Algorithmique graphique

M. AMMI
ammi@limsi.fr

Historique

- 1944 : Le premier ordinateur électrique et programmable est construit aux États-Unis
- 1950 : MIT, premier écran (vectoriel) contrôlé par ordinateur
- 1962 : P. Bezier (Renault) met au point une méthode pour tracer des courbes ou des surfaces
- 1963 : Sketchpad (thèse de Ivan Sutherland), propose un premier modèle complet de système graphique interactif (sélectionner, pointer, dessiner, éditer); identifie les structures de données et algorithmes nécessaires. Avancée majeure dans le domaine du graphisme
- 1965 : J. Bresenham met au point un algorithme efficace pour le tracé de lignes
- 1969 : J. Warnock propose un algorithme de subdivision pour la suppression des faces cachées
- 1971 : H. Gouraud fait une thèse avec Evans à l'Université d'Utah où il met au point un algorithme de lissage d'ombres
- 1971 : R. Goldstein et R. Nagel posent les bases de la CSG (Constructive Solid Geometry)
- 1972 : R. Shoup (Xerox) met au point le premier frame-buffer 8 bits
- 1974 : E. Catmull (Utah) soutient sa thèse sur le placage de texture, le Z-buffer et le rendu de surfaces courbes
- 1974 : B. Phong (Utah) invente un nouvel algorithme de lissage d'ombres
- 1974 : Sutherland et Hodgman développent un algorithme de clipping de polygones
- 1977 : J. Bresenham met au point un algorithme efficace pour le tracé de cercles
- 1978 : Cyrus et Beck proposent un algorithme de clipping de segments



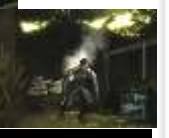
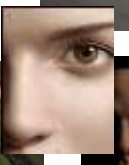
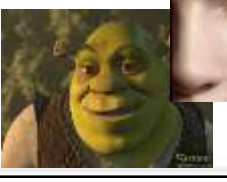

Historique

- 1979 : James H. Clark conçoit le premier processeur programmable dédié au graphisme 3D
- 1982 : création de Silicon Graphics, d'Adobe et d'AutoDesk
- 1984 : premiers travaux sur la radiosité à Cornell University
- 1985 : création de ATI : conception de circuits intégrés graphiques
  - 1987 : création de la 1ère carte graphique
- 1987 : Marching Cubes : A High Resolution 3D Surface Construction Algorithm, par Lorensen et Cline (GE)
- 1992 : OpenGL 1.0
- 1993 : création de NVIDIA
- 1993 : Jurassic park
- 1994 : standard VRML
- 1996 : Microsoft lance DirectX
- 1997 : Sun lance Java 3D
- 1998 : logiciel Alias Maya
- 2007 : Spécifications d'OpenGL 3.0

Pour plus de détails : http://accad.osu.edu/~waynec/history/timeline.html


Domaines d'application

- Art & divertissement
  - Films d'animation, Effets spéciaux
  - Jeux (Temps réel, Interaction)


Domaines d'application

- Suivi de processus et Téléopération (online)




Domaines d'application

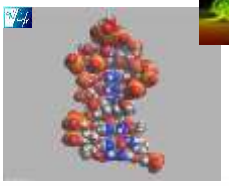
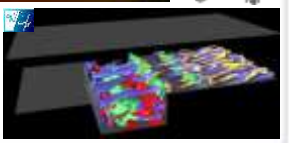
- Simulateurs (offline)
  - Conduite/Pilotage, Processus, Téléopération, Jeux!




Domaines d'application

- Visualisation de données
  - Médicales
  - Géophysiques
  - Biologiques



Domaines d'application

- Conception assistée par ordinateur
  - Immersive ou pas


Définitions

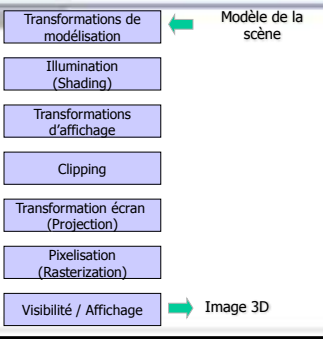
- Définition officielle de l'infographie (office de la langue française
  - « Application de l'informatique à la création, au traitement, et à l'exploitation des images numériques »
- Infographie est un mot formé à partir de d'INFOrmatique et GRAPHIque
  - Appellation déposée par la société Benson en 1974

Définitions

- La géométrie est l'élément constitutif essentiel de toute image de synthèse
- Les objets présents dans une image de synthèse sont toujours définis à partir d'objets géométriques dont l'expression mathématique est connue.
- Ces objets ainsi que l'organisation spatiale de ces objets les uns par rapport aux autres forment un modèle géométrique complet de la scène virtuelle à représenter et constituent un pré-requis essentiel.
- L'image de synthèse obtenue est le résultat de l'interaction entre les différentes sources de lumière de la scène et les objet. Cette interaction s'évalue à partir des caractéristiques physiques des objets et des sources de lumières, et des propriétés géométriques locales en tout point de la scène (géométrie locale) et globales (chaque objet interagit avec une partie ou la totalité de la scène).

Pipeline Graphique

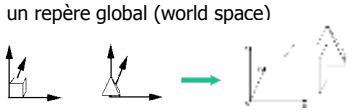
Pipeline Graphique



Transformations de modélisation

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

• Application des transformations de composition de scène : Passage du système de coordonnées local de chaque objet 3D (object space) vers un repère global (world space)



Object space World space

Illumination

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

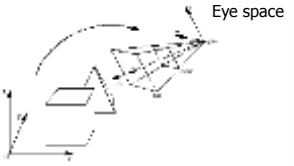
• Les primitives sont éclairées selon leur matériau, le type de surface et les sources de lumière.

• Les modèles d'illumination sont locaux (pas d'ombres) car le calcul est effectué par primitive : diffus, ambiant, Gouraud, Phong, etc.

Transformations d'affichage

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

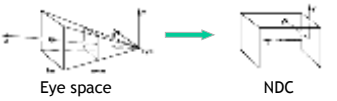
• Passe des coordonnées du monde à celles du point de vue (repère caméra ou eye space).
• En général le repère est aligné selon z.



Clipping

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

• Passage en coordonnées normalisées (NDC : normalized device coordinates)

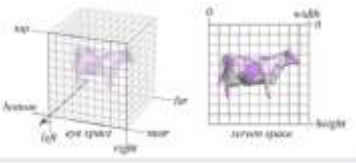


• Suppression des parties hors du volume de vision.

Transformation écran

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

• Les primitives 3D sont projetées sur l'espace image 2D (screen space)

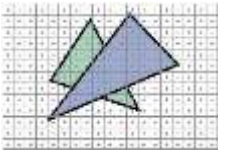


Rasterisation

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

• Découpe des primitives 2D en pixels

• Interpole les valeurs connues aux sommets : couleur, profondeur, etc. pour chaque fragment affiché

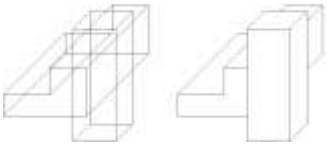


Visibilité et affichage

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

•Élimination des parties cachées

•Remplissage du frame buffer avec le bon format de couleur.



Système de coordonnées

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

Repère objet

Repère scène

Repère caméra

Repère caméra normalisé (NDC)

Espace écran

Le pipeline graphique

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

•Modèle géométrique : objets, surfaces, sources de lumière…

•Modèle d'illumination : calcul des interactions lumineuses

•Caméra : point de vue et ouverture (frustum)

•Fenêtre (viewport) : grille de pixel sur laquelle on plaque l'image

•Couleurs, intensités convenant à l'afficheur (ex : 24 bits, RVB)

Le pipeline graphique

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

•Chaque primitive passe successivement par toutes les étapes

•Le pipeline peut être implémenté de diverses manières avec des étapes en hardware et d'autres en software

•A certaines étapes on peut disposer d'outils de programmation (ex : vertex ou pixel program)

Implémentation

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

Software configurable

Sans carte graphique 3D

Implémentation

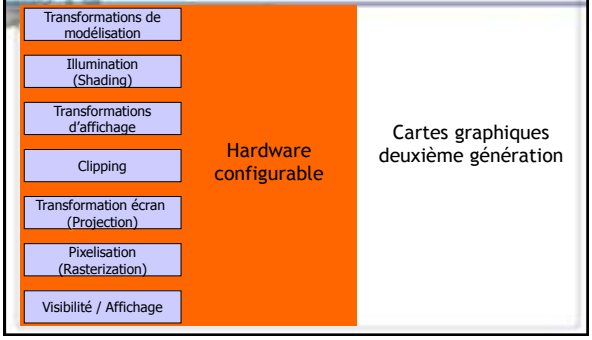
- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

Software configurable

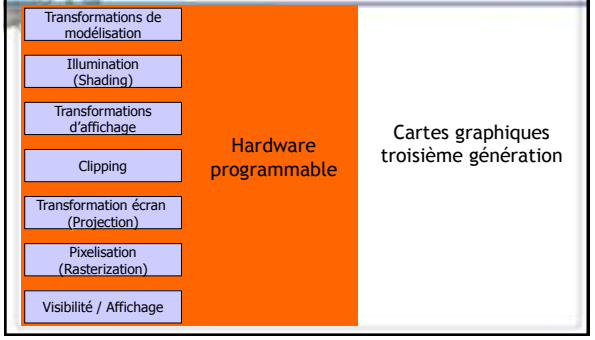
Hardware

Cartes graphiques première génération

Implémentation



Implémentation



Rappels mathématiques

Scalaire

- Un scalaire est une grandeur totalement définie par un nombre et une unité.
- Il a une valeur numérique mais pas d'orientation.
- Ex :
  - Masse
  - Distance
  - Température
  - Volume
  - Densité
  - Etc.
- Les scalaires obéissent aux lois de l'algèbre ordinaire

Scalaire

- Opérations élémentaires:
  - Addition & multiplication
  - Propriétés
    - Commutativité $\alpha+\beta=\beta+\alpha$ $\alpha\cdot\beta=\beta\cdot\alpha$
    - Associativité $\alpha+(\beta+\gamma)=(\alpha+\beta)+\gamma$ $\alpha\cdot(\beta\cdot\gamma)=(\alpha\cdot\beta)\cdot\gamma$
    - Distributivité $\alpha\cdot(\beta+\gamma)=(\alpha\cdot\beta)+(\alpha\cdot\gamma)$
- Identité
  - Addition : 0 $\alpha+0=0+\alpha=\alpha$
  - Multiplication : 1 $\alpha\cdot1=1\cdot\alpha=\alpha$

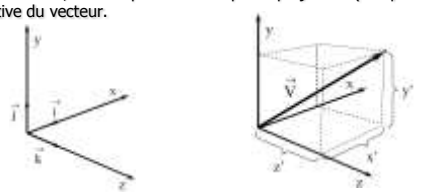
Vecteur

- Un vecteur est une entité mathématique définie par n valeurs numériques extraites du même ensemble E (par exemple N, Z, R, C,…)
- Ces valeurs numériques décrivent le module et l'orientation du vecteur
- n est appelé la dimension du vecteur.
  - On dit que le vecteur est défini dans En.
  - En est un espace de dimension n.
  - Exemple : dans Z2 , dans R3.
- Ex :
  - Déplacement
  - Vitesse
  - Accélération
  - Force

$$\begin{pmatrix} x \\ y \end{pmatrix}_{Z2} \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{R3}$$

Vecteurs unitaires

- Dans un repère les vecteurs sont décrit dans une base (unités de l'ensemble)
  - généralement unitaire
- Le vecteur est représenté par l'addition des vecteurs unitaires à chaque axe de coordonnées, en multipliant chacun par la projection (composante) respective du vecteur.



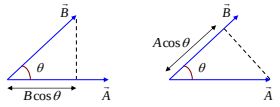
$$\vec{V} = x'\vec{i} + y'\vec{j} + z'\vec{k}$$

Vecteur

- Les vecteurs obéissent aux lois de l'algèbre vectorielle
- Opérations élémentaires
  - Produit scalaire
  - Produit vectoriel
  - Addition de vecteurs
  - Produit vecteur-scalaire
  - Normalisation

Produit scalaire

- Le produit scalaire de deux vecteurs est le produit du module du premier par la composante du second dans la direction du premier



- Produit scalaire en fonction
  - du module et de l'angle : $\vec{A}.\vec{B} = |A||B|\cos\theta$
  - des composantes : $\vec{A}.\vec{B} = A_x B_x + A_y B_y + A_z B_z$

Produit scalaire

- Propriétés :
  - Commutativité $u.v = v.u$
  - Distributivité par l'addition $(u+v).w = u.w + v.w$
  - Distributivité par un scalaire $u.(k \times v) = k \times (u.v)$

Produit scalaire

- Angle entre deux vecteurs

$$\vec{A}.\vec{B} = \|\vec{A}\|\,\|\vec{B}\|\cos\theta \quad \Rightarrow \quad \cos\theta = \frac{A_x B_x + A_y B_y + A_z B_z}{\|\vec{A}\|\,\|\vec{B}\|}$$

$$\vec{A}.\vec{B} = A_x B_x + A_y B_y + A_z B_z$$

- Signe du produit scalaire

$\vec{A}.\vec{B} > 0 \Rightarrow -90° < \theta < 90°$

$\vec{A}.\vec{B} = 0 \Rightarrow \theta = \pm 90°$

$\vec{A}.\vec{B} < 0 \Rightarrow -180° < \theta < -90°$ *ou* $90° < \theta < 180°$

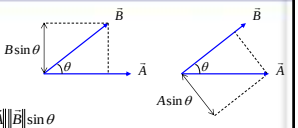
Produit scalaire

- Application
  - Projection d'un vecteur sur un autre
    - Élimination des faces cachées
  - Calcul d'angle entre deux vecteurs
    - Calcul de la quantité de lumière perçue par une face
      - Ombrage
  - Etc.

Produit vectoriel

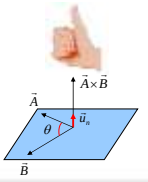
• Le module du produit vectoriel de deux vecteurs est le produit du module du premier par la composante du second qui est perpendiculaire au premier.

$$\left\|\vec{A}\times\vec{B}\right\| = \left\|\vec{A}\right\|\left\|\vec{B}\right\|\sin\theta) = \left\|\vec{B}\right\|\left\|\vec{A}\right\|\sin\theta) = \left\|\vec{A}\right\|\left\|\vec{B}\right\|\sin\theta$$



• Le produit vectoriel est un vecteur perpendiculaire à *A* et à *B* dont le sens est donné par la règle de la main droite.

$$\vec{A}\times\vec{B} = (\left\|\vec{A}\right\|\left\|\vec{B}\right\|\sin\theta)\vec{u}_n$$



• Le produit vectoriel est nul si les deux vecteur sont parallèles et maximal s'ils sont perpendiculaires.

Produit vectoriel

Produit vectoriel en fonction des composantes:

$$\vec{A}\times\vec{B} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} = +\vec{i}\begin{vmatrix} A_y & A_z \\ B_y & B_z \end{vmatrix} - \vec{j}\begin{vmatrix} A_x & A_z \\ B_x & B_z \end{vmatrix} + \vec{k}\begin{vmatrix} A_x & A_y \\ B_x & B_y \end{vmatrix}$$

$$\vec{A}\times\vec{B} = \vec{i}\,(A_yB_z - B_yA_z) - \vec{j}\,(A_xB_z - B_xA_z) + \vec{k}\,(A_xB_y - B_xA_y)$$

Exemple:

$$\vec{A}\times\vec{B} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 2 & -4 \\ 3 & -1 & 5 \end{vmatrix} = +\vec{i}\begin{vmatrix} 2 & -4 \\ -1 & 5 \end{vmatrix} - \vec{j}\begin{vmatrix} 1 & -4 \\ 3 & 5 \end{vmatrix} + \vec{k}\begin{vmatrix} 1 & 2 \\ 3 & -1 \end{vmatrix}$$

$$\vec{A}\times\vec{B} = \vec{i}\,(2\times5-(-1)\times(-4)) - \vec{j}\,(1\times5-3\times(-4)) + \vec{k}\,(1\times(-1)-3\times2)$$

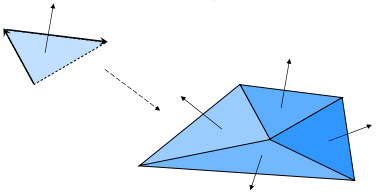
$$\vec{A}\times\vec{B} = 6\vec{i} - 17\vec{j} - 7\vec{k}$$

Produit vectoriel

- Propriétés
  - Anticommutativité $(u\times v) = -(v\times u)$
  - Distributivité sur l'addition $(u+v)\times w = u\times w + v\times w$
  - Distributivité par un scalaire $(u+v).k = u.k + v.k$
  - Non-associativité $(u\times v)\times w \neq u\times(v\times w)$

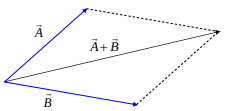
Produit vectoriel

- Application
  - Calcul de la normale à un plan



Addition de deux vecteurs

- L'addition de deux vecteurs obéit à la règle du parallélogramme.



- Algébriquement cela se met en oeuvre en additionnant les composantes individuellement.

$$\vec{A} = a_x\vec{i} + a_y\vec{j} + a_z\vec{k}$$
$$\vec{B} = b_x\vec{i} + b_y\vec{j} + b_z\vec{k}$$

$$\vec{A}+\vec{B} = (a_x+b_x)\vec{i} + (a_y+b_y)\vec{j} + (a_z+b_z)\vec{k}$$

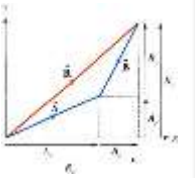
Addition de deux vecteurs

Un vecteur A peut être décomposé en ses composantes rectangulaires A_x et A_y.

$$A_x = A\cos\theta_A$$
$$A_y = A\sin\theta_A$$

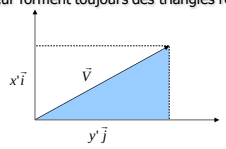
Il est possible d'additionner des vecteurs en additionnant les composantes de ces vecteur.

$$A_x = A\cos\theta_A \qquad A_y = A\sin\theta_A$$
$$B_x = B\cos\theta_B \qquad B_y = B\sin\theta_B$$
$$R_x = A_x + B_x \qquad R_y = A_y + B_y$$
$$R = \sqrt{R_x^2 + R_y^2} \qquad tg\,\theta_R = \frac{R_y}{R_x}$$



Norme

- La norme d'un vecteur est sa taille.
- Cette taille est calculée par le théorème de Pythagore, puisque les composantes d'un vecteur forment toujours des triangles rectangles deux à deux.



$$\vec{V} = x'\vec{i} + y'\vec{j}$$

$$|\vec{V}| = \sqrt{(x')^2 + (y')^2}$$

- Pour la dimension n on applique le même principe, en faisant que les deux côtés du triangle rectangle soient les projections du vecteur dans un sous-espace de dimension n–1 et dans l'axe de la dimension manquante.

$$|\vec{V}| = \sqrt{(x')^2 + (y')^2 + (z')^2}$$

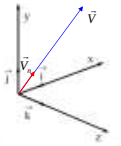
- La norme d'un vecteur AB est la distance de A à B

Normalisation d'un vecteur

- La normalisation fait qu'un vecteur devienne unitaire, c'est-à-dire, avec la norme 1.
- Pour normaliser un vecteur il suffit de diviser toutes ses composantes par sa norme.

$$\frac{\vec{V}}{|\vec{V}|} = \frac{x'\vec{i} + y'\vec{j} + z'\vec{k}}{\sqrt{(x')^2 + (y')^2 + (z')^2}}$$

$$\vec{V}_n = \left(\frac{x'}{\sqrt{(x')^2 + (y')^2 + (z')^2}}, \frac{y'}{\sqrt{(x')^2 + (y')^2 + (z')^2}}, \frac{z'}{\sqrt{(x')^2 + (y')^2 + (z')^2}}\right)$$



$$\vec{V} = |\vec{V}| . \vec{V}_n$$

Matrice

- On appelle matrice M un tableau à deux indices de n*m valeurs numériques extraites du même ensemble E.
- Exemple : $\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} & m_{15} \\ m_{21} & m_{22} & m_{23} & m_{24} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} & m_{44} \end{pmatrix}_R$ n = 4, m = 5.
- Usuellement n et m sont le nombre de lignes et le nombre de colonnes de la matrice.
- Si n = m la matrice est dite carrée.

Matrice

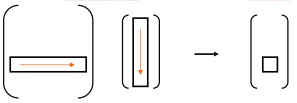
- Opération possible
  - Addition
    - Matrice-matrice
    - Scalaire-matrice
  - Multiplication
    - Matrice-matrice
    - Matrice-vecteur
    - Scalaire-matrice
  - Inversion
  - Transposition

Produit matrice par vecteur

- Soient :
  - un vecteur $\vec{V}$ de dimension n (vi, 1 <= i <= n)
  - une matrice carrée M de dimension m x n (mij, 1 <= i <= m, 1 <= j <= n).
- Le vecteur $\vec{W}$ produit de M par $\vec{V}$ est : $\vec{W} = M.\vec{V}$
- On calcule le produit de chaque ligne de la matrice par le vecteur colonne.

$$w_i = \sum_{k=1}^{n} m_{ik} . v_k \quad 1 \le i \le n$$

m x n n x 1 m x 1



Produit matrice par vecteur

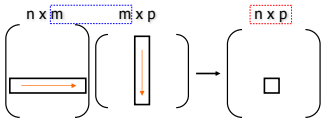
- Exemple

$$\begin{pmatrix} 1 & 3 & 5 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix} = \begin{pmatrix} 70 & 88 \end{pmatrix}$$

Produit matrice par matrice

- Soient deux matrices $M1$ et $M2$ de dimensions respectives :
  - n x m (m1ij, 1 <= i <= n, 1 <= j <= m)
  - m x p (m2ij, 1 <= i <= m, 1 <= j <= p)
- La matrice M produit de M1 par M2 est de dimension n x p est calculée par la formule suivante :

$$m_{ij} = \sum_{k=1}^{m} m1_{ik} \cdot m2_{kj} \quad (1 \le i \le n, 1 \le j \le p)$$



Produit matrice par matrice

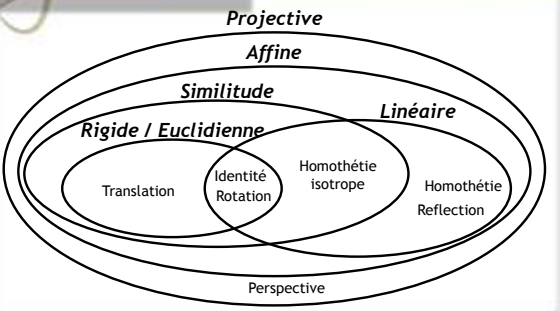
- Exemple

$$\begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix} = \begin{pmatrix} 70 & 88 \\ 178 & 232 \end{pmatrix}$$

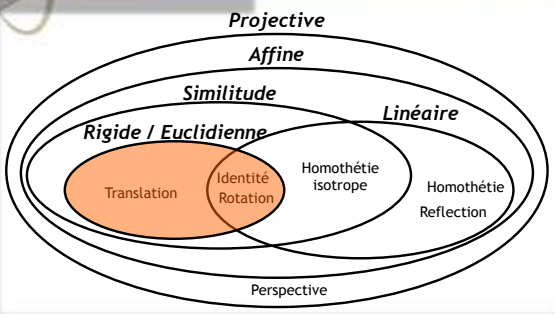
Espace vectoriel - Espace affine

- **Un espace vectoriel** est l'espace où vivent les vecteurs (déplacements ou directions). Ses principales propriétés sont l'existence d'un vecteur nul et la sta bilité de l'espace pour toute combinaison linéaire de vecteurs.
- **Un espace affine** est l'espace dans lequel vivent les points à partir desquels on définit les objets géométriques usuels (droites, ...). Il se construit à partir d'un point de référence (l'origine) et d'un espace vectoriel (déplacements autorisés à partir de ce point).

