Pipeline Graphique Géométrie et Affichage

Graphics Programming

Eric Cannet

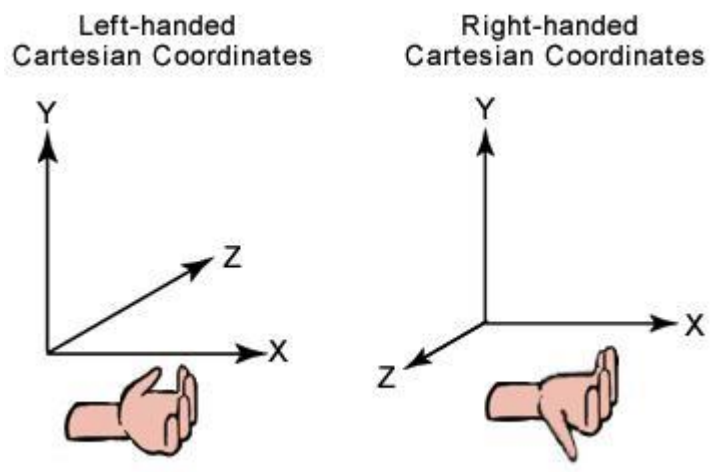
But de ce cours

- Les transformations dans l'espace
- La géométrie en 3D et l'affichage
- Le pipeline graphique

Transformation dans l'espace

Je suis perdu

- Comment on sait comment sont les axes ?
  - L'axe vertical, représentant la hauteur(altitude), dépend du vecteur de up dans la matrice de camera
  - Ensuite il y a deux type de repère
  - On choisi au moment où on créer les matrices



Unité

- Unité: combien 1.0f vaut de mètre ?
- Il n'y a pas d'unité. Tout est une question de proportionnalité.
- C'est à vous de définir votre propre unité.
- Mais attention à la précision d'un float

Les trois mousquetaires

- Translation
  - Sert à positionner, déplacer un objet
  - Représenté par un vecteur à 3 composantes
- Rotation
  - Sert à faire tourner un objet autour des axes.
  - Représenté par un quaternion, une matrice, un angle et un axe
- Redimensionnement (scale)
  - Sert à agrandir, rétrécir les objets
  - Représenté par un vecteur à 3 composantes

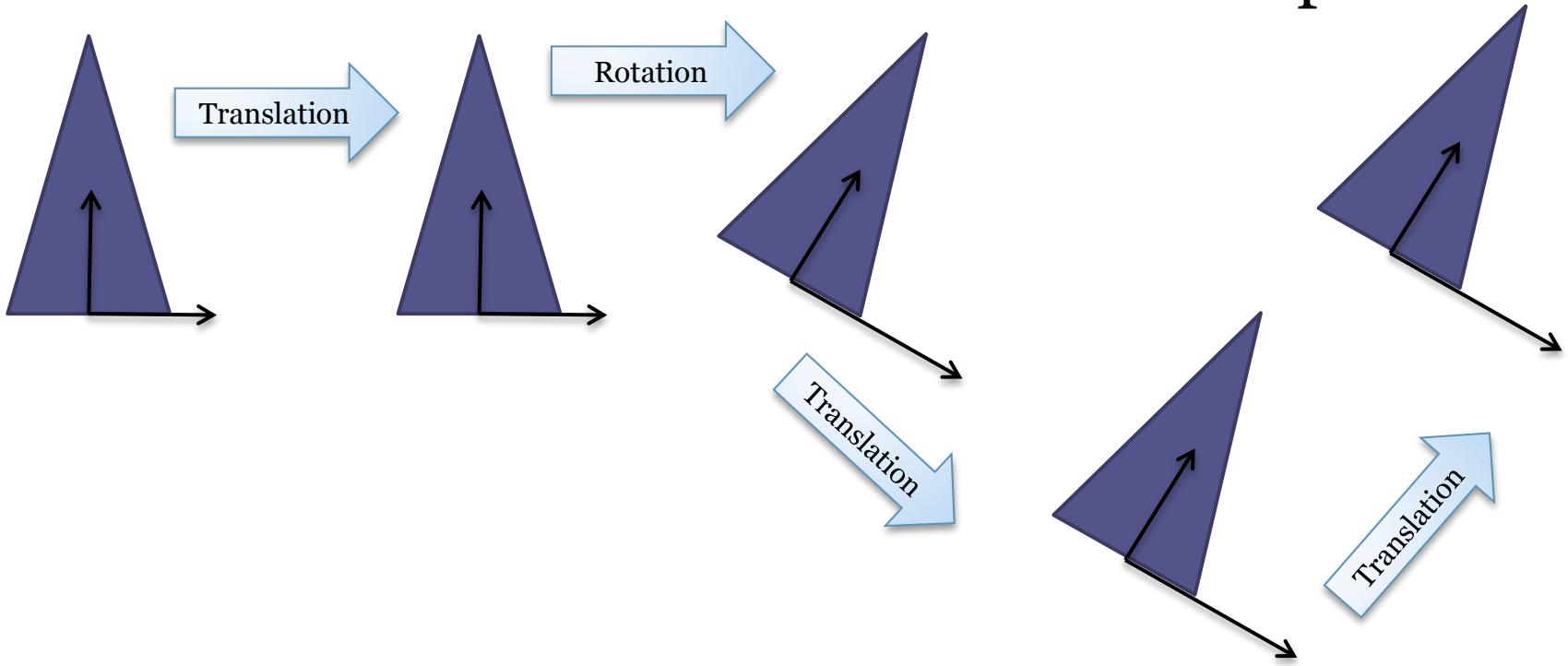
D’artagnan

- Quand on combine les trois on représente la transformation sous forme de matrice

Rotation et Scale	Rotation	Rotation	0
Rotation	Rotation et Scale	Rotation	0
Rotation	Rotation	Rotation et Scale	0
Translation	Translation	Translation	1

Becareful

- La multiplication de matrices n'est pas commutatif
- L'ordre des transformations est donc importante



Camera

- La camera correspond au point depuis lequel on voit la scène.

- Est en général définit par:
  - La position de la camera
  - Le point que regarde la camera
  - Un vecteur qui définit vers où est le haut

- On parle de matrice LookAt

Axe, plus t'en met plus t'en as

- Si vous avez besoin des axes de la camera vous pouvez les retrouver dans la matrice comme ceci:

Axe X x	Axe Y x	Axe Z x	0
Axe X y	Axe Y y	Axe Z y	0
Axe X z	Axe Y z	Axe Z z	0
-Dot(AxeX, Eye)	-Dot(Axey, Eye)	-Dot(Axez, Eye)	1

Quand on est en main gauche

Projection …

- Une dernière matrice, la projection:
  - Elle définit comment la géométrie sera projetée à l'écran

- FovY: Angle de champs de vision en y
- Aspect: Taille en X / taille en Y (4:3, 16:9)

- Ces deux là auraient pu suffire dans un monde parfait, mais on ne peut faire une projection infinie

…de sangs

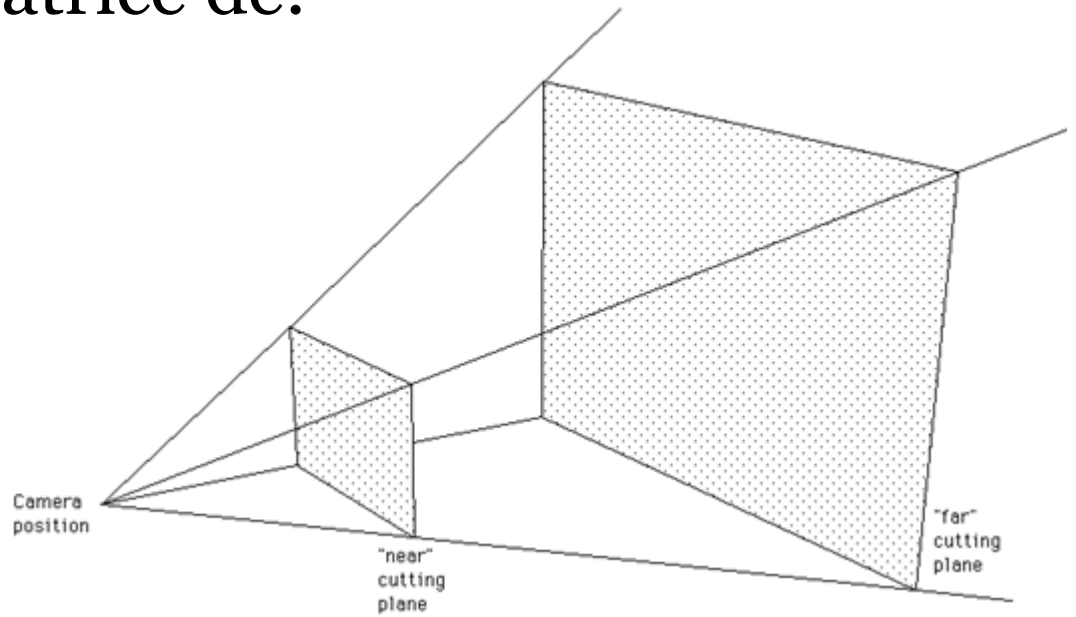
- Z near : plus petite valeur en z qui sera affichée
- Z far : plus grande valeur en z qui sera affichée

