# Synthèse d'Images Avancée

## *Niveaux de détails*

gael.guennebaud@inria.fr

http://www.labri.fr/perso/pbenard/teaching/sia

# LOD
## *(Level Of Details)*

# LOD : Objectif

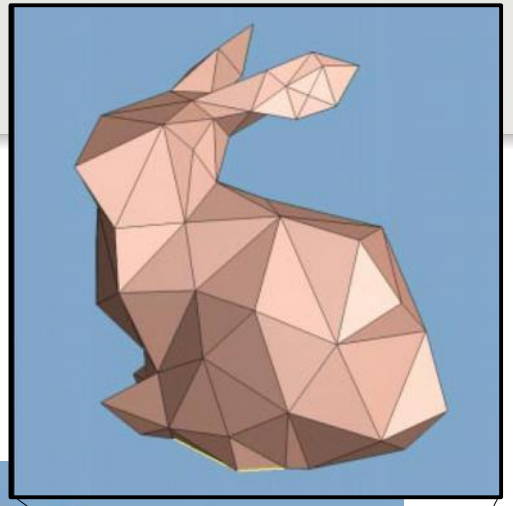
- **Observations :**
  - objet détaillé → énormément de polygones
  - coût de rendu = fonction du nombre de polygones
  - taille des polygones en espace image = fonction de la distance à l'observateur
    → objet lointain → nombreux micro polygones se projetant sur un même pixel

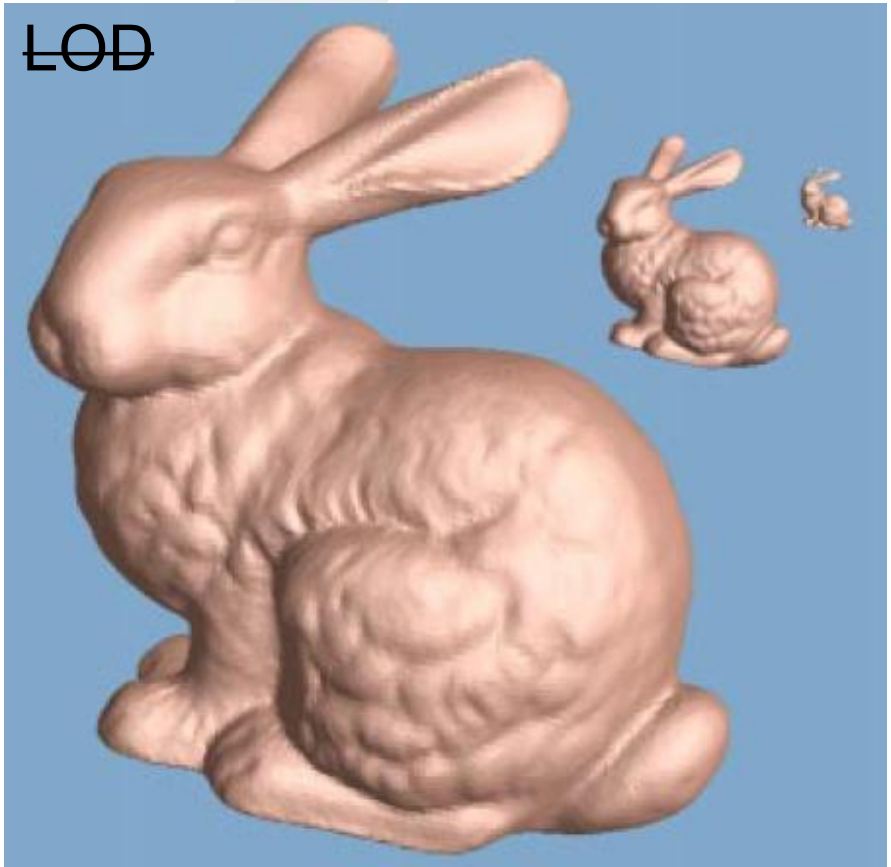
- **Idée :**
  - adapter la résolution (le nombre de polygones) du maillage en fonction du point de vue

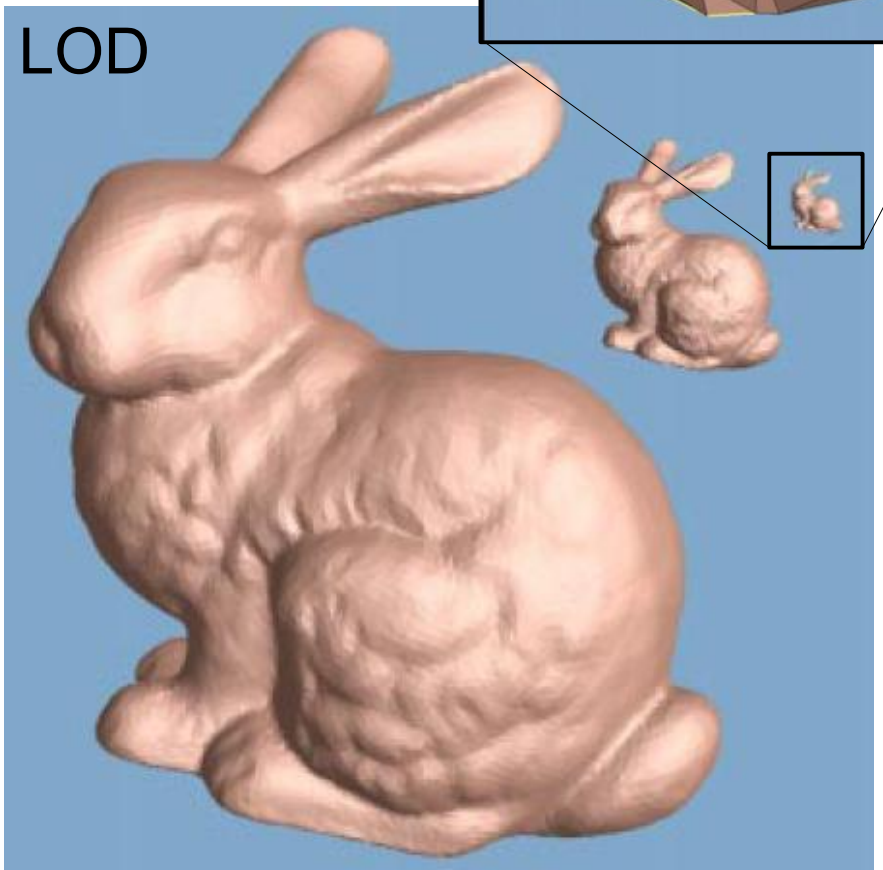
- **Défis :**
  - comment calculer des versions simplifiées du maillage ?
  - comment choisir la résolution adaptée ?
  - comment rendre cela efficace du point de vu du GPU ?

