

GAUTHIER Silvère
LAMEIRA Yannick
PELADAN Cécile



Année : 2014-2015

SOMMAIRE

- **Introduction**
 - Sujet
 - Objectifs Principaux
- **Développement**
 - API
 - Modèles
 - Outils
 - Interface
- **Conclusion et Perspectives**
 - Conclusion
 - Perspectives

Introduction

- Sujet

Introduction

- Objectifs principaux
 - maillage malléable, rendu rapide, contenu diversifié, Algorithmes, réutilisabilité et extensibilité, ergonomie et intuitivité

Développement

API

« *Application Programming Interface* »

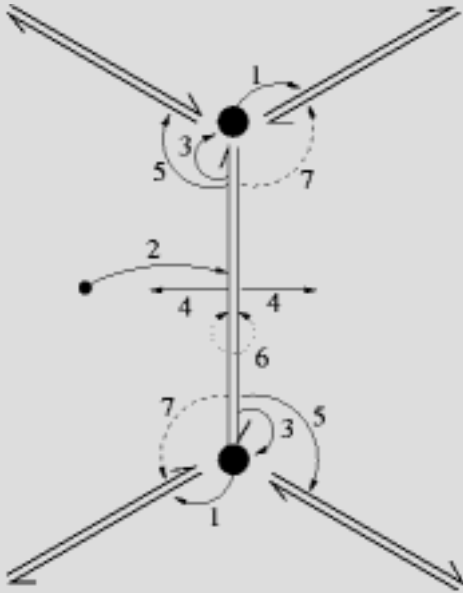
Ce terme désigne un ensemble normalisé de classes, méthodes ou fonctions servant de façade.

La programmation se fait alors en réutilisant des « *briques* » de fonctionnalités présentes dans l'API, sans avoir besoin de connaître les détails internes.

Développement : API

Structure de Maillage

« *HalfEdge data structure* »



Une arête est décomposée en deux demi-arêtes orientées

Pour chaque demi-arête, on mémorise un pointeur vers la demi-arête opposée, le sommet vers lequel elle pointe, la face à laquelle elle appartient, la demi-arête suivante et la demi-arête précédente.

Chaque sommet contient un pointeur sur une demi-arête sortante et chaque face, un pointeur vers une de ses demi-arêtes.

Très pratique pour parcourir un maillage de diverses façons.

Octets par sommet ? (120)

```
typedef struct HalfEdge {  
    Vertex* vertex;  
    Face* face;  
    HalfEdge* next;  
    HalfEdge* opposite;  
    HalfEdge* previous;  
} HalfEdge;
```

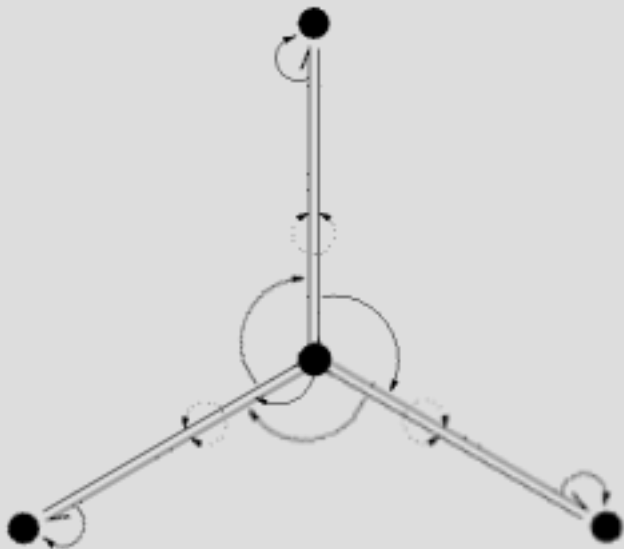
```
typedef struct Vertex {  
    QVector3D coords;  
    HalfEdge* outgoing;  
    int index;  
} Vertex;
```

```
typedef struct Face {  
    HalfEdge* edge;  
} Face;
```

