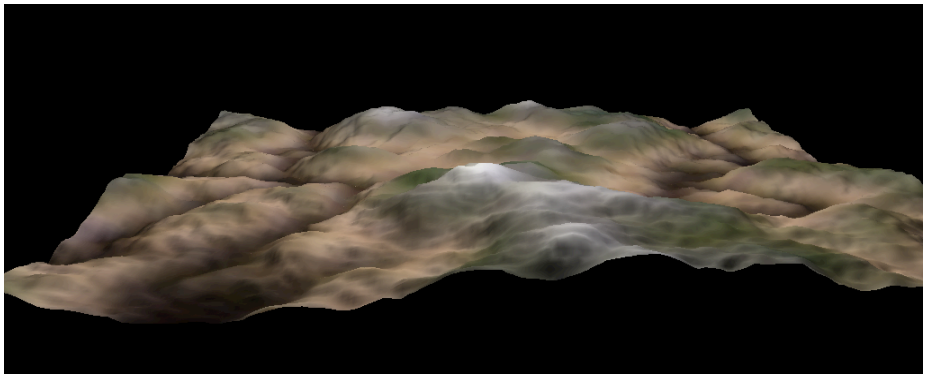
TD2 - Modélisation : vase, terrain



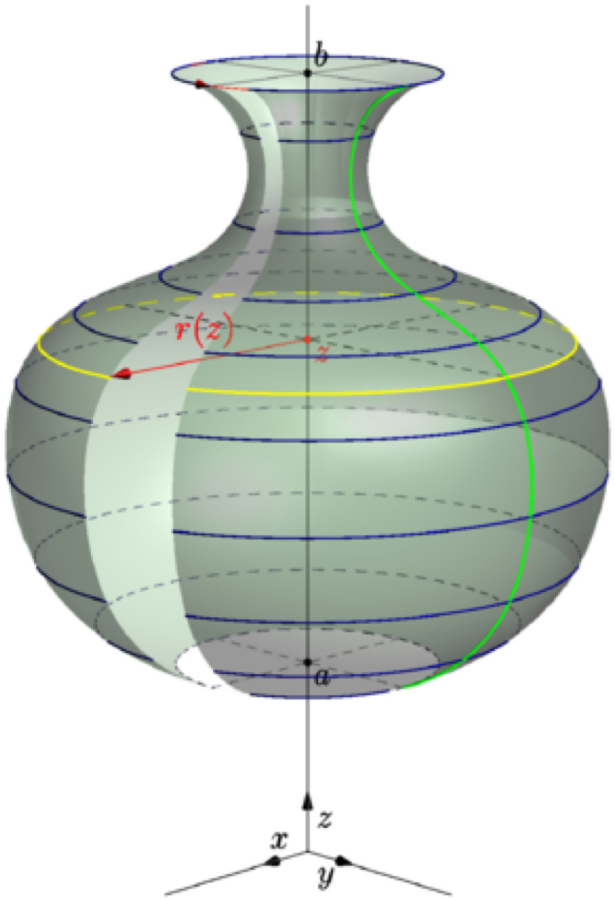
Eric Galin (semestre printemps)

Florence Zara (semestre automne)

LIRIS-ORIGAMI, Université Lyon 1

Exercice 1 - Création d'un vase

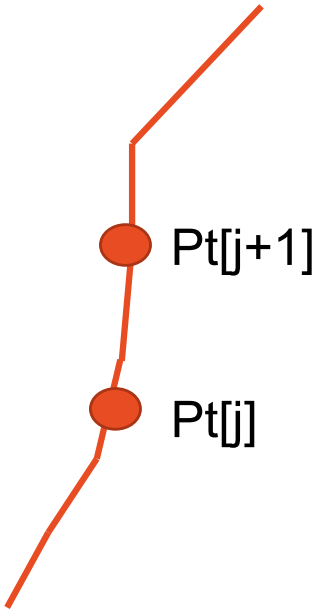
- Comment créer un vase par révolution ?