# LOD discret

- **Différentes représentations du même objet sont calculées, chacune à un niveau de détail différent. Pendant l'exécution de l'application, une représentation de l'objet est sélectionnée et visualisée.**

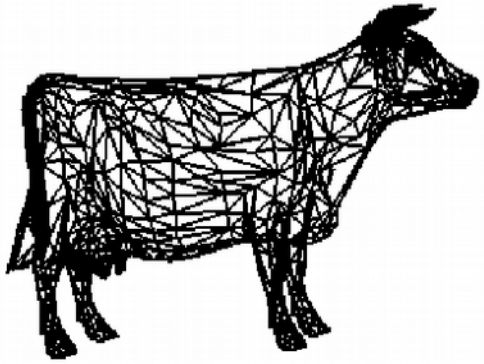




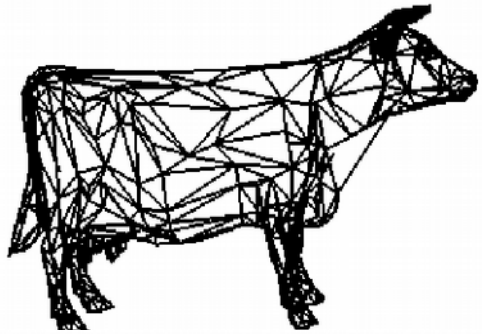



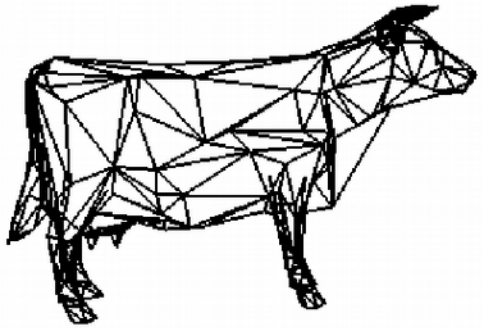
# LOD discret

- **Précalculer les niveaux de détails**
  - surfaces de subdivision : trivial !
  - maillage quelconque :
    → simplification de maillage



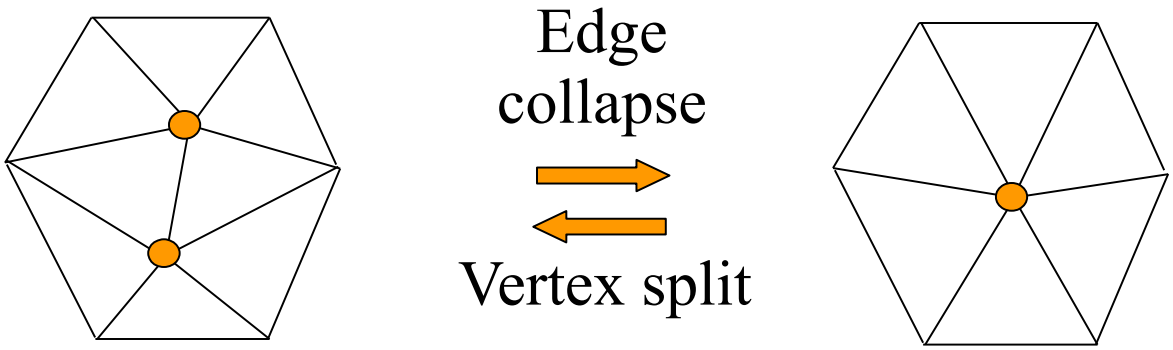
- **Critères ?**
  - Minimiser la distance entre le maillage simplifié et le maillage original
  - Distance de Hausdorff
    - difficile en pratique
  - Heuristiques...





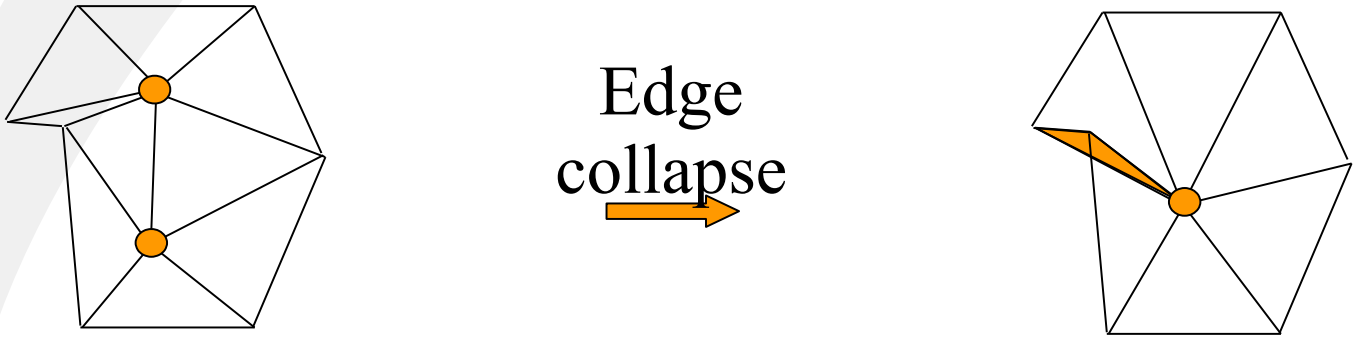
# Edge collapse

- Cet opérateur transforme une arête en un sommet. L'opérateur inverse appelé "vertex split" ajoute une arête et le triangle qui lui est adjacent.

