Carte graphique dédiée
&
GPU (Graphics Processor Unit)
&
Pipeline graphique

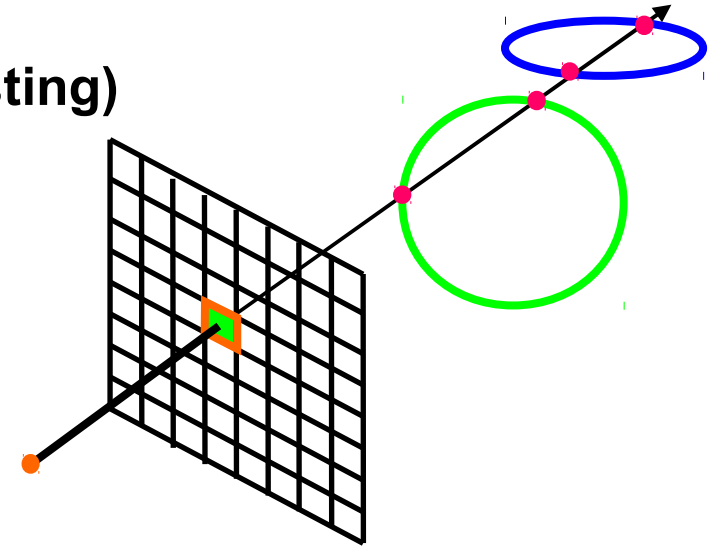
Rendu d'une scène 3D

- **En entrée :**
  - Un point de vue
  - Une description géométrique de la scène
    (= ensemble de primitives, ex. polygones)
  - Des attributs de matériaux associés à chaque objet
  - Un ensemble de lumières

- **En sortie :**
  - Un tableau de couleur

Différentes approches de rendu

- **But de la synthèse**
  - Calcul d'une couleur par pixel
- **Tracé de Rayon (Ray-tracing – Ray Casting)**
  - Propagation d'un rayon a partir des pixels
  - **Avantages :**
    - Simples à mettre en oeuvre
  - **Désavantages**
    - Besoin de structures d'accélération
    - Dépend purement de la résolution

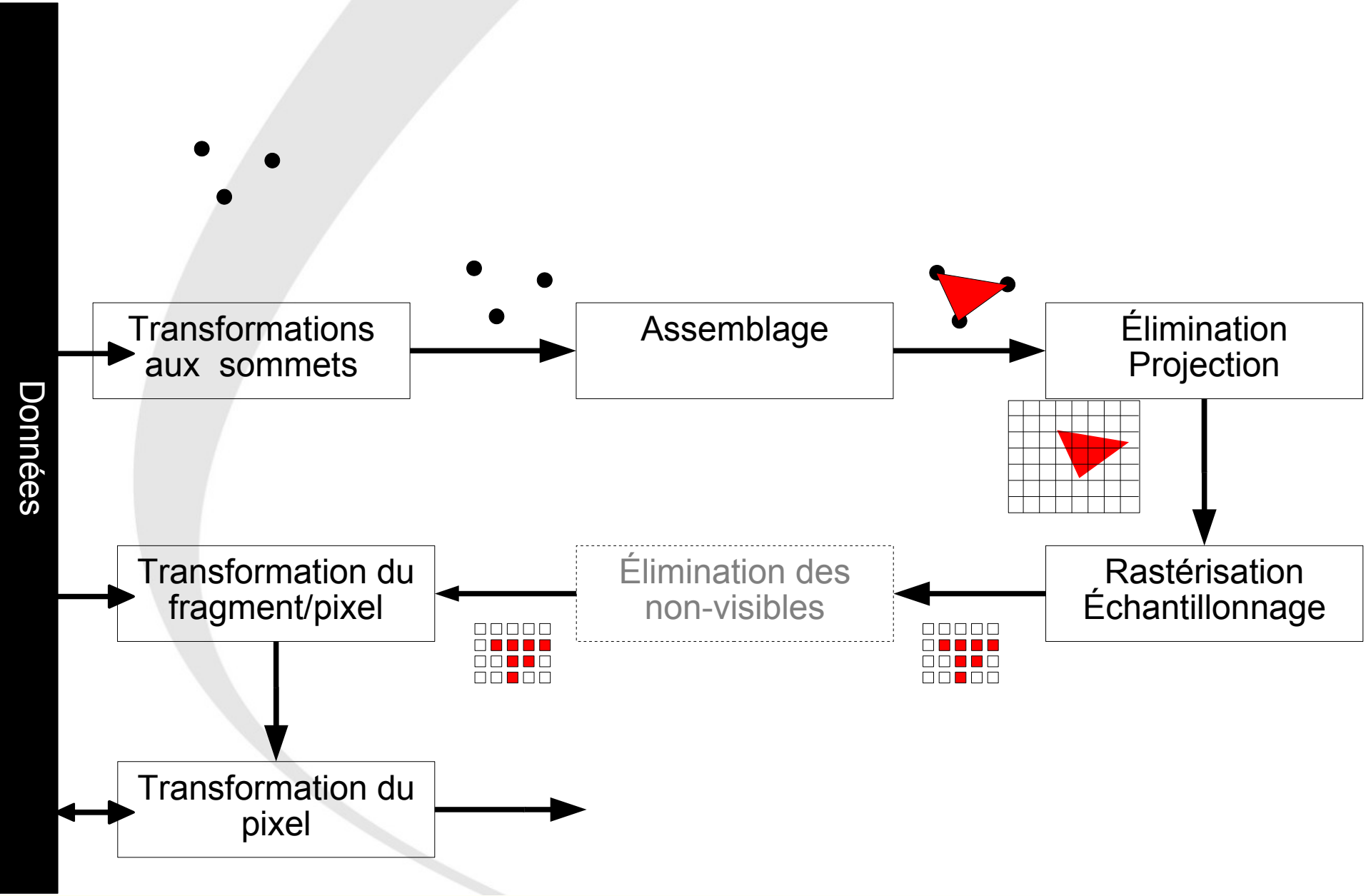


- **Reyes (Renderman - PIXAR) : 1987**
  - Subdivision en micro-primitive (plus d'un micro-primitive par pixel)
  - **Avantages**
    - Cohérence locale pour un ensemble de micro-primitive: vectorisation
  - **Désavantages**
    - Création d'une large quantité de micro-polygones

« Graphics Pipeline »

- **Factorisation des calculs.**
  - Balance entre la complexité par pixel pour une complexité par primitive
- **Notion de tampon (Buffer)**
  - Outil de transfert
    - Communication avec le processus central
    - Communication avec le système d'affichage
    - Boucle sur le pipeline
  - Un tampon =
    - Une zone mémoire (partagée ou localisée)
    - Un identificateur unique
  - Avantage
    - Transfert par paquet
    - Réduction des synchronisations nécessaires
- **Machine à état**
  - Toute définition de propriété reste valide tant que non-modifié ou non-terminé
    - Couleur Face (Sommets)* => une couleur par face
    - Face (Couleur Sommet)* => une couleur par sommet

La chaîne de rendu



« Graphics Pipeline »

- **Factorisation des calculs.**
  - Balance entre la complexité par pixel pour une complexité par primitive
- **Notion de tampon (Buffer)**
  - Outil de transfert
    - Communication avec le processus central
    - Communication avec le système d'affichage
    - Boucle sur le pipeline
  - Un tampon =
    - Une zone mémoire (partagée ou localisée)
    - Un identificateur unique
  - Avantage
    - Transfert par paquet
    - Réduction des synchronisations nécessaires

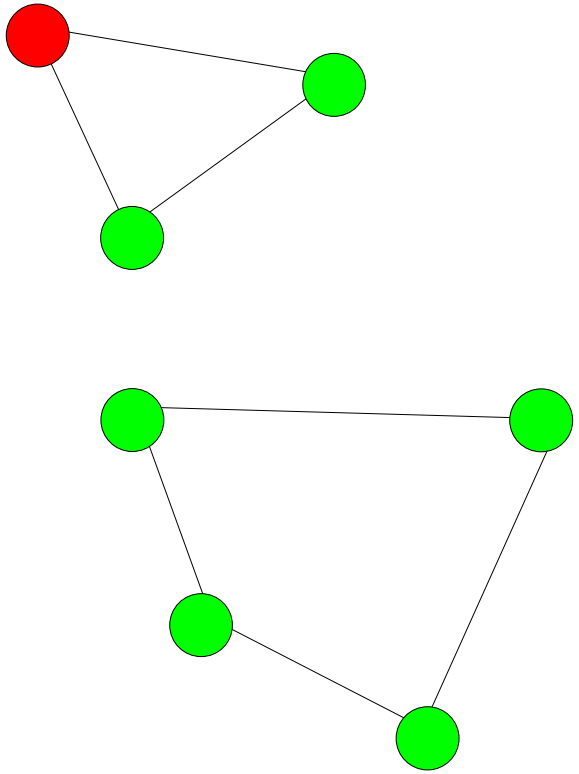
« Graphics Pipeline »

- **Factorisation des calculs.**
  - Balance entre la complexité par pixel pour une complexité par primitive
- **Notion de tampon (Buffer)**
  - Outil de transfert
    - Communication avec le processus central
    - Communication avec le système d'affichage
    - Boucle sur le pipeline
  - Un tampon =
    - Une zone mémoire (partagée ou localisée)
    - Un identificateur unique
  - Avantage
    - Transfert par paquet
    - Réduction des synchronisations nécessaires
- **Machine à état**
  - Toute définition de propriété reste valide tant que non-modifié ou non-terminé
    - Couleur Face (Sommets)* => une couleur par face
    - Face (Couleur Sommet)* => une couleur par sommet

Machine à état

- **Factoriser l'envoie d'information**
- **Tant qu'elle n'est pas modifiée, la valeur courante s'applique**

>couleur = rouge
>forme = triangle
>point 1
>point 2
>couleur = vert
>point 3
>forme = quadrangle
>point 4
>point 5
>point 6
>point 7



Introduction

Petit historique

- **Années 80 : SGI IRIS GL**
  - Concept de « graphic pipeline »
- **1992 : OpenGL 1.0**
  - Standard ouvert
  - Machine à états
- **1993 : 1ere carte supportant OpenGL 1.0**
- **1995 : DirectX/Direct3D 1.0**
  - GeForceConcept d'objets
- **1999 : 1er pipeline hardware grand public**
  - NVIDIA

La série de machine IRIS SGI

- IRIS = Integrated Raster Imaging System
- Système d'exploitation: IRIX (IRIS UNIX)

Indigo

Indy

Onyx

O2

Octane

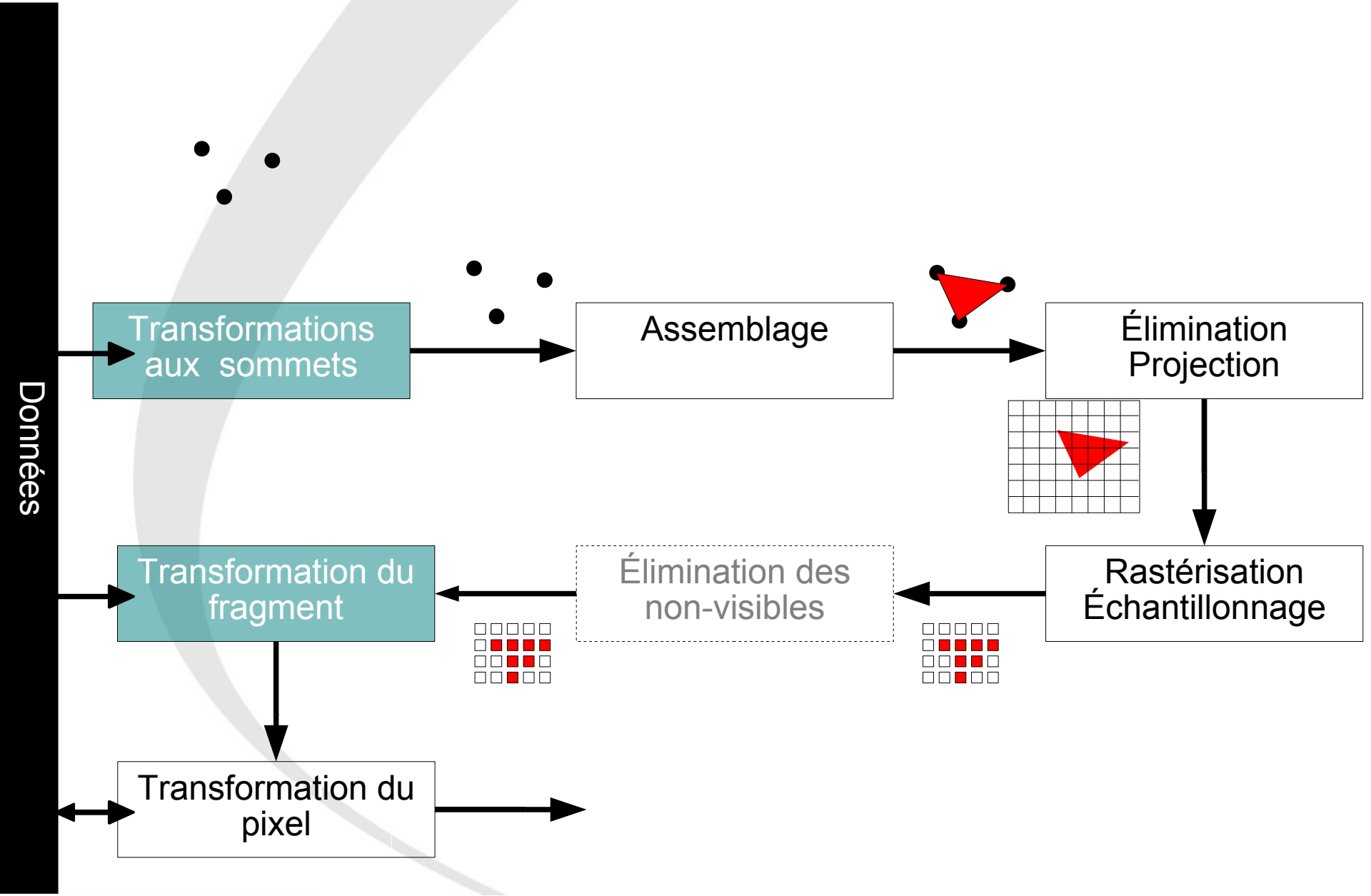
Visual Workstation

Tezro et Onyx4

Vers la programmation

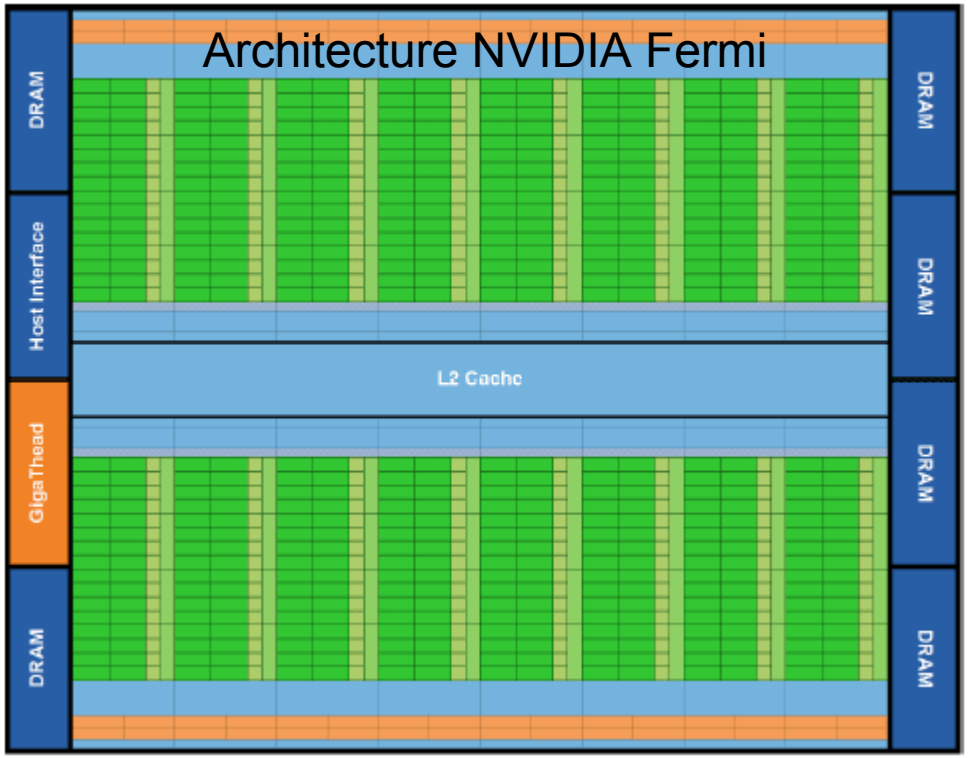
- **2000 : Register Combiners**
  - NVIDIA GeForce 2
  - Operation sur les textures
- **2001 :**
  - Vertex Program : NVIDIA GeForce 3
  - Fragment Shader : ATI Radeon 8500
  - Assembleur pour Cartes Graphiques
- **2002 :**
  - NVIDIA Cg : C pour le Graphique
  - Langage de haut niveau
- **2003 :**
  - Direct3D HLSL
  - OpenGL GLSL
- **2006**
  - Générateur de primitives
- **GPUGPU**

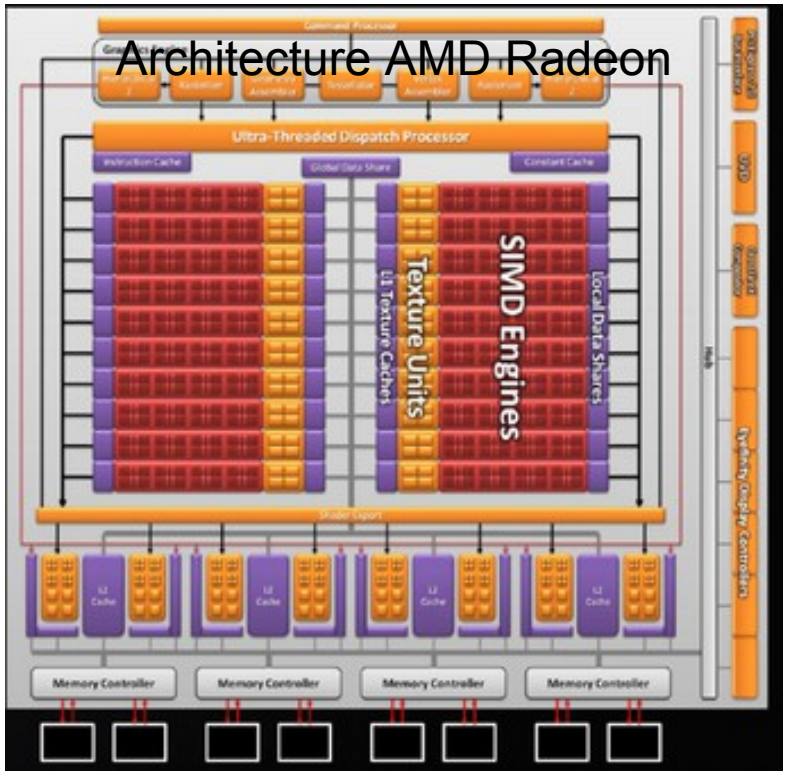
La chaîne de rendu



Processeur massivement parallèle

- **Plus d'une centaine de processeurs**
- **Calcul vectoriel (SIMD)**
  - Données : Vecteur 4D
  - Opérations : Unitaire par composantes (+ - * / …) , produits scalaires, ...
- **Des unités spécialisées**
  - Compression/Décompression d'images
  - Conversion de primitives en pixels (rastérisation).