Création du vase

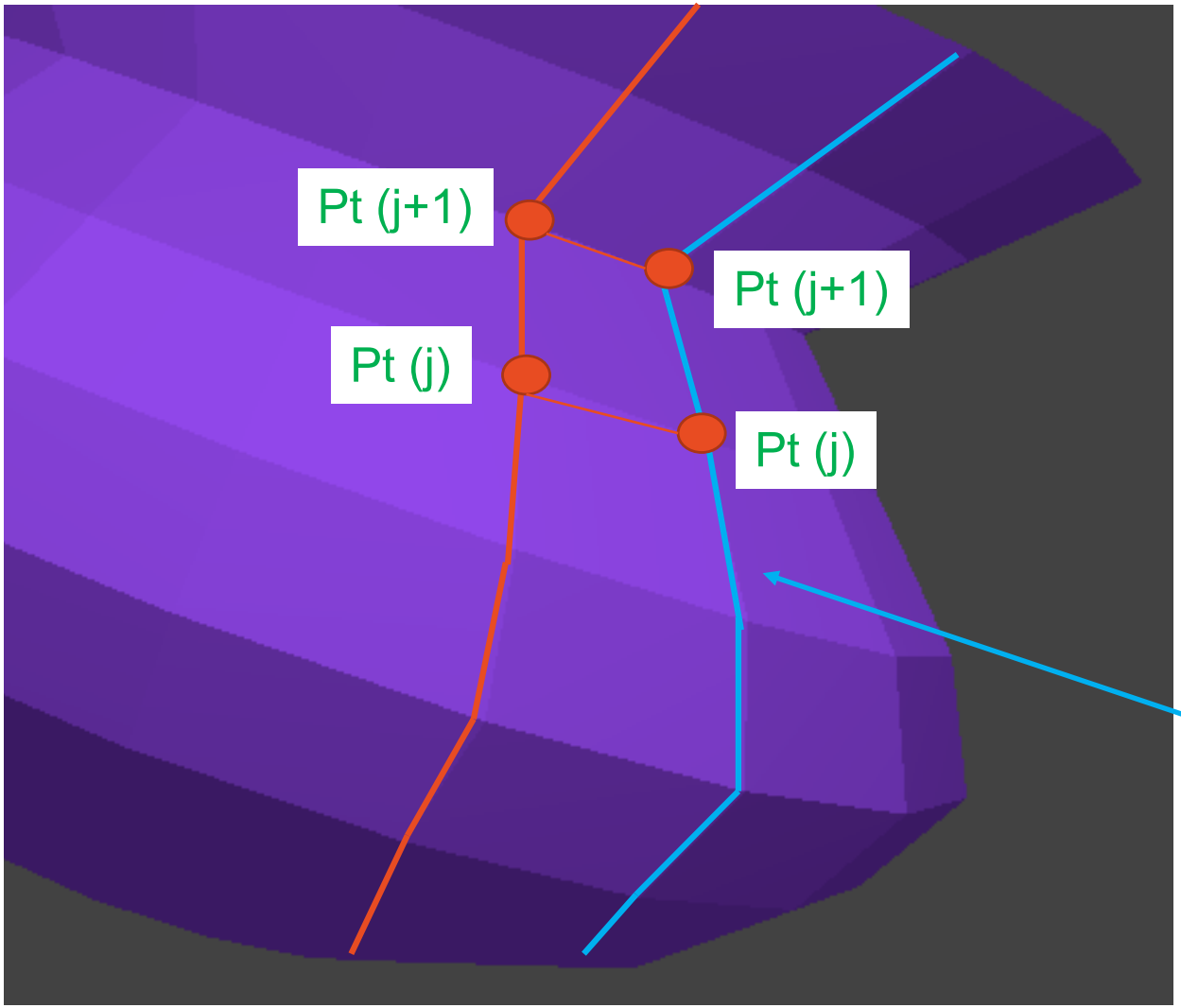
Etape 1 = définition de la silhouette 2D

```
int ViewerEtudiant::create_vertex_normal_vase(){

    // Nombre de points de la silhouette 2D
    vase_NBPT = 10;  // déclaré dans la class ViewerEtudiant

    /// Points de la silhouette 2D
    vase_p[0] = Point(0,0,0);
    vase_p[1] = Point(0.6,0.2,0);
    vase_p[2] = Point(1.0,0.4,0);
    vase_p[3] = Point(1.2,0.6,0);
    vase_p[4] = Point(1.3,0.8,0);
    vase_p[5] = Point(1.2, 1.0, 0);
    vase_p[6] = Point(1.0, 1.2,0);
    vase_p[7] = Point(0.8,1.4,0);
    vase_p[8] = Point(1,1.6,0);
    vase_p[9] = Point(1.2,1.8,0) ;
```



