Animation et déformation

Master IMAGINA

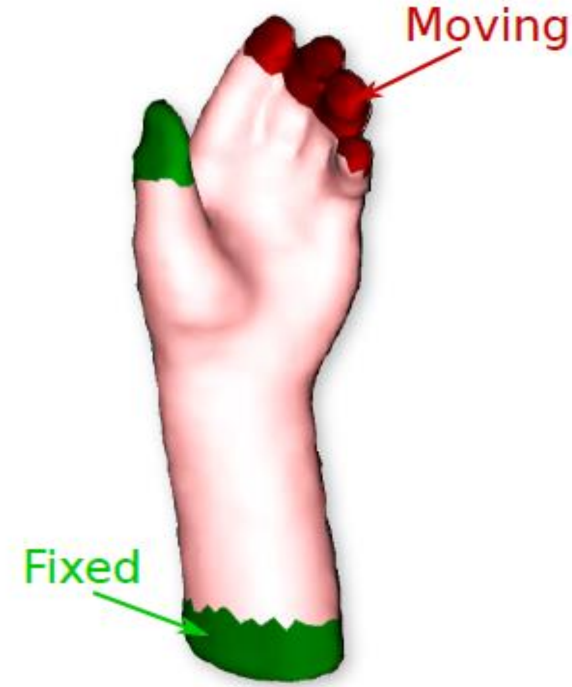
Déformation

- Animation de personnages
  - Sans squelette ou en complément
  - Expressions du visage
  - Changement d’épaisseur

- Modélisation simplifiée : générer de nouvelles formes en déformant une existante

- Poser les personnages pour l’animation

Déformation



- Déformation de surface avec des manipulateurs
  - Linéaires
  - Non-linéaires

- Déformation de l’espace
  - Modèle base resolution guide la deformation de l’espace à l’intérieur de celui-ci
  - Pas de contrainte sur la representation du modèle

Déformation surfacique

- Déformation de surface
  - Surface déformation
  - Forme est vide

- Courbe pour les déformations 2D
- Surface pour les déformations 3D

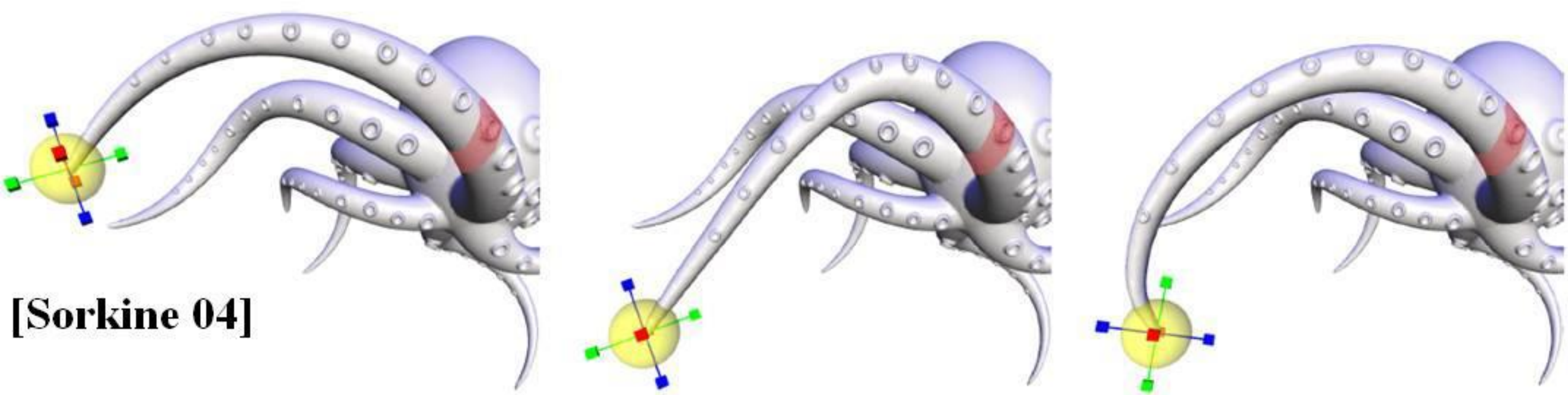
- Déformation :
  - définie seulement **sur** la forme
  - couplée avec la representation de la forme

Principes

- Donner le moins de contraintes/informations possibles et laisser l'algorithme faire le reste
- Ensemble de sommets déformes localement tout en préservant les détails
- Basée sur la géométrie différentielle discrète

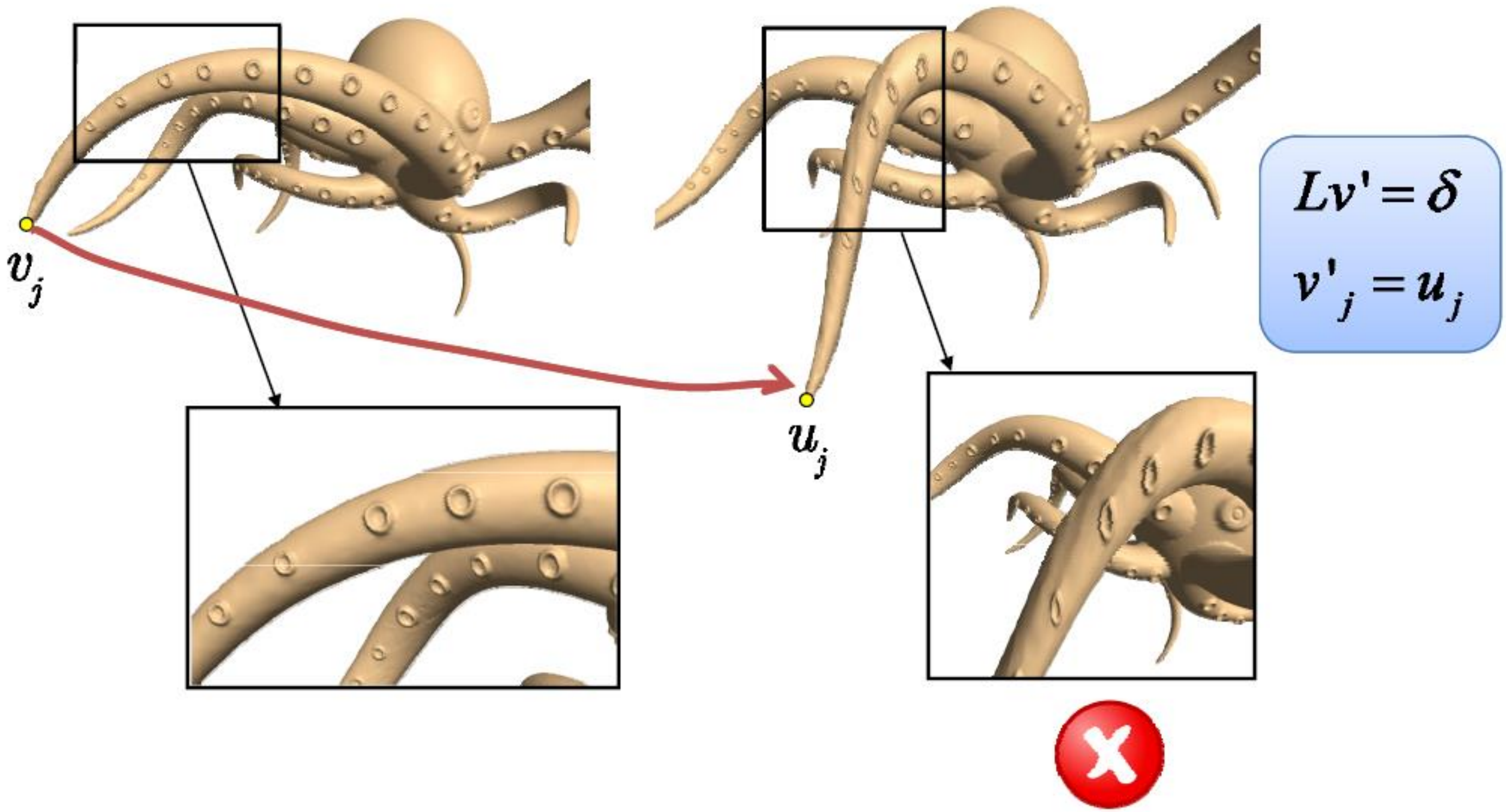
Coordonnées laplaciennes

- Chaque coordonnée de sommet est remplacée par la différence avec ses voisins : $D = LV$ (L : poids)
- Ajout des contraintes $\Leftrightarrow$ ajout de lignes à L et D $\Rightarrow$ L' et D'
- Reconstruction de V par approximation : V' $= \text{argmin}_V(\|L'V - D'\|)$



[Sorkine 04]

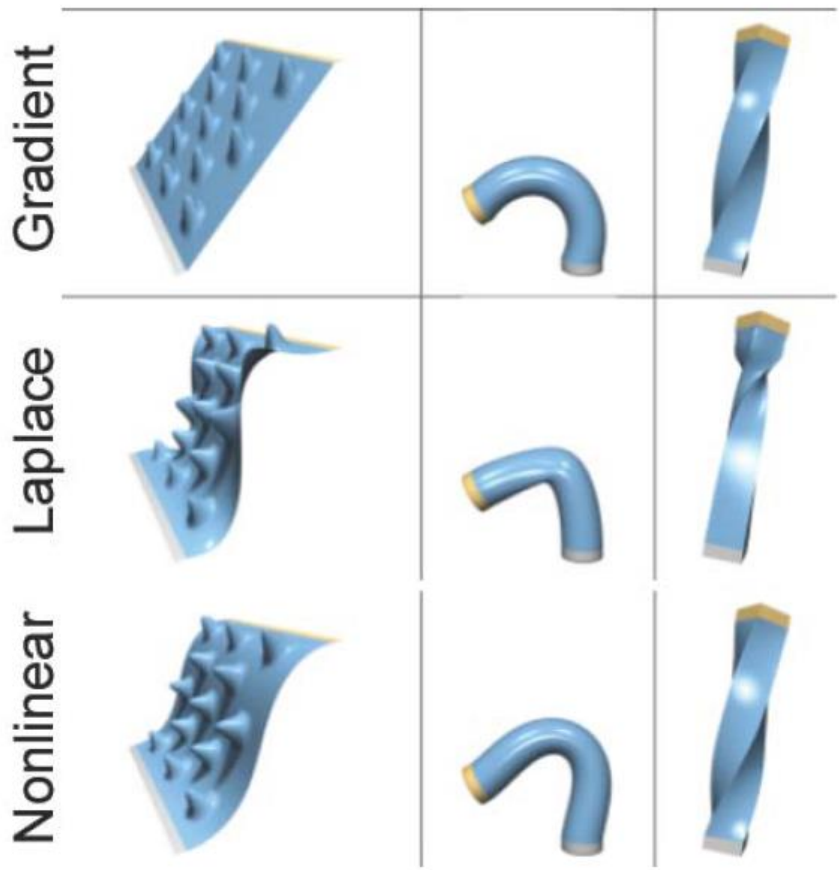
Limitations



Qualité de déformation

- Méthodes non-linéaires fonctionnent pour les grandes rotations et les translations

- Beaucoup plus long à calculer



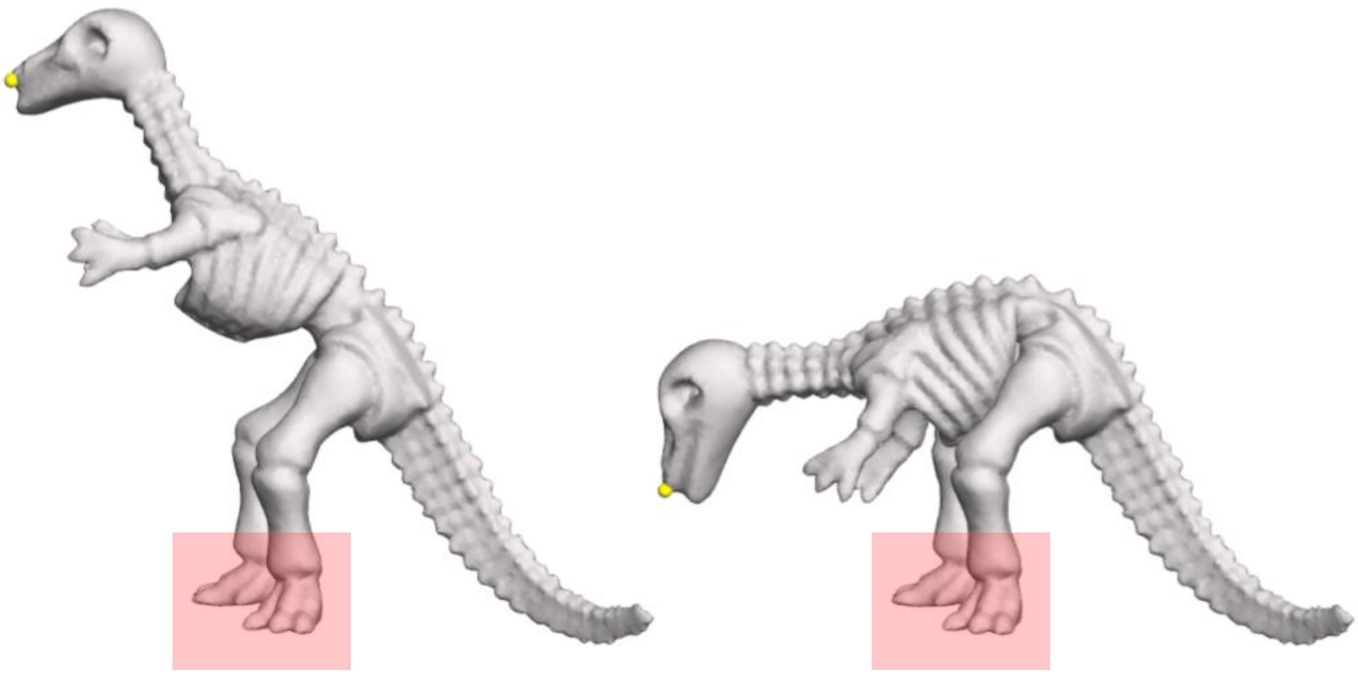
ARAP

- Etat de l’art pour la manipulation directe de formes

[Sorkine & Alexa] : As-Rigid-As-Possible Surface Modeling

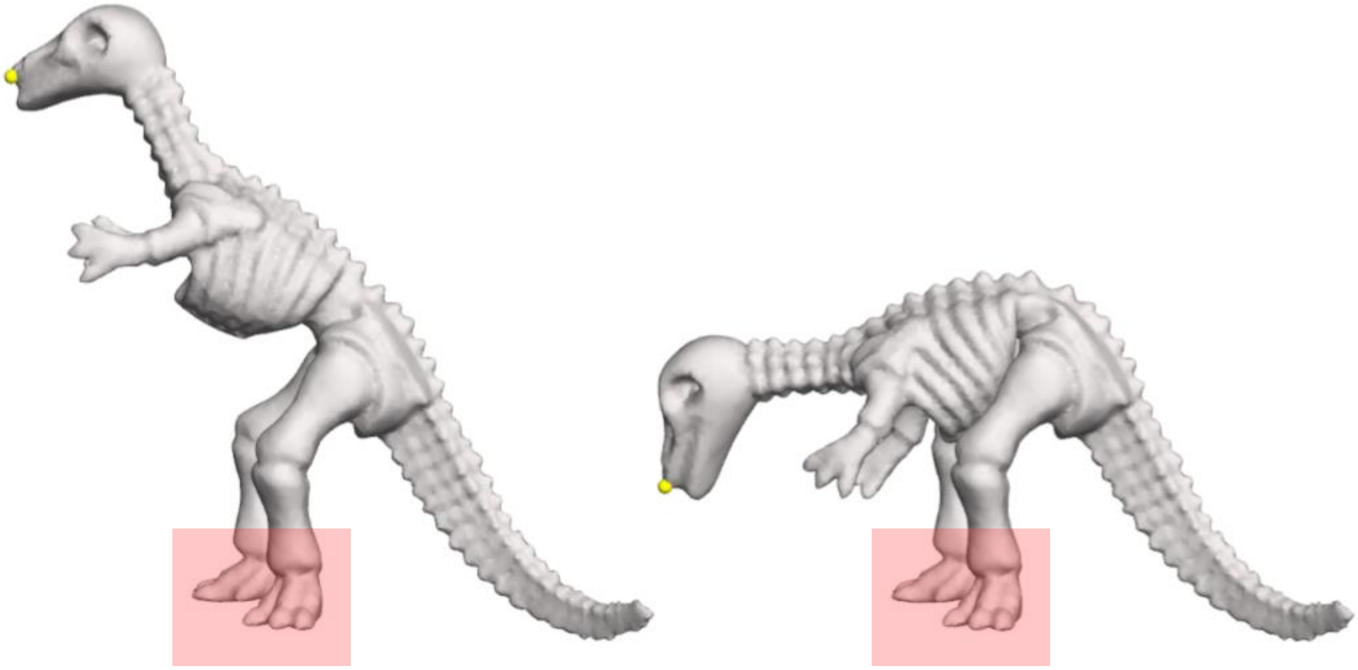
Principe

- Quelques sommets sont placés par l'utilisateur
- Les autres seront déplacé « naturellement »



