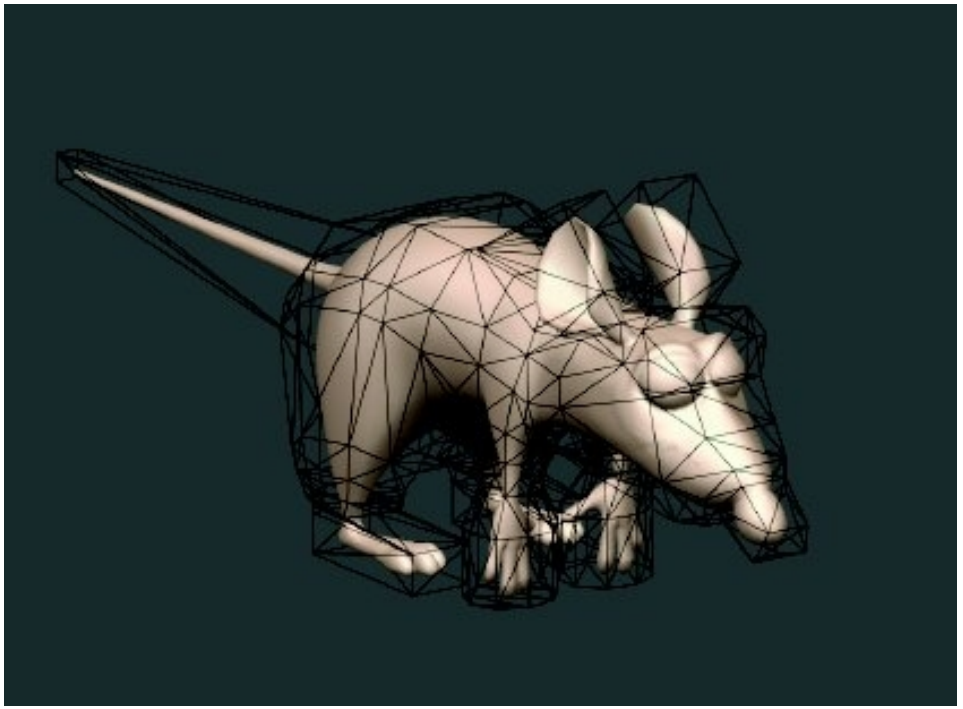
Cage-based deformations

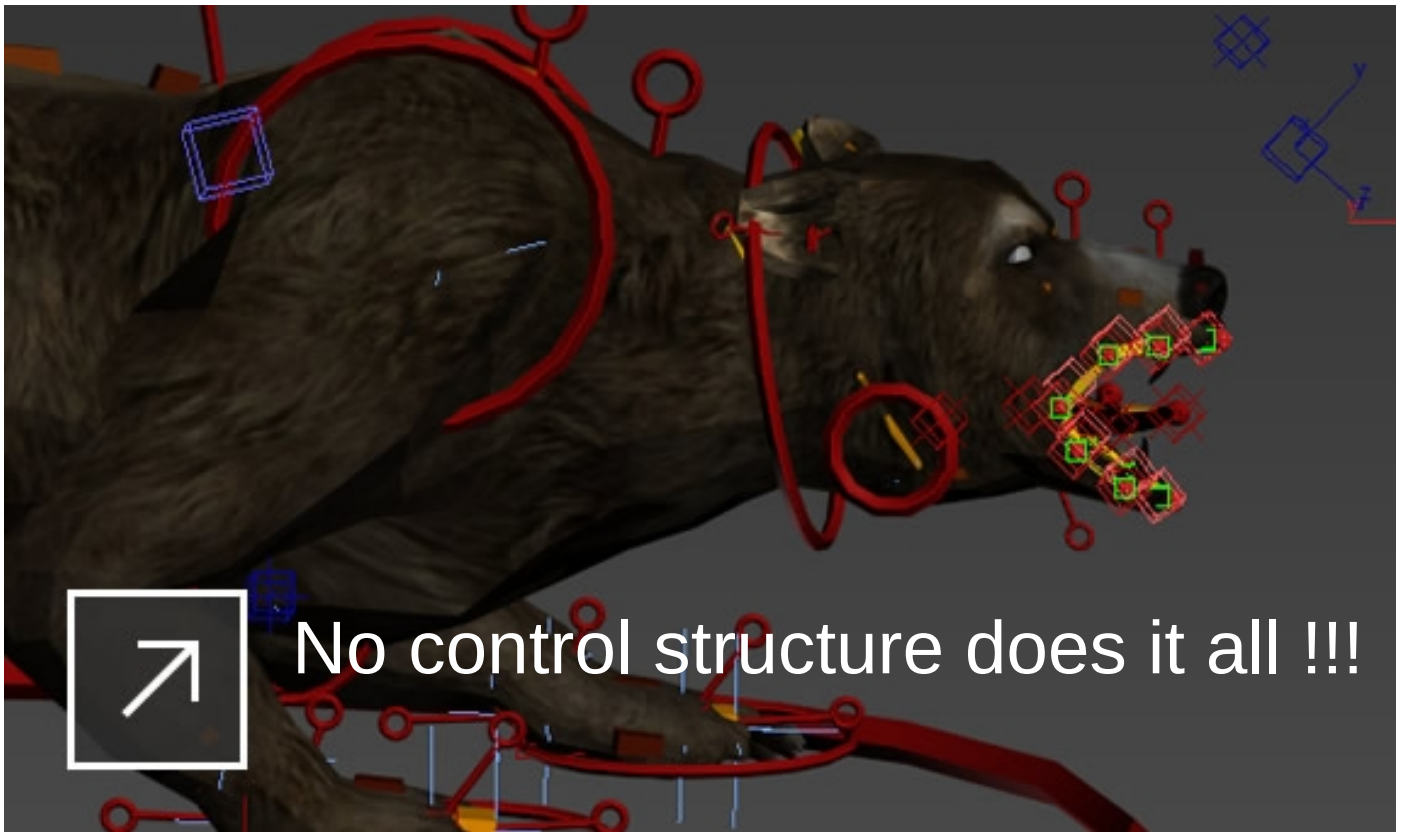


Where ?

Ratatouille (2007) and later movies from Disney & Pixar

What ?

- Animate bodies
  - Not fur
  - Not particles
  - Probably not facial animations

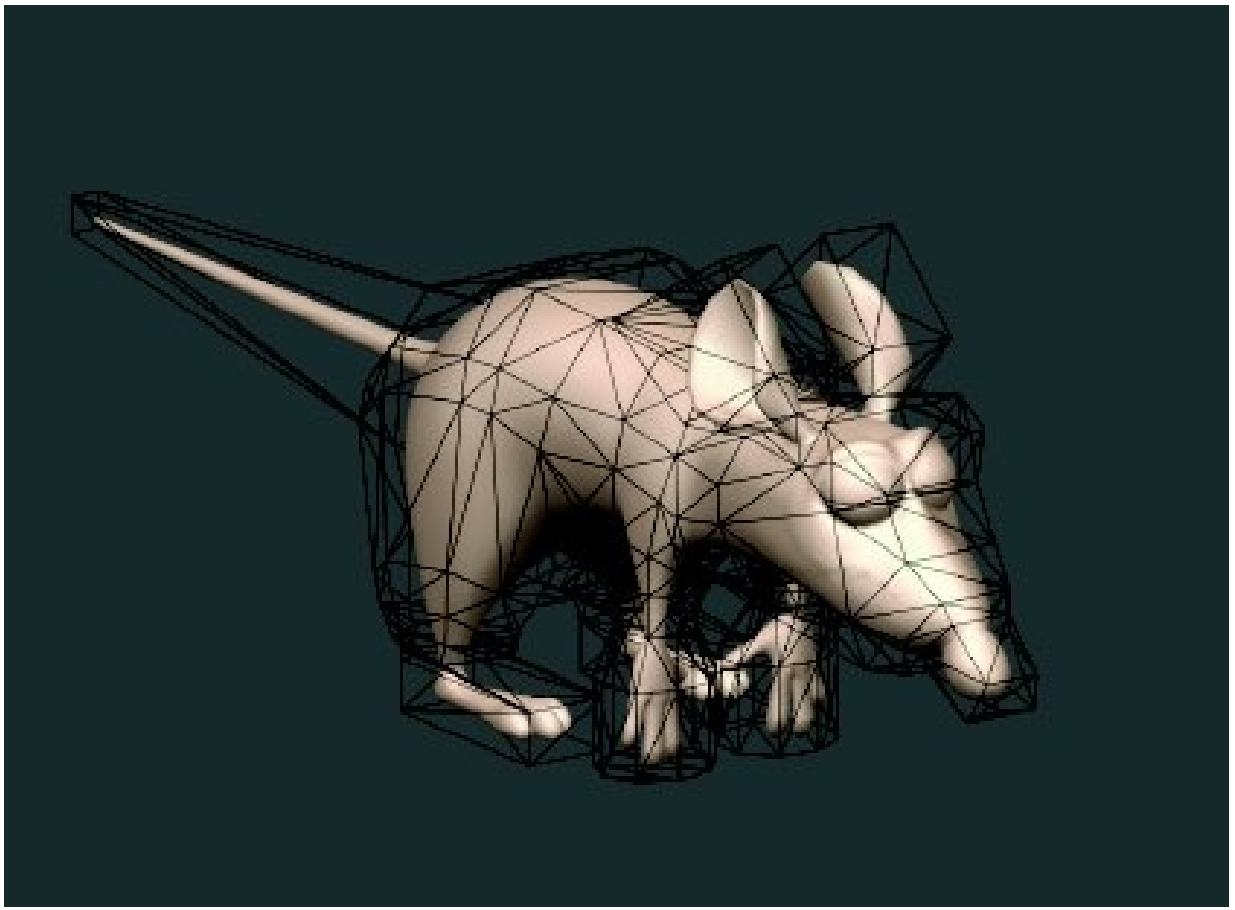


problematics

- What is a cage?
- How to define what is its transformation ?
- How to transfer its deformation to the mesh ?
- ...

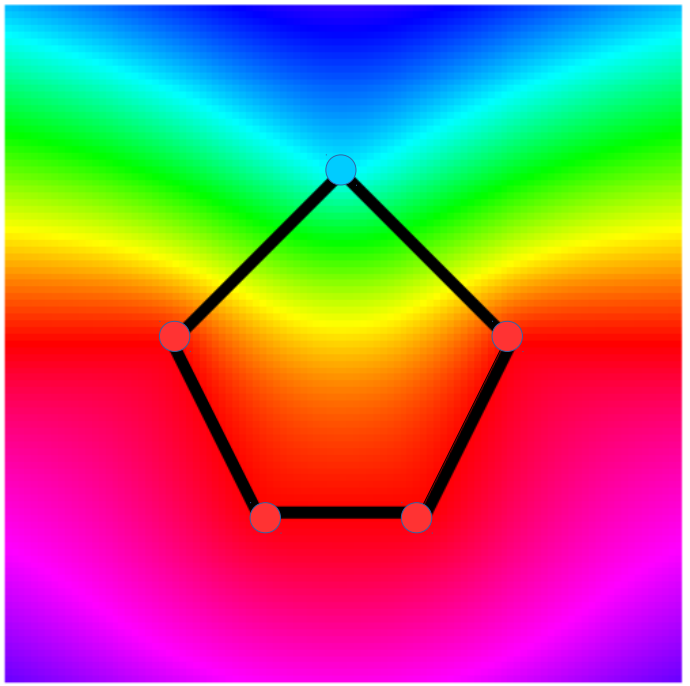
A cage

- Typically : a closed mesh englobing the shape



Used for « boundary interpolation »

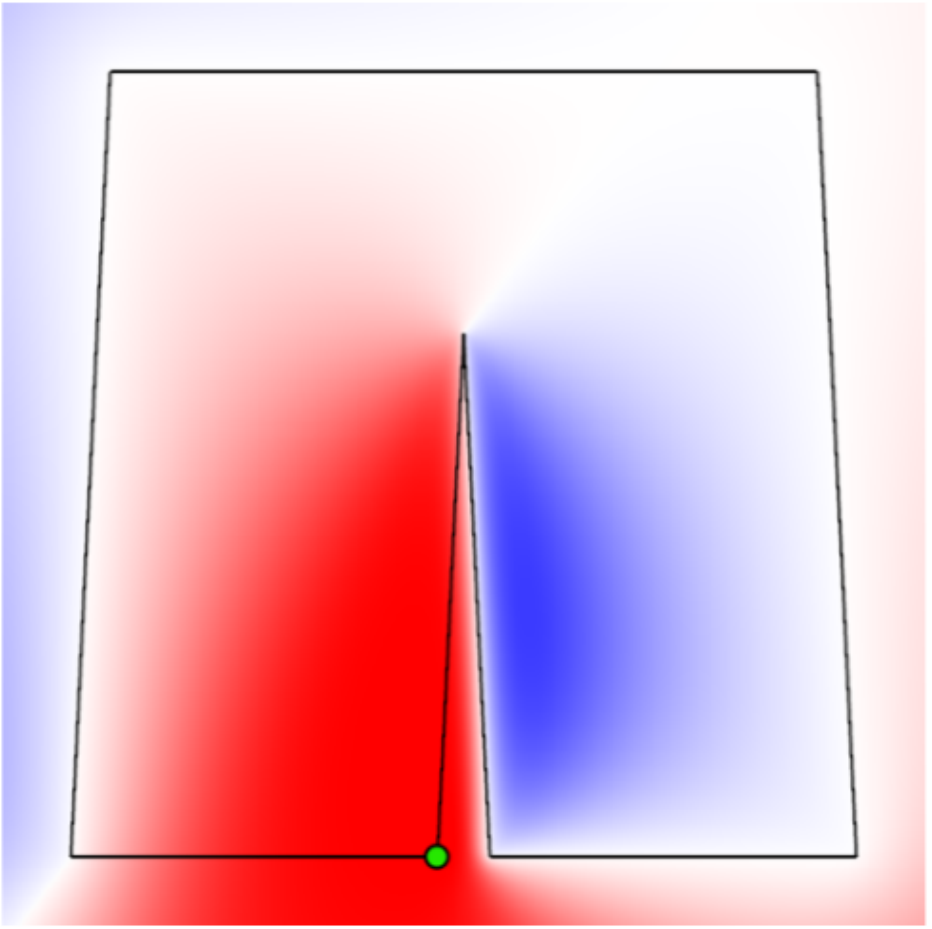
- A function f is given on the cage (the boundary)
- Extrapolate f inside the cage
- f = « new position » → deform space



Deformation

Demo time !

