Translation par (a,b,c)

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

P(x,y,z)




```
float M[4][4] = { {1,0,0,0},
                  {0,1,0,0},
                  {0,0,1,0},
                  {a,b,c,1}
                };
float P[4] = {xw,yw,zw,w};
```

Représentation par matrice 4x4

- Permet les transformations précédentes

Translation par (a,b,c)

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

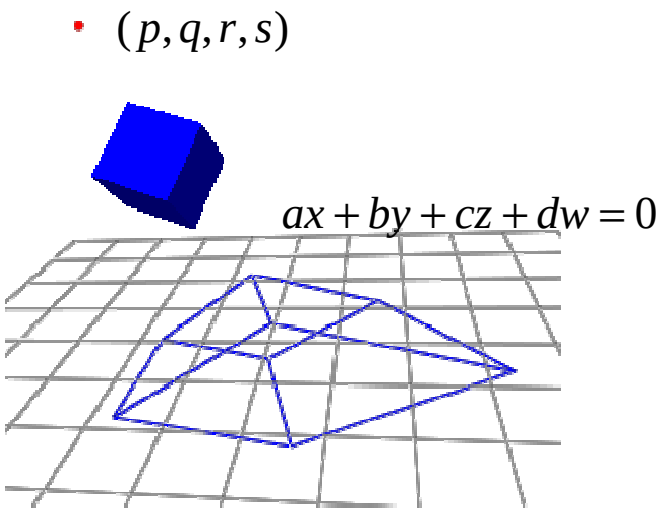
Rotation de α autour de (x,y,z)

$$c = \cos(\alpha)$$
$$s = \sin(\alpha)$$

$$\begin{bmatrix} x^2(1-c)+c & xy(1-c)+zs & xz(1-c)+ys & 0 \\ yx(1-c)-zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ zx(1-c)-ys & zy(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Représentation par matrice 4x4

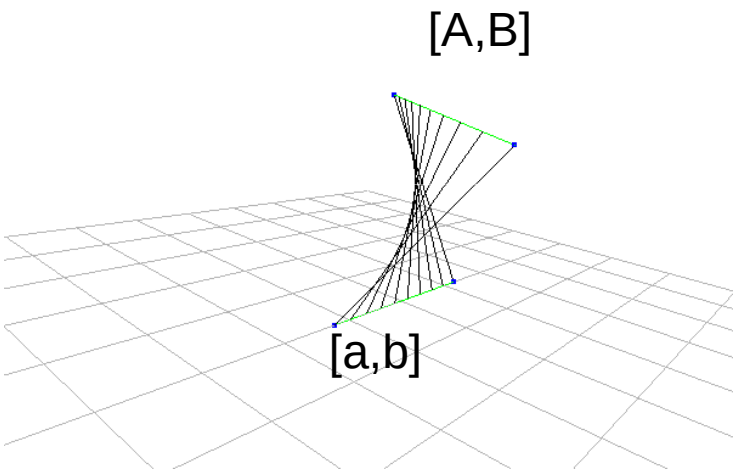
- Permet plus encore
  - Projection sur un plan depuis un point (ou une direction)



$$M = [abcd]^T \times [pqrs] - [abcd] \times [pqrs]^T I$$

Représentation par matrice 4x4

- Permet plus encore
  - Transformation d'un segment en un autre



$$M = u^T ab + ab^T \times \left(a\text{-}(A.u)ab\right)$$

$$u = \frac{AB}{\left|AB\right|^2}$$

Les vecteurs u,ab,a et A sont des coordonnées homogènes avec w=0 pour u et ab, et avec w=1 pour a et A

Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation

- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

Obtention d'une image

Ray-tracing

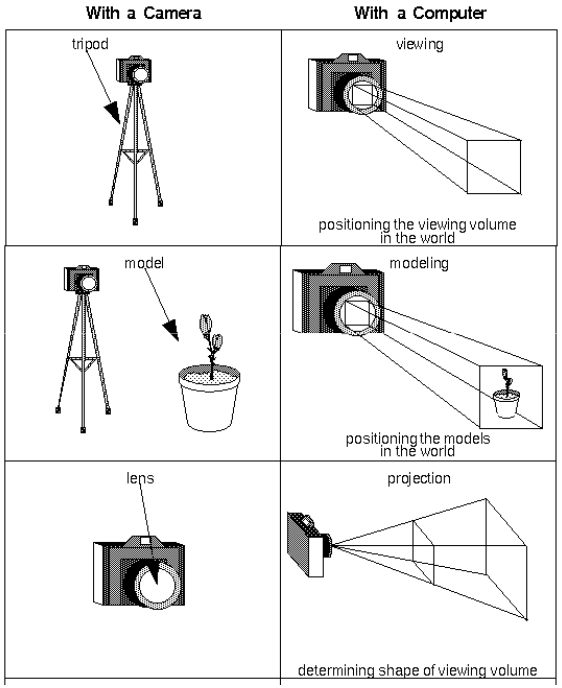
- On part de la caméra et on regarde quelles parties de la scène elle voit.

Rasterisation

- On part de la scène et on la projette sur l'écran de la caméra

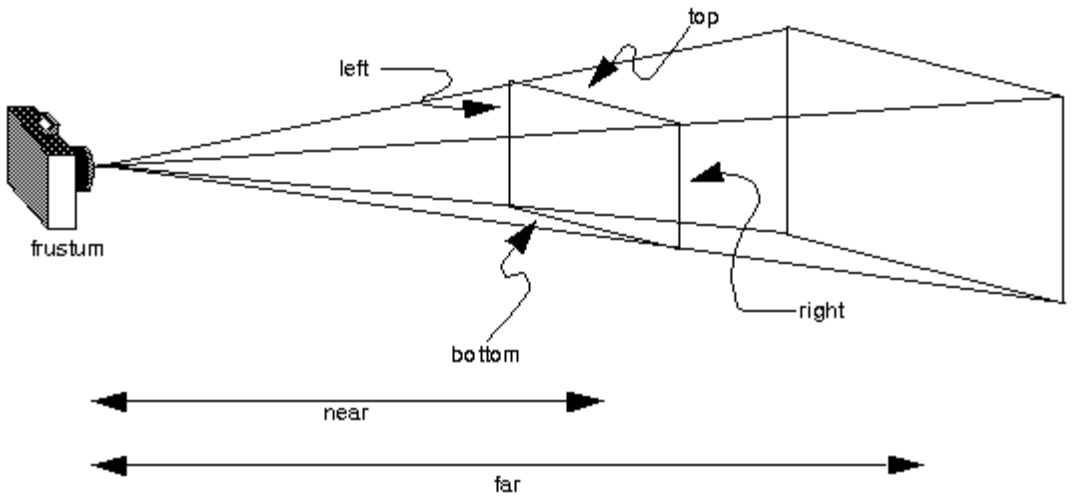
Caméra *pinhole*

- Paramètres intrinsèques et extrinsèques
  - Position/orientation
  - Pyramide de vue (*view frustum*)



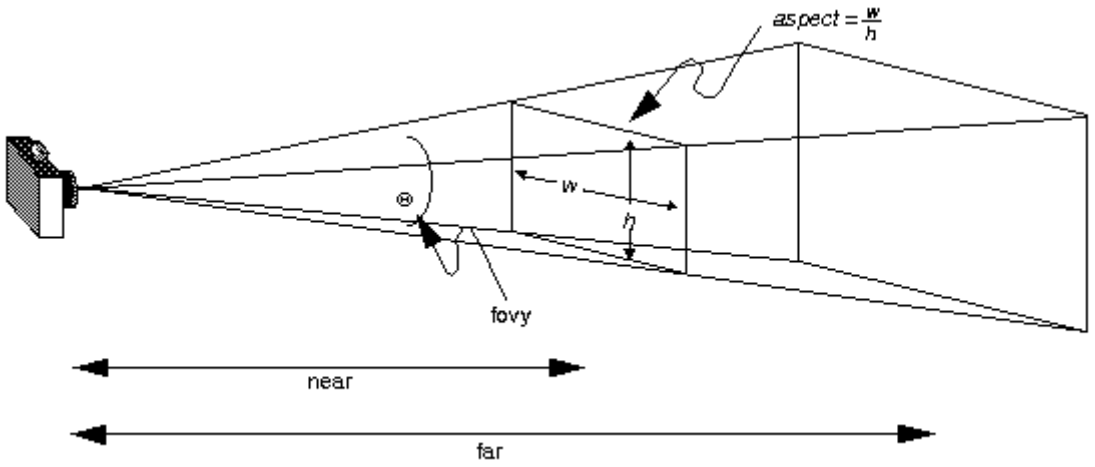
Caméra *pinhole*

- Paramètres intrinsèques et extrinsèques
  - Position/orientation
  - Pyramide de vue (*view frustum*)