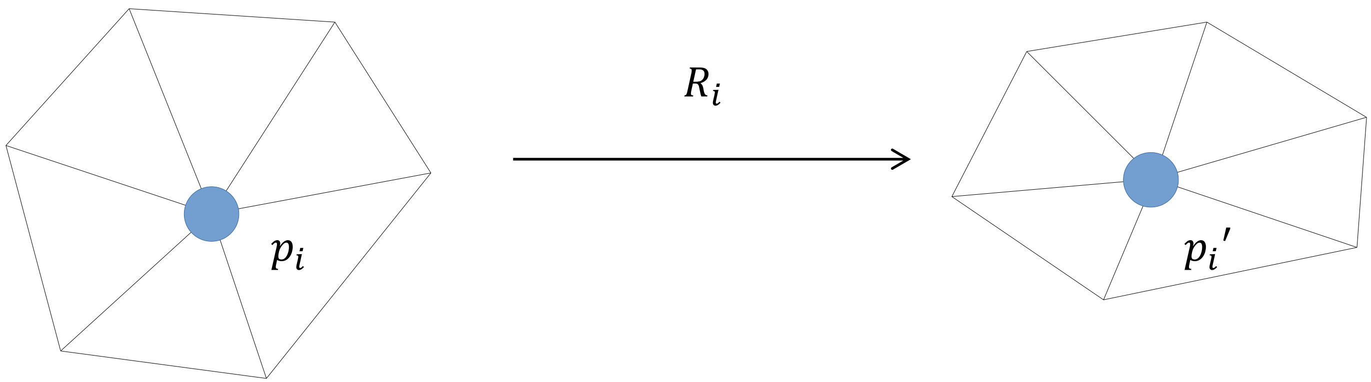
Minimiser l’étirement

- Forcer la rigidité
- Physiquement « plausible »
- Impacte sur les texture, l’étirement et le volume



Qu'est-ce que la rigidité ?

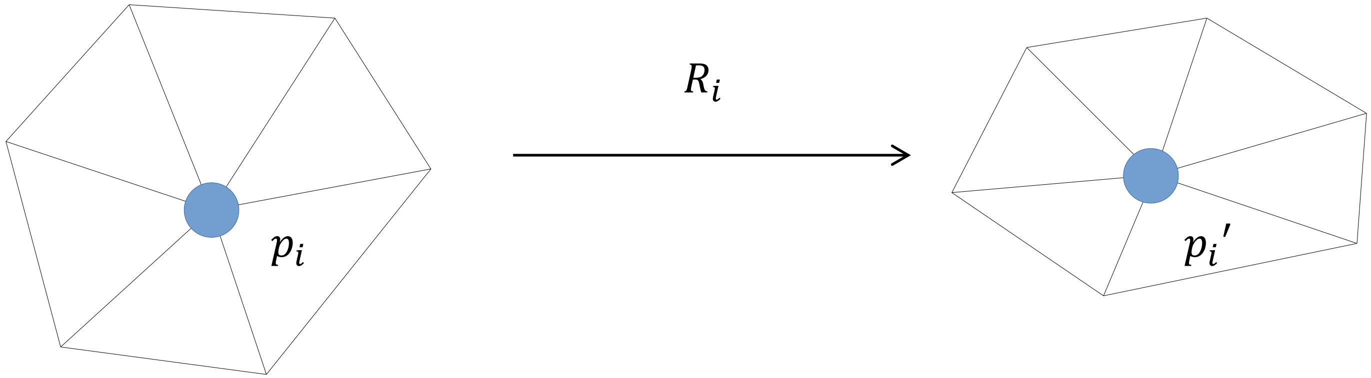
- Un petit morceau de la forme doit être globalement tourné :



$$\mathbf{p}_i' - \mathbf{p}_j' = \mathbf{R}_i \left(\mathbf{p}_i - \mathbf{p}_j \right), \ \ \forall \ j \in \mathcal{N}(i)$$

Energie rigide

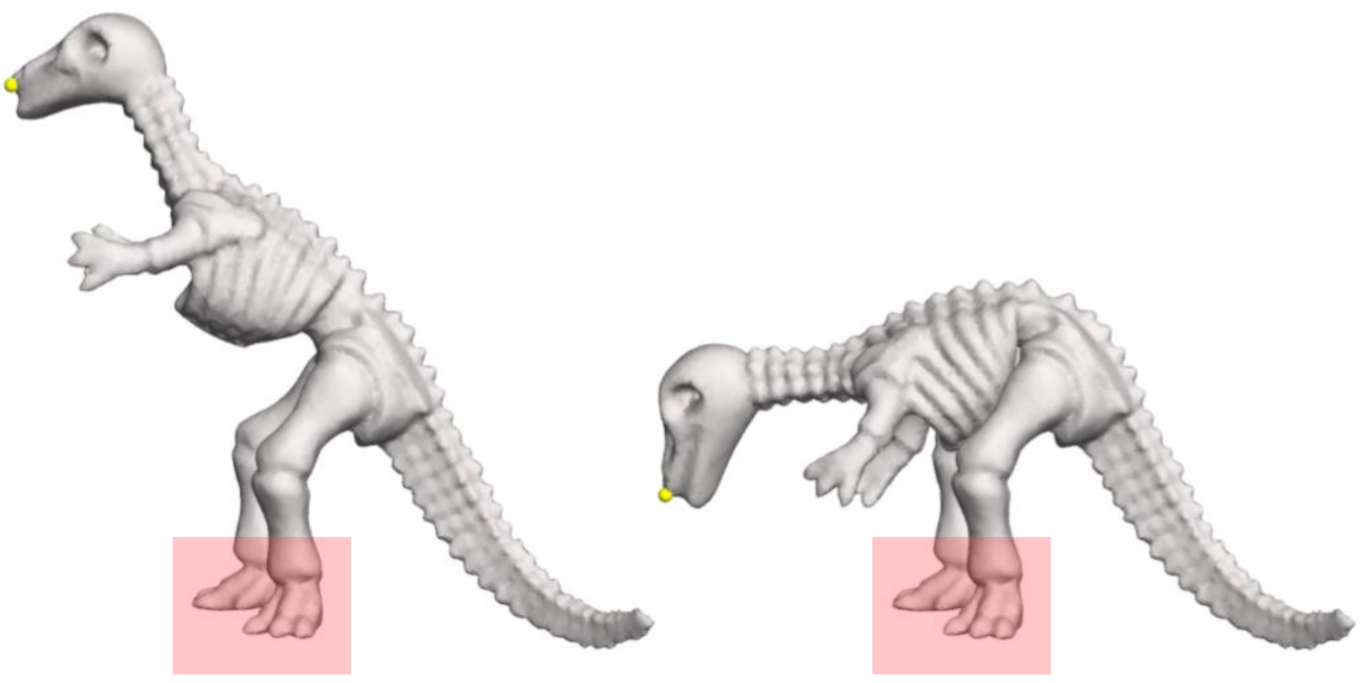
- Un petit morceau de la forme doit être globalement tourné :



$$E(\mathcal{C}_i, \mathcal{C}_i') = \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}_i' - \mathbf{p}_j') - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

ARAP framework

- Inconnues : Nouvelles positions p’ ET rotations R
- Contraintes : Quelques positions spécifiques



Problème

- Inconnues : Nouvelles positions p’ ET rotations R
- NON-LINEAIRE et NON-CONVEXE
- Minimiser R et p’ en même temps n’est pas faisable
- C’est l’energie rigide seulement, il faut ajouter l’energies des constraines « handles » !

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| \left(\mathbf{p}_i' - \mathbf{p}_j'\right) - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

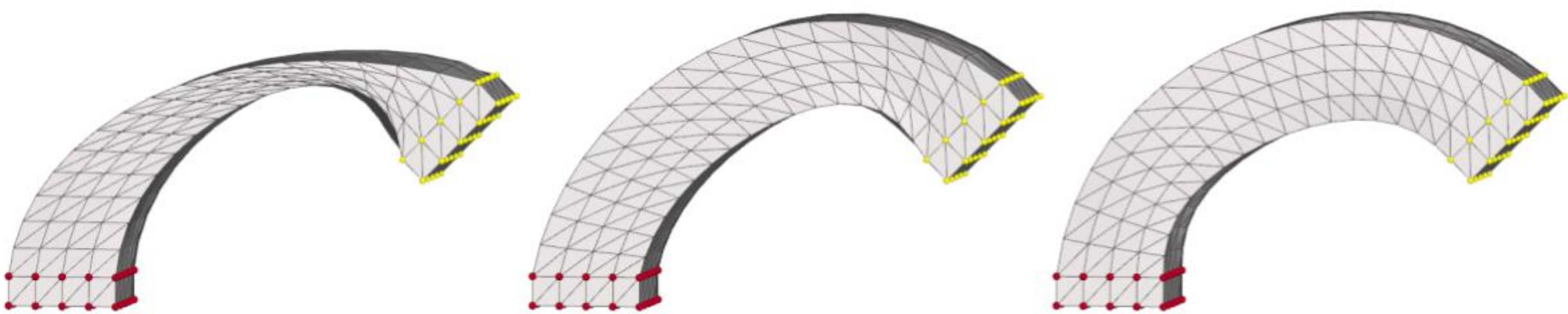
Ad hoc solution

- Fixer R, optimiser p’
- Fixer p’, optimiser R
- Fixer R, optimiser p’
- …

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| \left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

Ad hoc solution

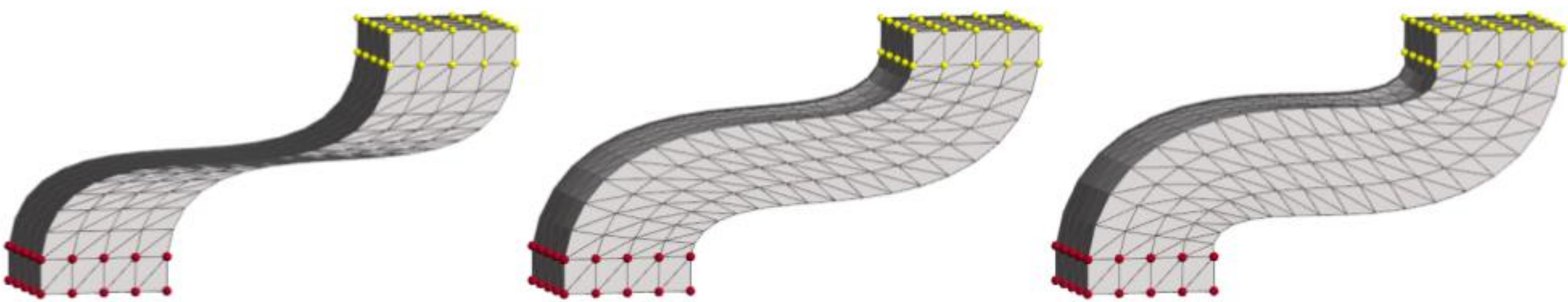
- Fixer R, optimiser p’
- Fixer p’, optimiser R
- Fixer R, optimiser p’
- …



initial guess 1 iteration 4 iterations

Ad hoc solution

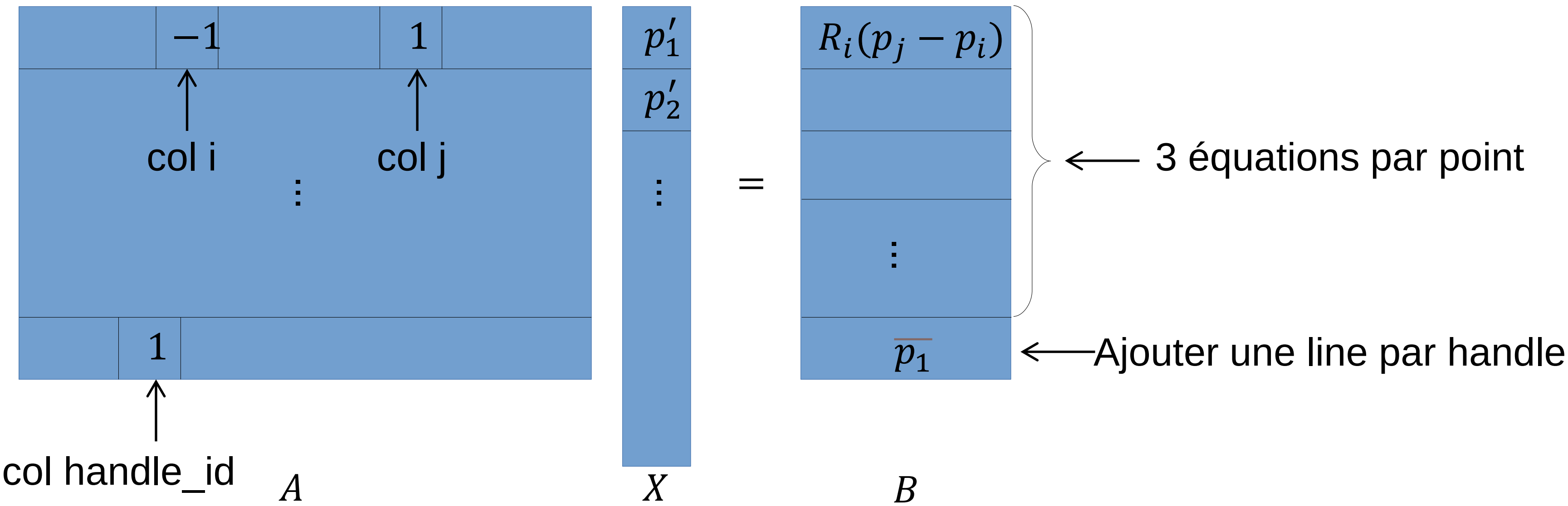
- Fixer R, optimiser p’
- Fixer p’, optimiser R
- Fixer R, optimiser p’
- …



initial guess 1 iteration 2 iterations

1) Fixer R, optimiser p’

Construction du système linéaire



$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| \left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i \left(\mathbf{p}_i - \mathbf{p}_j\right) \right\|^2$$