Cage coordinates



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

« Ideal » coordinates

- 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

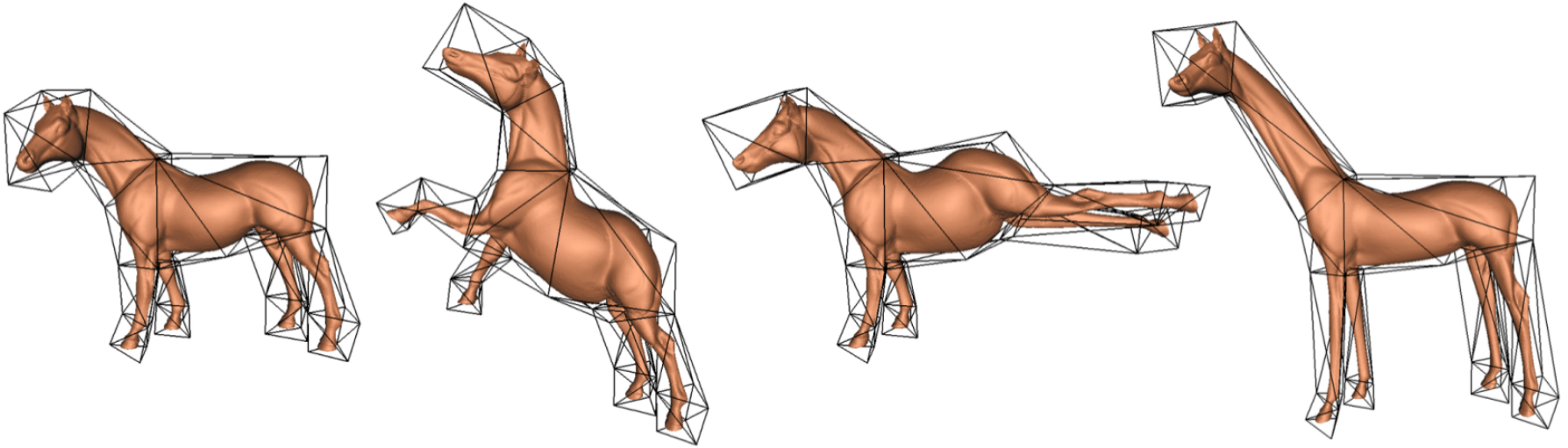
$$f(p) = \sum_i \phi_i(p) \cdot f_i$$

Various coordinates

- 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

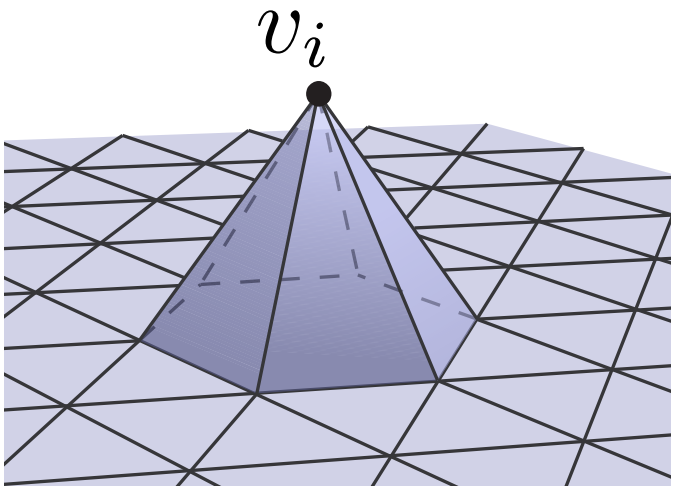
Mean-value coordinates

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

: « averaging » of values on the cage

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

f is linear per triangle on the cage



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

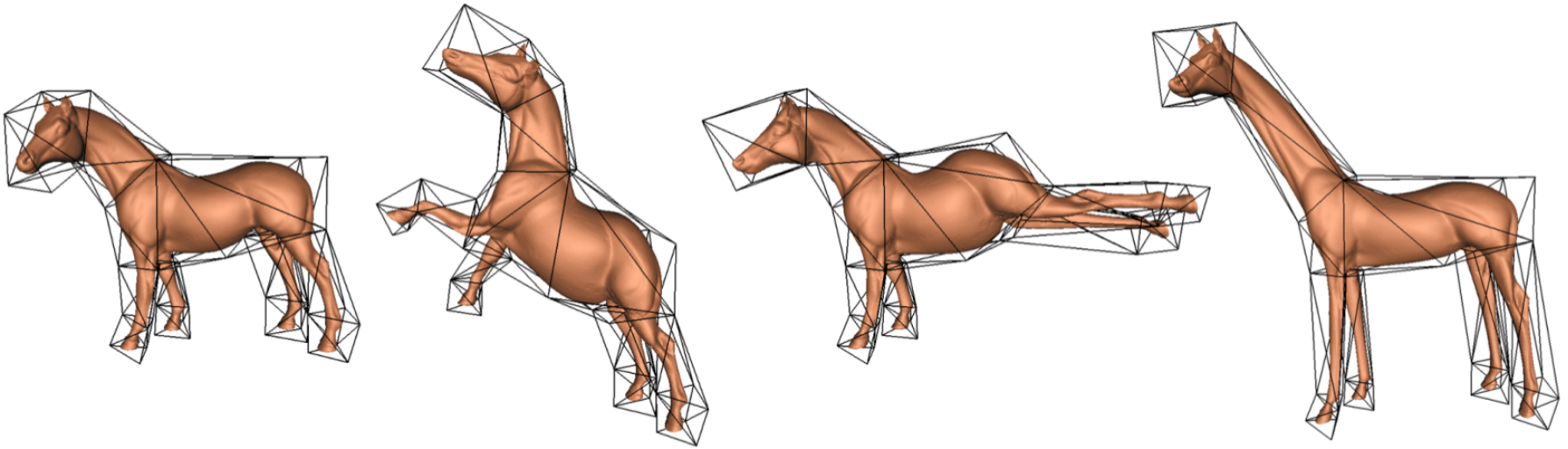
Let’s check on the board its properties :

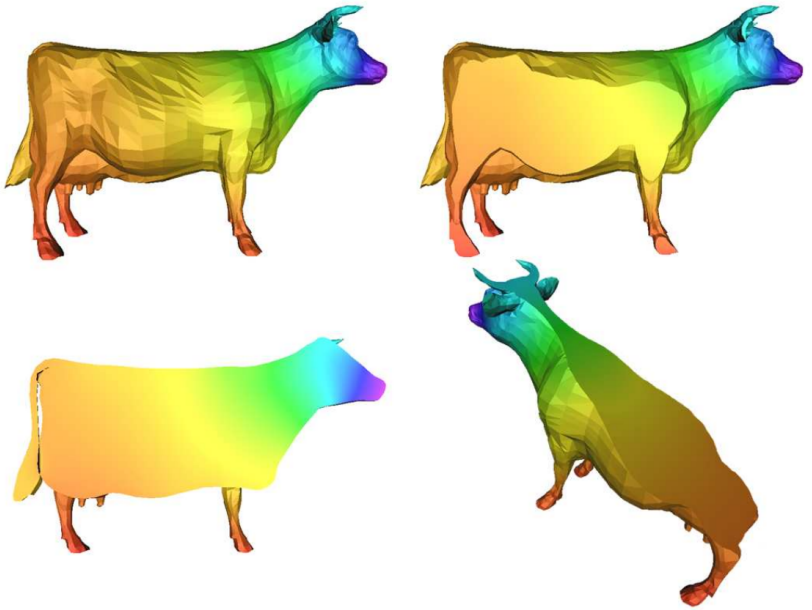
$$f^{MVC}(\eta) = \frac{\int_{B_\eta(M)} \frac{f(\xi)}{|\xi-\eta|} dS_\eta(\xi)}{\int_{B_\eta(M)} \frac{1}{|\xi-\eta|} dS_\eta(\xi)}$$

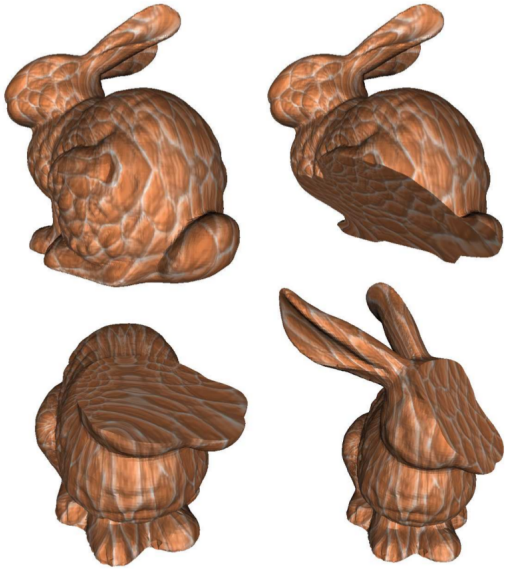
$$\lambda_i^{MVC}(\eta) = \frac{\int_{B_\eta(M)} \frac{\Gamma_i(\xi)}{|\xi-\eta|} dS_\eta(\xi)}{\int_{B_\eta(M)} \frac{1}{|\xi-\eta|} dS_\eta(\xi)}$$

- Smooth
- Sum to 1 for every vertex : $\sum_i \phi_i(p) = 1$
- Linear reproduction : $p = \sum_i \phi_i(p).c_i$
- Boundary interpolation

