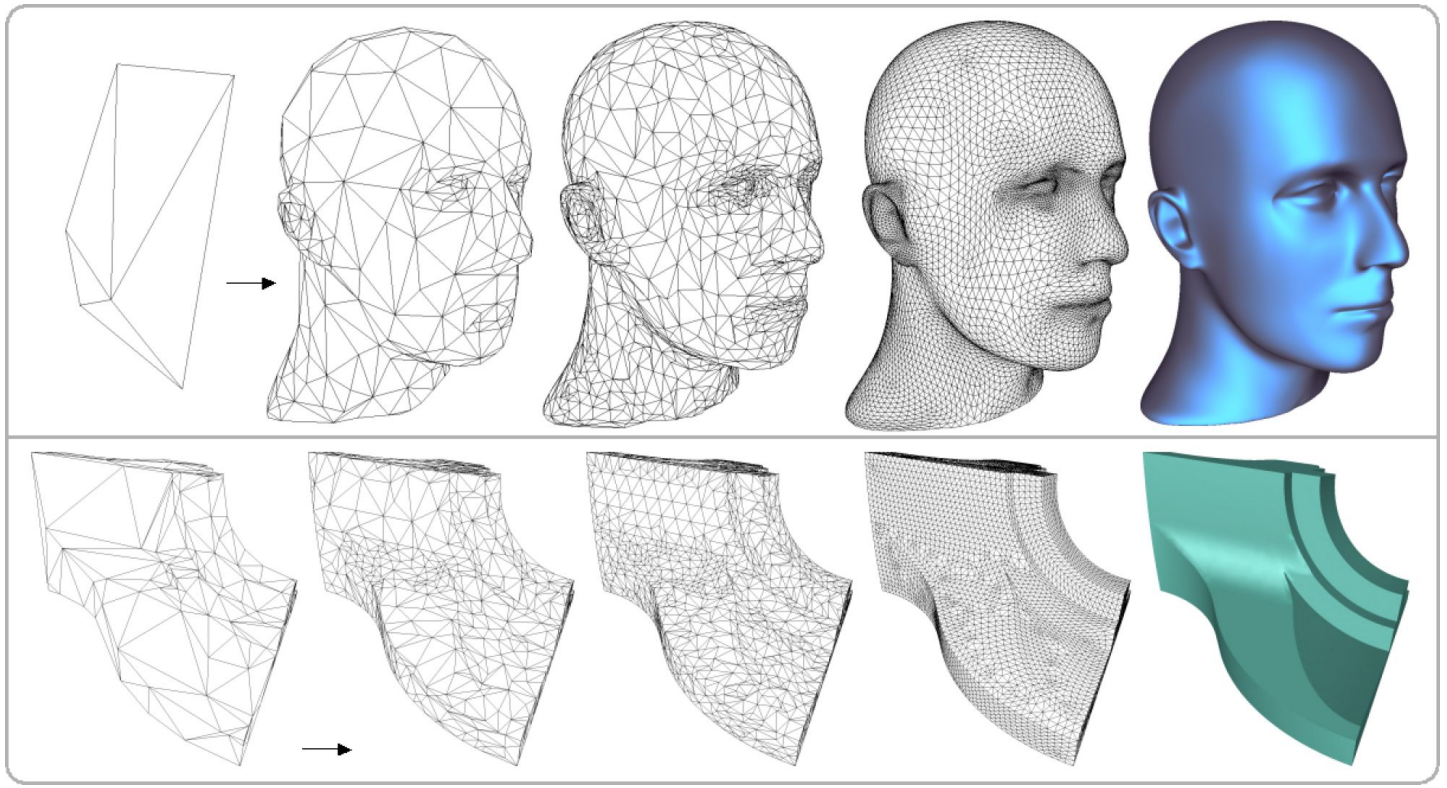
# Progressive Compression for Lossless Transmission of Triangle Meshes