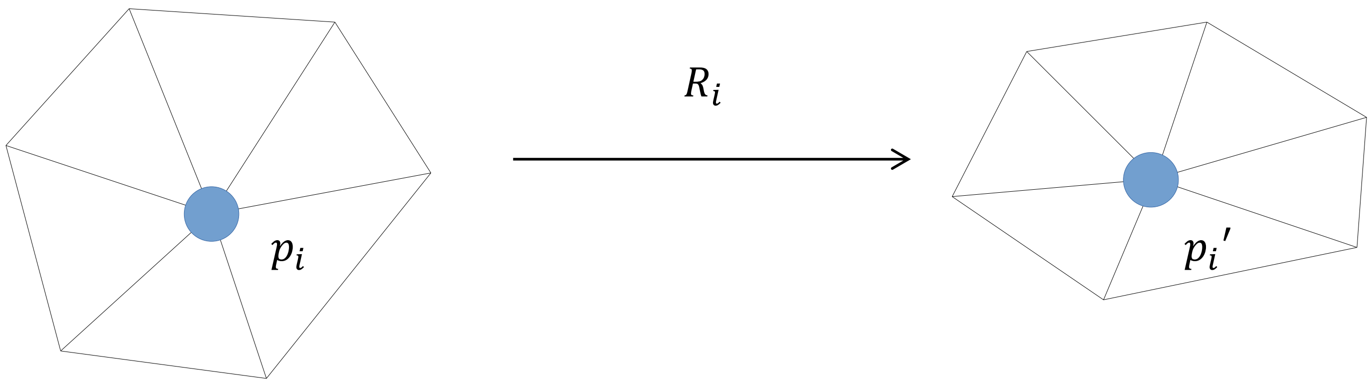
2) Fixer p', optimiser R

- Peut-être fait par sommet i

$$E\left(\mathcal{S}'\right) = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| \left(\mathbf{p}'_i - \mathbf{p}'_j\right) - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

2) Fixer p', optimiser R

- Peut-être fait par sommet i
- Minimiser E(Ci, Ci')



$$E(\mathcal{C}_i, \mathcal{C}_i') = \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}_i' - \mathbf{p}_j') - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

2) Fixer p', optimiser R

$\sum_j w_j \|e_j' - R.e_j\|^2$ $\longrightarrow$ A minimiser

$$\sum_j w_j \|e_j' - R.e_j\|^2 = \sum_j w_j \|e_j'\|^2 + \sum_j w_j \|R.e_j\|^2 - 2\sum_j w_j (R.e_j)^T.e_j'$$

$$\sum_j w_j \|e_j' - R.e_j\|^2 = const - 2\sum_j w_j Trace(R.e_{j.}e_j'^T)$$

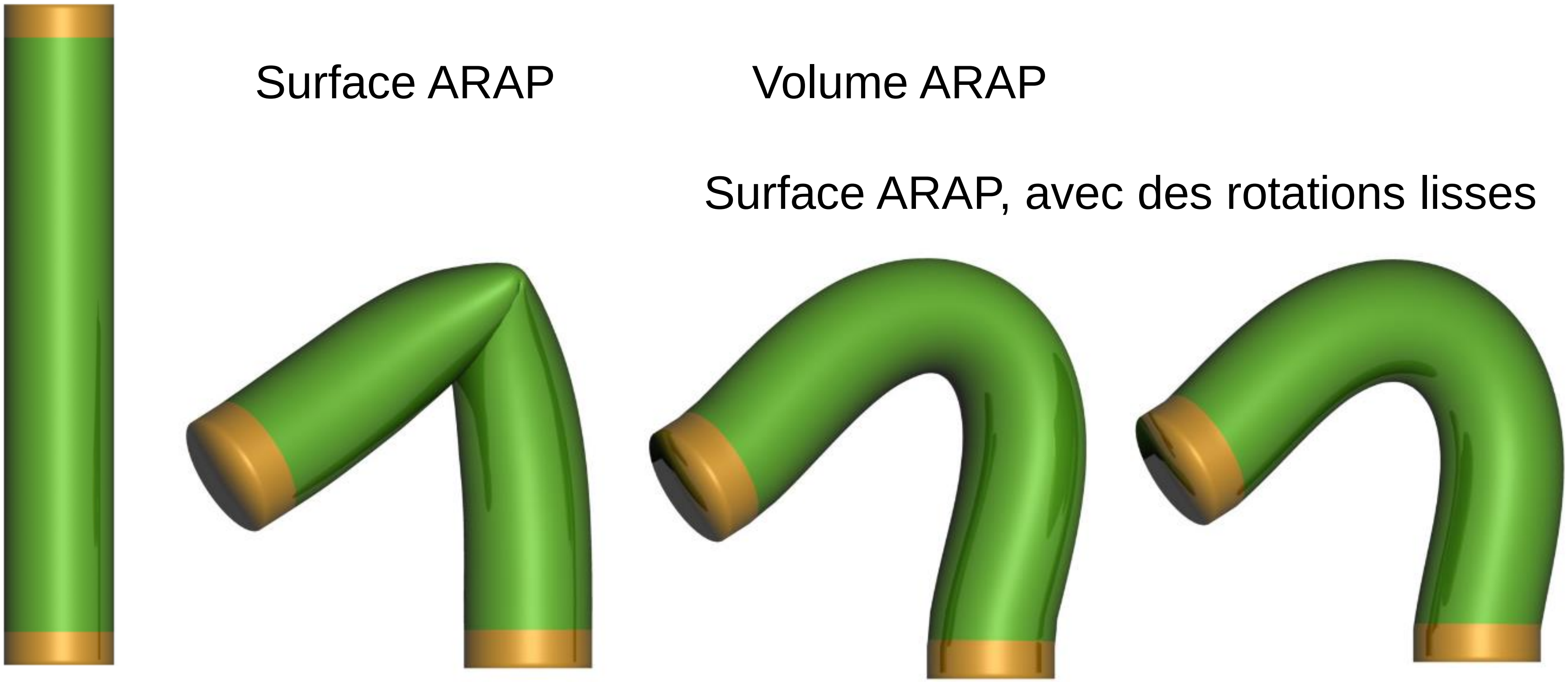
$Trace\left(R.\sum_j w_j e_{j.}e_j'^T\right)$ $\longrightarrow$ A maximiser

1) Build $S := \sum_j w_j e_j'.e_j^T$

2) Compute SVD : $S := U.\Sigma.V^T$

3) Solution : $R := U.diag\left(1,1,det(U.V^T)\right).V^T$

Different flavours of ARAP



[Levi & Gotsman] : Smooth Rotation Enhanced As-Rigid-As-Possible Mesh Animation

Déformation de l’espace

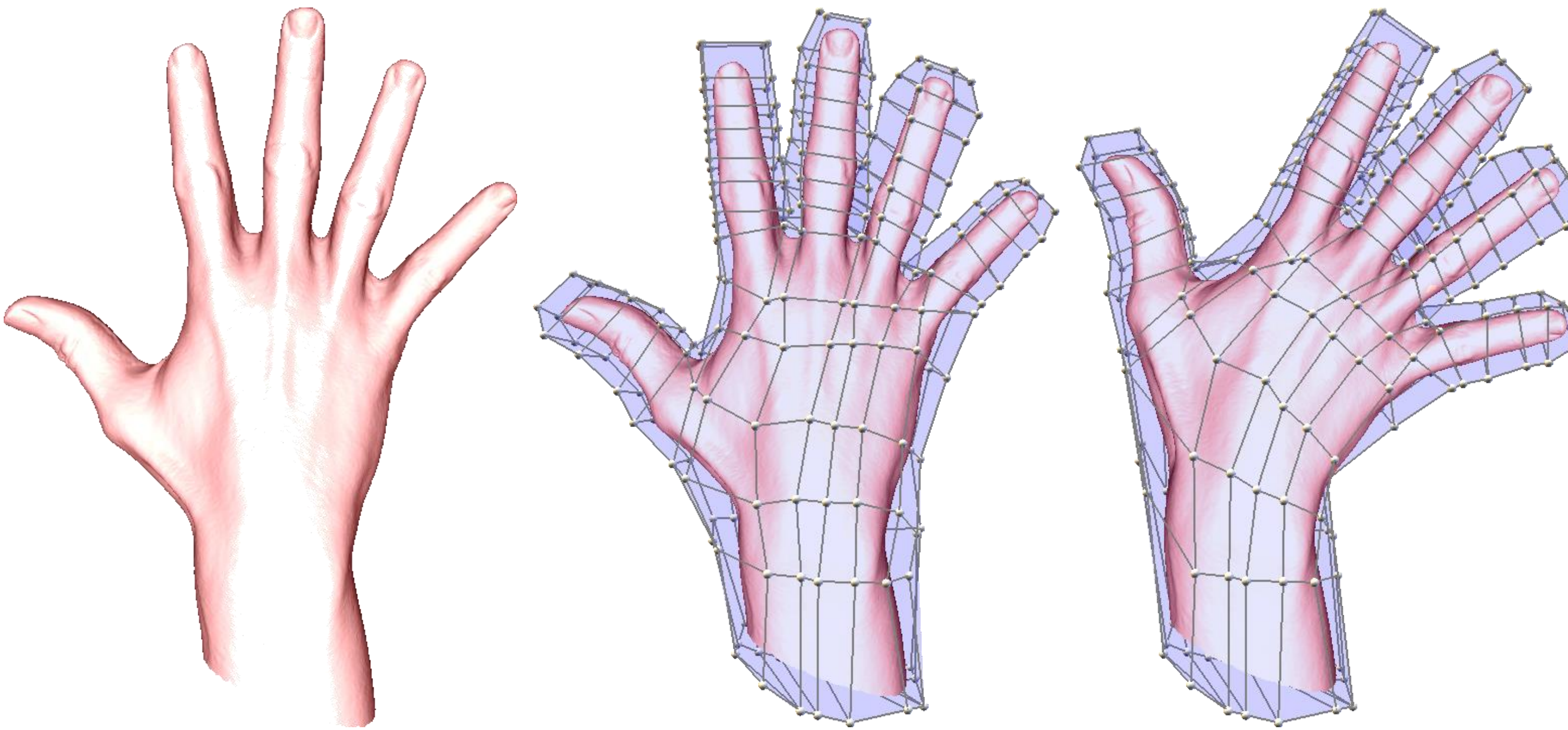
- Peut-être appliquée à toutes les géométries :
  - maillages (non-variétés, plusieures composantes)
  - Soupe de polygones
  - Nuages de points
  - Données volumiques

- Complexité découplée de la complexité de la géométrie
  - Choix de la complexité adaptée à la déformation

Déformation de l’espace

- Utilisation d'une cage (maillage a géométrie libre) pour contrôler l'animation d'un maillage
- Points de contrôle de déformation = Sommets de la cage $\boldsymbol{p}_j$
- $\boldsymbol{v}_i = \sum_j w_{ij}\, \boldsymbol{p}_i$ avec précalcule des w_{ij}
- Comment calculer les influences des sommets de la cage sur le maillage a animer? Quelles coordonnées utiliser?

Déformation de l’espace

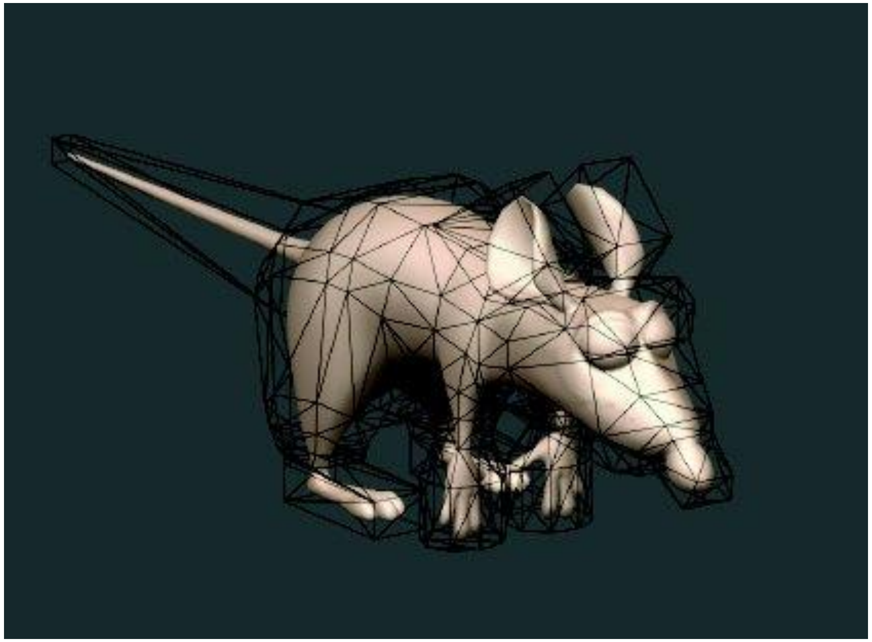


High resolution model

Cage coordinates computation

High resolution model deformation

Cage-based deformations



Utilisé ?

Ratatouille (2007) and later movies from Disney & Pixar

Pour animer quoi ?

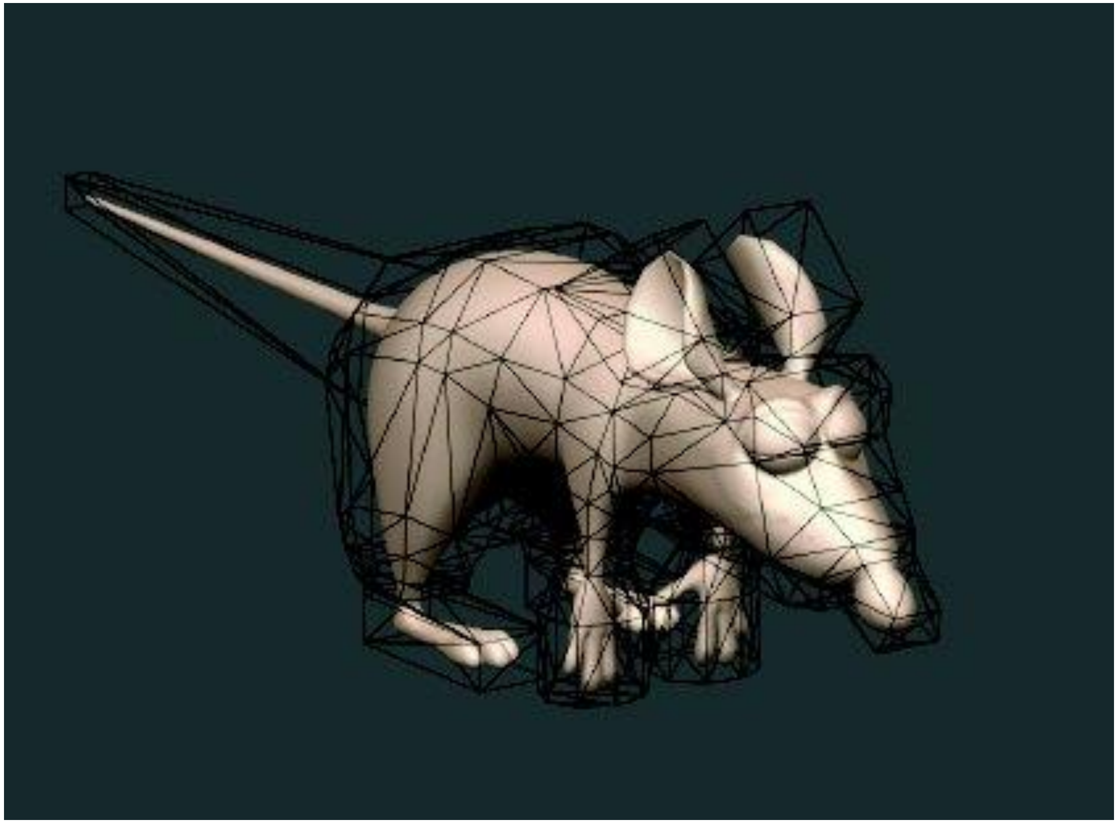
- Animer le corps
  - Pas la fourrure
  - Pas les particules …

Problèmatique

- Qu’est-ce qu’une cage ?
- Comment définir sa transformation ?
- Comment la transférer au maillage ?
- ...