- Les valeurs de Z une fois projetées seront toujours entre 0 et 1.
  Il faut donc les choisir au plus juste de notre scène.
- C'est non linéaire.

Frustum

- Il s'agit d'une pyramide avec le haut coupé qui définit le champ de vision de la caméra

- Dépends de la matrice de:
  - View/Camera
  - Projection

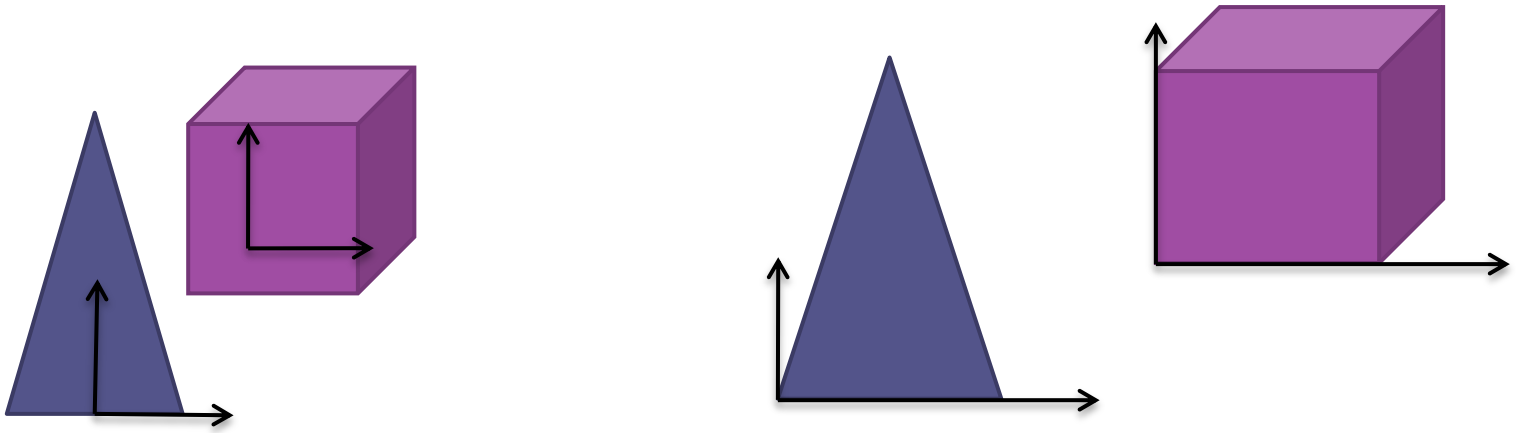


Frustum culling

- Le frustum culling est le fait de ne pas afficher les géométries en dehors du frustum.
- Il est préférable de ne pas « tenter » d'afficher quelque chose que de laisser la carte graphique l'éliminer
- Le frustum est défini correspond à 6 plans

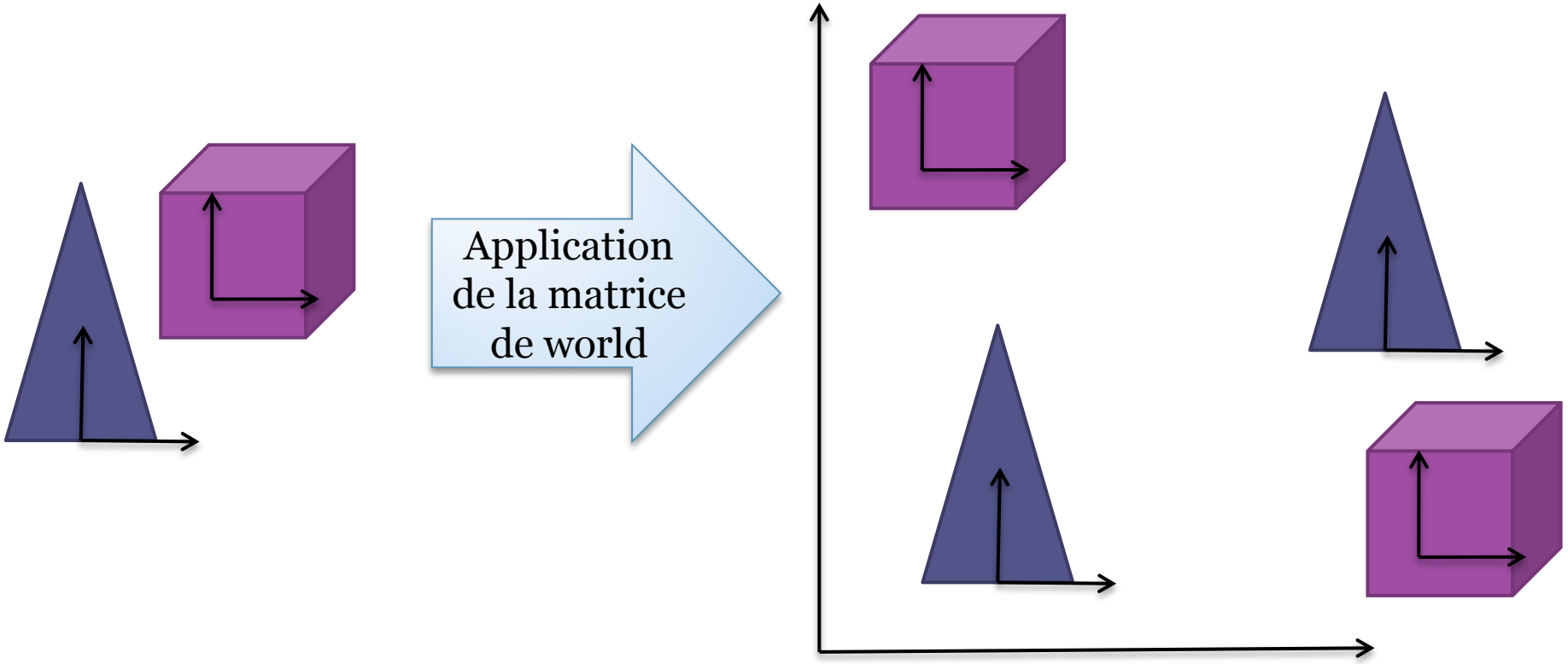
Local

- Chaque objet a son repère local
- Quand il a été fait dans un logiciel de modélisions les points on été placés par rapport au (0,0,0)