Création du vase

Etape 2 = définition des points par révolution



Rotation autour axe y

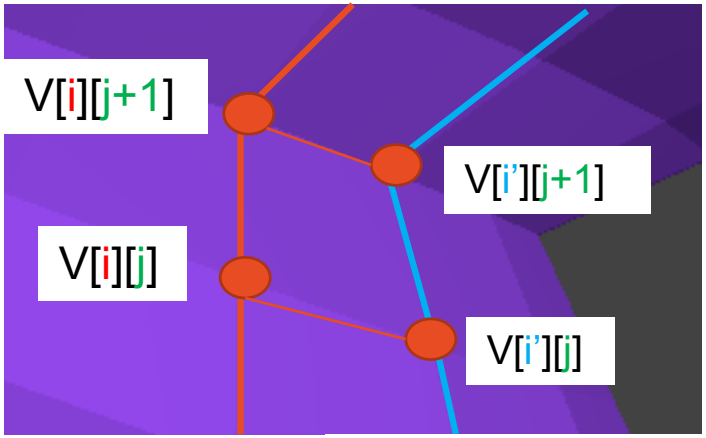
Variation angle de 0 à 2π

Points obtenus par rotation de la silhouette

Création du vase

Etape 2 = définition des points par révolution

```
// Nombre de rotations pour créer le vase par rotation
    vase_NBROT = 20; // déclaré dans la class ViewerEtudiant
```

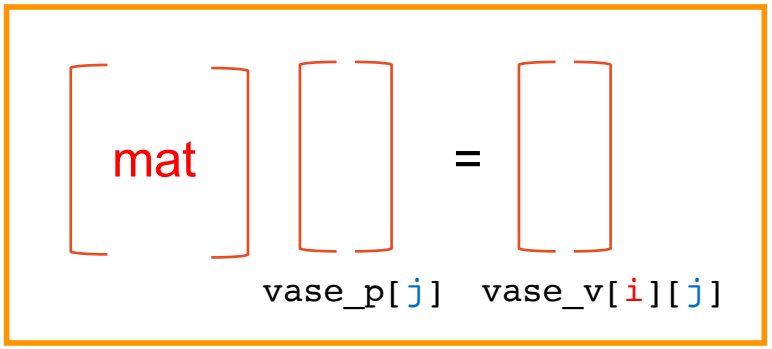


Points ayant subi la rotation i

Points ayant subi la rotation i' = (i+1) % vase_nbrot

[Matrice de rotation autour axe des y angle **θ**] [Point j de la silhouette `vase_p[j]`] = [Point j ayant subi la rotation i `vase_v[i][j]`]

Création du vase - Calcul des coordonnées des sommets (stockées dans `vase_v[i][j]`)




```
for(int i=0; i < vase_NBROT; i++){

//i indice rotation (angle : 2 pi * i / nbrot)

    // Angle qui varie de 0 à 2 pi
    float teta = 2 * M_PI * i / vase_NBROT;

    // Matrice de rotation de l'angle theta autour axe des y
    // en coordonnées homogènes : 4 x 4
    float mat[16] = {
        cos(teta),    0,    -sin(teta),    0,
        0,            1,    0,             0,
        sin(teta),    0,    cos(teta),     0,
        0,            0,    0,             1 };

    // Calcul des coordonnées des sommets
    for(int j=0; j < vase_NBPT; j++){
    // j indice du point de la silhouette
     // Application de ma matrice de rotation au point j qui subit la rotation (2 pi * i / nbrot)
     vase_v[i][j].x = mat[0] * vase_p[j].x + mat[1] * vase_p[j].y + mat[2]  * vase_p[j].z + mat[3]  * 1;
     vase_v[i][j].y = mat[4] * vase_p[j].x + mat[5] * vase_p[j].y + mat[6]  * vase_p[j].z + mat[7]  * 1;
     vase_v[i][j].z = mat[8] * vase_p[j].x + mat[9] * vase_p[j].y + mat[10] * vase_p[j].z + mat[11] * 1;

    }//for_i
}//for_i
```

Création du vase - Calcul des normales aux sommets

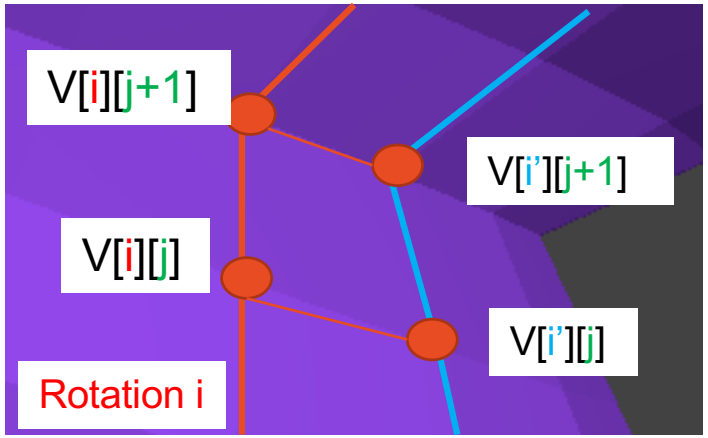
- On va calculer la normale au sommet [i][j]
- On la répercute sur les 3 autres sommets de la face
- Normale au sommet = moyenne des normales des faces adjacentes