Transformations

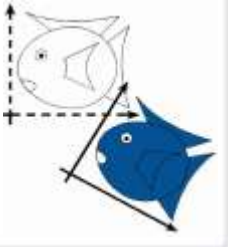


Transformations

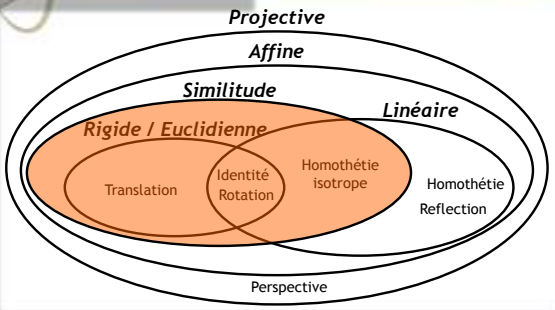


Transformations

- Transformations Euclidiennes rigides : 6 DDL
  - Rotation (3DDL), Translation (3DDL).
- Propriétés
  - Préservent les angles.
  - Préservent les distances.

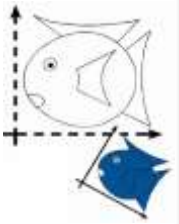


Transformations

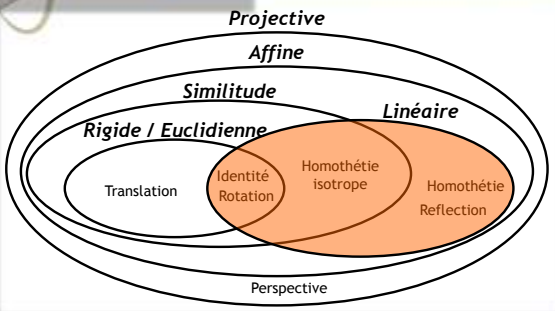


Transformations

- Similitudes : 7 DDL
  - Rotation (3DDL), Translation (3DDL), Homothétie isotrope (1DDL)
- Propriétés
  - Préservent les angles
  - Préservent les rapports de longueurs
  - Préservent les rapports de surfaces
  - Préservent le parallélisme
  - Préservent les formes (un cercle reste un cercle, un carré reste un carré, etc.)
- Généralisation des transformations Euclidiennes et Homothétie

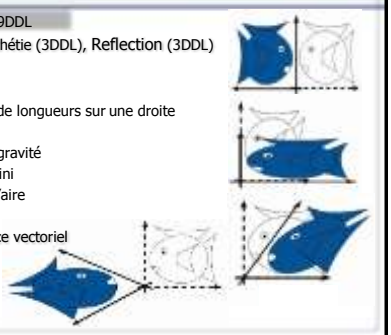


Transformations

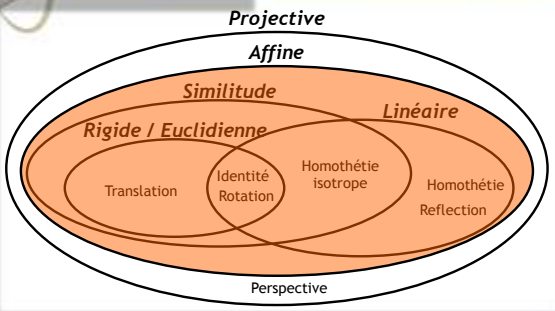


Transformations

- Transformations linéaires : 9DDL
  - Rotation (3DDL), Homothétie (3DDL), Reflection (3DDL)
- Propriétés
  - Préservent les rapports de longueurs sur une droite
  - Préservent les parallèles
  - Préservent le centre de gravité
  - Préservent le plan à l'infini
  - Préservent le rapports d'aire
- Transformations dans espace vectoriel

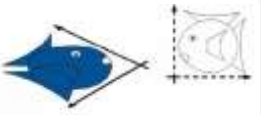


Transformations

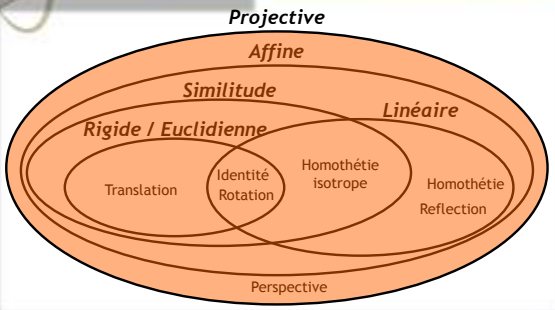


Transformations

- Transformations affines : 12 DDL
  - Rotation (3DDL), Translation (3DDL), Homothétie (3DDL), Reflection (3DDL)
- Propriétés
  - Préservent les rapports de longueurs sur une droite
  - Préservent les parallèles
  - Préservent le centre de gravité
  - Préservent le plan à l'infini
  - Préservent le rapports d'aire
- Transformation dans un espace affine

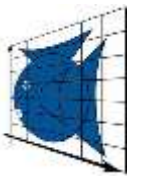


Transformations



Transformations

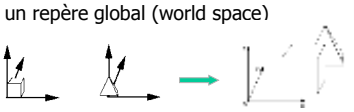
- Transformations projectives ou homographie : 15 DDL
  - Rotation (3DDL), Translation (3DDL), Homothétie (3DDL), Reflection (3DDL), Projection (3DDL)
- Propriétés
  - Préservent les droites (Colinéarité)
  - Préservent le birapport
  - Préservent les intersections
  - Préservent l'incidence



Transformations de modélisation

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

• Application des transformations de composition de scène : Passage du système de coordonnées local de chaque objet 3D (object space) vers un repère global (world space)



Object space World space

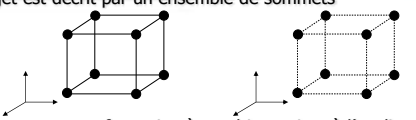
Les transformations élémentaires

Transformations

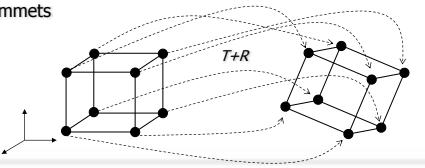
- Utilisations :
  - Déplacement d'un objet dans une scène
  - Déplacement d'un observateur par rapport a une scène
  - Réplication d'un motif ou d'un objet
  - Déformation d'un objet
  - Projection
  - etc.

Transformations

- Un objet est décrit par un ensemble de sommets



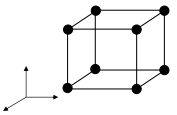
- Appliquer une transformation à un objet revient à l'appliquer à tous ses sommets



Transformations

- On utilise la notation vectorielle
  - Les sommets sont représentés sous forme de vecteurs

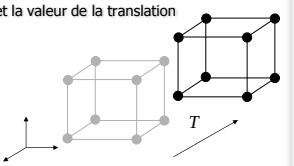
$$p_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$



Transformations

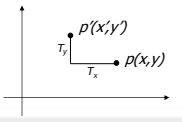
- Translation :
  - Ajouter aux coordonnées du sommet la valeur de la translation

$$x'_i = T_x + x_i$$
$$y'_i = T_y + y_i$$
$$z'_i = T_z + z_i$$



  - Notation matricielle
    - Addition du vecteur de translation

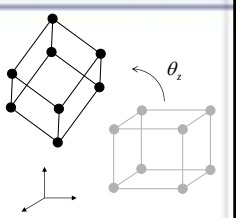
$$\begin{pmatrix} x'_i \\ y'_i \\ z'_i \end{pmatrix} = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} + \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$



Transformations

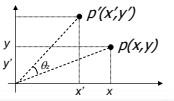
- Rotation par rapport à l'origine autour de l'axe Z
  - Calcul à l'aide de l'algèbre vectorielle

$$x'_i = x_i.\cos\theta_z - y_i.\sin\theta_z$$
$$y'_i = x_i.\sin\theta_z - y_i.\cos\theta_z$$
$$z'_i = z_i$$



  - Notation matricielle
    - Multiplication par la matrice de rotation

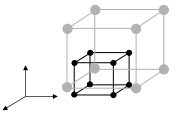
$$\begin{pmatrix} x'_i \\ y'_i \\ z'_i \end{pmatrix} = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$



Transformations

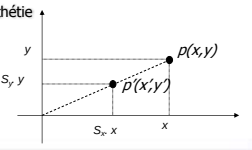
- Homothétie
  - Multiplication par les facteurs d'échelles

$$x'_i = S_x.x_i$$
$$y'_i = S_y.y_i$$
$$z'_i = S_z.z_i$$



  - Notation matricielle
    - Multiplication par la matrice d'Homothétie

$$\begin{pmatrix} x'_i \\ y'_i \\ z'_i \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$

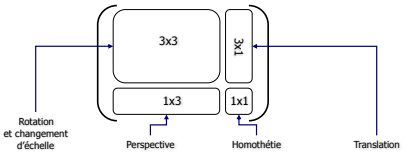


Transformations

- La notation matricielle
  - Permet une notation simple, concise
  - Mais pas vraiment unifiée
    - Addition ou bien multiplication en fonction de la transformation (translation, rotation…)
- On veut une notation unique
  - concaténer plusieurs transformations
  - permette de noter aussi les combinaisons de transformations

Coordonnées homogènes

- Les coordonnées homogènes sont utilisées en synthèse d'image afin d'unifier le traitement des transformations géométriques d'une scène et de les regrouper dans une seule matrice. En effet, si l'on utilise une matrice 2*2 pour les scènes bidimensionnelles et une matrice 3*3 pour les scènes tridimensionnelles, ces matrices ne peuvent exprimer que des rotations.
- Pour exprimer aussi les translations, les changements d'échelle et les projections, on va utiliser des matrices 4*4 pour les scènes tridimensionnelles.



- On rajoute également une 4ème coordonnées aux points manipulés w (w=1) : (x,y,z,w)

Coordonnées homogènes

- Ainsi, si (x,y,z) sont les coordonnées d'un point de la scène à transformer et M_H la matrice 4*4 de coordonnées homogènes, on effectuera la multiplication :

$$M_H * [\ x\ y\ z\ 1\ ]$$

- Ce qui donnera comme résultat $[X\ Y\ Z\ H]$.
- Les coordonnées du point transformé seront alors $(x'\ y'\ z') = (X/H, Y/H, Z/H)$.
- La matrice 4*4 des coordonnées homogènes peut être considérée comme étant composée de 4 sous-matrices, chacune d'elle étant associée à un type de transformation.

Manipulations géométriques

- Soit un point p
- Soit une transformation géométrique définie par la matrice M donnée en coordonnées homogènes.
- Le Point p' transformé de par la matrice M est : $p' = M \,.\, p$
- Transformations :
  - Translation
  - Rotation
  - Changements d'échelle
  - Symétries
  - Projection
  - Affinités orthogonales
  - etc.

Translations

$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} I_3 & T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix}$$

$$P' = T.P = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} T_x + x \\ T_y + y \\ T_z + z \\ 1 \end{pmatrix} = T + P$$

Rotations (angles d'Euler)

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix}$$

$$R = R_x.R_y.R_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P' = R.P = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotations (angles d'Euler)

- Règles pour la construction de la matrice de rotation d'angle Θ
  - Ligne 1 associée à x, ligne 2 à y et ligne 3 à z
  - 1 sur la diagonale pour l'axe de rotation et la coordonnée homogène
  - cos(Θ) sur la diagonale pour les deux autres axes
  - sin(Θ) sur les diagonales supérieure et inférieure pour "compléter le carré"
  - Sur la ligne suivant celle de l'axe de rotation, le sinus est précédé d'un signe '-'

Homothétie isotrope

$$R = \begin{pmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix}$$

$$P' = S.P = \begin{pmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S.x \\ S.y \\ S.z \\ 1 \end{pmatrix} = S.P$$

Homothétie : Affinités orthogonales

$$R = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_{3x3} & 0_{3x1} \\ 0_{1x3} & 1 \end{pmatrix}$$

$$P' = S.P = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S_x.x \\ S_y.y \\ S_z.z \\ 1 \end{pmatrix} = S.P$$

$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Affinité d'axe x par rapport au plan yOz

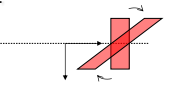
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Affinité d'axe y par rapport au plan xOz

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Affinité d'axe z par rapport au plan xOy

Glissement

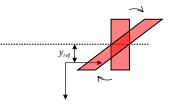
- Appelée aussi *shear* ou cisaillement : étirement suivant un axe
  - La matrice d'un glissement parallèle à x et de rapport k est :

$$\begin{pmatrix} 1 & k & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



  - La matrice d'un glissement parallèle à x, de rapport k et de ligne de base y = y_{ref} est :

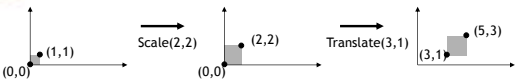
$$\begin{pmatrix} 1 & k & -k.y_{ref} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Composition de Transformations

Composition de Transformations

Exemple :



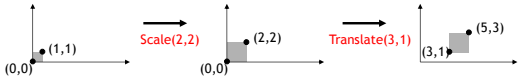
Multiplication de matrices : $p' = T.(S.p) = T.S.p$

$$T.S = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

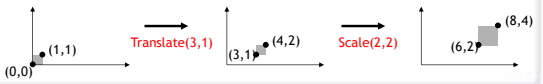
Composition de Transformations

Non-commutatif

homothétie puis translation : $p' = T.(S.p) = T.S.p$



translation puis homothétie : $p' = S.(T.p) = S.T.p$



Composition de Transformations

$$T(3,1).S(2,2) = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$S(2,2).T(3,1) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Composition de Transformations

Cas général

$p' = M \,.\, p$ et $p'' = M' .\, p'$

=>

$p'' = M' \,.\, M \,.\, p$

ou

$p'' = M'' \,.\, p$ avec $M'' = M' \,.\, M$

Composition de Transformations

Ordre d'Application

$p'' = M' \,.\, M \,.\, p$ => M puis M'

On applique M à P, puis M' au résultat :
l'ordre d 'application des transformations se lit de droite à gauche, et non dans le sens de la lecture

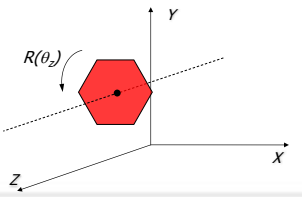
Composition de Transformations

- Les transformations élémentaires sont définies par rapport à l'origine du repère de la scène
- Pour se placer d'un points quelconque , on doit :
  1. Revenir à l'origine du repère (translation) : $T_{P->O}$
  2. Faire la(les) transformation(s) voulue(s)
  3. Se remettre au point de départ (translation) : $T_{O->P}$

Composition de Transformations

Exemple 1 :

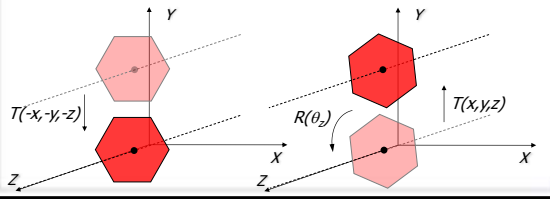
On désire établir la transformation consistant à effectuer une rotation de Θ_z radians autour de l'axe colinéaire à z passant par le point P de coordonnées (T_x, T_y, T_z).



Composition de Transformations

Cette transformation M est réalisée en amenant p à l'origine par une translation T de $-p$, puis en effectuant une rotation R d'angle Θ_z autour de l'axe Oz, et enfin en ramenant p à sa position initiale par une translation T' de p :

$$p' = \left(T(x,y,z) \,.\, R(\theta_z) \,.\, T(-x,-y,-z)\right) \,.\, p = M \,.\, p$$



Composition de Transformations

$$M = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & -T_x\cos\theta_z + T_y\sin\theta_z \\ \sin\theta_z & \cos\theta_z & 0 & -T_x\sin\theta_z + T_y\cos\theta_z \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & -T_x\cos\theta_z + T_y\sin\theta_z + T_x \\ \sin\theta_z & \cos\theta_z & 0 & -T_x\sin\theta_z + T_y\cos\theta_z + T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

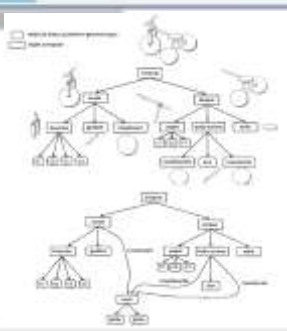
Composition de Transformations

- Les compositions de transformations servent à
  - Décrire une hiérarchie de transformations : graphe de scène
  - Changement de repère

Modélisation hiérarchique

- Un modèle hiérarchique permet de décrire facilement des objets complexes composés d'objets simples
- La scène est organisée dans un arbre
- Les objets ne sont plus définis par leur transformation absolue par rapport au repère global, mais par leur transformation relative dans cet arbre :
  - Le repère associé à la racine est le repère de la scène
  - A chaque nœud est associé un repère
  - A chaque arc est associée une transformation géométrique qui po sitionne l'objet fils dans le repère de son père
- Un objet peut être inclus plusieurs fois dans la hiérarchie :
  - la structure de données est un Graphe Orienté Acyclique (DAG)

Modélisation hiérarchique

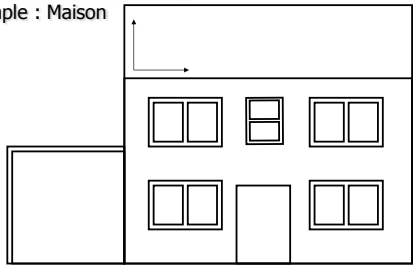


Modélisation hiérarchique

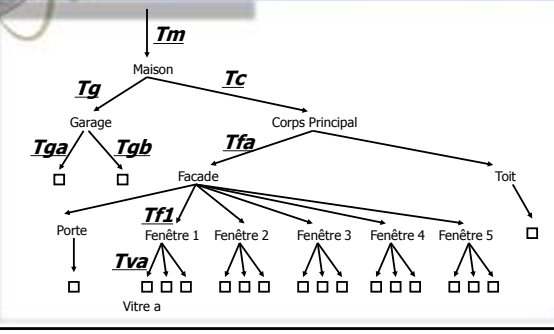
- Construction du graph
  - On commence par un processus descendant ("top-down") dans lequel on effectue une décomposition récursive du modèle géométrique en objets plus simples jusqu'à aboutir à des objets élémentaires (primitives géométriques)
  - On construit la chaine cinématique (graph) depuis la racine pour aboutir aux nœuds

Modélisation hiérarchique

Exemple : Maison



Modélisation hiérarchique



Modélisation hiérarchique

Pile de transformations

porte a du garage ➡ Tm.Tg.Tga

porte b du garage ➡ Tm.Tg.Tgb

vitre a ➡ Tm.Tc.Tfa.Tf1.Tva

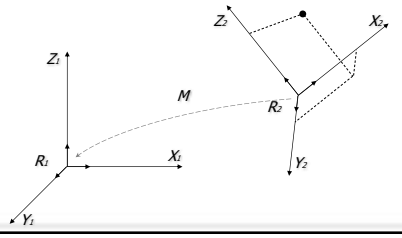
etc . . . .

Changement de repère

- Permet de transformer les coordonnées d'un point exprimées dans un premier repère en coordonnées exprimées dans deuxième repère
- Utile lorsque :
  - Les objets manipulés sont définis dans des repères locaux
    - Animation, modélisation, etc.
  - la modélisation des caméras.

Changement de repère

- Soient deux repères R_1 et R_2
- Soit M la matrice de passage du repère R_2 au repère R_1
- Soient $(x_2, y_2, z_2, 1)$ les coordonnées de p_2 dans R_2
- Soient $(x_1, y_1, z_1, 1)$ les coordonnées de p_1 dans R_1



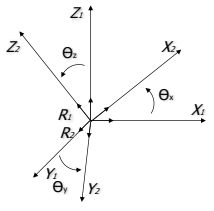
Changement de repère

$$p_2 = M.p_1$$

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}_{R_1} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}_{2\to1} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix}_{R_2}$$

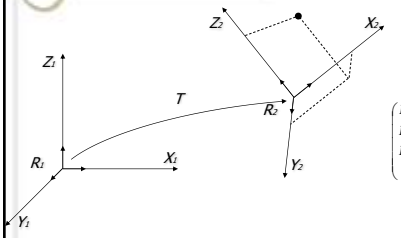
$$M = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}_{2\to1}$$

Changement de repère



$$\begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{2\to1}$$

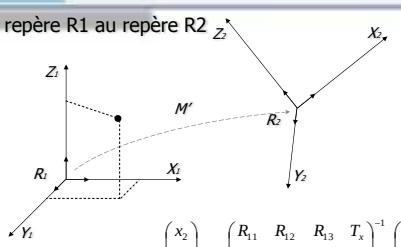
Changement de repère



$$\begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}_{2\to1}$$

Changement de repère

- Passage du repère R1 au repère R2



$$p_1 = M'.p_2 \Rightarrow p_1 = M^{-1}.p_2$$

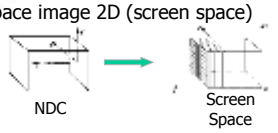
$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix}_{R_2} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}_{2\to 1} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}_{R_1}$$

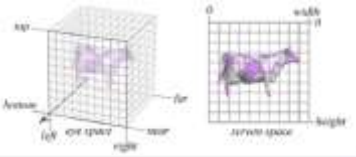
La projection

Transformation écran

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- Pixelisation (Rasterization)
- Visibilité / Affichage

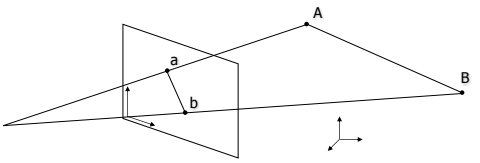
- Les primitives 3D sont projetées sur l'espace image 2D (screen space)





La projection

- La projection est une réduction du nombre de dimensions.
- L'infographie est concernée par les projections de 3-D vers 2-D.

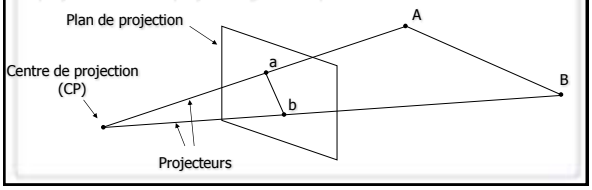


XX

- On note que la projection d'une droite en 3D est une droite en 2D.
- Il est suffisant de projeter les sommets d'une droite 3D et puis de tracer la droite 2D entre ces projections.

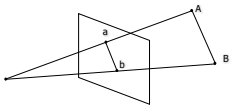
La projection

- Les projections des objets sont formées par les intersections de lignes appelées projecteurs avec un plan appelé plan de vue ou plan de projection.
- Les projecteurs sont des lignes partant d'un point arbitrairement appelé centre de projection (CP), en traversant chaque point d'un objet.
- Si les projecteurs sont des lignes droites, et que le plan de projection est plat, la projection est une projection géométrique 2D.

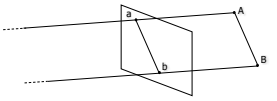


Les projections

- Si le centre de projection (CP) est localisé à un point fini de l'espace tridimensionnel, le résultat est une projection en perspective.
- Si le CP est localisé à l'infini, tous les projecteurs sont parallèles et le résultat est une projection parallèle.