Caméra *pinhole*

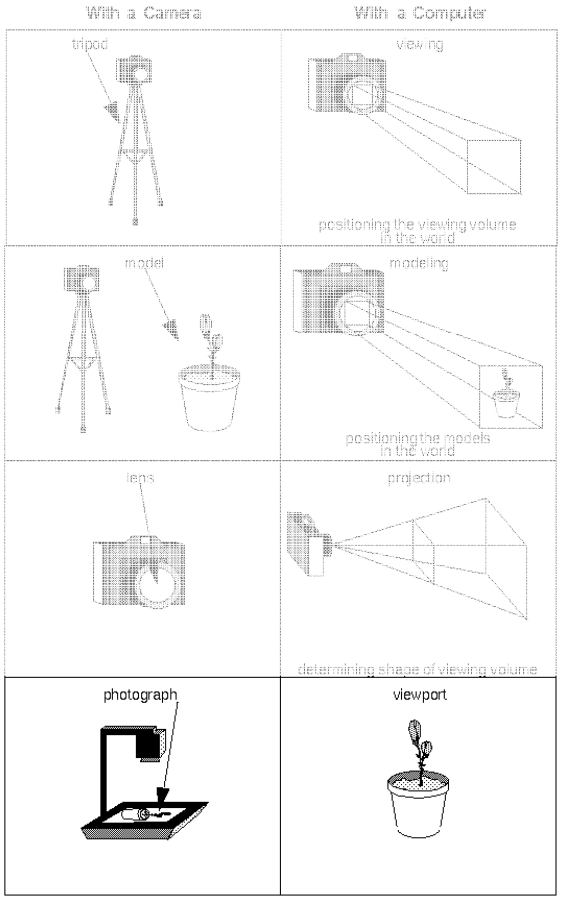
- Paramètres intrinsèques et extrinsèques
  - Position/orientation
  - Pyramide de vue (*view frustum*)



Caméra *pinhole*

- Modélisé par deux matrices 4x4
  - *View* : place la caméra dans le monde ou, de manière équivalente, transforme le monde dans le repère propre de la caméra.
  - *Projection* : projette les points sur le plan image

- …et un *viewport*
  - Indique comment "mapper" le plan image de la caméra sur tableau de pixels



Caméra *pinhole*

- Projection orthogonale et perspective

Typiquement utile pour la conception (vue de dessus, de côté), mais aussi pour modéliser des sources lumineuses directionnelles comme le soleil.