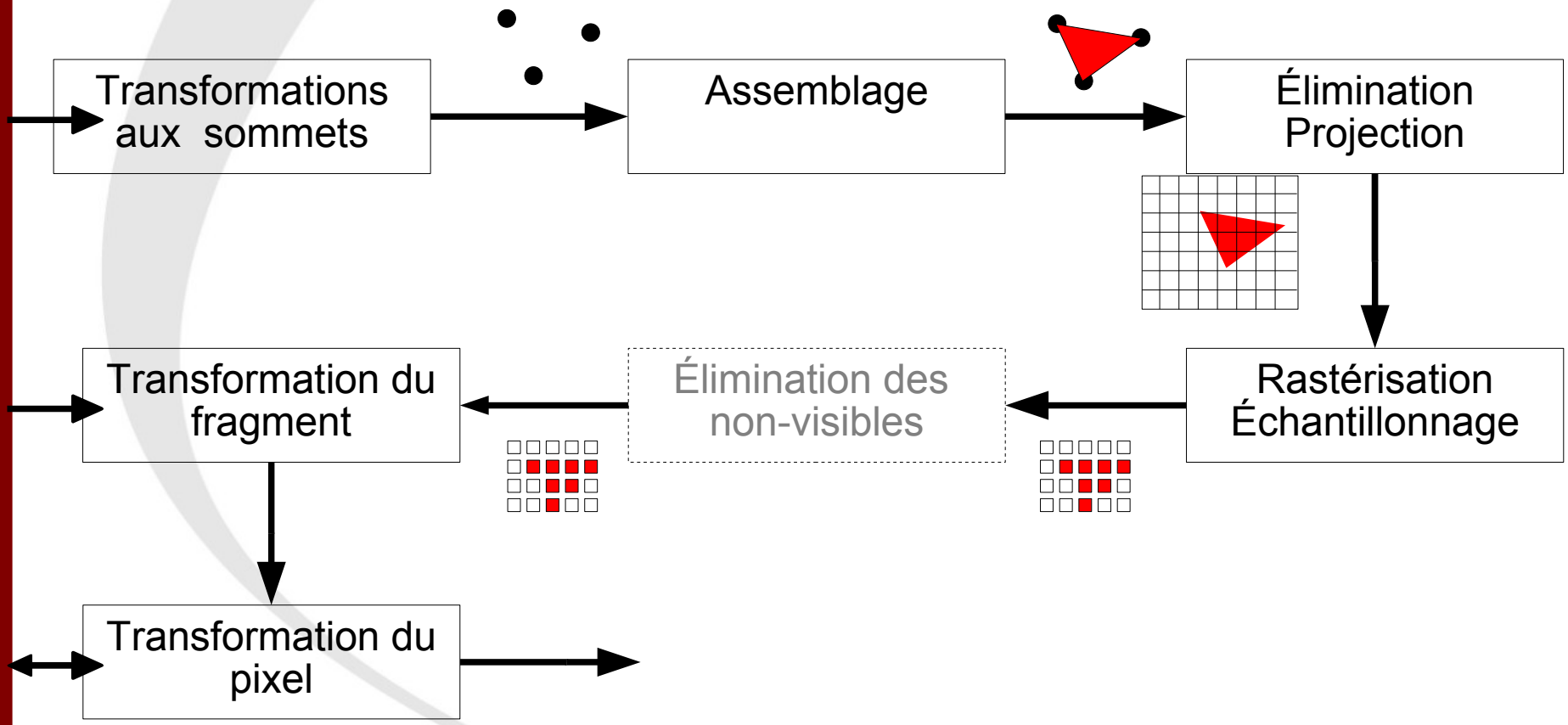


La chaîne de rendu

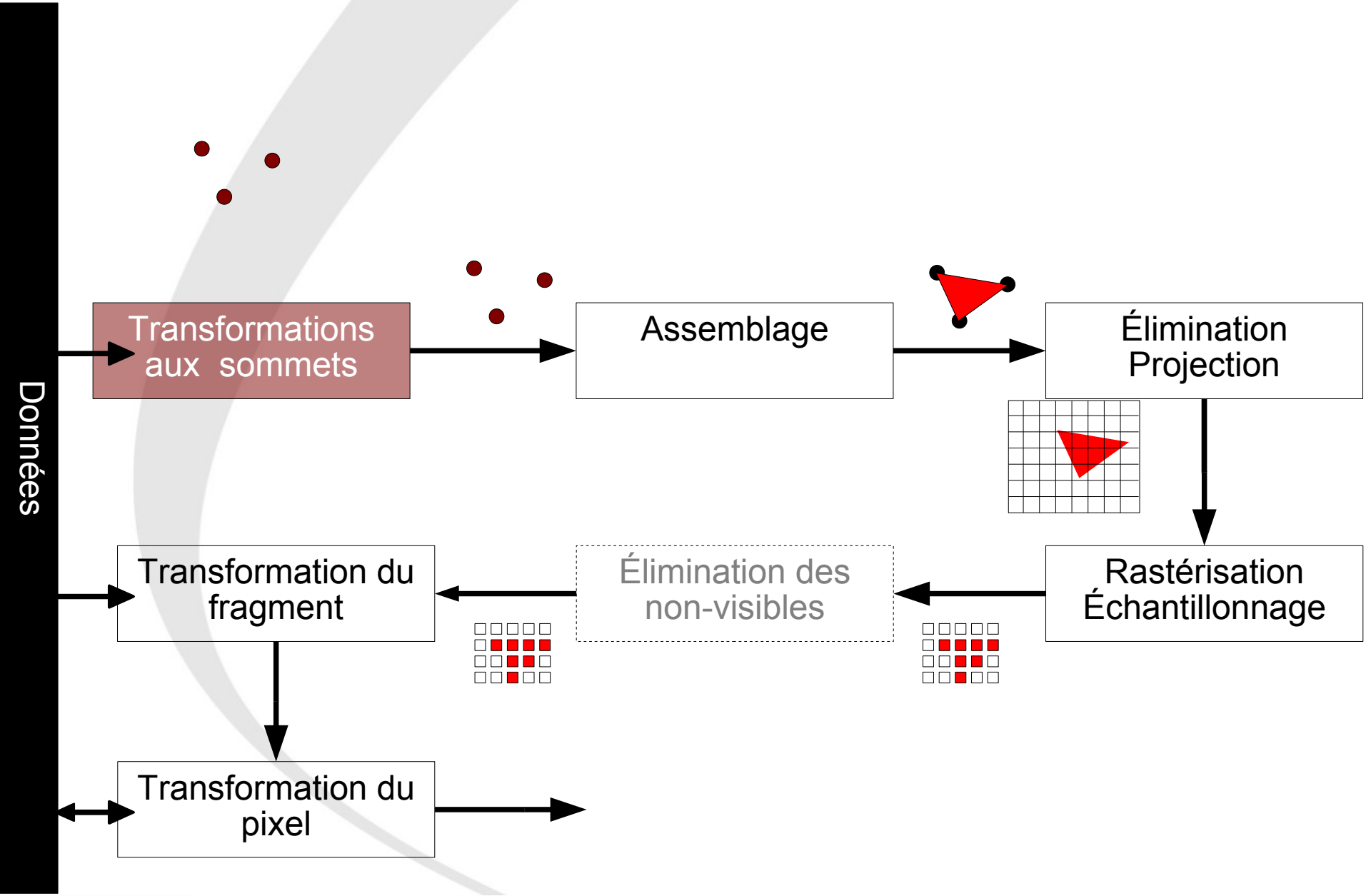
Données

Envoie de la géométrie au GPU

- Maillage indexé
- Primitives (triangles, quadrangles,...)
- Tableau de primitives



La chaîne de rendu



Opération par sommet

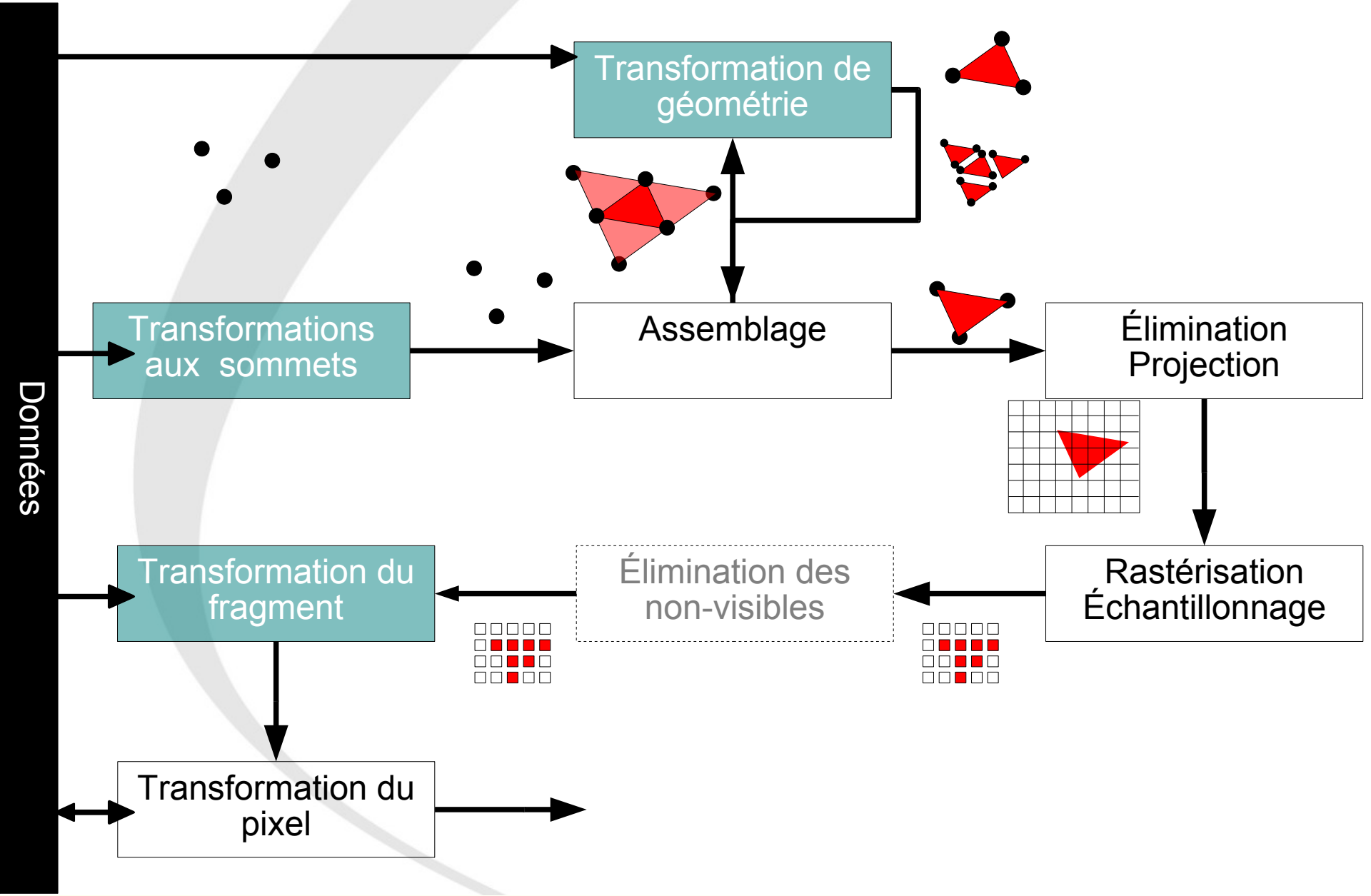
- **Transformation de projection**
  - Coordonnées monde vers coordonnées caméra
  - Coordonnées caméra vers coordonées viewport
- **Calcul de l'éclairement**
  - Calcul de l'intensité lumineuse au sommet
  - Coordonnée des textures (voir cours suivants)
- **Transformation géométriques**
  - Translation, rotation etc …
- **Toute autre opération par sommets**
  - Pas de connaissance des autres sommets
  - En parallèle

Opérations sur les sommets

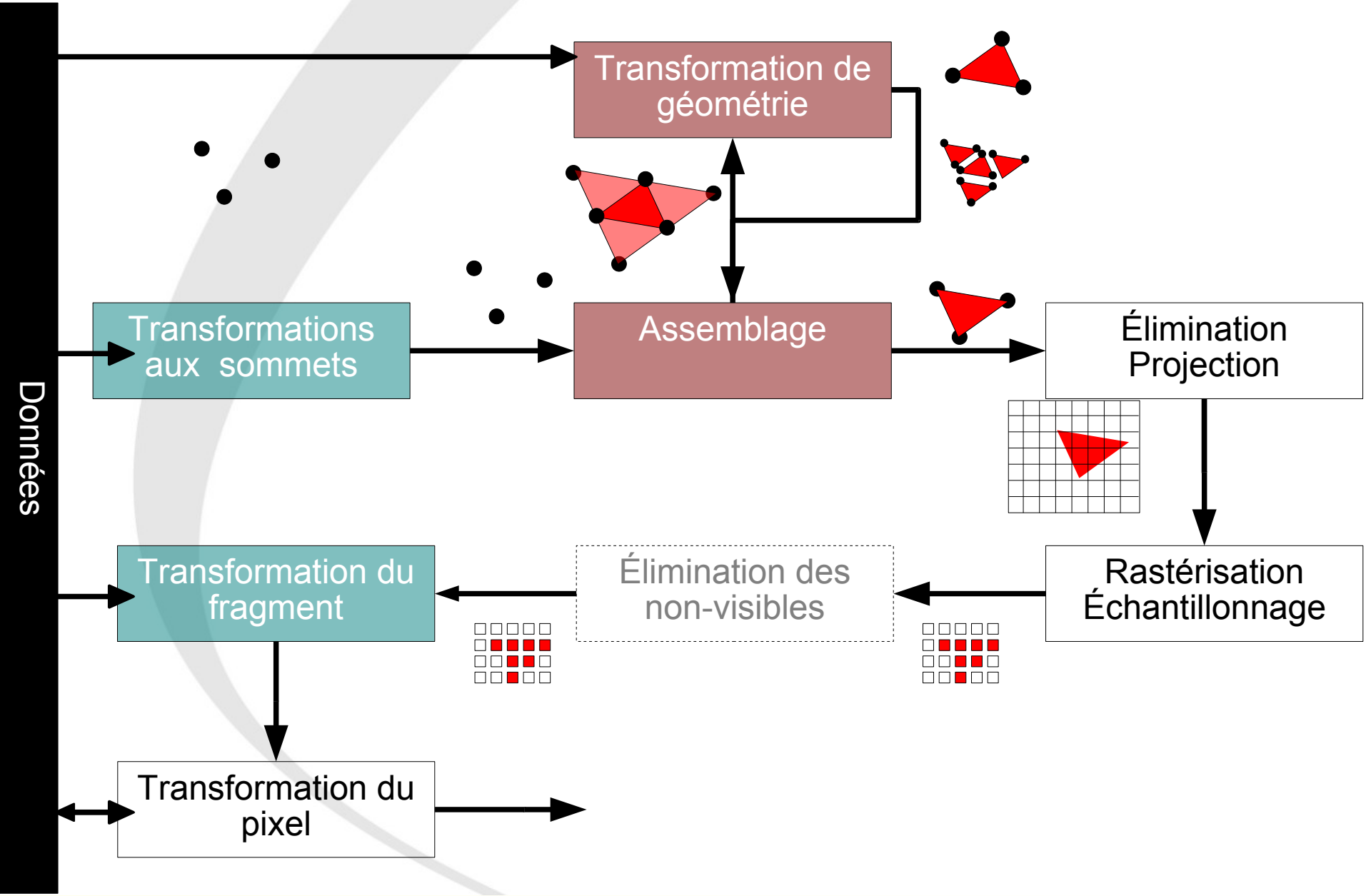
- **Notions absentes**
  - Voisinage
  - Pixel déjà traité
  - Face

- **Données présentes**
  - Variable globales
    - Matrices
    - Texture
    - .....
  - Données par face
  - Données par sommet
    - Profondeur, normale, couleur, paramètres d'éclairement