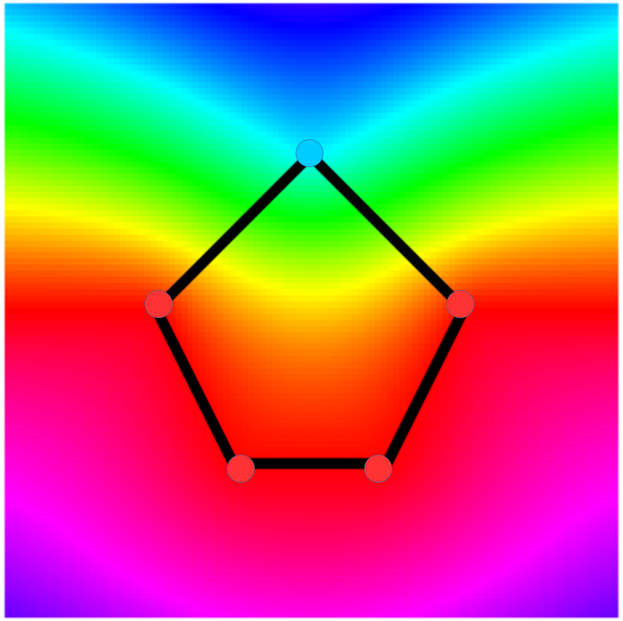
Une cage

- Typiquement : un maillage fermé englobant la surface

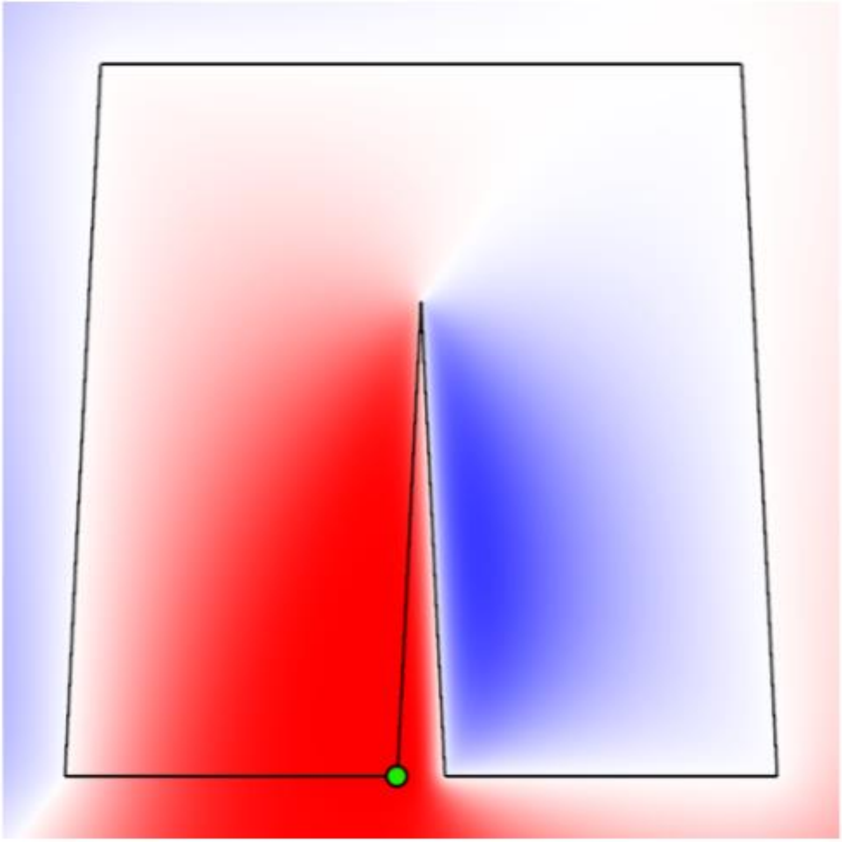


« boundary interpolation »

- Une fonction f est définie sur la cage (la frontière)
- Extrapoler f à l'intérieur de la cage
- f = « new position » → espace déformé



Coordonnées de cage



$$f(p) = \sum_i \phi_i(p) . f_i$$

Coordonnées idéales

- Linear reproduction : $p = \sum_i \phi_i(p).\, c_i$
- Sum to 1 for every vertex : $\sum_i \phi_i(p) = 1$
- Smooth
- Positive : $\phi_i(p) \geq 0$
- Local « enough »
- Closed-form expression

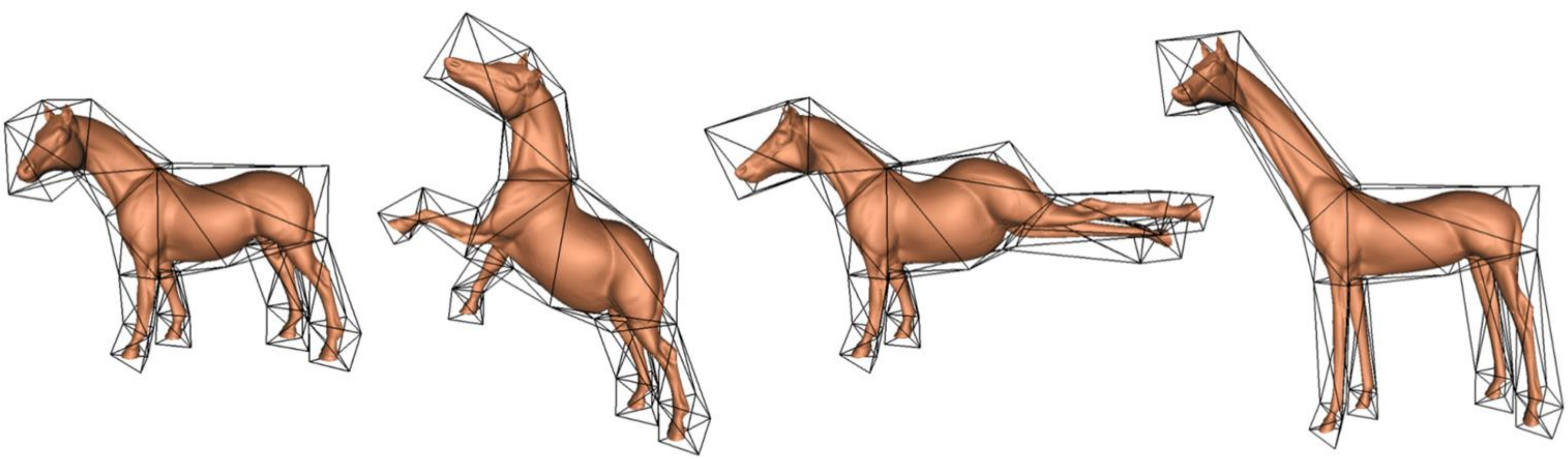
$$f(p) = \sum_i \phi_i(p).\, f_i$$

Plusieurs sortes

- Mean-value coordinates
- Harmonic coordinates
- Positive mean-value coordinates
- Green coordinates
- ...

Mean-value coordinates

$$f^{MVC}(\eta) = \frac{\int_{B_\eta(M)} \frac{f(\xi)}{|\xi-\eta|} dS_\eta(\xi)}{\int_{B_\eta(M)} \frac{1}{|\xi-\eta|} dS_\eta(\xi)}$$



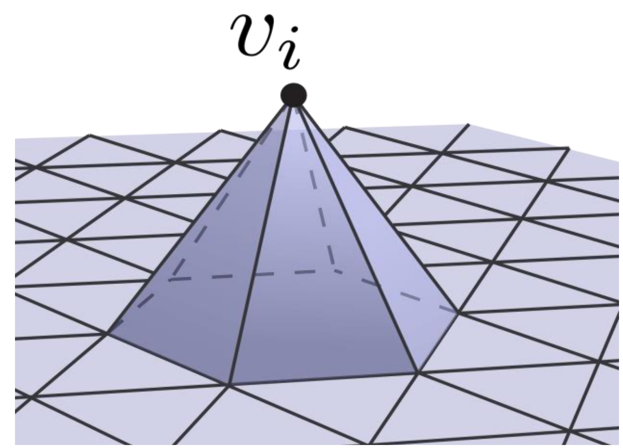
[Ju et al.] : Mean Value Coordinates for Closed Triangular Meshes

Mean-value coordinates

$$f^{MVC}(\eta) = \frac{\int_{B_\eta(M)} \frac{f(\xi)}{|\xi-\eta|} dS_\eta(\xi)}{\int_{B_\eta(M)} \frac{1}{|\xi-\eta|} dS_\eta(\xi)}$$

: « Moyennage » des valeurs de la cage

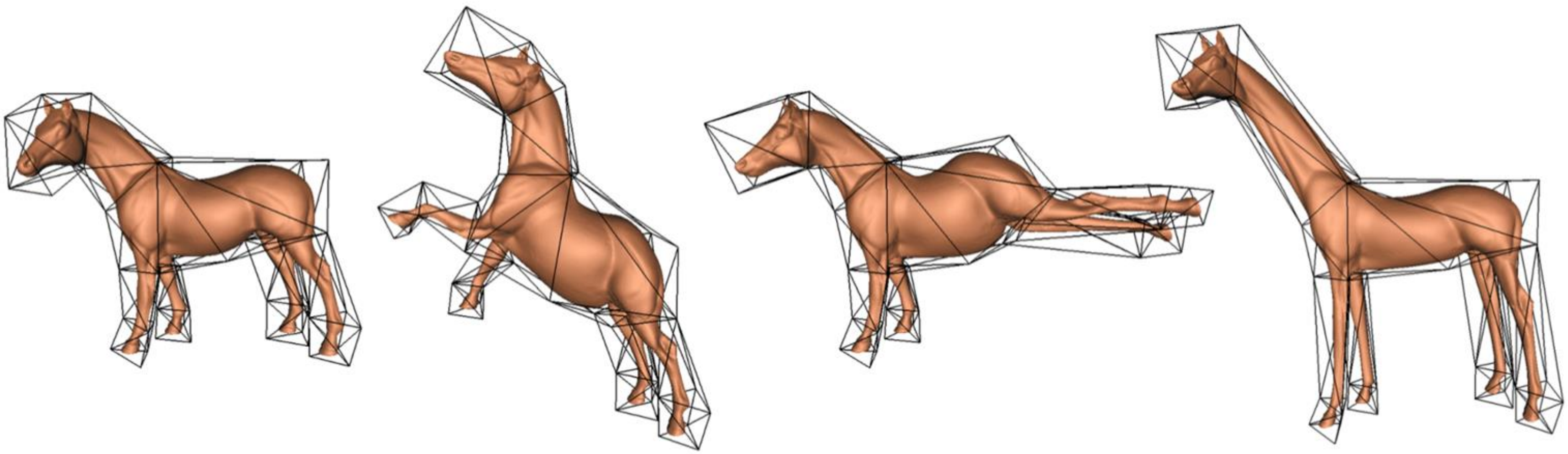
$$f(\xi) = \sum_{v_i} \Gamma_i(\xi) f_i$$

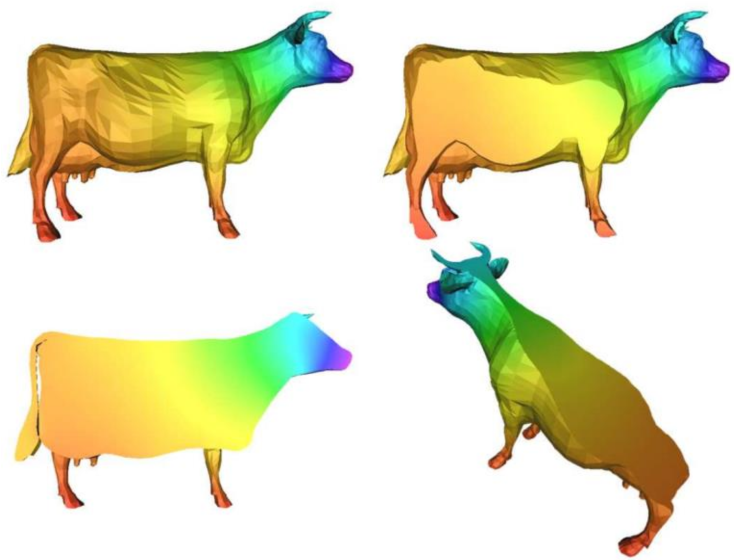


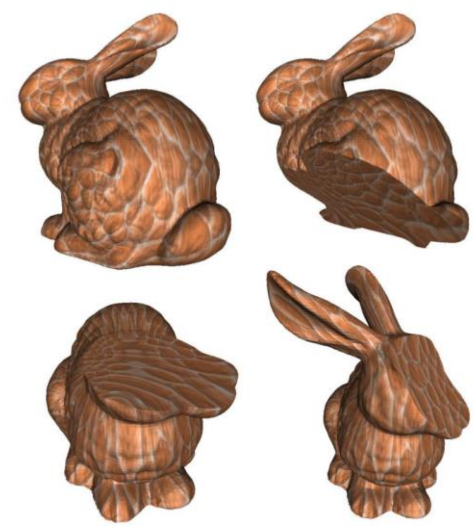
f est linéaire par triangle

$$f^{MVC}(\eta) = \sum_{v_i} \lambda_i^{MVC}(\eta) f_i \qquad \lambda_i^{MVC}(\eta) = \frac{\int_{B_\eta(M)} \frac{\Gamma_i(\xi)}{|\xi-\eta|} dS_\eta(\xi)}{\int_{B_\eta(M)} \frac{1}{|\xi-\eta|} dS_\eta(\xi)}$$