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

avec $f = \text{cotan}(\frac{\theta}{2})$

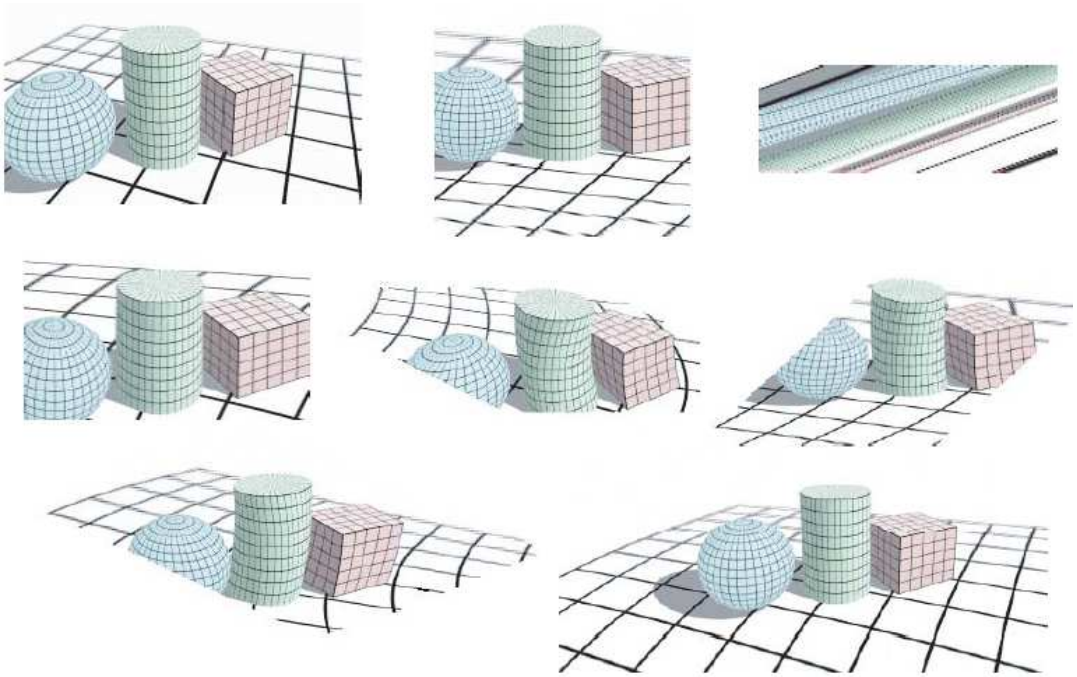
Caméra pinhole

- C'est un modèle simple et pratique
- Ne simule pas les phénomènes optiques dus aux lentilles
  - pas de profondeur de champ
  - pas d'aberrations
- On peut les rajouter par des "post-traitements" image

http://www.linternaute.com/photo_numerique/dossier/aberrations/index.shtml

Caméra *pinhole*

- Appartient à une classe de caméra plus large



Yu, J. and L. McMillan. ECCV 2004
General Linear Cameras.

Yu, J. and L. McMillan. EGSR 2004
A Framework for Multiperspective Rendering.

Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

Ray-tracing (1/3)

- Pour chaque point P de l'écran
  - tracer un rayon joignant l'oeil et P
  - intersecter ce rayon avec les triangles de la scène
  - calculer la couleur du point d'intersection I
    - lancer éventuellement des rayons secondaires depuis I
    - on y reviendra

Ray-tracing (2/3)

- Calcul de l'intersection
  - de nombreuses méthodes
  - faisable en parallèle
  - utilisation de structures d'accélération

Fast, minimum storage ray/triangle intersection
Möller T., Trumblore B. Siggraph 2005 course

- Généralisable à d'autres primitives
  - Quadriques (sphères, ellipsoides, cylindres, cônes)
  - Grille régulière ou hiérarchiques

Ray-tracing (3/3)

- Proche du phénomène "physique"
- Nombreuses variantes & applications
  - *Ray-tracing*
  - *Path tracing*
  - *Photon mapping*
- Belles images mais coûteux

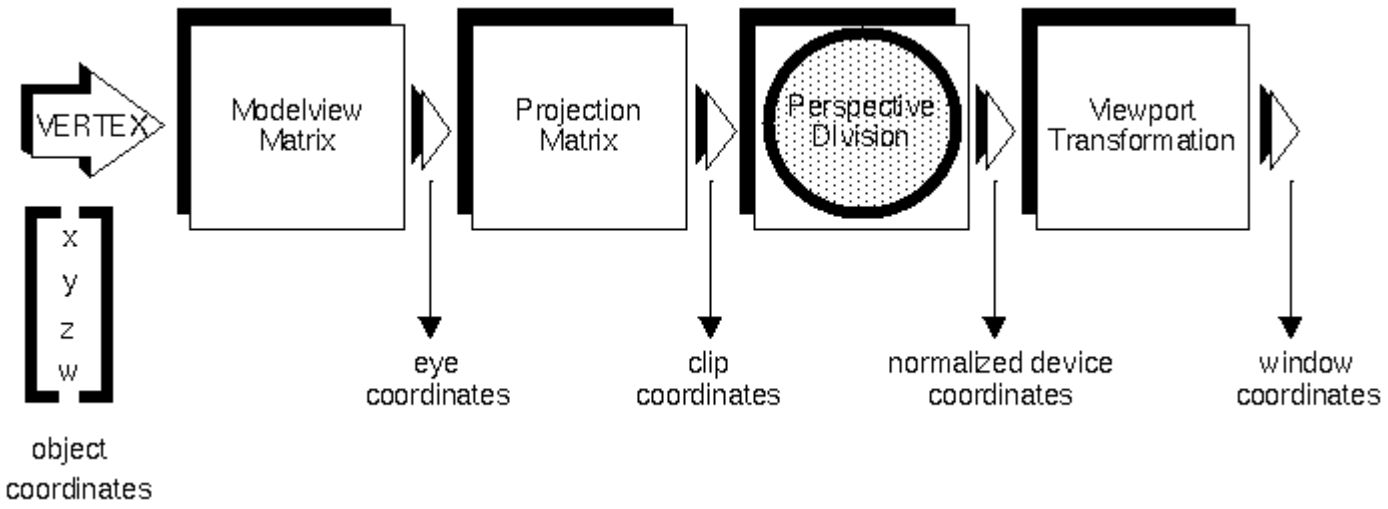
Rasterisation (1/3)

