

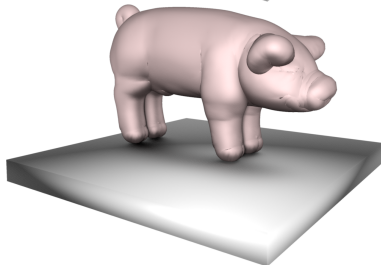
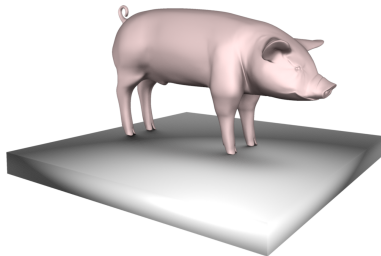
5ETI Synthèse d'images: Déformations de surfaces

CPE Lyon
damien.rohmer@cpe.fr

2012/2013

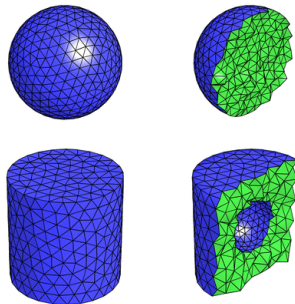
- On cherche à modifier les coordonnées
- Pas la connectivité
⇒ Changement de Topologie
= Complexe

```
OFF
40 95 75
-0.175114 -0.047799 -0.046492
-0.199566 0.738914 -0.064795
-0.010689 0.674496 0.008908
-0.015538 0.153071 0.107408
-0.070148 0.767894 -0.116107
-0.053836 -0.782815 0.109714
-0.162416 -0.785481 0.088014
-0.112365 -0.782492 0.135482
-0.240928 0.031451 0.031966
-0.259289 0.208557 0.035420
0.296891 -0.707385 0.143375
-0.190129 -0.069002 0.109358
-0.010148 0.024179 -0.067283
-0.112968 -0.089127 0.092391
-0.185028 0.377372 -0.111155
3 20 4 1
3 34 11 13
3 12 30 0
3 30 13 17
3 23 22 21
3 29 38 17
3 32 0 13
3 14 0 37
3 24 4 21
3 14 32 1
3 24 2 22
3 3 12 25
3 4 24 15
3 21 15 26
3 35 34 13
3 19 32 13
3 19 13 27
```



Déformations : Methodes physiques ?

- On exprime les equations de la physique sur des éléments volumiques (il faut les construire)= equations différentielles sur des tetraèdres linéaires.
- On applique les conditions initiales (forces, paramètres) sur le maillage volumique
- On résout les equations par méthode numérique = Inversion de larges matrices creuses.
- On attend ...
- On réitère sur les paramètres



Déformations : Methode non physique

- ⊕ On déplace comme on souhaite ce que l'on souhaite.
- ⊕ Orienté Résultat.
- ⊕ On ne traite que ce qui est visualisé.
- ⊖ On introduit des procédés non réalistes physiquement.
⇒ Contraintes

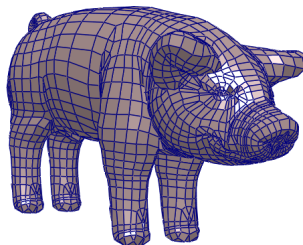
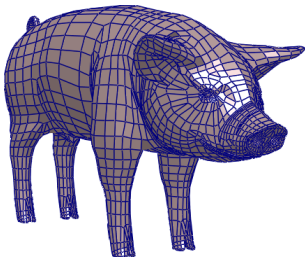


Déformation d'un maillage

- Animation - Déformation de surface
= trouver f tel que

$$(x'_0, x'_1, x'_2, \dots, x'_n) = f(x_0, x_1, x_2, \dots, x_n)$$

On affiche le maillage formé par $(x'_0, \dots, x'_n)$ avec l'ancienne connectivité.



- Comment choisir f ?

