

Déformations interactives de maillages

Application à l'animation de personnages

CPE Lyon
damien.rohmer@imag.fr

Decembre 2009

Déformation de maillages

Surface
Déformation

Modélisation
Maillage

Surfaces

Déformation de maillages

1 Surface

- Modélisation
 - Choix de modélisation - Explicite/Implicite
 - Discrétisation - Triangulation
- Maillage
 - Enregistrement - Chargement : Cas du off
 - Structure de données

2 Déformation

- Transformation classiques
 - Objectif de la déformation - Physique / non physique
 - Transformations matricielles
- Skinning
 - Skinning rigide
 - Skinning lisse
 - Limitation et amélioration possibles

Déformation de maillages

Surface
Déformation

Modélisation
Maillage

1 Surface

- Modélisation
 - Choix de modélisation - Explicite/Implicite
 - Discrétisation - Triangulation
- Maillage
 - Enregistrement - Chargement : Cas du off
 - Structure de données

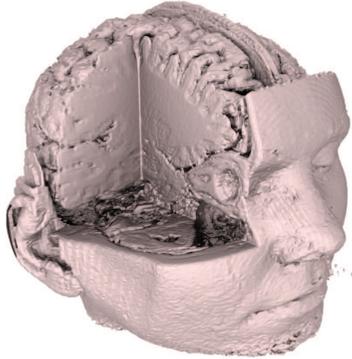
2 Déformation

- Transformation classiques
 - Objectif de la déformation - Physique / non physique
 - Transformations matricielles
- Skinning
 - Skinning rigide
 - Skinning lisse
 - Limitation et amélioration possibles

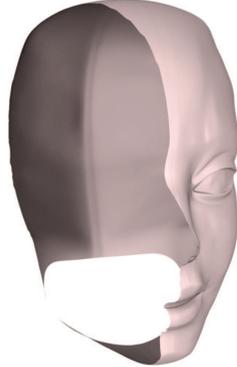
Déformation de maillages

Modélisation d'un objet 3D

■ Modélisation volumique



■ Modélisation surfacique



Discrétisation

Cas des surfaces implicites

On remplit l'espace de voxels à l'intérieur de l'objet.

- ⊕ Adapté aux objets volumiques : applications médicales
- ⊖ Cout mémoire énorme.
- ⊖ Visualisation longue et laide.
- ⊖ Information de voisinage à construire (pas de textures)



Surfaces

2 Façons de définir une surface mathématiquement

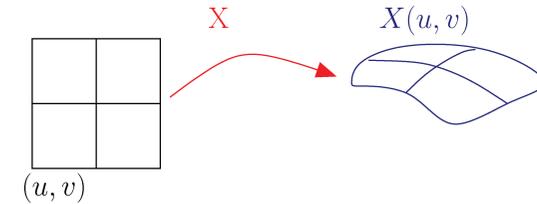
■ Explicitement =

paramétriquement :
 $X : (u, v) \mapsto X(u, v)$ = On définit une transformation.

- ⊕ **Presque** Parfait quand on connaît cette transformation

■ Implicitement : $(x, y, z) \in \mathbb{R}^3$ tels que $F(x, y, z) = a$.

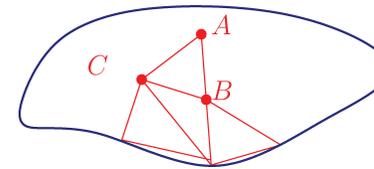
- ⊖ On maîtrise mal la surface sous-jacente



Discrétisation

Cas des surfaces explicites

- On ne définit que la surface d'intérêt $X(u, v)$.
- Ne connaissant pas X globalement, on l'estime localement : morceaux par morceaux.
- Cas le plus simple :
 - On prend X linéaire
 - On définit une surface C^0 morceaux par morceaux.