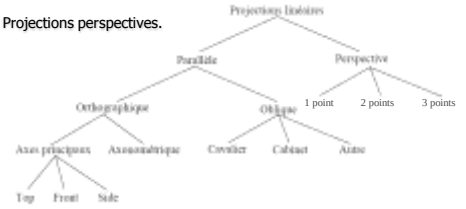
Projection perspective



projection parallèle

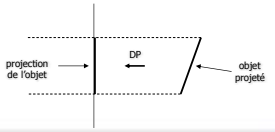
Les projections

- Deux familles de projections :
  - Projections parallèles.
  - Projections perspectives.



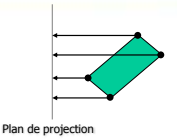
Projections parallèles

- Conditions :
  - La projection se fait dans le plan de projection suivant une direction de projection (DP).
  - Les lignes parallèles restent parallèles
- Propriétés
  - Les projections parallèles conservent les rapports des distances selon une direction donnée
  - Pas réaliste mais peut être utilisée pour des mesures exactes
- Il existe deux types de projections parallèles dépendant du fait que la direction de projection soit perpendiculaire au plan de projection :
  - Projections orthographiques
  - Projections obliques



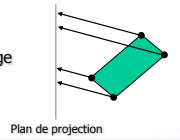
Projections parallèles

- Projection orthographique
  - Projecteurs parallèles entre eux
  - Projecteurs perpendiculaires au plan image



Plan de projection

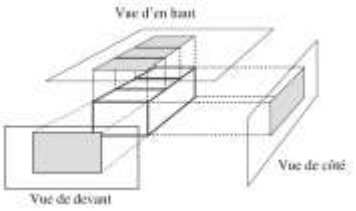
- Projection Oblique
  - Projecteurs parallèles entre eux
  - Projecteurs NON perpendiculaires au plan image



Plan de projection

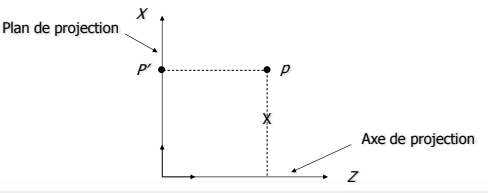
Projection orthographique

- Vue de devant, d'en haut, et de côté



Projection orthographique

- La projection orthographique consiste à supprimer une dimension
  - La dimension supprimée est celle suivant laquelle on réalise la projection



Projection orthographique

- Forme matricielle :

$$p' = M_{PZ} . p$$

$$\begin{pmatrix} x' \\ y' \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Projection orthographique

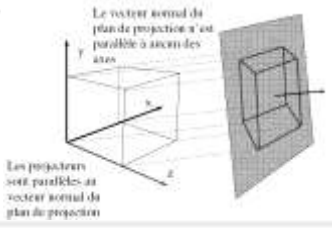
- Vues de devant, d'en haut et de côté : la projection est faite sur un des plans perpendiculaires aux axes des coordonnées en mettant une des coordonnées du point à 0.

$$M_{PX} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{PY} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{PZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Elimination de la dimension suivant la quelle nous réalisons la projection

Projection orthographique

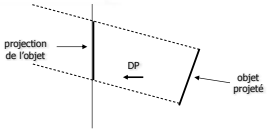
- Projection orthographique axonométrique
  - Le plan de projection n'est pas perpendiculaire aux axes des coordonnés, donc plusieurs facettes de l'objet sont montrées en même temps.

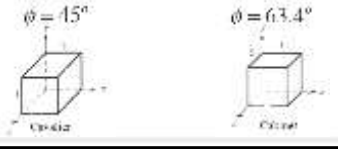


Projection oblique

- Direction de la projection n'est pas perpendiculaire au plan de projection
- DP coupe le plan de projection en faisant un angle oblique avec celui-ci.
- Une projection cavalière est obtenue quand l'angle est de 45º
- Pour une projection cabinet l'angle est de 63.4º.
- La matrice de projection est une combinaison de transformations d'étirement et de projection orthographique.

Projection oblique





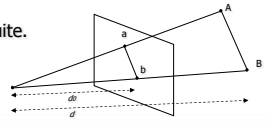
Projection perspective

Projection perspective

- Réaliste : La taille de la projection d'un objet varie inversement avec sa distance de CP

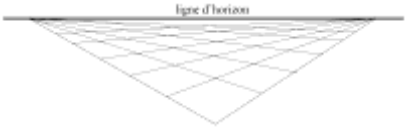
$$|ab| = d_0 \cdot 1/d \cdot |AB|$$

- Ne maintient pas la forme et les mesures exactes
- Les projections perspectives ne conservent pas le parallélisme des droites non parallèles au plan de projection
- La projection de n'importe quel ensemble de lignes parallèles non parallèles au plan de projection convergent vers un même point
  - Le point de fuite.



Projection perspective

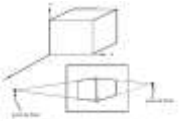
- Point de fuite
  - Si une droite D coupe le plan de projection, il existe un point F, appelé point de fuite appartenant à la projection de toute droite parallèle à D.



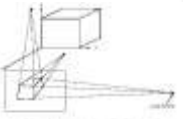
Projection perspective

- On différencie les projections en perspective par le nombre de points de fuite pour les directions des axes du repère.

Projection à un point de fuite



Projection à des points de fuite



Projection à trois points de fuite

Projection perspective

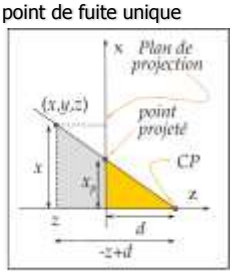
- Calcul d'une perspective de point de fuite unique

$$\frac{x_p}{d} = \frac{x}{-z+d} \Rightarrow$$

$$x_p = \frac{x\,d}{-z+d} \Rightarrow$$

$$x_p = \frac{x}{\frac{-z}{d}+1} = \frac{x}{-zr+1}$$

où $r = 1/d$



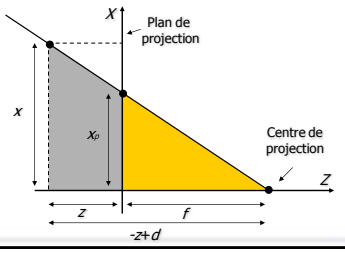
Projection perspective

- Calcul d'une perspective de point de fuite unique

$$\frac{x_p}{f} = \frac{x}{-z+f} \Rightarrow$$

$$x_p = \frac{x.f}{-z+f} \Rightarrow$$

$$x_p = \frac{x}{-\frac{z}{f}+1}$$



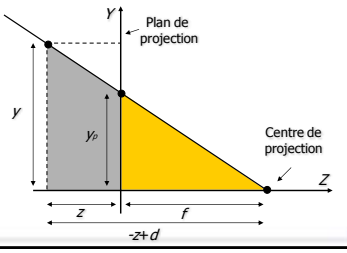
Projection perspective

- Calcul d'une perspective de point de fuite unique

$$\frac{y_p}{f} = \frac{y}{-z+f} \Rightarrow$$

$$y_p = \frac{y.f}{-z+f} \Rightarrow$$

$$y_p = \frac{y}{-\frac{z}{f}+1}$$



Projection perspective

- Représentation en forme matricielle :

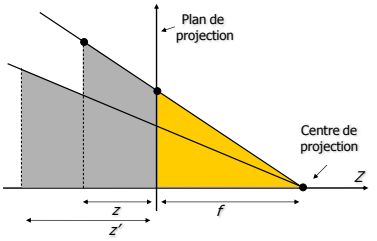
$$x_p = \frac{x}{-\frac{z}{f}+1},\ y_p = \frac{y}{-\frac{z}{f}+1} \Rightarrow$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{-1}{f} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow$$

Projection perspective

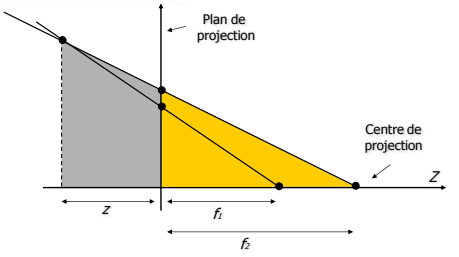
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -\frac{z}{f}+1 \end{pmatrix} \Rightarrow \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{x}{-\frac{z}{f}+1} \\ \frac{y}{-\frac{z}{f}+1} \\ \frac{z}{-\frac{z}{f}+1} \\ 1 \end{pmatrix}$$

Projection perspective



La taille de la projection varie en fonction de la distance de l'objet au plan de projection

Projection perspective



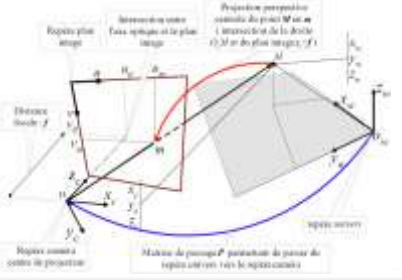
La taille de la projection varie en fonction de la distance du centre de projection au plan de projection

Projection perspective

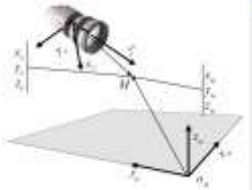
- Plus *f* est large, plus la projection tend vers une projection parallèle, où la profondeur a un petit effet dans les points projetés et moins de sensation dans la perspective obtenue.
- Plus *f* est petit, plus la projection diverge de la projection parallèle. La profondeur a une influence considérable sur les points projetés à la limite de créer des effets de perspective exagérés.

Projection perspective

- Transformations nécessaires

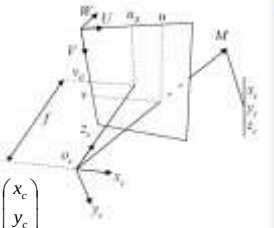


Projection perspective



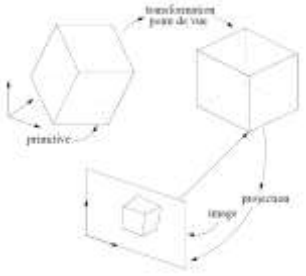
$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r11 & r12 & r13 & 0 \\ r21 & r22 & r23 & 0 \\ r31 & r32 & r33 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}$$

Projection perspective



$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & u_0 \\ 1 & 1 & 1 & v_0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}$$

Projection perspective



Projection perspective

Passage du repère monde au plan image

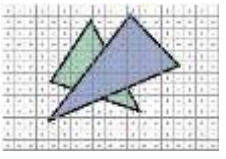
$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & u_0 \\ 1 & 1 & 1 & v_0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r11 & r12 & r13 & 0 \\ r21 & r22 & r23 & 0 \\ r31 & r32 & r33 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & u_0 & 0 \\ 1 & f & v_0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} r11 & r12 & r13 & t_x \\ r21 & r22 & r23 & t_y \\ r31 & r32 & r33 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}$$

Rasterisation

- Transformations de modélisation
- Illumination (Shading)
- Transformations d'affichage
- Clipping
- Transformation écran (Projection)
- **Pixelisation (Rasterization)**
- Visibilité / Affichage

- Découpe des primitives 2D en pixels
- Interpole les valeurs connues aux sommets : couleur, profondeur, etc. pour chaque fragment affiché



Rasterisation

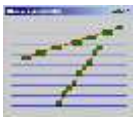
- Traçage :
  - Segments de droites
  - Cercles
  - Ellipses
- Remplissage des primitives projetées

Tracé de segments de droites

- Algorithme de base pour beaucoup de traitements de l'Informatique Graphique:
  - dessin en fil de fer,
  - remplissage,
  - élimination des parties cachées,
  - ...


Bons tracés


Mauvais tracés


Mauvais tracés

Tracé de segments de droites

- Trois impératifs:
  - Tout point du segment discret est traversé par le segment continu.
  - Tout point du segment discret touche au moins un autre point soit par l'un de ses cotés (4-connexité), soit par un de ses sommets (8-connexité).
  - On trace le moins de points possibles.


4-connexité

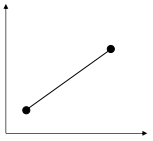

8-Connexité

Tracé de segments par l'équation cartésienne

- Le tracé d'un segment entre deux points (x_1,y_1) et (x_2,y_2) de R_2.
- On pose comme hypothèse simplificatrice :

$$x_2>x_1,\ y_2>y_1 \text{ et } (x_2-x_1) >= (y_2-y_1)$$

  - Tous les autres cas peuvent s'y rapporter.
- On utilise l'équation cartésienne de la droite : $y = ax + b$



avec

$$a = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

$$b = y_1 + ax_1$$

Tracé de segments par l'équation cartésienne

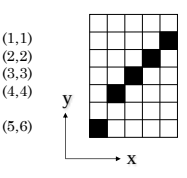
```
DroiteSimple(int x1, int x2, int y1, int y2)
{
   a =  y2 – y1 / x2 – x1 ;
   b = y1 – a . x1 ;
   x <- x1 ;
   while (x < x1)
        {
         x <- x1+1 ;
        AfficherPixel(x, (int) a.x+b) ;
        }
}
```

Tracé de segments par l'équation cartésienne

- Incrémentation suivant l'axe x

```
DroiteSimple(int x1, int x2, int y1, int y2)
{
   a = y2 – y1 / x2 – x1 ;   // a = 1.25
   b = y1 – a . x1 ;         // b = -0.25
   x <- x1 ;
   while (x < x2)            // x : 1 -> 5
        {
         x <- x1+1 ;
        AfficherPixel(x, (int) a.x+b) ;
        }
}
```

Tracé : (1,1) à (5,6)

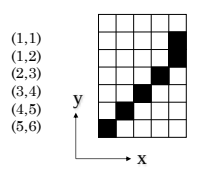


Tracé de segments par l'équation cartésienne

- Incrémentation suivant l'axe y

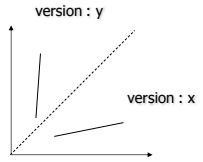
```
DroiteSimple(int x1, int x2, int y1, int y2)
{
   a = y2 – y1 / x2 – x1 ;   // a = 1.25
   b = y1 – a . x1 ;         // b = -0.25
   x <- x1 ;
   while (y < y2)            // y : 1 -> 6
        {
         y <- y1+1 ;
        AfficherPixel((int) (y-b)/a, y) ;
        }
}
```

Tracé : (1,1) à (5,6)



Tracé de segments par l'équation cartésienne

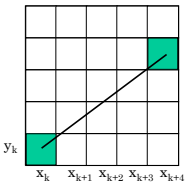
- Il faut switcher entre les deux versions en fonction de l a pente de la droite
  - Pente < 1 : version x
  - Pente > 1 : version y



Tracé de segments par l'équation cartésienne

- Caractéristiques:
  - Simplicité algorithmique
  - Lenteur due à l'utilisation de réels, d'une division et de multiplications et d'opérations d'arrondissement (cast)

Algorithmes de Bresenham

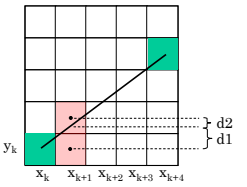


Jack Bresenham

1962

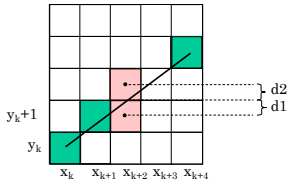
Algorithmes de Bresenham

d1-d2 > 0 ?



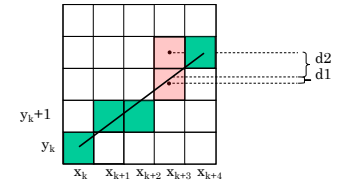
d1-d2 > 0

Algorithmes de Bresenham



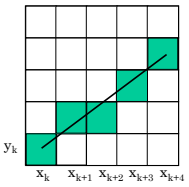
d1-d2 = 0

Algorithmes de Bresenham



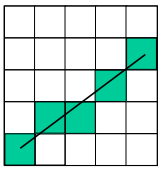
d1-d2 < 0

Algorithmes de Bresenham



Algorithmes de Bresenham

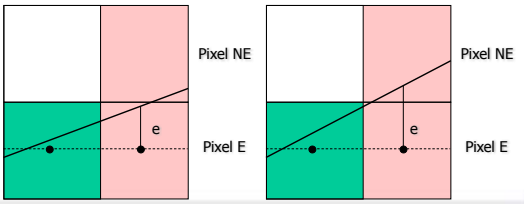
```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
   x <- x1 ;
   y <- y1 ;
   e = 2 . dy - dx ;
   IncH = 2 . dy ;
   IncD = 2 . (dy – d x) ;
   while (x < x2)
   {
        AfficherPixel(x, y) ;
        x = x + 1;
        if (e > 0)
        {
            y = y + 1;
            e = e + IncD ;
        }
        else
            e = e + IncH ;
    }
}
```



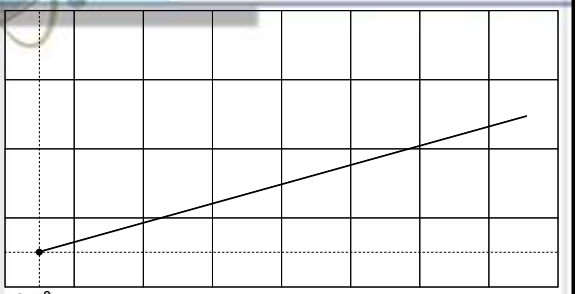
Algorithmes de Bresenham

Principe :

- Paramètre de décision : e (distance pixel -> droite)
  - Si e < 0.5 pixel = > activer le pixel E
  - Si e > 0.5 pixel = > activer le pixel NE

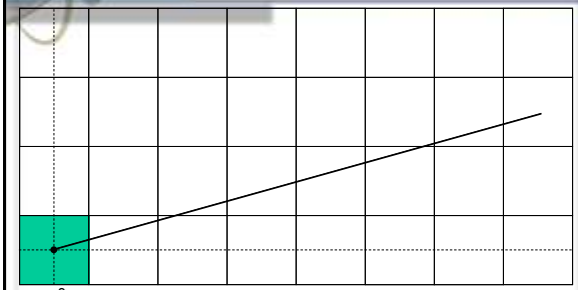


Algorithmes de Bresenham



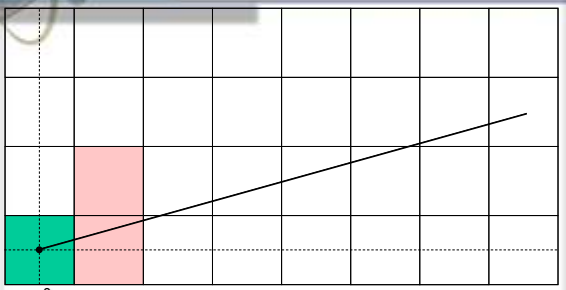
e = 0
x= x_1
y= y_1

Algorithmes de Bresenham



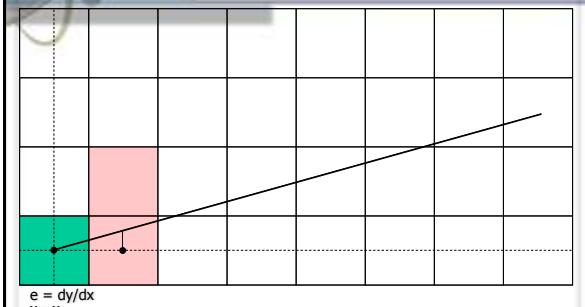
e = 0
x= x_1
y= y_1

Algorithmes de Bresenham



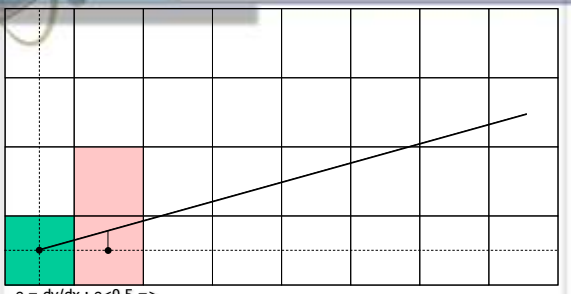
e = 0
x= x_1
y= y_1

Algorithmes de Bresenham



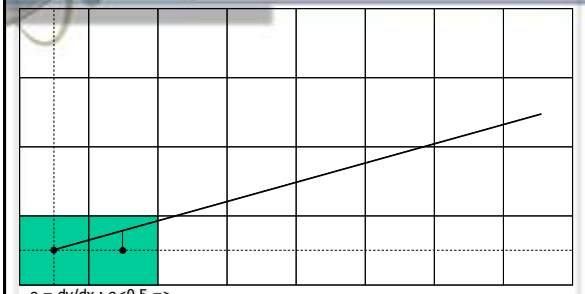
e = dy/dx
x= x_1
y= y_1

Algorithmes de Bresenham



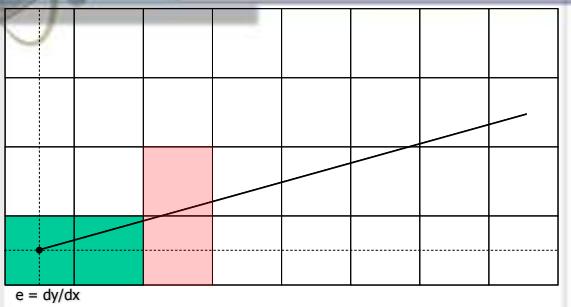
e = dy/dx ; e<0.5 =>
x= x+1
y= y

Algorithmes de Bresenham



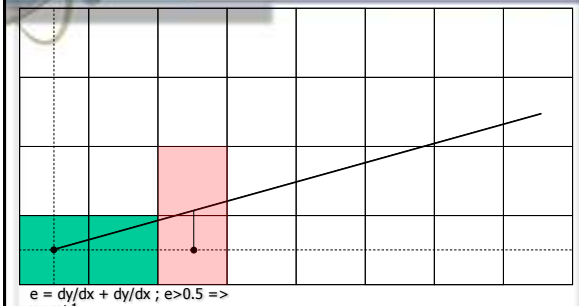
e = dy/dx ; e<0.5 =>
x= x+1
y= y

Algorithmes de Bresenham



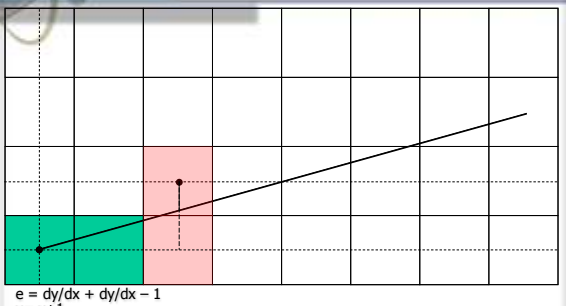
e = dy/dx

Algorithmes de Bresenham



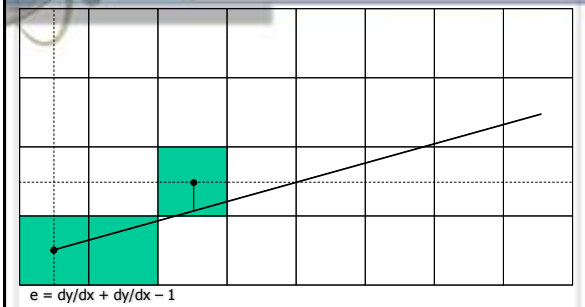
e = dy/dx + dy/dx ; e>0.5 =>
x= x+1
y= y+1