Mean-value coordinates







Mean-value coordinates

```
// MVC : Code from "Mean Value Coordinates for Closed Triangular Meshes" Schaeffer Siggraph 2005
template< class int_t , class float_t , class point_t >
bool computeCoordinatesOriginalCode(
        point_t const & eta ,
        std::vector< std::vector< int_t > > const & cage_triangles , std::vector< point_t > const & cage_vertices , std::vector< point_t > const & cage_normals ,
        std::vector< float_t > & weights , std::vector< float_t > & w_weights)
{
    typedef typename point_t::type_t    T;
    unsigned int n_vertices = cage_vertices.size() , n_triangles = cage_triangles.size();
    assert( cage_normals.size() == cage_triangles.size()   &&    "cage_normals.size() != cage_triangles.size()" );
    T epsilon = 0.00000001;

    w_weights.clear();
    weights.clear();
    weights.resize( n_vertices , 0.0 );
    T sumWeights = 0.0;

    std::vector< T > d( n_vertices , 0.0 ); std::vector< point_t > u( n_vertices );

    for( unsigned int v = 0 ; v < n_vertices ; ++v ) {
        d[ v ] = ( eta - cage_vertices[ v ] ).norm();
        if( d[ v ] < epsilon ) {
            weights[v] = 1.0;
            return true;
        }
        u[ v ] = ( cage_vertices[v] - eta ) / d[v];
    }

    w_weights.resize(  n_vertices , 0.0 );

    unsigned int vid[3];     T l[3]; T theta[3] ; T w[3]; T c[3]; T s[3];
    for( unsigned int t = 0 ; t < n_triangles ; ++t ) { // the Norm is CCW :
        for( unsigned int i = 0 ; i <= 2 ; ++i ) vid[i] =  cage_triangles[t][i];
        for( unsigned int i = 0 ; i <= 2 ; ++i ) l[ i ] = ( u[ vid[ ( i + 1 ) % 3 ] ] - u[ vid[ ( i + 2 ) % 3 ] ] ).norm();
        for( unsigned int i = 0 ; i <= 2 ; ++i ) theta[i] = 2.0 * asin( l[i] / 2.0 );
        T h = ( theta[0] + theta[1] + theta[2] ) / 2.0;
        if( M_PI - h < epsilon ) { // eta is on the triangle t , use 2d barycentric coordinates :
            for( unsigned int i = 0 ; i <= 2 ; ++i ) w[ i ] = sin( theta[ i ] ) * l[ (i+2) % 3 ] * l[ (i+1) % 3 ];

            sumWeights = w[0] + w[1] + w[2];

            w_weights.clear();
            weights[ vid[0] ] = w[0] / sumWeights;
            weights[ vid[1] ] = w[1] / sumWeights;
            weights[ vid[2] ] = w[2] / sumWeights;
            return true;
        }

        for( unsigned int i = 0 ; i <= 2 ; ++i ) c[ i ] = ( 2.0 * sin(h) * sin(h - theta[ i ]) ) / ( sin(theta[ (i+1) % 3 ]) * sin(theta[ (i+2) % 3 ]) ) - 1.0;

        T sign_Basis_u0u1u2 = 1;
        if( point_t::dot( point_t::cross(u[vid[0]] , u[vid[1]]) , u[vid[2]] ) < 0.0 ) sign_Basis_u0u1u2 = -1;
        for( unsigned int i = 0 ; i <= 2 ; ++i ) s[ i ] = sign_Basis_u0u1u2 * sqrt( std::max<double>( 0.0 , 1.0 - c[ i ] * c[ i ] ) );
        if( fabs( s[0] ) < epsilon   ||   fabs( s[1] ) < epsilon   ||   fabs( s[2] ) < epsilon ) continue; // eta is on the same plane, outside t  ->  ignore triangle t :
        for( unsigned int i = 0 ; i <= 2 ; ++i ) w[ i ] = ( theta[ i ] - c[ (i+1)% 3 ]*theta[ (i+2) % 3 ] - c[ (i+2) % 3 ]*theta[ (i+1) % 3 ] ) / ( 2.0 * d[ vid[i] ] * sin( theta[ (i+1) % 3 ] ) * s[ (i+2) % 3 ] );

        sumWeights += ( w[0] + w[1] + w[2] );
        w_weights[ vid[0] ] += w[0];
        w_weights[ vid[1] ] += w[1];
        w_weights[ vid[2] ] += w[2];
    }

    for( unsigned int v = 0 ; v < n_vertices ; ++v ) weights[v]  = w_weights[v] / sumWeights;

    return false;
}
```

50-60 lines of code

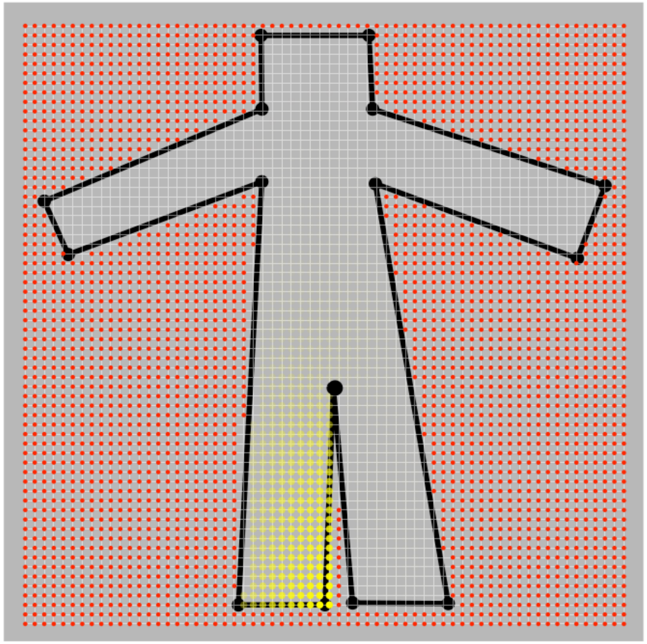
Mean-value coordinates

- Closed-form expression : +
- Définies partout dans l’espace : +
- Pas toujours positives: -

[Ju et al.] : Mean Value Coordinates for Closed Triangular Meshes

Harmonic coordinates

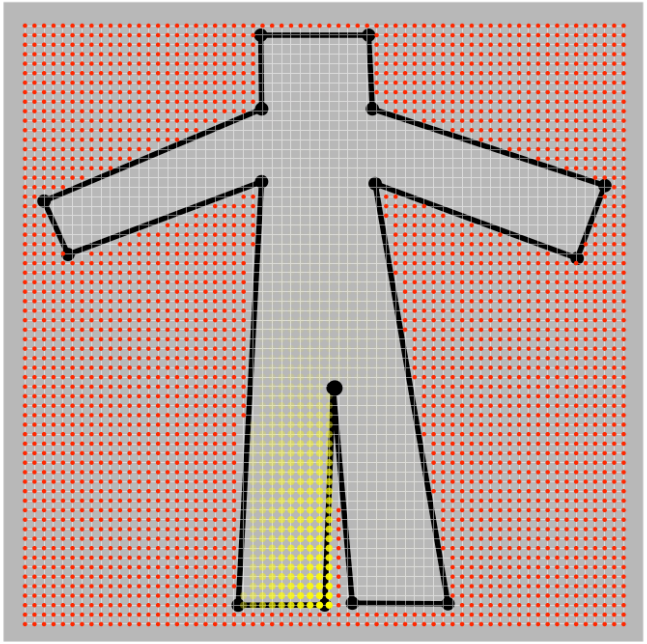
- Algo, pour chaque sommet de la cage :
  - Setup discrete grid
  - Set boundary values
  - Résoudre pour un Laplacian nul
    - Système linéaire
    - Moyennage itératif
  - Valeur aux sommets du maillage en utilisant une interpolation bilinéaire de la grille



[DeRose & Meyer] : Harmonic Coordinates

Harmonic coordinates

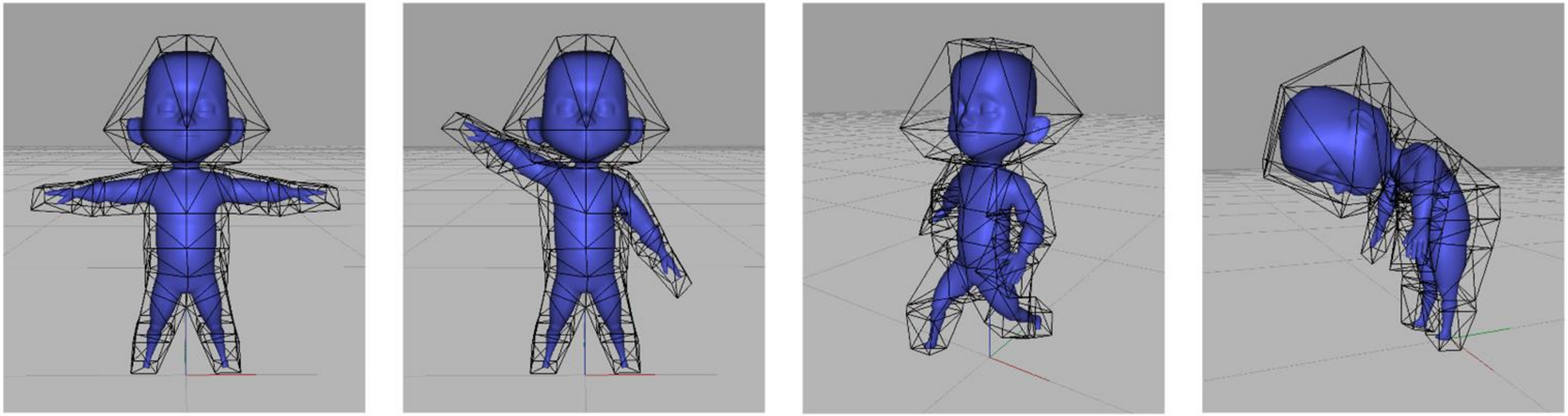
- No closed-form expression : -
- Définies uniquement à l'intérieur : -
- Toujours positive: +
- Plutôt locales : +



[DeRose & Meyer] : Harmonic Coordinates

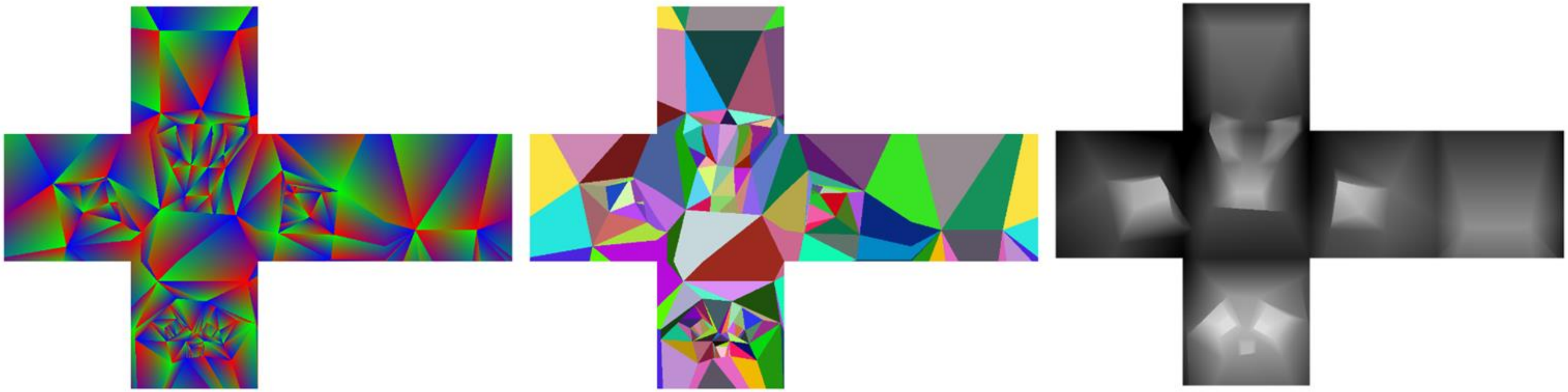
Harmonic coordinates

- Se trouve :
  - Chez Pixar
  - Dans Blender



Positive mean-value coordinates

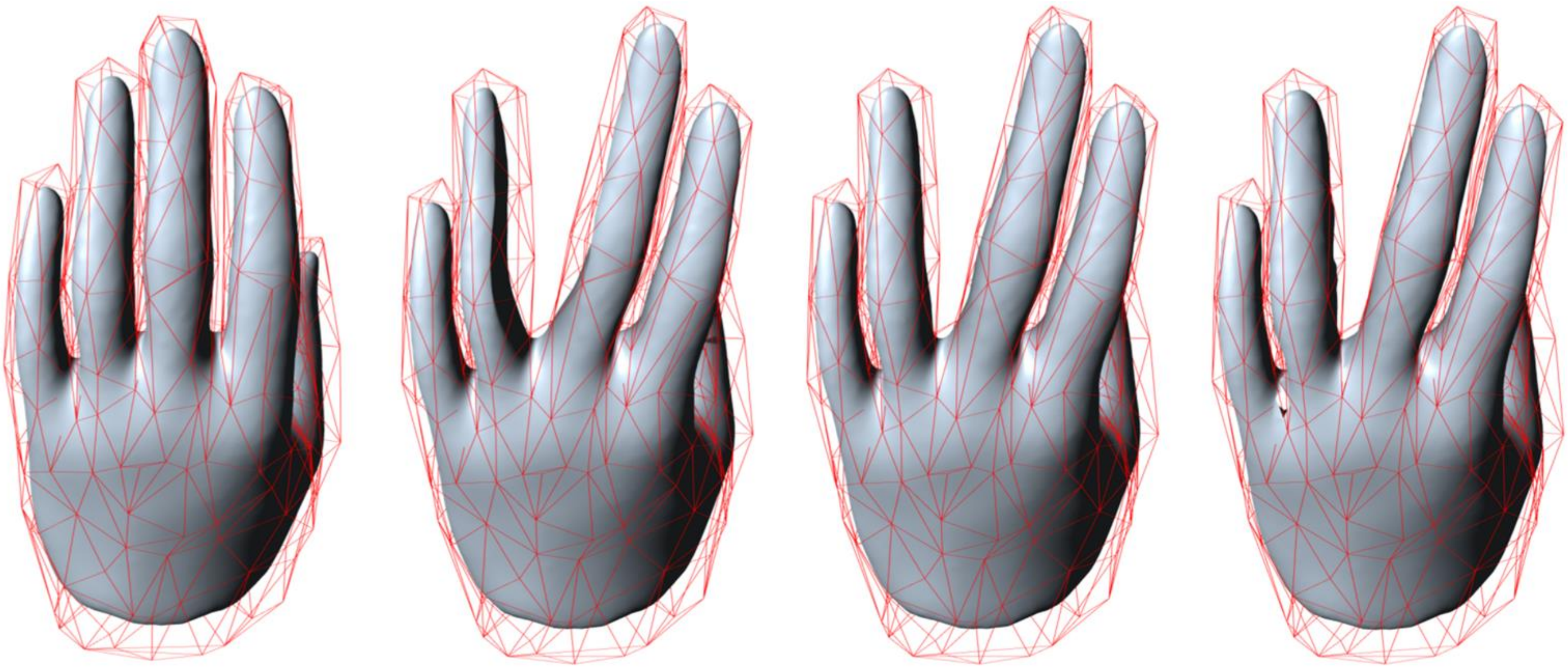
- Basées sur les mean-value coordinates
- Utilise uniquement la partie de la cage visible de p
  - No closed-form expression
  - Positives
- Utilise le GPU pour approximer une intégrale sphérique



Color Buffer Index Buffer Depth Buffer

[Lipman et al.] : GPU-assisted Positive Mean Value Coordinates for Mesh Deformations

Positive mean-value coordinates



Undeformed MVC (13.5 sec) PMVC (18.5 sec) HC (333.61 sec, 64^3 voxels)