Fonction de Deformations

- On maîtrise assez peu de transformations f

$$f : \mathbf{x} \in \mathbb{R}^3 \mapsto \mathbf{x}' \in \mathbb{R}^3$$

- On prend f linéaire : $f = M$.

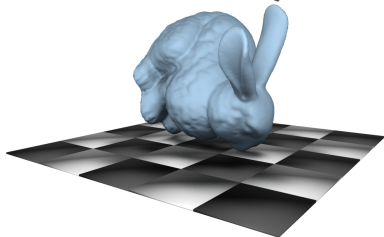
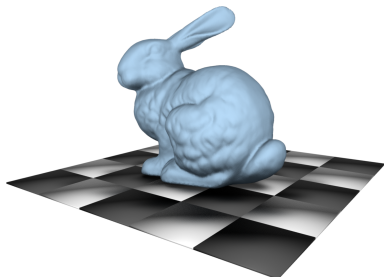
- Isométries

- Translation
- Symétries
- Rotations

$$\mathbf{x}' = M\mathbf{x} \quad \text{et} \quad |\det(M)| = 1$$

- Les homothéties

$$\mathbf{x}' = \text{diag}(s_1, s_2, s_3)\mathbf{x}$$



Important !

Rotation autour d'un axe n par un angle ϕ !

$$R(\mathbf{n}, \phi) = \begin{pmatrix} \cos(\phi) + n_x^2(1 - \cos(\phi)) & n_x n_y(1 - \cos(\phi)) - n_z \sin(\phi) & n_y \sin(\phi) + n_x n_z(1 - \cos(\phi)) \\ n_z \sin(\phi) + n_x n_y(1 - \cos(\phi)) & \cos(\phi) + n_y^2(1 - \cos(\phi)) & -n_x \sin(\phi) + n_y n_z(1 - \cos(\phi)) \\ -n_x \sin(\phi) + n_x n_z(1 - \cos(\phi)) & n_x \sin(\phi) + n_y n_z(1 - \cos(\phi)) & \cos(\phi) + n_z^2(1 - \cos(\phi)) \end{pmatrix}$$

- Rotation Globale

$$\forall i, \mathbf{x}'_i = R \mathbf{x}_i$$

- Revient au même qu'une expression par quaternion.
(interpolations)

Rotation

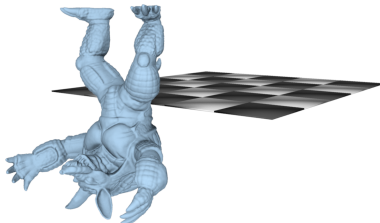
Ne pas oublier de centrer la rotation

$$\mathbf{x}' = R(\mathbf{x} - \mathbf{x}_0) + \mathbf{x}_0$$

Ou sous forme matricielle

$$\mathbf{x}' = T R T^{-1} \mathbf{x}$$

T est une simple translation, ou un changement de base locale !
(Inversion de matrice 3×3 est connu explicitement)



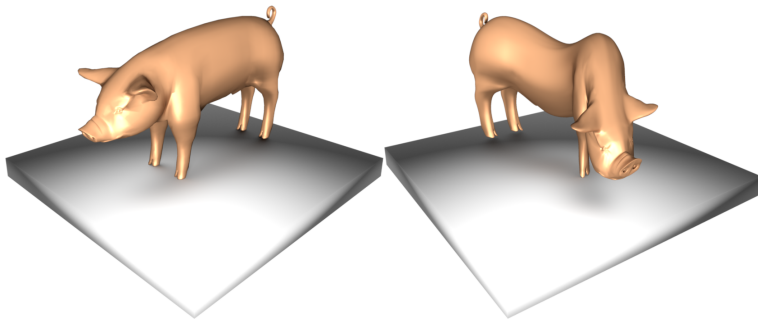
Transformations rigides

Utilisation supplémentaire :

- On peut faire varier les paramètres de la transformation dans l'espace !

$$f : \mathbf{x} \in \mathbb{R}^3 \mapsto \mathbf{x}' = M(\mathbf{x}) \mathbf{x}$$