Création du vase - Calcul des normales aux sommets (stockées dans `vase_vn[i][j]`)

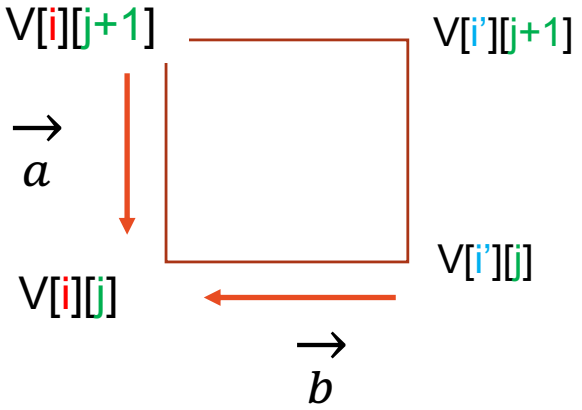
```
// Initialise à 0 les normales
 for(int i=0; i<vase_NBROT; i++)
     for(int j=0; j<vase_NBPT; j++)
          vase_vn[i][j] = Vector(0, 0, 0);
```

rotation i

point j



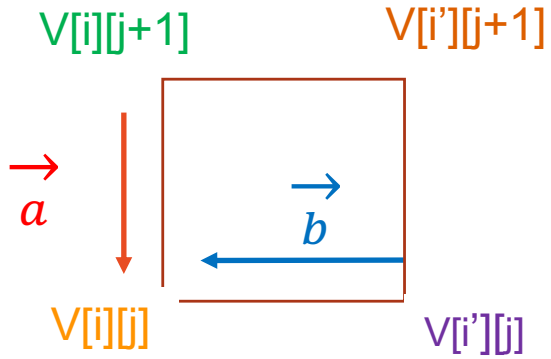
Rotation i' = (i+1) % nbrot



Normale au sommet [i][j] =

$$\vec{a} \wedge \vec{b}$$

Création du vase - Calcul des normales aux sommets




```
for(int i=0; i< vase_NBROT; i++){
   for(int j=0; j< vase_NBPT-1; j++){
       Vector a, b, vntmp;

       a = normalize(vase_v[i][j] - vase_v[i][j+1]);
       b = normalize(vase_v[i][j] - vase_v[(i+1) % vase_NBROT][j]);

       vntmp = cross(a, b); // Produit vectoriel = a ∧ b

       // On répercute cette normale sur les 4 sommets de la face
       // (accumulation des normales)
       vase_vn[i][j] = vntmp + vase_vn[i][j];
       vase_vn[(i+1) % vase_NBROT][j] = vntmp + vase_vn[(i+1) % vase_NBROT][j];
       vase_vn[(i+1) % vase_NBROT][j+1] = vntmp + vase_vn[(i+1) % vase_NBROT][j+1];
       vase_vn[i][j+1] = vntmp + vase_vn[i][j+1];
   }
}
```

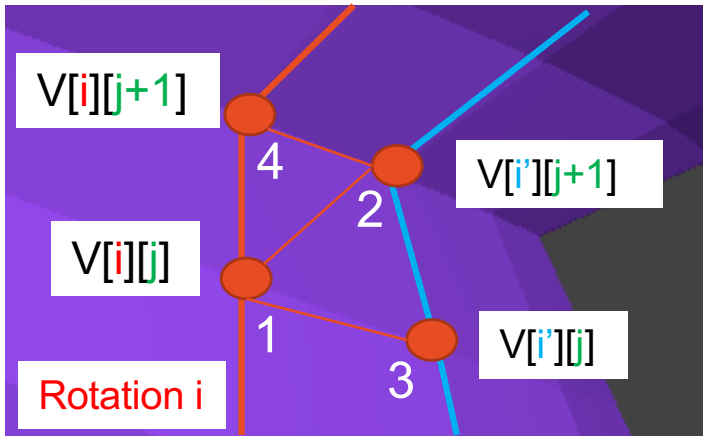
Création du vase - Calcul des normales aux sommets

```
// Normale à un sommet = moyenne de la normale des 4 sommets de la face

    for(int i=0; i<vase_NBROT; i++){

        for(int j=0; j<vase_NBPT; j++){

            float q = 4.0f;

            if (j == vase_NBPT-1) // Points du bord
                q = 2.0f;

            vase_vn[i][j] =  vase_vn[i][j] / q;

        }//for_i

    }//for_j

}//void_create_vertex_normal_vase
```