La chaîne de rendu

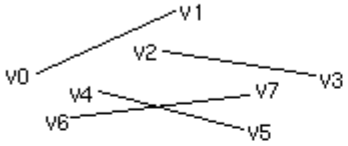


La chaîne de rendu

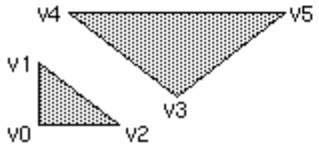


Primitives de Rendu

POINTS



LINES

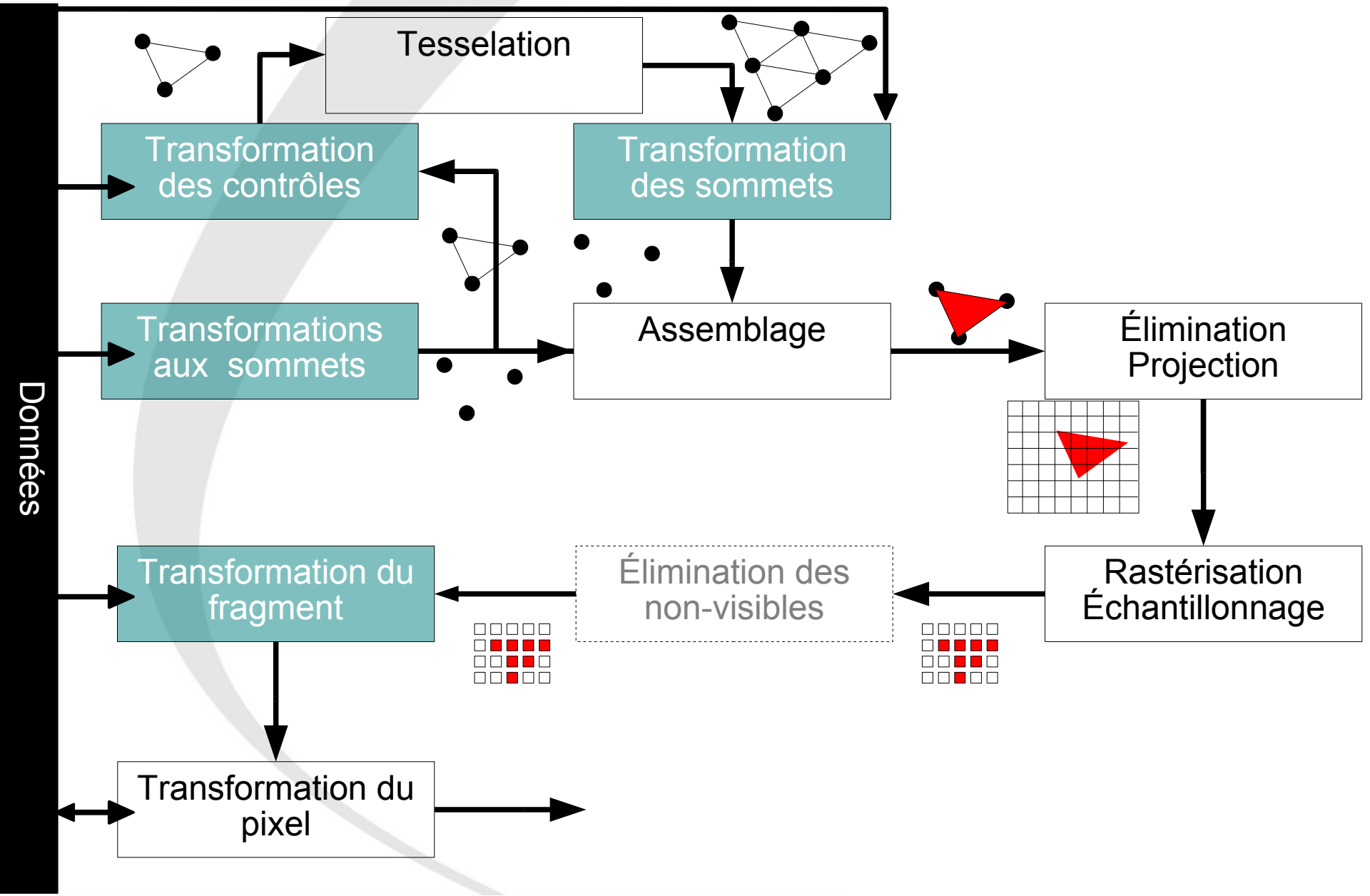


TRIANGLES

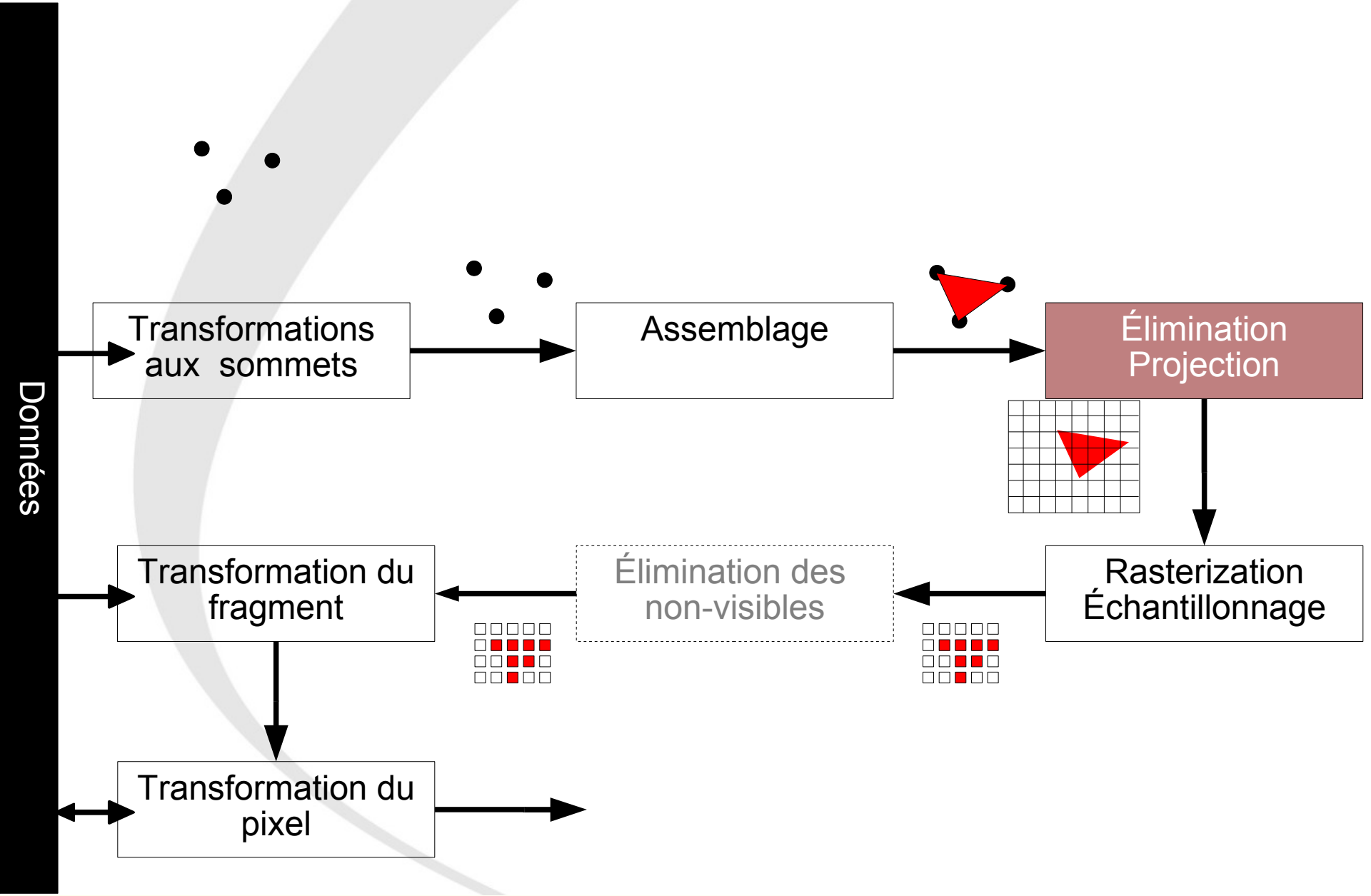
Opérations sur les primitives

- **Notions absentes**
  - Voisinage étendu
- **Données présentes**
  - Variable globales
  - Données par face
  - Données sur tous les sommets
    - Profondeur, normale, couleur, paramètres d'éclairement
  - 1 - Voisinage
- **Utilisation Classique**
  - Subdivision

La chaîne de rendu

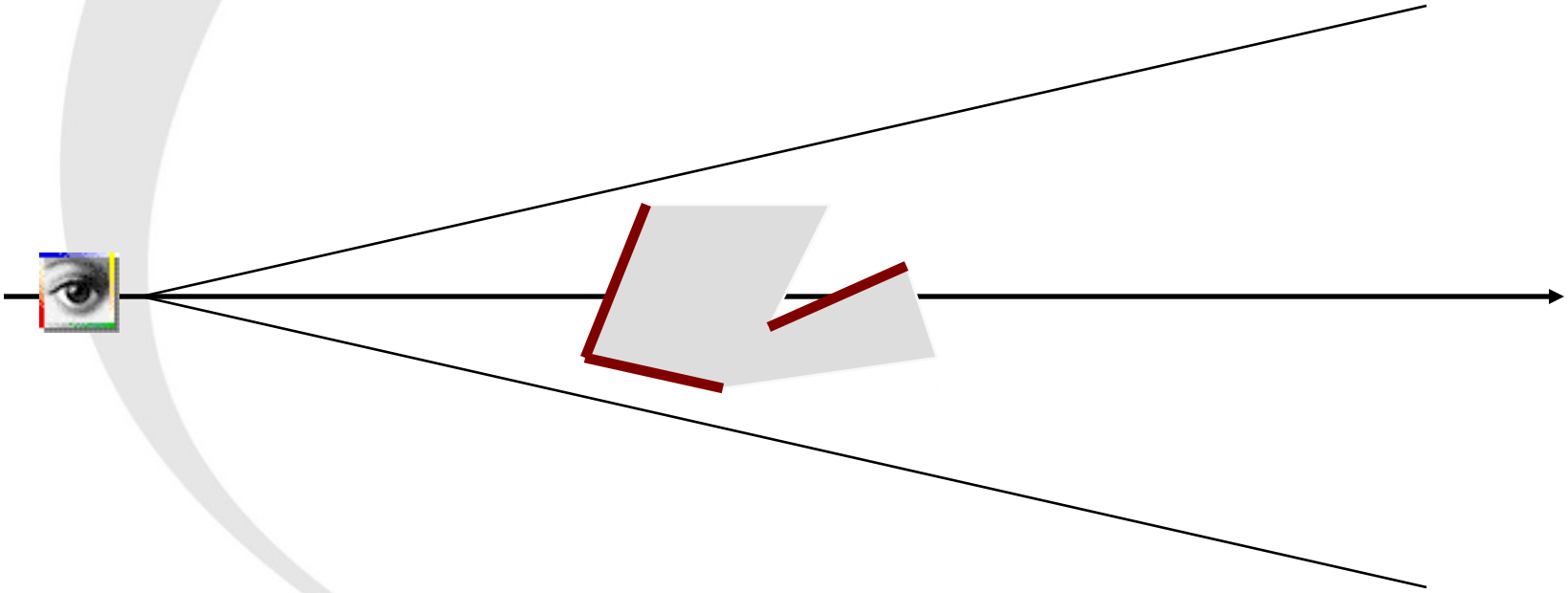


La chaîne de rendu



Back Face Culling : algorithme

- **Si le point de vue n'est pas devant le polygone, on n'affiche pas le polygone**
  - Processus - Objet
- **Produit scalaire :**
  - (Sommet-PointDeVue)*normale
  - > 0 : on garde le polygone
  - < 0 : on l'élimine

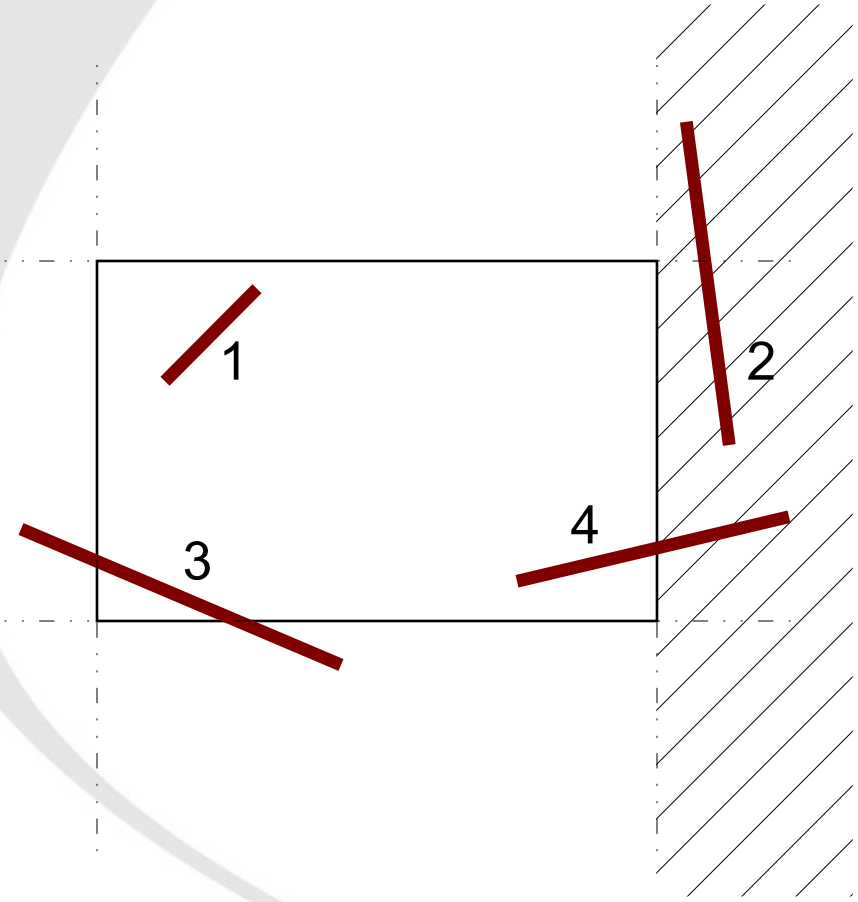


Back Face Culling

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

Découpage/Clipping

- **Points**
  - Teste d'appartenance à l'écran (ViewPort)
- **Segments**
  - Quatre types de segments



1: segment visible

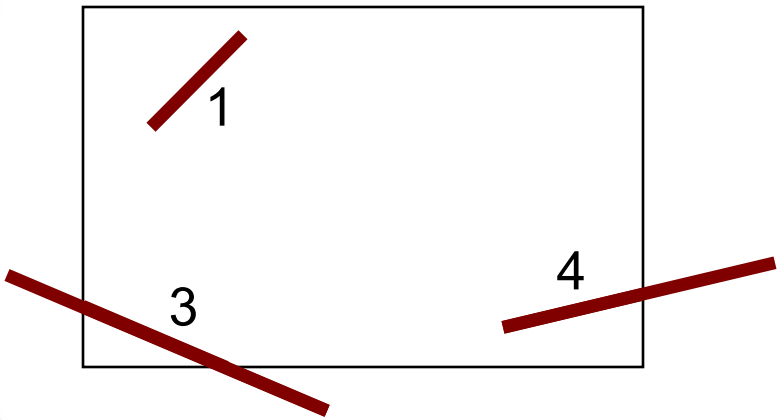
2 : segment invisible

3 : segment partiellement visible

4 : segment partiellement visible

Découpage/Clipping

- **Points**
  - Teste d'appartenance à l'écran (ViewPort)
- **Segments**
  - Quatre types de segments