Ex. Translation dépendant de la position :



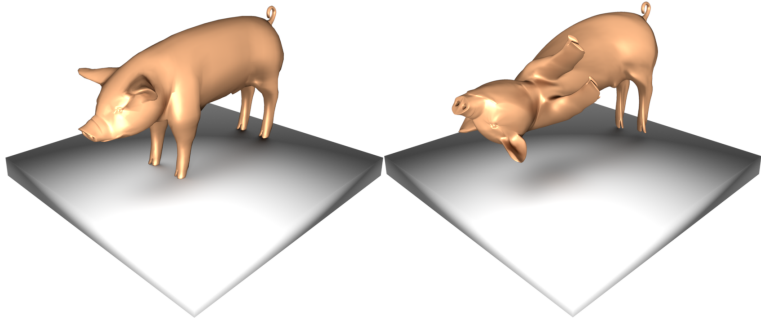
- Angle de rotation qui varie

$$\forall i, \mathbf{x}_i' = R\left(\mathbf{n}, \frac{z}{z_{max}}\right) \mathbf{x}_i$$

```
for (k=0; k<N; k++)  
{  
    Vec x = mesh.get_vertex(k);  
    double angle = x[0] * PI * time;  
    Matrix R = Matrix::rotation(Vec(1,0,0), angle);  
    mesh.set_vertex(k, R*x);  
}
```

Transformations rigides : Exemple

Exemple de Rotation d'angle variable



- Interpolation de matrices pas ok

$$M = \alpha R_1 + (1 - \alpha) R_2 \notin SO(3)$$

- Problème : Interpolation dans espace non Cartésien (Sphère unité = variété).
- ⇒ Généralisation de l'interpolation linéaire dans un **groupe de Lie** :

$$R = \left(R_2 R_1^{-1} \right)^\alpha R_1$$

- Exponentielle de matrices

- Quaternions : Rotation axe n , angle θ .

$$q = (n_x \sin(\theta), n_y \sin(\theta), n_z \sin(\theta), \cos(\theta)) \in \mathbb{R}^4$$

- ⇒ Représentation concise rotation (quaternion unitaires).
- ⇒ Généralisation nombres complexes : $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$.
- Produit de rotation : $q_1 q_2 \Leftrightarrow R_1 R_2$ (non commutatif)

Algebre

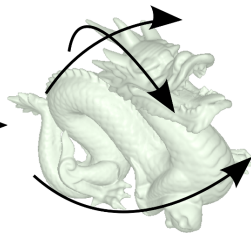
$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$



Lie
Group
 $SO(3)$



rotations



- Quaternion $q = (q_0, q_1, q_2, q_3) \Rightarrow$ Matrice R :

$$R = \begin{pmatrix} 1 - 2(q_1^2 + q_2^2) & 2(q_0 q_1 - q_3 q_2) & 2(q_0 q_2 + q_3 q_1) \\ 2(q_0 q_1 + q_3 q_2) & 1 - 2(q_0^2 + q_2^2) & 2(q_1 q_2 - q_3 q_0) \\ 2(q_0 q_2 - q_3 q_1) & 2(q_1 q_2 + q_3 q_0) & 1 - 2(q_0^2 + q_1^2) \end{pmatrix}$$

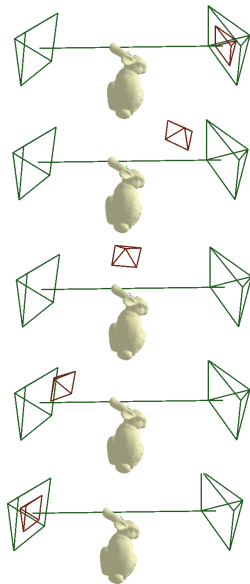
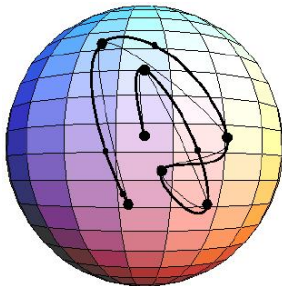
- Représentation concise multiplications : $q = (\mathbf{v}, w)$

$$q_1 q_2 = (w_0 \mathbf{v}_1 + w_1 \mathbf{v}_0 - \mathbf{v}_0 \times \mathbf{v}_1, w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1)$$