Algorithmes de Bresenham



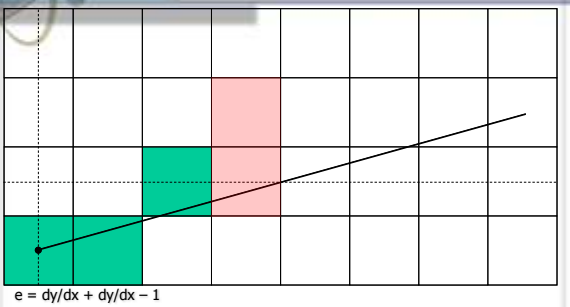
e = dy/dx + dy/dx – 1
x= x+1
y= y+1

Algorithmes de Bresenham



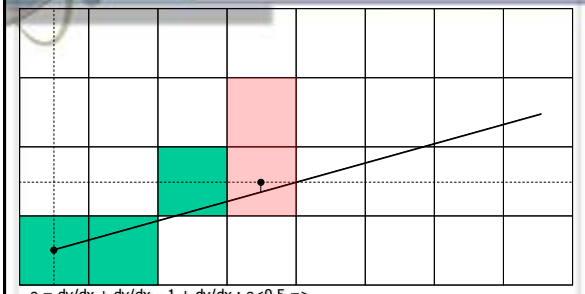
e = dy/dx + dy/dx – 1

Algorithmes de Bresenham



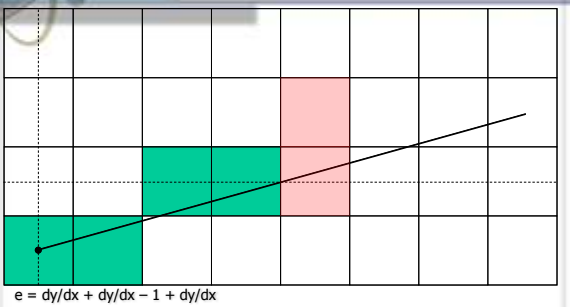
e = dy/dx + dy/dx – 1

Algorithmes de Bresenham



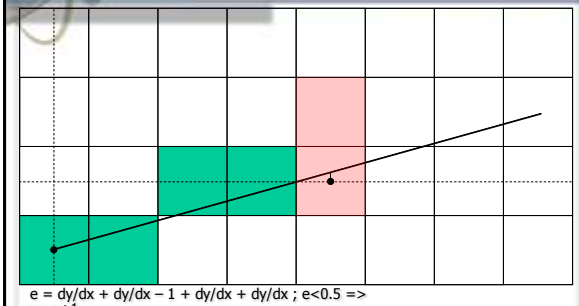
e = dy/dx + dy/dx – 1 + dy/dx ; e<0.5 =>
x= x+1
y= y

Algorithmes de Bresenham



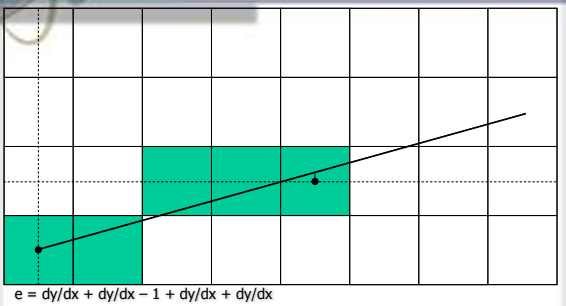
e = dy/dx + dy/dx – 1 + dy/dx

Algorithmes de Bresenham



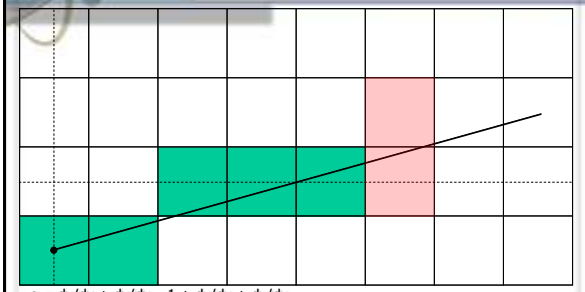
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx ; e<0.5 =>
x= x+1
y= y

Algorithmes de Bresenham



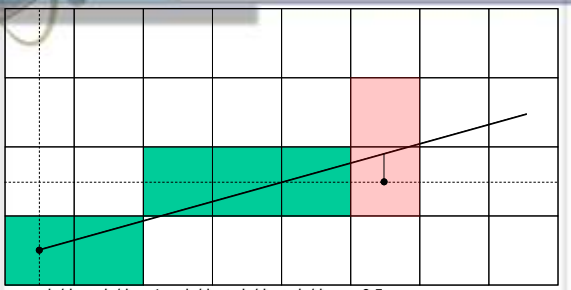
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx

Algorithmes de Bresenham



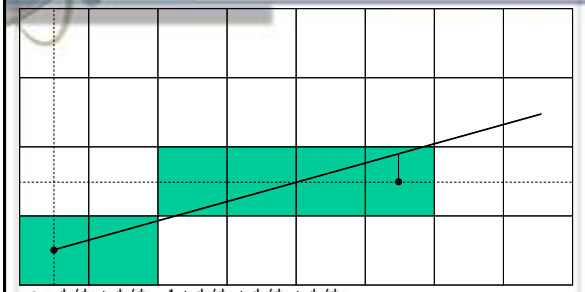
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx

Algorithmes de Bresenham



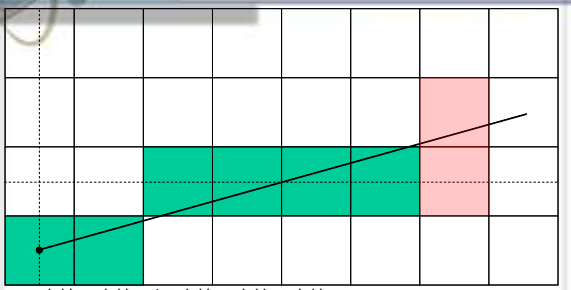
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx ; e<0.5 =>
x= x+1
y= y

Algorithmes de Bresenham



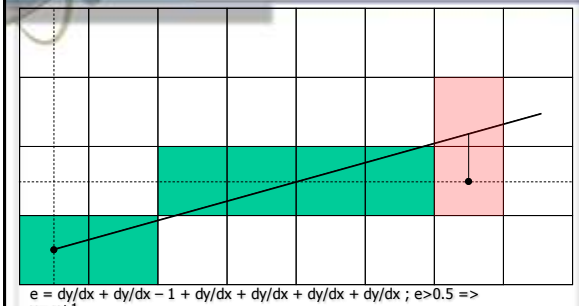
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx

Algorithmes de Bresenham



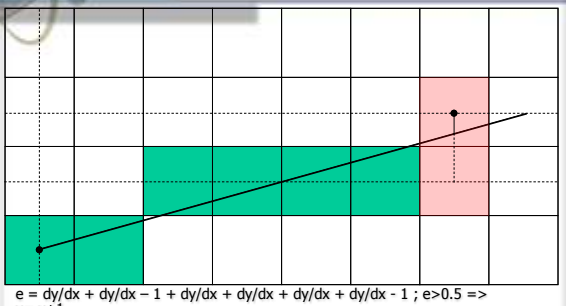
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx

Algorithmes de Bresenham



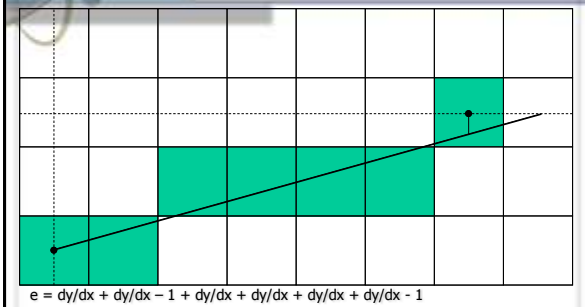
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx + dy/dx ; e>0.5 =>
x= x+1
y= y+1

Algorithmes de Bresenham



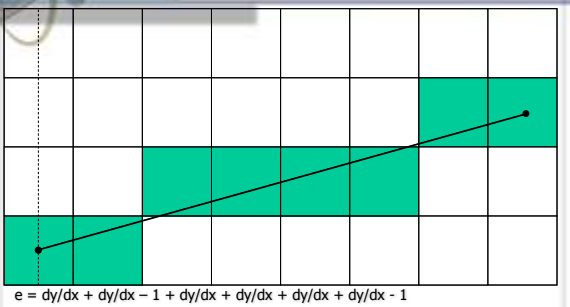
e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx + dy/dx - 1 ; e>0.5 =>
x= x+1
y= y+1

Algorithmes de Bresenham



e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx + dy/dx - 1

Algorithmes de Bresenham



e = dy/dx + dy/dx – 1 + dy/dx + dy/dx + dy/dx + dy/dx - 1

Algorithmes de Bresenham

. Algorithme de base

```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
  a = dy / dx ;
  x <- x1 ;
  y <- y1 ;
  e = 0 ;
  while (x < x2)
  {
      AfficherPixel(x, y) ;
      e = e + a ;
      x= x + 1;
      if (e > 0.5)
      {
          y =y + 1;
          e =e - 1;
      }
  }
}
```

Algorithmes de Bresenham

. supprimer la division par dx

```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
  x <- x1 ;
  y <- y1 ;
  e = 0 ;
  while (x < x2)
  {
      AfficherPixel(x, y) ;
      e = e + dy ;
      x = x + 1;
      if (e > dx/2)
      {
          y = y + 1;
          e = e - dx;
      }
  }
}
```

Algorithmes de Bresenham

. division par 2 à supprimer

```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
  x <- x1 ;
  y <- y1 ;
  e = 0 ;
  while (x < x2)
  {
      AfficherPixel(x, y) ;
      e = e + 2 . dy ;
      x = x + 1;
      if (e > dx)
      {
          y = y + 1;
          e = e – 2 . dx;
      }
  }
}
```

Algorithmes de Bresenham

. optimisations pour rendre l'algorithme plus rapide : tester le signe de e au lieu de le comparer à un autre nombre

```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
  x <- x1 ;
  y <- y1 ;
  e = - dx ;
  while (x < x2)
  {
      AfficherPixel(x, y) ;
      e = e + 2 . dy ;
      x = x + 1;
      if (e > 0)
      {
          y = y + 1 ;
          e = e – 2 . dx ;
      }
  }
}
```

Algorithmes de Bresenham

. modifier e qu'une seule fois lors de chaque itération

```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
  x <- x1 ;
  y <- y1 ;
  e = 2 . dy - dx ;
  while (x < x2)
  {
      AfficherPixel(x, y) ;
      x = x + 1;
      if (e > 0)
      {
          y = y + 1;
          e = e + 2 . (dy – d x) ;
      }
      else
          e = e + 2 . dy ;
  }
}
```

Algorithmes de Bresenham

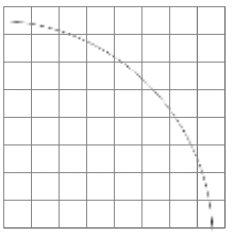
. calculer d'avance les deux incréments possibles pour e : déplacement horizontal ou diagonal

```
DroiteBresenham(int x1, int x2, int y1, int y2)
{
  dx = / x2 – x1
  dy = y2 – y1
  x <- x1 ;
  y <- y1 ;
  e = 2 . dy - dx ;
  IncH = 2 . dy ;
  IncD = 2 . (dy – d x) ;
  while (x < x2)
  {
      AfficherPixel(x, y) ;
      x = x + 1;
      if (e > 0)
      {
          y = y + 1;
          e = e + IncD ;
      }
      else
          e = e + IncH ;
  }
}
```

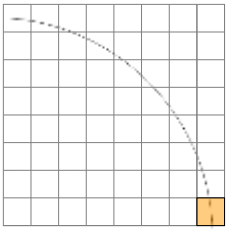
Algorithmes de Bresenham

- Caractéristiques :
  - Plus grande complexité algorithmique.
  - Rapidité due à
    - Utilisation exclusive d'entiers courts (valeurs maximales de l'ordre de la résolution de l'écran -> petites valeurs)
    - Opérations arithmétiques simples sur ces entiers (additions, soustractions et comparaisons).

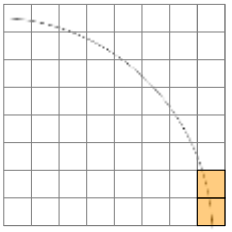
Tracé de cercles



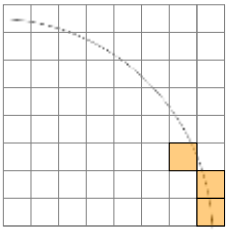
Tracé de cercles



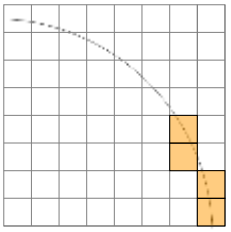
Tracé de cercles



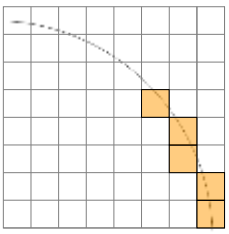
Tracé de cercles



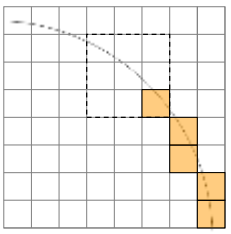
Tracé de cercles



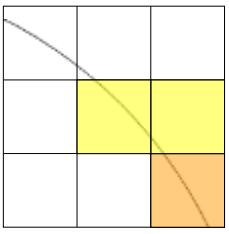
Tracé de cercles



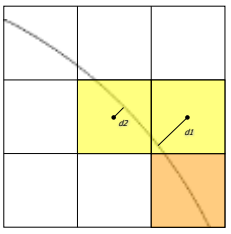
Tracé de cercles



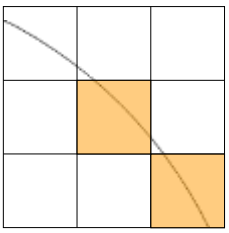
Tracé de cercles



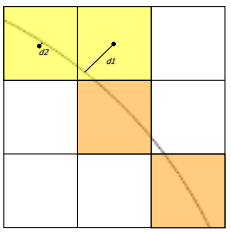
Tracé de cercles



Tracé de cercles

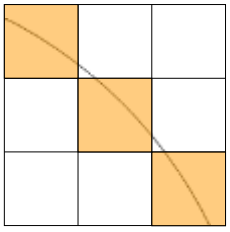


Tracé de cercles



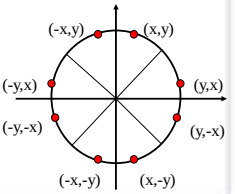
d1>d2

Tracé de cercles

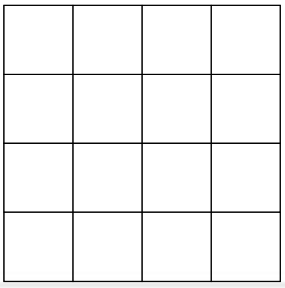


Algorithmes de Bresenham pour le tracé de cercles

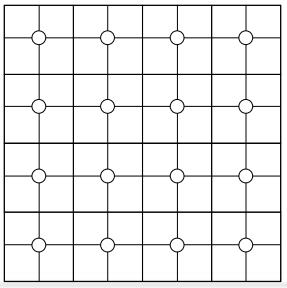
- On suppose le centre à l'origine
- Le rayon *r* est un entier
- On trace seulement le deuxième octant
- Les autres octants tracés par symétrie



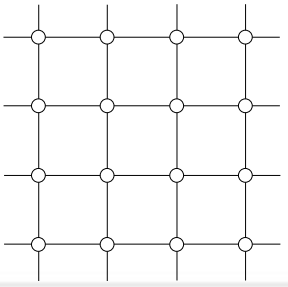
Algorithmes de Bresenham pour le tracé de cercles



Algorithmes de Bresenham pour le tracé de cercles

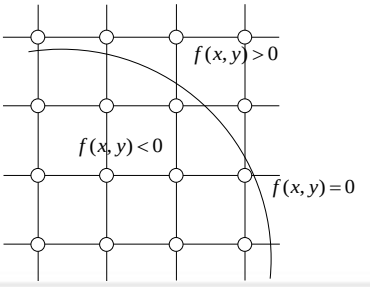


Algorithmes de Bresenham pour le tracé de cercles



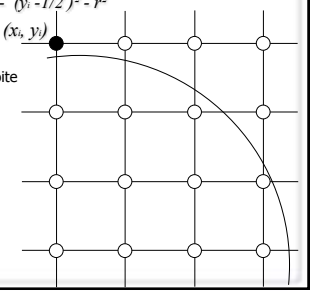
Algorithmes de Bresenham pour le tracé de cercles

- Forme implicite du cercle : $f(x,y) = x^2 + y^2 - r^2$



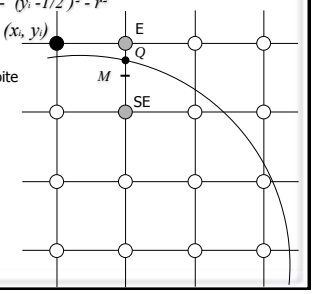
Algorithmes de Bresenham pour le tracé de cercles

- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$
  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E



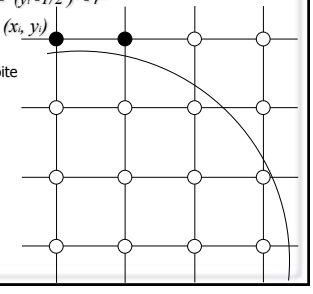
Algorithmes de Bresenham pour le tracé de cercles

- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$
  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E



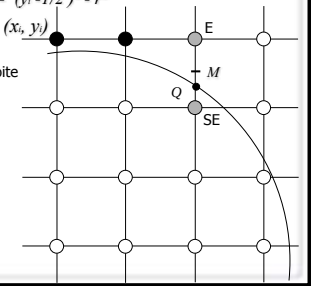
Algorithmes de Bresenham pour le tracé de cercles

- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$
  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E



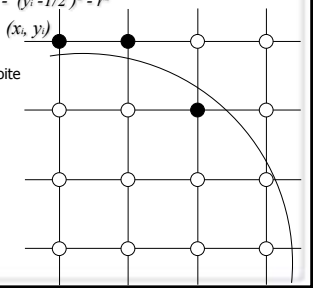
Algorithmes de Bresenham pour le tracé de cercles

- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$
  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E



Algorithmes de Bresenham pour le tracé de cercles

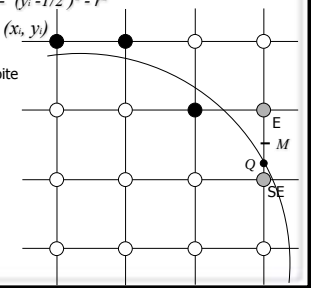
- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$



  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E

Algorithmes de Bresenham pour le tracé de cercles

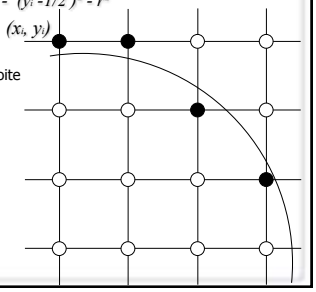
- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$



  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E

Algorithmes de Bresenham pour le tracé de cercles

- On calcule la valeur de $f(x,y)$ au point M, noté e_i :
  - $e_i = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 - (y_i - 1/2)^2 - r^2$



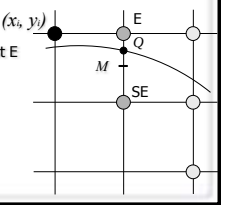
  - $e_i \geq 0$: M est au-dessus de la droite
    - Q est sous M on va à SE
  - $e_i < 0$: M est sous la droite
    - Q est au-dessus de M on va à E

Algorithmes de Bresenham pour le tracé de cercles

- Le calcul de l'erreur se fait manière incrémentale
  - Pour chaque pixel activé (en cours de trainement) nous calculons le paramètre de décision du pixel suivant

$e_i = f(x_i + 1, y_i - 1/2)$

- Si $e_i > 0$ (M à l'extérieur) alors le pixel à afficher est SE
  - Le prochain point milieu sera $(x_i + 2, y_i - 3/2)$
  - $e_{i+1} = f(x_i + 2, y_i - 3/2) = e_i + 2 \cdot x_i - 2 \cdot y_i + 5$
- Si $e_i < 0$ (M à l'intérieur) alors le pixel à afficher est E
  - Le prochain point milieu sera $(x_i + 2, y_i - 1/2)$
  - $e_{i+1} = f(x_i + 2, y_i - 1/2) = e_i + 2 \cdot x_i + 3$



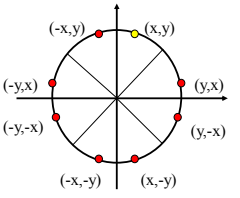
Algorithmes de Bresenham pour le tracé de cercles

```
CerclePointMilieu(int r)                    // le center du cercle est à l'origine
{
  int x  = 0 ;
  int y = r
  double e = 5.0/4.0 – r ;                  // point de départ(0,r) => point intermédiaire (1,r-1/2)
  AfficherPixel(x, y) ;
  While (y > x) {                           // octant 2
        if(e < 0)                           // pixel suivant E
              e = e + 2 . x + 3 ;
        else{                               // pixel suivant SE
              e = e + 2 . x – 2 . y  + 5 ;
              y = y – 1 ;
        }
        AfficherPixel(x, y) ;
        x = x + 1 ;
  }
}
```

Algorithmes de Bresenham pour le tracé de cercles

- La généralisation aux huit octants se fait en remplaçant la proc édure AfficherPixel() de l'algorithme par la procédure suivante :

```
AfficherPixelCercle(x, y)
{
      AfficherPixel(x, y) ;
      AfficherPixel(x, -y) ;
      AfficherPixel(-x, y) ;
      AfficherPixel(-x, -y) ;
      AfficherPixel(y, x) ;
      AfficherPixel(y, -x) ;
      AfficherPixel(-y, x) ;
      AfficherPixel(-y, x-) ;

}
```



- Optimisation
  - Au départ
    - $e_0 = 5.0/4.0 - r = 1.25 - r$
  - Si $e_i < 0$ alors on va à « E »
    - $e_{i+1} = e_i + 2 . x_i + 3$;
  - Sinon $e_i > 0$ alors on va à « SE »
    - $e_{i+1} = e_i + 2 . x_i - 2 . y_i + 5$;

Algorithmes de Bresenham pour le tracé de cercles

- Si on pose $h_i = e_i - 0.25$, on peut remplacer l'initialisation par:

$$h_0 = 1 - r$$

- On devrait donc aussi modifier la comparaison « if(e < 0) » par :

$$\text{if } (h_i < -0.25)\{$$

- Cependant, on remarque que *hi* sera toujours un entier donc :

$$h_i < -0.25 \Rightarrow h_i < 0$$

- On peut donc seulement changer l'initialisation et remplacer e_i par h_i ailleurs

Algorithmes de Bresenham pour le tracé de cercles

```
CerclePointMilieu(int r)                    // le center du cercle est à l'origine
{
  int x  = 0 ;
  int y = r
  double h = 1 – r ;                        // point de départ(0,r) => point intermédiaire (1,r-1/2)
  AfficherPixelCercle(x, y) ;
  While (y > x) {                           // octant 2
        if(h < 0)                           // pixel suivant E
              h = h + 2 . x + 3 ;
        else{                               // pixel suivant SE
              h = h + 2 . x – 2 . y  + 5 ;
              y = y – 1 ;
        }
        AfficherPixelCercle(x, y) ;
        x = x + 1 ;
  }
}
```

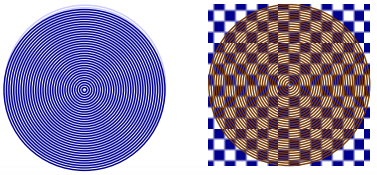
Anticrenelage (antialiasing)

Les algorithmes de tracé présentés précédemment ont un problème commun de précision.

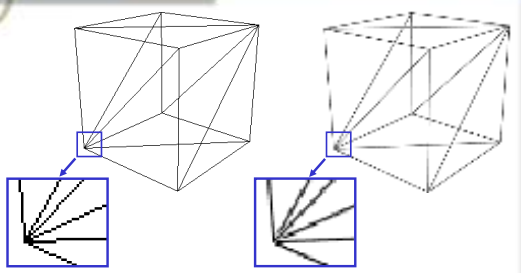
La discrétisation implique une perte d'informations qui se traduit par des contours en forme de marches d'escaliers.