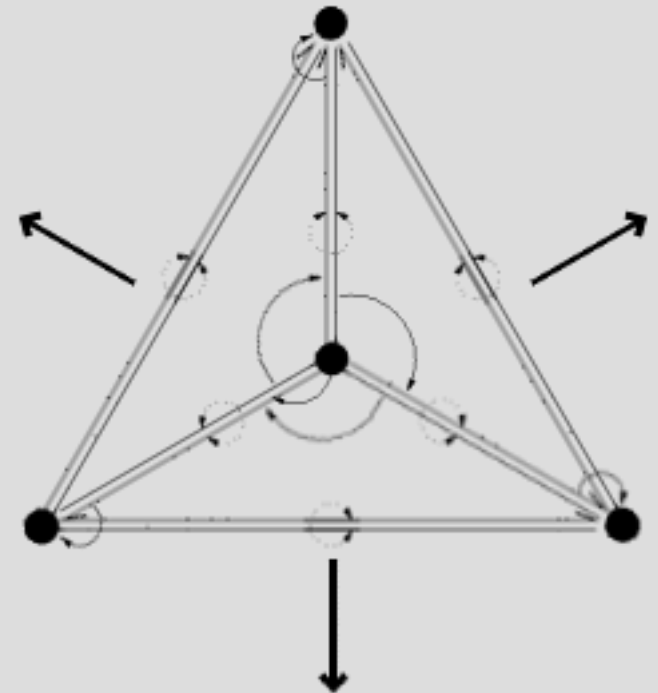
Développement : API

Structure de Maillage « *HalfEdge data structure* »

Voisinage des points



Voisinage des faces



Développement : API

- Fonctions de la surcouche

```
// ===== PUBLIC MEMBERS (high level) =====
public:
// Add a face with triangles fan wich begins in vertices[0], vertices must be given counterclockwise.
bool addFace(QVector<QVector3D> vertices);

// Cut an edge between two vertices in two parts equally
bool cutEdge(QVector3D vertex1, QVector3D vertex2);

// Merge the two edges vertex1-to-vertex2 and vertex2-to-vertex3
bool mergeEdge(QVector3D vertex1, QVector3D vertex2, QVector3D vertex3);

// Get a list of vertices in area "areaSize" around "position"
QVector<QVector3D> getVertices(QVector3D position, float areaSize);

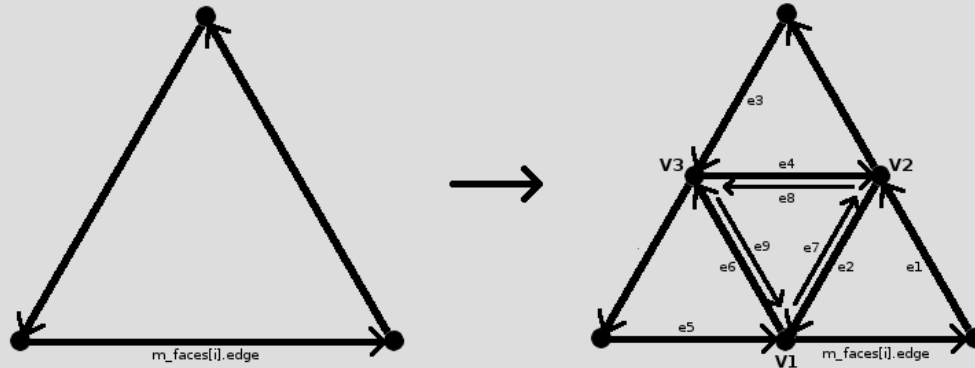
// Get a list of vertices neighbours of "vertex" with "degree" recursions around
QVector<QVector3D> getNeighbours(QVector3D vertex, int degree);

// Get the normal of the face containing position
QVector3D getNormal(QVector3D position);

// Move a vertex or many vertices by "move"
void moveVertex(QVector3D vertex, QVector3D move);
void moveVertex(int index, QVector3D move);
void moveVertices(QVector<QVector3D> vertices, QVector3D move);
// =====
```