Interpolation de rotations

- Interpolation sans exponentielle de matrices : (SLERP) *Spherical Linear intERPolation*

$$\left\{ \begin{array}{l} SLERP(q_0, q_1, \alpha) = \\ \frac{\sin(\alpha(1-\theta))}{\sin(\theta)} q_0 + \frac{\sin(\alpha\theta)}{\sin(\theta)} q_1 \Leftrightarrow (R_1 R_0)^\alpha R_1 \\ \cos(\theta) = q_0 \cdot q_1 \end{array} \right.$$



- ⊖ Transformations non locales.
- ⊖ Axe et angle variable des rotations : complexe à manipuler
- ⇒ On utilise des transformations constantes
- ⇒ On les rend locales artificiellement morceaux par morceaux

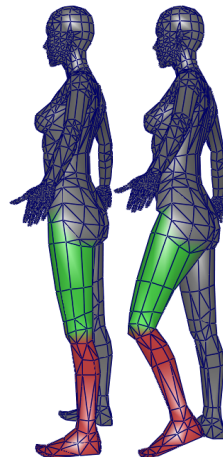
$$\forall i, \mathbf{x}'_i = M(\mathcal{E}(i)) \mathbf{x}$$

Et $M(\mathcal{E}(i)) = I$ pour les ensembles non déformés.

$\mathcal{E}(i)$ = segmentation du maillage.

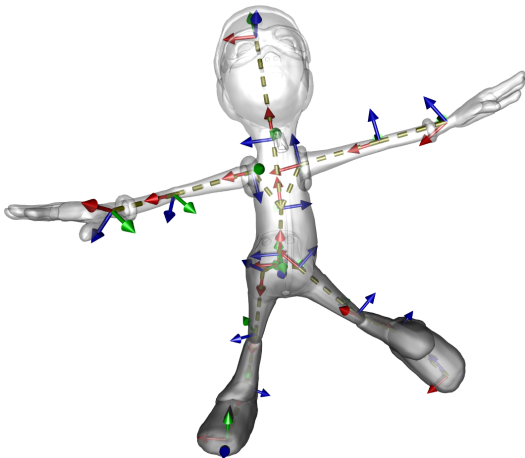
Transformation locales

- On définit des régions $\mathcal{E}$
 - On applique une transformation différente sur chaque partie $M(\mathcal{E})$
 - On utilise principalement des rotations. $R_{\mathcal{E}}$
- ⇒ Comment définir les paramètres de la transformation ?



Squelette d'animation

- Objets sont articulés autour d'un squelette d'animation.
- Squelette = Ensemble de repères hiérarchiques.
- Orientation par rotations successives.



Squelette d'animation

- Transformations hiérarchiques.

