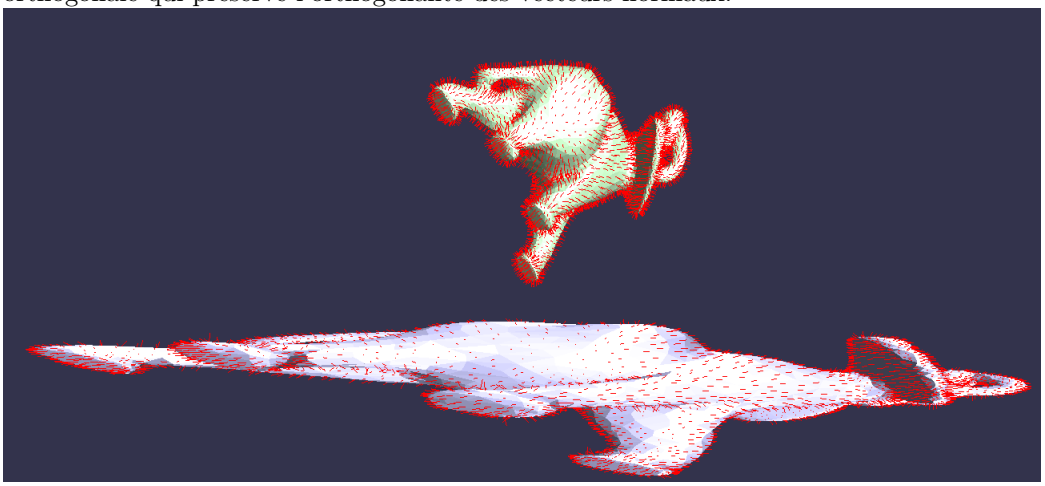
TP2: Algèbre linéaire

Mathieu LADEUIL, Sylvain LECLERC, Abdelillah TAKHTOUKH, Camille BERNADAS

October 24, 2021

1 Transformation

1.1:
Si on applique une transformation non orthogonale, on s'aperçoit que les vecteurs normaux appartenant au champs de vecteurs normaux k_vector ne sont pas orthogonaux, contrairement à une transformation orthogonale qui préserve l'orthogonalité des vecteurs normaux.



1.2:
$\langle M_v, Bn_k \rangle = 0$
$\Rightarrow \; M^t \cdot B = Id$
$\Rightarrow \; B = (M^t)^{-1}$
On a bien $\langle Bn_k, Bn_k \rangle = 1$ car Bn_k est bien dans la même direction que lui-même.

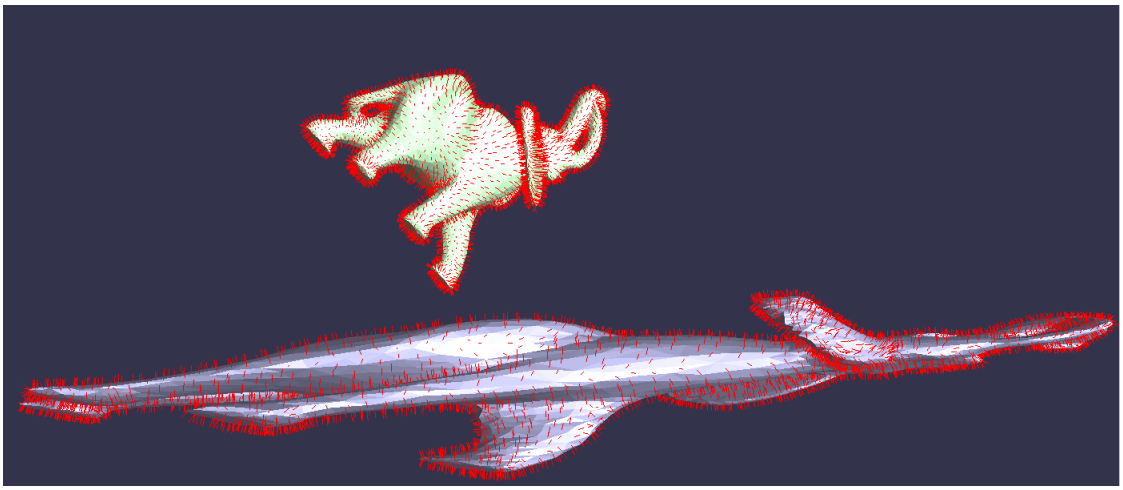
1.3:

```
    m_vector_transformation = Mat3::inverse(i_transformation).getTranspose();
```

1.4:

```
    result = apply_to_vector(i_vector);
    result.normalize();
    return result;
```

Les vecteurs normaux sont bien unitaires.



2 Analyse en Composantes Principales

2.1:
Pour la fonction compute_principal_axis :
On commence par calculer le centroïde **p** des points du maillage, en associant à chaque coordonnée de **p** la moyenne des coordonnées des points du maillage. Ensuite, on cherche à calculer la matrice de covariance **C**. On fait la somme des tenseurs pour calculer la somme des variances. On multiplie C par l'inverse du nombre de points.