Développement : API

Algorithmes : subdivisions



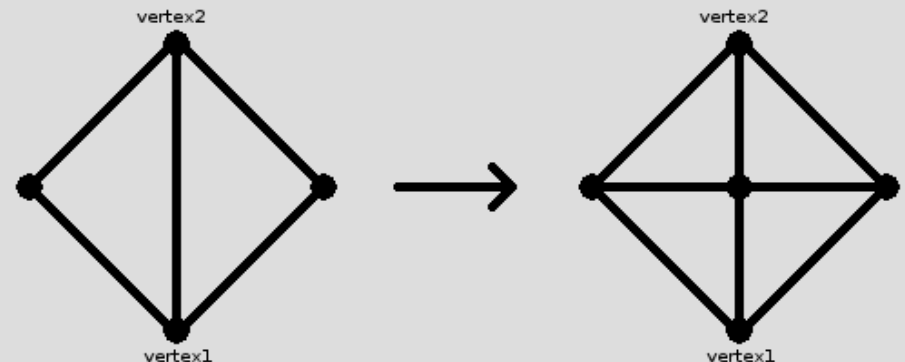
Subdivision globale :

- Couper le triangle en 4
- Raccorder les arêtes et points

VERSUS

Subdivision locale :

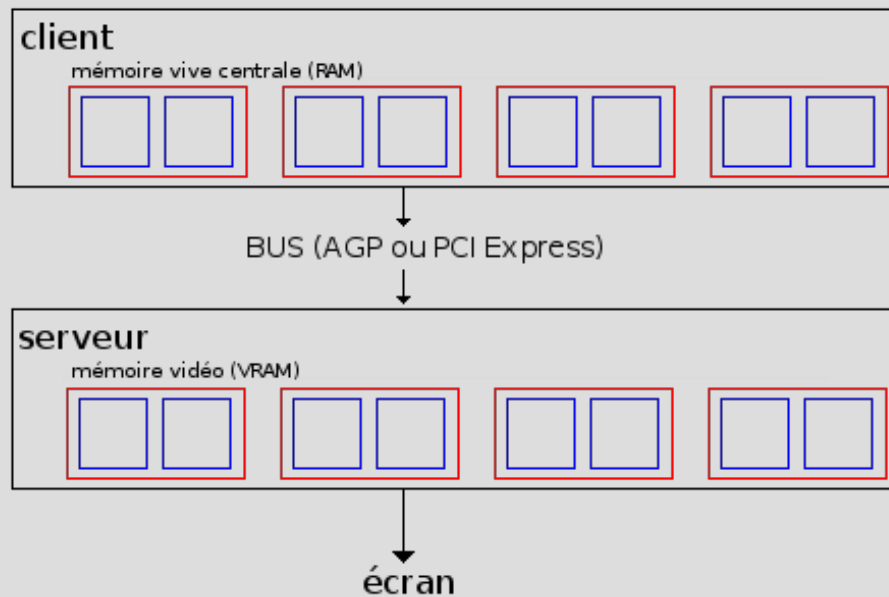
- Couper l'arête en deux
- Raccorder les points