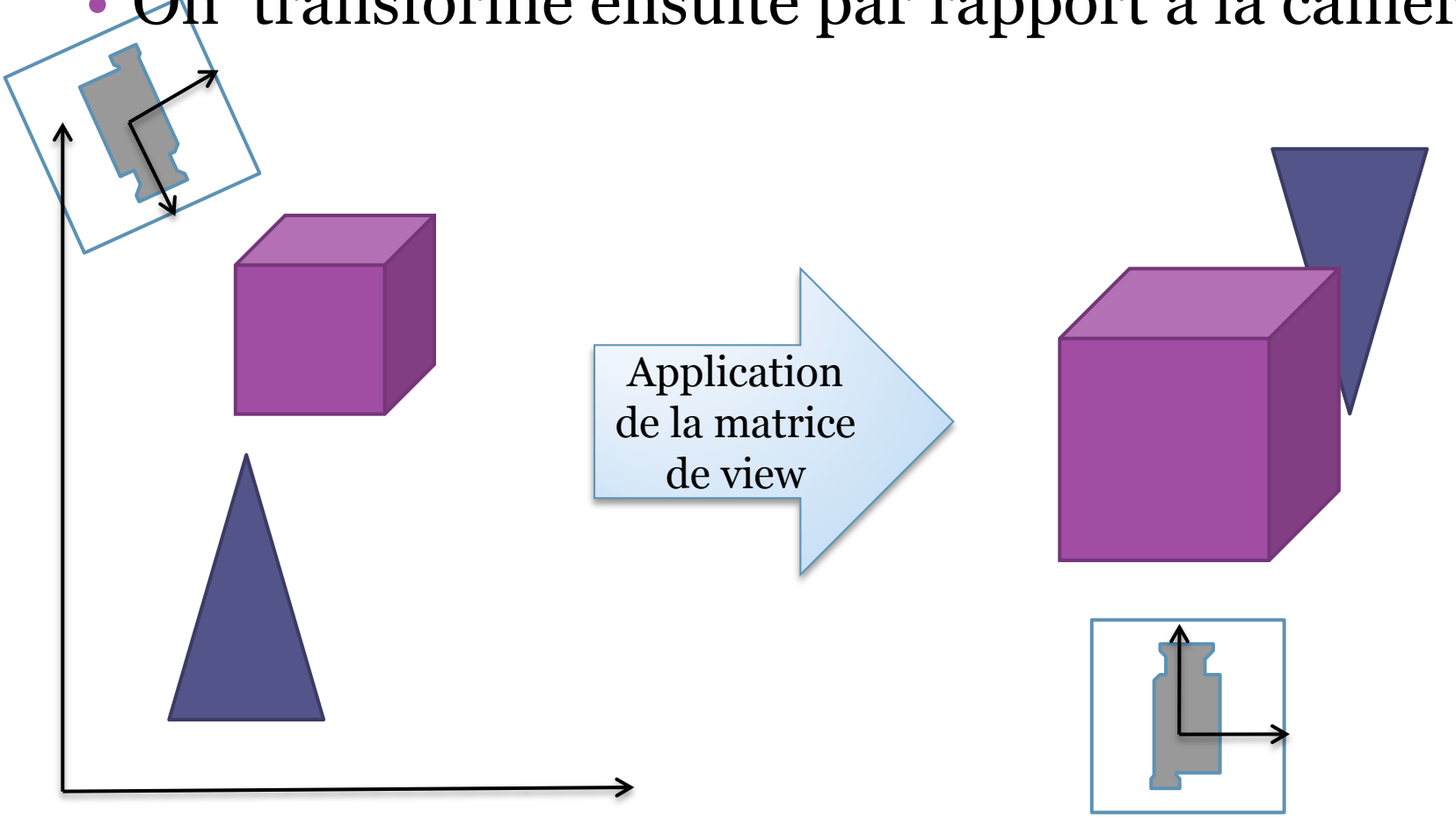
World

- Les objets sont placer pour composer la scène comme on le souhaite



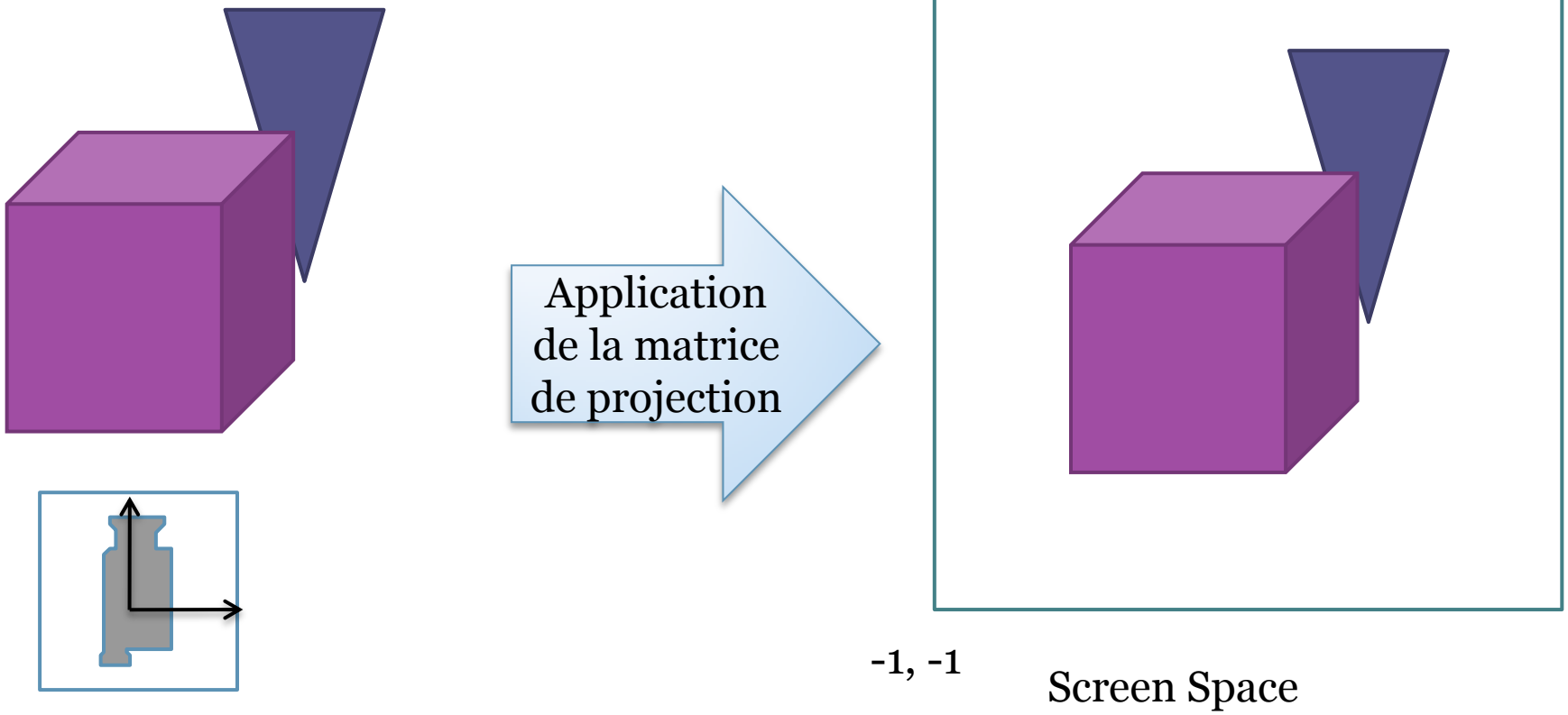
View

- On transforme ensuite par rapport à la camera

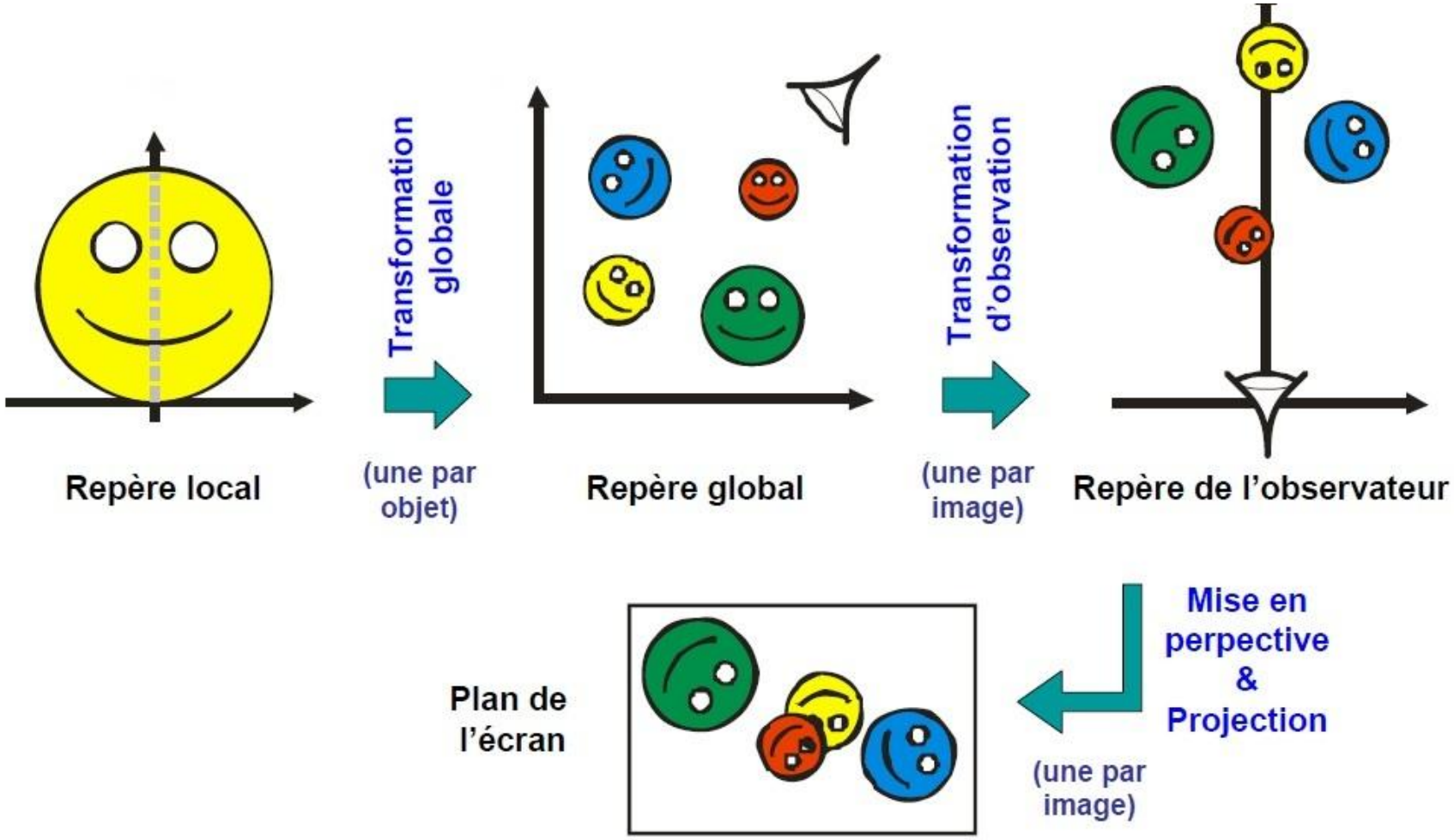


Proj

- Il nous reste plus qu'à projeter à l'écran



On récapitule: WorldViewProj

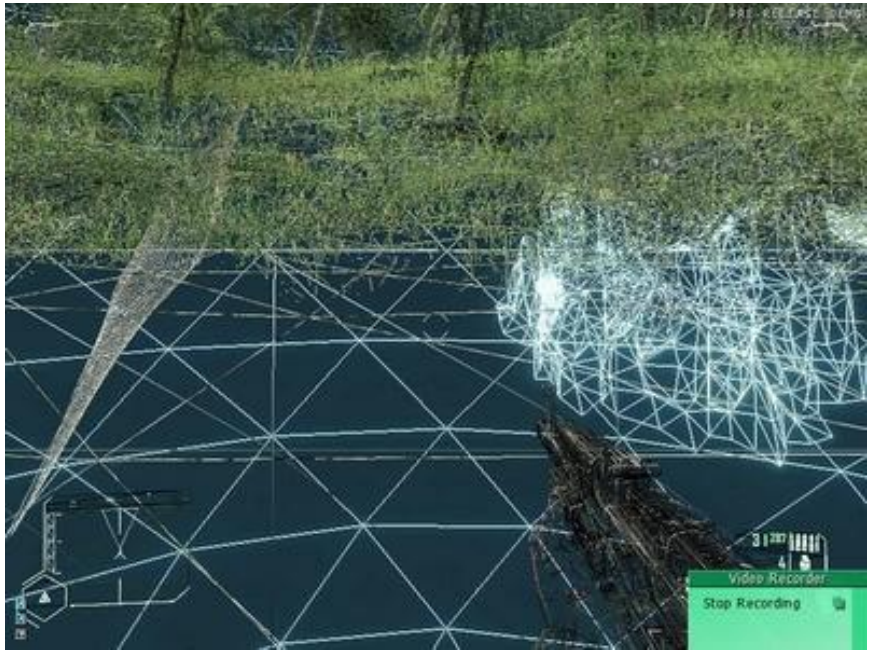


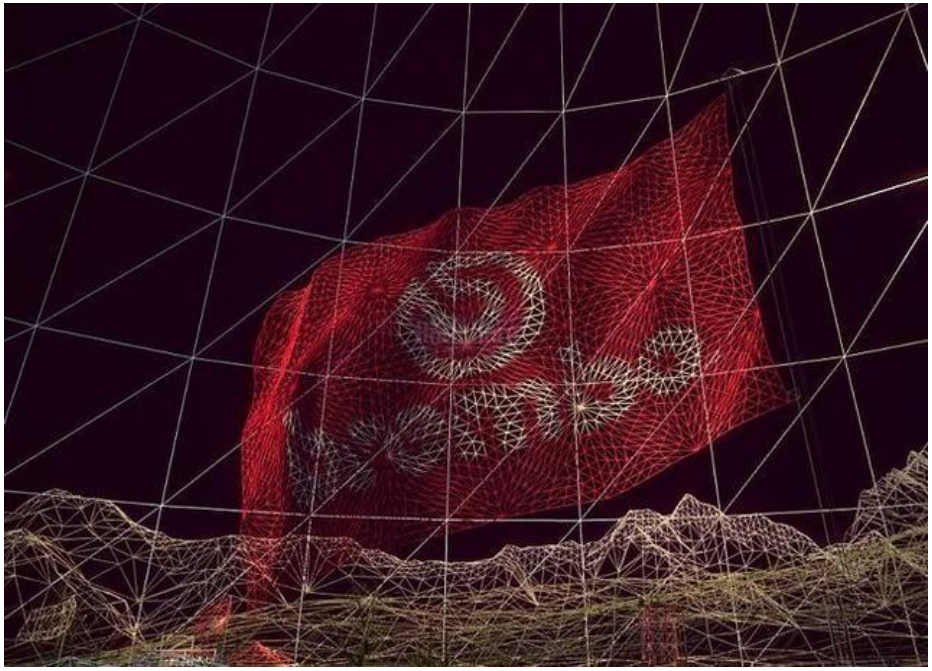
Géométrie, Topologie

- Vertex: c’est un sommet de la géométrie et qui comprend :
  - Positions
  - Couleur
  - Normal
  - Coordonnée de texture
  - Et un peu tout ce que l’on veut

Ho les triangles

- La carte graphique est capable d'afficher que des triangles, les objets sont donc composés de triangles



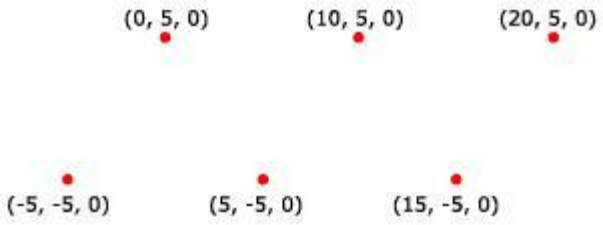


Vertex Buffer

- Un Vertex Buffer est un tableau de Vertex. Cela représente une géométrie
- C'est ce que l'on va donner à la carte graphique pour l'affichage
- La carte graphique va les prendre dans l'ordre pour composer la géométrie qu'elle va afficher tout en tenant compte de la topologie