Création vase - Création du Mesh

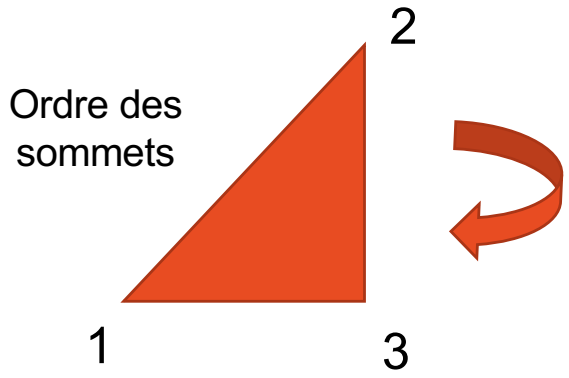
Mesh m_vase; // Déclaré dans ViewerEtudiant

```
void ViewerEtudiant::init_vase()
{
    m_vase = Mesh(GL_TRIANGLES); // Primitive OpenGL

    m_vase.color(1.0, 1.0, 1.0); // Couleur du Mesh
```

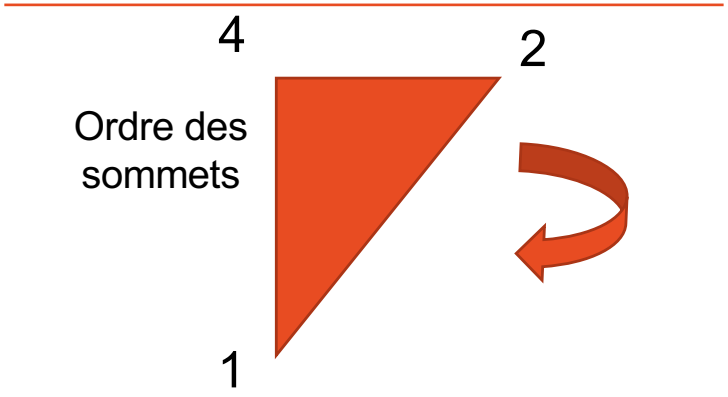


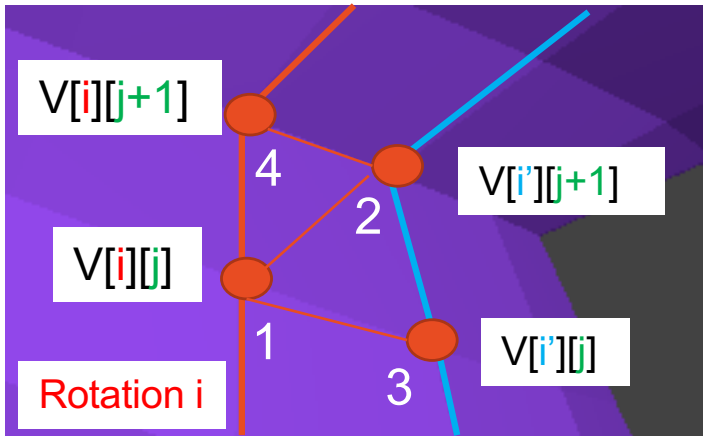
Rotation i' = (i+1) % nbrot

```
    for(int i=0; i<vase_NBROT; i++){

        for(int j=0; j<vase_NBPT-1; j++){ // Attention boucle de 0 à vase_NBPT-2 car (j+1)

            // Premier triangle

1           m_vase.normal(vase_vn[i][j]);
            m_vase.vertex(vase_v[i][j]);

2           m_vase.normal(vase_vn[(i+1) % vase_NBROT][j+1]);
            m_vase.vertex(vase_v[(i+1) % vase_NBROT][j+1]);

3           m_vase.normal(vase_vn[(i+1) % vase_NBROT][j]);
            m_vase.vertex(vase_v[(i+1) % vase_NBROT][j]);
```