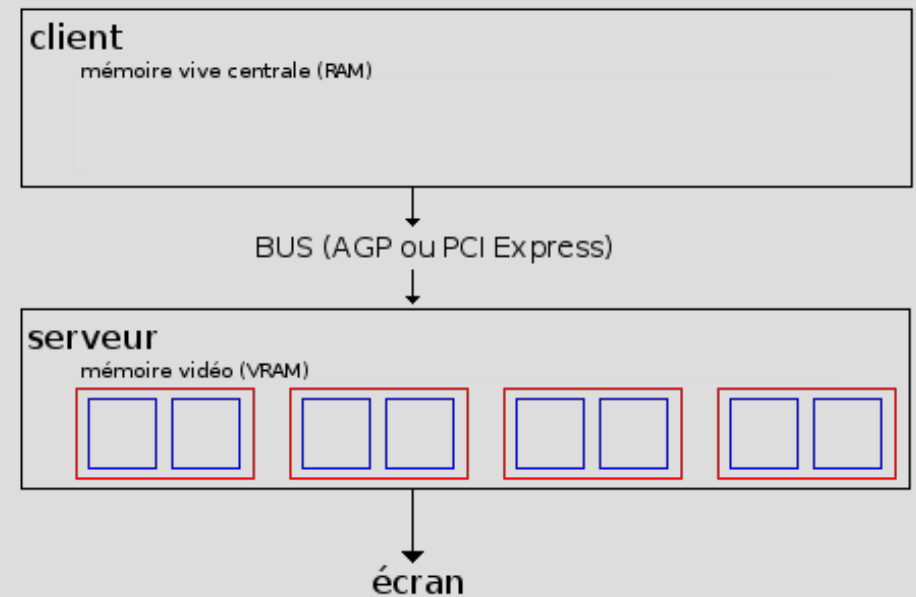
Développement : API

Technique de rendu

Vertex Array



Vertex Buffer Object



Gain de performance avec les Vertex Buffer Objects

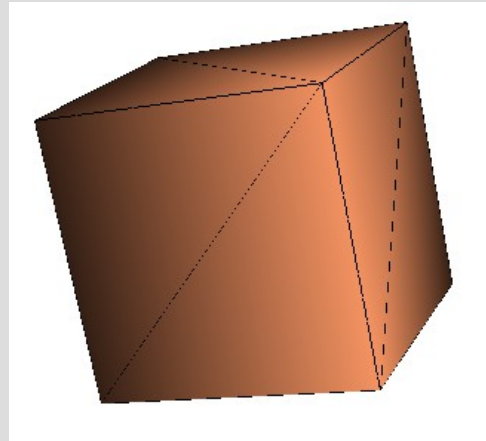
Modèles

- Nous avons 5 modèles implémentés définis par :
 - des paramètres géométriques,
 - un paramètre pour le pas de discrétisation.
- Pour créer chaque face, on utilise :
`addFace(QVector<QVector3D> vertices)`

Modèles

- Cube :

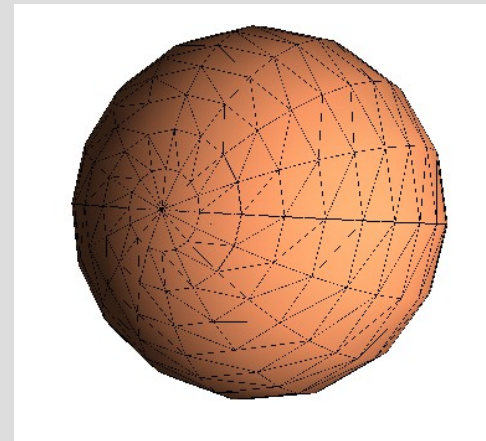
- Pour chaque point :
x varie entre $[-w/2; w/2]$
y varie entre $[-h/2; h/2]$
z varie entre $[-d/2; d/2]$



- Sphère :

- Pour chaque point, soient θ pour la latitude avec $\theta \in [0 ; \pi[$ et ϕ pour la longitude avec $\phi \in [0 ; 2\pi[$

$$\begin{aligned}x &= r \cos \theta \cos \phi \\y &= r \cos \theta \sin \phi \\z &= r \sin \theta\end{aligned}$$



Modèles

- Cylindre et Cône

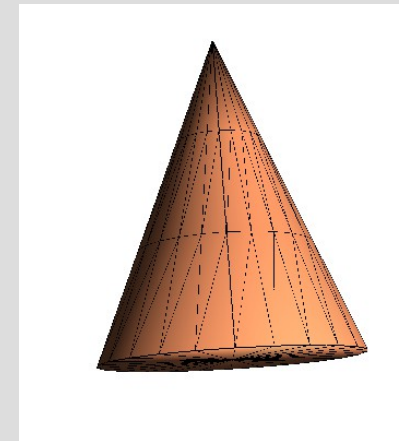
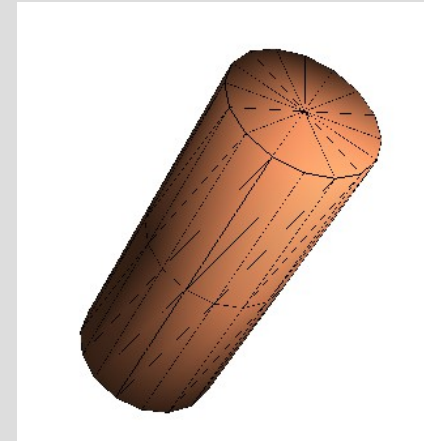
- Pour chaque point, soit $\alpha \in [0 ; 2\pi[$.

$$x = radius * \cos(\alpha * u)$$

$$y = \frac{\pm height}{2}$$

$$z = radius * \sin(\alpha * u)$$

- Pour le cône, on a une variation du radius selon la hauteur (thalès).