Cette perte d'informations est bien connu en traitement du signal (analyse de Fourier) et le terme aliasing qui en est issu est utilisé en graphisme pour caractériser ces artefacts dans les images produites : Moirées

L'antialiasing consiste alors à corriger (diminuer) ces effets.



Anticrenelage (antialiasing)



Sans correction Avec correction

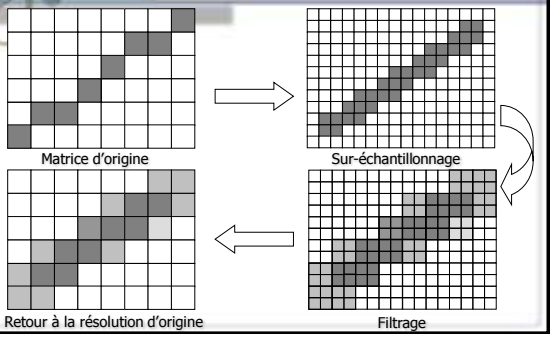
Anticrenelage (antialiasing)

- Les deux principales solutions :
  - Sur-échantillonnage de l'image
  - Algorithmes de tracé de segments corrigés

Sur-échantillonnage de l'image

- Cette méthode consiste à calculer, dans un premier temps, une image virtuelle à une résolution plus importante que celle de l'image désirée (en mémoire mais non à l'écran), en pratique X3, X5 ou X7, puis de réduire cette image par filtrage :
  - L'intensité de chaque pixel de l'image finale est déterminée par moyennage pondéré ou par filtrage (filtre passe-bas par convolution) de plusieurs pixels de l'image sur-échantillonnée.

Sur-échantillonnage de l'image



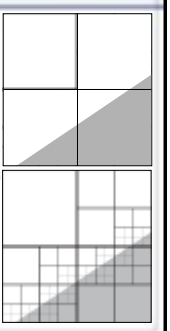
Sur-échantillonnage de l'image

- Paramètres
  - Echantillonnage
    - Type
      - Déterministe
        - Régulier
        - Adaptatif
      - Aléatoire
        - Uniforme
        - Disque de Poisson
        - Jittering
    - Résolution
  - Filtre
    - Forme du filtre
    - Taille du filtre

Sur-échantillonnage de l'image

Echantillonnage Déterministe

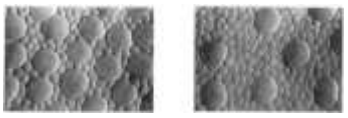
- Suréchantillonnage
  - Régulier
    - images avec des résolutions beaucoup plus grandes celles nécessaire,
      - le temps de calcul augmente (au moins d'un facteur n2 si n est le facteur du suréchantillonnage).
    - le suréchantillonnage est loin d'être nécessaire partout
  - Adaptatif : raffinement progressif de certaines zones particulières du pixel
    - Crtiéres
      - on se trouve sur un contour.
      - passage d'un objet à un autre.
      - auto-occlusion d'un objet complexe.
      - phénomène d'ombrage et de lumière
      - un des phénomènes précédents derriére un objet transparent.
      - etc.



Sur-échantillonnage de l'image

Echantillonnage aléatoire

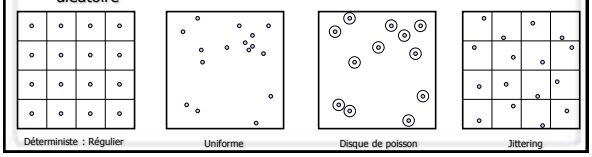
- l'incohérence d'un bruit est beaucoup moins choquante que la cohérence de l'aliasing
  - l'attention est d'abord attirée par les parties cohérentes d'une scène
  - les photo-récepteurs de l'oeil humain ne sont pas distribués de maniée uniforme : c'est l'une des raisons pour lesquelles l'oeil humain ne fait pas d'aliasing.



  - l'échantillonnage aléatoire
    - permet de transformer l'aliasing en bruit tout en perturbant relativement peu les parties basse fréquence de l'image.

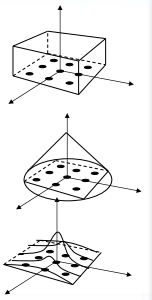
Sur-échantillonnage de l'image

- Uniforme
  - Le tirage de chaque point est complètement indépendant des autres
    - Possibilité de tirrage proche
- Disque de poisson : inspiré de la répartition des photorécepteurs
  - Chaque point est placé de façon à ce qu'il soit suffisamment loin de tous les points déjà acceptés.
- Jittering
  - consiste à partir d'une grille régulière et à la perturber avec un bruit aléatoire



Sur-échantillonnage de l'image

- Forme du Filtres
  - Boite
    - Moyenne des pixel recouverts
  - Cône
    - Moyenne pondérées en fonction de la distance du centre
  - Fonction Gaussienne
    - Moyenne pondérées suivant la fonction gaussienne



Filtre Gaussien

- Fonction Gaussienne avec différentes tailles

1	2	1
2	4	2
1	2	1

3x3

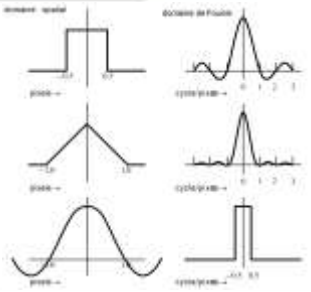
1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

5x5

1	2	3	4	3	2	1
2	4	6	8	6	4	2
3	6	9	12	9	6	3
4	8	12	16	12	8	4
3	6	9	12	9	6	3
2	4	6	8	6	4	2
1	2	3	4	3	2	1

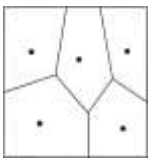
7x7

xx



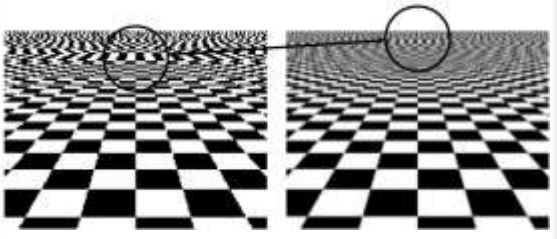
Sur-échantillonnage de l'image

- Filtrage des images Suréchantillonnées aléatoirement
  - Méthode : Monte-Carlo pondérée
    - Subdivisiser l'espace (du pixel) avec un diagramme de Voronoï
    - Pondération par l' aire du point échantillonné



Sur-échantillonnage de l'image

- Décalage du problème vers les plus hautes fréquences

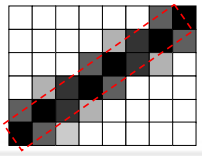


Algorithmes de tracé de segments corrigés

- Une solution, spécifique aux algorithmes de tracé, consiste à réduire l'effet de *crénelage* en allumant non pas un seul pixel à chaque itération, mais plusieurs, avec des intensités différentes suivant la proximité du pixel considéré et de la primitive.
- Les approches diffèrent sur les critères mis en oeuvre pour déterminer l'intensité d'un pixel.

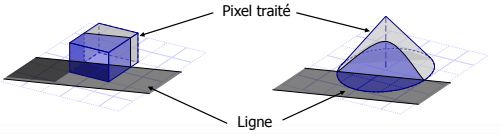
Algorithmes de tracé de segments corrigés

- Une première approche consiste à considérer un segment d'épaisseur non nulle (1 par exemple) et d'allumer chaque pixel couvert par le segment en fonction de l'aire de recouvrement du pixel.



Algorithmes de tracé de segments corrigés

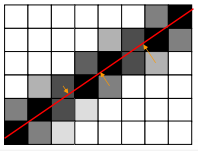
- Surface d'intersection entre la primitive et le pixel
  - Approche non-pondérée
    - Le calcul du niveau de gris est égal au volume d'intersession entre le pixel (forme carrée unitaire) et la droite théorique
  - Approche pondérée
    - Le calcul du niveau de gris intègre cette fois-ci la distance entre le ce ntre du pixel (forme circulaire unitaire) et la droite théorique



Algorithmes de tracé de segments corrigés

Algorithme Gupta-Sproull

- Cette approche considère uniquement la distance entre les pixel et le segment de droite
  - À chaque itération de l'algorithme de Bresenham on active le pixel en cours ainsi que le voisinage : généralement 3x3
  - Le niveau de gris est proportionnel à la distance entre le pixel activé et la droite (y compris le pixel calculé par l'algorithme de Bresenham)



Algorithmes de tracé de segments corrigés

Voisinage 3x3

Pixel voisin

$D = \left| \frac{a.x_i + b.y_i + c}{\sqrt{a^2 + b^2}} \right|$

D

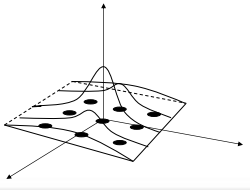
Pixel en cours

Droite d'équation :

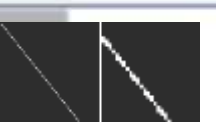
$a.x + b.y + c$

Algorithmes de tracé de segments corrigés

- Il est facile de modifier l'algorithme de trace de segments vu précédemme nt de manière à prendre en compte cet aspect.
- La question qui se pose ensuite est comment passer d'une distance à une intensité.
  - Classiquement, on applique une fonction de filtrage Gaussienne



Anticrénelage (résultats)



Tracé sans antialiasing

Tracé avec antialiasing
Méthode basée sur le moyennage (Gaussien) d'une image de plus grande résolution

Tracé avec antialiasing
Méthode basée sur les distances des pixels au segment

Le découpage (clipping)

- Clipping écran
  - Traitement permettant de réduire le dessin d'un objet graphique à une région de l'écran
- Cette région est classiquement un rectangle mais peut être de toute autre forme
  - Carré : Écran
  - Trapèze : Pare-brise/Rétroviseur
  - Circulaire : Lunette/Jumelle
  - …
- Traitement de base de l'Informatique Graphique :
  - Économiser des opérations inutiles
- Extension
  - en 3D : éliminer les calculs en dehors de l'espace visible

Le découpage (clipping)

Trois approches :

- AVANT LA CONVERSION
  - calcul analytique des intersections avec la frontière de la région de découpage
  - abordable pour des formes simples (p.e. algorithme Cohen-Sutherland pour segment découpé par région rectangulaire)
- PENDANT LA CONVERSION
  - on construit un masque de la région puis on vérifie que chaque pixel devant être « allumé » y figure
  - permet de représenter des régions de découpage arbitrairement compl exes
- APRÈS LA CONVERSION
  - on convertit tout l'objet virtuel dans un espace mémoire temporaire puis on copie le sous-ensemble d'intérêt (clôture)
  - efficace si on fait face à un objet complexe coûteux à convertir et qu'o n déplace la région de découpage de façon interactive

Le découpage (clipping)

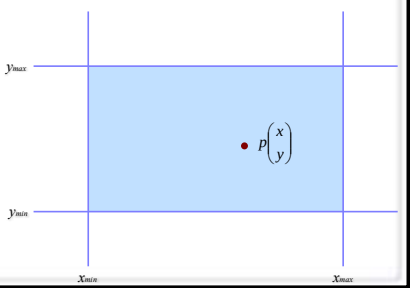
Familles d'algorithmes

- Clipping Point / Fenêtre
- Clipping Segment / Fenêtre
  - Cohen-Sutherland
  - Liang-Barsky
- Clipping Polygon / Fenêtre
  - Sutherland-Hodgeman

Le découpage (Clipping)

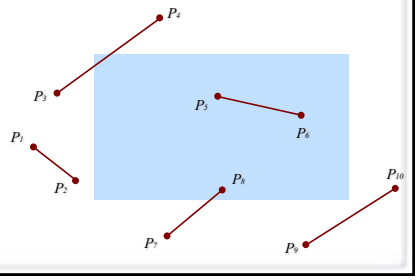
- Clipping de points : Point / Fenêtre
  - Test :

$$x_{\min} < x < x_{\max}$$
$$\&$$
$$y_{\min} < y < y_{\max}$$



Le découpage (Clipping)

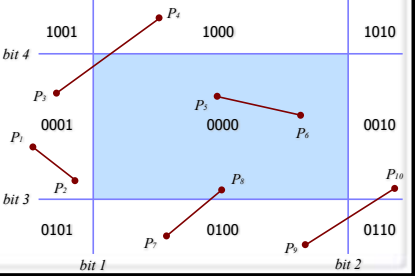
- Clipping de segments : Segment / Fenêtre
  - Déterminer les portions de segments à l'intérieur de la fenêtre



Le découpage (Clipping)

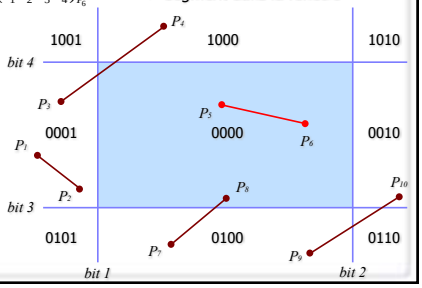
- Algorithme de Cohen et Sutherland
  - Traiter les paires de points relativement aux codes des zones


Ivan Sutherland



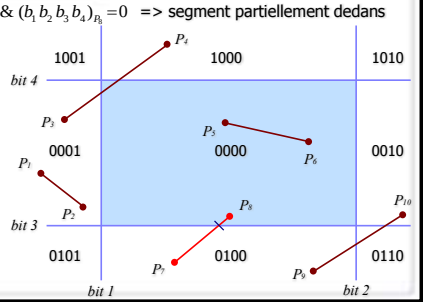
Le découpage (Clipping)

- Cas 1
  - $(b_1 b_2 b_3 b_4)_{P_5} = (b_1 b_2 b_3 b_4)_{P_6} = 0$ => segment dans la fenêtre



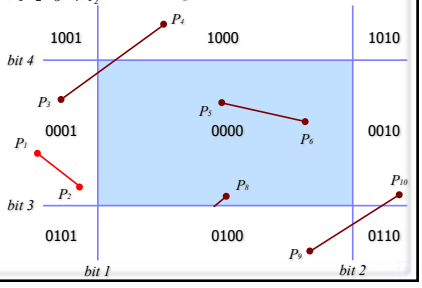
Le découpage (Clipping)

- Cas 2
  - $(b_1 b_2 b_3 b_4)_{P_7} \neq 0$ & $(b_1 b_2 b_3 b_4)_{P_8} = 0$ => segment partiellement dedans



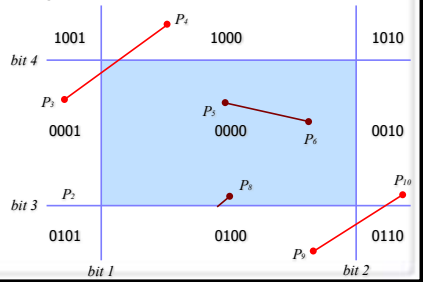
Le découpage (Clipping)

- Cas 3
  - $(b_1 b_2 b_3 b_4)_{P_1}$ & $(b_1 b_2 b_3 b_4)_{P_2} \neq 0$ => segment entièrement dehors



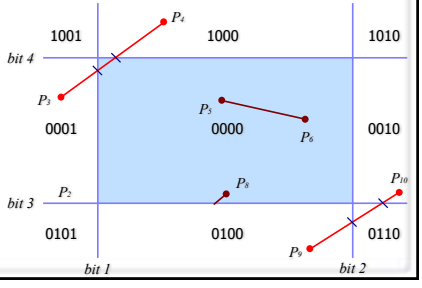
Le découpage (Clipping)

- Cas 4
  - $(b_1 b_2 b_3 b_4)$ & $(b_1 b_2 b_3 b_4) = 0$ => extrémités dans des zones différentes



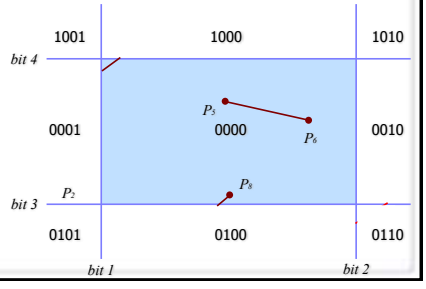
Le découpage (Clipping)

- Cas 4
  - calculs d'intersection



Le découpage (Clipping)

- Réitérer les tests avec les parties restantes



Le découpage (Clipping)

- Algorithme de Cohen & Sutherland
  - Représentation paramétrique d'un segment de droite $[p_1, p_2]$

$$p(t) = p_1 + (p_2 - p_1)t = (1-t)p_1 + t\,p_2, t \in [0..1]$$
$$x(t) = (1-t)x_1 + t\,x_2 ; y(t) = (1-t)y_1 + t\,y_2$$

  - Frontière supérieure de la fenêtre $(x(t'), y(t')) = (t', y_{max}), t' \in \mathbb{R}$
  - Intersection : $t / p(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} (1-t)x_1 + t\,x_2 \\ (1-t)y_1 + t\,y_2 \end{pmatrix}$ & $p(t) = \begin{pmatrix} x(t) \\ y_{max} \end{pmatrix}$
  - Résultat : $(1-t)y_1 + t\,y_2 = y_{max}$

$$t = \frac{y_{max} - y_1}{y_2 - y_1}$$

Le découpage (clipping)

Algorithme de Cohen et Sutherland voir sites :

http://www3.cc.gatech.edu/classes/AY2003/cs4451a_fall/ClippingApplets%20Folder/Cohen-Sutherland/index.html

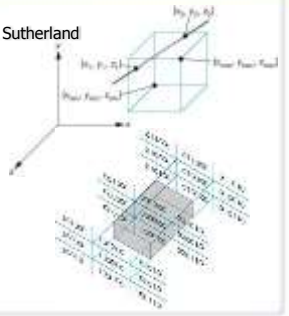
Ou encore :

http://www.cs.princeton.edu/~min/cs426/jar/clip.html

Le découpage (clipping)

Clipping 3D

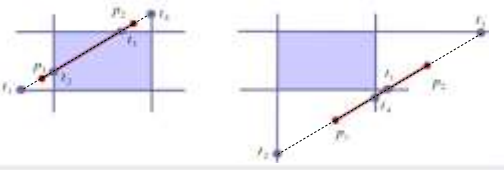
- Extension de algorithme de Cohen & Sutherland
- 6 Bits pour le outcode
  - b4 : z > zmax
  - b5 : z < zmin
- Nécessite d'autres précalculs



Le découpage (clipping)

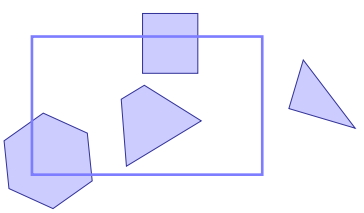
Clipping de segments : Algorithme de Liang et Barsky

- Intersection de la droite (p1 p2) avec les frontières dans l'ordre :
  - Bas puis Gauche puis Haut puis Droite
- Tri des paramètres et comparaison avec l'ordre attendu
  - si ($t_1 < t_2 < t_3 < t_4$) alors segment à conserver
  - sinon segment rejeté



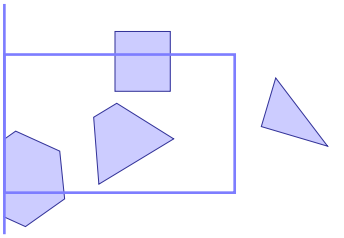
Le découpage (Clipping)

- Clipping de polygones
  - Déterminer les parties de polygones à l'intérieur de la fenêtre



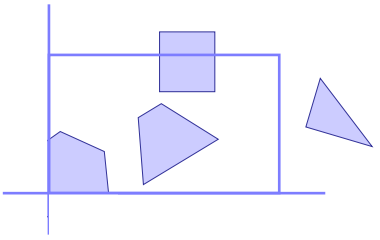
Le découpage (Clipping)

- Algorithme de Sutherland et Hodgeman
  - Tester relativement à chaque frontière de la fenêtre



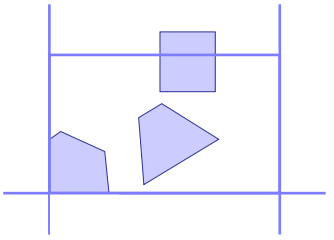
Le découpage (Clipping)

- Algorithme de Sutherland et Hodgeman
  - Tester relativement à chaque frontière de la fenêtre



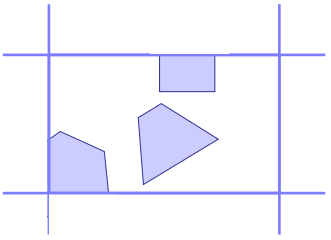
Le découpage (Clipping)

- Algorithme de Sutherland et Hodgeman
  - Tester relativement à chaque frontière de la fenêtre



Le découpage (Clipping)

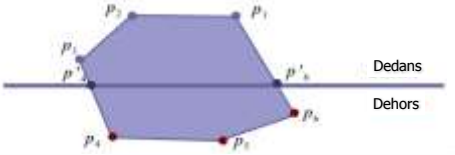
- Algorithme de Sutherland et Hodgeman
  - Tester relativement à chaque frontière de la fenêtre



Le découpage (Clipping)

Algorithme de Sutherland et Hodgeman

- Pour chaque frontière et chaque polygone :
  - tester l'appartenance sur les deux extrémités des segments
  - si cas 2 ou cas 4 calculer l'intersection avec la frontière
  - insérer les nouveaux points calculés
  - supprimer les points en dehors de la fenêtre



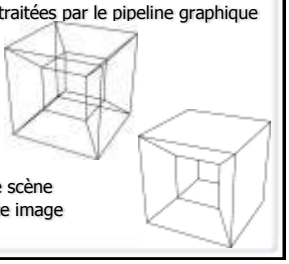
Le découpage (clipping)

- Algorithme de Sutherland et Hodgeman voir site

http://www3.cc.gatech.edu/classes/AY2003/cs4451a_fall/ClippingApplets%20Folder/Cohen-Sutherland/index.html

Élimination des parties cachées

- Déterminer les lignes, arêtes, surfaces et volumes visibles par un observateur
  - Cohérence visuelle de la scène
  - Réduire le nombre de primitives traitées par le pipeline graphique
- Algorithme de base
  - Équivalent au tri => O(n log n)
- Deux familles d'algorithme :
  - Algorithmes objet : dans l'espace scène
  - Algorithmes image : dans l'espace image



Élimination des parties cachées

- Dans l'espace scène : le masquage se fait sur le modèle de données physiques => Calcul en coordonnées du monde. le masquage des parties cachées est valable quelle que soit l'échelle du dessin.
- Caractéristiques :
  - Test objet par objet ($O(n^2)$)
  - Précision dépend de la résolution des objets
  - Techniques relativement complexes à mettre en œuvre
- Méthodes :
  - Élimination de faces arrières (Backface Culling)
  - Tri des primitives selon leur éloignement
    - Algorithme du peintre
    - Algorithme de tri par arbre binaire (BSP Trees)

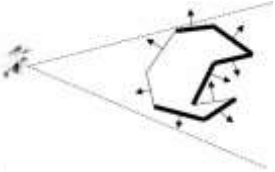
Élimination des parties cachées

- Dans l'espace image : le masquage se fait sur les pixels écran => Calcul en coordonnées fenêtre
- Caractéristiques :
  - Test de visibilité en chaque pixel ($O(n \times p)$)
  - Précision dépend de la résolution de l'espace image
  - Recalculer à chaque changement caméra/scène
  - Techniques relativement simples
- Méthodes :
  - Subdivision récursive de l'image (Warnock)
  - Test de profondeur
    - Ligne de balayage :
      - Scan-line-Watkins
      - Scan-line-Zbuffer
    - Tampon de profondeur - Zbuffer
  - Lancer de rayons (Whitted)

Algorithmes objet

Élimination de faces arrières (Backface Culling)

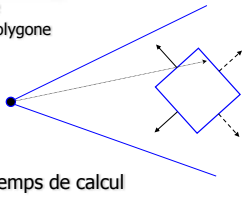
- Backface Culling : garder les parties de la surface pour lesquelles la normale pointe dans la direction opposée au point d'observation.
- Ces parties, vues du point d'observation, sont occultées par l'objet lui-même.