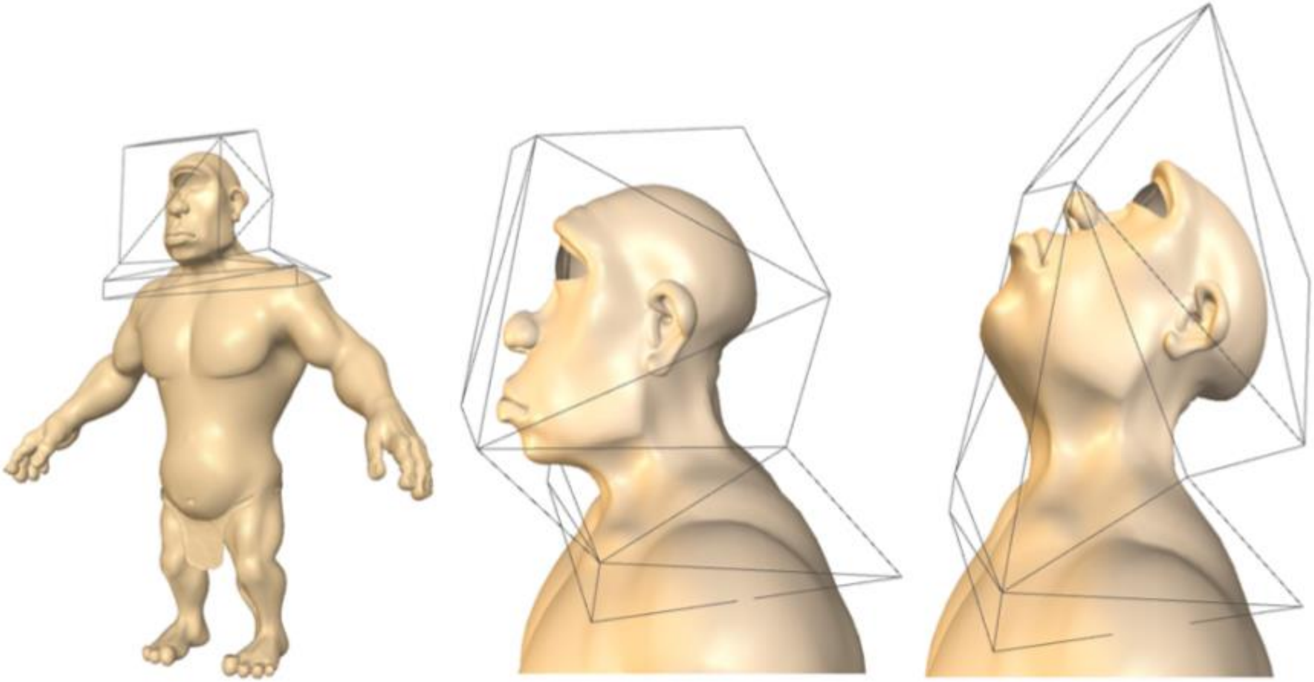
Positive mean-value coordinates

- No closed-form expression : -
- Définies uniquement à l'intérieur : -
- Toujours positives : +
- Plutôt locales : +

[Lipman et al.] : GPU-assisted Positive Mean Value Coordinates for Mesh Deformations

Green coordinates

$$f(\eta) = \int_{\xi \in \partial D} f(\xi) \frac{\partial_1 G}{\partial n_\xi}(\xi, \eta) ds_\xi - \int_{\xi \in \partial D} G(\xi, \eta) \frac{\partial f}{\partial n_\xi}(\xi) ds_\xi$$

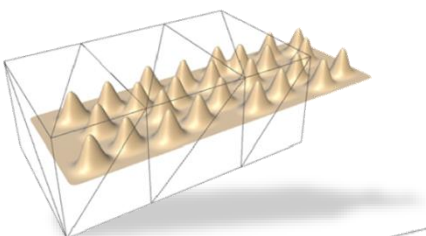


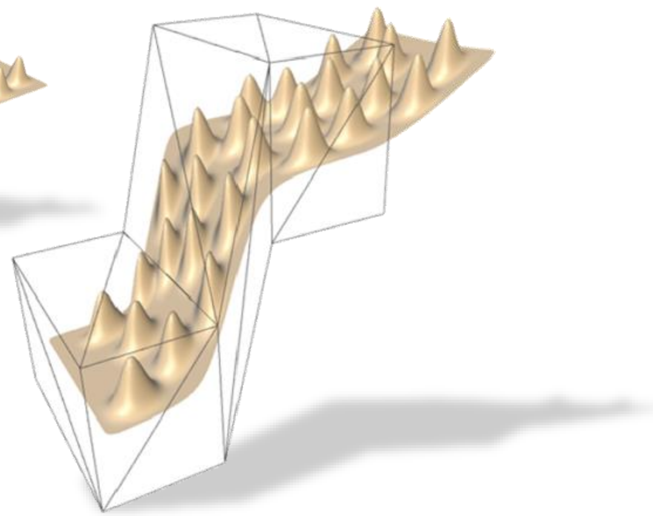
[Lipman et al.] : Green Coordinates

Coordonnées de cage

Green Coordinates [Lipman 2008]

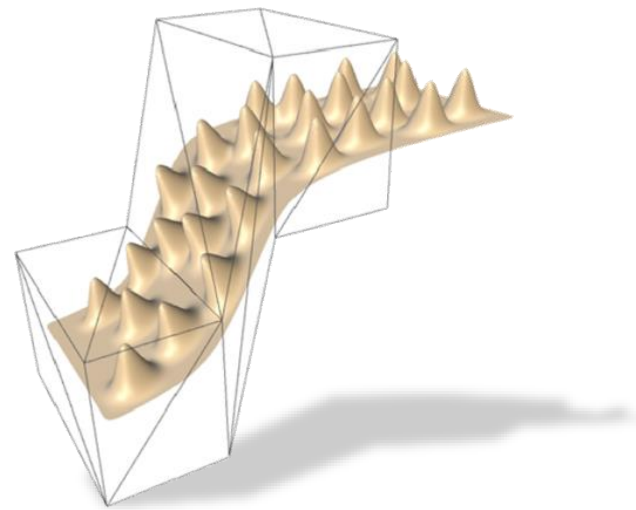
- Déformation quasi conforme de l'espace
- Préservation des caractéristiques





Mean Value Coordinates
MVC [JSW05]

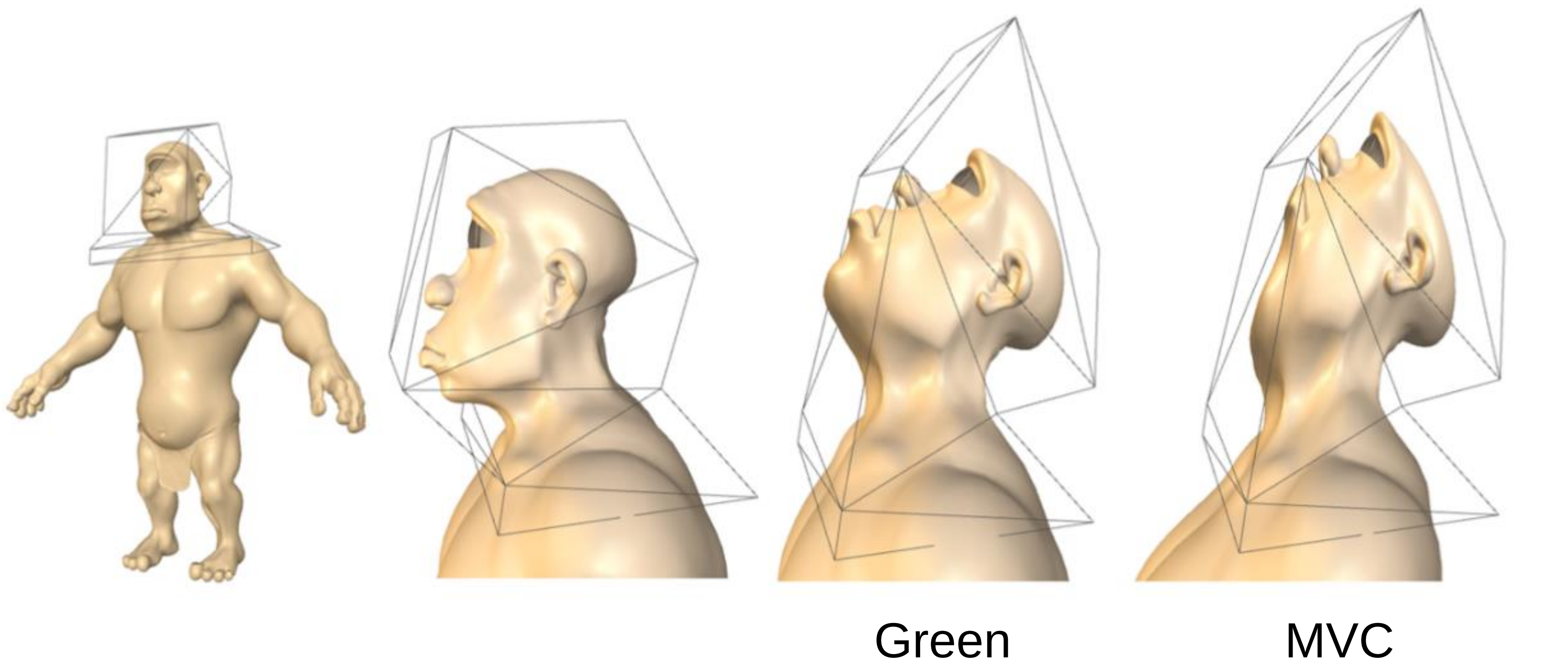
$$F(\boldsymbol{\eta}; P) = \sum_{i \in I_{\mathbb{V}}} \varphi_i(\boldsymbol{\eta}) \boldsymbol{v}_i$$



Green Coordinates
GC [LL08]

$$F(\boldsymbol{\eta}; P) = \sum_{i \in I_{\mathbb{V}}} \phi_i(\boldsymbol{\eta}) \boldsymbol{v}_i + \sum_{j \in I_{\mathbb{T}}} \psi_j(\boldsymbol{\eta}) \boldsymbol{n}(t_j)$$

Green coordinates



Green coordinates

- Closed-form expression : +
- Définies uniquement à l’intérieur : -
- Pas toujours positives mais produit des résultats attendus : +
- Lisses (harmonic) : +
- Quasi-conforme : +
- Utilise les normals pour inférer les rotations à partir de la translation des sommets de la cage : +

Green coordinates

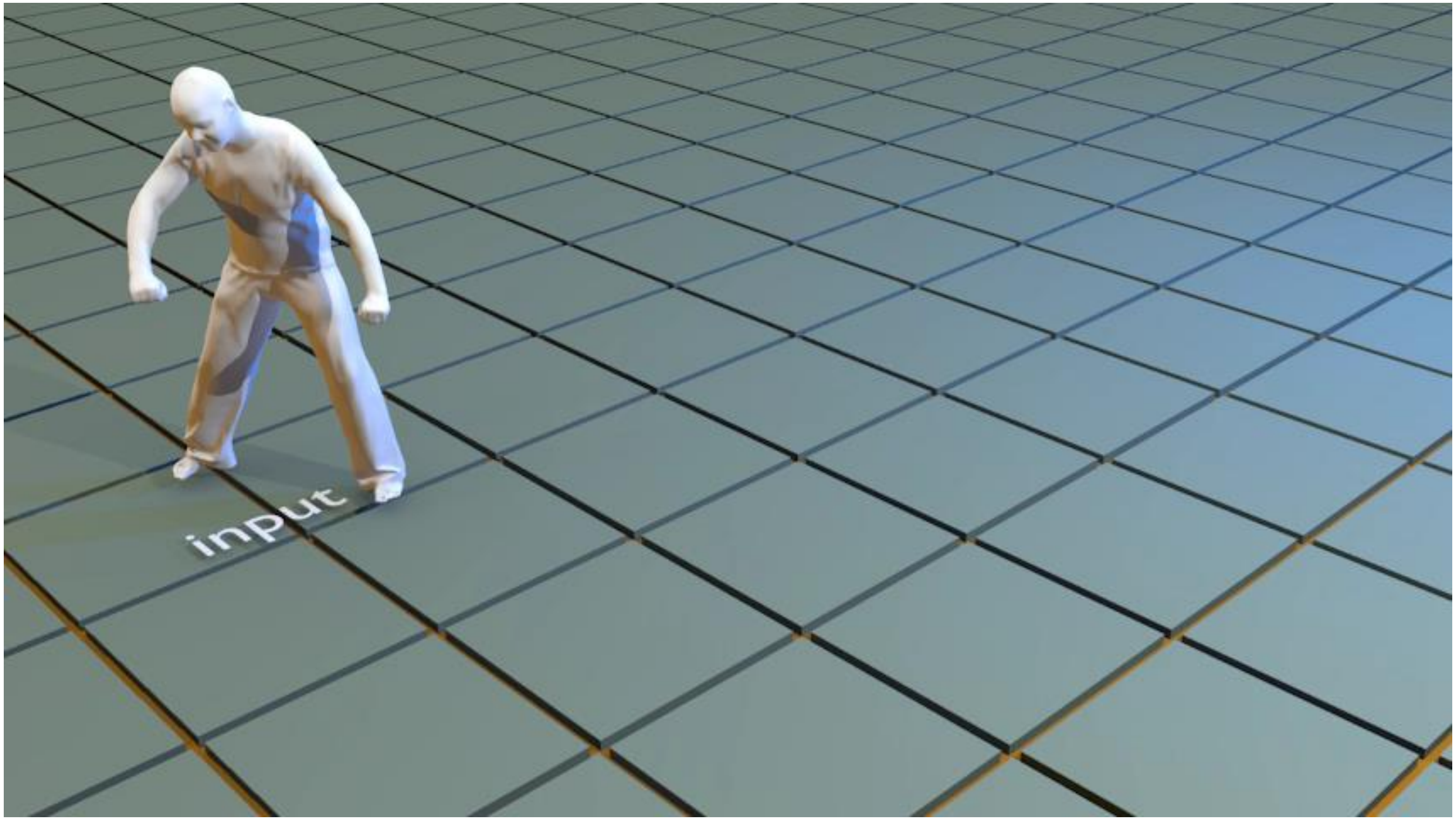
```
// return the signed solid angle of the tetrahedron supported by the three vectors a,b, and c:
template< class point_t >
double __signed_solid_angle(point_t const & a, point_t const & b, point_t const & c) {
    typedef typename point_t::type_t    T;
    T det = point_t::dot(a,point_t::cross(b,c));
    if( fabs(det) < 0.0000000001 ) // then you're on the limit case where you cover half the sphere
        return M_2_PI; // that is particularly shitty, because the sign is difficult to estimate...

    T al = a.norm(),   bl = b.norm(),   cl = c.norm();

    T div = al*bl*cl + point_t::dot(a,b)*cl + point_t::dot(a,c)*bl + point_t::dot(b,c)*al;
    T at = atan2( fabs(det) , div );
    if(at < 0) at += M_PI; // If det>0 && div<0 atan2 returns < 0, so add pi.
    T omega = 2.0 * at;

    if(det > 0.0) return omega;
    return -omega;
}

template< class int_t , class float_t , class point_t >
void computeCoordinates(
        point_t const & eta ,
        std::vector< std::vector< int_t > > const & cage_triangles , std::vector< point_t > const & cage_vertices , std::vector< point_t > const & cage_normals ,
        std::vector< float_t > & _VC_phi , std::vector< float_t > & _FC_psi) {
    assert( cage_normals.size() == cage_triangles.size()   &&    "cage_normals.size() != cage_triangles.size()" );
    typedef typename point_t::type_t    T;

    _VC_phi.clear();  _VC_phi.resize( cage_vertices.size() , 0.0 );
    _FC_psi.clear();  _FC_psi.resize( cage_triangles.size() , 0.0 );

    // iterate over the triangles:
    for( unsigned int t = 0 ; t < cage_triangles.size() ; ++t ) {
        std::vector<int_t> const & tri = cage_triangles[t];
        point_t const & Nt = cage_normals[t];

        point_t e[3];    T e_norm[3];   point_t e_normalized[3];    T R[3];    point_t d[3];    T d_norm[3];     T C[3];     point_t J[3];
        for( unsigned int v = 0 ; v < 3 ; ++v ) e[v] = cage_vertices[tri[v]] - eta;
        for( unsigned int v = 0 ; v < 3 ; ++v ) e_norm[v] =  e[v].norm();
        for( unsigned int v = 0 ; v < 3 ; ++v ) e_normalized[v] = e[v] / e_norm[v];

        T signed_solid_angle = __signed_solid_angle (e_normalized[0], e_normalized[1], e_normalized[2]) / (4.f * M_PI);
        T signed_volume = point_t::dot( point_t::cross(e[0],e[1]) , e[2] ) / 6.0;
        T At = point_t::cross( cage_vertices[tri[1]]-cage_vertices[tri[0]], cage_vertices[tri[2]]-cage_vertices[tri[0]]).norm() / 2.0;

        for( unsigned int v = 0 ; v < 3 ; ++v ) R[v] = e_norm[(v+1)%3] + e_norm[(v+2)%3];
        for( unsigned int v = 0 ; v < 3 ; ++v ) d[v] = cage_vertices[tri[(v+1)%3]] - cage_vertices[tri[(v+2)%3]];
        for( unsigned int v = 0 ; v < 3 ; ++v ) d_norm[v] =  d[v].norm();
        for( unsigned int v = 0 ; v < 3 ; ++v ) C[v] = log( (R[v] + d_norm[v]) / (R[v] - d_norm[v]) ) / (4.0 * M_PI * d_norm[v]);

        point_t Pt( - signed_solid_angle * Nt );
        for( unsigned int v = 0 ; v < 3 ; ++v ) Pt += point_t::cross( Nt , C[v]*d[v] );
        for( unsigned int v = 0 ; v < 3 ; ++v ) J[v] = point_t::cross( e[(v+2)%3] , e[(v+1)%3] );

        _FC_psi[t] = - 3.0 * signed_solid_angle * signed_volume / At ;
        for( unsigned int v = 0 ; v < 3 ; ++v ) _FC_psi[t] -= C[v]* point_t::dot(J[v],Nt);
        for( unsigned int v = 0 ; v < 3 ; ++v ) _VC_phi[ tri[v] ] += point_t::dot(Pt , J[v]) / (2.0 * At);
    }
}
```