Modèles

- Cylindre et Cône

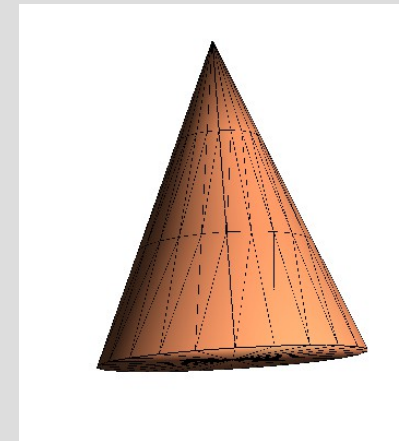
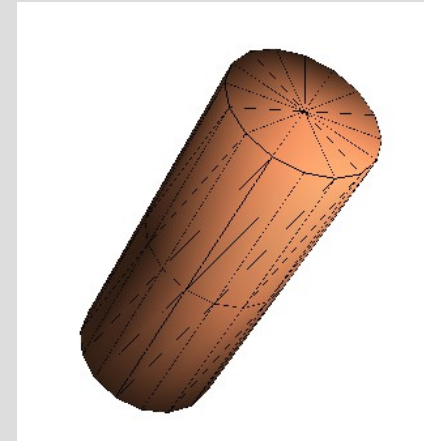
- Pour chaque point, soit $\alpha \in [0 ; 2\pi[$.

$$x = radius * \cos(\alpha * u)$$

$$y = \frac{\pm height}{2}$$

$$z = radius * \sin(\alpha * u)$$

- Pour le cône, on a une variation du radius selon la hauteur (thalès).



Modèles

- Tore :
 - Pour chaque point :

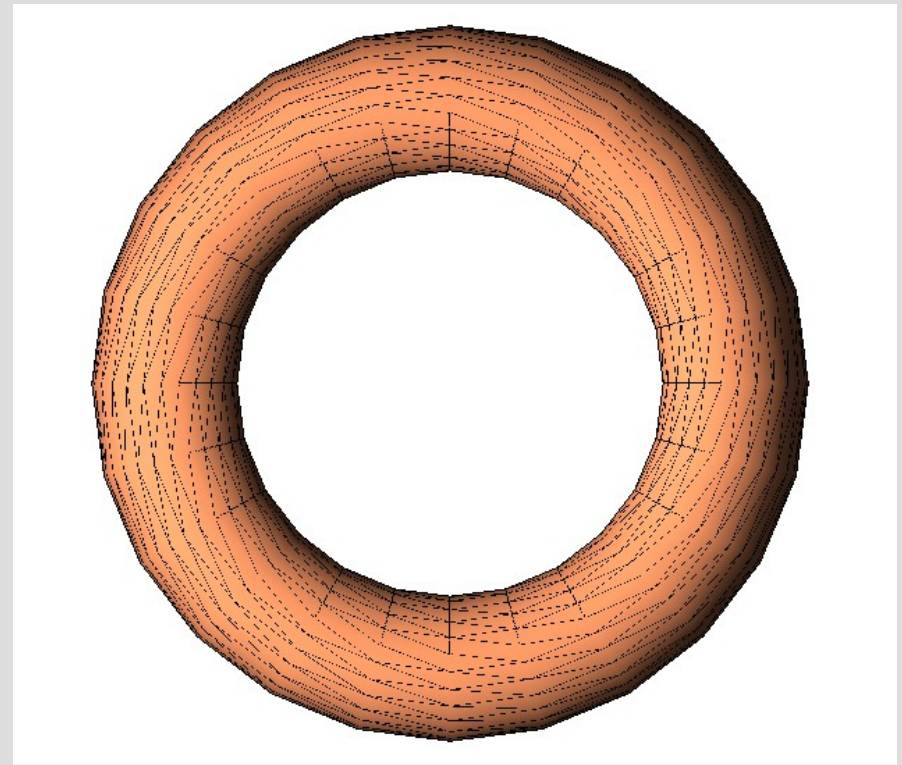
$$\begin{aligned}x(u, v) &= (R + r \cos v) \cos u \\y(u, v) &= (R + r \cos v) \sin u \\z(u, v) &= r \sin v\end{aligned}$$

Où :

u, v appartiennent à l'intervalle $[0, 2\pi[$,

R est la distance entre le centre du tube et le centre du tore,

r est le rayon du cercle C .