- Conditions : La suppression des faces arrières suffit à éliminer les parties cachées :
  - si l'objet est seul dans la scène
    - il ne faut pas qu'un objet puisse en masquer un autre
  - si l'objet est convexe
    - dans un objet concave, des faces avant peuvent être masquées par d'autres

Élimination de faces arrières (Backface Culling)

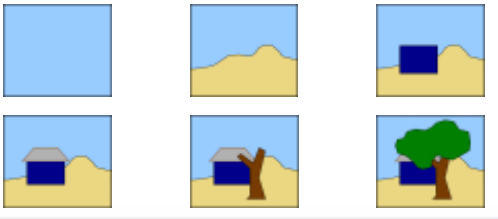
- Calcul : produit scalaire
  - (Œil-Face) . Normale
    - < 0 : on garde le polygone
    - > 0 : on l'élimine



- Économise 50 % du temps de calcul
  - En moyenne
- Faible coût par polygone
- Étape préliminaire pour les autres algorithmes
- Suffisant pour un seul objet convexe

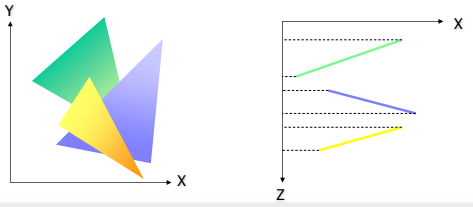
Algorithme du Peintre

- Dit aussi « Algorithme de Tri par la Profondeur » (Newell, Newell & Sancha 1972)
  - Peindre les facettes polygonales dans la mémoire vidéo suivant un ordre de distance décroissante au point d'observation.



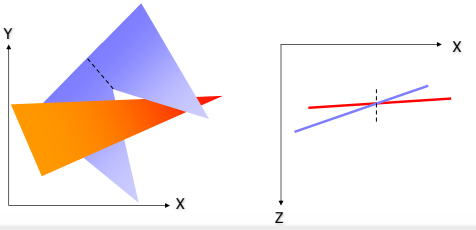
Algorithme du Peintre

- Principe :
  - Tracé de polygones bien orientés, du plus éloigné au plus proche
  - Tri des polygones
- Cas trivial (non chevauchement)



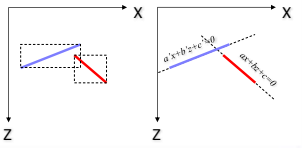
Algorithme du Peintre

- Principe :
  - Tracé des polygones « front » du plus éloigné au plus proche
  - Cas ambigu (chevauchement) : Découpage de polygones



Algorithme du Peintre

- Algorithme complet
  - Trier les polygones en fonction de leur plus grande coordonnée en *z*
    - *z* distance à la caméra suivant la direction de visée
  - Si deux polygones ont des étendues en *z* qui se recouvrent :
    - On test :
      - Si les boites englobantes de leurs projections sont séparées
      - Si leurs projections ne sont pas séparées : calcul d'intersection de segment de droite
    - Si rien ne marche, on coupe l'un des polygones par le plan de l'autre
      - Calcul d'intersection de plans
- Caractéristiques
  - Le plus intuitif des algorithmes
  - Coût en mémoire :
    - Affichage direct à l'écran : O(*p*)
    - Il faut trier les polygones : O(*n*log*n*)
  - Temps de calcul :
    - On affiche toute la scène
    - Efficace surtout sur des petites scènes

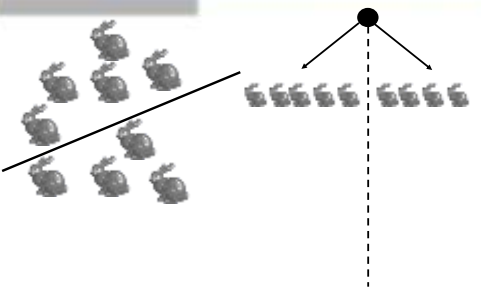


Partition binaire de l'espace par BSP Trees

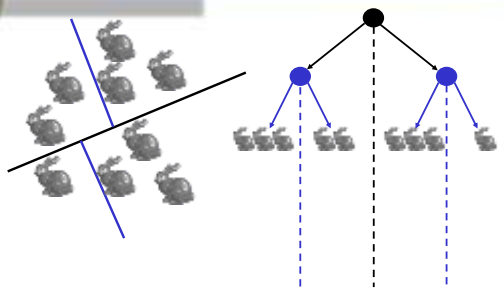
- BSP-tree (Binary Space Partition) : arbre binaire utilisé pour trier des primitives dans l'espace
- Principe (Schumaker 1969, Fluch 1980) :
  - Choix d'une arête de référence
    - Partage de l'espace en 2 demi-espaces (in et out)
    - Répartition des objet (& primitives) selon le demi-espace occupé
    - Les objet (& primitives) à cheval sont découpées selon le plan de référence
  - Réitération dans les deux demi-espaces résultants

Partition binaire de l'espace par BSP Trees

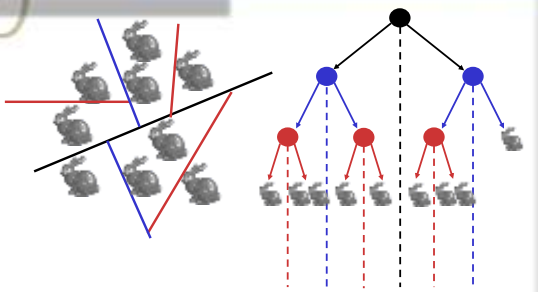
Partition binaire de l'espace par BSP Trees



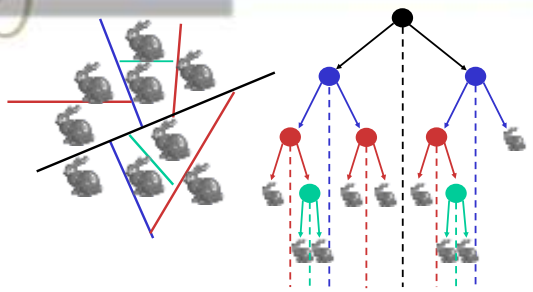
Partition binaire de l'espace par BSP Trees



Partition binaire de l'espace par BSP Trees



Partition binaire de l'espace par BSP Trees



Partition binaire de l'espace par BSP Trees

- Dans le cas d'une primitives ou objet partagé
  - Diviser la primitive en deux en fonction du plan séparateur
  - Répartir l'objet de part et d'autre du plan séparateur

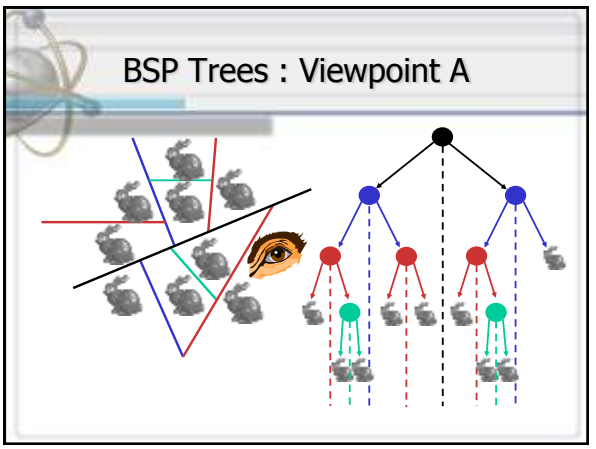
Parcours du BSP Trees

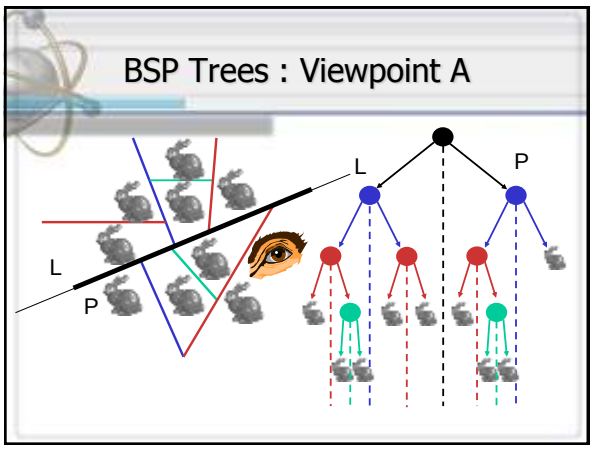
- Utilisation du BSP-tree :
  - Parcours récursif de l'arbre à partir de la racine :
    - Déterminer le demi-espace où se situe l'observateur, par rapport au nœud courant puis (Proche ou Loin) :
      - Traiter le sous-arbre correspondant au demi-espace où n'est pas situé l'observateur (Loin)
      - Dessiner la primitive correspondant au nœud courant
      - Traiter le sous-arbre correspondant au demi-espace où est situé l'observateur (Proche)

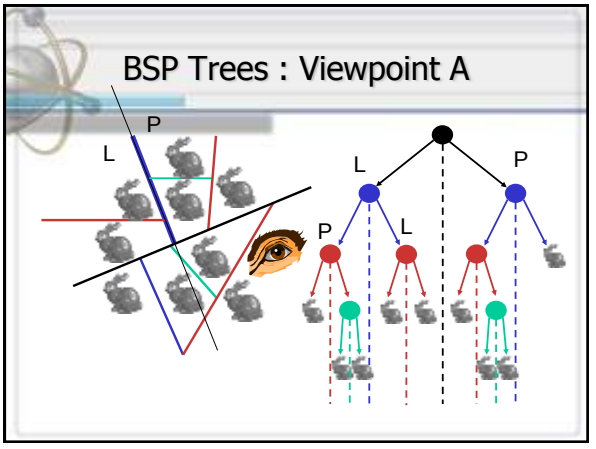
Parcours du BSP Trees

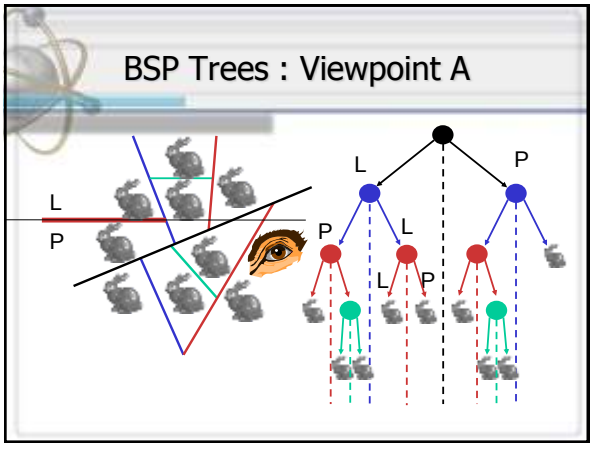
Algorithme

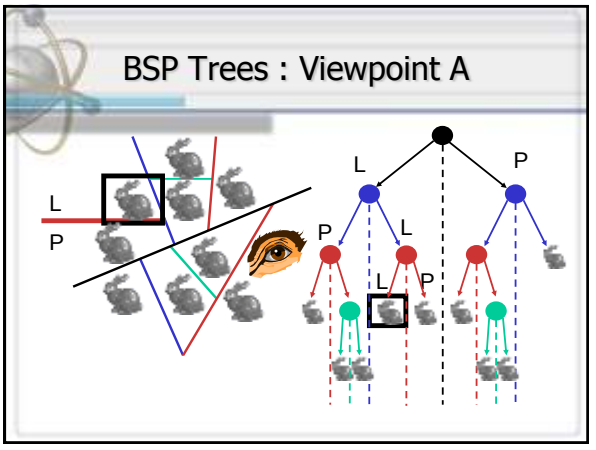
```
renderBSP(BSPtree *T)
    BSPtree *Proche, *Loin;
    if (oeil à gauche du T->plane)
            Proche = T->gauche; Loin = T->droite;
    else
            Proche = T->droite; Loin = T->gauche;

    renderBSP(Loin);

    if (T est un noeud fils)
            renderObject(T)

 renderBSP(Proche);
```

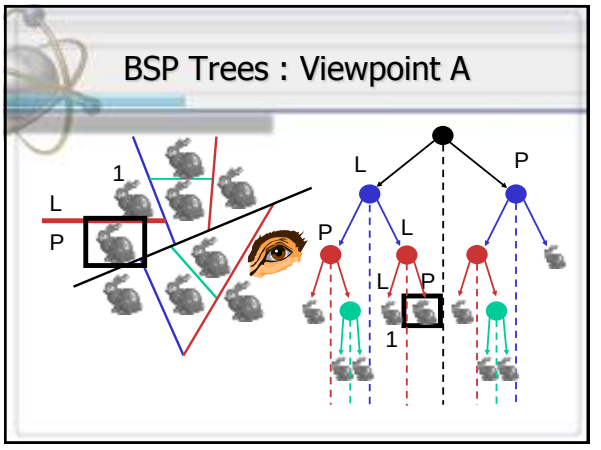

BSP Trees : Viewpoint A

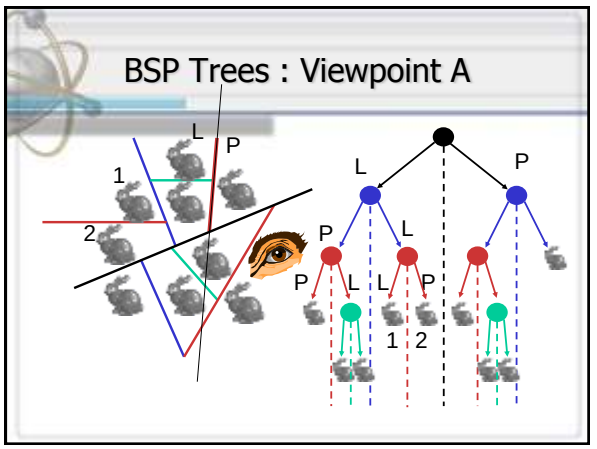

BSP Trees : Viewpoint A
L
P
L
P

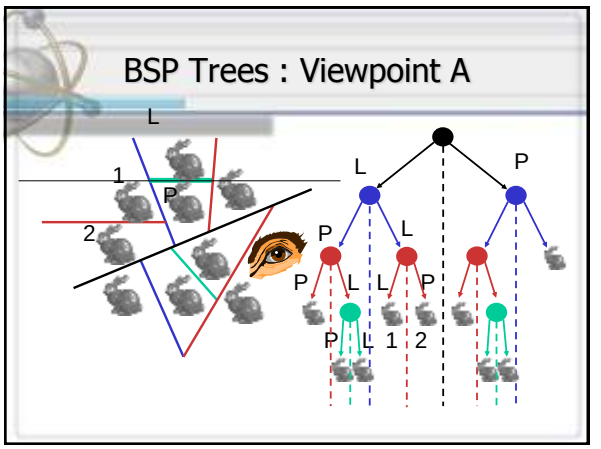

BSP Trees : Viewpoint A
P
L
L
P
P
L

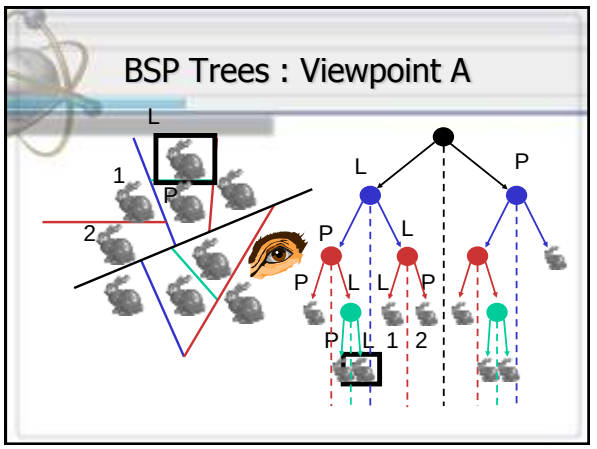

BSP Trees : Viewpoint A
L
P
L
P
P
L
L
P

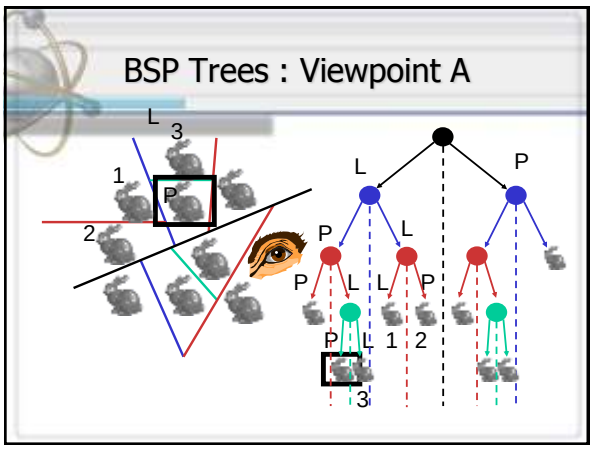

BSP Trees : Viewpoint A
L
P
L
P
P
L
L
P

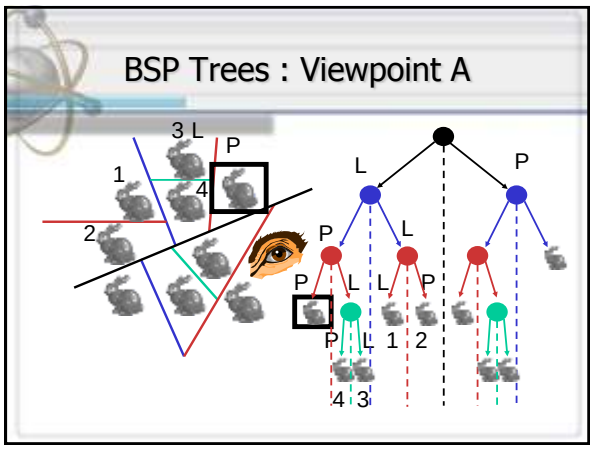

BSP Trees : Viewpoint A
1
L
P
L
P
P
L
L
P
1

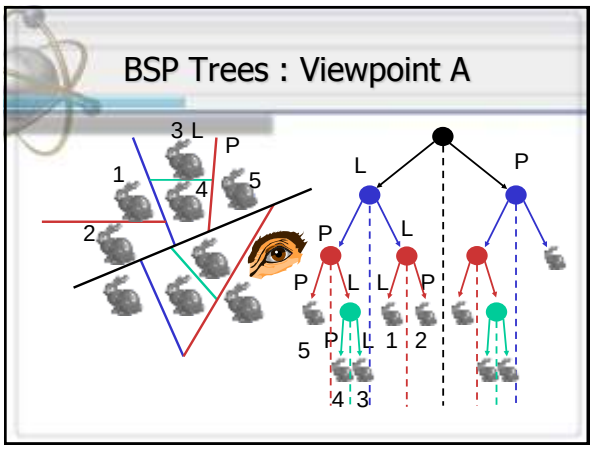

BSP Trees : Viewpoint A
L
P
1
2
L
P
P
L
L
P
P
L
L
P
1
2


BSP Trees : Viewpoint A
L
1
P
2
L
P
P
L
L
P
P
L
L
P
1
2

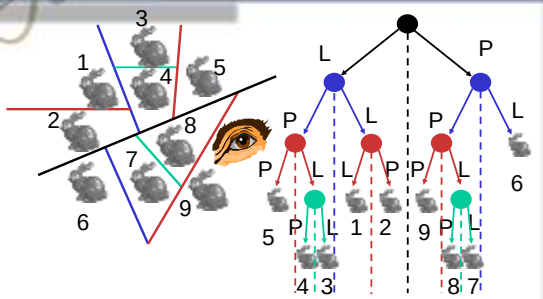

BSP Trees : Viewpoint A
L
1
P
2
L
P
P
L
L
P
P
L
1
2


BSP Trees : Viewpoint A
L
3
1
P
2
L
P
P
L
L
P
P
L
1
2
3

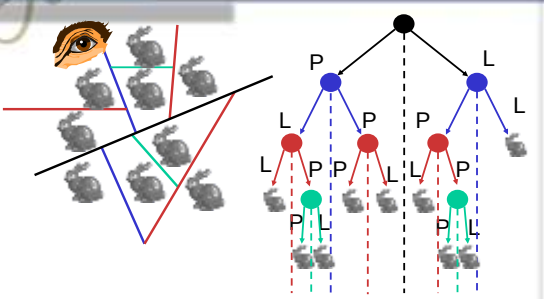

BSP Trees : Viewpoint A
3 L
P
1
4
2
L
P
P
L
L
P
P
L
1
2
4
3


BSP Trees : Viewpoint A
3 L
P
1
4
5
2
L
P
P
L
L
P
5
P
L
1
2
4
3

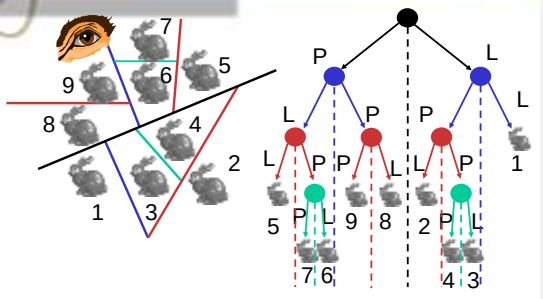
BSP Trees : Viewpoint A



BSP Trees : Viewpoint B



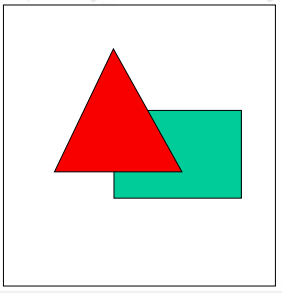
BSP Trees : Viewpoint B



Algorithmes image

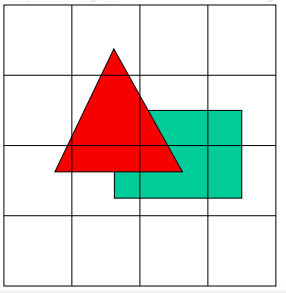
Subdivision récursive de l'image - Warnock

- Principe « diviser pour régner » : diviser l'image en quadtree



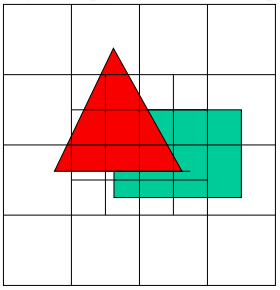
Subdivision récursive de l'image - Warnock

- Principe « diviser pour régner » : diviser l'image en quadtree



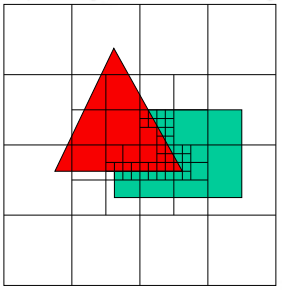
Subdivision récursive de l'image - Warnock

- Principe « diviser pour régner » : diviser l'image en quadtree



Subdivision récursive de l'image - Warnock

- Principe « diviser pour régner » : diviser l'image en quadtree

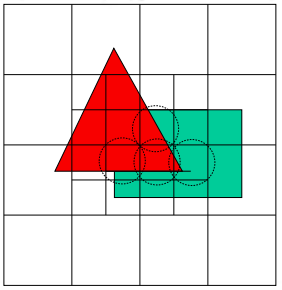


Subdivision récursive de l'image - Warnock

- Pour chaque quadrant :
  - Déterminer la liste des polygones potentiellement visibles
  - Tracer le quadrant de l'image directement dans les cas simples :
    - Rien dans le quadrant
      - Couleur du fond
    - Un seul polygone : contenu facile à dessiner avec la couleur du polygone
      - Le polygone couvrant le quadrant
        - Partiellement
        - Totalement