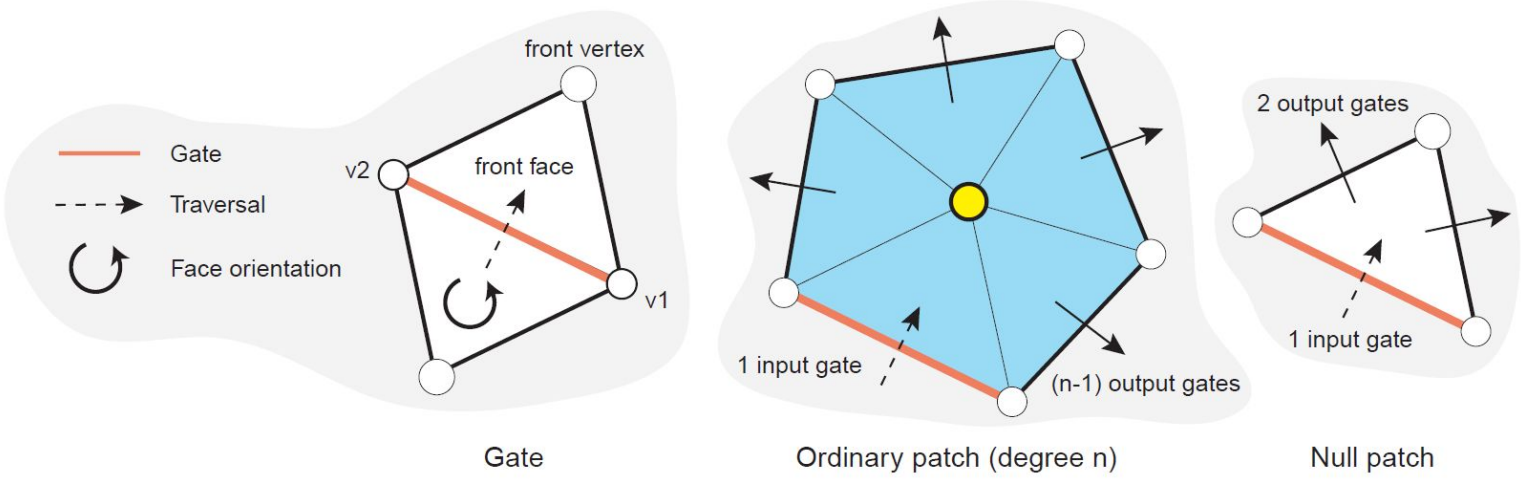


Git deposit du code : https://github.com/Roropey/CSI_Project

By:
Romain Peyremorte
Ghislain Réveiller
Anushree Shrivastava
Ying Liu

# Les notions nécessaires

- Les états de faces et vertices: free, conquered and To be removed
- Gate : une arête orienté ouvrant sur une face
- Valence : nombre de face ayant ce sommet parmi ses extrémités
- Patch : ensemble de face relié à un vertex à enlever/ajouter
- Null patch : un face qui n'appartient à aucun patch
- Retriangulation tags: un + ou un - attribué à chaque vertex pour trianguler de manière stratégique, indique si la valence augmente ou diminue

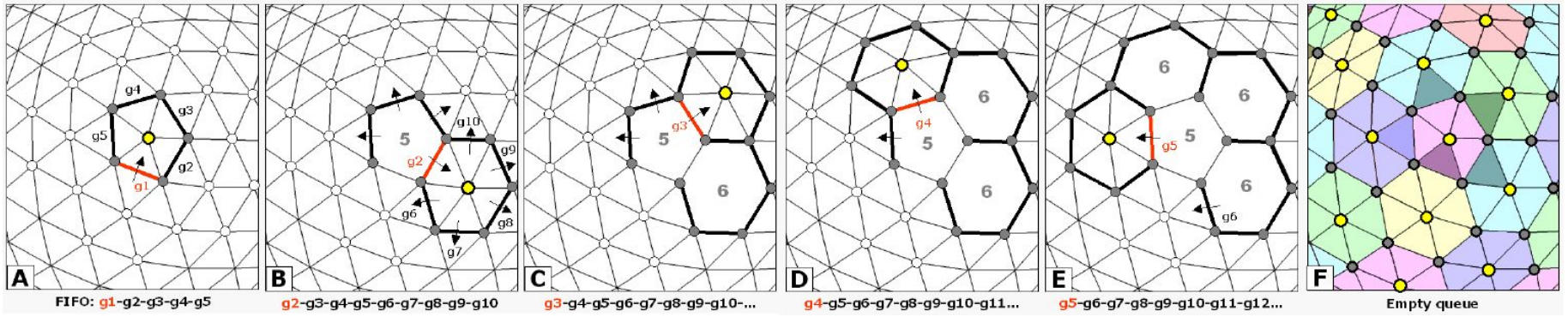


# Les différentes étapes de l'algorithme

- Décimation - Valence-driven decimating
  - Decimating conquest
    - Conquête
    - Retriangulation
  - Cleaning conquest
    - Conquête
    - Retriangulation
- Decodage - Reconstruction
  - Cleaning reconquest
  - Decimating reconquest

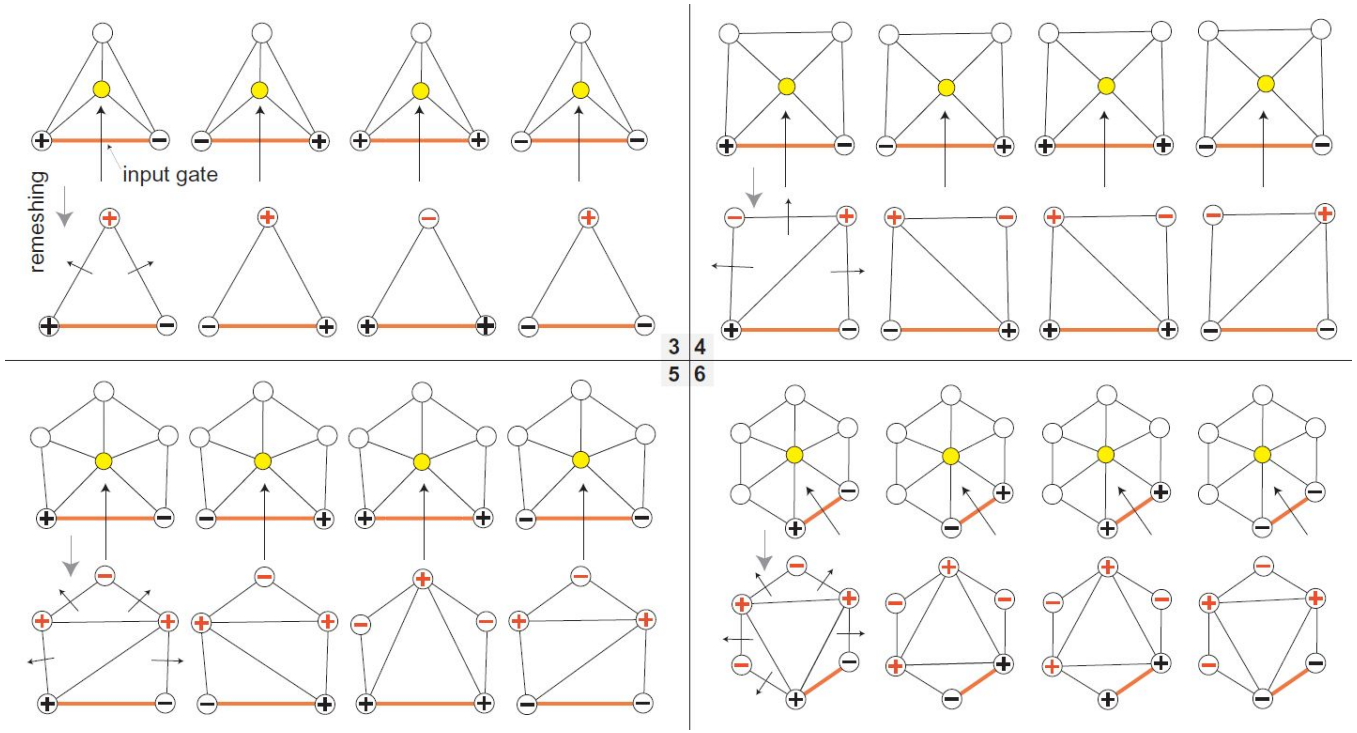
# Décimation - Valence-driven decimating