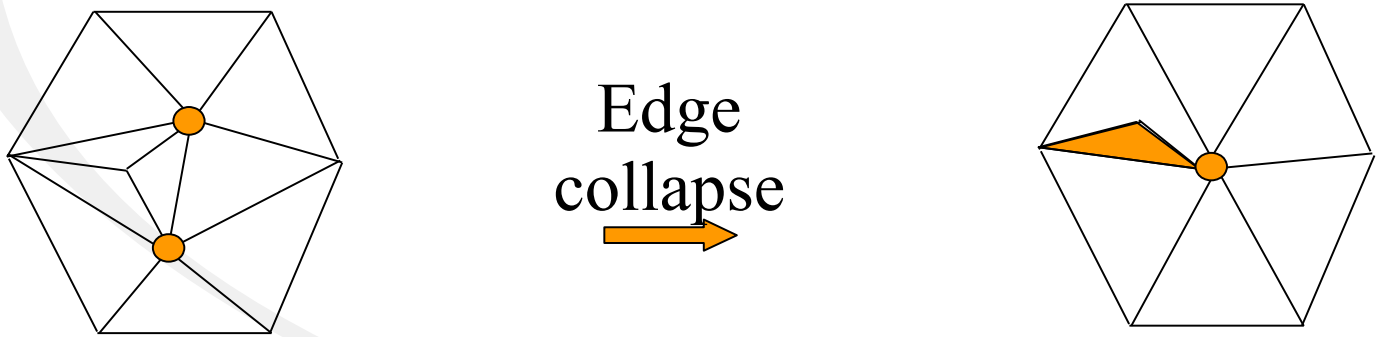


# Quand ne pas utiliser “Edge collapse”

- **Attention au problème de recouvrement:**

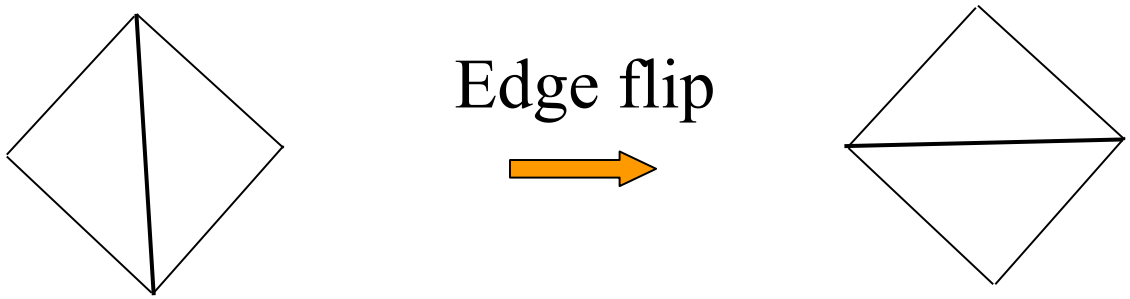



  - Le maillage reste 2-manifold, mais le résultat est indésirable
- **Le maillage peut aussi devenir non-manifold:**



# Edge flip

- **Edge flip**
  - uniformiser les valences
  - aligner les arêtes avec les angles saillants




- Il a été montré que toutes les opérations de simplification de la géométrie d'un maillage peuvent être réalisée à l'aide des opérateurs "edge flip" et "edge collapse".

# Simplification par edge collapse

- **Calculer une erreur pour chaque arête**
  - représente l'erreur introduite par la contraction de l'arête
    - Quadratic Error Metric
    - Linstorm & Turk's method
- **Tant que #triangles > seuil**
  - contracter l'arête avec l'erreur minimale
  - mettre à jour les erreurs des arêtes voisines
- **Améliorations :**
  - contracter plusieurs arêtes en même temps
  - passe de edge-flip pour régulariser le maillage
  - prise en compte de différents critères dans le calcul de l'erreur (géométrie, normales, texture, forme des triangles, etc.)

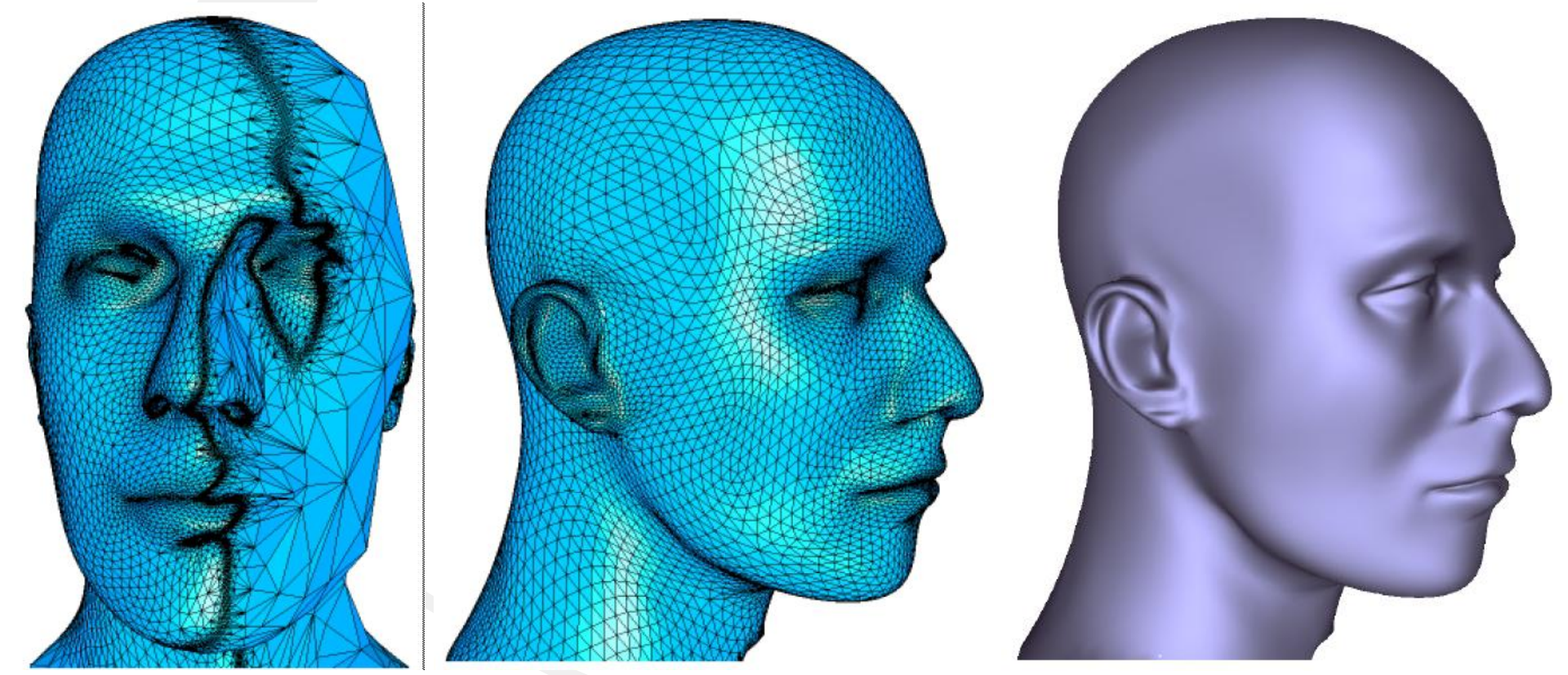
# LOD continu

**Adapter la résolution de l'objet au polygone prêt**

- une seule structure représentant l'objet avec une représentation continue des détails
- utiliser le nombre optimal de poly. pour représenter l'objet à la définition souhaitée
- modification de la géométrie au cours de l'exécution de l'application
  - approche « top-down »
    - supprimer les triangles un à un → très difficile en pratique !!!
  - approche « bottom-up »
    - partir d'une version extrêmement simplifiée
      - insérer les détails stockés dans une représentation multi-resolution
    - ex :
      - surface de subdivision
      - surface de subdivision + offsets
      - **progressive mesh**
      - quatree pour les terrains