- Pour chaque triangle
  - on projette les sommets sur l'écran
  - on rempli l'intérieur du projeté
  - en chaque point, on garde la projection la plus proche

Rasterisation (2/3)

- Projection sur l'écran
  - Position du sommet dans le monde
  - Position dans le repère caméra
  - Projection sur l'écran
  - Conversion en pixels

duales , modélisé par une seule matrice : modelview



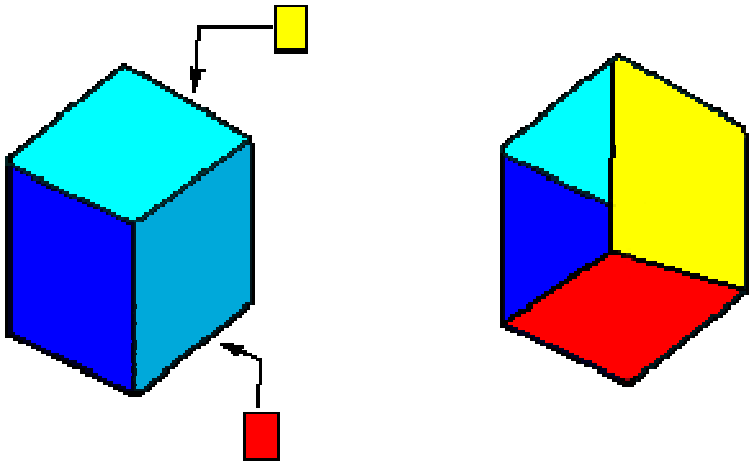
Rasterisation (3/3)

- Remplissage du projeté
  - Interpolation des attributs
  - Production d'une couleur (cf. plus tard)
  - Interpolation perspective

Rasterisation (3/3)

- Problème: élimination des faces cachées

http://www.cosc.brocku.ca/Offerings/3P98/course/lectures/hiddenlines/



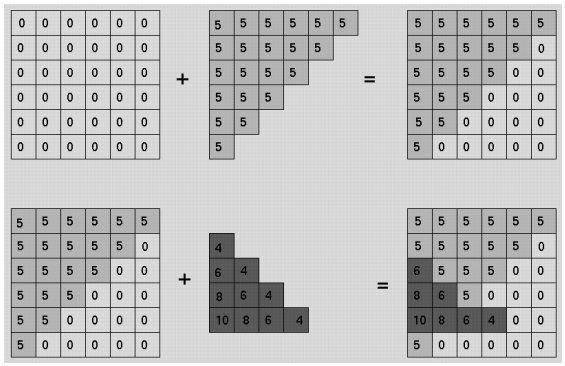
Drawing faces in different orders

Rasterisation (3/3)