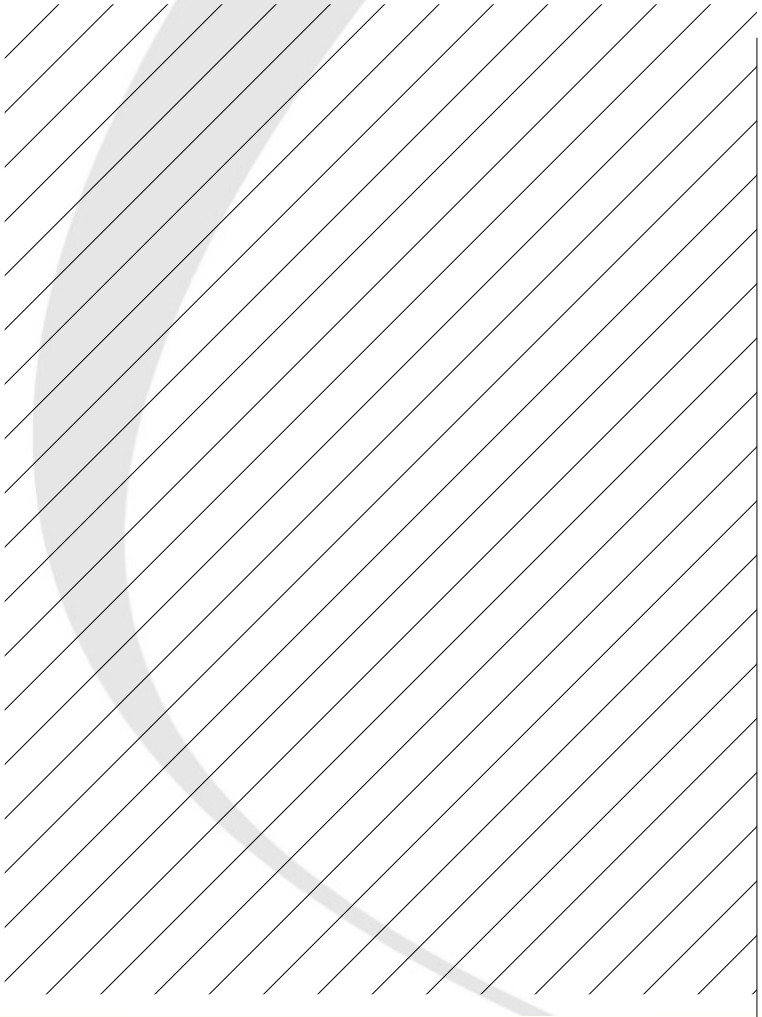
1: segment visible

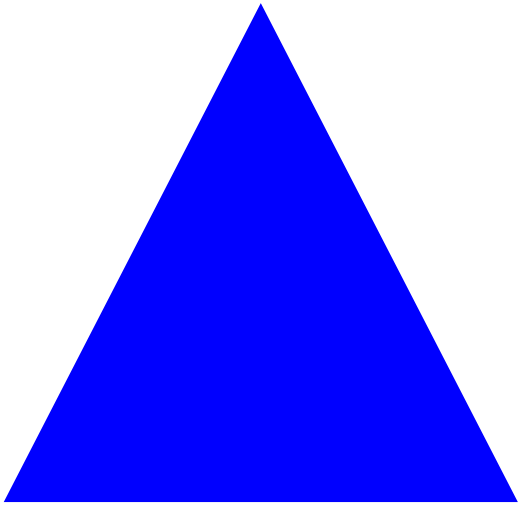
3 : segment partiellement visible

4 : segment partiellement visible

Triangle

- **Méthode de Sutherland-Hodgeman**
  - Test par rapport à chacun des cotés

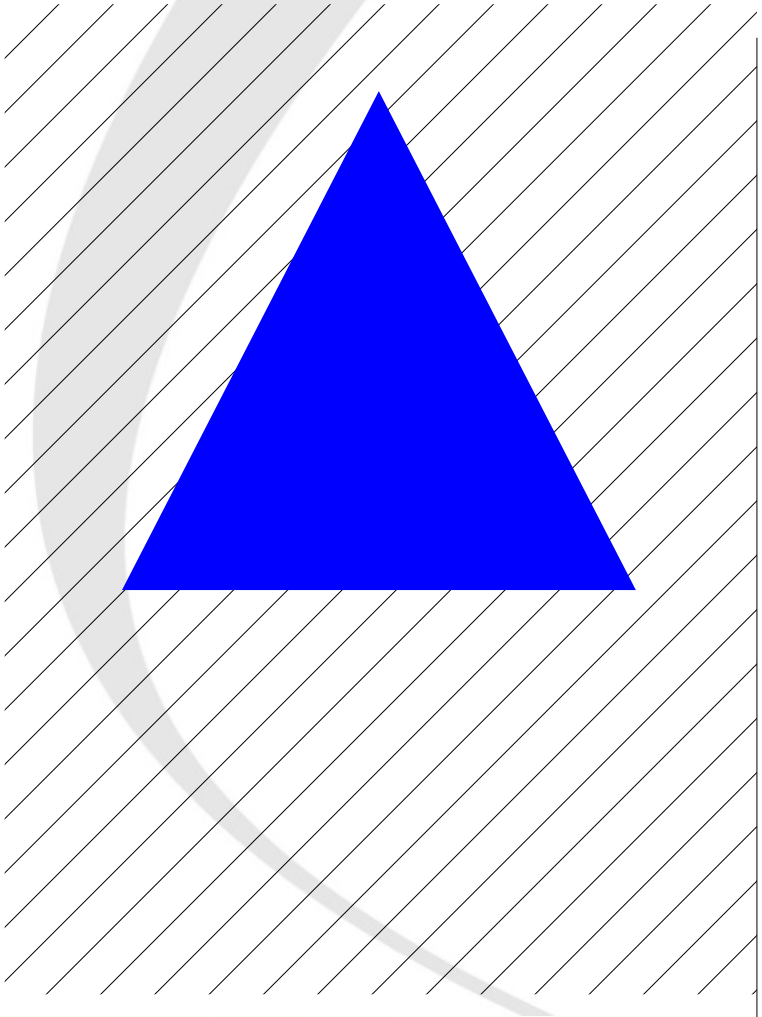




À l'intérieur

Triangle

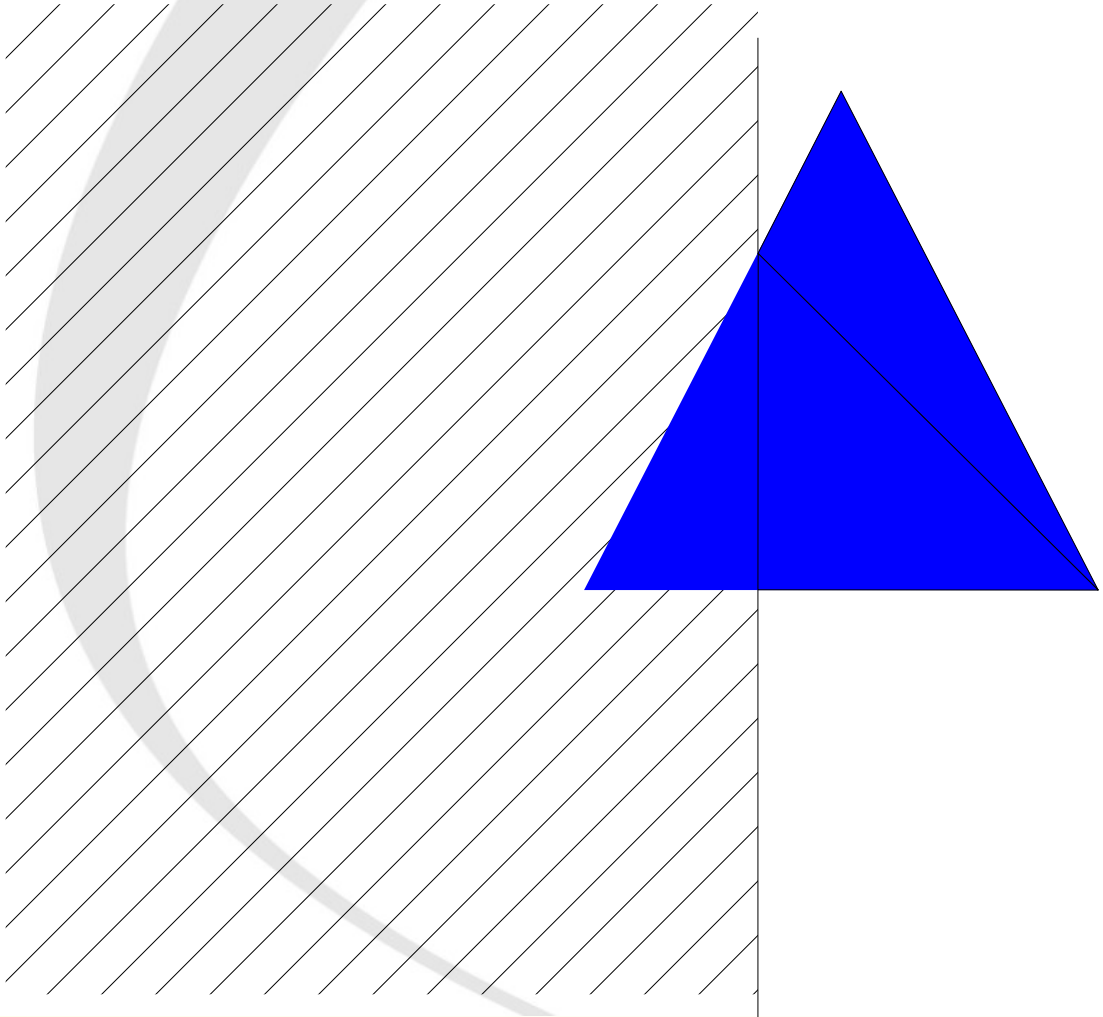
- **Méthode de Sutherland-Hodgeman**
  - Test par rapport à chacun des cotés



À l'extérieur

Triangle

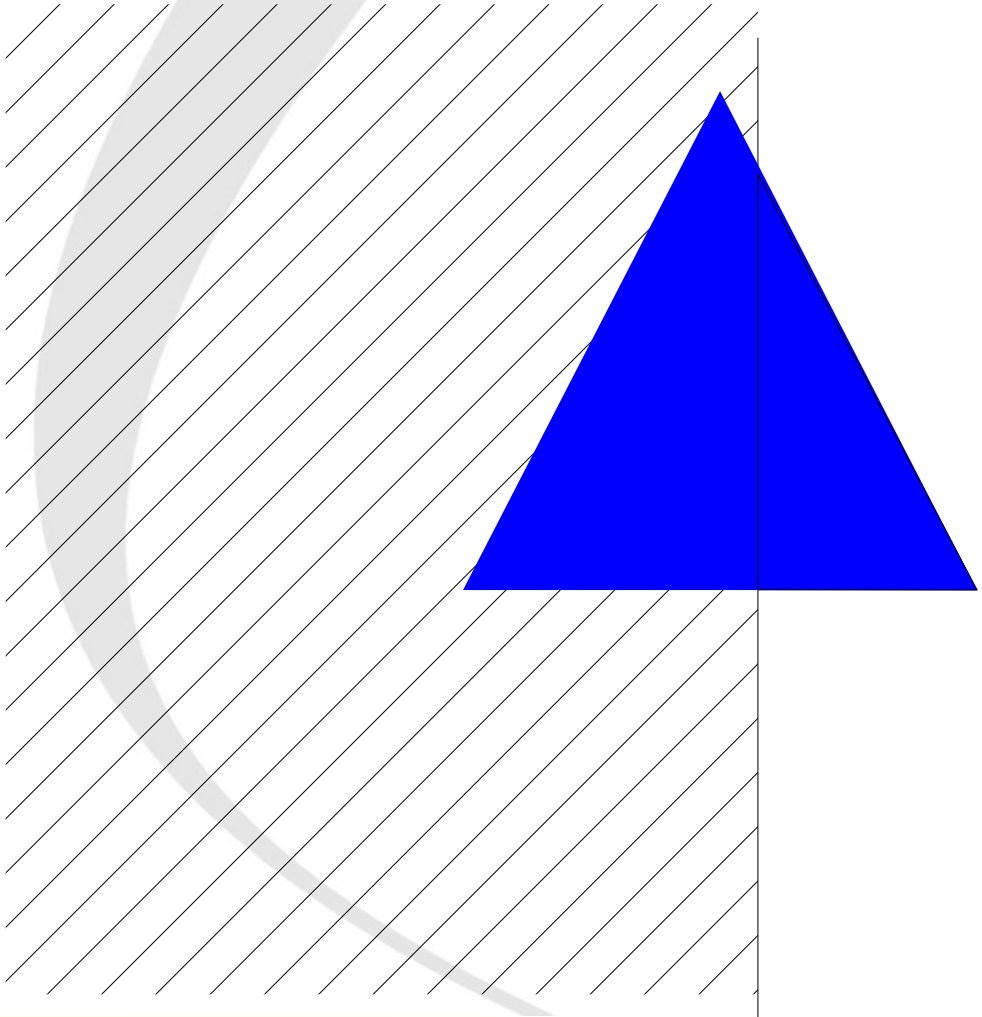
- **Méthode de Sutherland-Hodgeman**
  - Test par rapport à chacun des cotés



Intersection

Triangle

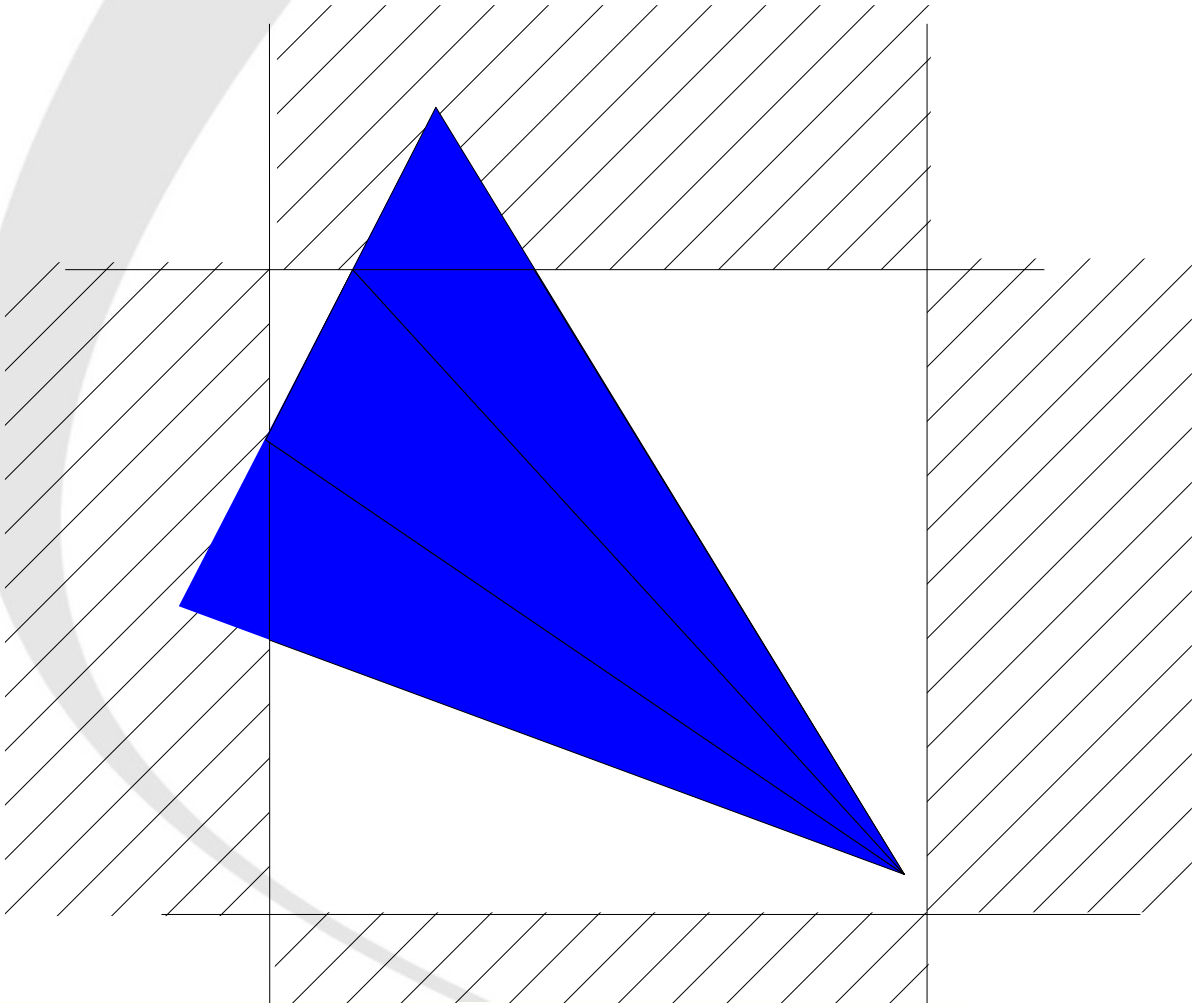
- **Méthode de Sutherland-Hodgeman**
  - Test par rapport à chacun des cotés



Intersection

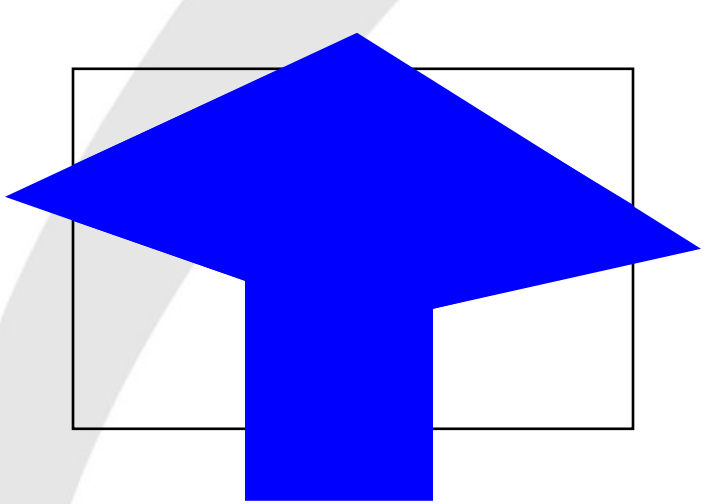
Triangle

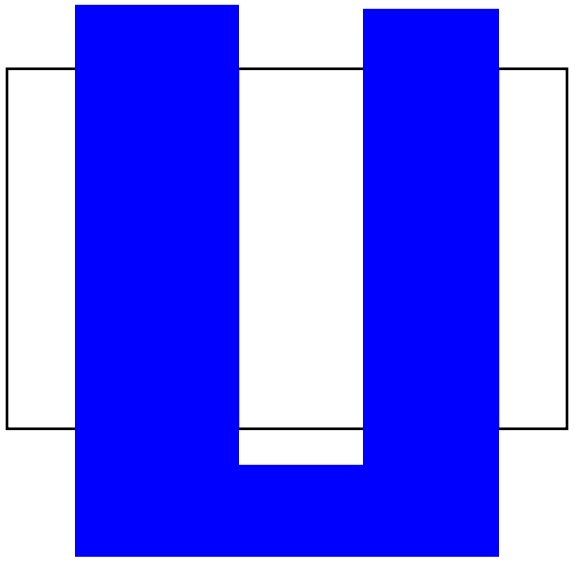
- **Méthode de Sutherland-Hodgeman**
  - Test par rapport à chacun des cotés
  - Itération sur les cotés