Création vase - Création du Mesh





Rotation i' = (i+1) % nbrot

```
        // Second triangle

1           m_vase.normal(vase_vn[i][j]);
            m_vase.vertex(vase_v[i][j]);

4           m_vase.normal(vase_vn[i][j+1]);
            m_vase.vertex(vase_v[i][j+1]);

2           m_vase.normal(vase_vn[(i+1) % vase_NBROT][j+1]);
            m_vase.vertex(vase_v[(i+1) % vase_NBROT][j+1]);


        }//for_j

    }for_i

}//void
```

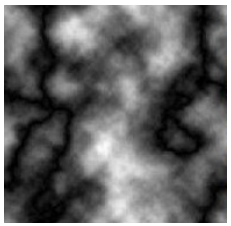
Création et affichage du Mesh du vase

```
int ViewerEtudiant::init(){
  // Calcul des coordonnées des sommets et des normales
   create_vertex_normal_vase();

   // Création du Mesh à partir des coordonnées calculées
   init_vase();
}

int ViewerEtudiant::render(){
   // Transformation géométrique appliquée au Mesh du vase
   Transform T = Translation( 2, 0, 0 );

   // Affichage du vase
   draw_vase(T);
}
```

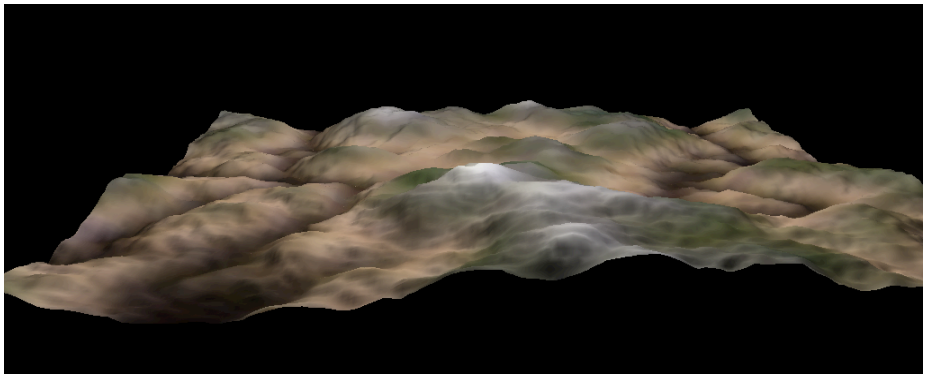
```
void ViewerEtudiant::draw_vase(const Transform& T){
    gl.model(T);      // Applique Transform T
    gl.draw(m_vase); // Affichage du Mesh
}
```

Exercice 2 - Création d'un terrain

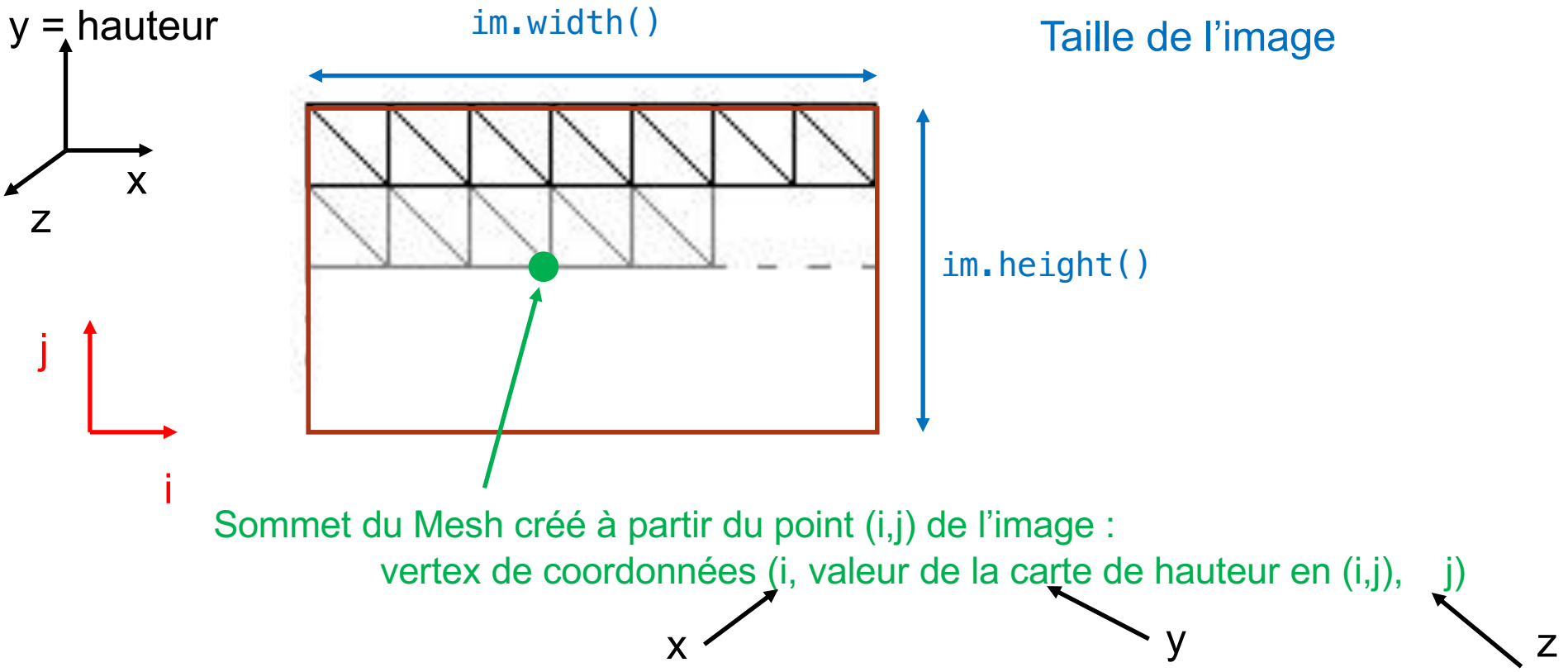
- Comment créer un terrain à partir d'une carte de hauteur ?
  - Carte de hauteur = image (niveau de gris) à charger