- Problème: élimination des faces cachées

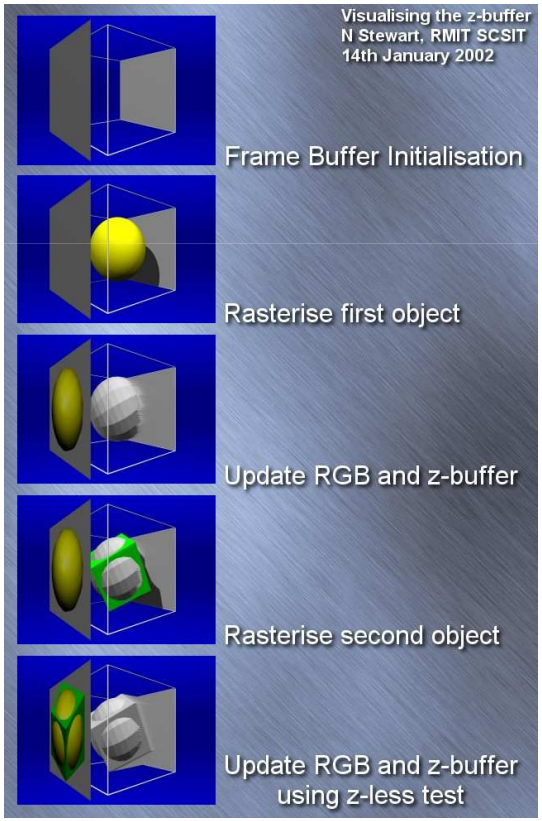
  http://www.cosc.brocku.ca/Offerings/3P98/course/lectures/hiddenlines/

- Solution: Z-Buffer et Z-test
  - Principe:
    - stocker une couleur plus une distance à l'oeil
    - comparer celles des fragments produits
    - garder le plus proche



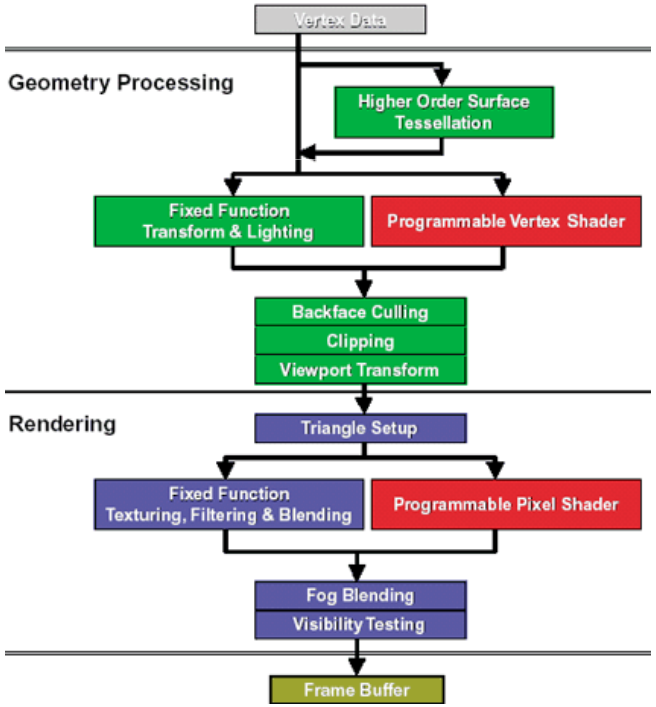
Principe du z-test

Visualisation d'un z-buffer



Conclusion

- On ne parlera que de rasterisation
- C'est très adapté aux cartes graphiques (aka GPU)
  - Pourquoi?
  - Comment marche une carte?
    - Transform & Lighting stage
    - Rasterisation
    - Framebuffer
    - Deph/Alpha/Scissor/Stencil Tests



Overview

- Représentation du monde
  - maillages
  - coordonnées homogènes
  - matrices de transformation
- Obtention d'une image
  - modèle de caméra
  - ray-tracing
  - rasterisation
  - découverte de l'API

API graphiques

- Permet d'envoyer des ordres à la carte
- Très proche du hard
- Deux grandes: DirectX OpenGL
- Un exemple vaut mieux qu'un long discours
  - spécifier un maillage en *direct mode*
  - instancier un objet
  - placer la caméra
- La suite en TD…