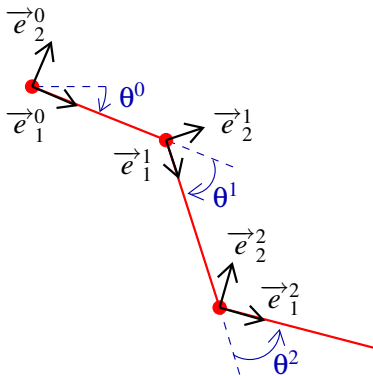
$$T_0 = R_0$$

$$T_1 = T_0 M_1 R_1 M_1^{-1} = R_0 M_1 R_1 M_1^{-1}$$

$$T_2 = T_1 M_2 R_2 T_2^{-1} = \dots$$

⋮

$$T_i = \prod_{k=0}^i M_k R_k M_k^{-1}$$

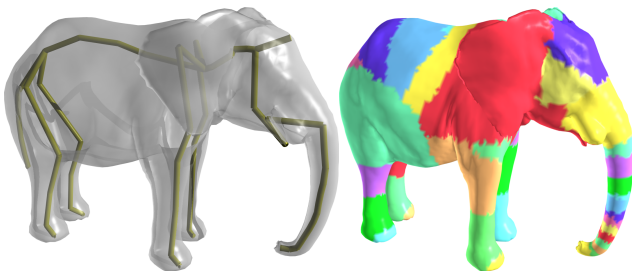


Skinning Rigide

- On lie un morceau local de la surface à un repère du squelette.
- Transformation du repère appliquée sur tous les points du morceau.

$$\mathbf{x}_i(t) = T_{\mathcal{E}(i)}(t) T_{\mathcal{E}(i)}^{-1}(0) \mathbf{x}_i(0)$$

$T(0)$ = Bind Pose.



```
class Frame
{
    Matrix R;
    Matrix Bind;
    Joint *father;
    std::list <Joint*> son;
}
```

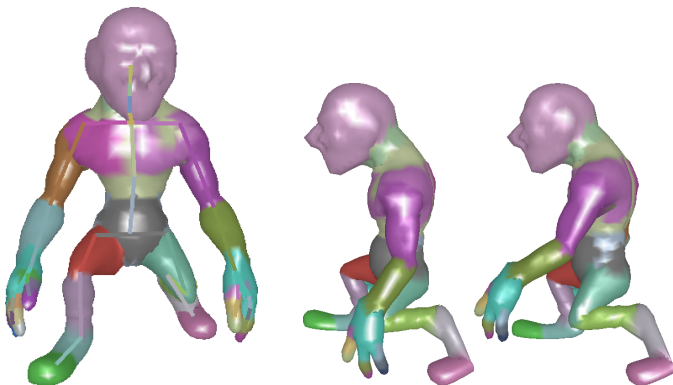
```
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)
{
    int bone_dependency =
        skinning.bone_dependency(k_vertex);
    Matrix T = skeleton.get_matrix(bone_dependency);
    Matrix B =
        (skeleton.get_bind_pose(bone_dependency)).invert();
    deformed_mesh(k_vertex) = T*B * mesh(k_vertex);
}
```

Skinning Rigide : Exemple

■ Action Hierarchique du Squelette

$$\mathbf{x}(t) = \mathbf{M} \mathbf{x}(0)$$

Ex. Une rotation $R_3 = R((1, 0, 0), 45)$



Le squelette est

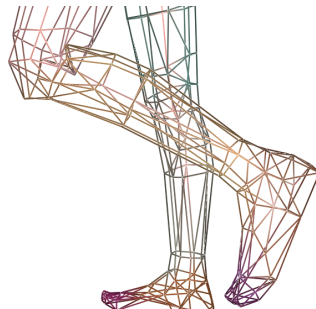
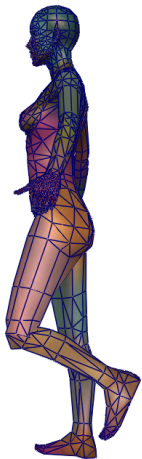
- ⊕ Facile à construire
- ⊕ Facile à animer
- ⊕ Intuitif pour animer un personnage.



Inconvénients Skinning

Mais :

⊖ Discontinuités



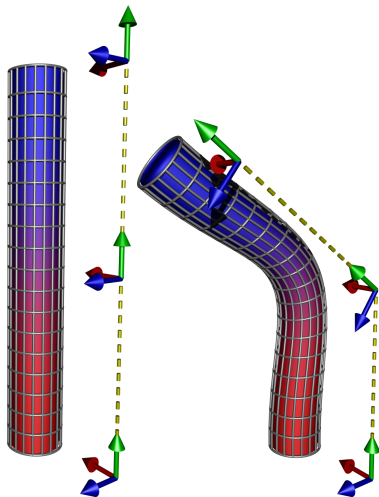
Interpolation des Transformations

Idee : Il faudrait interpoler les transformations entre les repères.