- **Decimating conquest - algorithme bouclé tant qu'il y a des gates :**
  - Récupérer un *gate* de la *queue* (FIFO)
  - Regarder l'état de la face où on entre
    - Si l'état est *conquered* ou *To be removed* → gate suivante
  - Regarder l'état du vertex d'en face (*front vertex*)
    - Si l'état est *free* et valence < = 6 → *front vertex* et face du patch misent en *To be removed*, ajout de nouvelles gates, retriangulation et valence donné en sortie
    - Si l'état est *conquered* ou (*free* et valence > 6) → *front face* mise en *Conquered*, ajout de nouvelles gates, *Null patch* indiqué en sortie



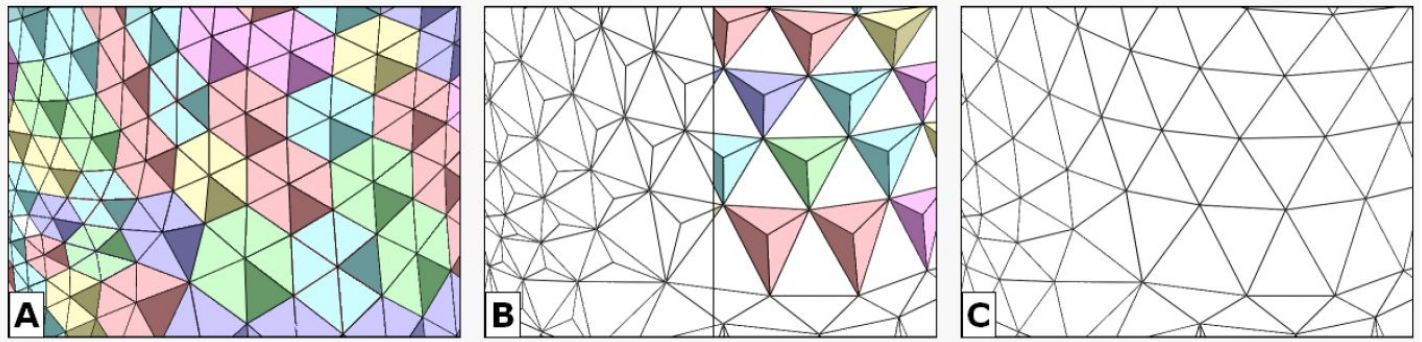
# Valence-driven decimating

- **Retriangulation du Decimating Conquest**
  - Récupérer tous les sommets voisins du sommet à supprimer et supprimer les faces voisines
  - Recréer faces à partir des *retriangulation tags* (+ et -) de la gate d'entrée, et associer *retriangulation tags* aux sommets

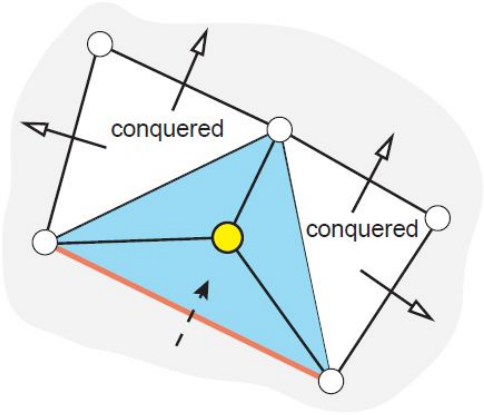


# Valence-driven decimating

- **Cleaning conquest - algorithme bouclé tant qu'il y a des gates :**



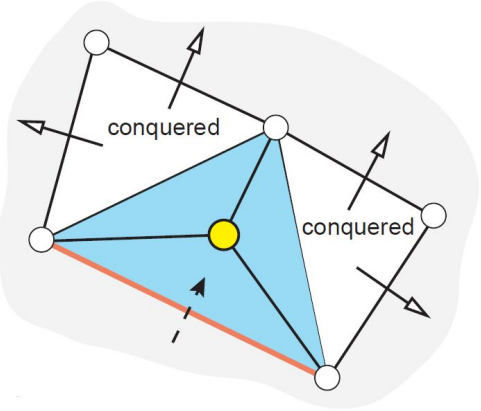

  - Récupérer une *gate* de la *queue* (FIFO)
  - Regarder l'état de la face où on entre
    - Si l'état est *conquered* ou *To be removed* → gate suivante
  - Regarder le *vertex* d'en face (*front vertex*)
    - Si sa valence = 3 → enlever ce vertex en taggant le *front vertex* et les faces connectées comme *To be removed*, ajout de nouvelle gate




    - Si (l'état est *free* et valence != 3) ou l'état *conquered* → marquer le *front face* comme *conquered*, ajouter les nouvelles *gates* dans la queue et *Null patch* donné en sortie.