Subdivision récursive de l'image - Warnock

- Cas complexes : plusieurs polygones dans le quadrant

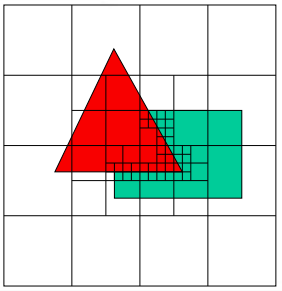


Subdivision récursive de l'image - Warnock

- Principe « diviser pour régner » : diviser l'image en quadtree
  - Cas complexes : plusieurs polygones dans le quadrant
    - Trier les polygones selon la profondeur
    - Si un polygone masque tous les autres :
      - Couvre tout le cadrant et est avant les autres (sans test complexe)
      - Tracer avec la couleur du polygone
    - Sinon subdiviser et itérer le processus
  - Arrêt quand le quadrant est plus petit qu'un pixel
    - Tracer avec la couleur du polygone en avant des autres

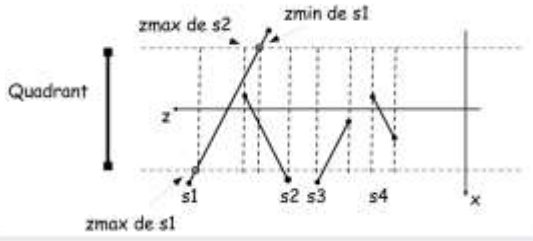
Subdivision récursive de l'image - Warnock

- Découpage récursif des quadrants contenant plusieurs polygones



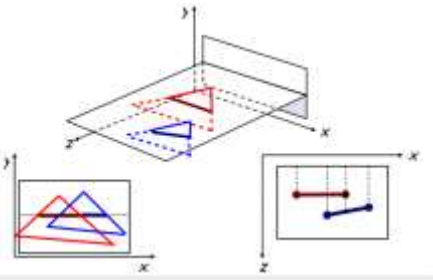
Subdivision récursive de l'image - Warnock

- Problème : plusieurs polygones dans le quadrant
  - S1 masque tous les autres polygones, mais…
    - Subdivision (masquage non détecté par les tests simples de profondeur)



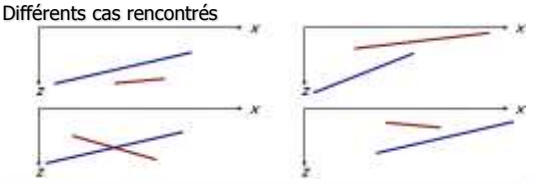
Algorithme de balayage (scan-line)-Watkins

- Principe : éliminer les lignes cachées, ligne par ligne



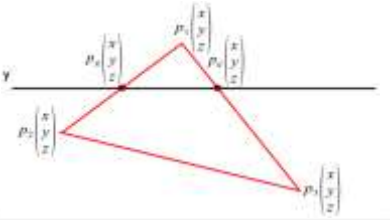
Algorithme de balayage (scan-line)-Watkins

- Pour une ligne de balayage donnée :
  - Obtention d'un segment par polygone
  - Détermination de la visibilité des segments
  - Affichage des segments visibles
- Différents cas rencontrés



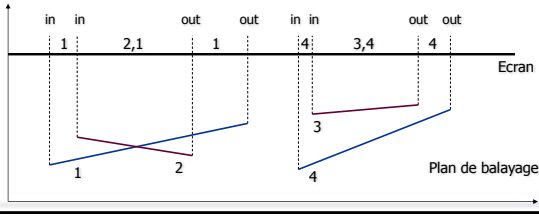
Algorithme de balayage (scan-line)-Watkins

- Obtention d'un segment par polygone
  - Intersection plan-segment
    - Pour un y fixé : on calcul les coordonnées x & z à partir de l'équation du segment traité de la primitive



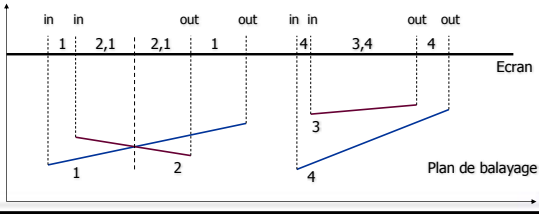
Algorithme de balayage (scan-line)-Watkins

- Détermination de la visibilité
  - Projection sur la ligne de balayage de l'écran
    - Déterminations des intervalles associés aux segments projetés [in, out]



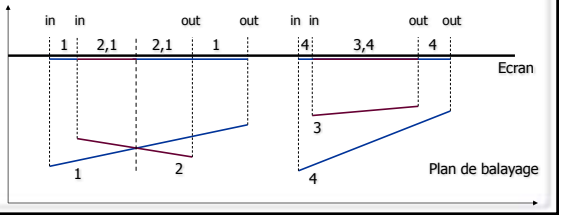
Algorithme de balayage (scan-line)-Watkins

- Détermination de la visibilité
  - Superposition d'intervalles -> plusieurs segments
    - Calcul d'intersection : intersection entre segments de droite
    - Si intersection -> ajout d'un point sur la ligne de balayage



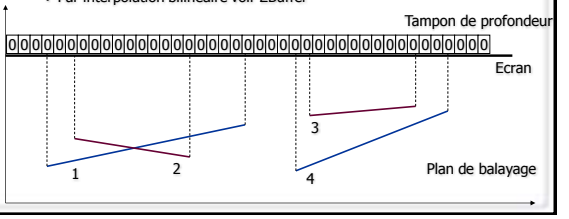
Algorithme de balayage (scan-line)-Watkins

- Affichage des segments visibles
  - Test de profondeur
  - Affichage de la couleur du segment traité



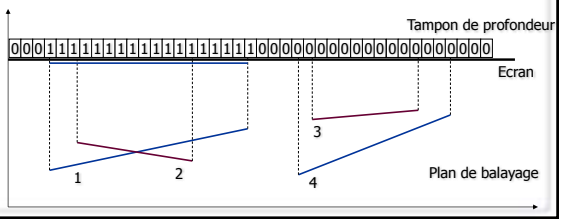
Algorithme de balayage : Scan-line-Zbuffer

- Détermination de la visibilité
  - Tampon de profondeur + mémoire d'image pour la ligne de balayage
    - Pour chaque élément traité on calcul sa profondeur
      - Par interpolation bilinéaire voir ZBuffer



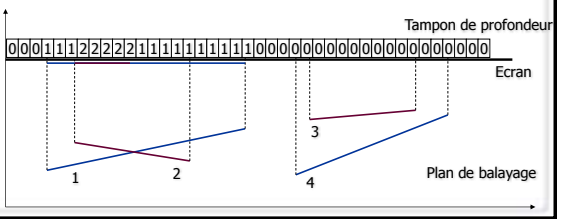
Algorithme de balayage : Scan-line-Zbuffer

- Détermination de la visibilité
  - Prise en compte du segment 1



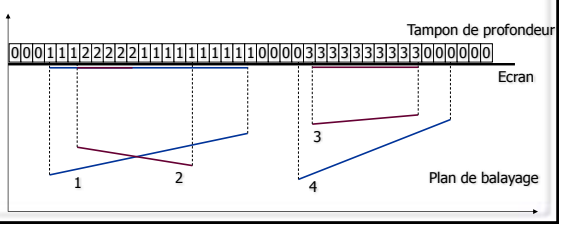
Algorithme de balayage : Scan-line-Zbuffer

- Détermination de la visibilité
  - Prise en compte du segment 2



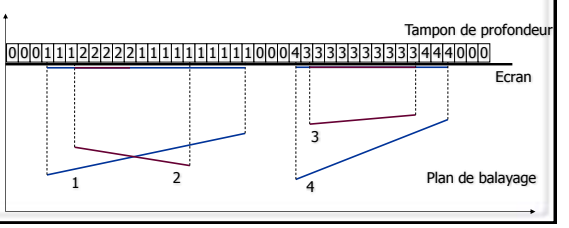
Algorithme de balayage : Scan-line-Zbuffer

- Détermination de la visibilité
  - Prise en compte du segment 3



Algorithme de balayage : Scan-line-Zbuffer

- Détermination de la visibilité
  - Prise en compte du segment 4



Tampon de profondeur – Z-buffer

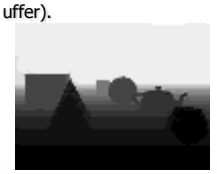

Edwin Catmull

- Zbuffer (Catmull-1974) :
  - Le calcul de l'image est effectué séquentiellement en traitant les objets extraits de la scène (classiquement des facettes) les uns à la suite des autres pour en extraire l'ensemble des pixels qui les représentent.
  - Au cours du processus de traitement d'un objet, chaque pixel calculé voit sa profondeur comparée à celle déjà calculée en cette place pour pouvoir n'afficher finalement pour chaque pixel de l'image que la facette la moins profonde présente en ce pixel (vis à vis de l'observateur).

    -> Il ne s'agit pas d'un tri, mais un calcul de maximum pour chaque pixel vis à vis de l'ensemble des objets présents en ce pixel.

Tampon de profondeur – Z-buffer

- On définit deux zones mémoire:
  - un tampon couleur destiné à contenir la couleur de chacun des pix els de l'image (initialisé à la couleur de fond souhaitée),
  - un tampon profondeur destiné à contenir la profondeur écran max imum de chacun des pixels de l'image (initialisée à -∞, utilisée co mme valeur courante de comparaison en cours d'exécution du Z-B uffer).

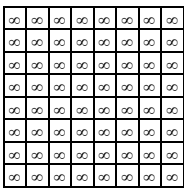



Tampon de profondeur – Z-buffer

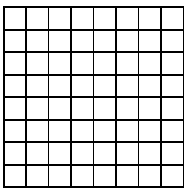
- Initialiser le Zbuffer à -∞
- Initialiser le buffer de couleur avec couleur du fond

Tampon de profondeur – Z-buffer

Z-buffer



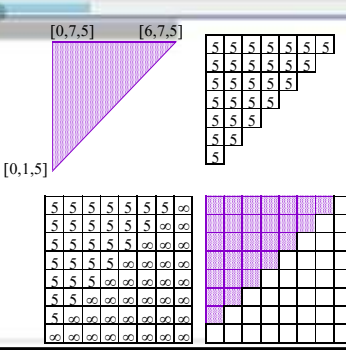
Buffer couleur



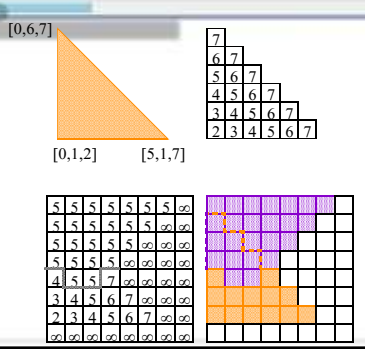
Tampon de profondeur – Z-buffer

- Pour chaque polygone
  - Pour chauque sommet
    - P1 = projection (sommet 1) sur l'écran
    - P2 = projection (sommet 2) sur l'écran
    - P3 = projection (sommet 3) sur l'écran
  - Pour chaque pixel(x,y) du polygone projeté
    - Déterminer la profondeur z du point correspondant du polygone par interpolation bilinéaire
    - Si z < Zbuffer(x,y) alors
      - Zbuffer(x,y) = z
      - FrameBuffer(x,y) = couleur du point
        - Application du modèle d'ombrage

Tampon de profondeur – Z-buffer

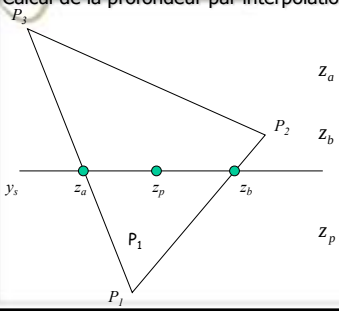


Tampon de profondeur – Z-buffer



Tampon de profondeur – Z-buffer

- Calcul de la profondeur par interpolation



$$z_a = z_1 + (z_3 - z_1)\frac{(y_1 - y_s)}{(y_1 - y_3)}$$

$$z_b = z_1 + (z_2 - z_1)\frac{(y_1 - y_s)}{(y_1 - y_2)}$$

$$z_p = z_a + (z_b - z_a)\frac{(x_a - x_p)}{(x_a - x_b)}$$

Tampon de profondeur – Z-Buffer

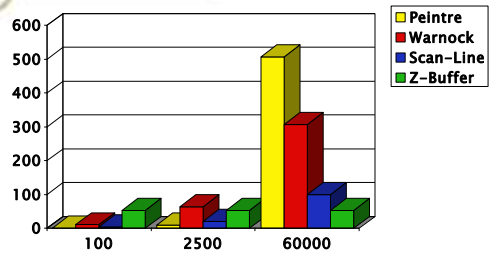
Algorithme

```
for all i,j {
 Depth[i,j] = MAX_DEPTH
 Image[i,j] = BACKGROUND_COLOUR
}
for all polygons P {
 for all pixels in P {
  if (Z_pixel < Depth[i,j]) {
   Image[i,j] = C_pixel
   Depth[i,j] = Z_pixel
  }
 }
}
```

Tampon de profondeur - Zbuffer

- Avantages
  - Simplicité de mise en œuvre
  - Pas de tri préalable
  - Rapidité (algorithme câblé, parallèlisable)
- Inconvénients
  - Traitement de tous les polygones, mêmes cachés
  - Taille mémoire nécessaire (de moins en moins)
  - Ne traite pas les effets particuliers
    - Inter-reflets, transparence, réfraction

Coûts comparés



Choix de l'algorithme

- L'algorithme idéal dépends :
  - De la complexité de la scène
  - Du matériel disponible
  - De ce qu'on veut faire en plus de l'affichage
- Z-Buffer en général :
  - Fourni gratuitement dans la librairie graphique
  - Pas de pré-traitement
  - Quelques effets de bord si on le connaît mal
- Scan-line pour les hackers :
  - Quand on ne peut pas utiliser la carte graphique
  - Faible coût mémoire
  - Lié à la fonction d'affichage, peut être très efficace

Remplissage

- Remplissage : Appelé également Rastérisation, cette étape consiste à remplir les polygones projetés (2D) pour donner un aspect plus réaliste au objets.
- Le remplissage se fait suivant le modèle d'Ombrage utilisé par le pipeline graphique (Gouraud, Phong, Lambert, etc.)
- Méthodes :
  - Test de l'appartenance d'un point à polygone
  - Test d'intersection
  - Algorithme de balayage de lignes
  - Remplissage de régions

Remplissage

Test d'appartenance d'un point à polygone

- Polygone convexe
  - Détermination des normales « extérieures »

$$\vec{N} = \overrightarrow{p_1 p_2} \times \overrightarrow{p_1 p_3}$$

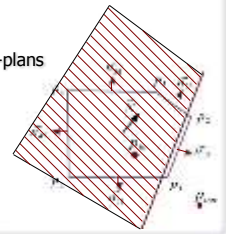
$$\vec{n}_{i,i+1} = \overrightarrow{p_i p_{i+1}} \times \vec{N}$$

  - Positionnement par rapport aux demi-plans
    - *p dans le polygone si*

$$\forall i \in [1,4], \vec{n}_{i,i-1} \cdot \overrightarrow{p_i p} \leq 0$$

    - *p hors du polygone si*

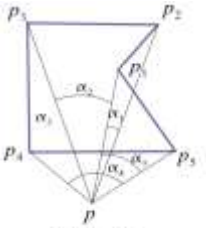
$$\exists i \in [1,4], \vec{n}_{i,i+1} \cdot \overrightarrow{p_i p} > 0$$



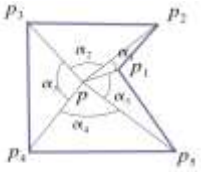
Remplissage

Test d'appartenance d'un point à polygone

- Test d'angles



$$\sum \alpha_i = 0$$

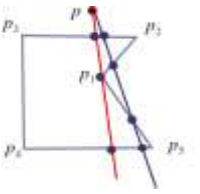


$$\sum \alpha_i = 2\pi$$

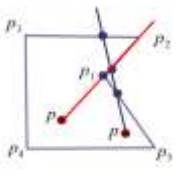
Remplissage

Test d'appartenance d'un point à un polygone

- Test du nombre de points d'intersection



4 points d'intersection



3 points d'intersection

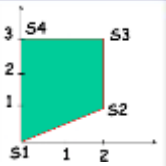
- Attention aux cas limites :
  - points extrémités de segments et segments tangents

Remplissage

Identification d'un polygone

- Polygone convexe
  - Si p1xp2 > 0 alors ccw

$$p1\begin{pmatrix} x \\ y \end{pmatrix} \times p2\begin{pmatrix} x \\ y \end{pmatrix} = p1_x \cdot p2_y - p1_y \cdot p2_x$$

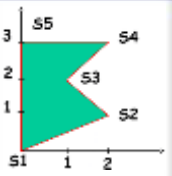


Sommets	Vecteurs	Produits vectoriels
S1	S4S1 ^ S1S2	[0 -3] x [2 1] = + 6
S2	S1S2 ^ S2S3	[2 1] x [0 2] = + 4
S3	S2S3 ^ S3S4	[0 2] x [-2 0] = + 4
S4	S3S4 ^ S4S1	[-2 -0] x [0 -3] = + 6

Remplissage

Identification d'un polygone

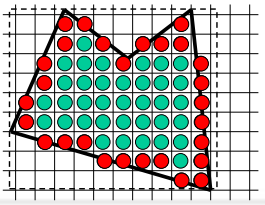
- Polygone concave
  - Changement de sens lors du parcours
    - S3 ré-entrant



Sommets	Vecteurs	Produits vectoriels
S1	S5S1 ^ S1S2	[0 -3] x [2 1] = + 6
S2	S1S2 ^ S2S3	[2 1] x [0 2] = + 4
S3	S2S3 ^ S3S4	[-1 1] x [1 1] = - 2
S4	S3S4 ^ S4S5	[1 1] x [-2 0] = + 2
S5	S4S5 ^ S5S1	[-2 -0] x [0 -3] = + 6

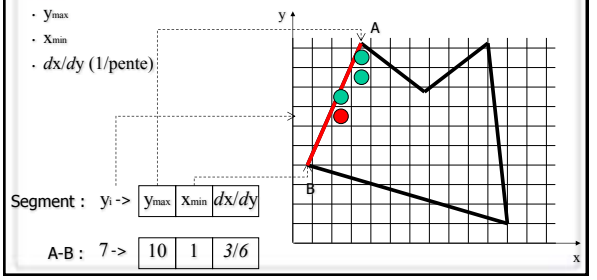
Remplissage

- Algorithme de balayage de lignes
  - Cet algorithme consiste à balayer les lignes horizontales appartenant au rectangle englobant le polygone à traiter, et à afficher les pixels intérieurs au polygone sur chaque ligne.
  - La recherche des intersections des lignes de balayage avec le polygone se fait en calculant l'approximation des segments à l'aide d'un algorithme de traçage et en mémorisant les point écrans correspondants.



Remplissage

- Pour chaque coté impliqué, et pour chaque ligne y_i traitée le calcul d'intersection nécessite les valeurs suivantes :
  - y_{max}
  - x_{min}
  - dx/dy (1/pente)



Segment : y_i -> | y_{max} | x_{min} | dx/dy |

A-B : 7 -> | 10 | 1 | 3/6 |

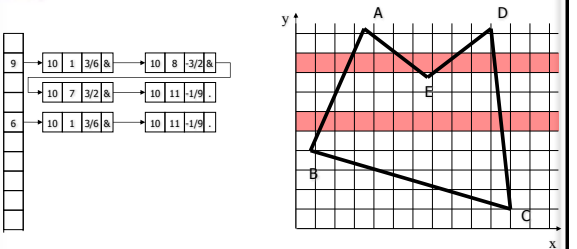
Remplissage

- Les points d'intersection sont organisés suivant une liste chaînée
- Chaque trame interceptera un nombre paire n de cotés du polygone
- La liste est triée à chaque ajout
  - Par ordre croissant des x_{min}
  - Pour deux x_{min} identiques, le tri se fait par ordre croissant de dx/dy

Segment : y_i -> | y_{max} | x_{min} | dx/dy | & | → | y'_{max} | x'_{min} | dx'/dy' | &

Début Fin

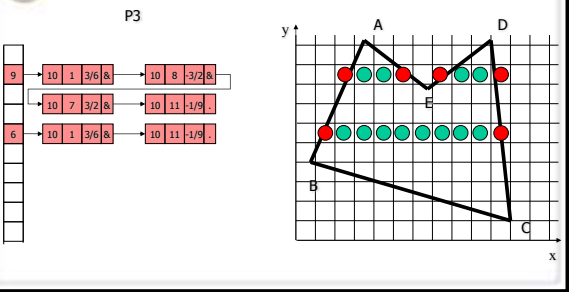
Remplissage



Remplissage

- Une fois la liste établie
  - Remplissage selon une règle de parité : incrémentation de la parité à chaque traversée de frontière et tracé si impair.
  - Après tri en x des n intersections Pi (1 <= i <= n), on tracera des segments entre chaque couple de coordonnées (P_{2*j-1},P_{2*j}) avec 1 <= j <= n.

Remplissage



Remplissage

- Gestion des conflits frontaliers
  - En raison de l'approximation de l'algorithme de traçage, les points d'intersections peuvent être à l'extérieur du polygone.
  - On prend les points intérieurs au polygone pour éviter les chevauchements.
    - D'où l'intérêt d'avoir l'équation de la droite traitée : y_{max}, x_{min}, dx/dy

XX

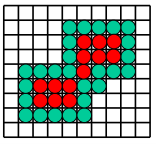
Algorithme

```
Procedure remplir_polygone (p:polygone)

début

  créer la structure SI
  initialiser la structure LCA à vide
  pour chaque trame intersectant le polygone

    début

    gérer les entrées dans LCA à partir de SI
    gérer les sorties de LCA à partir des informations contenues dans SI
    afficher tous les morceaux de trames décrits dans LCA

    fin

fin
```

Remplissage

Remplissage de régions (méthode du germe)

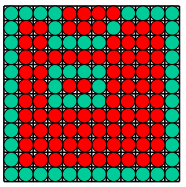
- Cet algorithme traite les régions définies par une frontière.
- À partir d'un pixel intérieur à la région, on propage récursivement la couleur de remplissage au voisinage de ce pixel jusqu'à atteindre la frontière.



Remplissage

- A chaque itération:
  - On remplit tous les pixels pi jusqu'à la couleur limite à droite et à gauche du germe courant.
  - On recherche parmi les pixels au dessus et en dessous des Pi ceux qui sont le plus à gauche d'une suite horizontale maximale à remplir.
  - Ces pixels sont empilés comme germes des itérations suivantes.

Remplissage



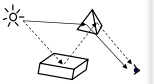
Apparence (Visibilité)

- Les calculs d'ombrage (shading)
- Le placage de texture (plaquage, mappage)

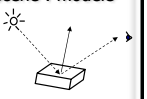
Les calculs d'éclairement et d'ombrage

Définitions

- Illumination
  - Transport du flux lumineux direct ou indirect depuis les sources lumineuses : modèle local vs. global



- Éclairement
  - Calcul de l'intensité lumineuse en un point de la scène : modèle d'interaction source lumineuse point éclairé



- Ombrage (shading)
  - Utilisation du modèle éclairement pour obtenir la couleur d'un pixel

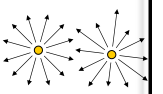
Éclairement et Ombrage

- Dépend de :
  - Position du point dans l'espace,
  - Orientation du point (élément de surface),
  - Caractéristiques de la surface (diffusion, réflexion, transparence…)
  - Sources de lumière (directionnelles, ponctuelles…).
  - Position et orientation de la "caméra" ou du point de vue.

Sources de lumières

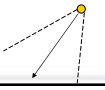
- la lumière **ambiante** : c'est une lumière qui éclaire toute la scène uniformément et qui est uniquement caractérisée par son intensité.