- Le plus simple :

$$T = \omega T_0 + (1 - \omega) T_1$$

- Comment définir les ω ?

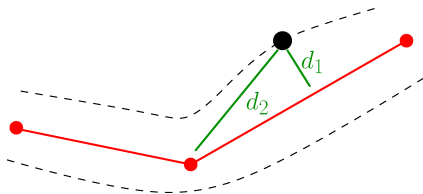
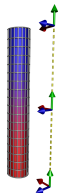


Poids de Skinning

Méthode Possible :

- Fonction de la distance au squelette puis on normalise.

$$\alpha_1 = \frac{1}{d_1}$$
$$\alpha_2 = \frac{1}{d_2}$$
$$\omega_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2}$$
$$\omega_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2}$$



Distance cylindrique

Distance d'un point $\mathbf{x}$ à un segment $[AB]$.

$$\mathbf{u}_1 = \mathbf{x} - \mathbf{x}_A, \quad \mathbf{u}_2 = \mathbf{x}_B - \mathbf{x}_A$$

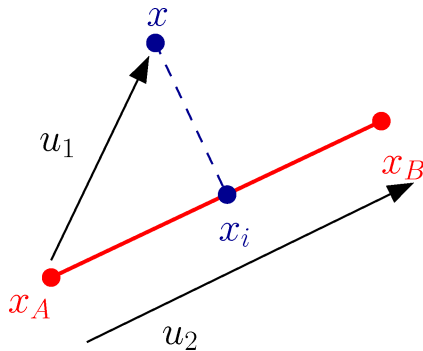
$$p = \frac{\mathbf{u}_1 \cdot \mathbf{u}_2}{\|\mathbf{u}_2\|^2}$$

Si $p < 0$ $\mathbf{x}_i = \mathbf{x}_A$

Si $p > 1$ $\mathbf{x}_i = \mathbf{x}_B$

Sinon $\mathbf{x}_i = \mathbf{x}_A + p\mathbf{u}_2$.

$$d = \|\mathbf{x} - \mathbf{x}_i\|$$

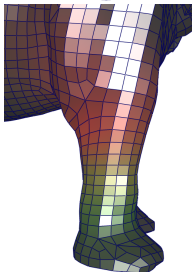
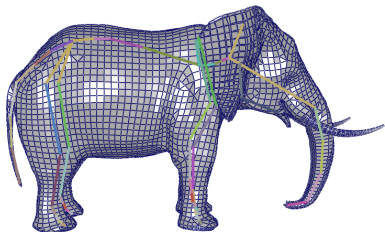


- Pour tout sommet i , on associe les couples (os,poids) :
 $(b_k, \omega_k)_{k \in \mathcal{E}(i)}$.

```
std::vector <std::vector <int>> bone_dependencies;  
std::vector <std::vector <double>> skinning_weights;  
  
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)  
{  
    N_dep = bone_dependencies[k_vertex].size();  
    for(k_dep=0;k_dep<N_dep;k_dep++)  
    {  
        bone = bone_dependencies[k_vertex][k_dep];  
        weight = skinning_weight[k_vertex][k_dep];  
    }  
}
```

```
//skinning weights
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)
{
    for(k_bone=0;k_bone<N_bone;k_bone++)
    {
        alpha = push_back(1/distance(k_vertex,k_bone));
        if(alpha>epsilon)
        {
            skinning_weights[k_vertex].push_back(alpha);
            bone_dependencies[k_vertex].push_back(k_bone);
        }
    }
    norm_skinning_weights();
}
```