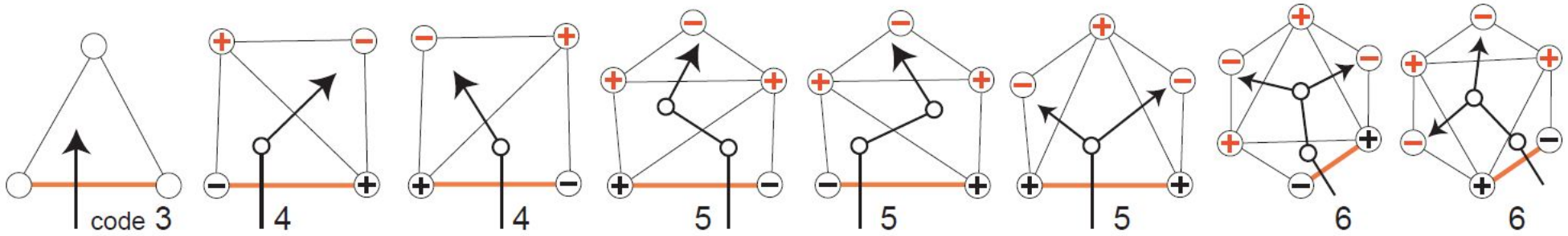
# Reconstruction

- **Cleaning reconquest :**
  - Récupérer de la liste “output” de nettoyage composé de *Null patch* et de paire valence/vertex
  - Parcours des gates comme pour la Cleaning conquest
    - Récupérer une *gate* de la *queue* (FIFO)
    - Regarder l'état de la face où on entre
      - Si l'état est *conquered* → gate suivante
    - Sinon regarder élément suivant dans la liste “output”
      - Si paire valence/vertex (valence = 3) → ajouter du vertex et des faces correspondantes, marquer les faces du *patch* et adjacentes en *conquered*, ajouter les nouvelles gates
      - Si *Null patch* → marquer face comme *conquered* et ajouter les nouvelles gates



# Reconstruction

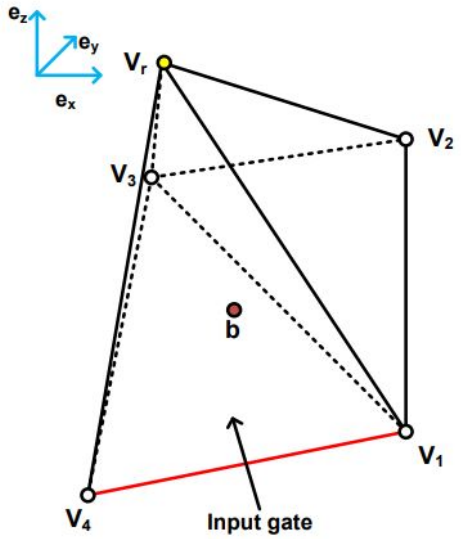
- **Decimating reconquest :**
  - Reprendre liste “output” de décimation composer de *Null patch* et de paire valence/vertex
  - Conquérir les gates comme la décimation
    - Récupérer une *gate* de la *queue* (FIFO)
    - Regarder l'état de la face où on entre
      - Si l'état est *conquered* → gate suivante
    - Sinon regarder élément suivant dans la liste “output”
      - Si paire valence/vertex → retrouver le *patch* grâce à la valence et *retriangulation tags* d’entrée, ajouter le vertex et les nouvelles faces, marqué les nouveaux éléments en *conquered* et ajouter les nouvelles gates




      - Si *Null patch* → marquer face comme *conquered* et ajouter les nouvelles gates

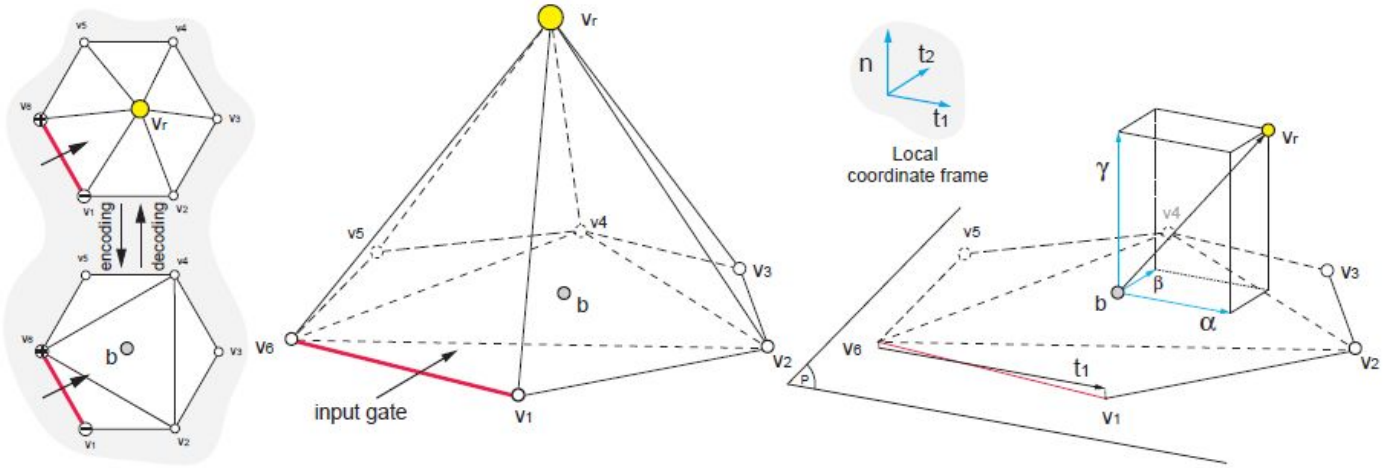
# Elèments non traités : encodage géométrie

1. **Barycentric Prediction**
   - Réalisée lors de l'étape de décodage pour la reconstruction
   - Le sommet est ajouté au milieu d'un patch dont les voisins sont connus du codeur et du décodeur
   - Le barycentre de tous les sommets du patch est utilisé comme première approximation de la position du nouveau sommet.