- les sources **ponctuelles** : ce sont des sources de lumière, supposées placées en un point précis et qui rayonnent la lumière radialement; elles sont donc caractérisées par leur intensité et leur position. Elles peuvent être **Isotropique** ou **Anisotropique**.
  - Lorsque la source prend du volume elle devient « **source Étendue** »



- les sources **directionnelles** : ce sont des sources de lumière, supposées à l'infini et qui éclairent la scène avec des rayons parallèles à une direction donnée; elles sont donc caractérisées par leur intensité et leur direction.

- les sources de type **Projecteur** ou **spot** : ce sont des sources de lumière caractérisées par leur position, leur direction et un facteur de concentration.

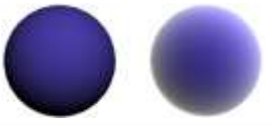


Modèles d'éclairement

- La lumière émise
- La lumière ambiante
- La réflexion diffuse
- La réflexion spéculaire
- Brillance

Modèles d'éclairement

- **Lumière émise** : Les objets ne sont pas intrinsèquement émetteurs de lumière et n'éclairent donc pas les autres objets mais possèdent ce niveau minimum d'éclairement.



Modèles d'éclairement

- **Lumière ambiante** : correspond au modèle le plus simple. On considère qu'il existe une source lumineuse présente partout et qui éclaire de manière égale dans toutes les directions.
- Ce modèle de lumière correspond au niveau minimum d'éclairage qui sera appliqué sur les objets.
- En terme physique cela correspond un peu au soleil réfléchi par tout l'environnement qui donne une sorte de lumière présente partout.

Modèles d'éclairement

- On définie l'intensité de cette lumière sur une surface comme suit :

Ip = **pa.Ia**

- Cette intensité lumineuse est constante sur toute la surface.
  - **Ia** désigne l'intensité de la lumière
  - **pa** est le coefficient de réflexion de la lumière ambiante par la surface
    - 0 <= **pa** <= 1
  - **Ip** correspond à l'intensité de la lumière résultant de la réflexion sur la surface.

Modèles d'éclairement

On augmente **pa**

Propriétés :

- On ne voit pas la 3D;
- Modélise simplement l'interréflexion entre toutes les surfaces d'une scène;
- Evite qu'un objet dans l'ombre soit complètement noir

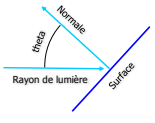
Modèles d'éclairement

- **La réflexion diffuse** : Dans ce modèle, l'intensité en un point d'une surface dépend de l'angle formé entre le rayon de lumière qui touche le point de la surface et la normale à la surface.
- Plus l'angle formé entre le rayon de lumière et la normale au plan est faible, plus l'intensité lumineuse réfléchie visible par l'observateur est forte.
- Le principe physique qui se cache derrière ce modèle est simple. Notre source lumineuse émet une certaine énergie au mètre carré. Suivant l'incidence des rayons lumineux, cette énergie sera répartie sur une plus ou moins importante surface de l'objet. Si les rayons et la surface sont perpendiculaires, alors l'énergie lumineuse sera répartie sur la plus petite surface possible et donc l'énergie par unité de surface sera maximale.

Modèles d'éclairement

- Si on considère que la surface est une surface Lambertienne (surface mate), alors, on peut considérer que la lumière qui arrive sur la surface est réfléchie dans toutes les directions de manière égale. Dans ce cas, la position de l'observateur ne compte pas. La lumière émise en direction de l'observateur dépend donc de :
  - L'intensité de la source lumineuse **Il**
  - L'angle **theta** formé par le rayon de lumière et la normale au plan
  - Coefficient de réflexion **pd** de la lumière diffuse par la surface
    - 0 <= **pd** <= 1
- On obtient la formule :

Ip = **pd** . **Il** . cos(**theta**)



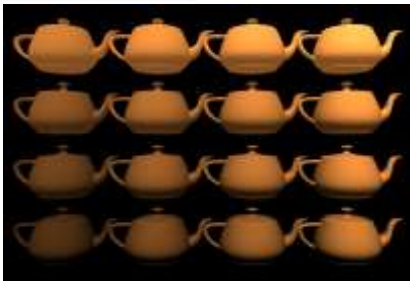
Modèles d'éclairement

On augmente **pd**, **pa** =0

Modèles d'éclairement

pa

diffuse + ambiante



pd

Modèles d'éclairement

- **La réflexion spéculaire** (Modèle de Phong [1973]) :
  - Le modèle de réflexion spéculaire se différencie du modèle de diffusion en faisant intervenir le point d'observation. Dans ce modèle les rayons de lumière sont réfléchis par symétrie par rapport à la normale à la surface. Ce modèle correspond aux propriétés de "miroir" des objets.

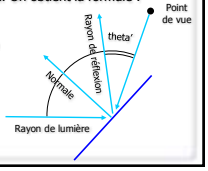
On observe bien l'aspect brillant des objets et les fortes variations en fonction du point de vue.

Modèles d'éclairement

- Il faut calculer le rayon réfléchi sur la face. Ensuite, l'intensité de la lumière observée dépend de :
  - **theta'** qui correspond à l'angle entre le rayon réfléchi et le point d'observation
  - **Il** l'intensité de la source de lumière
  - **ps** : coefficient de réflexion de la lumière spéculaire par la surface
    - 0 <= **ps** <= 1
- Si on se prend le rayon réfléchi dans l'oeil alors on a l'intensité maximum. Autour de cela, on peut quand même voir quelque chose d'un peu atténué. Cela veut dire que la surface ne réfléchit pas directement la rayon mais qu'il y a une certaine "diffusion" autour du rayon réfléchi. La fonction cosinus joue bien son rôle ici mais comme on veut pouvoir régler la diffusion autour du rayon réfléchi, on introduit le coefficient **n**. On obtient la formule :

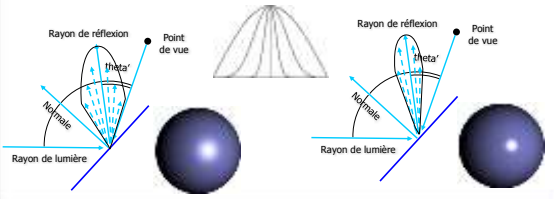
Is = **ps** * **Il** * cos(**theta'**)^**n**

- **n** : rugosité
  - ∞ (1024) pour un miroir
  - 1 pour une surface très rugueuse.



Modèles d'éclairement

- La rugosité corresponds à la brillance (*shininess*) : cette valeur détermine la taille et l'intensité de la tâche de réflexion spéculaire.
- Plus la valeur est grande, et plus la taille est petite et l'intensité importante.



Modèles d'éclairement

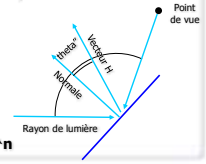
- **La réflexion spéculaire** (Modèle Blinn-Phong [1977])
  - Évaluation du rayon de réfléchi relativement coûteuse
    - Version simplifiée/accélérée de l'éclairage Phong
  - Évaluation du vecteur H : vecteur séparateur point de vue-rayon de lumière

Is = **ps** * **Il** * cos(**theta''**)^**n**

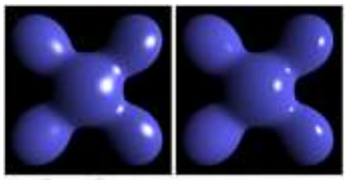
$$H = \frac{V + L}{2}$$

V : direction de vision
L : source d'éclairage

Is = **ps** * **Il** * cos(**theta''**)^**n**

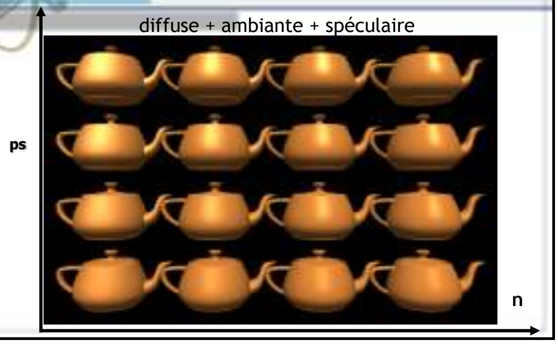


Modèles d'éclairement



Modèles d'éclairement

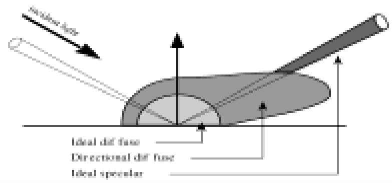
diffuse + ambiante + spéculaire



Modèles d'éclairement

- **Modèle complet** : on ajoute tout, on pondère avec un coefficient d'atténuation **Fd**

I(P) = pa . Ia + Fd . (pd . Il . cos(theta1) + ps . Il. cos(theta2)^n)



Modèles d'éclairement

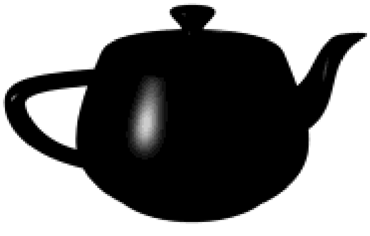
- **I(P) = pa . Ia + Fd .(pd . Il . cos(theta1) + ps . Il cos(theta2)^n)**

Modèles d'éclairement

- **I(P) = pa . Ia + Fd .(pd . Il . cos(theta1) + ps . Il cos(theta2)^n)**

Modèles d'éclairement

- **I(P) = pa . Ia + Fd .(pd . Il . cos(theta1) + ps . Il cos(theta2)^n)**



Modèles d'éclairement

- **I(P) = pa . Ia + Fd .(pd . Il . cos(theta1) + ps . Il cos(theta2)^n)**

Modèles d'éclairement

- **Modèle coloré**
  - une intensité par composante de couleur.
- **Plusieurs sources lumineuses**
  - somme des intensités.
- **Transparence**
  - manière de combiner couleur de fond et couleur de l'objet.
  - Un paramètre de transparence t.

I = t.I(P) + (1-t).I(derriere P)

- **Halo**
  - la couleur dépend de l'épaisseur traversée.

Modèles d'ombrage

Modèles d'ombrage

- Les modèles locaux
  - La luminance à la surface d'un objet est calculée à partir
    - Des paramètres de l'objet
    - Des paramètres des sources de lumière
  - Un objet est considéré comme isolé
- Les modèles globaux
  - La luminance à la surface d'un objet est calculée à partir
    - Des paramètres de tous les objets de la scène
    - Des paramètres des sources de lumière

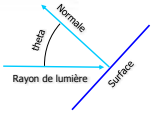
Les modèles locaux

Ombrage de Lambert

- **Ombrage de Lambert (plat)** :
  - La méthode d'ombrage la plus simple pour les facettes polygonales est l'ombrage plat (ou constant).
  - L'idée est de calculer une seule valeur d'illumination pour l'ensemble de la facette.
    - Par exemple au point milieu de la facette en prenant pour normale à la surface celle du plan contenant la facette.

Ombrage plat de la sphère pour : 16 16 *facettes,* 32 32 *et* 64 64

Ombrage de Lambert

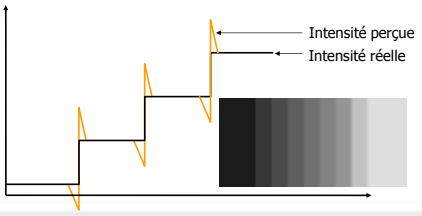


Ip = **pd** . **Il** . cos(**theta**)

- Composantes :
  - **Il** : intensité de la source lumineuse
  - **theta** : angle formé par le rayon de lumière et la normale au plan
  - **pd** : coefficient de réflexion de la lumière diffuse par la surface
    - matériau diffus
- Noue répétons le même schéma de calcul pour les autres composantes d'éclaire ment : spéculaire, ambiant, etc.

Ombrage de Lambert

- **Problème** :
  - Il y a des discontinuités le long des facettes;
  - L'œil exagère les changements d'intensité et les changements de pente de l'intensité => effet Mach Banding;
- **Solution => Interpolation.**



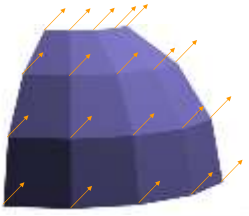
Ombrage de Gouraud


Henri Gouraud

- **Ombrage de Gouraud :**
  - La méthode développée par Gouraud [1971] élimine les discontinuités d'intensité sur une facette polygonale par interpolation des valeurs d'intensité aux sommets de la facette.
  - Cette méthode est largement utilisée et se retrouve dans la majorité des matériels graphiques existants (librairies, cartes graphiques).

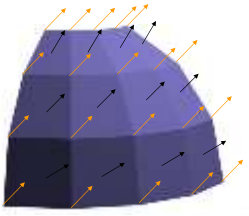
Ombrage de Gouraud

- Cette méthode requiert la connaissance de la normale à la surface aux sommets des facettes polygonales.



Ombrage de Gouraud

- Cette méthode requiert la connaissance de la normale à la surface aux sommets des facettes polygonales.
- Ces normales peuvent être soit données soit déterminées en calculant la moyenne des normales des facettes partageant un sommet P

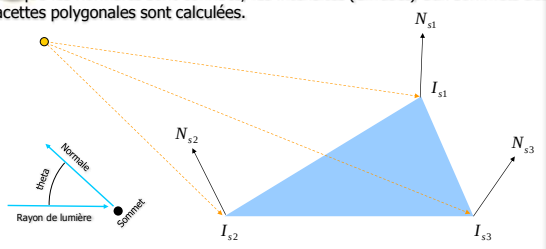


$$N_s = \frac{\sum N_i}{\left|\sum N_i\right|}$$

N_s : *normale du sommet traité*
N_i : *normales des faces incidentes*

Ombrage de Gouraud

- Lorsque les normales sont connues, les intensités (diffuses) aux sommets des facettes polygonales sont calculées.

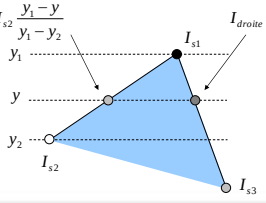


Ip = **pd** . **Il** . cos(**theta**)

Ombrage de Gouraud

- On réalise interpolation bilinéaire des intensités lumineuse sur toute la face
  - Interpolation suivant les arrêtes

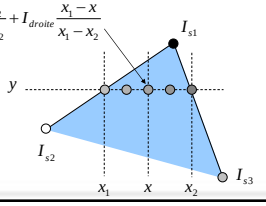
$$I_{gauche} = I_{s1}\frac{y-y_2}{y_1-y_2} + I_{s2}\frac{y_1-y}{y_1-y_2} \qquad I_{droite} = I_{s1}\frac{y-y_3}{y_1-y_3} + I_{s2}\frac{y_1-y}{y_1-y_3}$$



Ombrage de Gouraud

- On réalise interpolation bilinéaire des intensités lumineuse sur toute la face
  - Interpolation suivant les arrêtes
  - Interpolation horizontale à l'intérieur de la face
- L'interpolation s'effectue à l'aide de l'algorithme de balayage de ligne utilisée pour le remplissage de polygone et le z-buffer : Scan-line

$$I = I_{gauche}\frac{x-x_2}{x_1-x_2} + I_{droite}\frac{x_1-x}{x_1-x_2}$$



Ombrage de Gouraud

```
InterpolationGaugaud(polygone)
{
POUR chaque polygone FAIRE
    calcul de la normale au polygone

POUR chaque sommet (S1, S2, etc.) du polygone FAIRE
    calcul de la normale : moyenne des normales des faces incidentes

POUR chaque sommet (S1, S2, etc.) du polygone FAIRE
    calcul l'intensité lumineuse d'un modèle d'éclairement (diffus, spéculaire, etc.)

POUR chaque point S FAIRE
    SI S appartient a une arrête [S1,S2] du polygone ALORS
            calcul de l'intensité lumineuse Igauche || Idroite par interpolation linéaire entre [Is1, Is2]
    Sinon
            calcul de l'intensité lumineuse I par interpolation linéaire horizontale entre[Igauche, Idroite
    ]
}
```

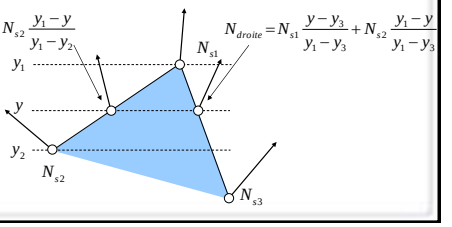
Ombrage de Gouraud

- Caractéristiques :
  - Technique efficace en terme de temps de calcul pour l'obtention d'un réalisme moyen.
  - Implantation facile dans le cadre de la programmation de l'algorithme du Z -Buffer.
  - L'utilisation d'une interpolation entre des valeurs numériques n'est qu'une approximation qui peut conduire à des défauts de visualisation.

Ombrage de Phong

- **Ombrage de Phong [1973] :**
  - Consiste à déterminer la normale en un point d'une facette polygonale par interpolation des normales aux sommets de cette facette.
    - Interpolation suivant les arrêtes

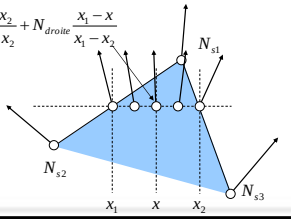
$$N_{gauche} = N_{s1}\frac{y-y_2}{y_1-y_2} + N_{s2}\frac{y_1-y}{y_1-y_2} \qquad N_{droite} = N_{s1}\frac{y-y_3}{y_1-y_3} + N_{s2}\frac{y_1-y}{y_1-y_3}$$



Ombrage de Phong

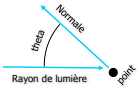
- **Ombrage de Phong [1973] :**
  - Consiste à déterminer la normale en un point d'une facette polygonale par interpolation des normales aux sommets de cette facette.
    - Interpolation suivant les arrêtes
    - Interpolation horizontale à l'intérieur de la face

$$N = N_{gauche}\frac{x-x_2}{x_1-x_2} + N_{droite}\frac{x_1-x}{x_1-x_2}$$



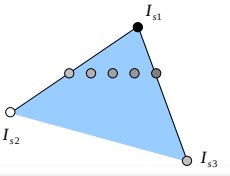
Ombrage de Phong

- Utilisation des modèles d'illumination pour calculer les intensités lumineuses
  - Modèle diffus



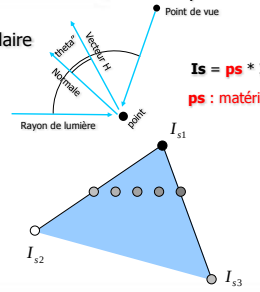
Ip = **pd** . **Il** . cos(**theta**)

pd : matériau diffus



Ombrage de Phong

- Utilisation des modèles d'illumination pour calculer les intensités lumineuse
  - Modèle spéculaire



Is = **ps** * **Il** * cos(**theta''**)^**n**

ps : matériau spéculaire

Ombrage de Phong

- **Ombrage de Phong [1973] :**
  - L'intérêt de cette approche par rapport à l'ombrage de Gouraud réside principalement dans sa capacité à traiter les réflexions spéculaires.
  - Gouraud ne permet pas de prendre en compte les réflexions spéculaires lorsque celles-ci sont localisées au centre d'une facette.

Modèles d'ombrage

Ombrage plat Ombrage de Gouraud Ombrage de Phong

Modèles d'ombrage

- **Ombrage de Phong [1973] :**
  - L'ombrage de Phong est d'une manière générale meilleur que celui de Gouraud car l'interpolation est effectuée sur les normales et non les intensités.
    - Cela même pour un modèle d'illumination sans réflexion spéculaire.
  - Augmente le coût du rendu.

Modèles d'ombrage

Ombrage plat

Ombrage de Gouraud

Ombrage de Phong

Modèles d'ombrage

InterpolationPhong(polygone)
{
POUR chaque polygone FAIRE
  calcul de la normale au polygone

POUR chaque sommet (S1, S2, etc.) du polygone FAIRE
  calcul de la normale : moyenne des normales des faces incidentes

POUR chaque point S du polygone FAIRE
  SI S appartient a une arrête [S1,S2] du polygone ALORS
    calcul de la normale N_{gauche} || N_{droite} par interpolation linéaire entre [N_{s1}, N_{s2}]
  Sinon
    calcul de l'intensité lumineuse N par interpolation linéaire horizontale entre[N_{gauche}, N_{droite}]

POUR chaque point S du polygone FAIRE
  calcul l'intensité lumineuse d'un modèle d'éclairement (diffus, spéculaire, etc.)

}

Modèles d'ombrage

- Rajouter des couleurs
  - Trois composantes

$$\begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \begin{bmatrix} Ia_r pa_r \\ Ia_v pa_v \\ Ia_b pa_b \end{bmatrix} + (N \cdot L) \begin{bmatrix} Il_r pd_r \\ Il_v pd_v \\ Il_b pd_b \end{bmatrix} + (N \cdot H)^n \begin{bmatrix} Il_r ps_r \\ Il_v ps_v \\ Il_b ps_b \end{bmatrix}$$

Modèles d'ombrage

- Rajouter des couleurs
  - Trois composantes

$$\begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \begin{bmatrix} Ia_r pa_r \\ Ia_v pa_v \\ Ia_b pa_b \end{bmatrix} + (N \cdot L) \begin{bmatrix} Il_r pd_r \\ Il_v pd_v \\ Il_b pd_b \end{bmatrix} + (N \cdot H)^n \begin{bmatrix} Il_r ps_r \\ Il_v ps_v \\ Il_b ps_b \end{bmatrix}$$

Rouge
Vert
Bleu

Modèles d'ombrage

- Rajouter des couleurs
  - Trois composantes

$$\begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \begin{bmatrix} Ia_r pa_r \\ Ia_v pa_v \\ Ia_b pa_b \end{bmatrix} + (N \cdot L) \begin{bmatrix} Il_r pd_r \\ Il_v pd_v \\ Il_b pd_b \end{bmatrix} + (N \cdot H)^n \begin{bmatrix} Il_r ps_r \\ Il_v ps_v \\ Il_b ps_b \end{bmatrix}$$

Matériau ambiant — Matériau diffus — Matériau spéculaire

Modèles d'ombrage

- Rajouter des couleurs
  - Trois composantes

$$\begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \begin{bmatrix} Ia_r pa_r \\ Ia_v pa_v \\ Ia_b pa_b \end{bmatrix} + (N \cdot L) \begin{bmatrix} Il_r pd_r \\ Il_v pd_v \\ Il_b pd_b \end{bmatrix} + (N \cdot H)^n \begin{bmatrix} Il_r ps_r \\ Il_v ps_v \\ Il_b ps_b \end{bmatrix}$$

Source ambiante — Source diffuse — Source spéculaire

Modèles d'ombrage

- Rajouter des couleurs
  - Trois composantes

$$\begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \begin{bmatrix} Ia_r pa_r \\ Ia_v pa_v \\ Ia_b pa_b \end{bmatrix} + (N \cdot L) \begin{bmatrix} Il_r pd_r \\ Il_v pd_v \\ Il_b pd_b \end{bmatrix} + (N \cdot H)^n \begin{bmatrix} Il_r ps_r \\ Il_v ps_v \\ Il_b ps_b \end{bmatrix}$$

Couleur finale

Modèles d'ombrage

- Plusieurs sources d'éclairage

$$\begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \begin{bmatrix} Ia_r pa_r \\ Ia_v pa_v \\ Ia_b pa_b \end{bmatrix} + \sum_{Il=lights} \left((N \cdot L) \begin{bmatrix} Il_r pd_r \\ Il_v pd_v \\ Il_b pd_b \end{bmatrix} + (N \cdot H)^n \begin{bmatrix} Il_r ps_r \\ Il_v ps_v \\ Il_b ps_b \end{bmatrix} \right)$$