Exemples

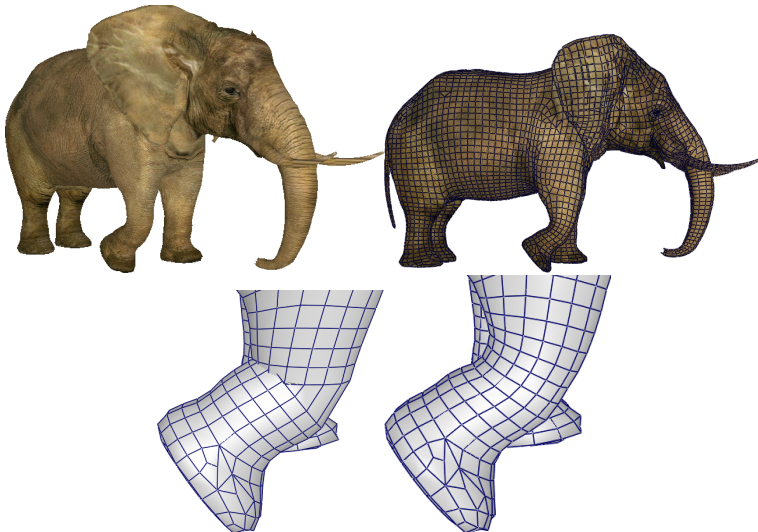


$$\mathbf{x} = \left(\sum_i \omega_i \mathbf{M}_i \right) \mathbf{x}_0$$

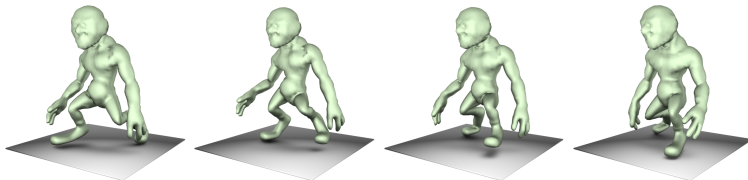


```
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)
{ int N_dep = bone_dependencies[k_vertex].size();
  Matrix D;
  for(k_dep=0;k_dep<N_dep;k_dep++)
  { int k_bone = bone_dependency[k_vertex][k_dep];
    double weight = skinning_weights[k_vertex][k_dep];
    D += weight*skeleton.matrix(k_bone); }
  deformed_mesh(k_vertex) = D * mesh(k_vertex);
}
```

Exemples



- Animation = Les angles des rotations de M dépendent du temps.



```
+ <animation id="ele_R_shoulder_rotateY"></animation>
+ <animation id="ele_R_shoulder_rotateZ"></animation>
+ <animation id="ele_R_elbow_rotateX"></animation>
- <animation id="ele_R_elbow_rotateY">
  - <source id="ele_R_elbow_rotateY_ele_R_elbow_rotateY_ANGLE-input">
    - <float_array id="ele_R_elbow_rotateY_ele_R_elbow_rotateY_ANGLE-input-array" count="134">
      0.040000 0.080000 0.120000 0.160000 0.200000 0.240000 0.280000 0.320000 0.360000 0.400000 0.440000 0.480000 0.520000
      0.560000 0.600000 0.640000 0.680000 0.720000 0.760000 0.800000 0.840000 0.880000 0.920000 0.960000 1.000000 1.040000
      1.080000 1.120000
    </float_array>
  </source>
  - <source id="ele_R_elbow_rotateY_ele_R_elbow_rotateY_ANGLE-output">
    - <float_array id="ele_R_elbow_rotateY_ele_R_elbow_rotateY_ANGLE-output-array" count="134">
      0 -0.072670 -0.223444 -0.389830 -0.549435 -0.692583 -0.812962 -0.905247 -0.964565 -0.985765 -0.985721 -0.985668 -0.985607
      -0.985540 -0.961561 -0.887361 -0.755854 -0.560878 -0.285142 -0.201935 -0.173549 -0.175789 -0.182313 -0.192826 -0.207032
      -0.224636
    </float_array>
```