Mean-value coordinates







Mean-value coordinates

```cpp
// 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

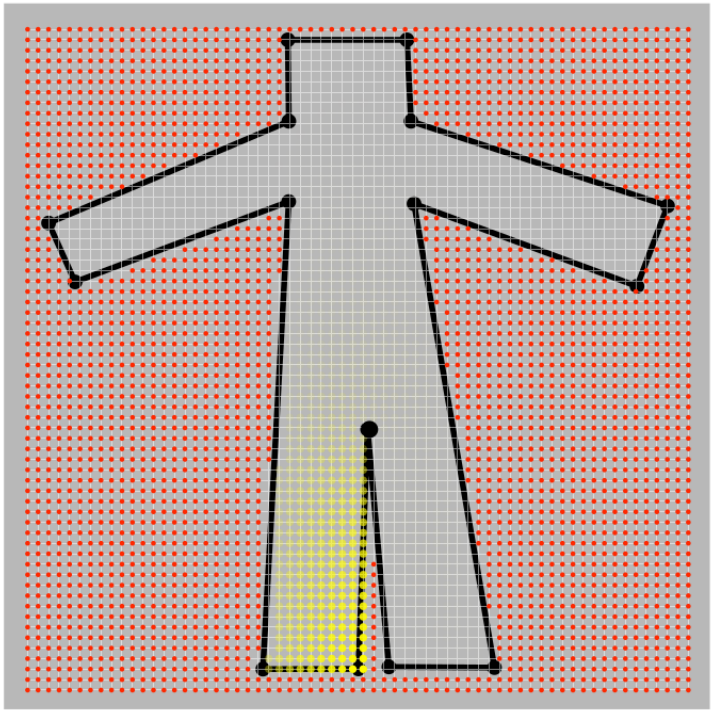
Mean-value coordinates

- Closed-form expression : +
- Defined everywhere in space : +
- Not always positive : -

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

Harmonic coordinates

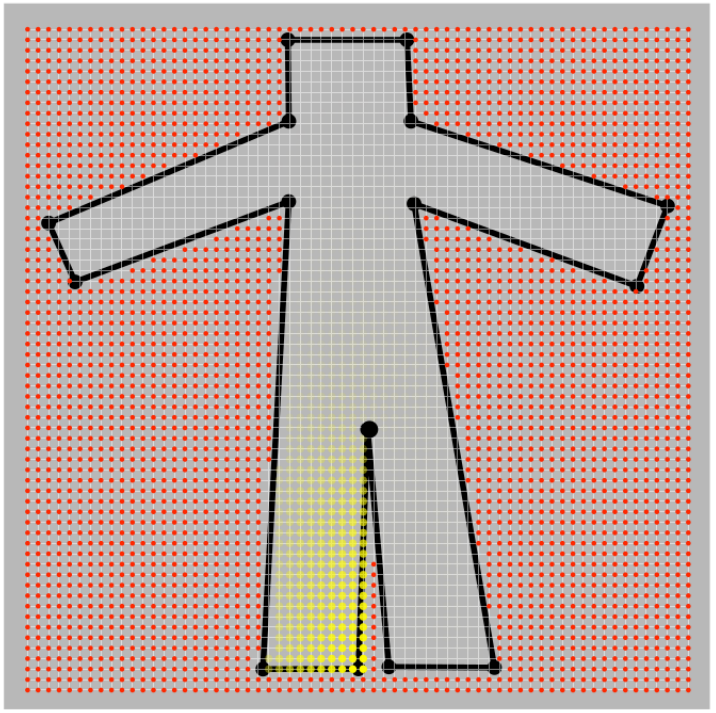
- Algo, for each cage vertex :
  - Setup discrete grid
  - Set boundary values
  - Solve for null Laplacian
    - Linear system
    - Iterative averaging
  - values at mesh vertices using grid bilinear interpolation



[DeRose & Meyer] : Harmonic Coordinates

Harmonic coordinates

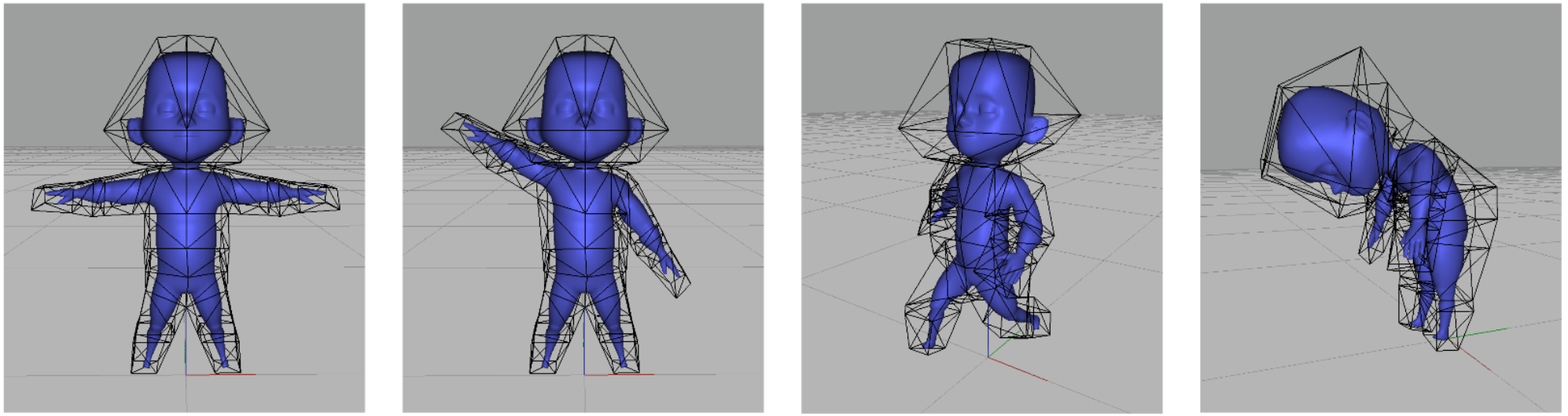
- No closed-form expression : -
- Defined inside only : -
- Always positive : +
- Kind of local : +



[DeRose & Meyer] : Harmonic Coordinates

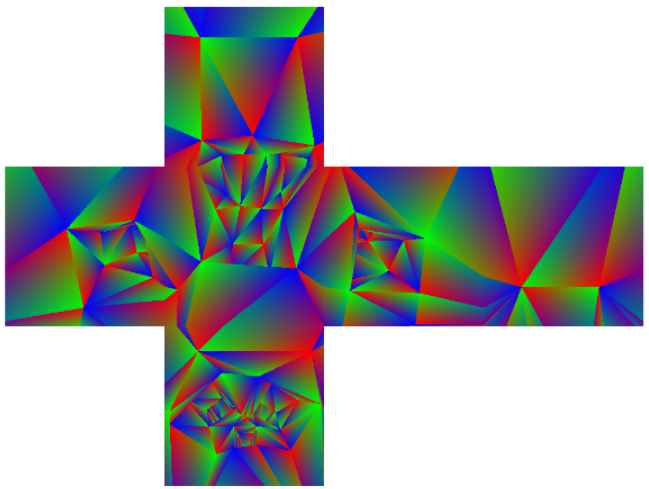
Harmonic coordinates

- You can find it :
  - At Pixar (someone told me they still use it)
  - In Blender

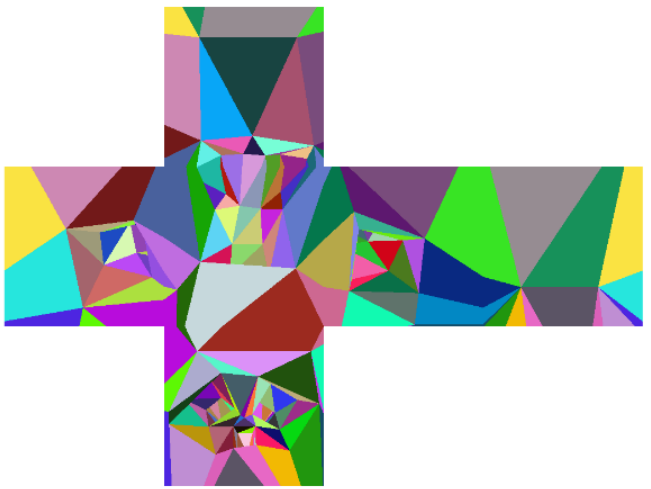


Positive mean-value coordinates

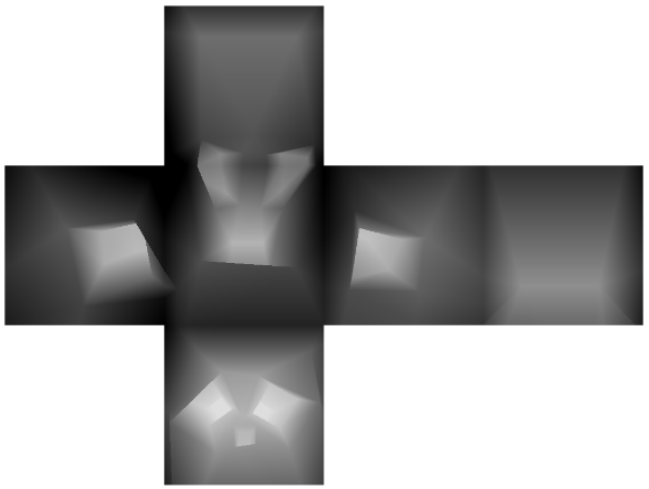
- Based on mean-value coordinates
- Use part of the cage that is visible from p only
  - positive
  - No closed-form expression
- Use GPU for approximation of spherical integral



Color Buffer



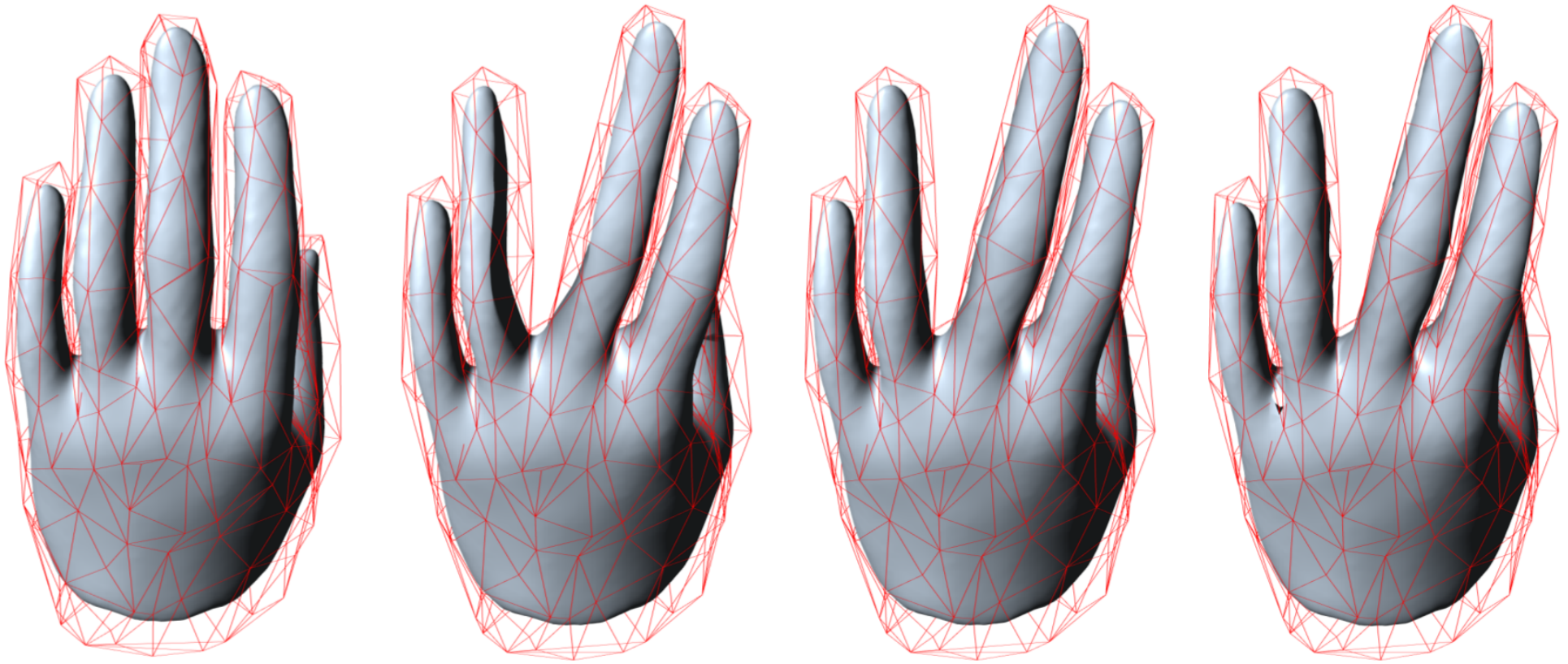
Index Buffer



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)

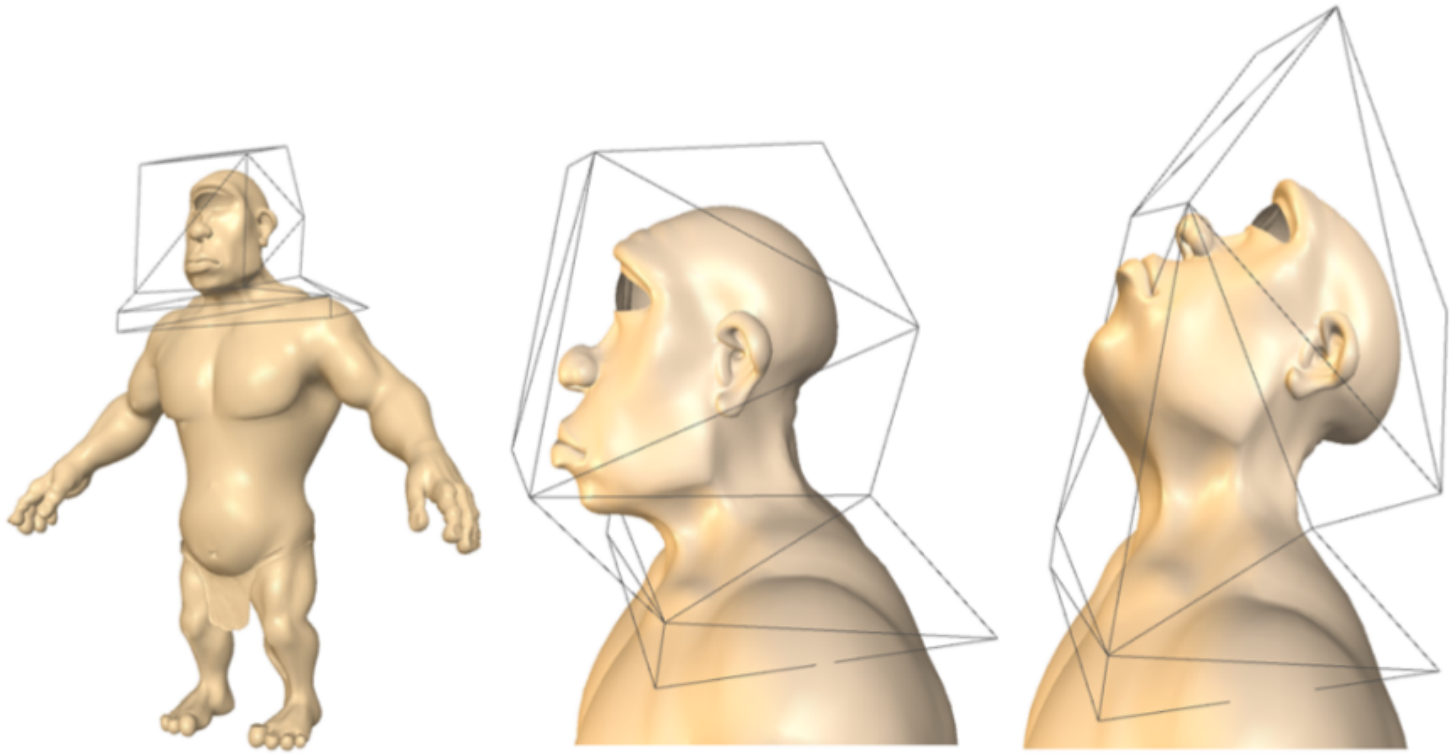
Positive mean-value coordinates

- No closed-form expression : -
- Defined inside only : -
- Always positive : +
- Kind of local : +

[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

Green coordinates

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

for harmonic functions (3rd identity of Green)

with D : the domain, ∂D : the boundary (i.e., the cage)

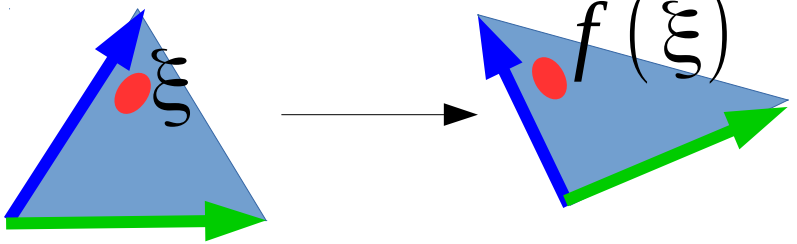
$$\Delta_1 G(x, y) = \Delta_2 G(x, y) = \delta_0(||x - y||)$$

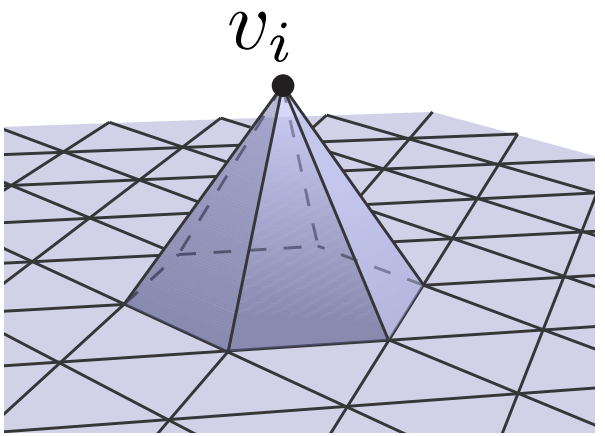
$$G(x, y) = \begin{cases} \frac{1}{(2-d)|S^d|} ||x - y||^{2-d} & d \geq 3 \\ \frac{1}{2\pi} \log(||x - y||) & d = 2 \end{cases}$$

Green coordinates

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

$$\forall \xi \in \partial D \ : \ f(\xi) = \sum_{v_i} \Gamma_i(\xi) \cdot v_i$$



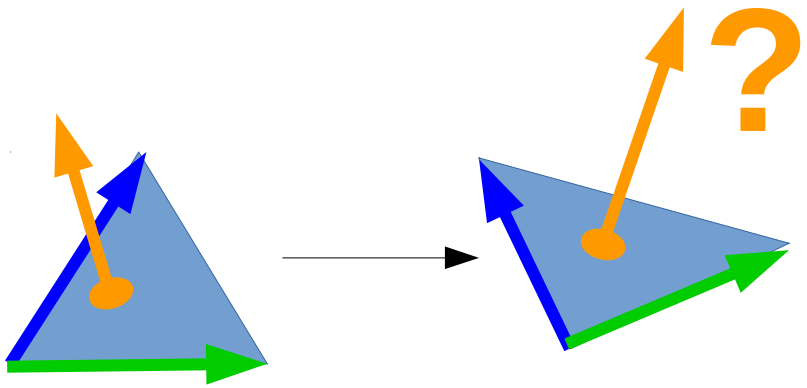


Deformation of the cage (linear on each triangle)

Green coordinates

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

Along the normal :



$$\forall \xi \in t_j \subset \partial D \ : \ \frac{\partial f}{\partial n_\xi}(\xi) = s_{t_j} \cdot n(t_j)$$

$$s_j = \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}}$$

Heuristic : What's the meaning of this ?

Green coordinates

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

$$\forall \xi \in \partial D \ : \ f(\xi) = \sum_{v_i} \Gamma_i(\xi) \cdot v_i$$

Dirichlet condition

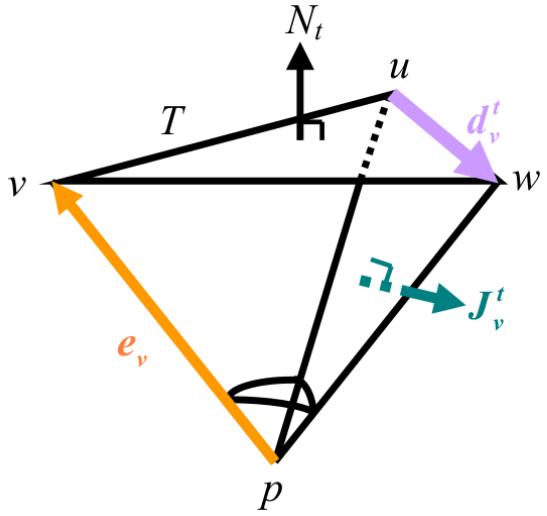
$$\forall \xi \in t_j \subset \partial D \ : \ \frac{\partial f}{\partial n_\xi}(\xi) = s_{t_j} \cdot n(t_j)$$

Neumann condition

$$f(\eta) = \sum_{v_i} \phi_i(\eta) \cdot v_i + \sum_{t_j} \psi_j(\eta) s_{t_j} \cdot n(t_j)$$

Green coordinates

Simple closed-form expressions in the appendix of :
[Benchen et al.] : Variational Harmonic Maps for Space Deformation



$$\tilde{e}_v = \frac{e_v}{\|e_v\|}$$

$$R_v^t = \|e_w\| + \|e_u\|$$

$$C_v^t = \frac{1}{4\pi\|d_v^t\|}\log\left(\frac{R_v^t + \|d_v^t\|}{R_v^t - \|d_v^t\|}\right)$$

$$\tilde{C}_v^t = \frac{1}{2\pi\left(R_v^t + \|d_v^t\|\right)\left(R_v^t - \|d_v^t\|\right)}$$

$$P_t = N_t \times \sum_{i\in t} d_i^t C_i^t - \omega_t N_t$$

$$\nabla\omega_t = \sum_{i=1}^{3} J_{v_i}^t \tilde{C}_{v_i}^t \left(\|e_{v_{i+1}}\|^{-1} + \|e_{v_{i+2}}\|^{-1}\right)$$

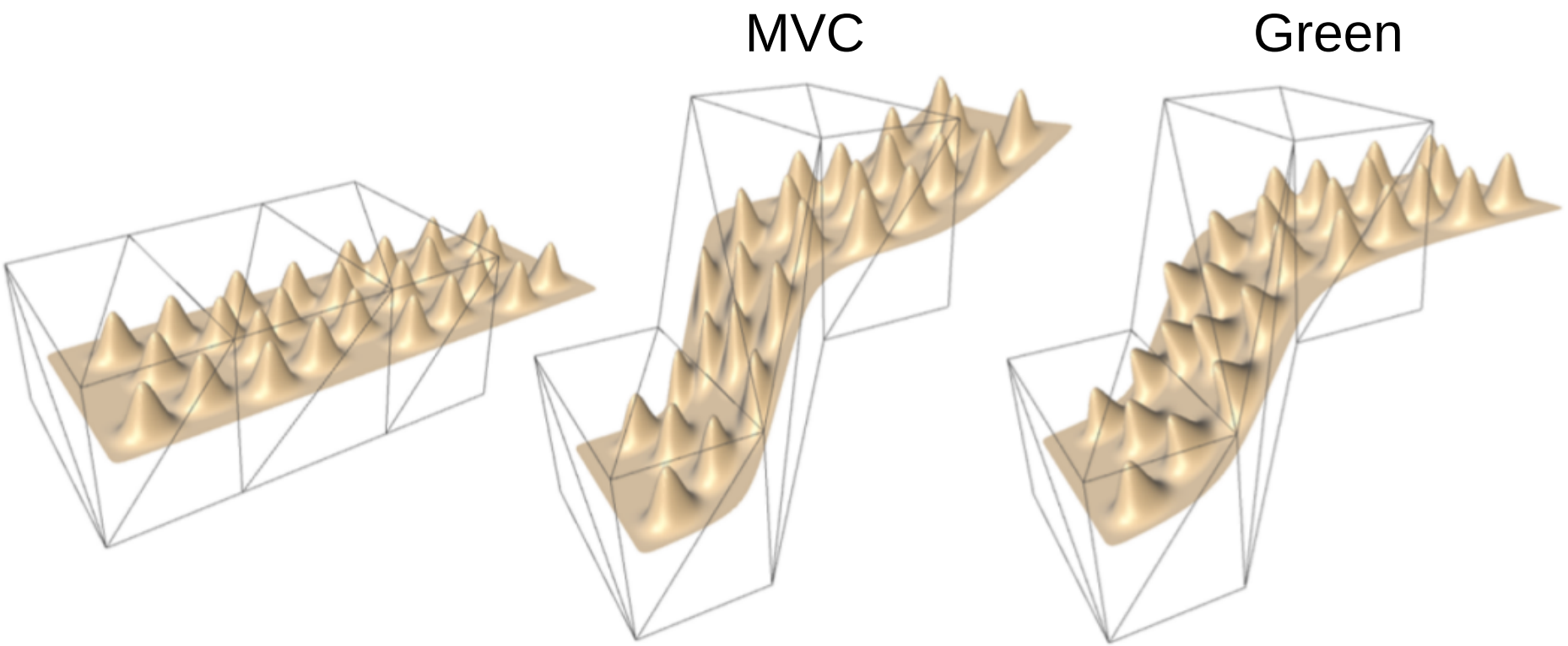
$$J\left(P_{t=(u,v,w)}\right) = \begin{pmatrix} (\tilde{e}_v + \tilde{e}_w)\tilde{C}_u^t \\ (\tilde{e}_u + \tilde{e}_w)\tilde{C}_v^t \\ (\tilde{e}_u + \tilde{e}_v)\tilde{C}_w^t \end{pmatrix}_{3x3}^{T} \begin{pmatrix} d_u^t \\ d_v^t \\ d_w^t \end{pmatrix}_{3x3} [N_t]_\times^T + \nabla\omega_t^T N_t$$

$$\psi_t = -\sum_{i\in t} C_i^t \left(J_i^t \cdot N_t\right) - \frac{3}{A_t}\omega_t vol_t$$

$$\varphi_v = \sum_{t\in N(v)} \frac{1}{2A_t} P_t \cdot J_v^t$$

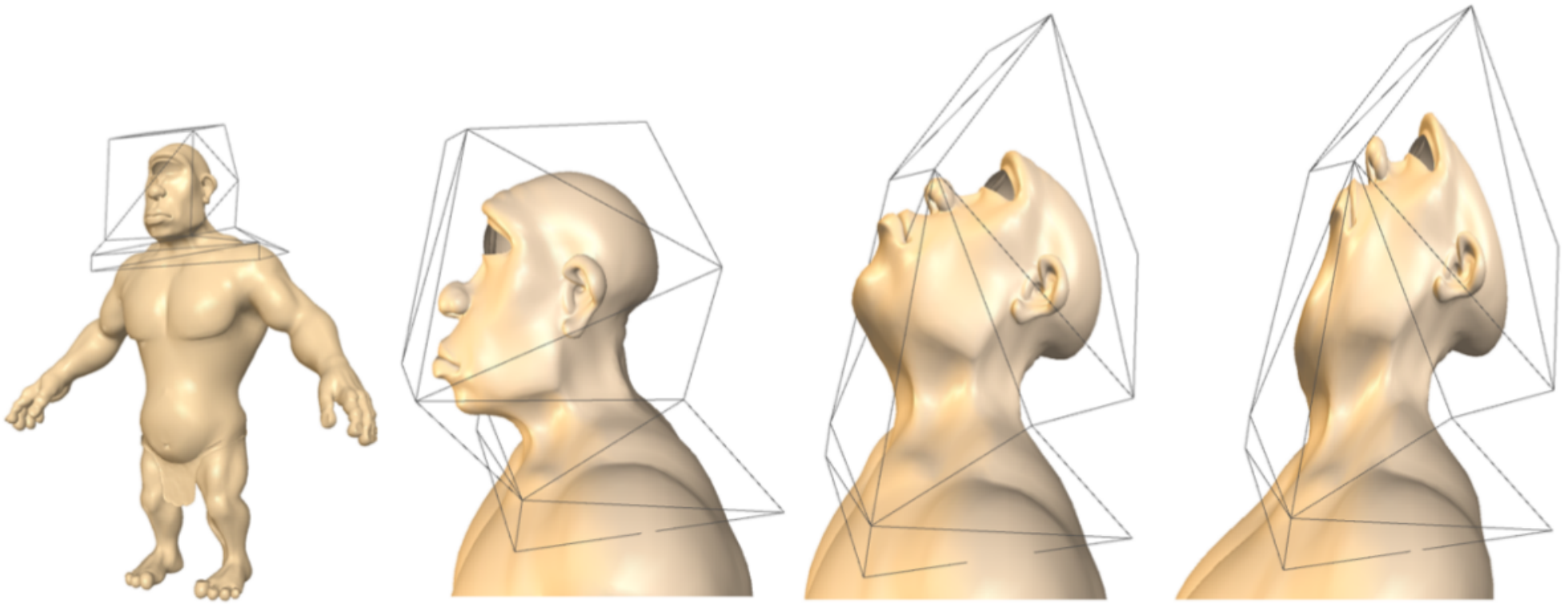
$$f(\eta) = \sum_{v_i} \phi_i(\eta) \cdot v_i + \sum_{t_j} \psi_j(\eta) \cdot t_j \cdot n(t_j)$$

Green coordinates



$$f(\eta) = \sum_{v_i} \phi_i(\eta) \cdot v_i + \sum_{t_j} \psi_j(\eta) s_{t_j} \cdot n(t_j)$$

Green coordinates



Green MVC

Green coordinates

- Closed-form expression : +
- Defined inside only : -
- Not always positive, but produces expected results : +
- Smooth (harmonic) : +
- Quasi-conformal : +
- Uses normals to infer rotations from cage vertex translations : +

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);
    }
}
```