Outils

- **Définition de « outils » :**

« Les outils permettent de modifier les maillages suivant leurs spécifications sur un objet donné grâce à un brush. »

- **Outils à modification globale :**

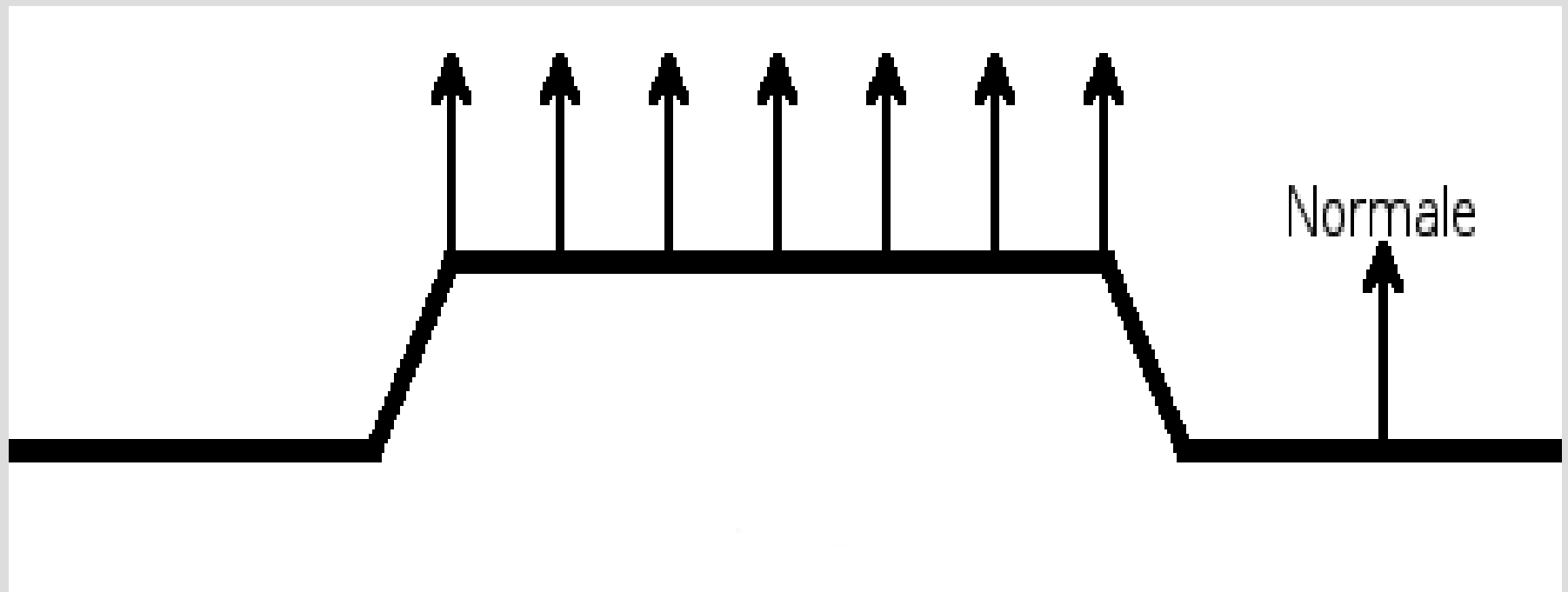
- *gtmove* : permet de déplacer un objet dans un repère scène.

- *gtrotate* : permet de tourner un objet dans un repère scène.