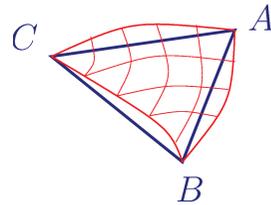
$$X_i(u, v) = u\vec{AB} + v\vec{AC}$$

$$0 \leq u + v \leq 1$$

⇒ Interpolation

Triangulation

- Pour définir un plan :
3 points = triangle.
- Échantillonnage de la surface
 $X(u_i, v_i)$.
- Triangulation = approximation de X
au premier ordre.
 - ⊕ On peut construire n'importe quelle surface.
 - ⊕ Visualisation optimisée pour cartes graphiques.
 - ⊕ Pas d'utilisation mémoire pour le volume.
 - ⊕ Définition surfacique
⇒ Interpolation ⇒ Texture
 - ⊖ Approximation ordre 1.



1 Surface

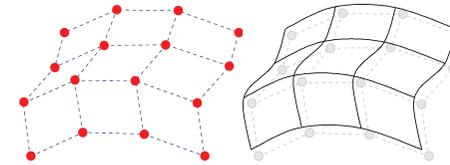
- Modélisation
 - Choix de modélisation - Explicite/Implicite
 - Discrétisation - Triangulation
- Maillage
 - Enregistrement - Chargement : Cas du off
 - Structure de données

2 Déformation

- Transformation classiques
 - Objectif de la déformation - Physique / non physique
 - Transformations matricielles
- Skinning
 - Skinning rigide
 - Skinning lisse
 - Limitation et amélioration possibles

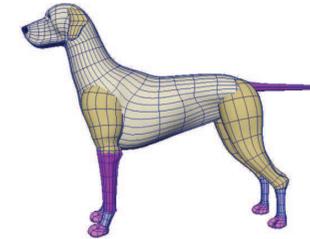
Ordre supérieur

- Ordre supérieur = Meilleure approximation.
- Courbes plus lisses (G^2 : courbure continue).
 - Surfaces Splines - NURBS - ...
 - Patch (4 x 4) ⇒ bi-cubique.



- ⊖ Contraintes trop importantes
pour construire les patches
pour des surfaces
quelconques.

- Plutôt utilisé en CAO



Maillage

Structure de données :

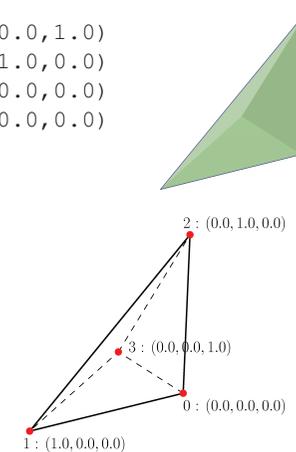
- Ex. Représenter un tétraèdre :

Idée 1 :

(0.0, 0.0, 0.0), (1.0, 0.0, 0.0), (0.0, 0.0, 1.0)
 (0.0, 0.0, 0.0), (0.0, 0.0, 1.0), (0.0, 1.0, 0.0)
 (0.0, 0.0, 0.0), (0.0, 1.0, 0.0), (1.0, 0.0, 0.0)
 (0.0, 1.0, 0.0), (0.0, 0.0, 1.0), (1.0, 0.0, 0.0)

Il y a mieux :

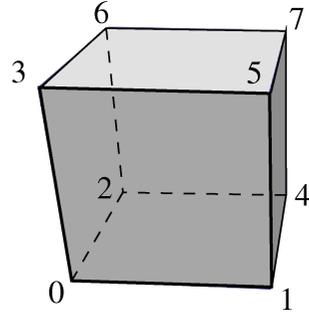
coordonnes:
 (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)
 Connectivite
 (0, 1, 3)
 (0, 3, 2)
 (0, 2, 1)
 (1, 2, 3)



Format off

Exemple de format d'échange. Format off.

```
OFF
8 6 12
0 0 0
1 0 0
0 1 0
0 0 1
1 1 0
1 0 1
0 1 1
1 1 1
4 0 1 4 2
4 1 5 7 4
4 3 6 7 5
4 2 6 3 0
4 2 4 7 6
4 0 3 5 1
```



Structure de données

- Vecteurs contigus dans la mémoire : Affichage rapide en OpenGL.

```
// (x0,y0,z0,x1,y1,z1,...)
std::vector<double> vertex

// (i00,i01,i02,i10,i11,i12,...)
std::vector<int> connectivity

std::vector<double> normal, color, texture ...
```