Polygones

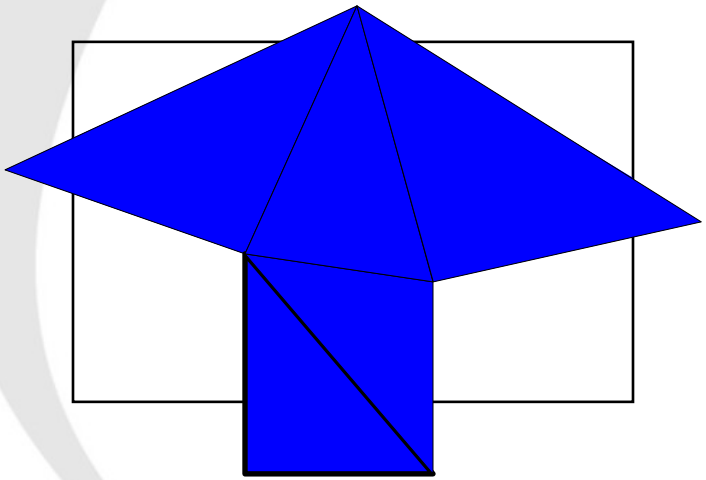
- **Découpage généralisé**

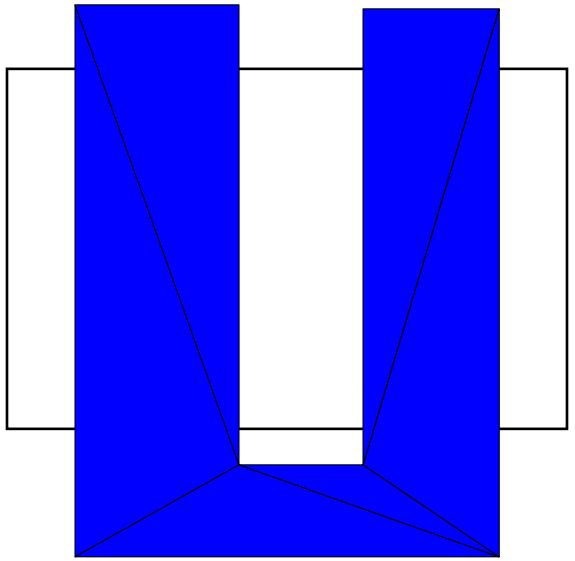




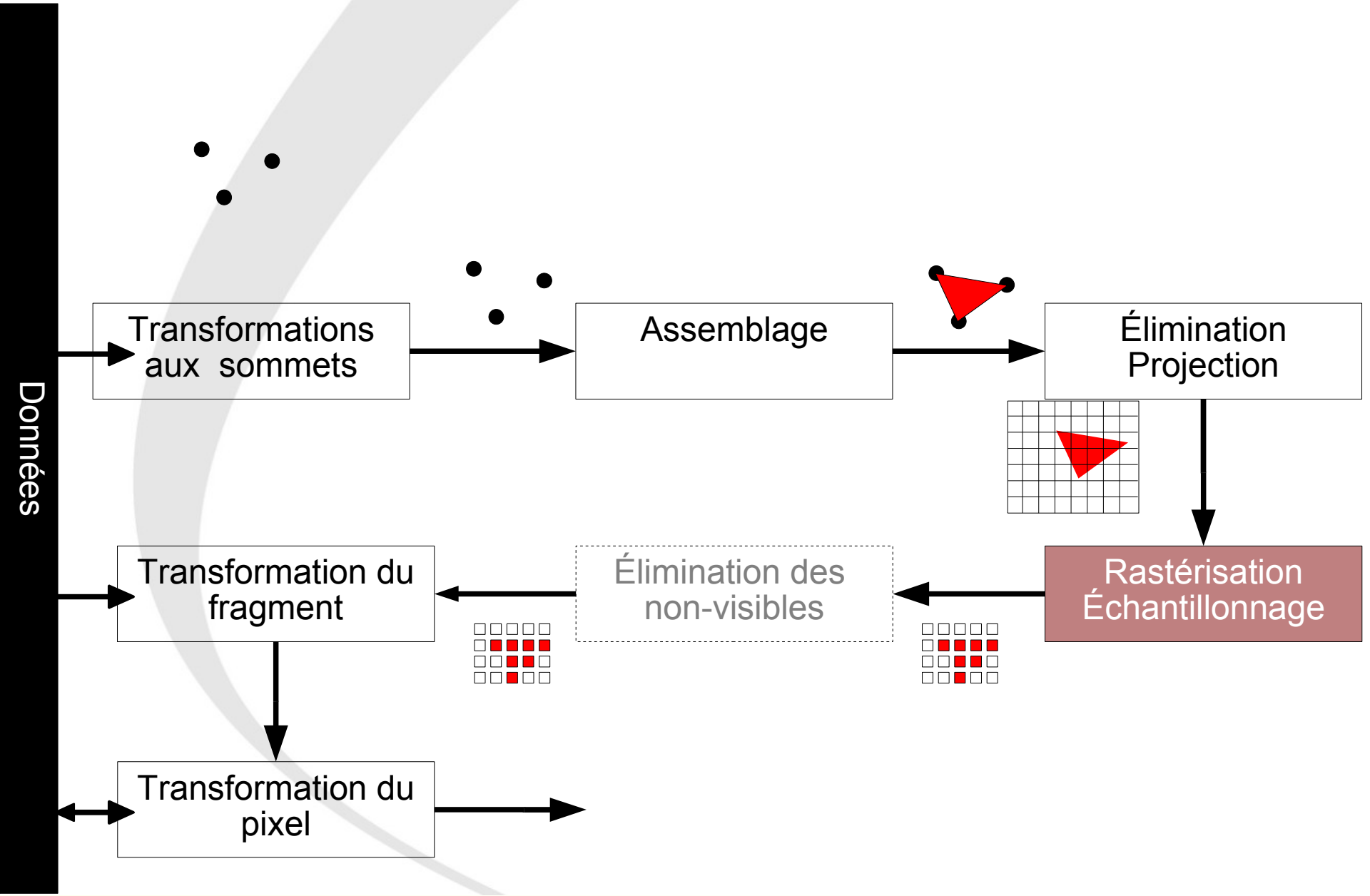
Polygones

- **Découpage généralisé**
- **En général**
  - Triangulation
  - Puis découpage





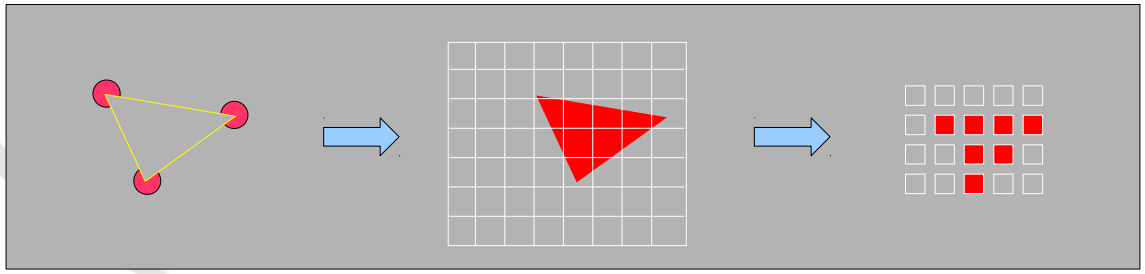
La chaîne de rendu



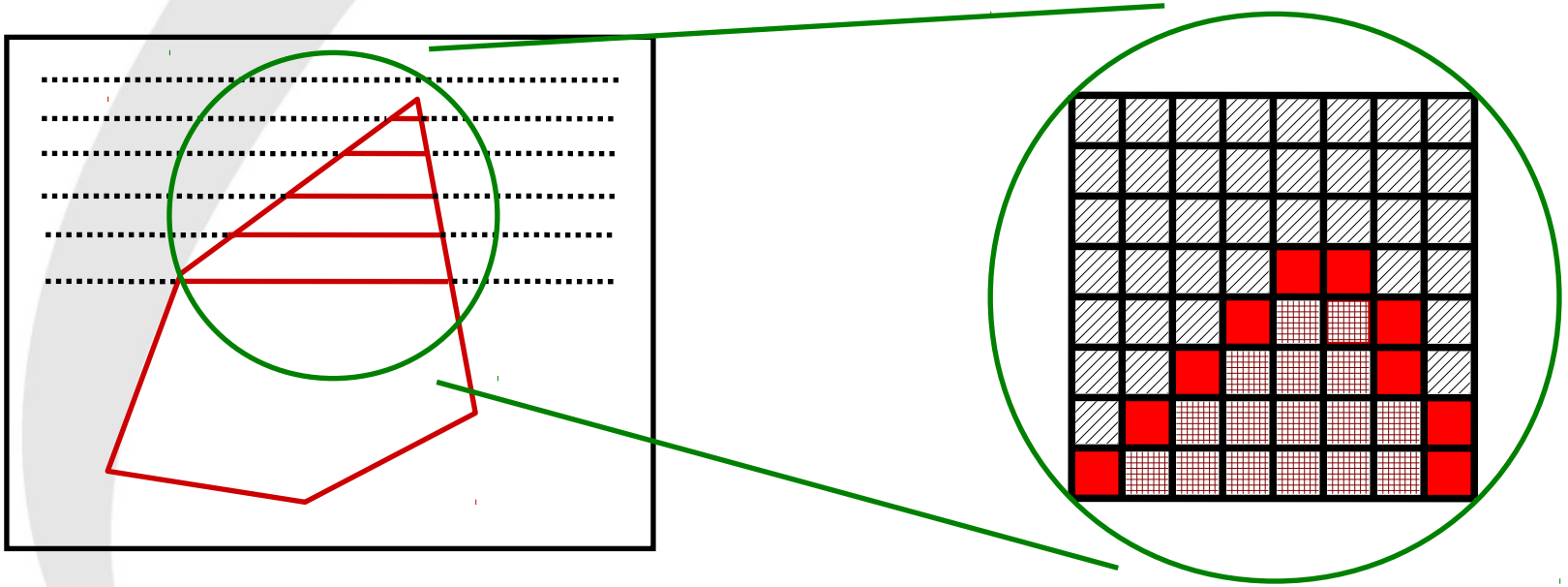
Rastérisation

- **Cette partie comprend :**
  - Culling : élimination des faces cachées
  - Clipping : élimination des faces non visibles
  - Rastérisation : génération des fragments

- **Qu'est-ce que la rastérisation ?**
  - Conversion des triangles en fragments
  - Interpolation des données

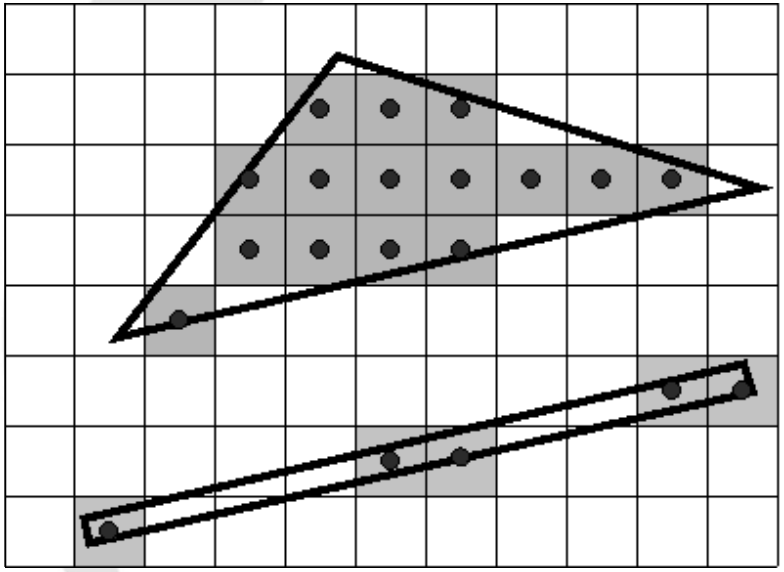


Rastérisation / Bayalage / Scanline

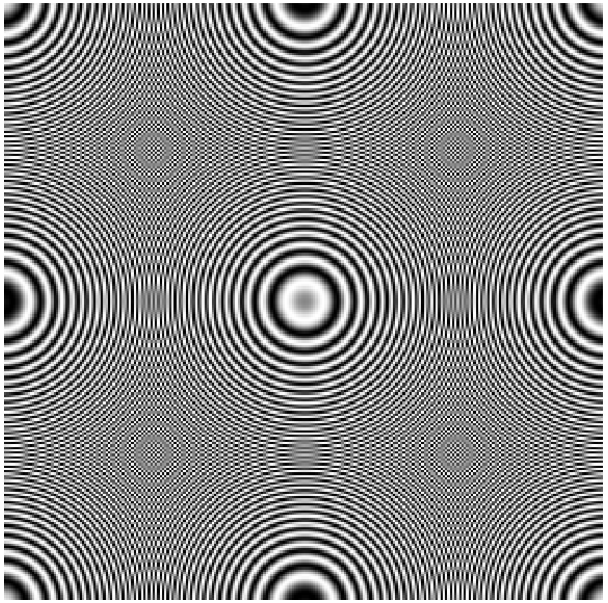


Aliassage

L'aliassage est du à un échantillonnage insuffisant



Trous
Effet d'escalier

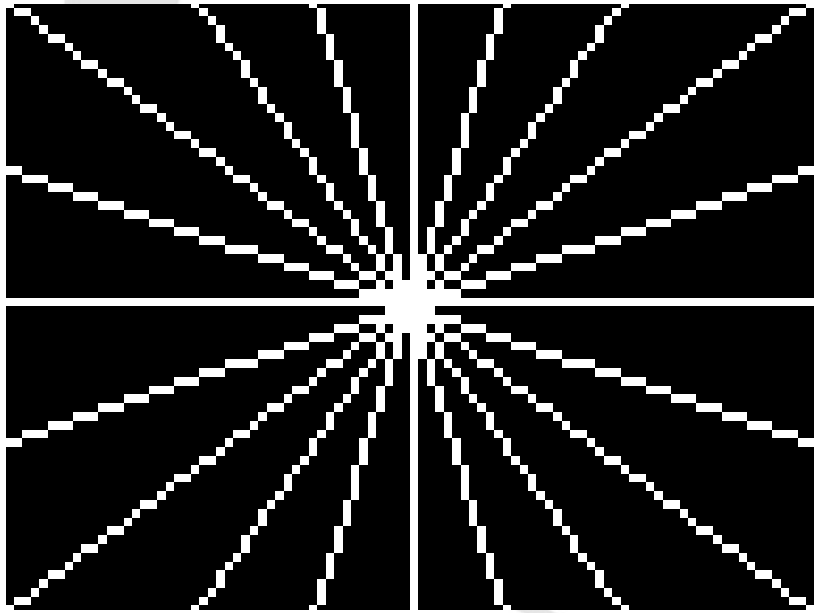


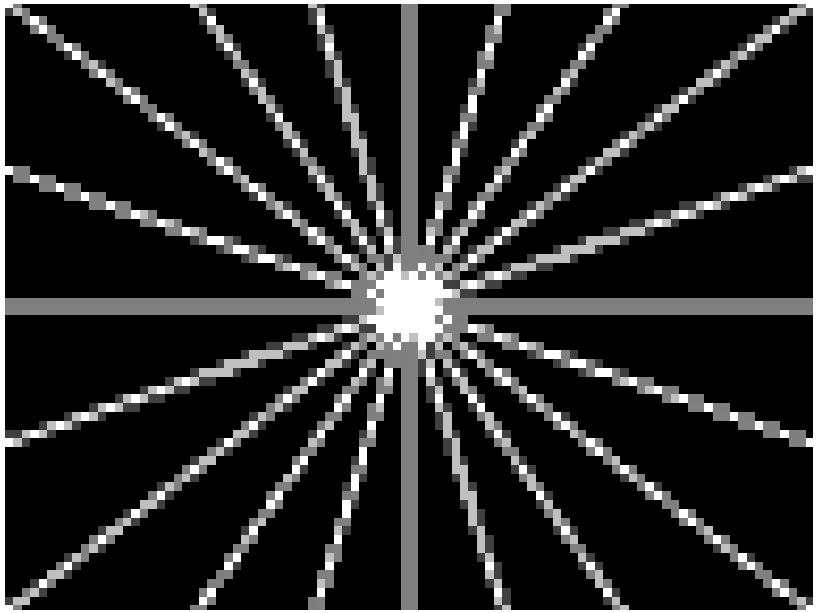
sin(x^2+y^2), x,y dans [-20,20]

Moirés

Anti-aliasing

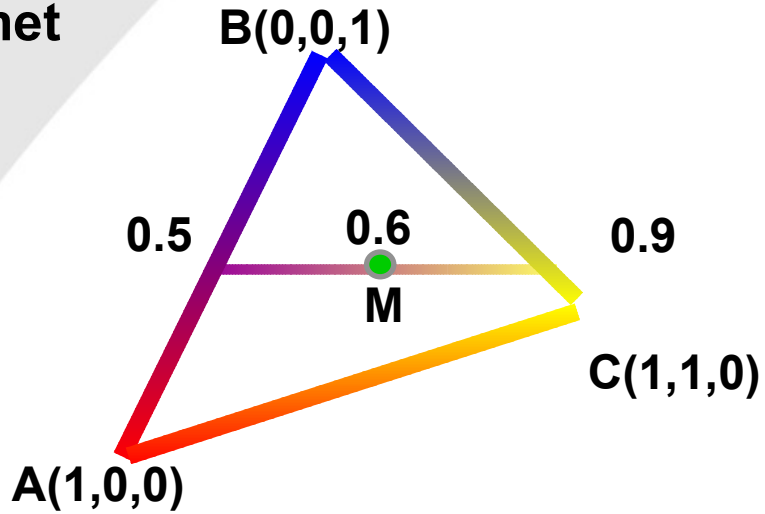
- **Exemple de solution : Sur-échantillonnage**
  - Principe : Plusieurs échantillon par pixel
    - Similaire à calculer une image de haute résolution puis à réduire la taille
  - Avantages : simplicité et généricité
  - Problèmes :
    - Coût de calcul
    - Choix des échantillons