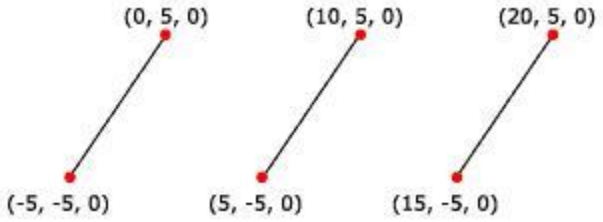
Point

- Liste de point, chaque sommet est un point

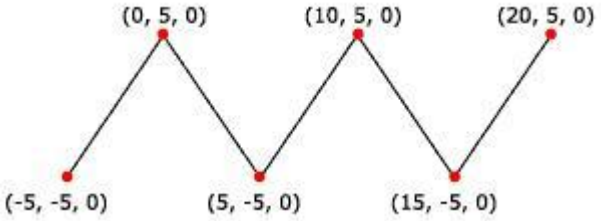


Line

- Line List: Les sommets sont prit deux par deux pour former une ligne

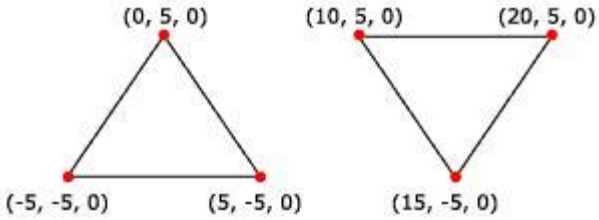


- Line Strip: Pour composer une ligne on prend le dernier sommet que l'on relie au courant

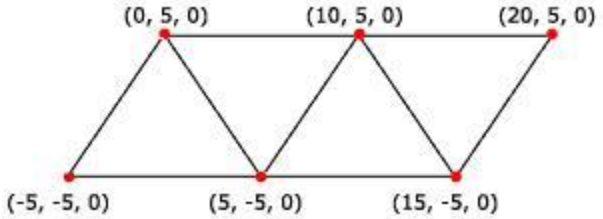


Triangle

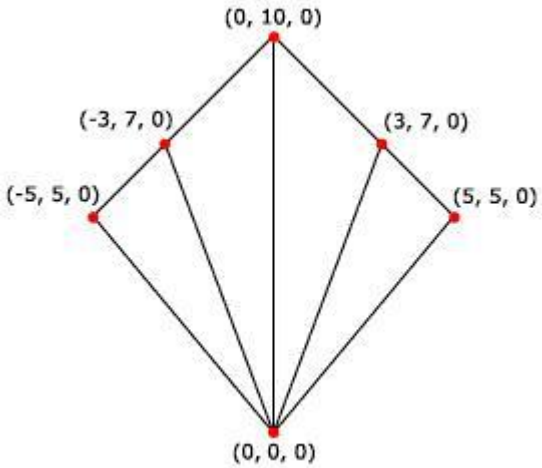
- Triangle List



- Triangle Strip



- Triangle Fan: Le 1er sommet est toujours utilisé et ensuite même principe que le triangle strip



Les 1er sont les derniers

- De base les sommets sont pris dans l'ordre dans lequel ils sont dans le vertex buffer. Si on veut changer cette ordre il faut utiliser un index buffer

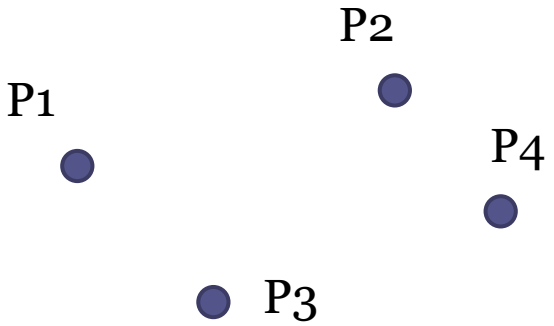
- Cela permet aussi des optimisations en ne dupliquant pas les sommets.

Index Buffer

- Un index buffer est un tableau d'entier qui contient les indexes des sommets du vertex buffer
- La carte graphique va prendre les indices dans l'ordre de l'index buffer pour composer la géométrie
- Si on devait faire le code qui correspond on aurait:
  I^e sommet : VertexBuffer[IndexBuffer[I]]

Exemple Triangle List

- Un exemple Triangle List:

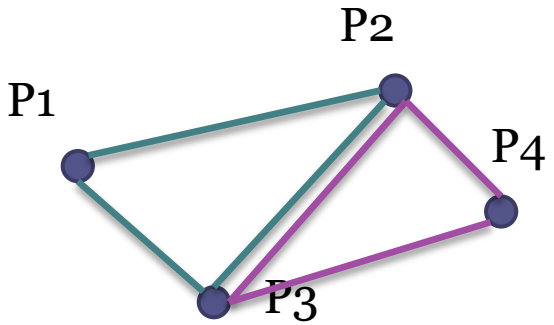


Vertex Buffer :

P1	P2	P3	P4

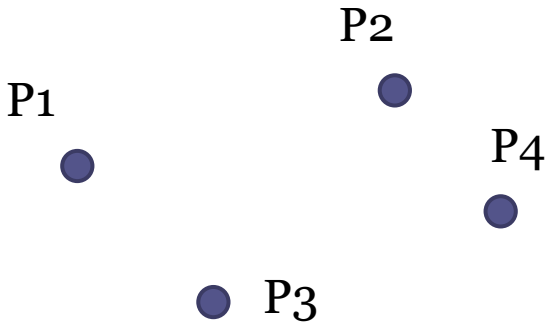
Index Buffer :

0	1	2	3	1	2



Exemple Triangle Strip

- Autre exemple en Triangle Strip

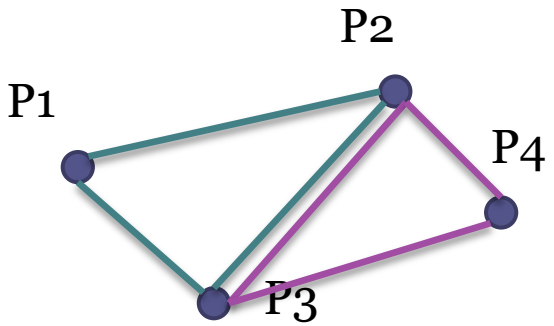


Vertex Buffer :

P1	P2	P3	P4

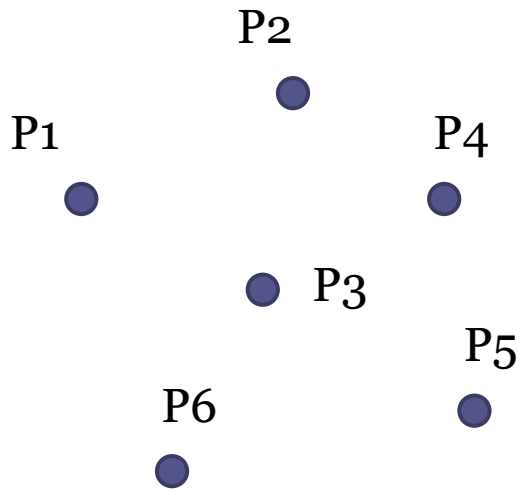
Index Buffer :

0	1	2	3



Exemple Triangle Fan

- Et encore un exemple: Triangle Fan

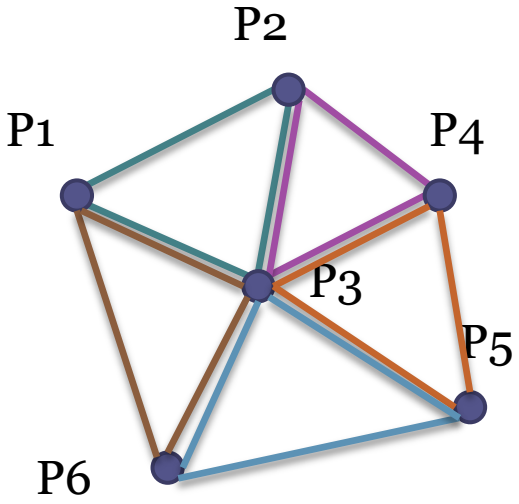


Vertex Buffer :

P1	P2	P3	P4	P5	P6

Index Buffer :

2	0	1	3	4	5	0



Back face culling

- L'ordre des sommets est important car la carte graphique peut faire du Back face culling
- L'ordre des sommets définie un sens par rapport à ce que voit la camera (sens des aiguilles d'une montre ou sens inverse)
- La carte graphique n'affichera pas les triangles qui sont définie dans le mauvais sens

Un peu d'opium, heu d'optim

- Un appel à une fonction de dessin est appelé un drawcall.
- Il faut limiter au maximum les modifications d'état et de n'importe quoi au GPU.
- Il est mieux de faire un gros Draw Call que plusieurs petits.