- Accès à la coordonnée y du sommet k .
`vertex[3*k+1]`
- Accès à la coordonnée y du sommet s (1,2 ou 3) du triangle t .
`vertex[3*connectivity[3*t+s]+1]`

Lecture fichier

```
while (k_vertex < N_vertex)
{
    fscanf(fid, "%f %f %f", X, X+1, X+2);
    add_vertex(X[0], X[1], X[2]);
    k_vertex++;
}
for (k_poly=0; k_poly < N_poly; k_poly++)
{
    fscanf(fid, "%d", &size_poly);
    std::vector v_poly;
    for (k=0; k < size_poly; k++)
    {
        fscanf(fid, "%d", &temp);
        v_poly.push_back(temp);
    }
    add_polygon(v_poly);
}
```

Structure de données : Voisinage

- **1-Voisinage** = Sommets voisins d'un sommet donné

```
std::vector<std::vector<int>> one_ring

// exemple pour le cube:
one_ring[0] = [1,2,3]
one_ring[1] = [5,4,0]
...
```

- Triangles voisins d'un autre
- Triangles voisins d'un point (calcul des normales !)

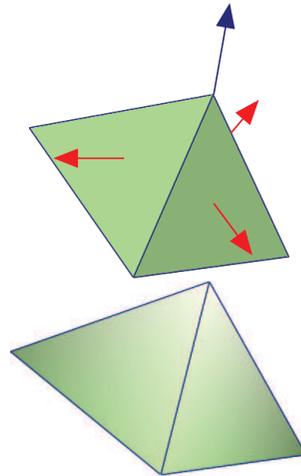
! Attention aux structures de données.
Compromis entre : temps accès / temps recherche / espace mémoire / facilité ...

Normale d'un maillage

- Aspect lisse
⇒ 1 normale par sommet.
- Moyenne de normales (faux mais répandue)

$$\mathbf{n}_k = \frac{\sum_{i \in \mathcal{V}(k)} \mathbf{n}_i}{\left\| \sum_{i \in \mathcal{V}(k)} \mathbf{n}_i \right\|}$$

k : indice sommet
 i : indice face
 $\mathcal{V}(k)$: faces voisines du sommet k



Affichage en OpenGL

Version lente

```
glBegin(GL_TRIANGLES);
for(k_tri=0;k_tri<N_tri;k_tri++)
for(k_vertex=0;k_vertex<3;k_vertex++)
{
for(k_dim=0;k_dim<3;k_dim++)
{
x[k_dim] = vertex[3*connectivity
[3*k_tri+k_vertex]+k_dim];
n[k_dim] = normal[3*connectivity
[3*k_tri+k_vertex]+k_dim];
}
glNormal3d(n[0],n[1],n[2]);
glVertex3d(x[0],x[1],x[2]);
}
glEnd();
```

Affichage en OpenGL

Version Rapide

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_DOUBLE, 0, &vertex[0]);

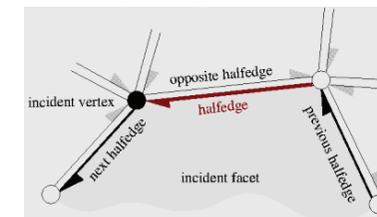
glEnableClientState(GL_NORMAL_ARRAY);
glNormalPointer(GL_DOUBLE, 0, &normal[0]);

glDrawElements(GL_TRIANGLES, 3*N_tri,
GL_UNSIGNED_INT, &connectivity[0]);

glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
```

Structure de données

- Limitation : Vecteur contigus en mémoire
⊖ Ajout/suppression en $O(N)$
- Limitation : Tableau d'index
⊖ Information de voisinage indirecte
- Autres représentation (géométrie)
⇒ Structure en demi-arête (half-edge structure)



- ⊖ Non contigus, codage fastidieux (il existe des librairies)

Structure de données : Résumé

- Choisir avec soin la structure qui convient le mieux pour l'application.
- Tableaux d'indices (contigues) :
 - ⊕ Simple et général
 - ⊕ Si contigue : Accès/Affichage rapide
 - ⊖ Voisinage non direct (choix de structures)
 - ⊖ Ajout/Suppression lent (MAJ voisinage)
- Halfedge Structure (ou autre)
 - ⊕ Ajout/Suppression rapide
 - ⊕ Information de voisinage directement encodée
 - ⊖ Technique
 - ⊖ Accès et affichage moins rapide
 - ⊖ Limité aux variétés

1 Surface

- Modélisation
 - Choix de modélisation - Explicite/Implicite
 - Discrétisation - Triangulation
- Maillage
 - Enregistrement - Chargement : Cas du off
 - Structure de données

2 Déformation

- Transformation classiques
 - Objectif de la déformation - Physique / non physique
 - Transformations matricielles
- Skinning
 - Skinning rigide
 - Skinning lisse
 - Limitation et amélioration possibles

Déformation

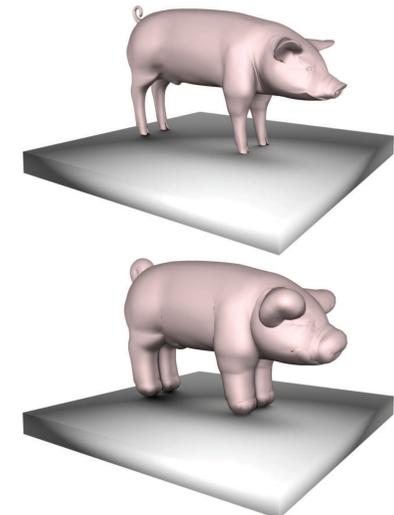
But

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

```

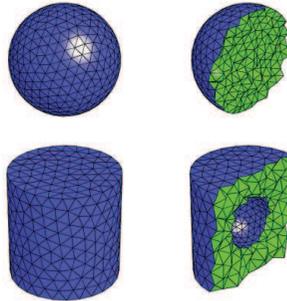
off
40 95 75
-0.175114 -0.847799 -0.046492
-0.199566 0.759914 -0.864795
-0.018603 0.674496 0.080909
-0.015538 0.153071 0.187408
-0.070148 0.767894 -0.116187
-0.053836 -0.700815 0.100714
-0.162416 -0.785481 0.888014
-0.112965 -0.762490 0.135482
-0.249928 0.033451 0.021366
-0.259289 0.286557 0.035420
0.296091 -0.767385 0.143375
-0.120129 -0.860905 0.100508
-0.010148 0.024179 -0.867283
-0.112968 -0.889127 0.892391
-0.185828 0.373372 -0.111555
3 26 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 tétraèdres 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

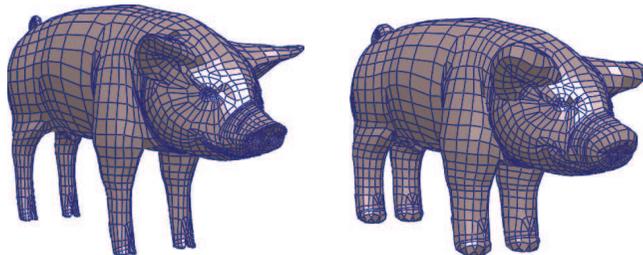


Déformation d'un maillage

- Animation - Deformation 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 : $f = M$.

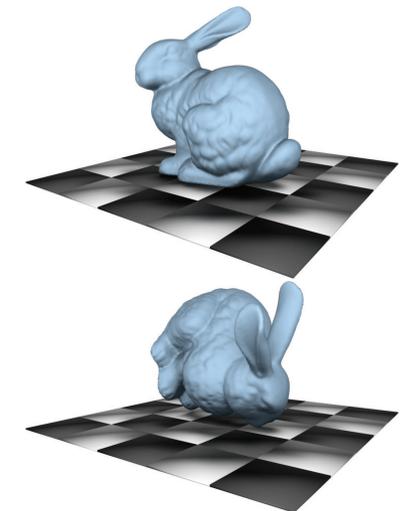
- Isométries

- Translation
- Symétries
- Rotations

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

- Les homothéties

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



Rotation

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)