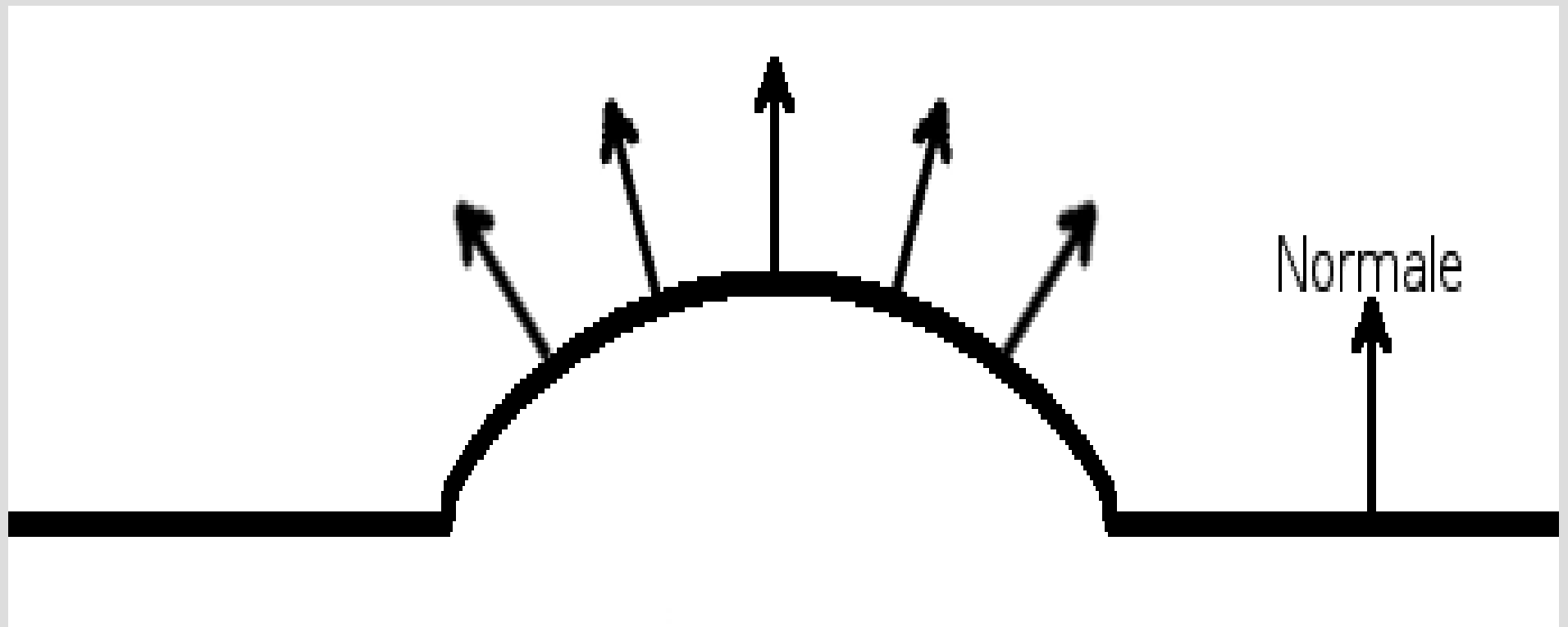
- *gtscale* : permet d'agrandir ou de rétrécir un objet dans un repère scène.