  - Création du Mesh (définition coordonnées des sommets)
  - Ajout des normales aux sommets
  - *Affichage du terrain avec une texture (après le CM texture)*



Création des sommets à partir d'une image

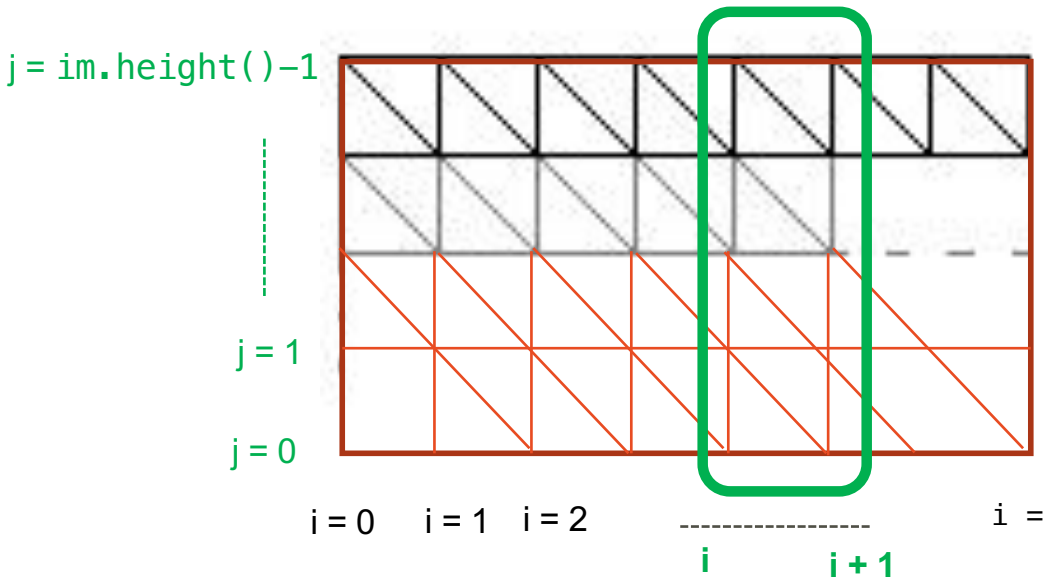


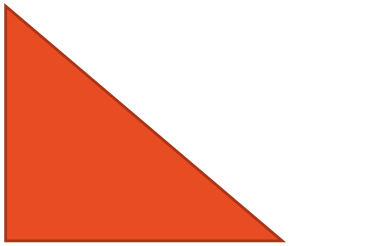
Création de l'ensemble des sommets du Mesh :