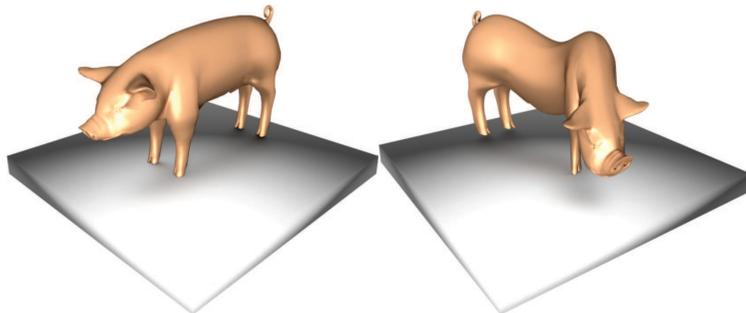
Transformations rigides

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 :



Rotation

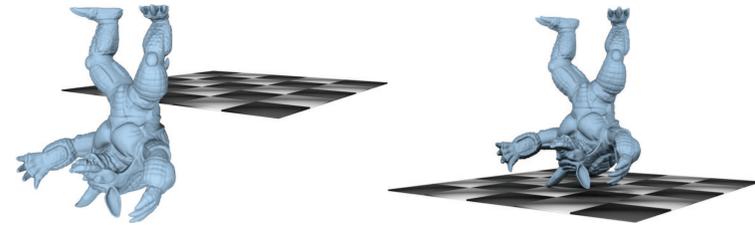
Ne pas oublier de centrer la rotation

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

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 ou projective est connu explicitement)