Pipeline graphique

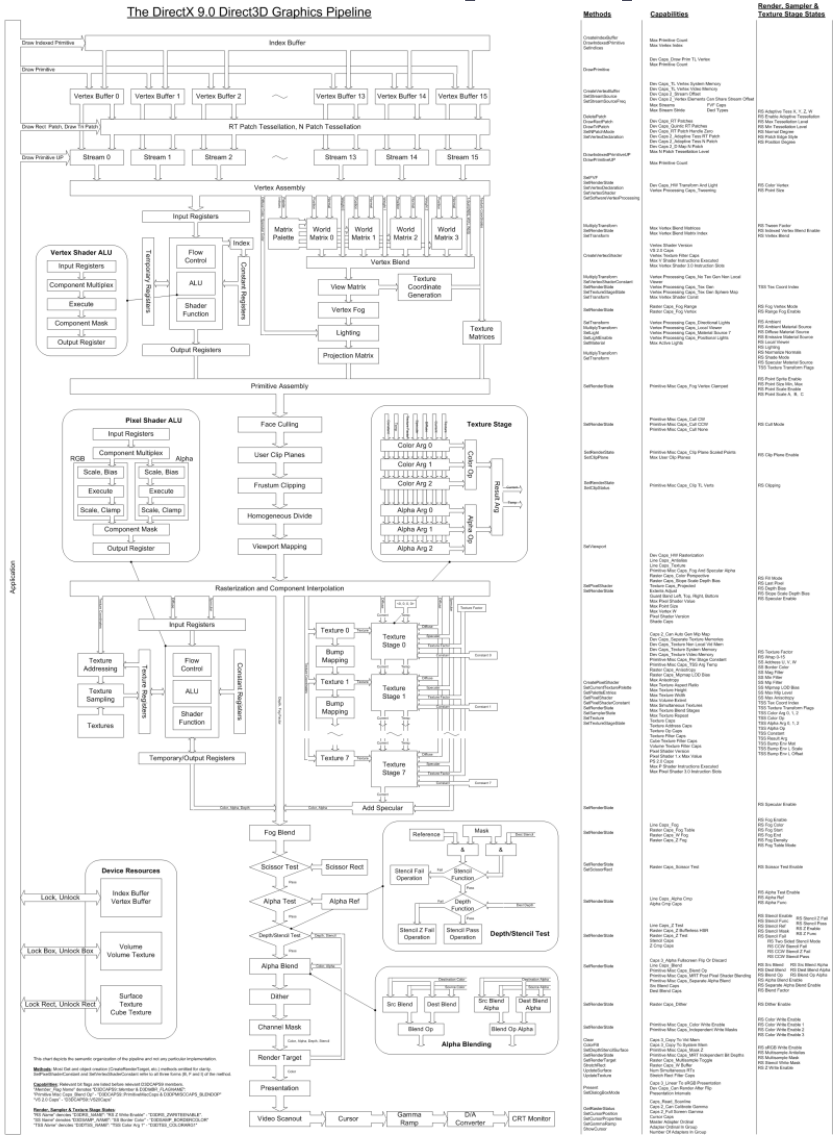
Un coup de tuyau

- Canalisation pour le transport des hydrocarbures à longue distance.
- Pipeline = tuyau !!
- Pipeline peut aussi est traduit par rumeur.

Non, pipeline graphique !

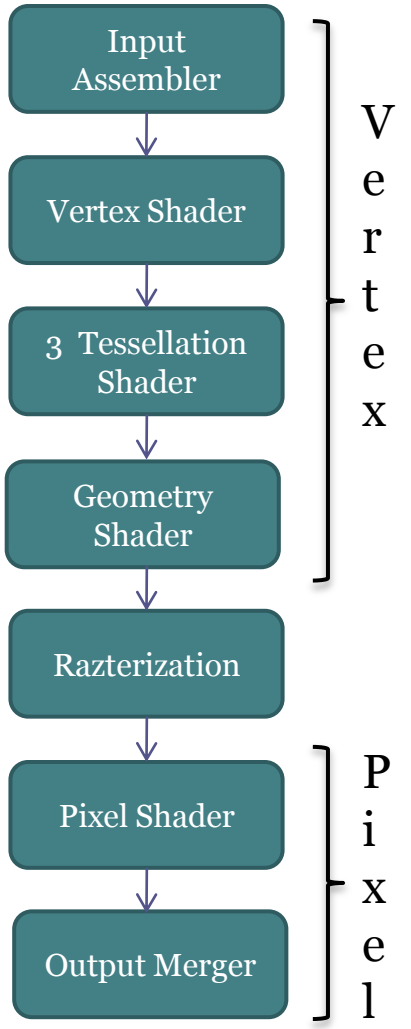
- C'est l'ensemble des différentes étapes par lesquelles passe la géométrie pour être affichée à l'écran.
- On peut voir le pipeline graphique comme une machine à états.
- Très important à connaitre

Outch, c’est compliqué



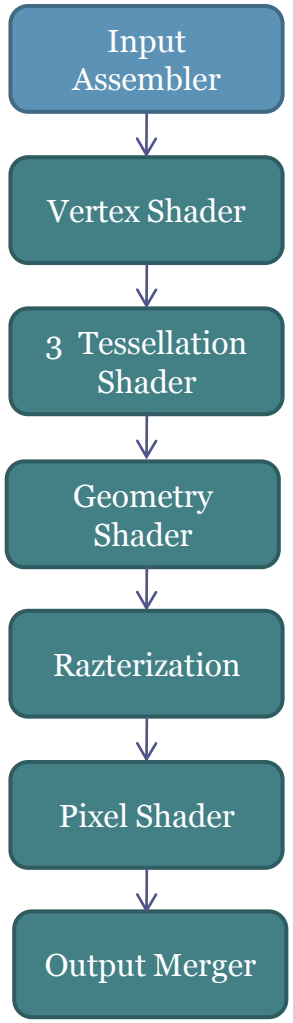
En plus simple

- Est programmable depuis DirectX 8
- En entré on a une géométrie que l'on veut afficher
- En sortie on a la géométrie qui est afficher comme on l'a demandé dans la render target.



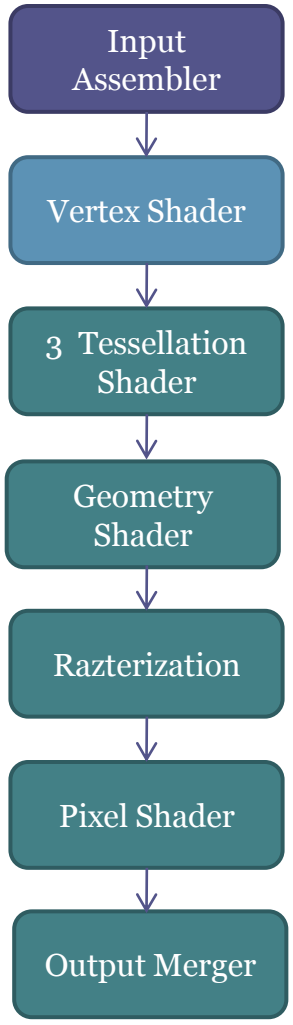
Input Assembler

- Première étape du pipeline
- S'occupe de récupérer et de regrouper les informations des vertex pour les envoyer un par un au Vertex Shader