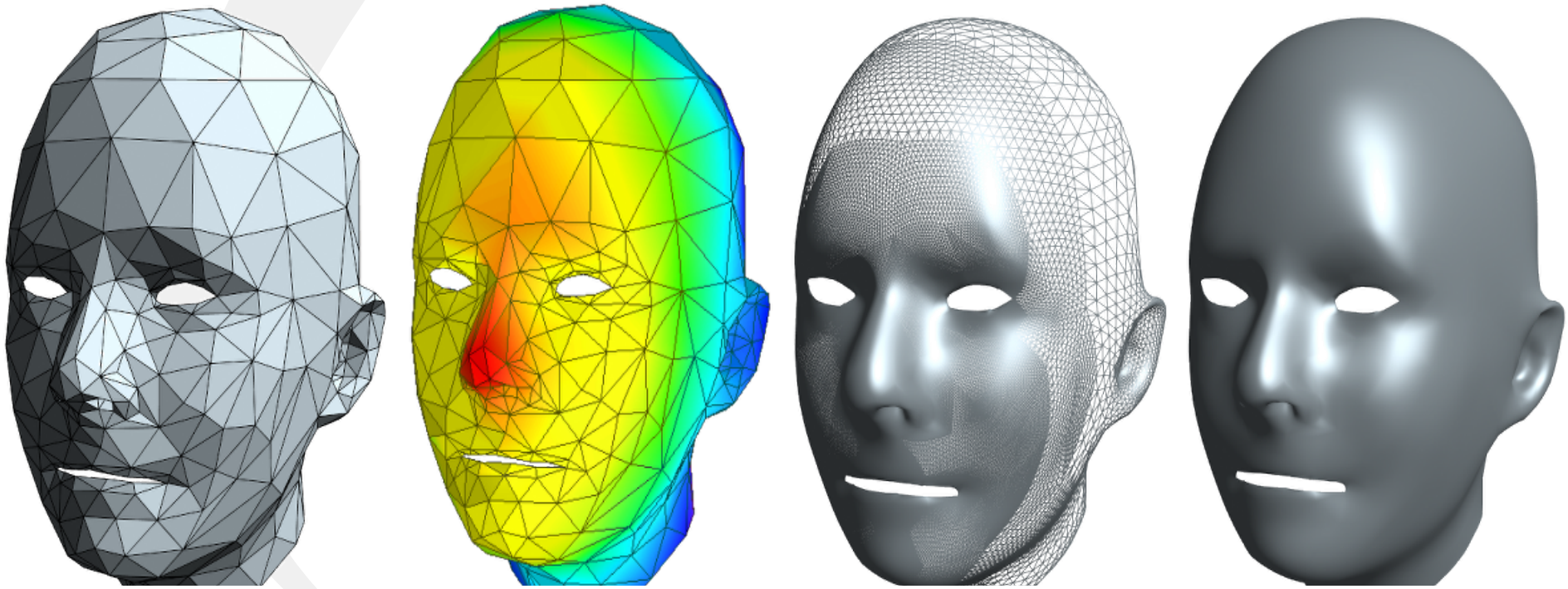
# LOD dépendant du point de vue

- **Extension des LOD continus**
  - critère de simplification qui est dépendant du point de vue
  - représentation anisotrope : différentes zones du même objet sont visualisées à des niveaux de détail différents



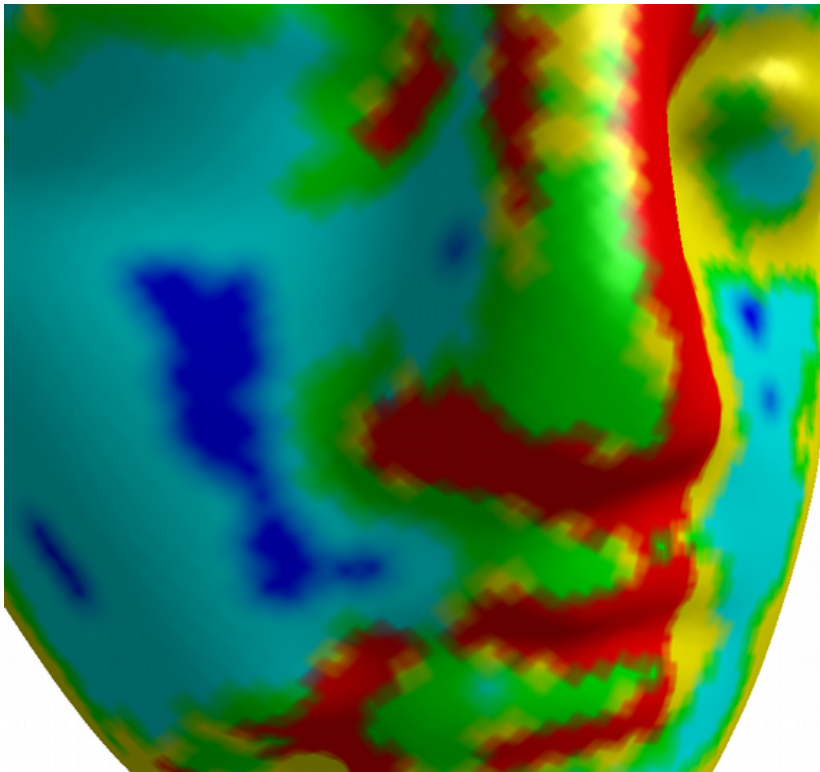
# Raffinement à la volé

- **Résolution dépendante du point de vue**



# LOD dépendant du point de vue

- **Métrique d'erreur**
  - Dépend de nombreux facteurs
    - distance à l'observateur, résolution de l'image,
    - orientation, courbure,
    - attributs, éclairage, effets de masquage,
    - texture, etc.



# Représentations des maillages

- **Pour le rendu : « la soupe de polygones »**
  - liste de sommets (avec attributs)
  - liste de faces
- **Requêtes :**
  - comment lister les arêtes de manière unique ?
  - quelles sont les faces voisines d'une arêtes ? d'une face ? d'un sommet ?
  - quels sont les sommets voisins d'un sommet ?
  - etc.
- **Opérations de bases :**
  - insertion/suppression de sommets, faces, et arêtes
  - contraction/split d'arêtes
  - edge flip

**→ ces opérations doivent maintenir la validité du maillage**

**→ besoin de structures de données plus sophistiquées**

- le standard : les halfedges (demies-arêtes)

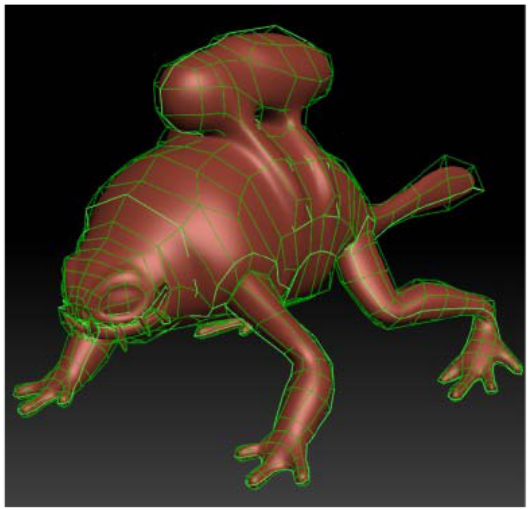
# et le GPU....