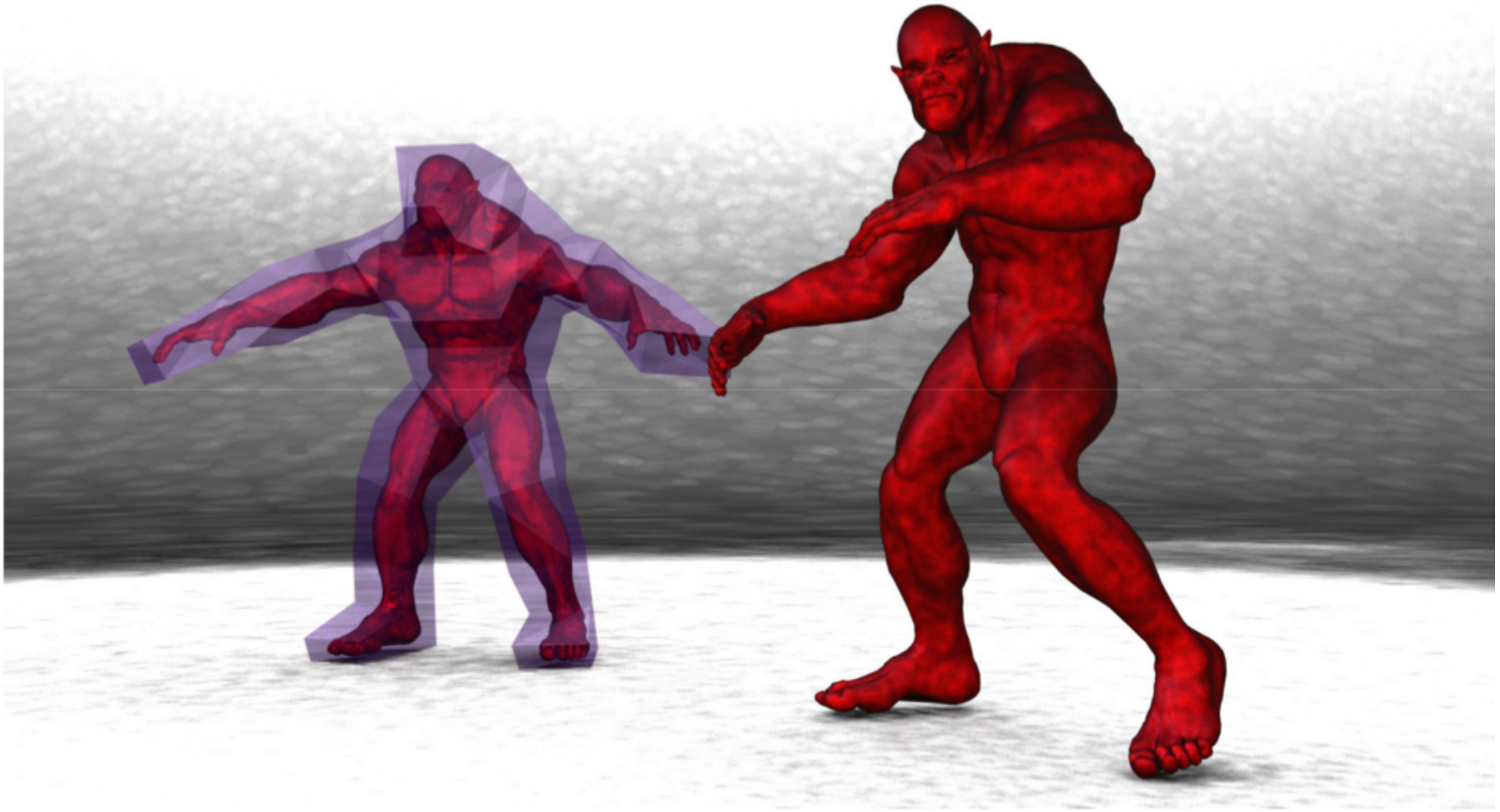
Roughly 50 lines of code

Conclusions

- Use it for volumetric deformations
- You have guarantees on the spatial function (does not rely on the discretization of the deformed mesh)
- Plenty of schemes
- Plenty of open problems