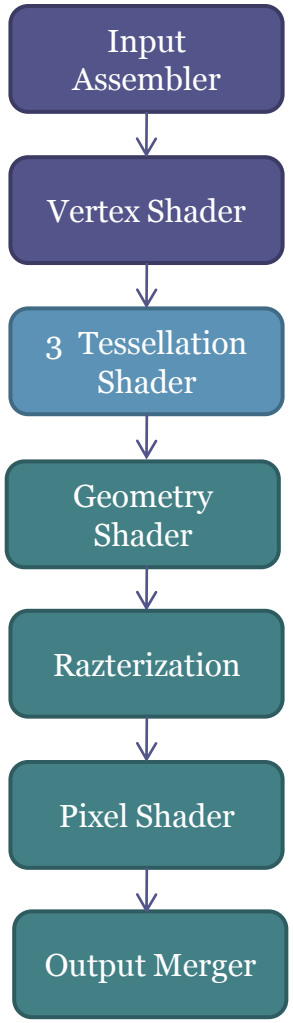
Vertex Shader

- Prend un seul vertex en entrée à la fois
- S'occupe de transformer le vertex comme on le souhaite
- À la sortie on a le vertex :
  - Si aucune Tesselation ou Geometry Shader on a le vertex qui est projeté à l'écran
  - Sinon on le transforme juste en world



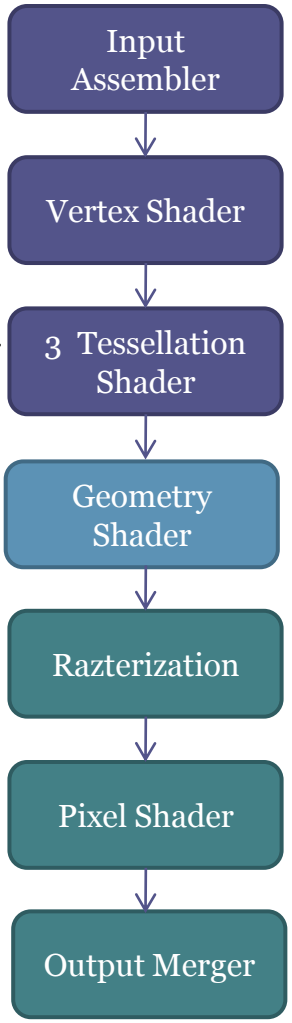
Tessellation

- Depuis DirectX 11
- Composée de 3 étapes:
  - Hull
  - Tessellation (non programmable)
  - Domain
- Hull: rempli et configure l'input du Tessellator
- Domain: Interpole la sortie du Tessellator



Geometry Shader

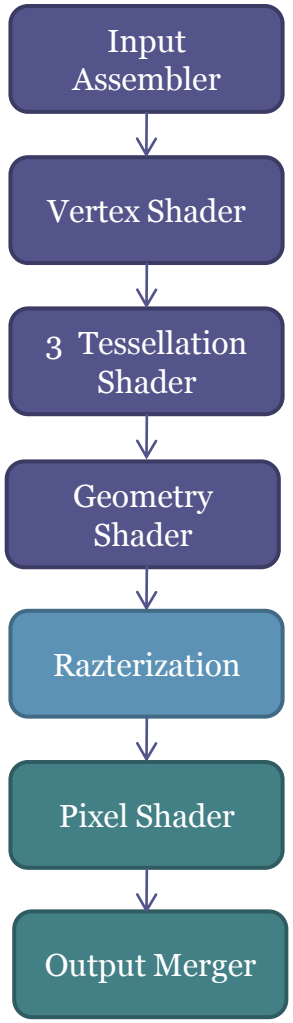
- Depuis DirectX 10
- En entré on a une géométrie en fonction de la topologie choisit.
- En sortie on a un ou plusieurs triangles strip, line strip ou point
- Les triangles en sortie ont été projetés à l'écran (world-view-proj appliquée)



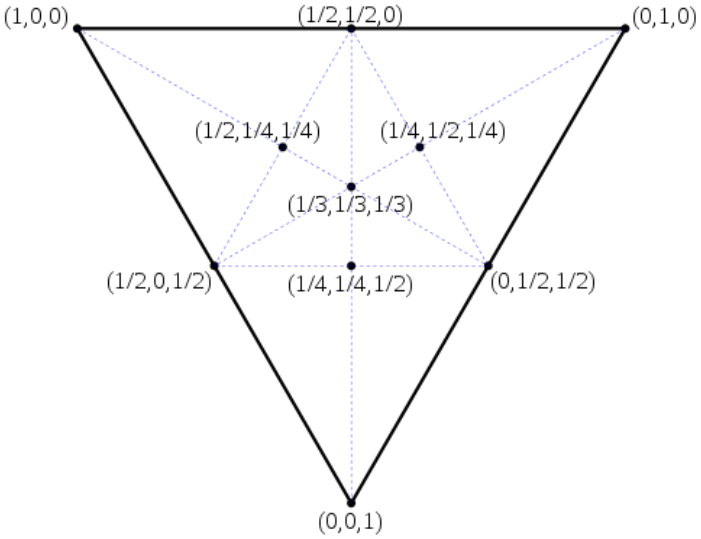
Rasterisation

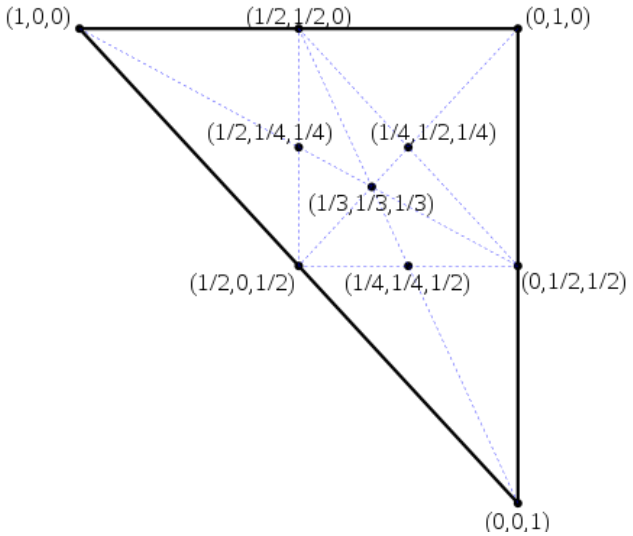
- Étape non programmable.
- S'occupe de transformer votre triangle en pixels

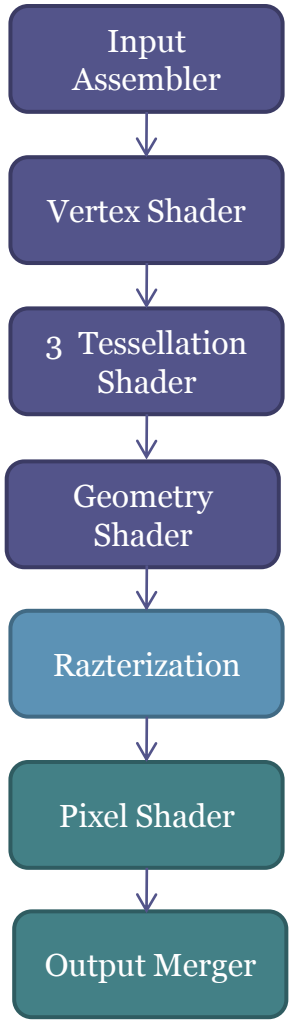
- Pour chaque pixel les valeurs sont interpolées entre les sommets de la géométrie