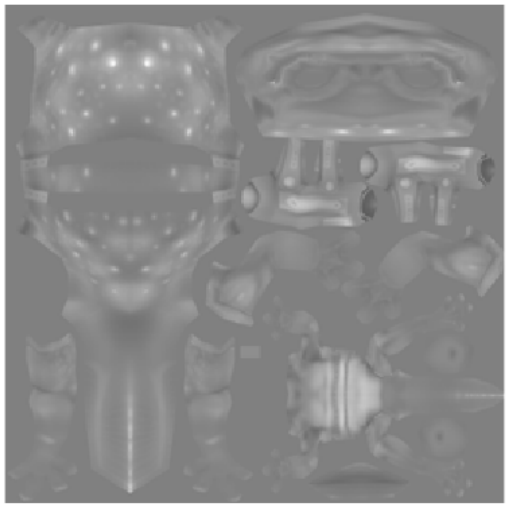
# LOD hybride

- **Simplification de maillage + normal mapping**
  - transfère des détails sous la forme de carte de normales

# Displacement Maps

- **Simplification de maillage**
  - + displacement map (offset le long de la normale, ou vecteur 3D)
  - + *normal map*
- **Compact : pas de coordonnées 3D, pas de connectivité**



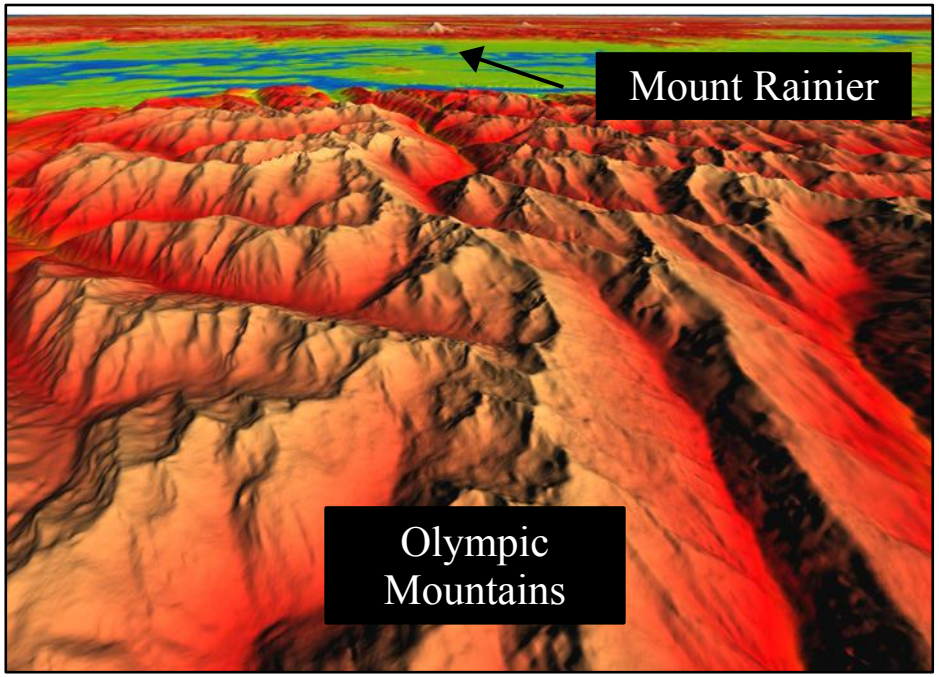


© Bay Raitt

| | Level 8 | Level 16 | Level 32 | Level 64 |
|---|---|---|---|---|
| Regular Triangle Mesh | 16MB | 59MB | 236MB | 943MB |
| Displaced Subdivision Surface | 1.9MB | 7.5MB | 30MB | 118MB |

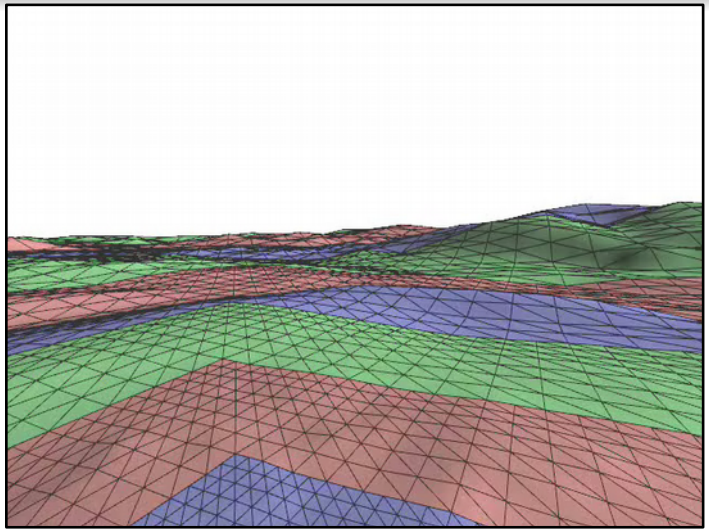
# Cas particulier : terrain

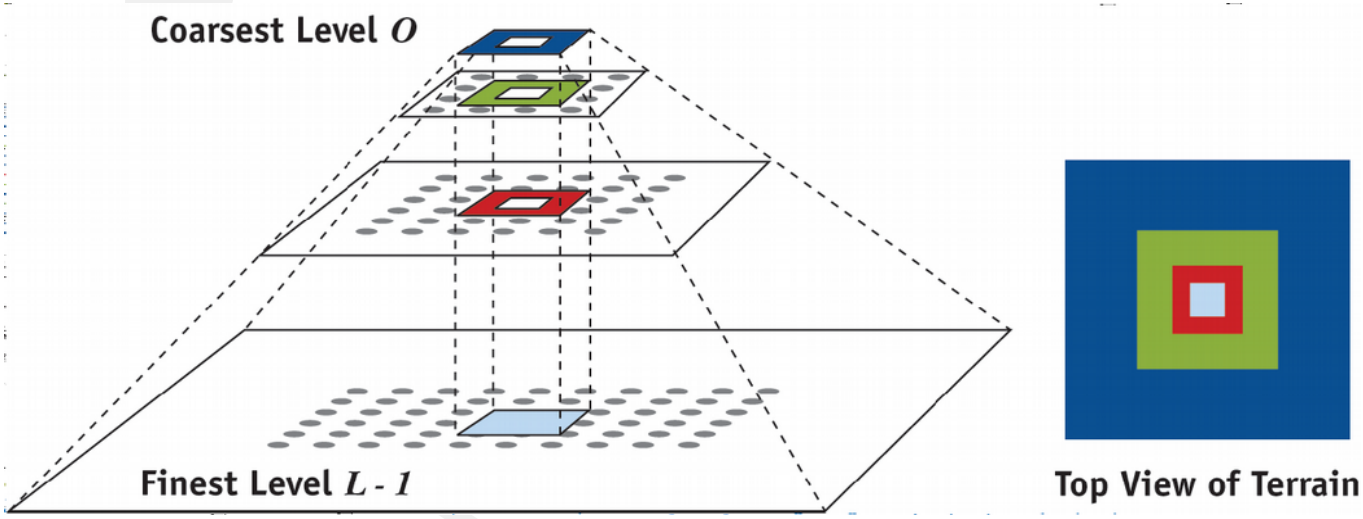
- **Problème : terrain = infini !**
  - => géométrie multi-résolution spécialisée
  - ex. : « *geometry clipmaps* » [Losasso04]