Roughly 50 lines of code

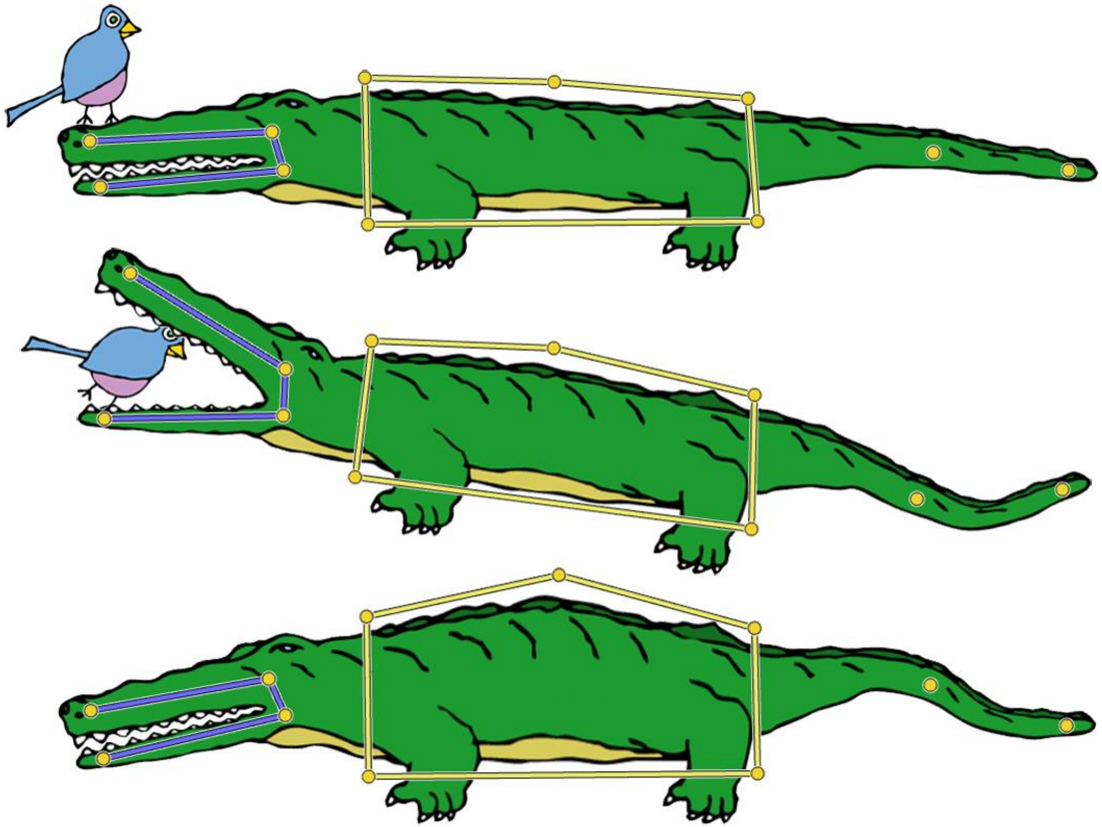
Autres applications

[Thiery et al 2013]

- Compression ou transfert d'animation

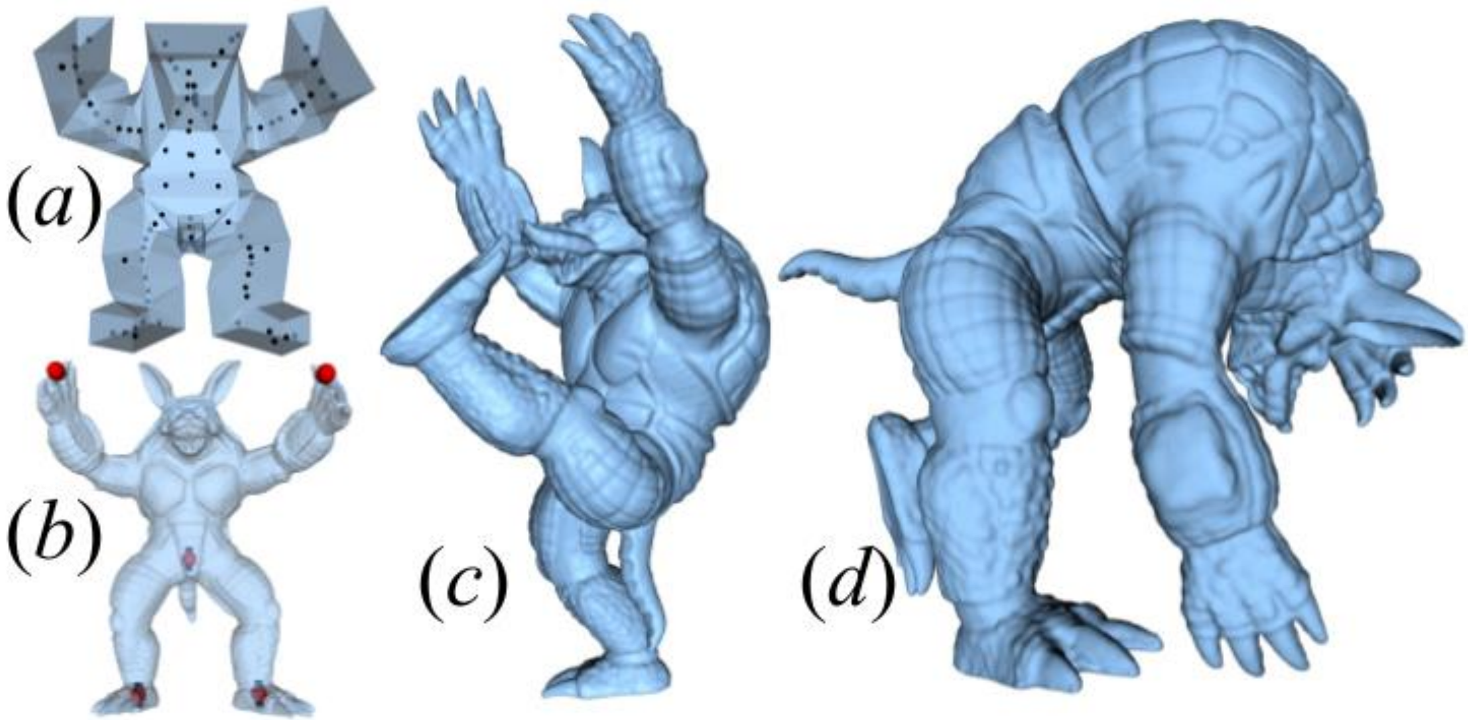


Combinaison de manipulateurs



Variational harmonic maps

- Let’s setup some functional together on the board
  - Positional constraints (red points)
  - Rigidity constraints (blue points near the medial axis)



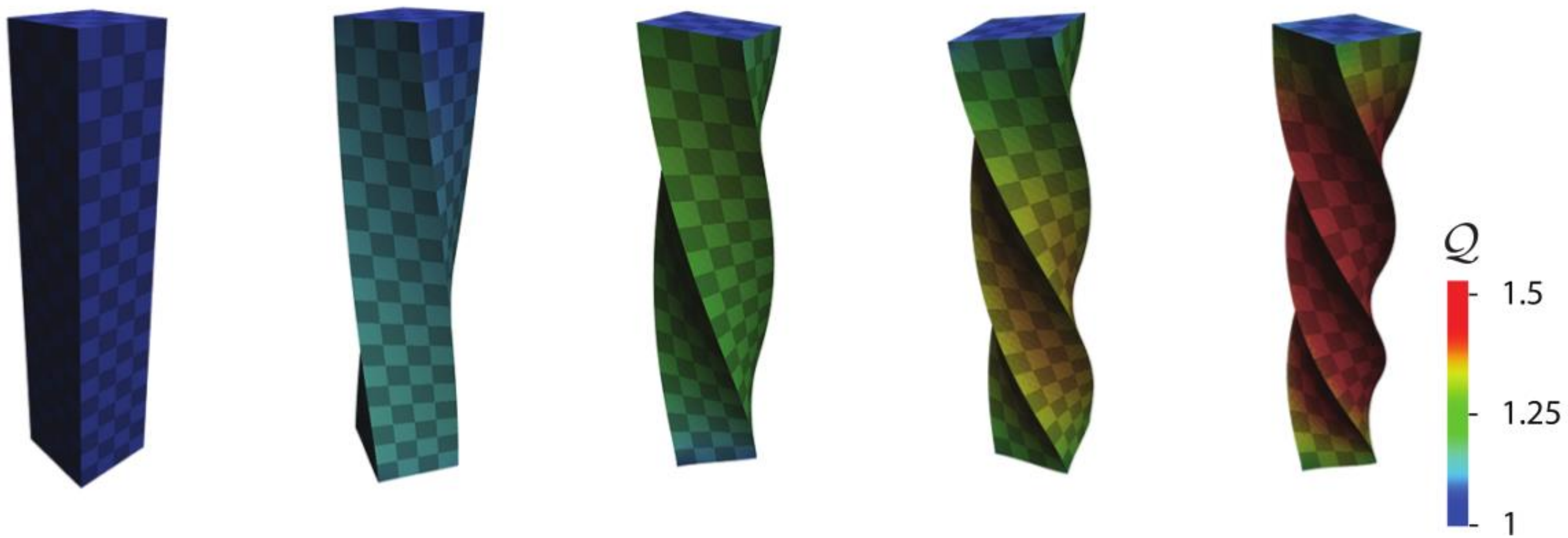
Conformal 3d surface deformations

[Crane et al.] : Spin Transformations of Discrete Surfaces

Conformal 3d surface deformations

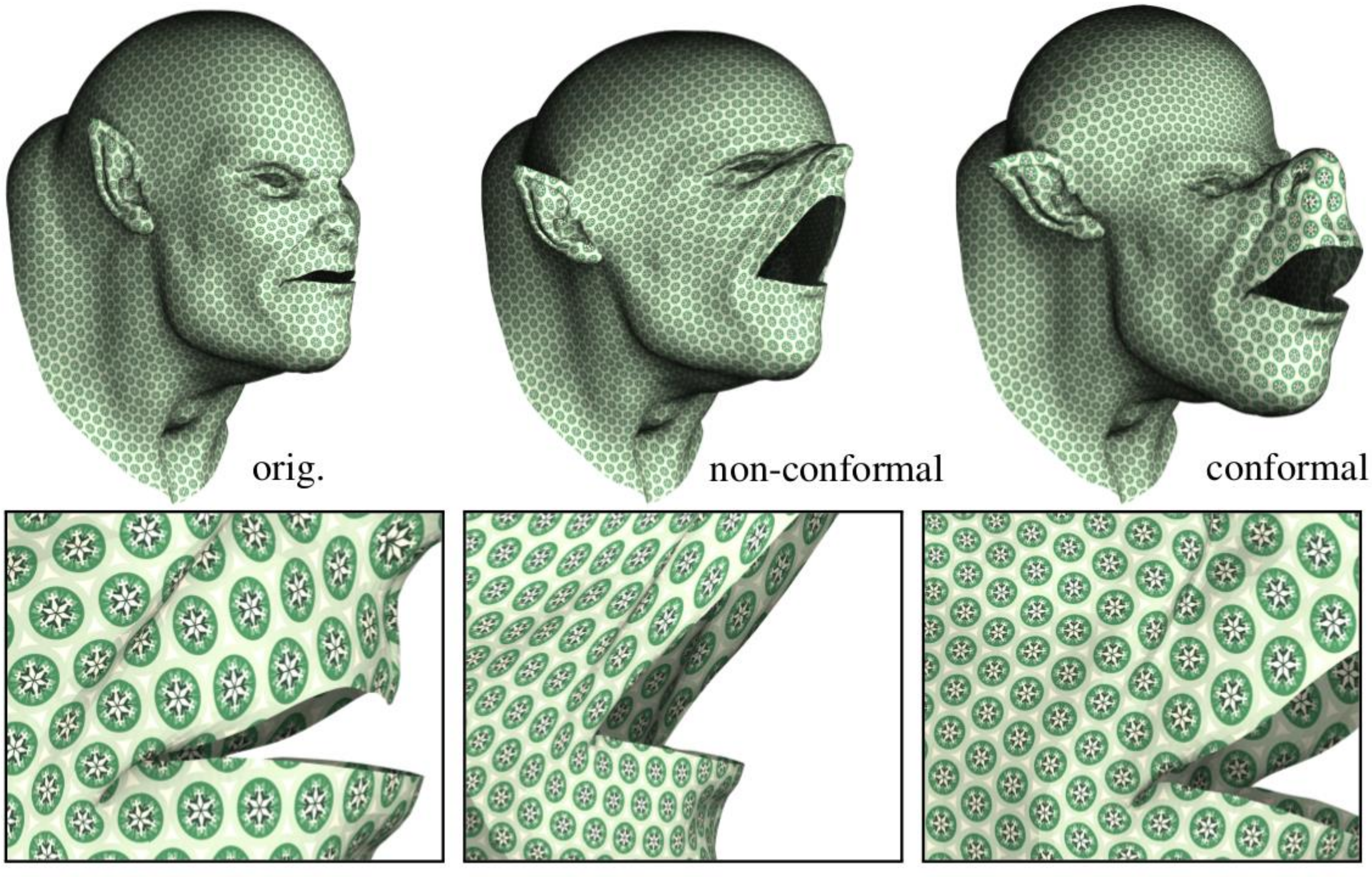
[Crane et al.] : Spin Transformations of Discrete Surfaces