Transformations rigides

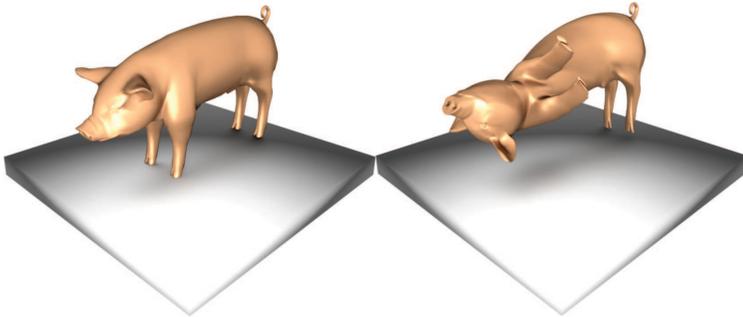
- Angle de rotation qui varie

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

```
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



1 Surface

- Modélisation
 - Choix de modélisation - Explicite/Implicite
 - Discrétisation - Triangulation
- Maillage
 - Enregistrement - Chargement : Cas du off
 - Structure de données

2 Déformation

- Transformation classiques
 - Objectif de la déformation - Physique / non physique
 - Transformations matricielles
- Skinning
 - Skinning rigide
 - Skinning lisse
 - Limitation et amélioration possibles

Transformation rigides

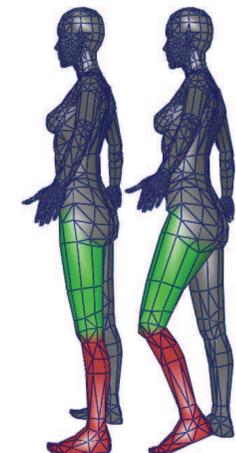
- ⊖ Transformations non locales.
- ⊖ Axe et angle variable des rotations : complexe à manipuler
- ⇒ On utilise des transformations constantes
- ⇒ On les rend locales artificiellement morceaux par morceaux

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

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

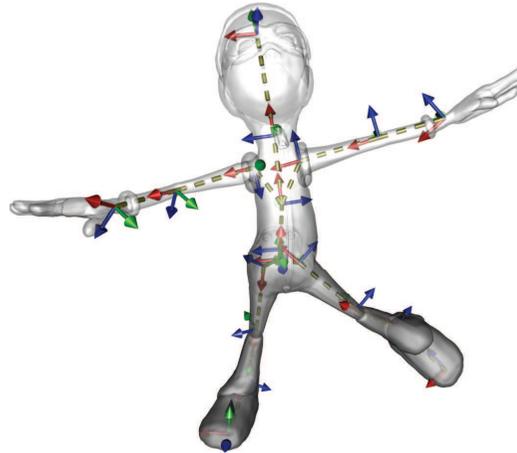
Transformation locales

- 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.

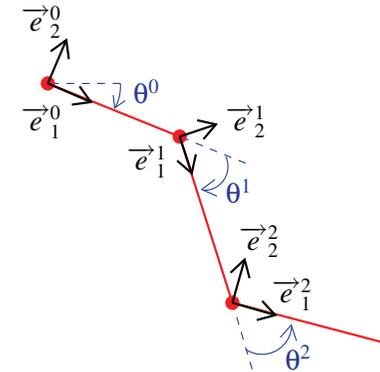
$$T_0 = R_0$$

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

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

$$\vdots$$

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



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.

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

$T(0) = \text{Bind Pose.}$



Squelette

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

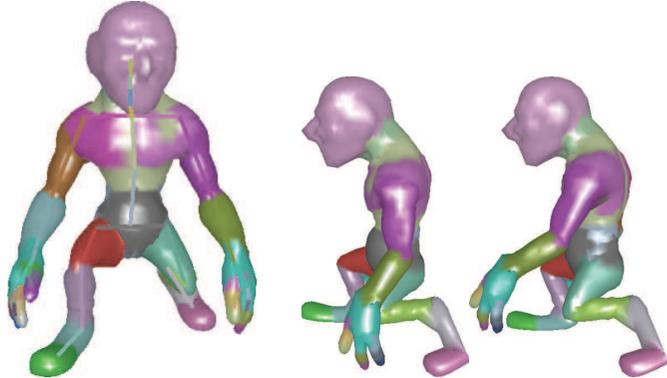
```
for (k_vertex=0; k_vertex<N_vertex; k_vertex++)
{
    int bone_dependency =
        skinning.bone_dependency(k_vertex);
    Matrix T = skeleton.get_matrix(bone_dependency);
    Matrix B =
        (skeleton.get_bind_pose(bone_dependency)).invert();
    deformed_mesh(k_vertex) = T*B * mesh(k_vertex);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

Skinning Rigide : Exemple

- Action Hierarchique du Squelette

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

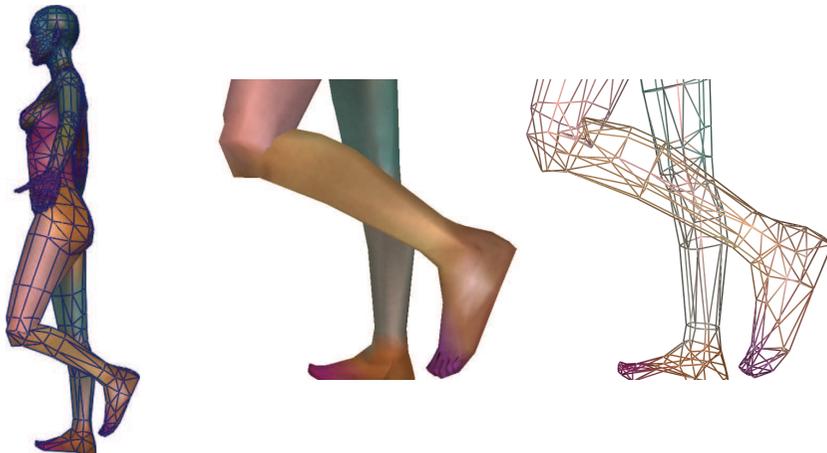
Ex. Une rotation $R_3 = R((1, 0, 0), 45)$



Inconvénients Skinning

Mais :