Variational methods

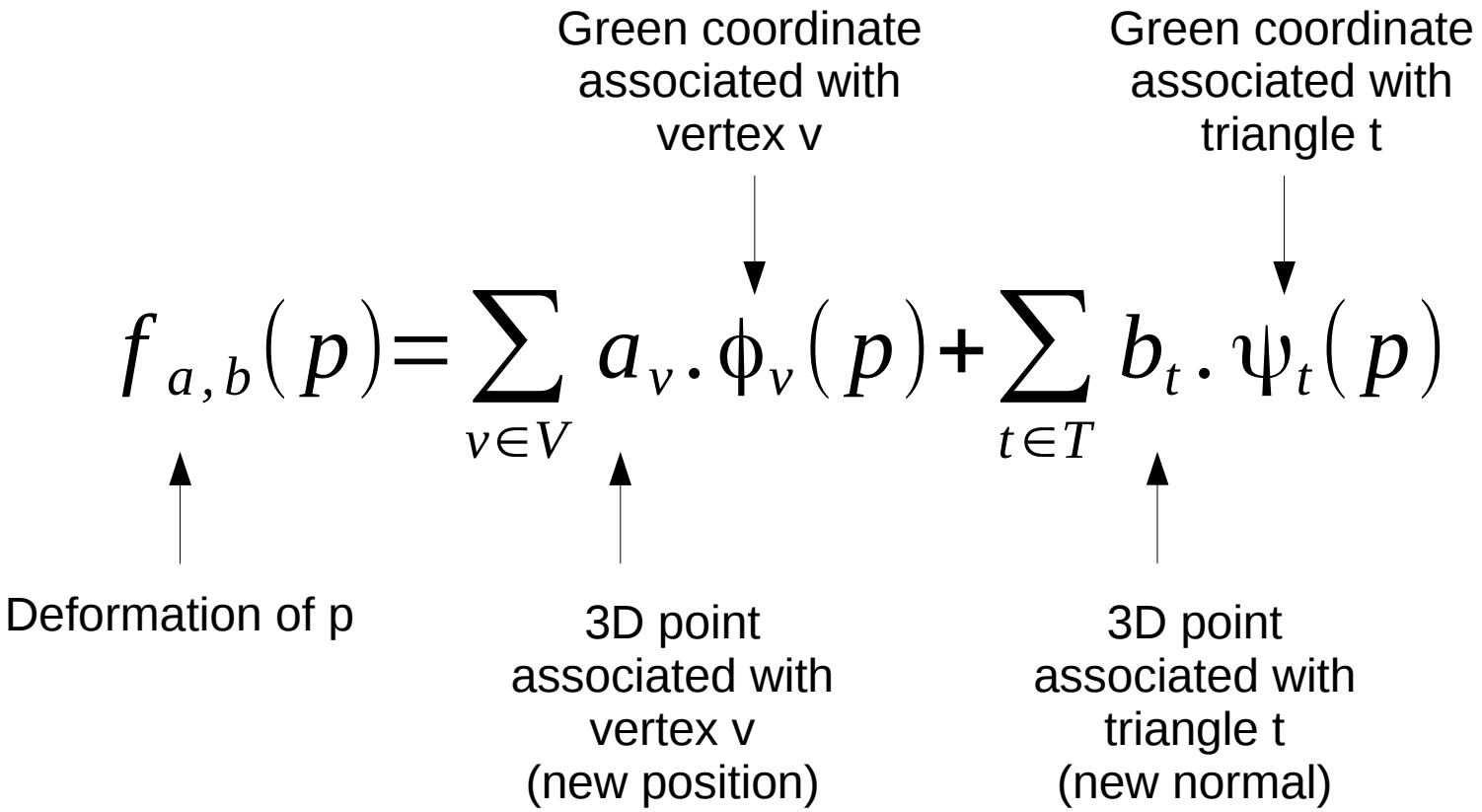
Variational harmonic maps



[Benchen et al.] : Variational Harmonic Maps for Space Deformation

Main idea

- Use a cage whose parameters define an harmonic deformation space of dimension 3(|V|+|T|) :

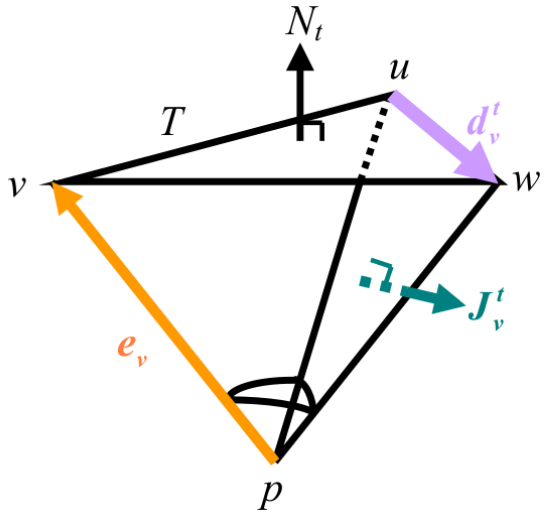


Function and derivatives can be computed everywhere inside the cage

	Matrix form
function $f_{a,b}(p)=\sum_{v\in V} a_v.\phi_v(p)+\sum_{t\in T} b_t.\psi_t(p)$	$\left(f_{\boldsymbol{a},\boldsymbol{b}}(p)\right)_{1\times 3}=\begin{pmatrix}\boldsymbol{\varphi}_{1\times n} & \boldsymbol{\psi}_{1\times m}\end{pmatrix}\begin{pmatrix}\boldsymbol{a}_{n\times 3}\\ \boldsymbol{b}_{m\times 3}\end{pmatrix}$
Jacobian $Jf(p)=\sum_{v\in V} a_v.\vec{\nabla}\phi_v(p)^T+\sum_{t\in T} b_t.\vec{\nabla}\psi_t(p)^T$	$\left(\boldsymbol{J}_f(p)\right)^T_{3\times 3}=\begin{pmatrix}\left(\boldsymbol{G}_{\varphi}\right)_{3\times n} & \left(\boldsymbol{G}_{\psi}\right)_{3\times m}\end{pmatrix}\begin{pmatrix}\boldsymbol{a}_{n\times 3}\\ \boldsymbol{b}_{m\times 3}\end{pmatrix}$
Hessian $H\begin{pmatrix}x\\y\\z\end{pmatrix}(p)=\sum_{v\in V}\begin{pmatrix}x\\y\\z\end{pmatrix}(a_v).H\phi_v(p)+\sum_{t\in T}\begin{pmatrix}x\\y\\z\end{pmatrix}(b_t).H\psi_t(p)$	$\left(\boldsymbol{H}_f(p)\right)_{5\times 3}=\begin{pmatrix}\left(\boldsymbol{H}_{\varphi}\right)_{5\times n} & \left(\boldsymbol{H}_{\psi}\right)_{5\times m}\end{pmatrix}\begin{pmatrix}\boldsymbol{a}_{n\times 3}\\ \boldsymbol{b}_{m\times 3}\end{pmatrix}$

Note : Each coordinate Hessian (3x3 sym matrix) is written as a column of 5 entries. Why 5 ? (and not 9 ?)

Closed-form expressions



$$\tilde{e}_v = \frac{e_v}{\|e_v\|}$$

$$R_v^t = \|e_w\| + \|e_u\|$$

$$C_v^t = \frac{1}{4\pi\|d_v^t\|}\log\left(\frac{R_v^t + \|d_v^t\|}{R_v^t - \|d_v^t\|}\right)$$

$$\tilde{C}_v^t = \frac{1}{2\pi\left(R_v^t + \|d_v^t\|\right)\left(R_v^t - \|d_v^t\|\right)}$$

$$P_t = N_t \times \sum_{i\in t} d_i^t C_i^t - \omega_t N_t$$

$$\nabla\omega_t = \sum_{i=1}^{3} J_{v_i}^t \tilde{C}_{v_i}^t \left(\|e_{v_{i+1}}\|^{-1} + \|e_{v_{i+2}}\|^{-1}\right)$$

$$J\left(P_{t=(u,v,w)}\right) = \begin{pmatrix} (\tilde{e}_v + \tilde{e}_w)\tilde{C}_u^t \\ (\tilde{e}_u + \tilde{e}_w)\tilde{C}_v^t \\ (\tilde{e}_u + \tilde{e}_v)\tilde{C}_w^t \end{pmatrix}_{3x3}^{T} \begin{pmatrix} d_u^t \\ d_v^t \\ d_w^t \end{pmatrix}_{3x3} [N_t]_\times^T + \nabla\omega_t^T N_t$$

$$\psi_t = -\sum_{i\in t} C_i^t \left(J_i^t \cdot N_t\right) - \frac{3}{A_t}\omega_t vol_t$$

$$\nabla\psi_t = -P_t$$

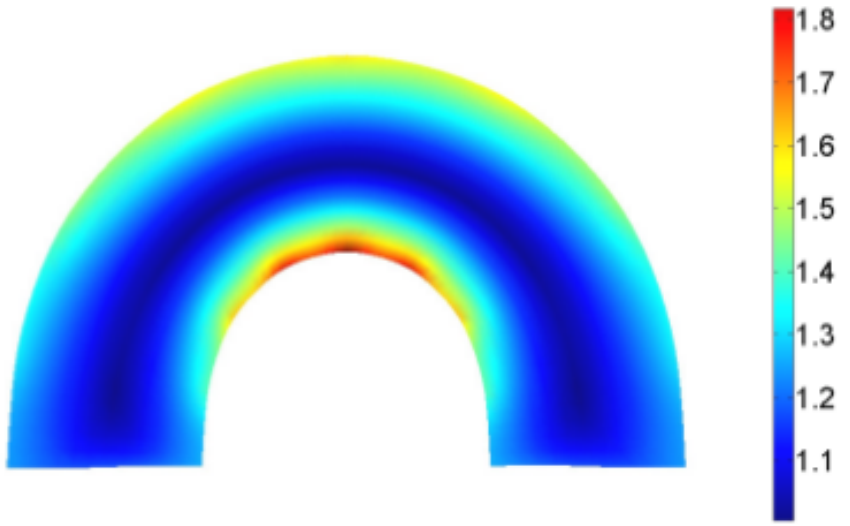
$$H(\psi_t) = -J\left(P_t\right)$$

$$\varphi_v = \sum_{t\in N(v)} \frac{1}{2A_t} P_t \cdot J_v^t$$

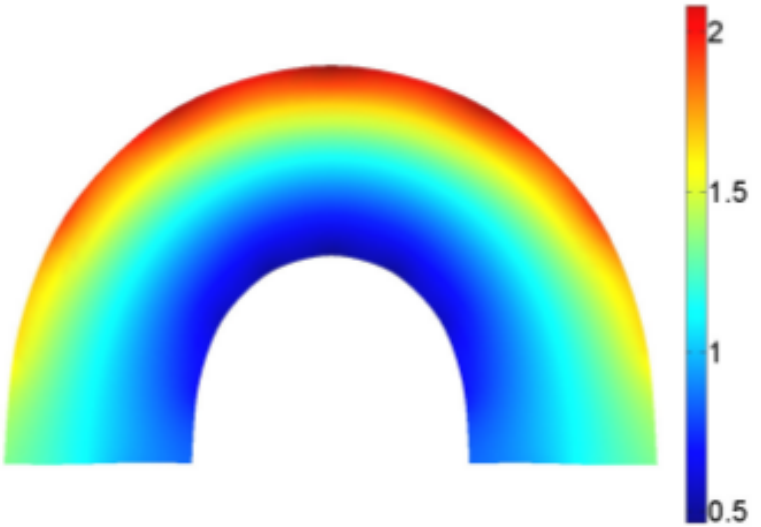
$$\nabla\varphi_v = \sum_{t\in N(v)} \frac{1}{2A_t} P_t \times d_v^t$$

$$H(\varphi_v) = -\sum_{t\in N(v)} \frac{1}{2A_t}\left[d_v^t\right]_\times J\left(P_t\right)$$