# Geometry clipmaps [Losasso04]

- = clipped mimap hierarchy
  - construction en fonction du point de vue (distance uniquement)
  - mise à jour incrémentale
- permet la compression, synthèse
- => simple mais efficace !

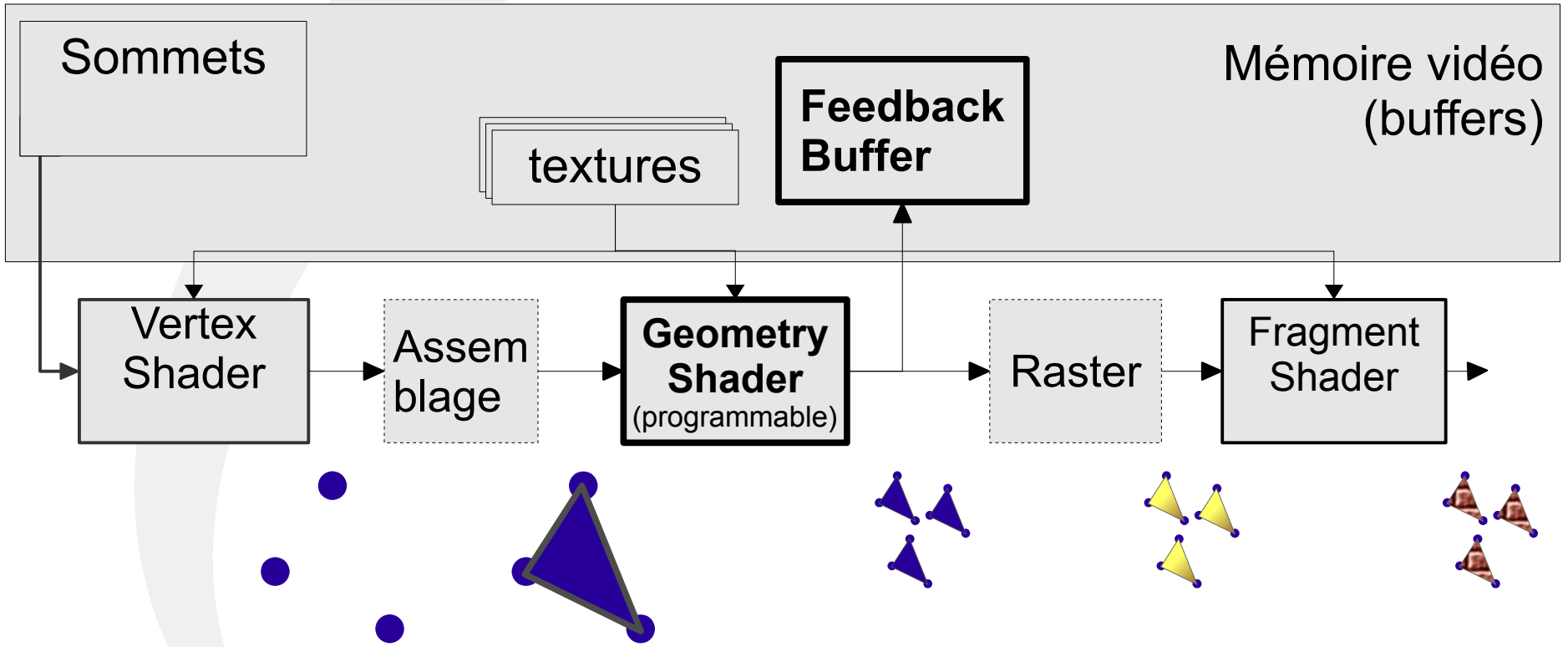







Top View of Terrain

# Geometry Shader



# Geometry Shader

- **En entrée**
  - une primitive (POINT, LINE, TRIANGLE)
    - = tableau des sommets (+ attributs) en sortie du vertex shader
- **En sortie**
  - gl_Position + variables « out »
    - EmitVertex() pour générer un sommet
    - EndPrimitive() pour finaliser une primitive
      - POINTS, LINE_STRIP, TRIANGLE_STRIP
      - peut être différent du type en entrée (POINT → TRIANGLE)
  - variables en sorties (out) interpolées par le raster et deviennent entrées du fragment shader
  - on peut générer plusieurs primitives (ou zéro → culling)
- **Feedback-buffer**
  - les sommets en sortie peuvent être empilés (sauvegardés) dans un VBO
  - permet de mettre en cache la géométrie ainsi générée

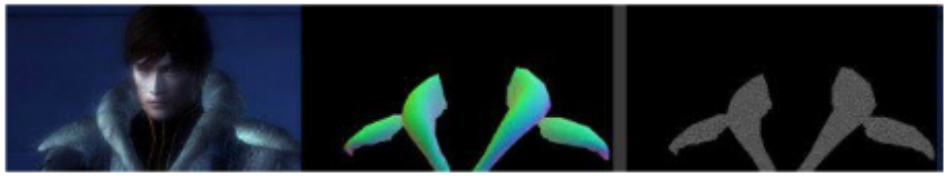
# Geometry Shader

```
layout (triangles) in;
layout (triangle_strip, max_vertices=6) out;
uniform mat4 mat_mvp;
in vec3 in_normals[3];
in vec3 in_positions[3];
out vec3 out_normal;

void main() {

  //Pass through the original vertex
  for(int i=0; i<3; i++)  {
    gl_PerVertexOut.gl_Position = gl_PerVertexIn[i].gl_Position;
    out_normal = in_normals[i];
    EmitVertex();
  }
  EndPrimitive();

  //Push the vertex out a little using the normal
  for(int i=0; i<3; i++) {
    vec3 p = in_positions[i] + 0.1 * in_normals[i];
    gl_PerVertexOut.gl_Position = mat_mvp * vec4(p, 1.0);
    out_normal = in_normals[i];
    EmitVertex();
  }
  EndPrimitive();
}
```