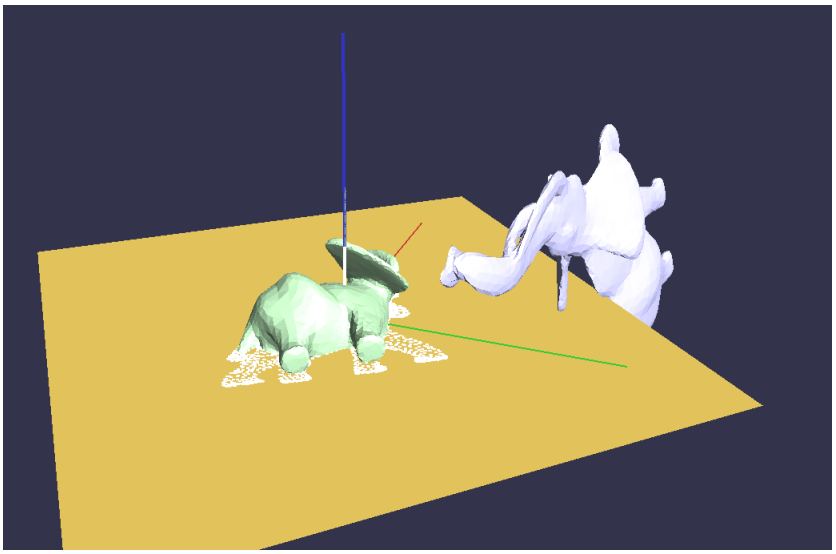
2.2:

```
    gsl_eigen_symmv(covariance_matrix,gsl_eigenvalues,gsl_eigenvectors,workspace);
    gsl_eigen_symmv_sort(gsl_eigenvalues,gsl_eigenvectors,GSL_EIGEN_SORT_VAL_DESC);
```

2.3:

```
    Vec3 project(Vec3 const &input_point, Plane const &i_plane) {
        Vec3 result = input_point - Vec3::dot(input_point - i_plane.point, i_plane.normal)*i_plane.norma
        return result;
    }
```

```
    Vec3 project(Vec3 const &input_point, Vec3 const &i_origin, Vec3 const &i_axis) {
        Vec3 result = i_origin + (Vec3::dot(input_point-i_origin,i_axis)/Vec3::dot(i_axis,i_axis))*i_ax
    return result;
    }
```

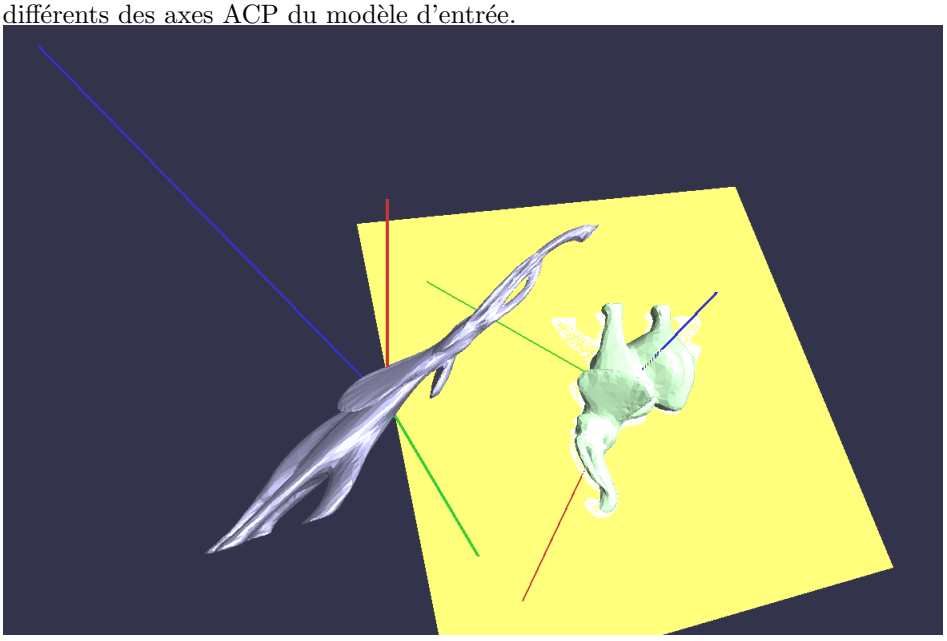


2.4
On veut un facteur de 2 pour le facteur d'échelle pour avoir une ellipsoide de confiance de 2 écart-type. (2-sigma ellipsoid)

2.5

```
    variance[i] = (projection_on_basis[i] - ((projection_on_basis[0] +
    projection_on_basis[1] + projection_on_basis[2])/3)).squareLength() / 3;
```

2.6:
On constate qu'avec une transformation orthogonale, les axes ACP du modèle transformé suivent la transformation. Dans le cas d'une transformation non orthogonale, les axes ACP sont déformés, et donc différents des axes ACP du modèle d'entrée.



3 Décomposition en Valeurs Singulières

3.1:

La fonction find_transformation_SVD parcourt les listes de Vec3 **ps** et **qs** et en fait les sommes dans **p** et textbfq avant de les diviser par le nombre de vecteurs dans chaque liste. On obtient ainsi les centroïdes. Ensuite, on génère la matrice de covariance. On calcule la rotation avec setRotation:

- Calcul de la SVD
- Calcul de la rotation (U*Vt)
- Si le déterminant est négatif, modification de l'orientation

Enfin, on renvoie la rotation et la translation.

3.2:

```
1     gsl_linalg_SV_decomp(u, v, s, workspace);
```

3.3: Les deux modèles se superposent parfaitement.