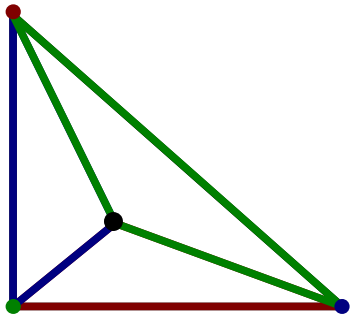


Coordonnés barycentriques

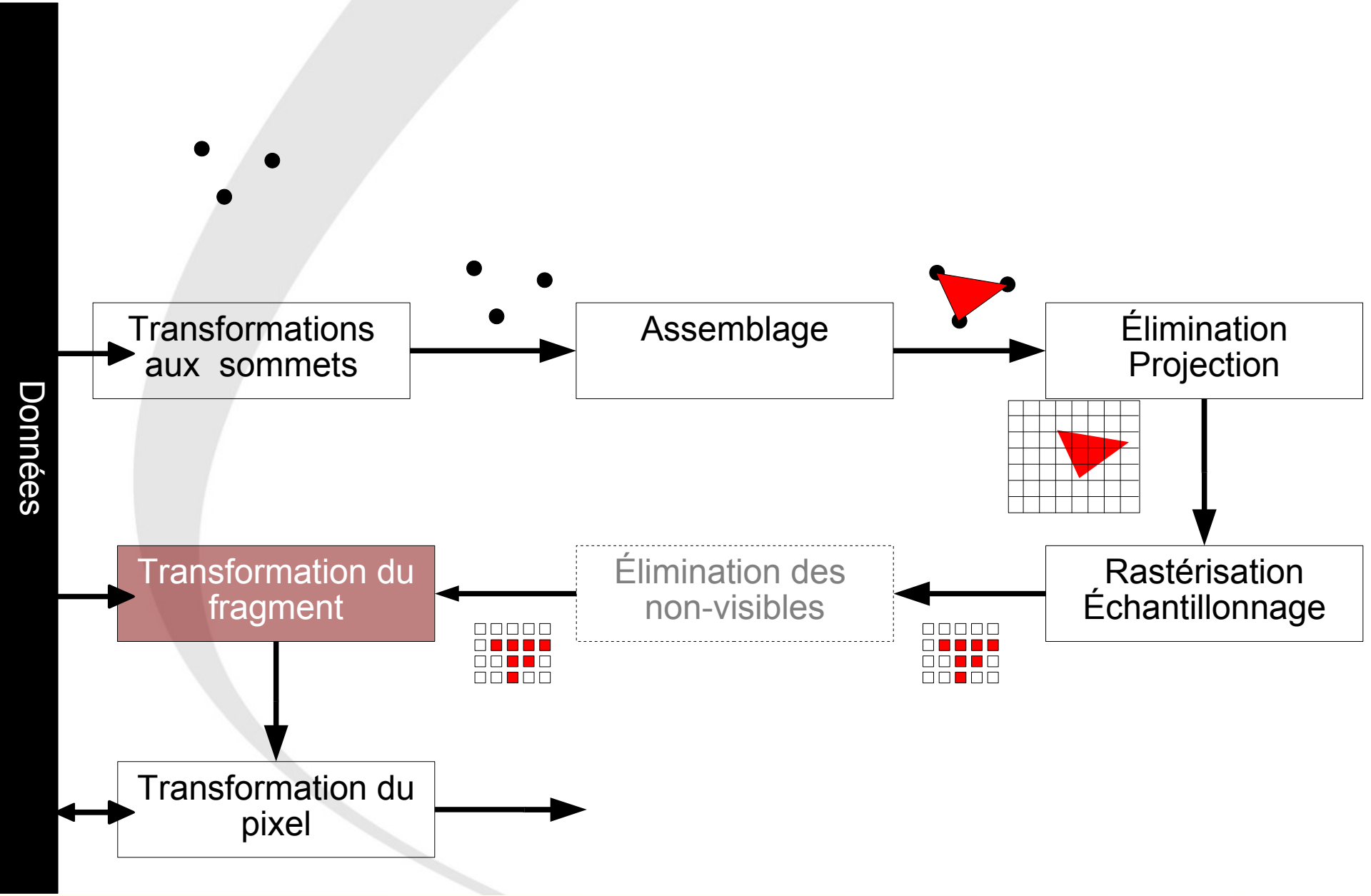
- **Valeur par sommet**
  - Interpolation



- **Coordonnées barycentriques**



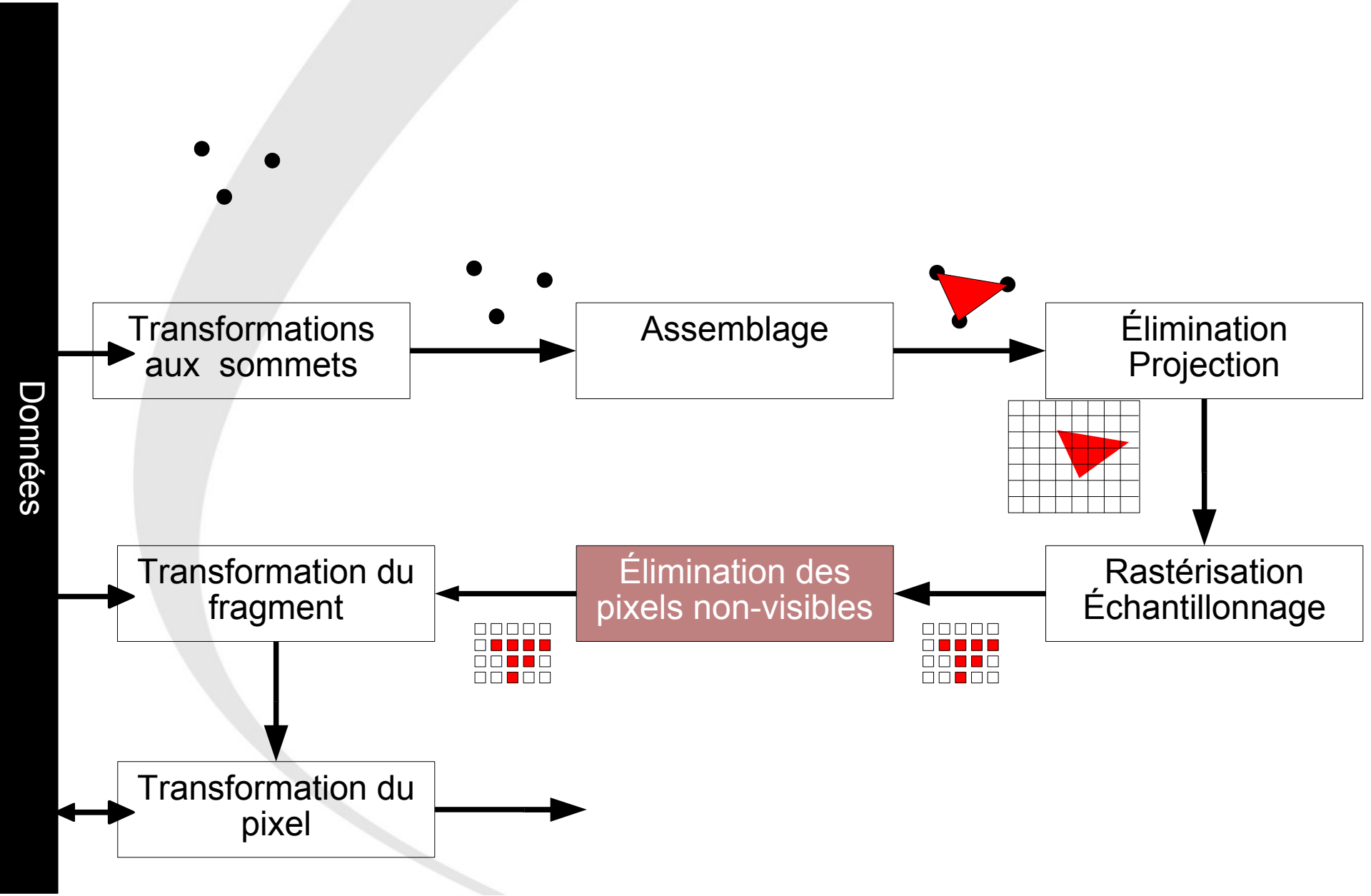
La chaîne de rendu



Opérations sur les fragments

- **Notions absentes**
  - Voisinage
  - Pixel
  - Face/Sommets
- **Données présentes**
  - Variable globales
  - Données issues de la rastérisation
    - Interpolation des valeurs par sommet et leur variation
    - Position, normale, couleur, paramètre d'éclairement, éclairement
- **Utilisation Classique**
  - Éclairement, texturage
  - ....
- **Utilisation Avancée**
  - Trou dans les objets
  - Modification de la profondeur
  - Filtrage d'images
  - ....

La chaîne de rendu



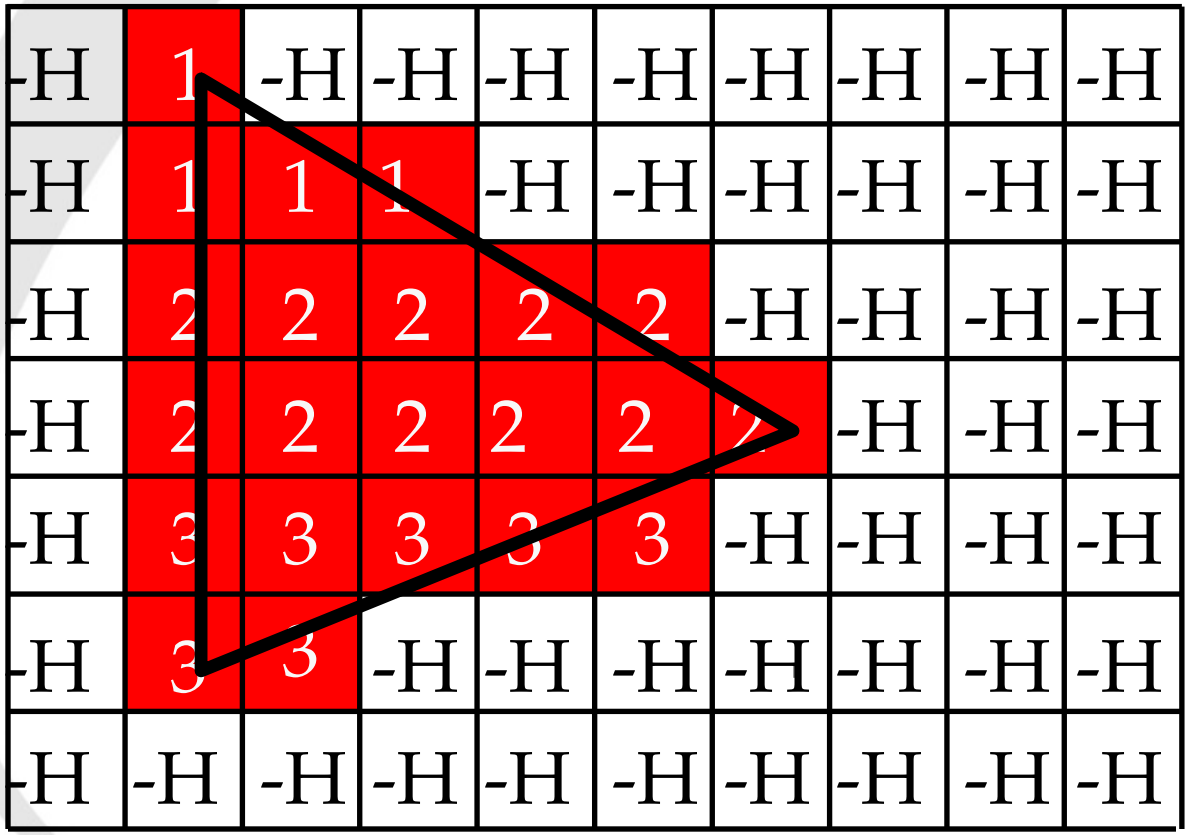
Z-Buffer

- **Un tableau, de la taille de l'écran**
- **On stocke la valeur maximale de z pour chaque pixel**
  - *z* est la direction de visée, exprime la distance à l'oeil
  - z est la distance à l'oeil
- **Initialisation : tous les pixels à moins l'infini**
- **Pour chaque polygone :**
  - Projeter le polygone sur le plan image
  - Pour chaque pixel dans la projection du polygone
    - Calculer la valeur de *z* pour ce pixel
    - Si z est supérieur à la valeur courant de z max
      - Changer z maximal
      - Afficher le pixel à l'écran, de la couleur du polygone

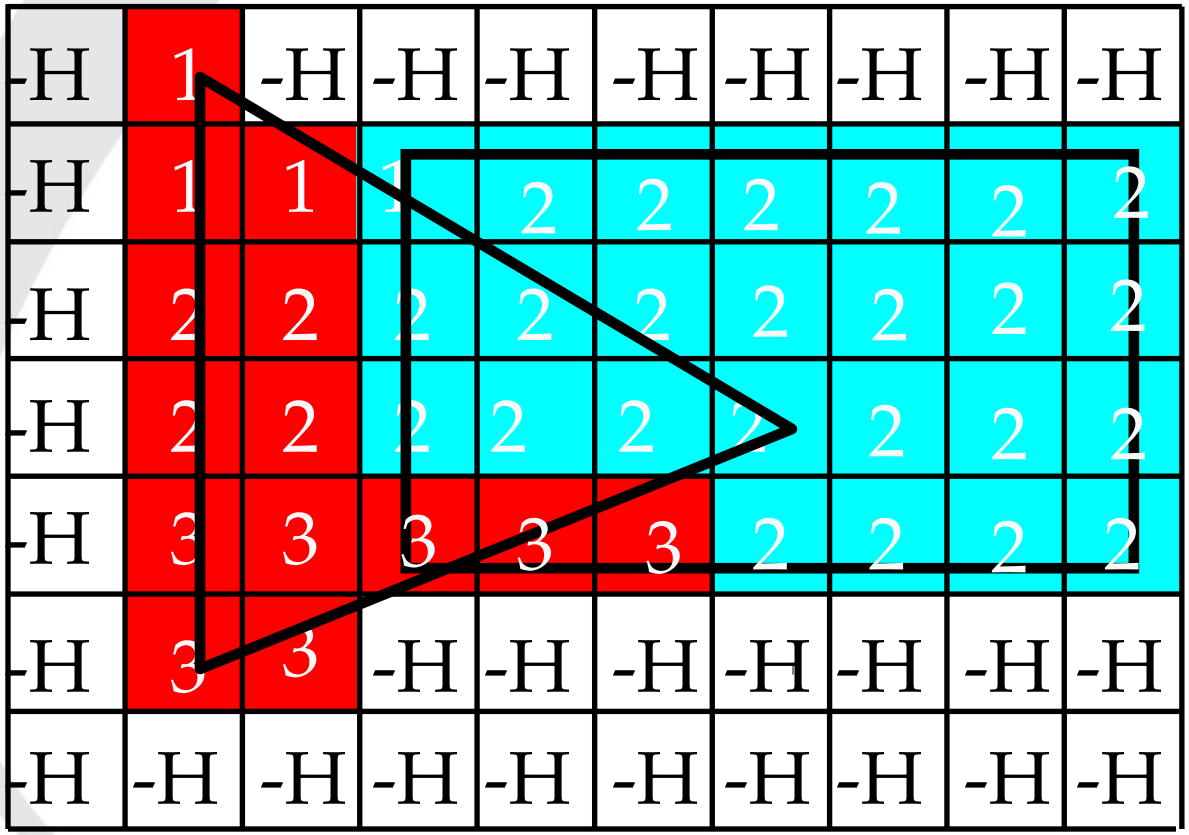
Z-buffer (1)

-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H

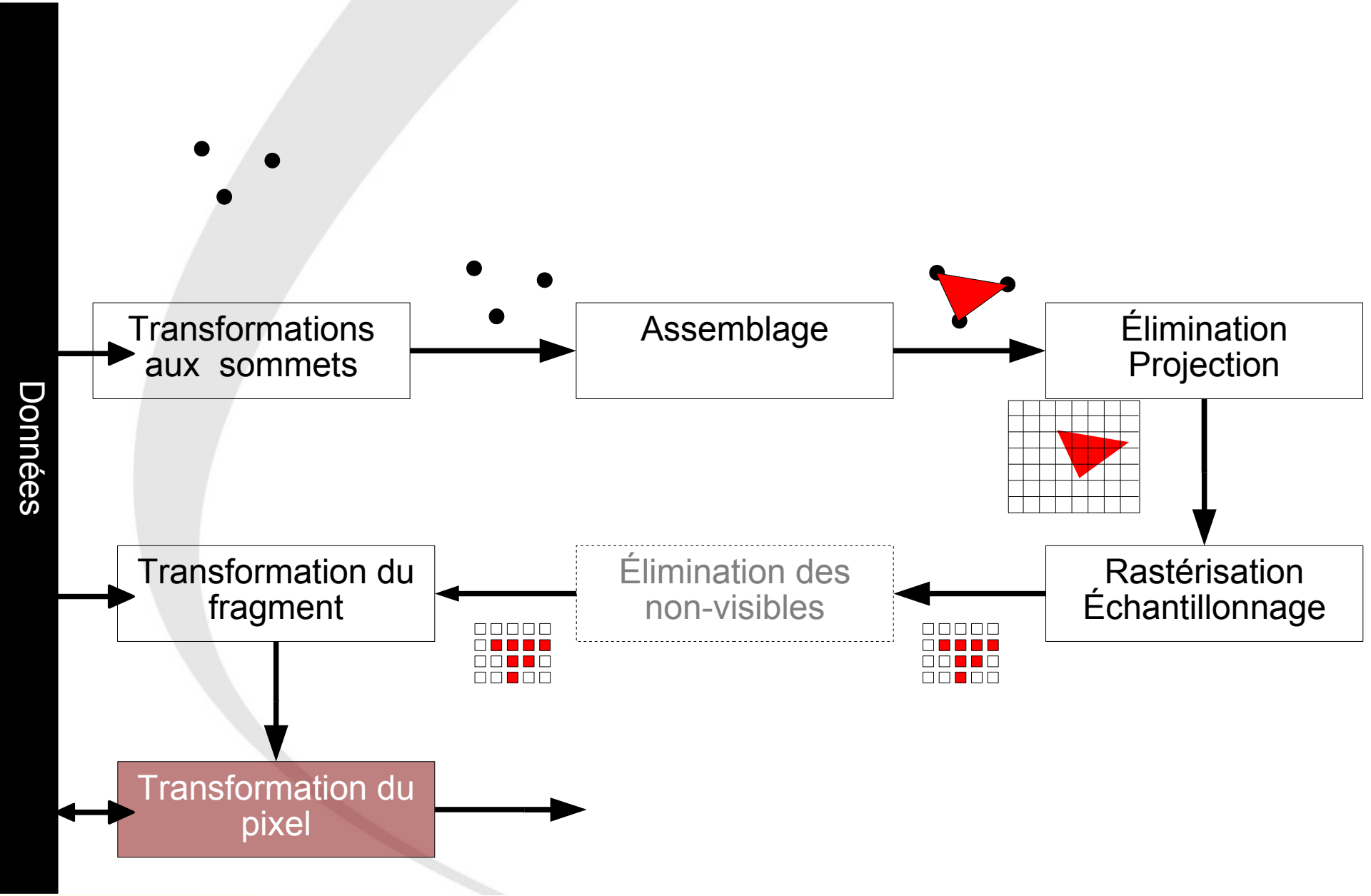
Z-buffer (2)


-H 1 -H -H -H -H -H -H -H -H
-H 1 1 1 -H -H -H -H -H -H
-H 2 2 2 2 2 -H -H -H -H
-H 2 2 2 2 2 2 -H -H -H
-H 3 3 3 3 3 -H -H -H -H
-H 3 3 -H -H -H -H -H -H -H
-H -H -H -H -H -H -H -H -H -H