# Elèments non traités : encodage géométrie

2. **Approximate Frenet Coordinate Frame:**
   - Approximer la normale n, par une somme pondérée des normales de chaque triangle présent dans le patch.
   - Utiliser la normale et le barycentre du patch pour définir le plan tangent approximatif de la surface
   - projeter le gate sur le plan tangent pour obtenir $t_1$
   - utilise le *cross product* de $t_1$ et n pour obtenir $t_2$



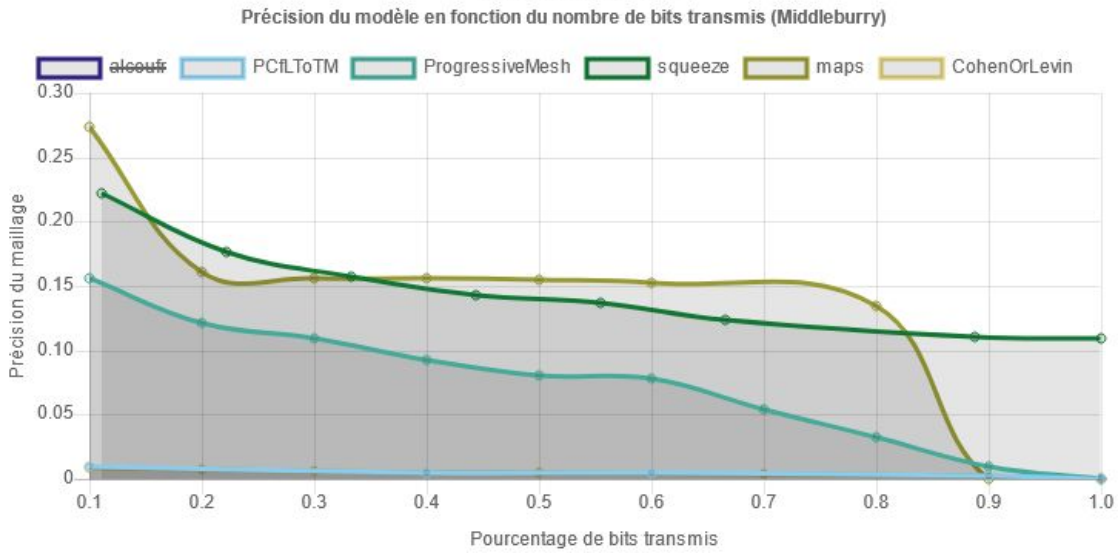
# Elèments non traités : encodage géométrie

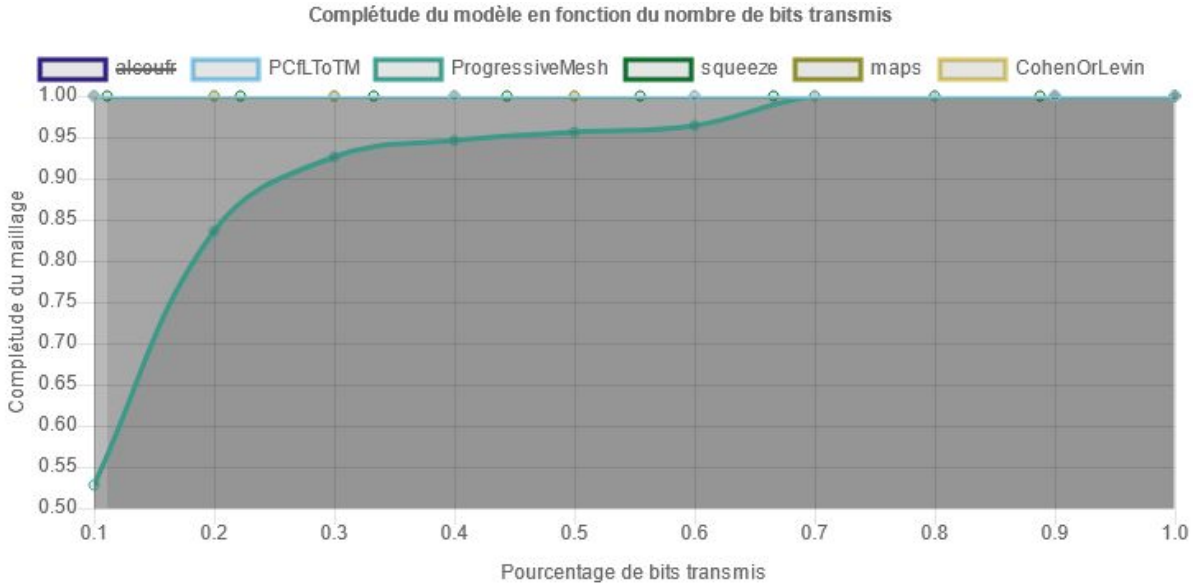
**3. Quantization of Frenet Frame Coordinates:** encoder les positions des nouveaux sommets créés au milieu d'un patch