- ⊖ Discontinuités



Avantage Skinning

Le squelette est

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



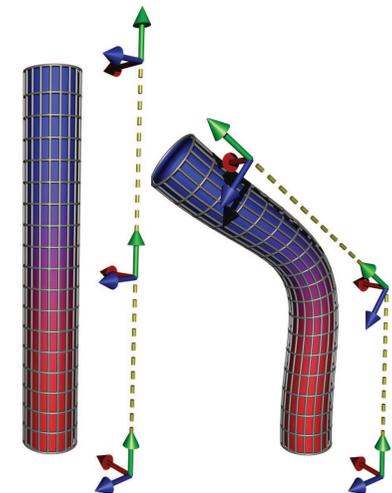
Interpolation des Transformations

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 :

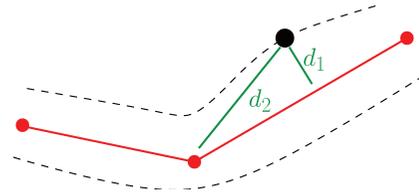
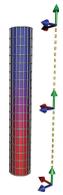
- Fonction de la distance au squelette puis on normalise.

$$\alpha_1 = \frac{1}{d_1}$$

$$\alpha_2 = \frac{1}{d_2}$$

$$\omega_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2}$$

$$\omega_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2}$$



Distance cylindrique

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

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

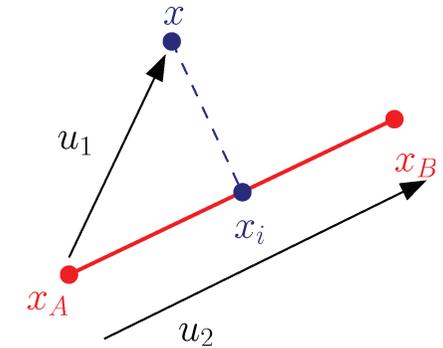
$$p = \frac{\mathbf{u}_1 \cdot \mathbf{u}_2}{\|\mathbf{u}_2\|^2}$$

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

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

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

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



Structure de données : Poids de Skinning

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

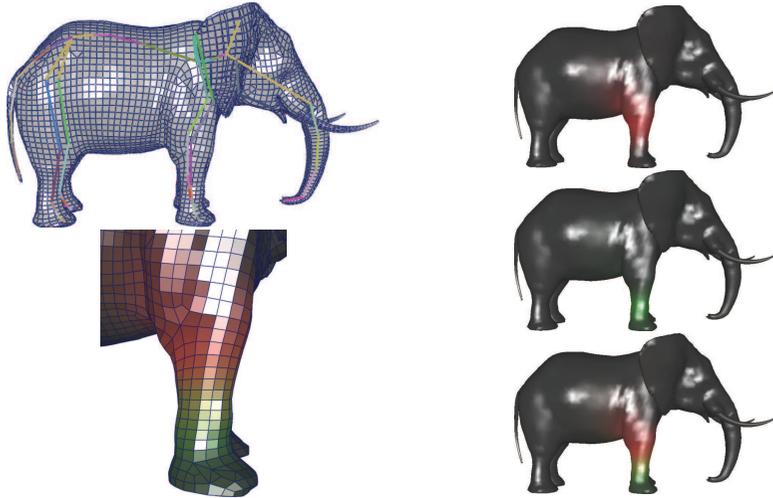
```
std::vector<std::vector<int>> bone_dependencies;
std::vector<std::vector<double>> skinning_weights;

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];
    }
}
```

Calcul des Poids

```
//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



Déformation de maillages

Exemples



Déformation de maillages

Skinning Lisse

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

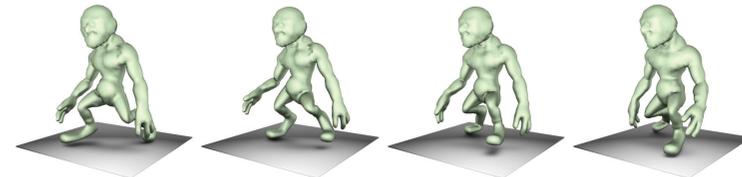