Assumption on ARAP deformations



Condition number

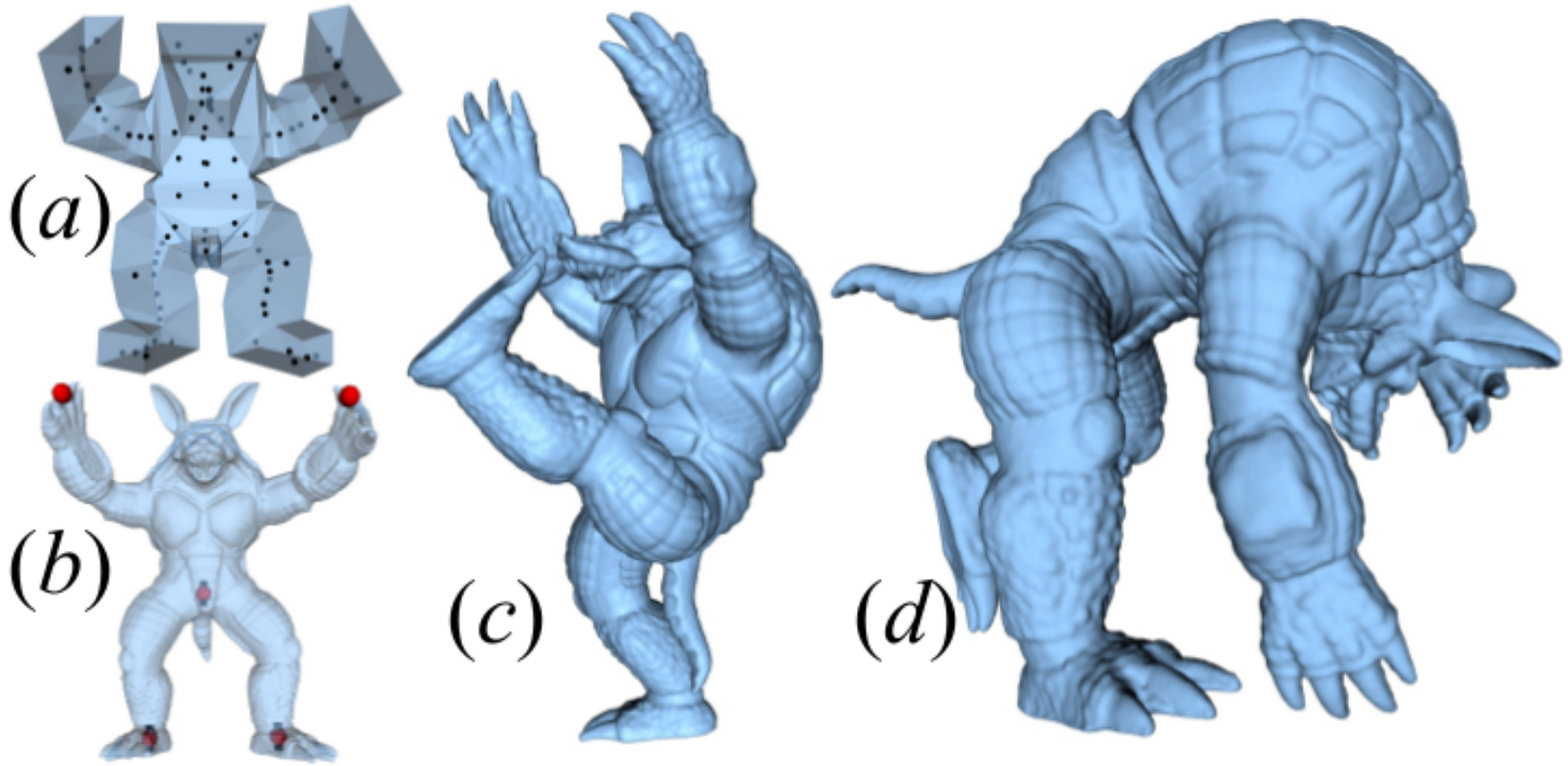


Determinant

« Volumetric ARAP deformations should involve rotations on the medial axis »

Variational harmonic maps

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



Variational harmonic maps

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

$$\sum_{i \in PC} \left\| \sum_v \phi_v(h_i).a_v + \sum_t \psi_t(h_i).b_t - \bar{h}_i \right\|^2$$

$$+ \quad \sum_{i \in RC} \left\| \sum_v \nabla \phi_v(h_i).a_v^T + \sum_t \nabla \psi_t(h_i).b_t^T - \bar{R}_i^T \right\|^2$$

Variational harmonic maps

- Let’s setup some functional together on the board
  - Positional constraints (red points)
  - Rigidity constraints (blue points near the medial axis)
  - Smoothness constraints (near the cage). What is the effect of having a small second derivative ?

$$\sum_{i \in PC} \left\| \sum_v \phi_v(h_i).a_v + \sum_t \psi_t(h_i).b_t - \bar{h}_i \right\|^2$$

$$+ \quad \sum_{i \in RC} \left\| \sum_v \nabla \phi_v(h_i).a_v^T + \sum_t \nabla \psi_t(h_i).b_t^T - \bar{R}_i^T \right\|^2$$

$$+ \quad \sum_{i \in RC2} \left\| \sum_v H \phi_v(h_i).xa_v + \sum_t H \psi_t(h_i).xb_t \right\|^2$$

$$+ \quad \sum_{i \in RC2} \left\| \sum_v H \phi_v(h_i).ya_v + \sum_t H \psi_t(h_i).yb_t \right\|^2$$

$$+ \quad \sum_{i \in RC2} \left\| \sum_v H \phi_v(h_i).za_v + \sum_t H \psi_t(h_i).zb_t \right\|^2$$

Variational harmonic maps

- Algo :
  - Init Ri (ex : Identity)
  - Solve linear system
  - Update Ri (closest rotation matrix, remember last week, using SVD)
  - Solve linear system...

$$\sum_{i \in PC} \left\| \sum_v \phi_v(h_i).a_v + \sum_t \psi_t(h_i).b_t - \bar{h}_i \right\|^2$$

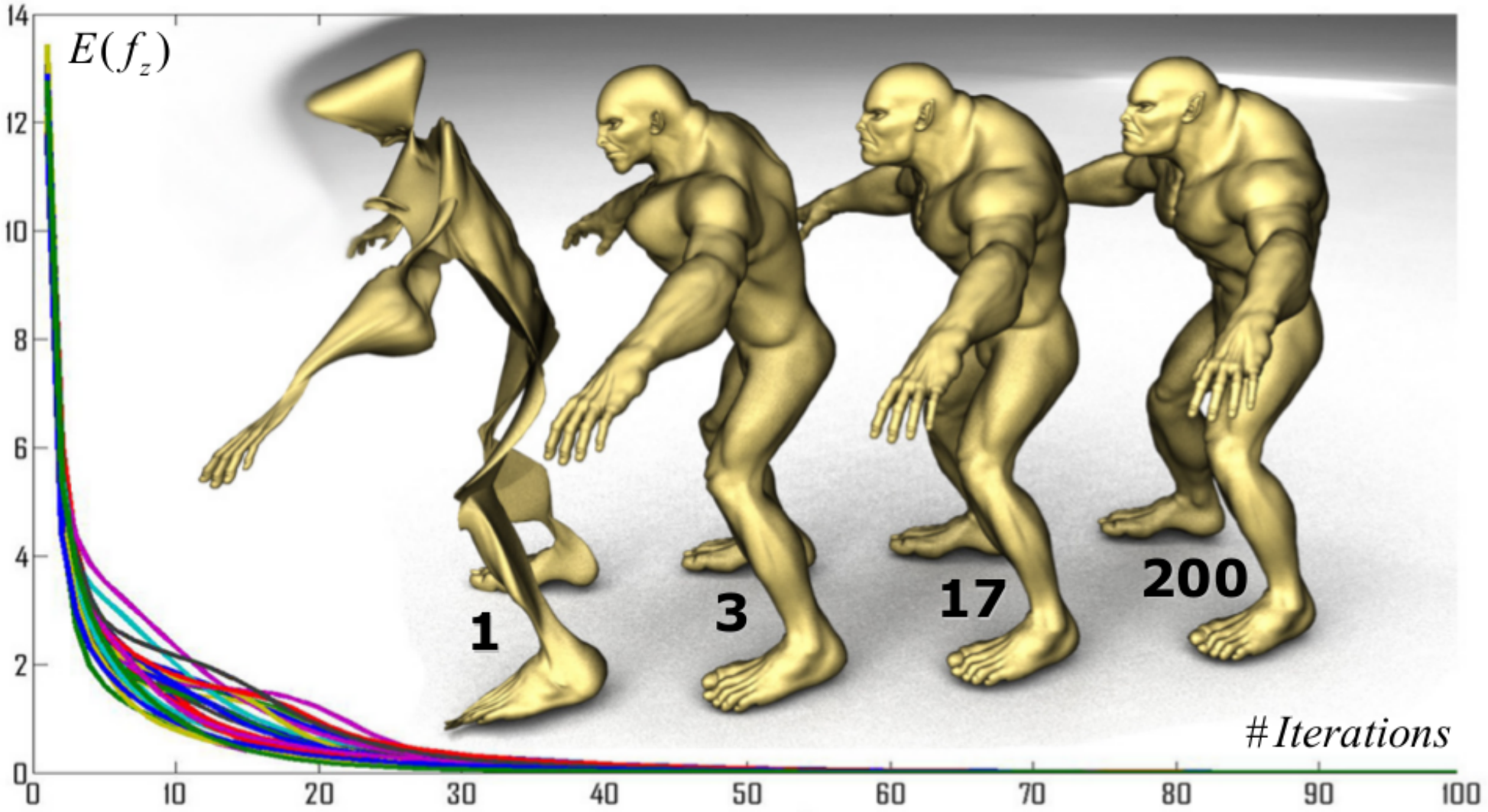
$$+ \quad \sum_{i \in RC} \left\| \sum_v \nabla \phi_v(h_i).a_v^T + \sum_t \nabla \psi_t(h_i).b_t^T - \bar{R}_i^T \right\|^2$$

$$+ \quad \sum_{i \in RC2} \left\| \sum_v H\,\phi_v(h_i).xa_v + \sum_t H\,\psi_t(h_i).xb_t \right\|^2$$

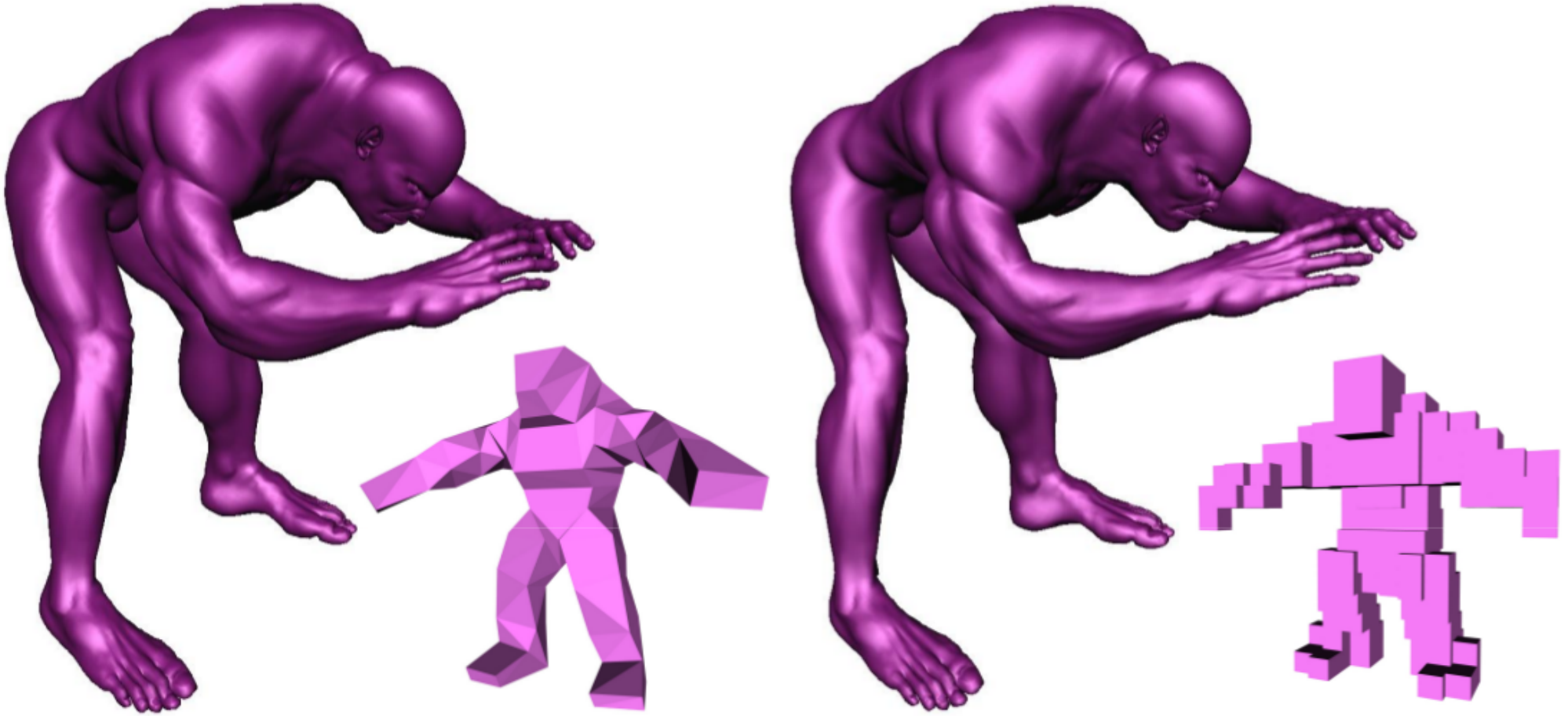
$$+ \quad \sum_{i \in RC2} \left\| \sum_v H\,\phi_v(h_i).ya_v + \sum_t H\,\psi_t(h_i).yb_t \right\|^2$$

$$+ \quad \sum_{i \in RC2} \left\| \sum_v H\,\phi_v(h_i).za_v + \sum_t H\,\psi_t(h_i).zb_t \right\|^2$$