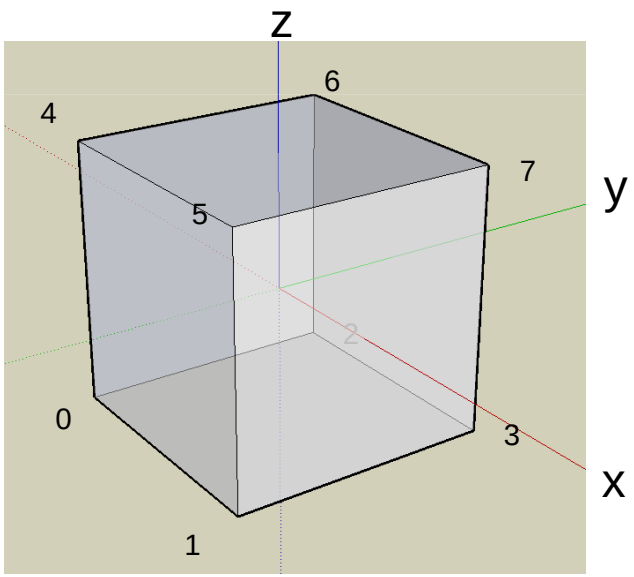
Retour sur les maillages

- Considérations pratiques
  - Stockage
  - Performance
    - Coût des traitements
    - Coûts des transferts CPU/GPU
- Considérations théoriques
  - Information non structurée
  - Forme mais pas topologie

- On peut faire mieux que la soupe de triangles…

Indexed Face Set (1/2)

- Principe
  - Un tableau décrit les sommets (4 floats)
  - Un tableau décrit les triangles reliant ces sommets (integers)



Vertices

#	X	Y	Z
0	-0.5	-0.5	-0.5
1	+0.5	-0.5	-0.5
2	-0.5	+0.5	-0.5
3	+0.5	+0.5	-0.5
4	-0.5	-0.5	+0.5
5	+0.5	-0.5	+0.5
6	-0.5	+0.5	+0.5
7	+0.5	+0.5	+0.5

Facets

v0	v1	v2	v3
0	2	6	4
1	3	7	5
0	1	5	4
2	3	7	6
0	1	3	2
4	5	7	6

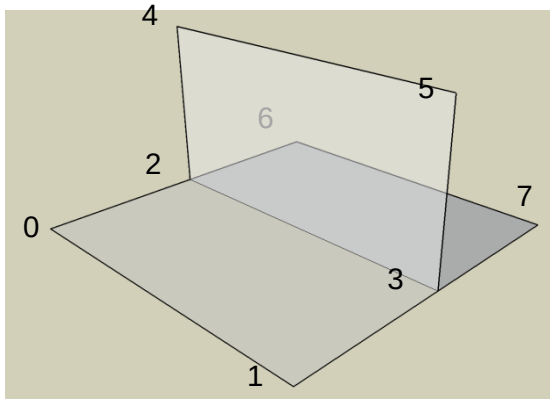
Indexed Face Set (2/2)

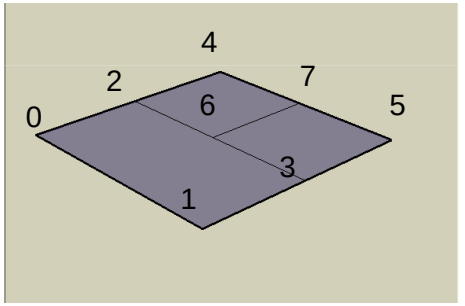
- Avantages
  - Diminue les coûts
    - de stockage/transfert
    - lors du traitement
  - Encode forme et topologie
- Inconvénients
  - Pas de contraintes topologiques
    - *non manifold*
    - *T-vertices*

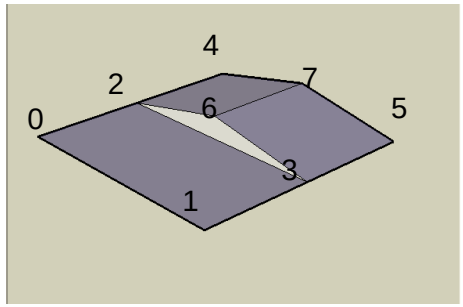
6x4x(3 `float`)= 288 bytes soupe de triangles

8x(3 `float`)+6x4 `short int`= 120 bytes *indexed face set*

8 sommets à “projeter” contre 24







C’est fini pour aujourd’hui

C’est l’heure de la pause