```
for(k_vertex=0;k_vertex<N_vertex;k_vertex++)
{ int N_dep = bone_dependencies[k_vertex].size();
  Matrix D;
  for(k_dep=0;k_dep<N_dep;k_dep++)
  { int k_bone = bone_dependency[k_vertex][k_dep];
    double weight = skinning_weights[k_vertex][k_dep];
    D += weight*skeleton.matrix(k_bone); }
  deformed_mesh(k_vertex)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

Déformation de maillages

Animation

- 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>
```

Déformation de maillages

Skinning Lisse : Conclusion

Avantages

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

Inconvénients

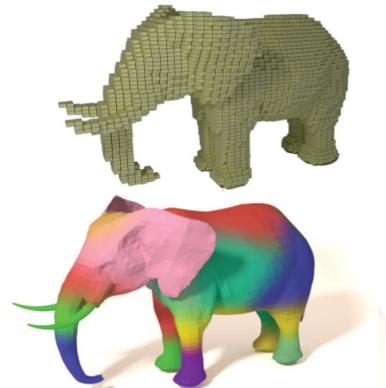
- ⊖ Artefacts pour des angles importants.



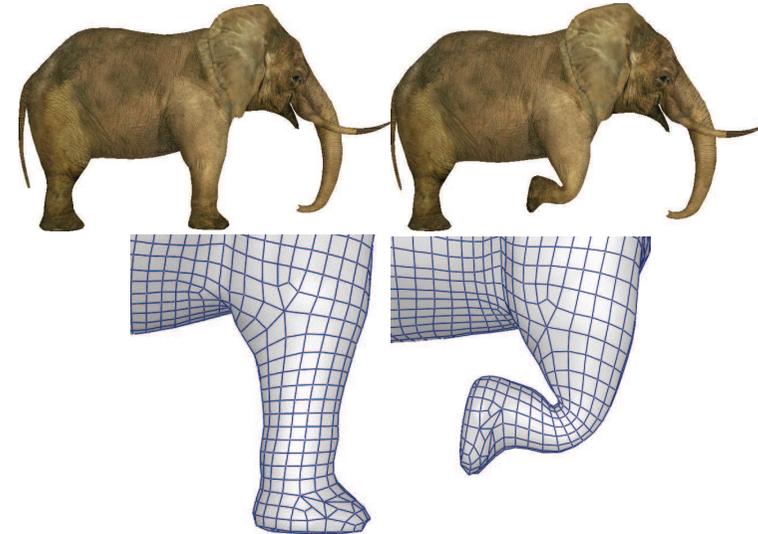
Améliorations Poids de skinning

Poids de skinning : Problème de distance cartésienne.

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



Artefact : Collapsing Elbow



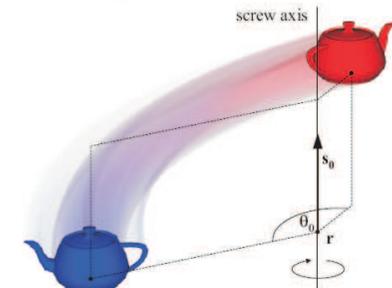
Améliorations Possibles

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

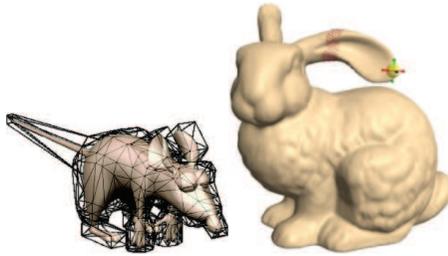


Autres methodes

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 hierarchiques = Multiresolution
- Déformations libres = Inversion de matrices + contraintes

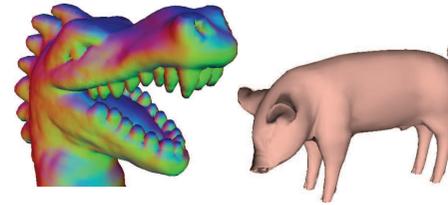


Déformation de maillages

TP

Objectif :

- Manipuler des maillages (structure de données, gestion d'une animation simple, problématiques spécifiques, ...).
- Mise en place d'une déformation interactive (application de déformations locales, problématique de la continuité, ...).



Déformation de maillages