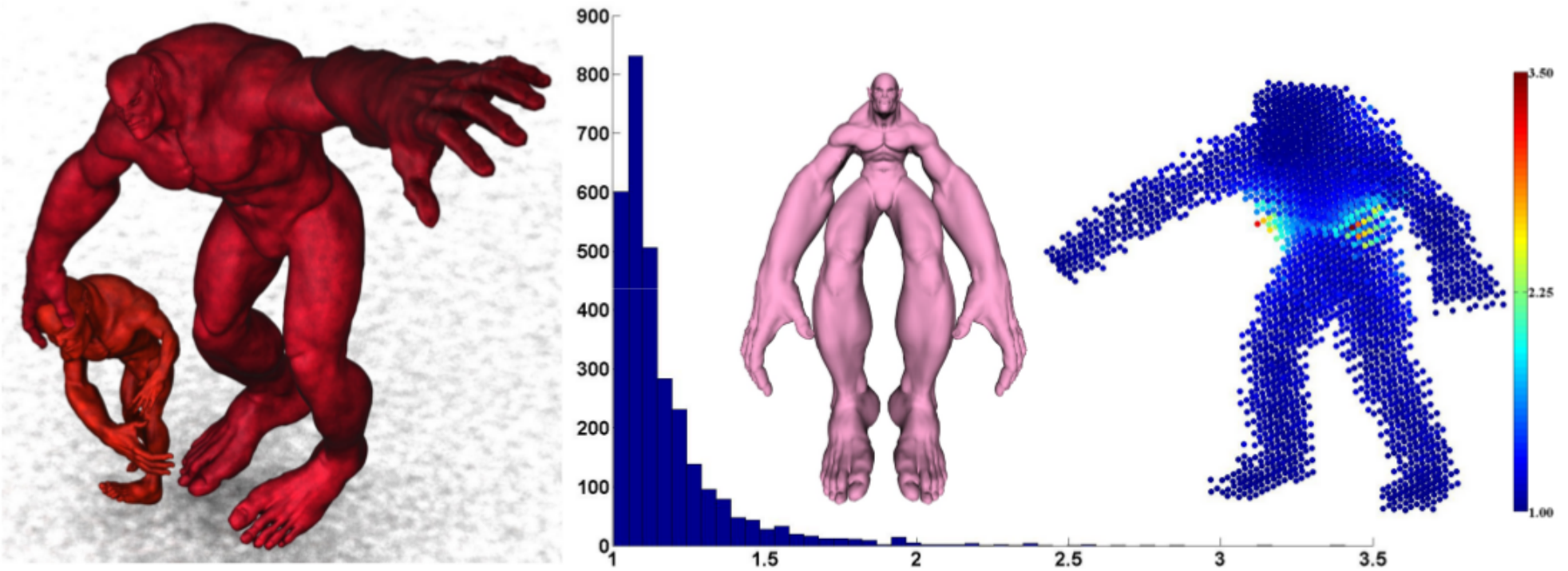
Iterations of the solver



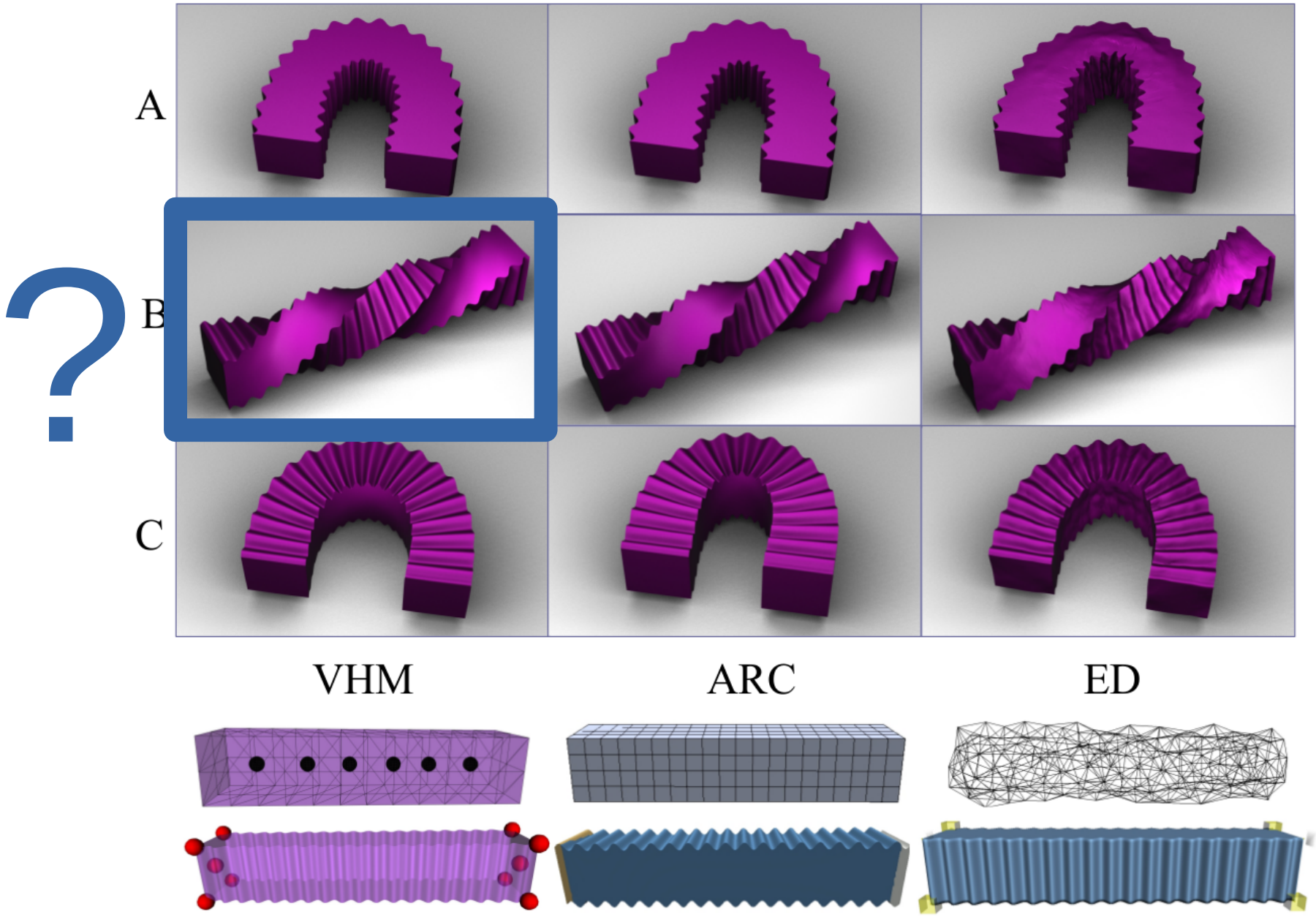
Sensitivity to input cage



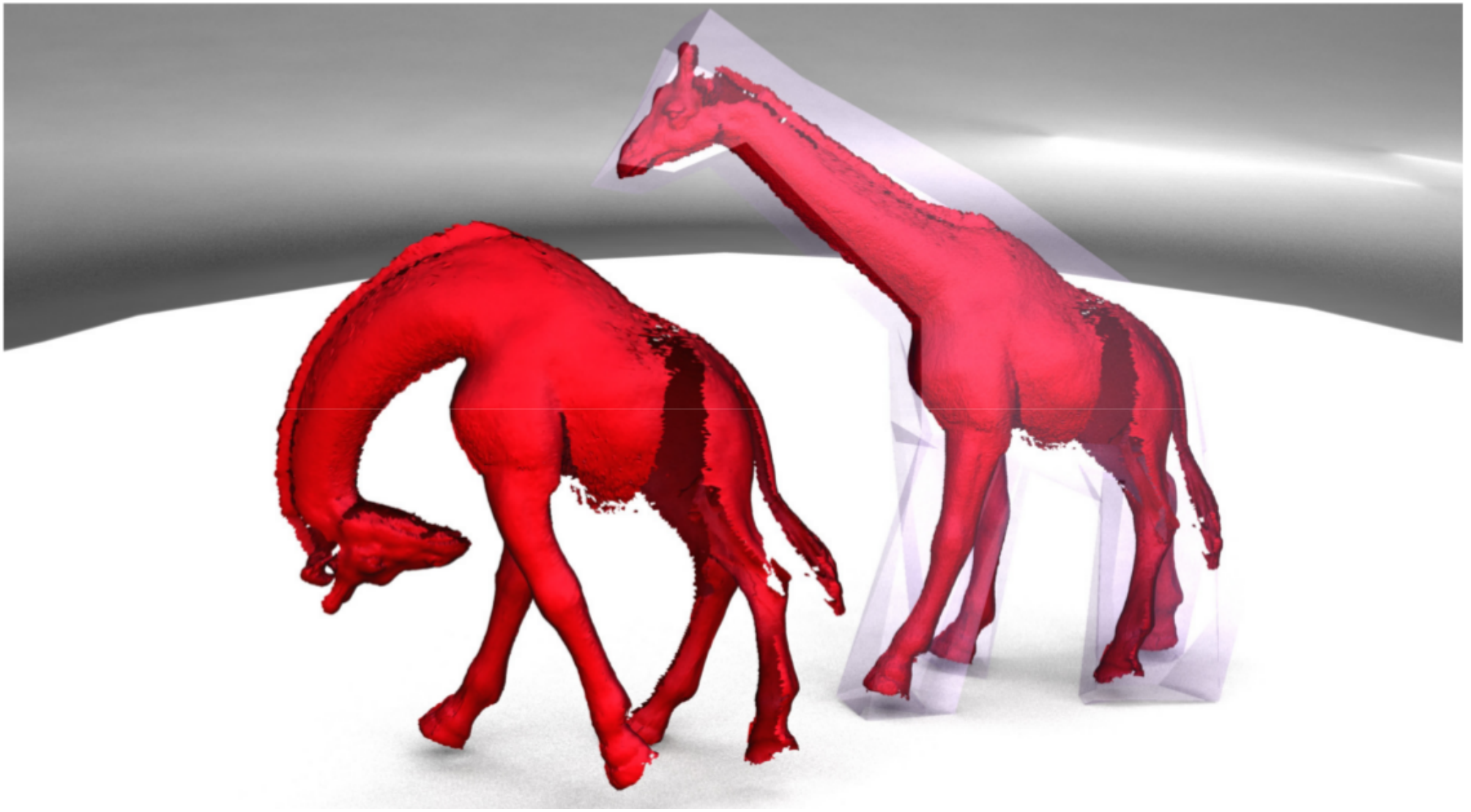
As-similar-as-possible deformations



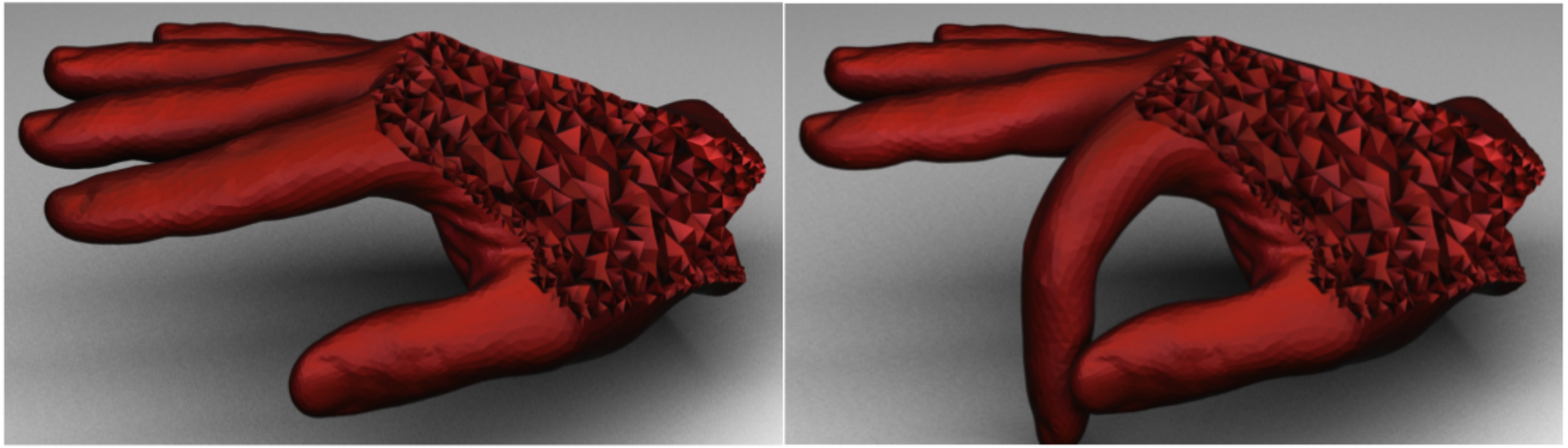
Results



Cage what you want



Deform volume meshes



Real-time ARAP deformation

Model	Verts	Cage faces	Ancrs	Iters	Solve (ms)	Def (ms)	Tot (ms)
Bar	32,908	208	6	15	0.27	1.84	5.89
Tet Hand	28,796	288	28	9	0.43	2.27	6.14
Giraffe	79,226	204	27	33	0.37	3.08	15.29
Beast	32,311	226	50	10	0.53	2.60	7.90
Arma leg	28,829	68	6	26	0.27	1.87	8.89
Arma	173,101	250	88	13	0.69	10.60	19.57

*Optimization is done in a **very low-dimensional space** → Real-time ARAP !*

Conclusions

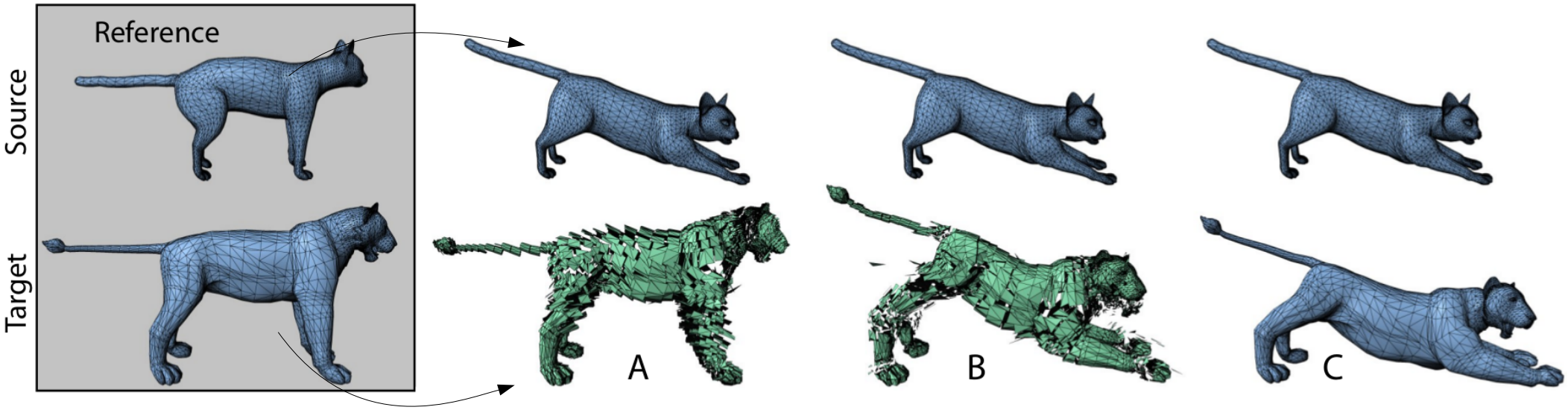
- Use existing framework as underlying (low-dimensional) machinery
- Relies on well-studied functions (harmonic)
- Infer «proximity» from the design of the function space itself (e.g., a cage to separate parts of the shape)

Spatial deformation transfer

[Benchen et al.] : Spatial Deformation Transfer

Spatial deformation transfer

Recall last week :