Conformal 3d surface deformations



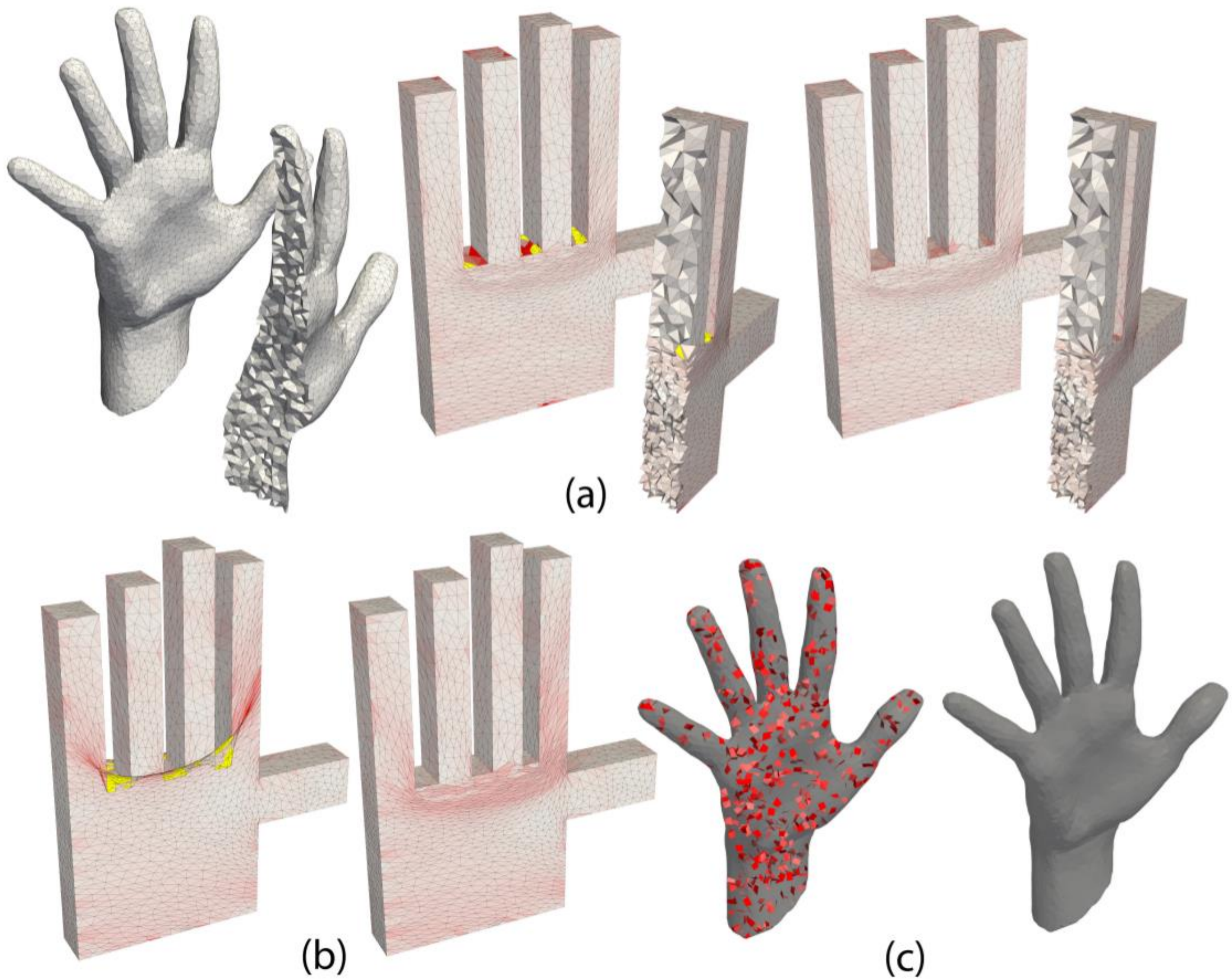
[Crane et al.] : Spin Transformations of Discrete Surfaces

Conformal 3d surface deformations



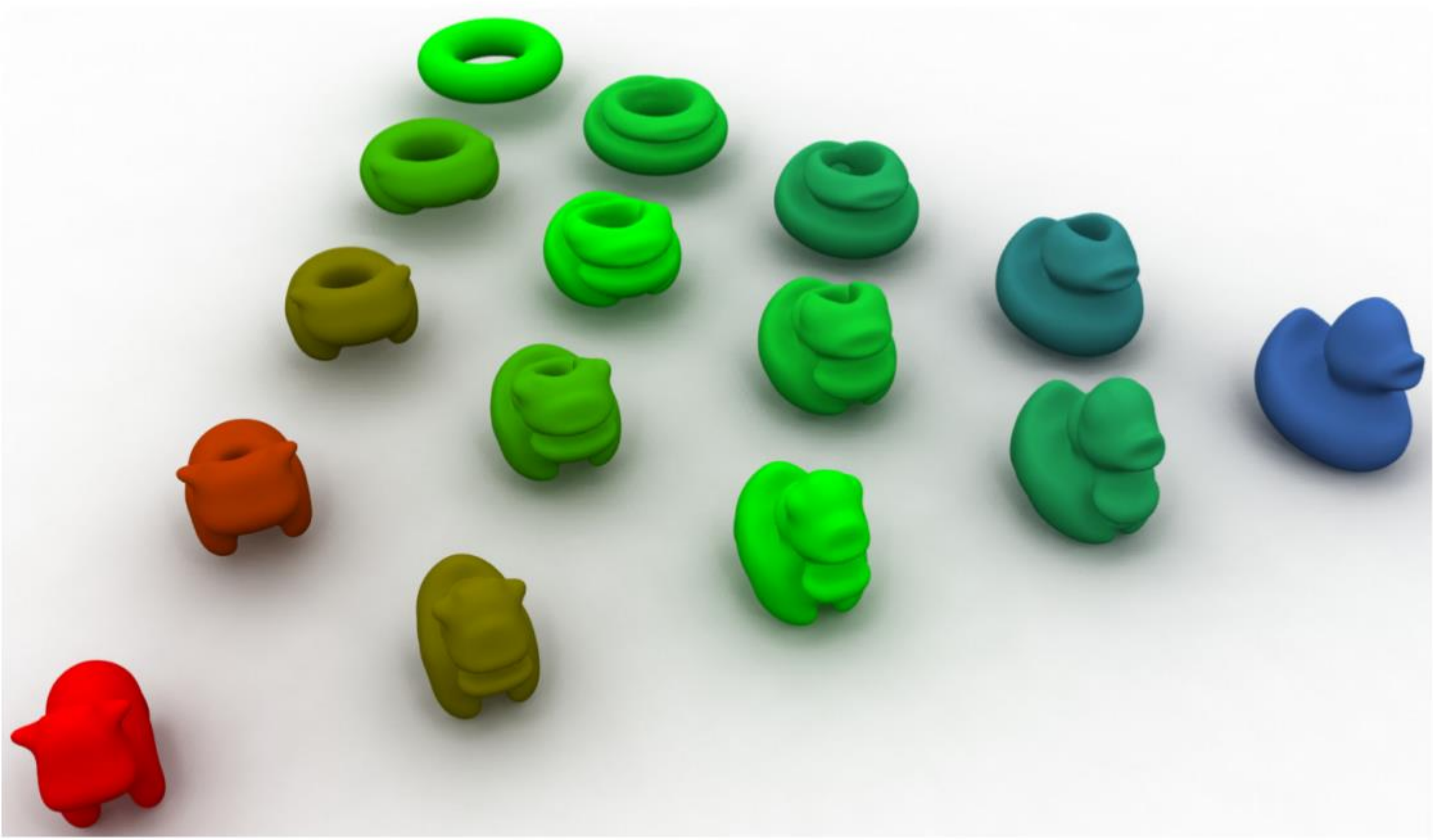
[Vaxman et al.] : Conformal Mesh Deformations with Möbius Transformations

Bounded 3d volume distortion



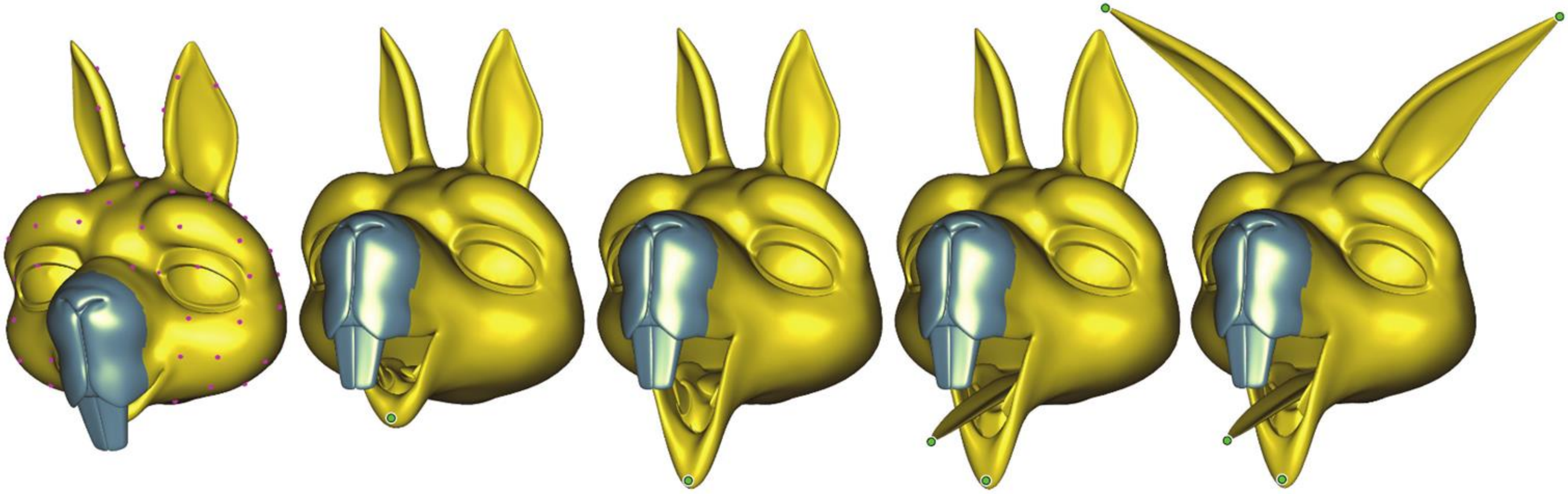
[Aigerman et al.] :Injective and Bounded Distortion Mappings in 3D

Shape interpolation using optimal transport



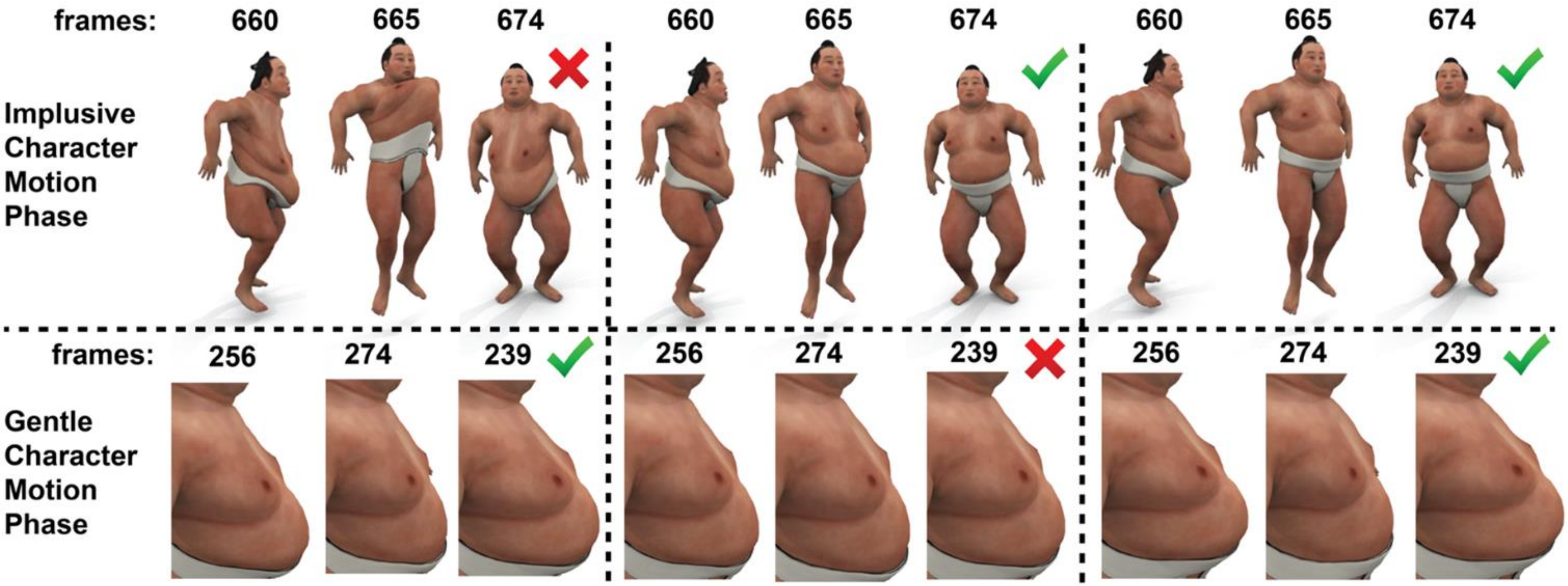
[Solomon et al.] :Convolutional Wasserstein Distances:
Efficient Optimal Transportation on Geometric Domains

Real-time ARAP



[Wang et al.] :Linear Subspace Design for Real-Time Shape Deformation

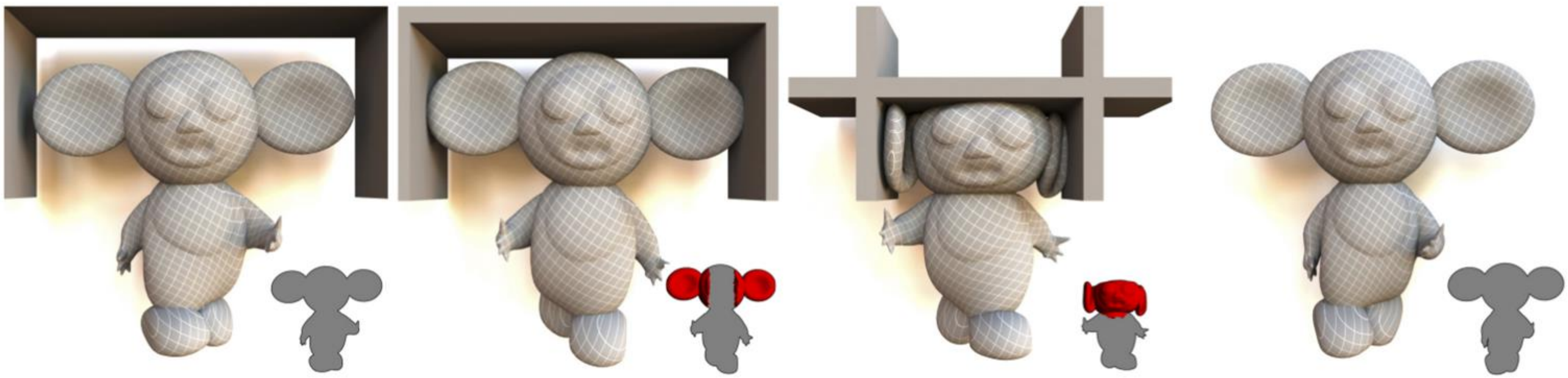
Material design



(a) Soft neo-Hookean (b) Stiff neo-Hookean (c) Our material

[Xu et al.] :Nonlinear Material Design Using Principal Stretches

Physically-correct animations



(a) (b) (c) (d)

[Teng et al.] :Subspace Condensation: Full Space Adaptivity for Subspace Deformations

Et beaucoup plus

Crédits

- Pierre Benard, Jean-Marc Thiery, Steve Marschner, Olivier Vaillancourt, Olivier Godin, Estelle Duveau, Marie-Paule Cani, Lionel Revert, François Faure, Ravi Ramamoorthi, Ronen Barzel, Bill Baxter…