Rasterisation

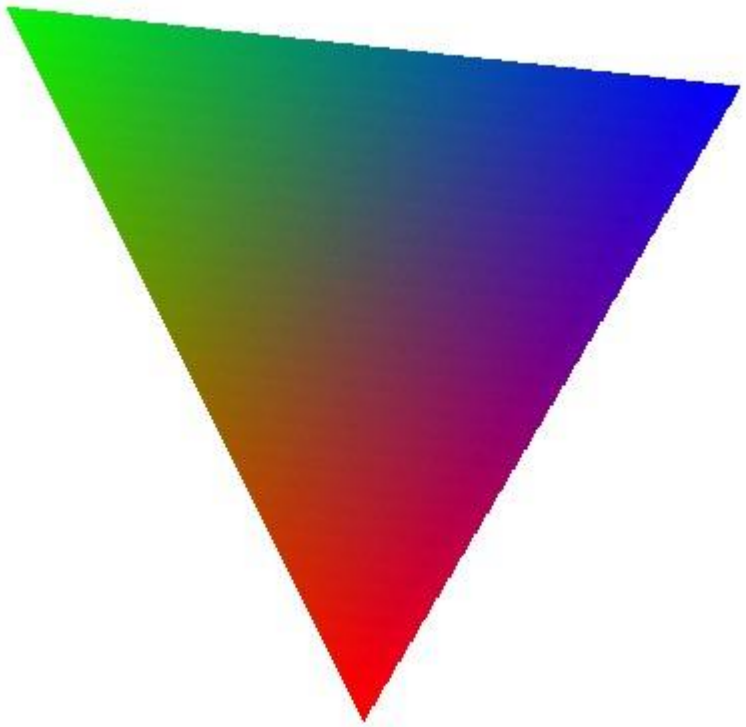


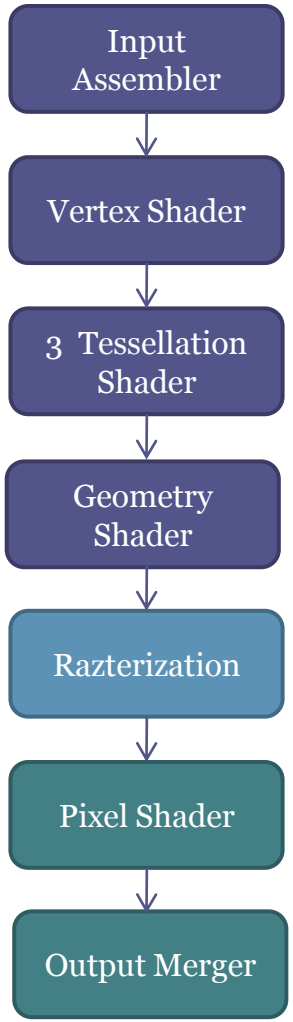




Rasterisation

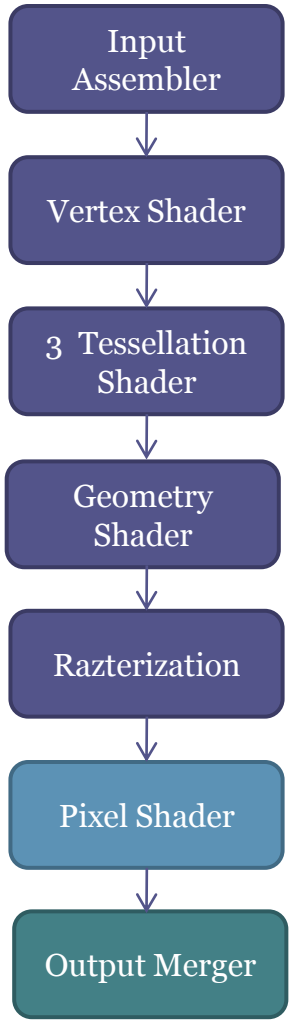
- Le même schéma avec de la couleur





Pixel Shader

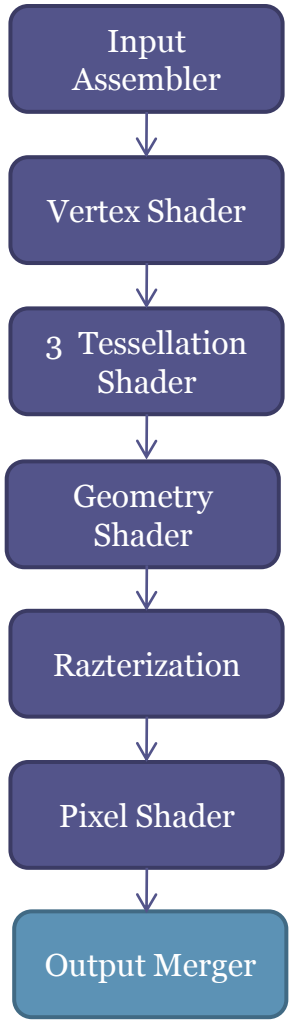
- Prend un pixel à la fois.
- S'occupe de donner la couleur à ce pixel
- Les données en entrée du Pixel Shader proviennent des données de sortie du Vertex Shader une fois interpolées par le rasteriser



Output Merger

- Dernière étape, elle s'occupe d'écrire la couleur de sortie du Pixel Shader dans le back buffer suivant ce que l'on a demandé

- S'occupe aussi de faire
  - le Z Test
  - Le Blend

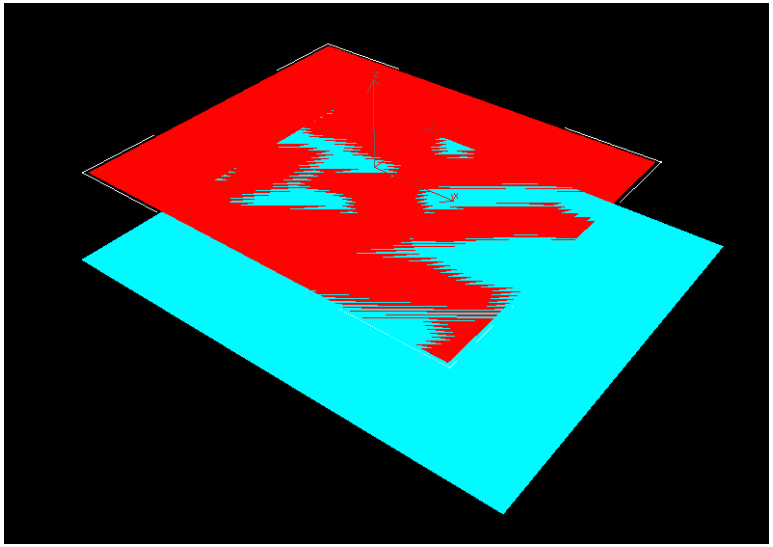


Ztest

- Les triangles sont rendu dans n'importe quelle ordre.
- Comment la carte graphique sait comment un pixel est devant un autre ?
  - Les derniers dessiner sont devant ?
  - Elle utilise un Z Buffer
- Peut être configurer

Z fight

- Le Z est compris entre 0 et 1:
  - 0 pour le Znear
  - 1 pour le Zfar
- Si les valeurs sont mal choisi on peut avoir ceci:



Baston !!

- Cela s'appelle du Z fighting
- C'est dû à l'imprécision des floats et que l'on se trouve uniquement dans un intervalle entre 0 et 1
- Cela arrive lorsque deux triangles sont coplanaires ou proche
- Plus on est éloigné de la caméra plus on peut avoir ce phénomène

Double Buffering

- Pour éviter de voir la scène se dessiner, on ne dessine pas directement dans la zone mémoire afficher à l'écran

- On utilise deux (ou plus) buffers:
  - Le front, celui qui est afficher
  - Le back, celui dans le quelle on dessine

Stencil Buffer

- Stencil = Pochoir
- Il s'agit d'une texture associée au depth buffer qui contient un entier
- Cela permet de rejeter des pixels et donc de gagner en temps de rendu
- Cela permet aussi de faire des effets comme:
  - Des ombres
  - De la réflexion

Stencil Buffer le retour

- C’est basé sur un test et des actions:
  - On fait un rendu de la zone qu’on veut afficher et on rempli le stencil buffer en fonction
  - On configure le stencil buffer (le test et la valeur de référence)
  - On refait un rendu de la scène avec le Stencil Test d’activer
- Ce teste est fait après le Z Test

Viewport

- Le viewport défini la zone dans laquelle va être afficher le front buffer
- Ce ne coupe pas la scène, ça la « redimensionne »
- Il s'agit d'un rectangle

Scissor

- Le Scissor test permet de rejeter une partie de la zone dans laquelle on dessine
- Cela coupe le rendu et donc on peut avoir un gain de performance
- On spécifie un rectangle et tout ce qui se trouve en dehors ne sera pas afficher
- Ce test est fait avant le Z Test

United State of Graphics

- Le pipeline graphique reste une machine à état

- On peut encore setter:
  - Le blending
  - Le depth
  - Le stencil
  - Solid / Wireframe
  - Le mode culling
  - Le scissor test
  - Et plein d'autre chose

Conclusion

- Il y a encore beaucoup de choses à voir
- Le prochain cours : Win32, les textures, les shaders, l'éclairage
- Des questions ?

Une technique

Cascaded Shadow Map + Variance Shadow Map