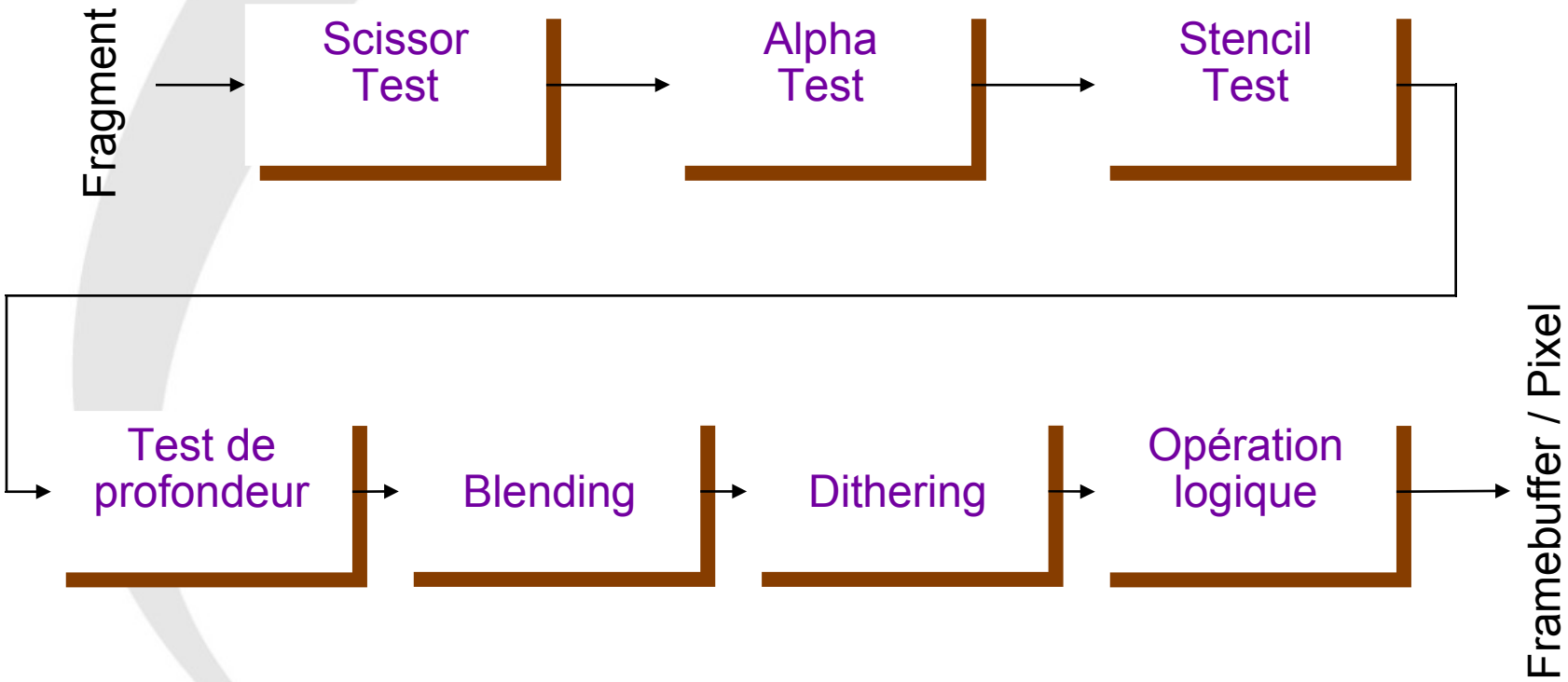
Z-buffer (3)


-H 1 -H -H -H -H -H -H -H -H
-H 1 1 1 2 2 2 2 2 2
-H 2 2 2 2 2 2 2 2 2
-H 2 2 2 2 2 2 2 2 2
-H 3 3 3 3 3 2 2 2 2
-H 3 3 -H -H -H -H -H -H -H
-H -H -H -H -H -H -H -H -H -H

La chaîne de rendu

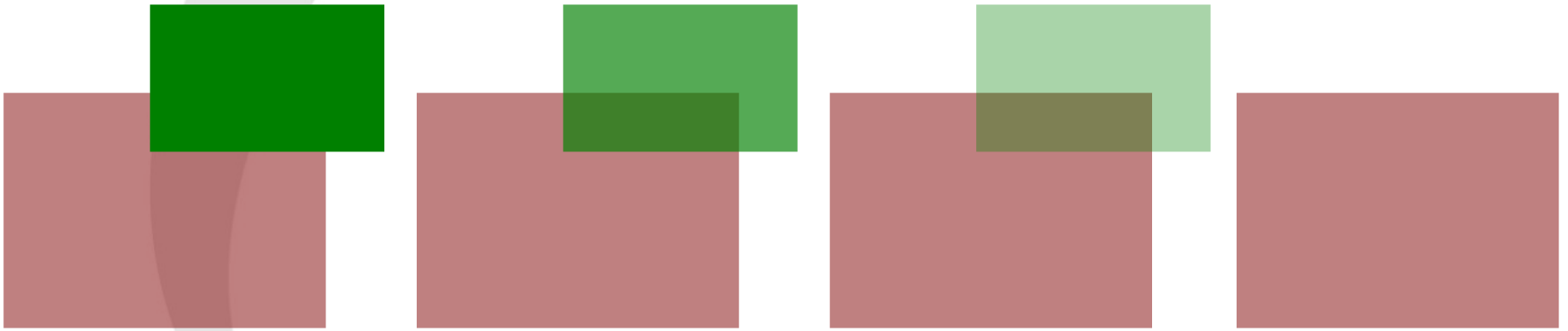


Operations sur les pixels



Transparence

- **Canal Alpha**
  - 1 = complètement opaque
  - 0 = complètement transparent
  - $C = C_{courant} * A_{courant} + (1 - A_{courant}) * C_{précédente}$

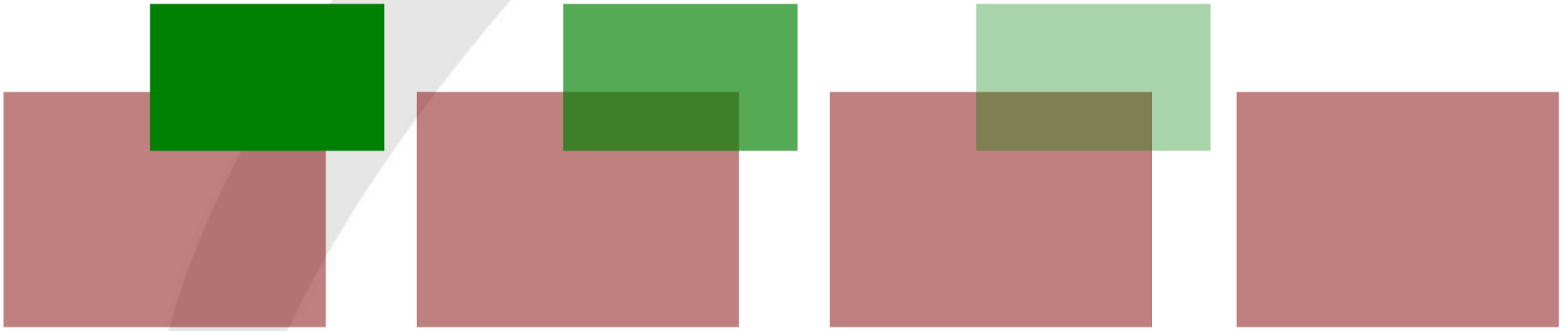


A = 1 A = 0,66 A = 0,33 A = 0

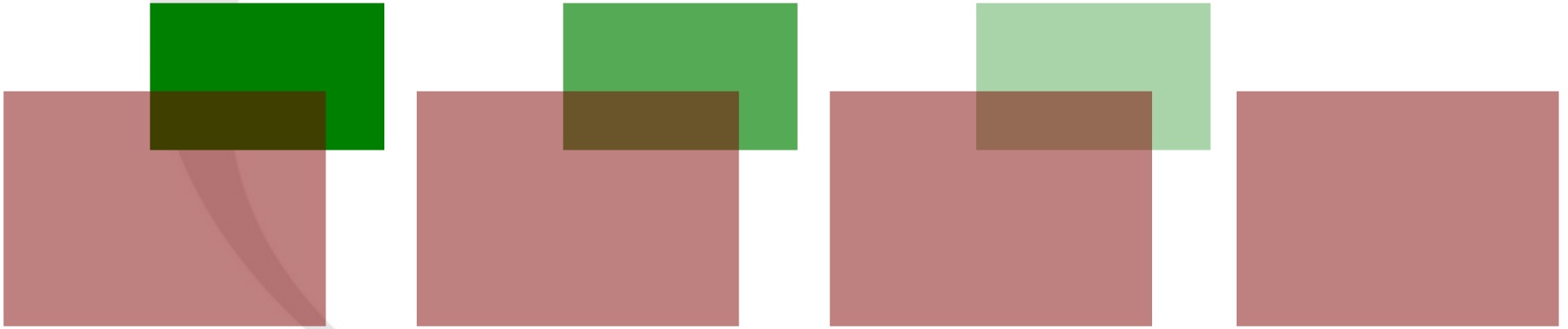
A = 0,5

Problème d'ordonnancement

- Quadrilatère Rouge puis Vert



- Quadrilatère Vert puis Rouge

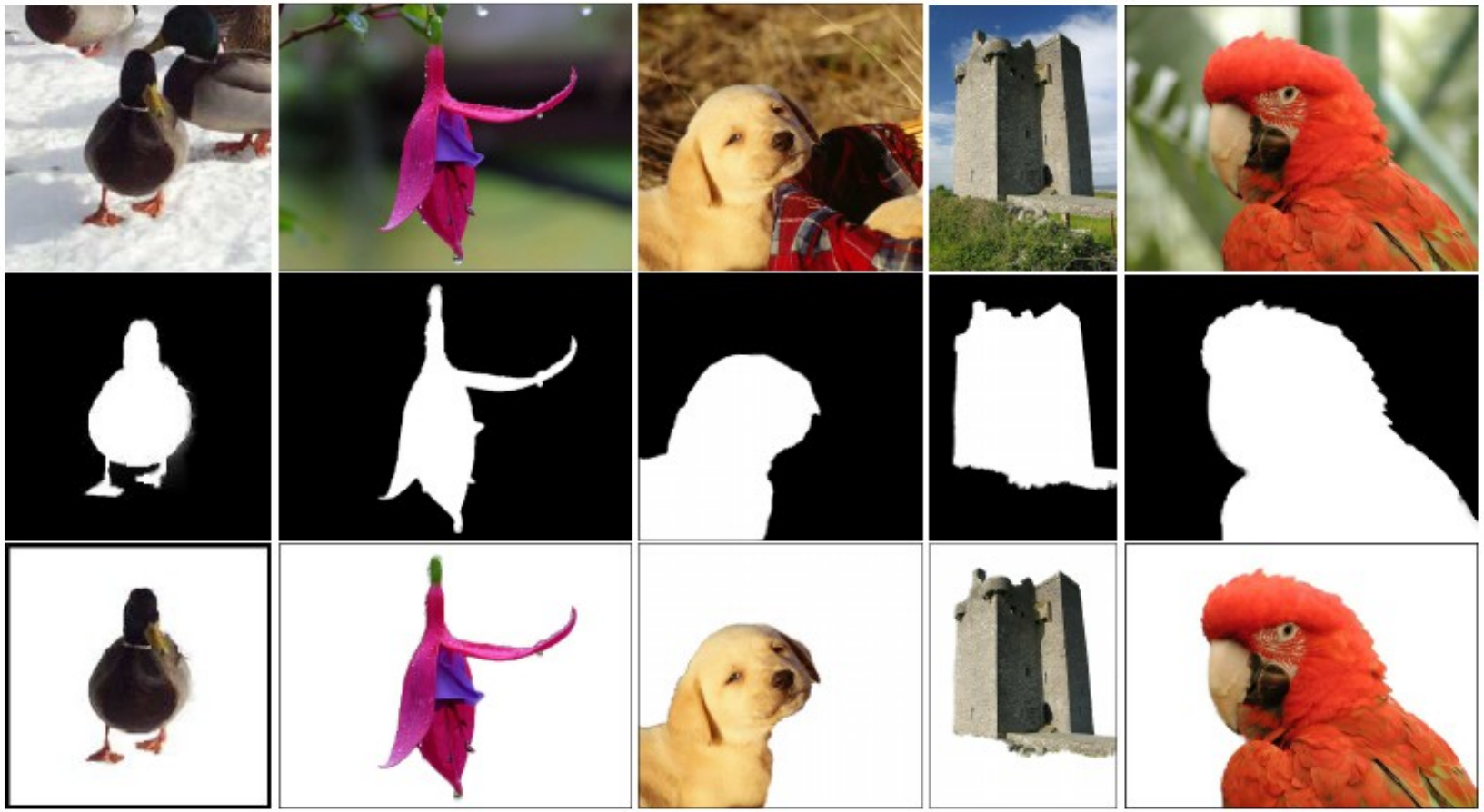


Composition (Compositing)

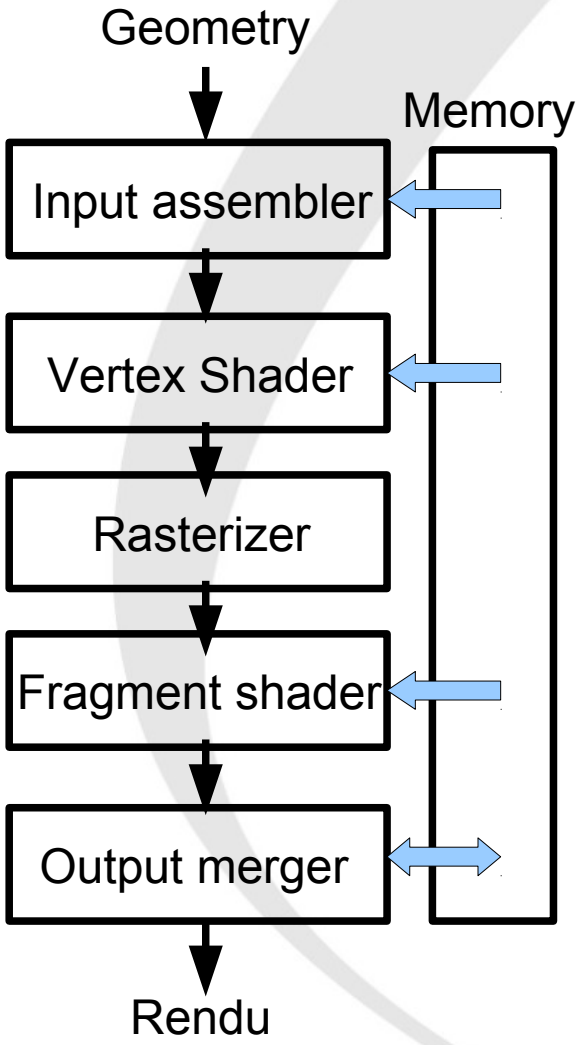
- **Masques**
  - Par exemple le Stencil
  - Permet de restreindre les zones d'affichages

Composition (Compositing)

- **Alpha Matte (calque alpha)**
  - Stencil = Limites dures => aliassage
  - Limites douces => transitions douces



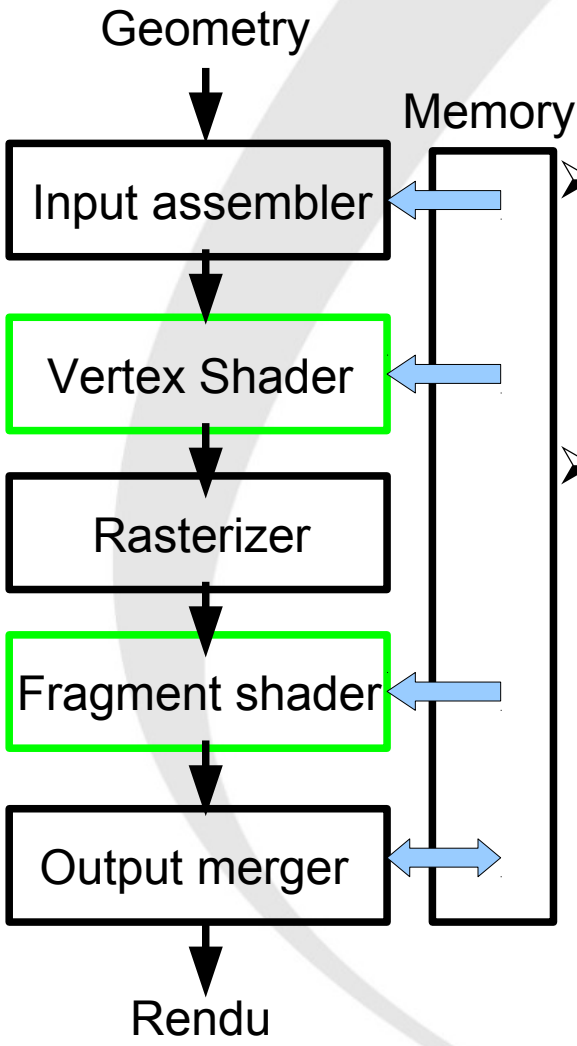
Pipeline classique



Pipeline fixe !