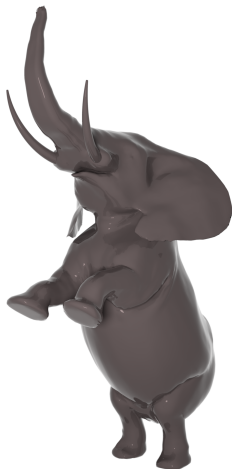
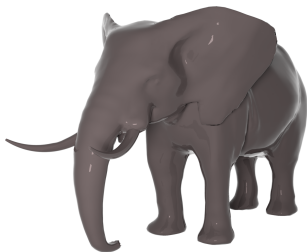
Skinning Lisse : Conclusion

Avantages

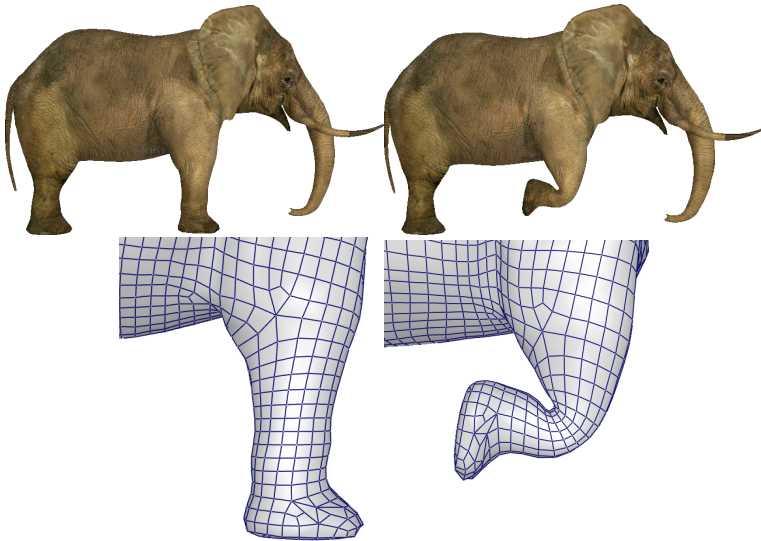
- ⊕ Calcul Rapide.
- ⊕ Squelette intuitif à paramétrer.

inconvénients

- ⊖ Artefacts pour des angles importants.

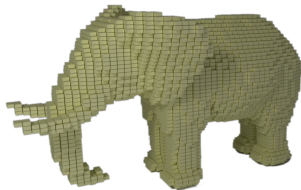
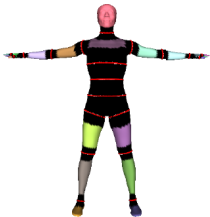


Artefact : Collapsing Elbow



Poids de skinning : Problème de distance cartésienne.

- Méthode 2 : Un artiste peint directement sur le maillage
- Méthode 3 : Plus intelligent (distance curviligne, geodesique, ...)



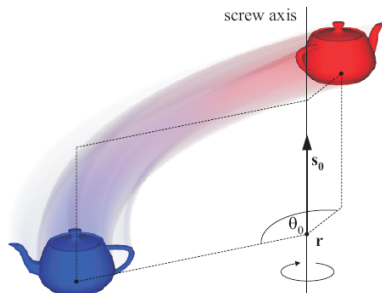
Améliorations Possibles

Interpolation : Interpolation linéaire n'est pas correcte.

$$\mathcal{R}_1 + \mathcal{R}_2 \neq \mathcal{R}.$$

⇒ Comment interpoler une matrice de transformation ?

- **Exponential map** : Problème de temps de calcul.
- **Quaternions** : Ici on a un axe de rotation qui se translate.
- **Dual Quaternions** : OK



Skinning = **S**keleton **S**ubspace **D**eformation

Methode la plus adaptée pour les mouvements articulés autour d'un squelette.

- Déformer un visage = interpolation de formes
- Déformation “molles” = FFD
- Déformations hiérarchiques = Multiresolution
- Déformations libres = Inversion de matrices + contraintes