# Geometry Shader

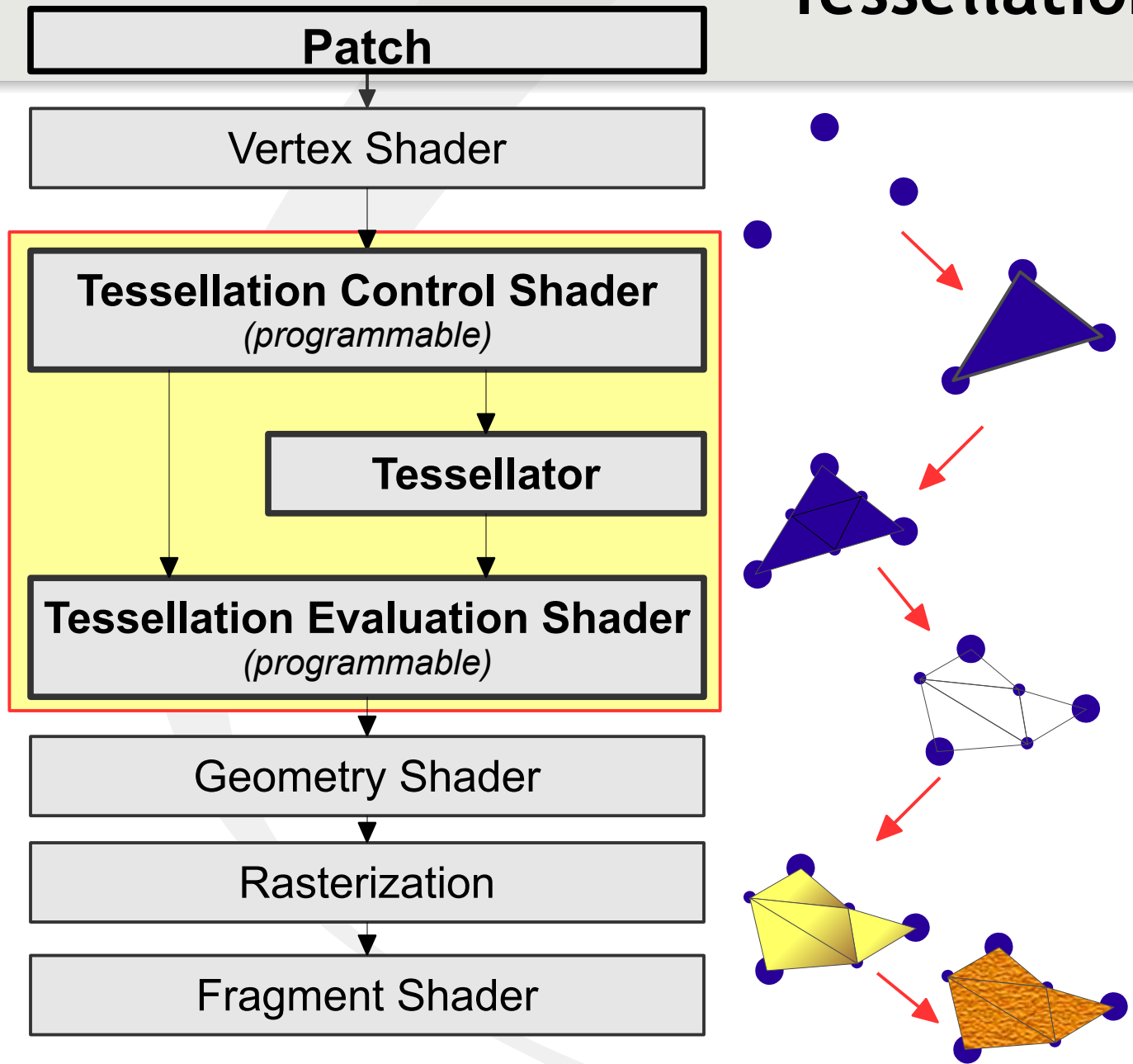
- **Limitations**
  - efficace uniquement si le nombre de primitives générées est ~ constant
    - en caricaturant :
      - cost = #input_primitives * #max_output_vertices
  - peu efficace pour faire du raffinement, LOD, …

- **Applications**
  - Occlusion culling
  - Rendu dans les 6 faces d'une cube map en une seule passe
  - Shadow volume : génération des faces des volumes d'ombres
  - Silhouette drawing, Line drawing (outline)
  - Fourrure




  - etc.

# Tessellation Engine

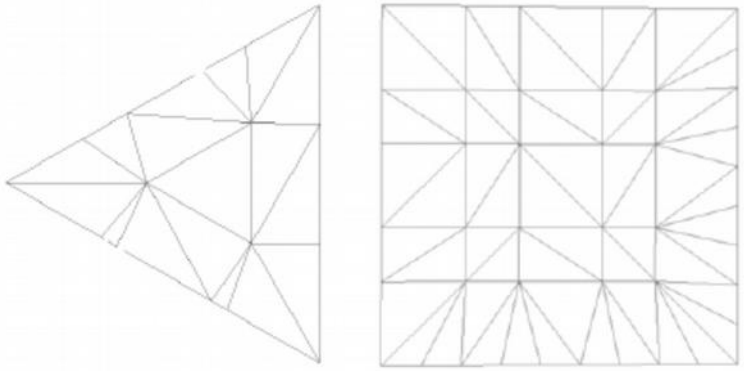


# Tessellation Engine

- **En entrée**
  - Nouveau type de primitive : « patch »
    - nombre arbitraire de sommets (points de contrôle), pas de topologie
      - ex. : NURBS
- **Vertex Shader**
  - transformation des points de contrôle
    - ex. : animation, déformation, skinning, etc.
    - → calculs couteux réalisés à une plus faible résolution

# Tessellation Engine

- **Tessellation Control Shader**
  - Patch → Base (triangle/quad)
    - Exécuté N fois où N = nombre de sommets en sortie
  - Niveau de subdivision par arête et par patch
    - réel (continue)
  - Précalcul des info par patch

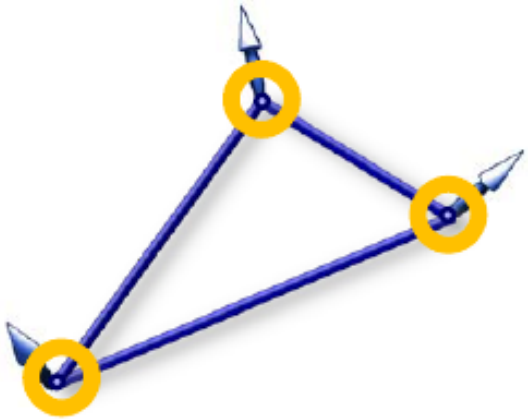


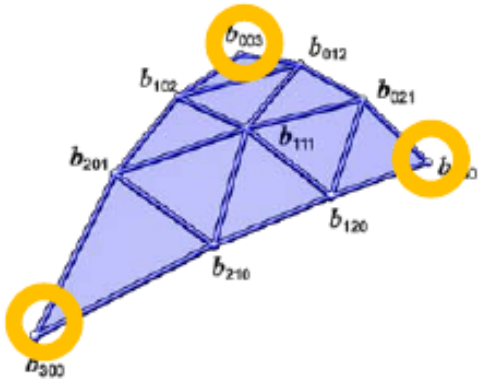
- **Tessellator**
  - Base → Triangles + uv(w) (coordonnées barycentriques)
  - Non programmable
  - Symétrique