- Choix d'un état OpenGL
  - Géométrie
  - Lumière
  - Test de profondeur
  - Blending
  - Culling
  - …
- Et toutes ces opérations sont réalisées...
  - Codées en hardware dans la carte graphique (GPU)
  - Pas d'interaction possible avec ces choix prédéfinis

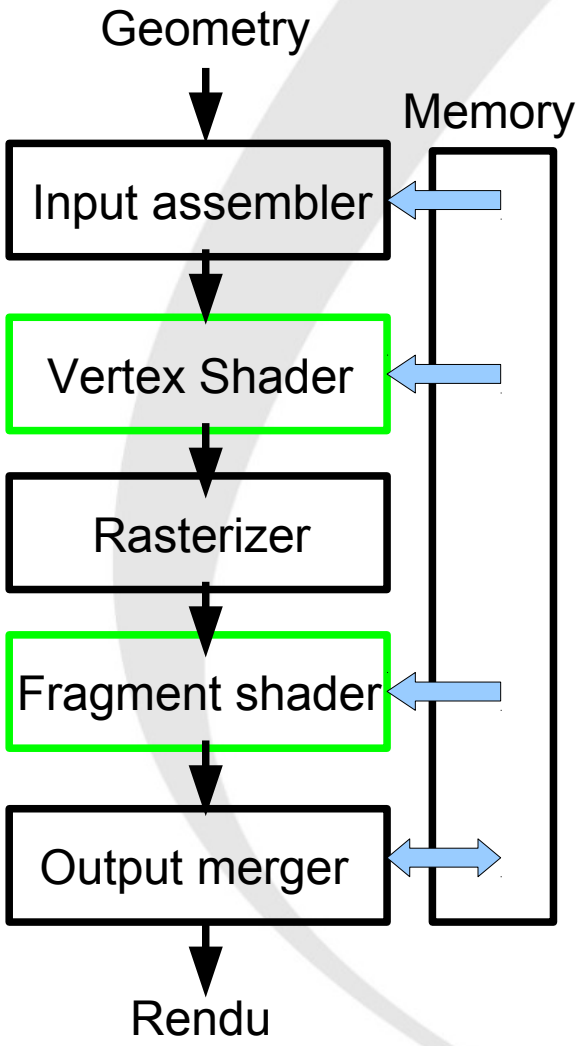
Pipeline dynamique



Depuis DirectX 9 / OpenGL 2.0

- Des opérations sont programmables sur le GPU
  - Vertex shader
  - Fragment shader
- Comment programmer sur le GPU ?
  - Avec des languages de programmation spécifiques
    - assembleur
    - glsl (OpenGL Shading Langage)
    - hlsl (High Level Shading Langage)
    - cg (C for Graphics)
  - Avec des liens entre CPU et GPU
    - DirectX et OpenGL possèdent des fonctions spécifiques

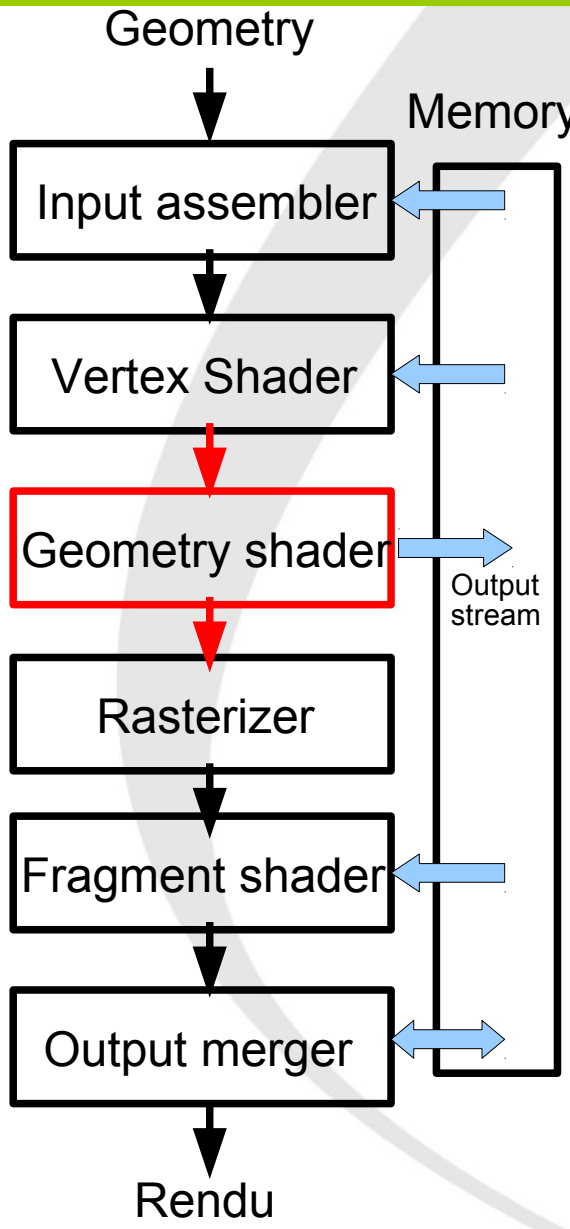
Pipeline dynamique



Depuis DirectX 9 / OpenGL 2.0

- Fini les opérations prédéfinies
- Opérations personnalisées par sommet
  - Scale / rotation / translation
  - Toute sorte de projection
  - Choix des variables à interpoler lors de la rastérisation
  - …
- Opérations personnalisées par fragment
  - Utilisation simplifiée des textures
  - Toute fonction possible sur l'image (par pixel)
  - HDR imaging
  - Multi-passe
  - …

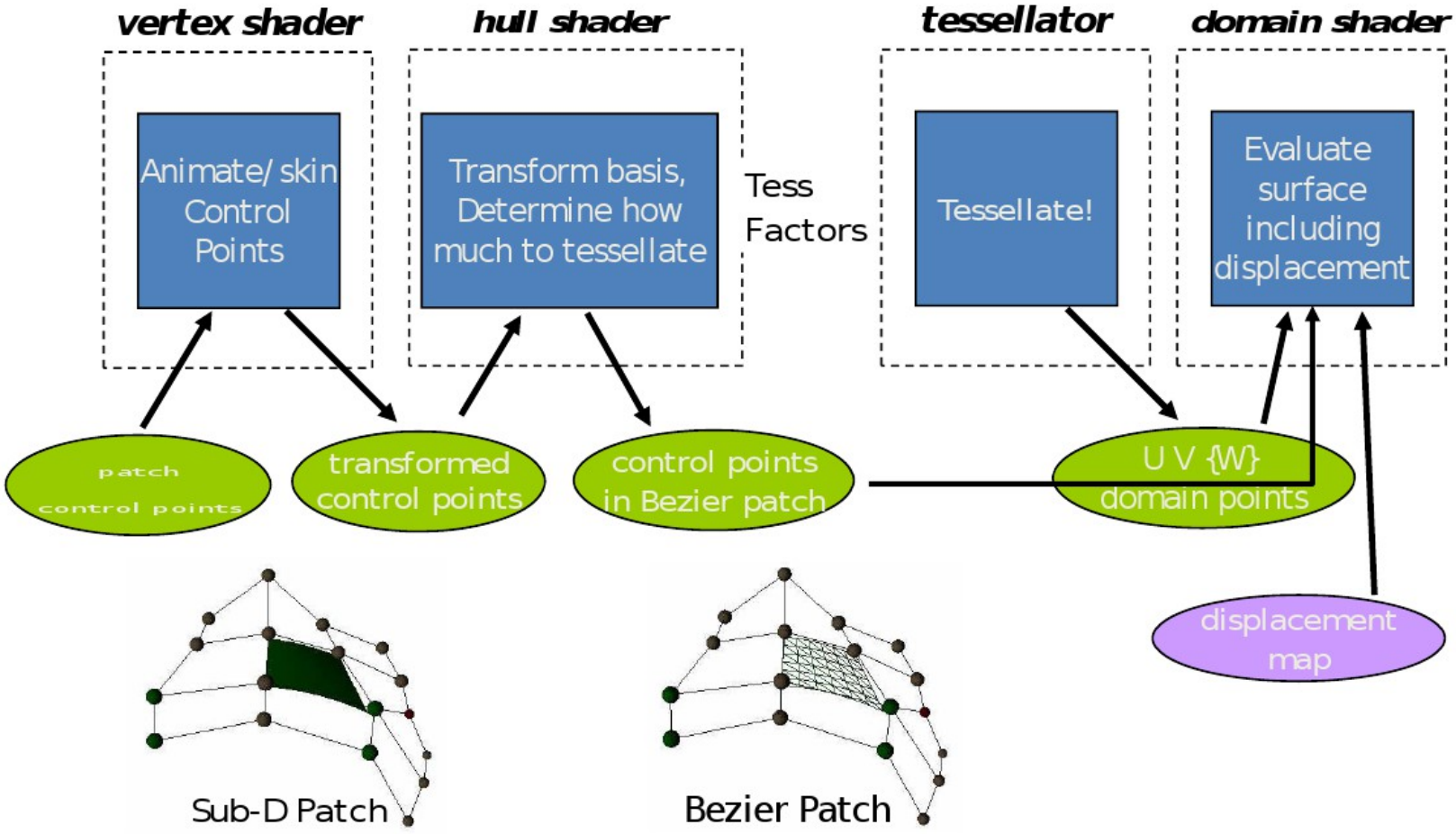
Pipeline dynamique



Depuis DirectX 10 / OpenGL 2.0 + EXT

- Une nouvelle étape programmable dans le pipe :
  - Géometry shader
- Opération par primitive complète (triangle)
  - Génération / suppression de primitives (à la volée)
  - LODs
  - Retours possible vers l'input assembler

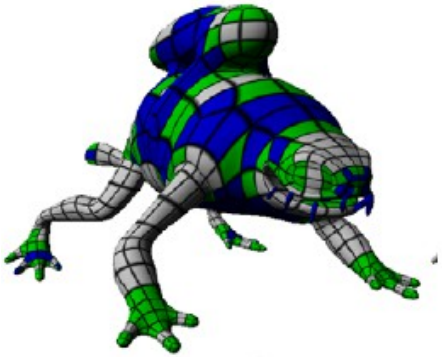
Pipeline dynamique



La primitive de base n'est plus le triangle mais le patch !

Pipeline dynamique

Sub-D Modeling



Animation

Displacement Map

GPU

Optimally Tessellated Mesh

Quelques effets connus

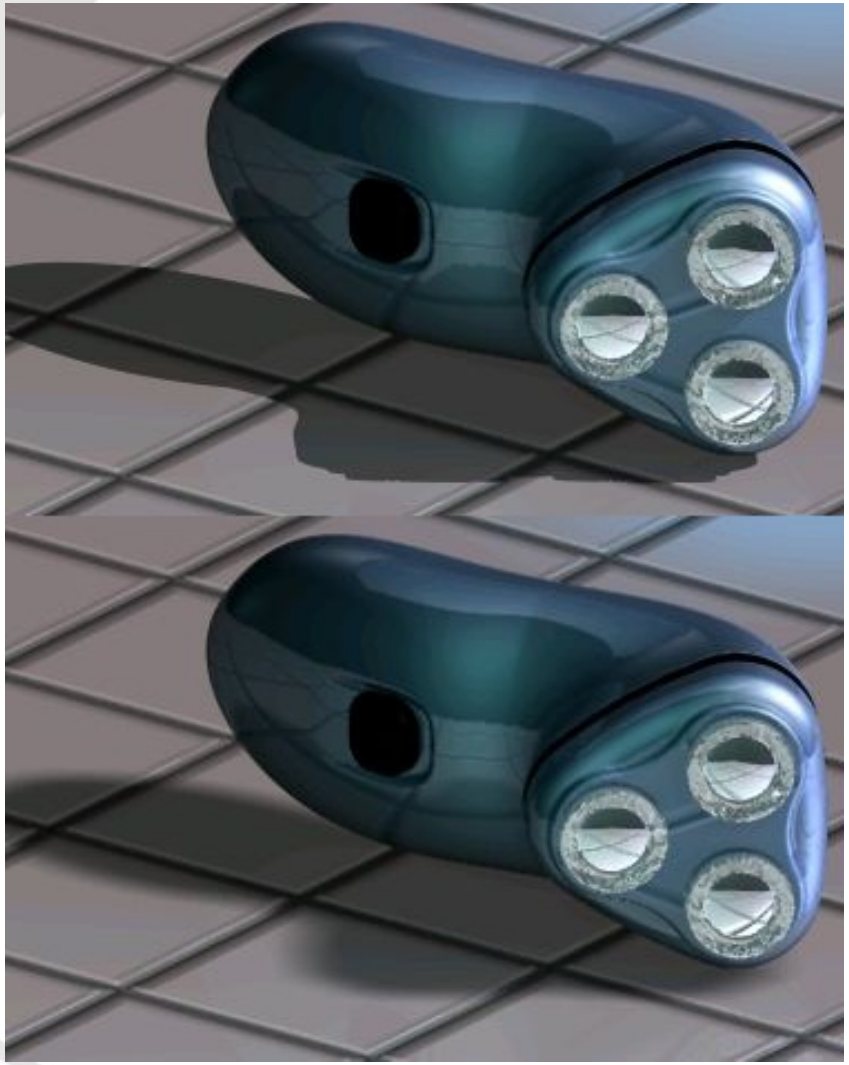
Bump mapping

Quelques effets connus

Depth of field effect

Quelques effets connus

Soft shadows

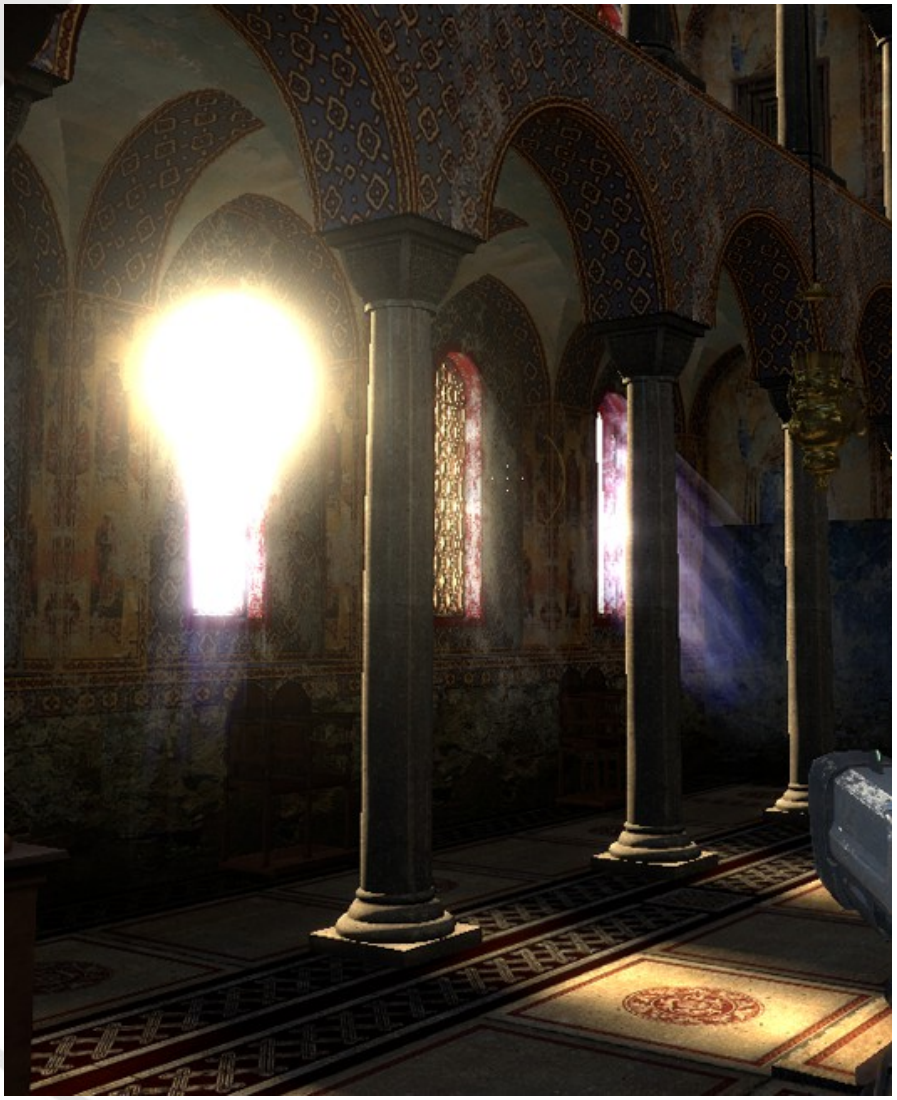


Quelques effets connus

Motion blur

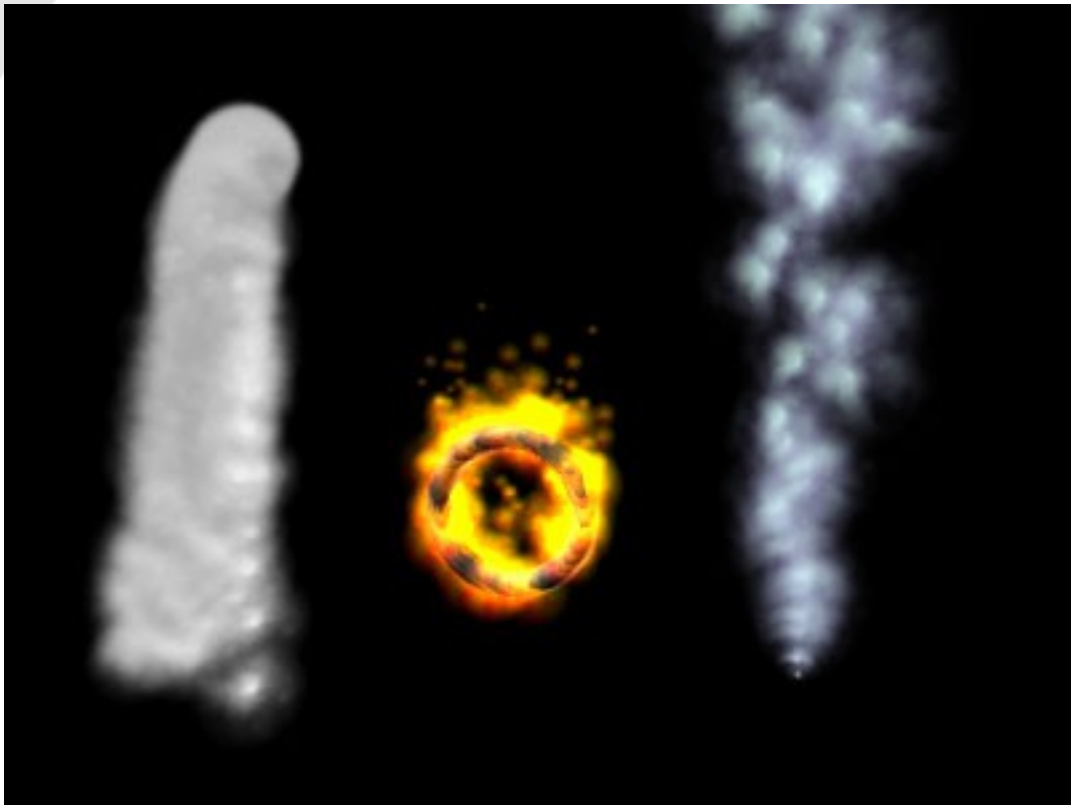
Quelques effets connus

HDR effects



Quelques effets connus

Systèmes de particules



Quelques effets connus

Cell-shading

Quelques effets connus

Fourrures