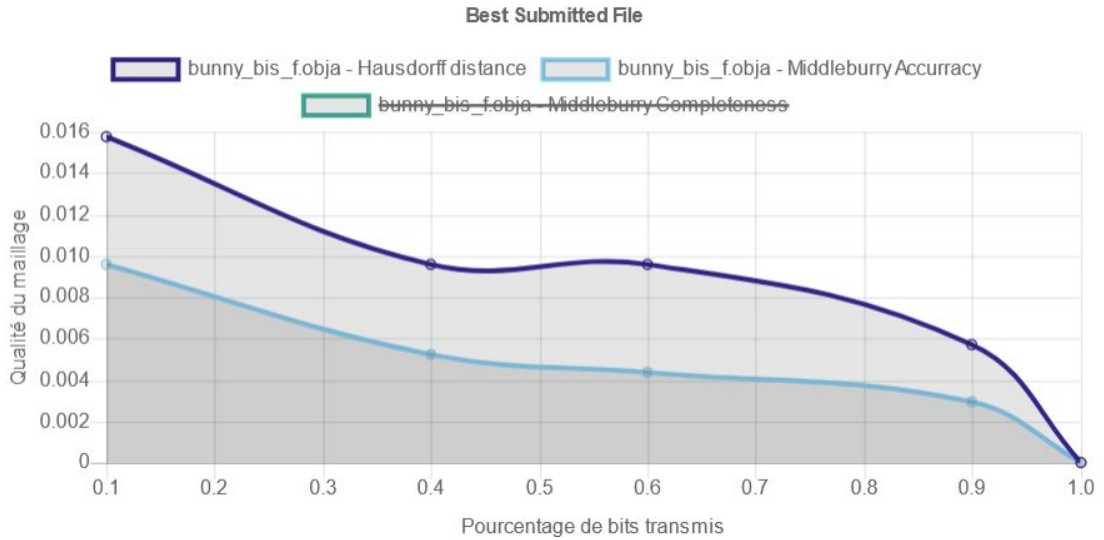
- $\mathbf{v_r}$ → la position du sommet que nous voulons maintenant coder/décoder.
- Les nouvelles coordonnées de *Frenet frame* : $v_r = b + \alpha.t_1 + \beta.t_2 + \gamma.n$
- Arrondir ces coordonnées $(\alpha,\beta,\gamma)$, aux valeurs entières les plus proches
- Encoder à l'aide d'un code de longueur variable qui tient compte de la distribution de l'erreur de quantification
- Le décodeur n'aura qu'à ajouter les coordonnées de Frenet au barycentre pour trouver la position finale du vertex inséré.

# Résultats

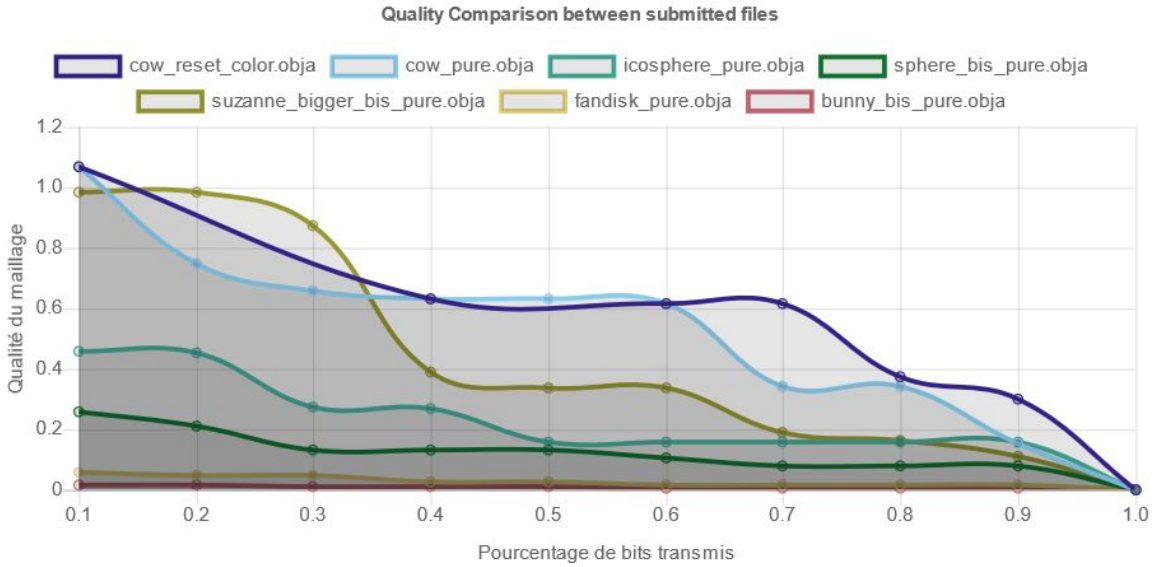

Précision du modèle en fonction du nombre de bits transmis (Distance de Hausdorff)
alcoufr
PCfLToTM
ProgressiveMesh
squeeze
maps
CohenOrLevin
Précision de la compression
Nombre de bits transmis