→ Boucle sur la dimension en x = i

→ Boucle sur la dimension en z = j

Création des sommets à partir d'une image





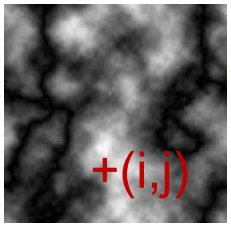
→ Boucle sur la dimension en i

→ Boucle sur la dimension en j

Création de la bande reliant sommets d'abscisses i et i+1
et ordonnées = variation de j

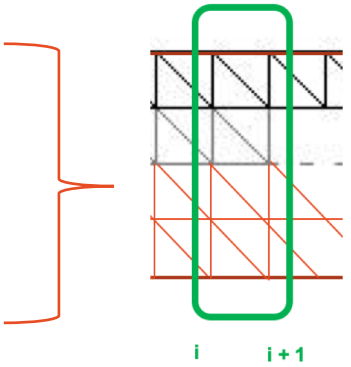
m_terrain.vertex (i+1, valeur de l'image en (i+1, j), j)
m_terrain.vertex (i, valeur de l'image en (i, j), j)

Création du Mesh du terrain




```
Mesh m_terrain; // Déclaré dans ViewerEtudiant

void ViewerEtudiant::init_terrain(Mesh& m_terrain, const Image& im){

    m_terrain = Mesh(GL_TRIANGLE_STRIP); // Choix primitive OpenGL


    for(int i=1;i<im.width()-2;++i){ // Boucle sur les i

        for(int j=1;j<im.height()-1;++j){ // Boucle sur les j

            m_terrain.normal( terrainNormal(im, i+1, j) );
            m_terrain.vertex( Point(i+1, 25.f*im(i+1, j).r, j)  );

            m_terrain.normal( terrainNormal(im, i, j) );
            m_terrain.vertex( Point(i, 25.f*im(i, j).r, j)  );

        }

        m_terrain.restart_strip(); // Affichage en triangle_strip par bande
    }
}
```

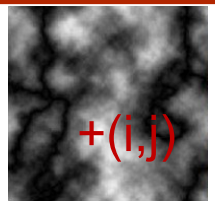
Calcul de la normale en (i+1, j)



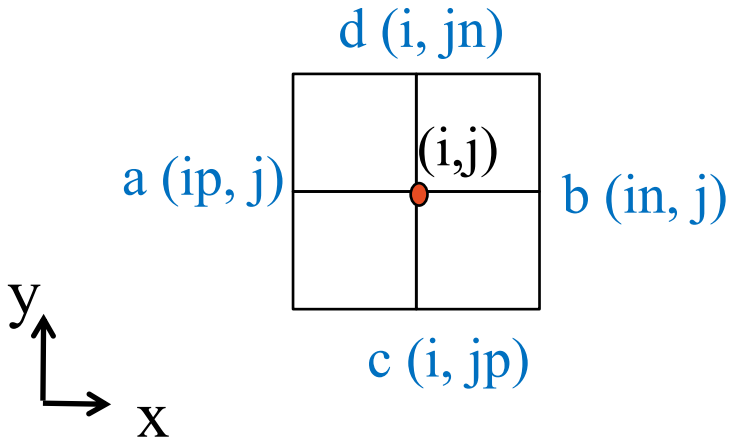
Calcul de la normale en (i, j)

im(i,j) = Couleur du pixel (i,j) de l'image
im(i,j).r = composante rouge de la Color

Calcul des normales




```
Vector terrainNormal(const Image& im, const int i, const int j){

// Calcul de la normale au point (i,j) de l'image
int ip = i-1;
int in = i+1;
int jp = j-1;
int jn = j+1;

Vector a( ip, im(ip, j).r,  j  );
Vector b( in, im(in, j).r,  j  );
Vector c( i,  im(i, jp).r,  jp );
Vector d( i,  im(i, jn).r,  jn );

Vector ab = normalize(b - a);
Vector cd = normalize(d - c);

Vector n = cross(ab,cd);

return n;

}
```



Normale au point (i,j) =

$$\overrightarrow{ab} \wedge \overrightarrow{cd}$$

Création et affichage du Mesh du terrain

```
int ViewerEtudiant::init(){

    // Chargement de l'image servant de carte de hauteur
    m_terrainAlti = read_image("data/terrain/terrain.png");

    // Création du Mesh
    init_terrain(m_terrain, m_terrainAlti);

}

int ViwerEtudiant::render(){

        Transform T = Translation( … ) * Scale( … );

        // Affichage du Mesh en appliquant T
        draw_terrain(T)

}
```

```
void ViewerEtudiant::draw_terrain(const Transform &T){
     gl.model( T );
     gl.draw( m_terrain );
}
```