- **Outils à modification locale :**

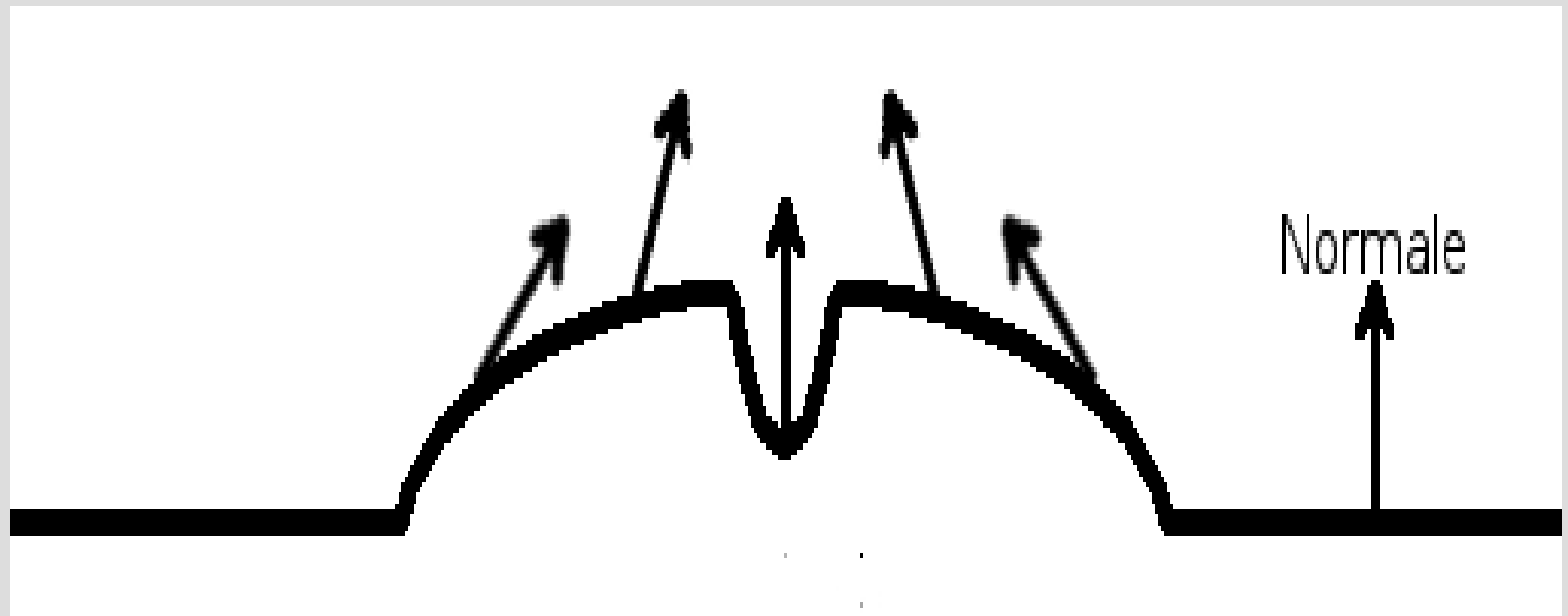
- *Ajout de matière :*



- *Gonflement de matière :*



- *Pincement de la surface de l'objet :*



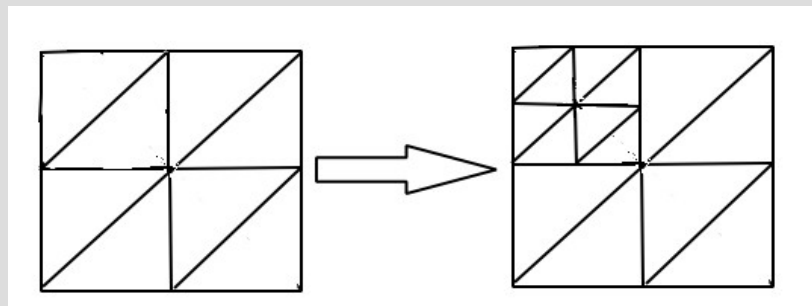
- *Autres outils*

Lissage de matière

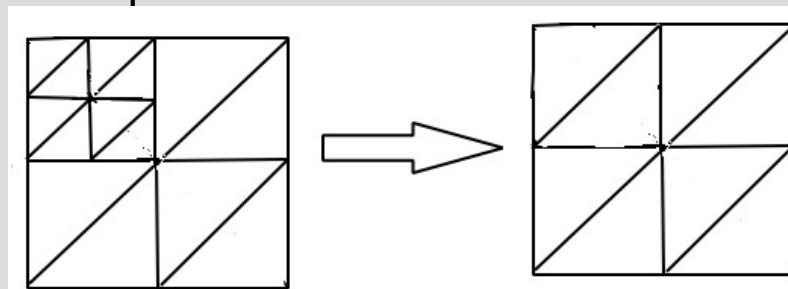
Déplacement de matière

Outils Spéciaux :

Subdivision automatique :



Décimation automatique :



- Vidéo montrant les outils

Conclusion et Perspectives

- Conclusion
→ état actuel... fonctionnalité, résultats...etc

Conclusion et Perspectives

- Perspectives
→ améliorations majeures...etc

Merci !



Merci de nous avoir écouté !

- Team OpenSculpt -