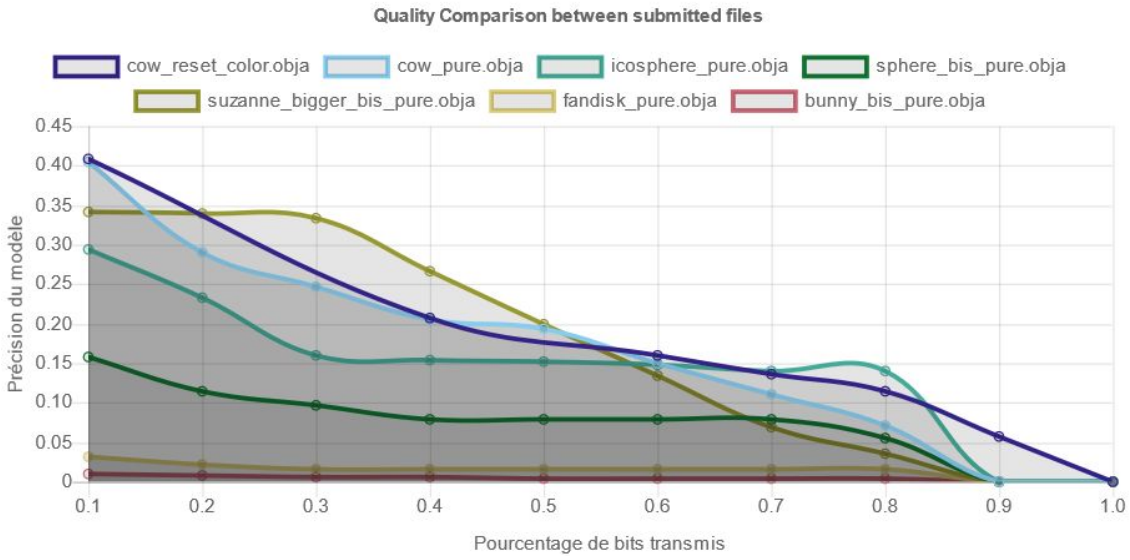

Précision du modèle en fonction du nombre de bits transmis (Middleburry)
alcoufr
PCfLToTM
ProgressiveMesh
squeeze
maps
CohenOrLevin
Précision du maillage
Pourcentage de bits transmis

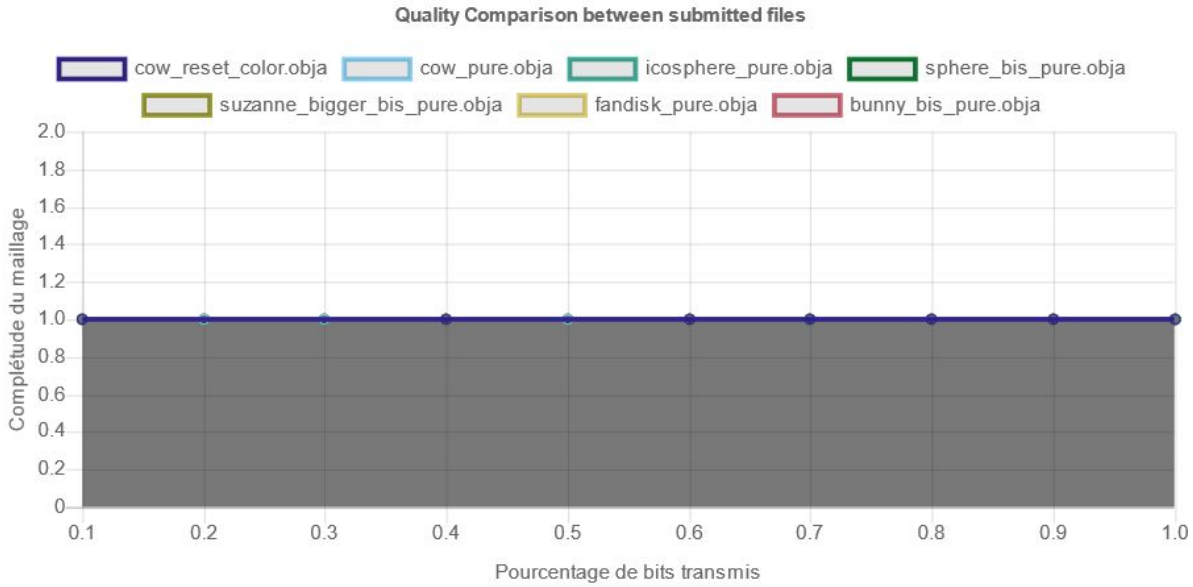

Complétude du modèle en fonction du nombre de bits transmis
alcoufr
PCfLToTM
ProgressiveMesh
squeeze
maps
CohenOrLevin
Complétude du maillage
Pourcentage de bits transmis

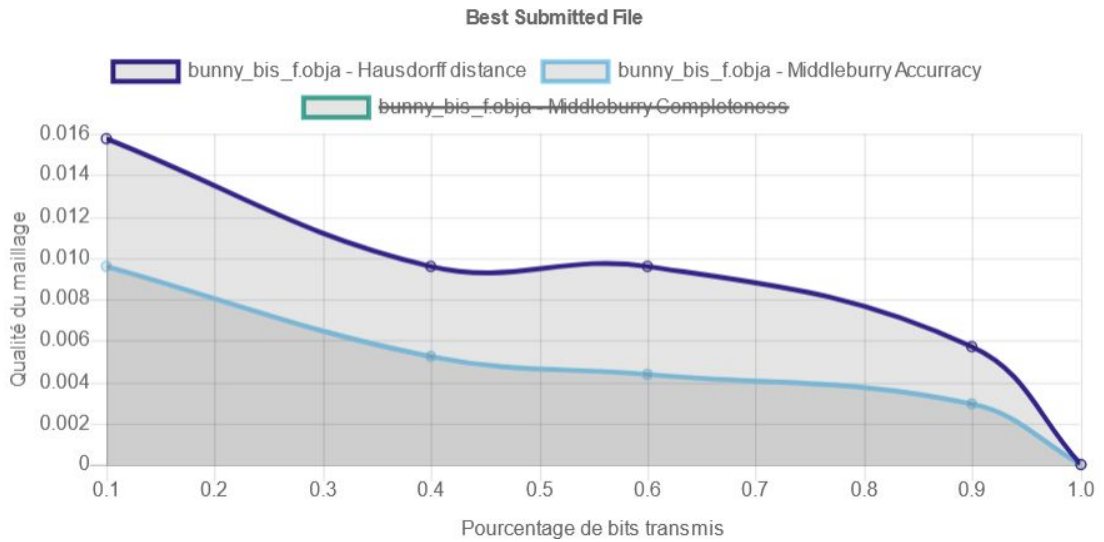

Best Submitted File
bunny_bis_f.obja - Hausdorff distance
bunny_bis_f.obja - Middleburry Accurracy
bunny_bis_f.obja - Middleburry Completeness
Qualité du maillage
Pourcentage de bits transmis