- **Tessellation Evaluation Shader**
  - coordonnées paramétriques uv(w) → 1 sommet
  - Plus info issues du Tess. Control Shader
  - Interpolation des attributs, application d'une carte de déplacement

# Exemple PN - triangles

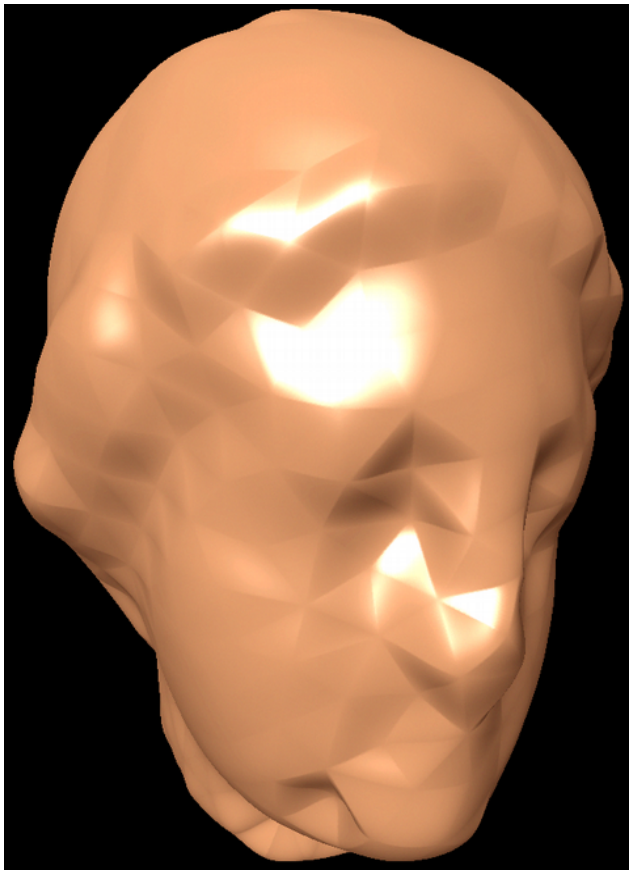
- **Idée**
  - Remplacer chaque triangle par un triangle de Bézier
  - Points de contrôles dépendent des normales






Input Triangles

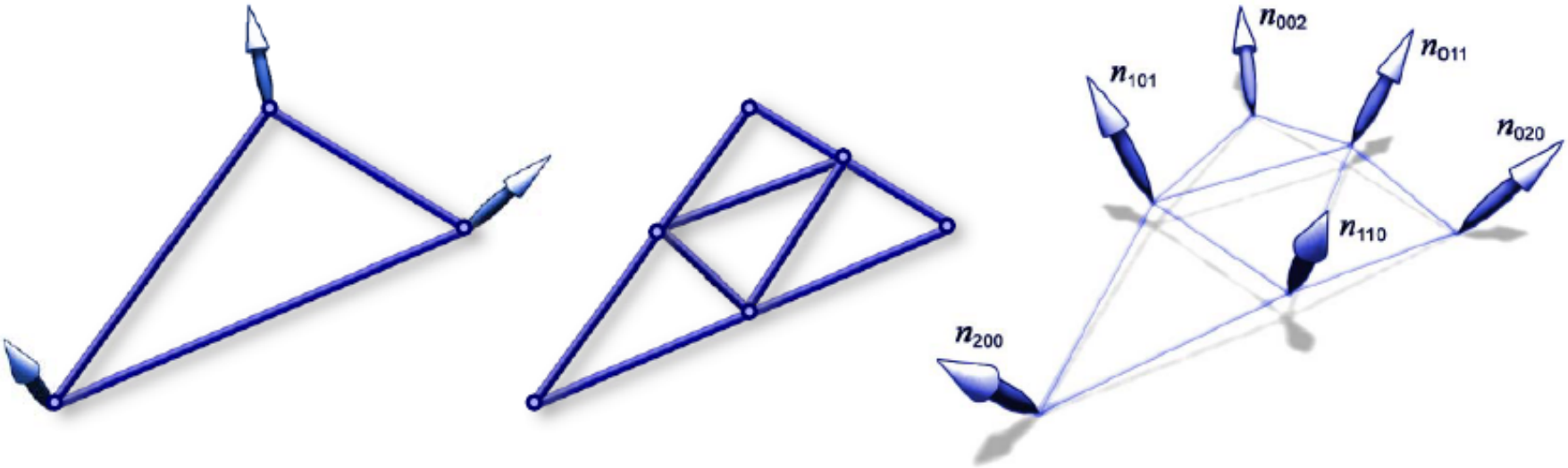
Output
Curved PN triangles



C^0 seulement !

# PN - triangles

- **+ interpolation quadratique des normales :**

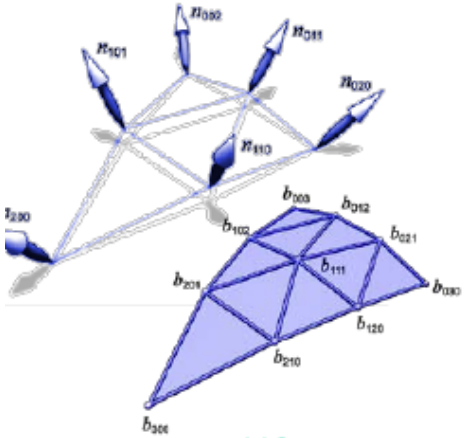


# Exemple PN - triangles

Entrée :
* points de contrôles
(triangle grossier)

**Tessellation Control**

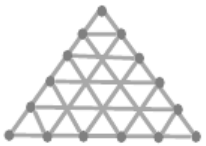
Sortie :
* points de contrôles
du triangle de Bézier



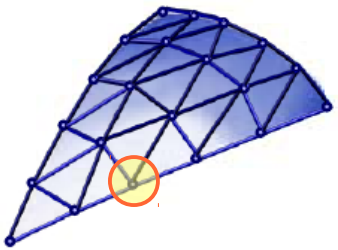
Entrée :
* niveaux de tessellation

**Tessellator**

Sortie:
* coordonnées u,v,w



**Tessellation Evaluation**



Sortie:
* un sommet interpolé