# Résultats

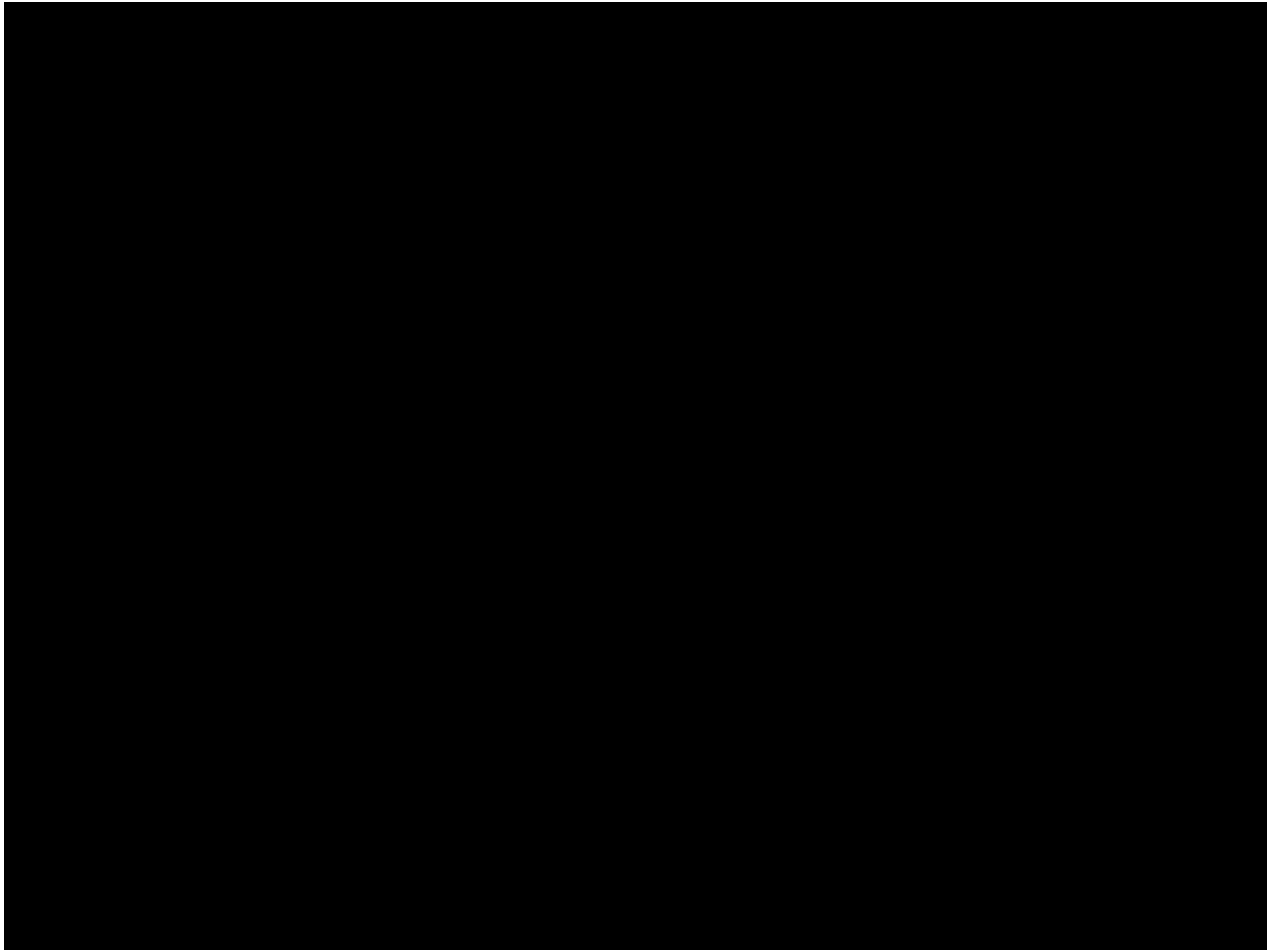

Quality Comparison between submitted files
cow_reset_color.obja
cow_pure.obja
icosphere_pure.obja
sphere_bis_pure.obja
suzanne_bigger_bis_pure.obja
fandisk_pure.obja
bunny_bis_pure.obja
Qualité du maillage
Pourcentage de bits transmis


Quality Comparison between submitted files
cow_reset_color.obja
cow_pure.obja
icosphere_pure.obja
sphere_bis_pure.obja
suzanne_bigger_bis_pure.obja
fandisk_pure.obja
bunny_bis_pure.obja
Précision du modèle
Pourcentage de bits transmis


Quality Comparison between submitted files
cow_reset_color.obja
cow_pure.obja
icosphere_pure.obja
sphere_bis_pure.obja
suzanne_bigger_bis_pure.obja
fandisk_pure.obja
bunny_bis_pure.obja
Complétude du maillage
Pourcentage de bits transmis


Best Submitted File
bunny_bis_f.obja - Hausdorff distance
bunny_bis_f.obja - Middleburry Accurracy
bunny_bis_f.obja - Middleburry Completeness
Qualité du maillage
Pourcentage de bits transmis

# Résultat - Icosphère



Git deposit du code :

https://github.com/Roropey/CSI_Project