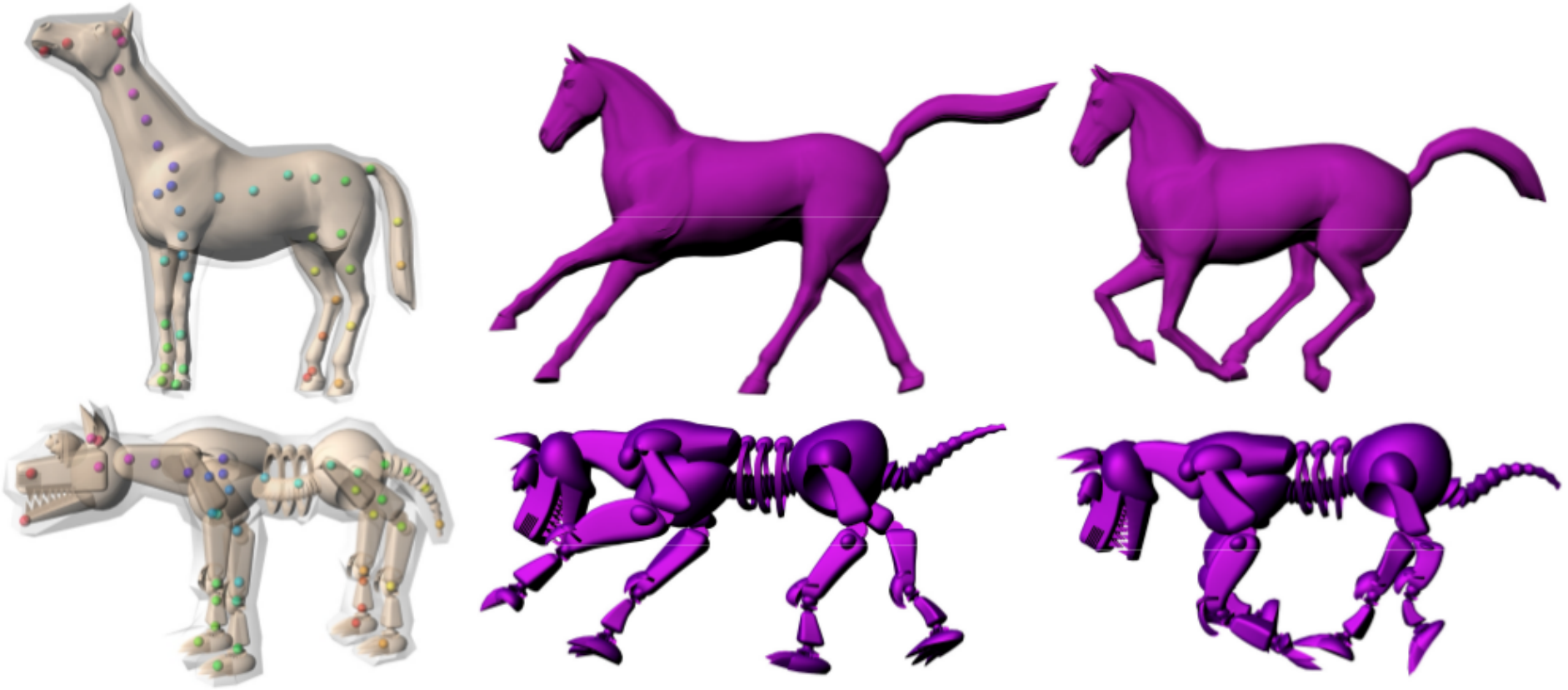


- Find transformation (3x3 matrix) of each triangle (A, top)
- Transfer transformations to triangles (A, bottom)
- Reconstruct shape (C)

Spatial deformation transfer

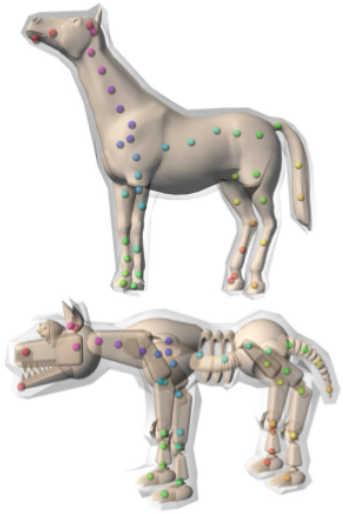
Today : transfer rotations of the medial axis

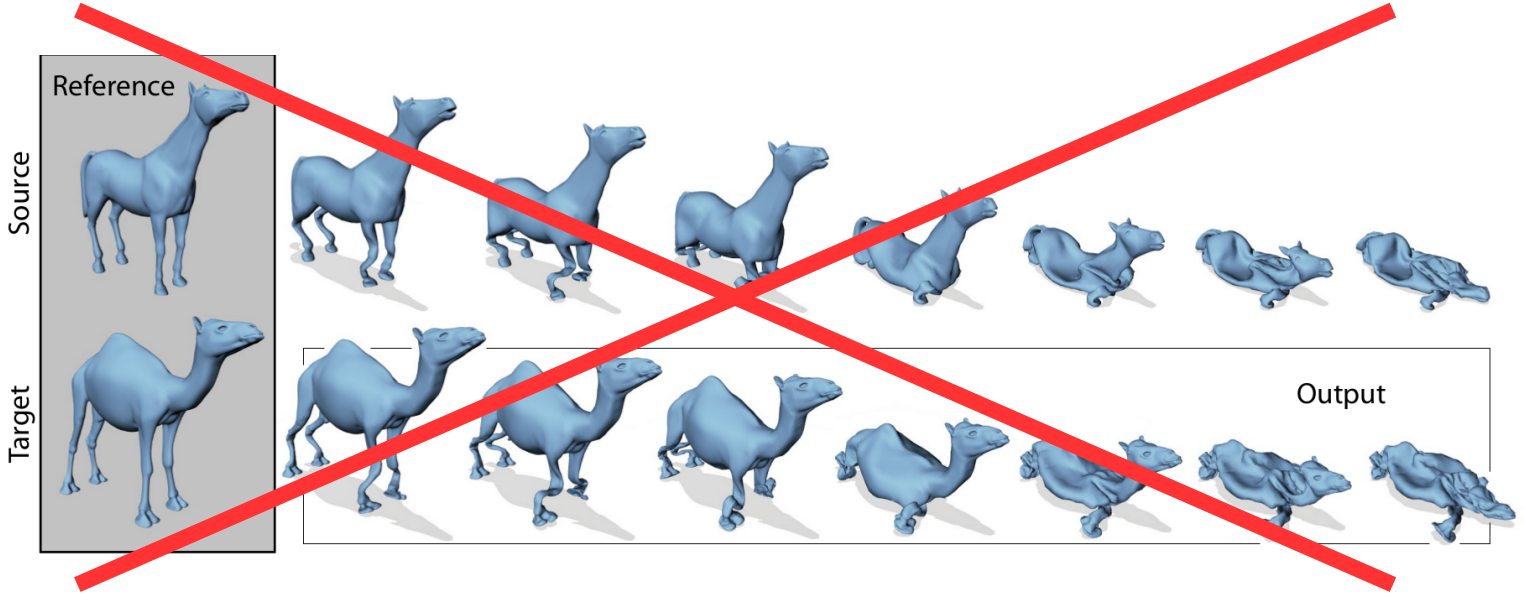
- Use VHM to obtain value of rotations on the medial axis on the horse
- Use VHM with no (a single) positional constraint to recover the deformation of the dead dog



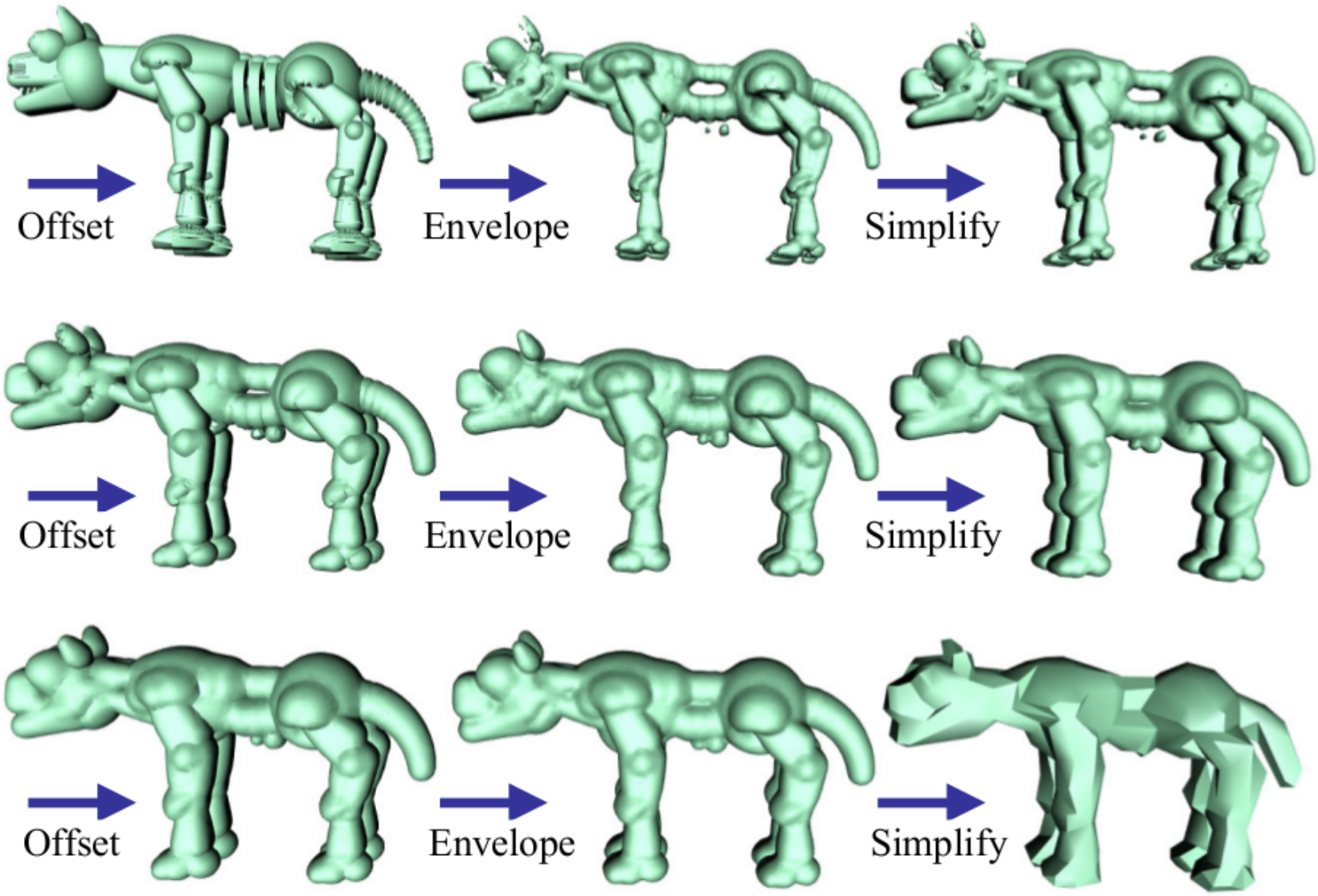
Spatial deformation transfer

- Pros :
  - Oblivious to topology : +++
  - Easy-to-define mapping : +++
- Cons :
  - Difficulty to « cage » shapes : -
  - Restricted to volumetric rigid deformations : -

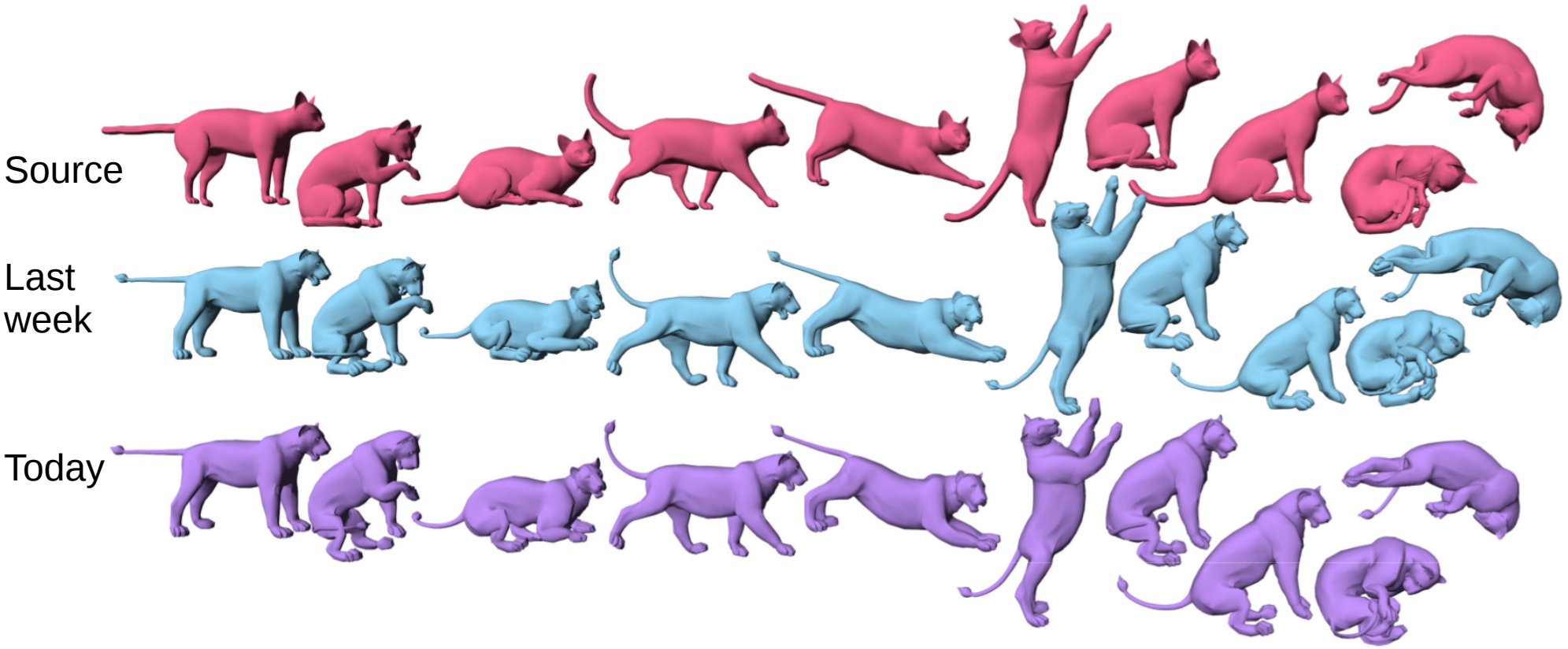




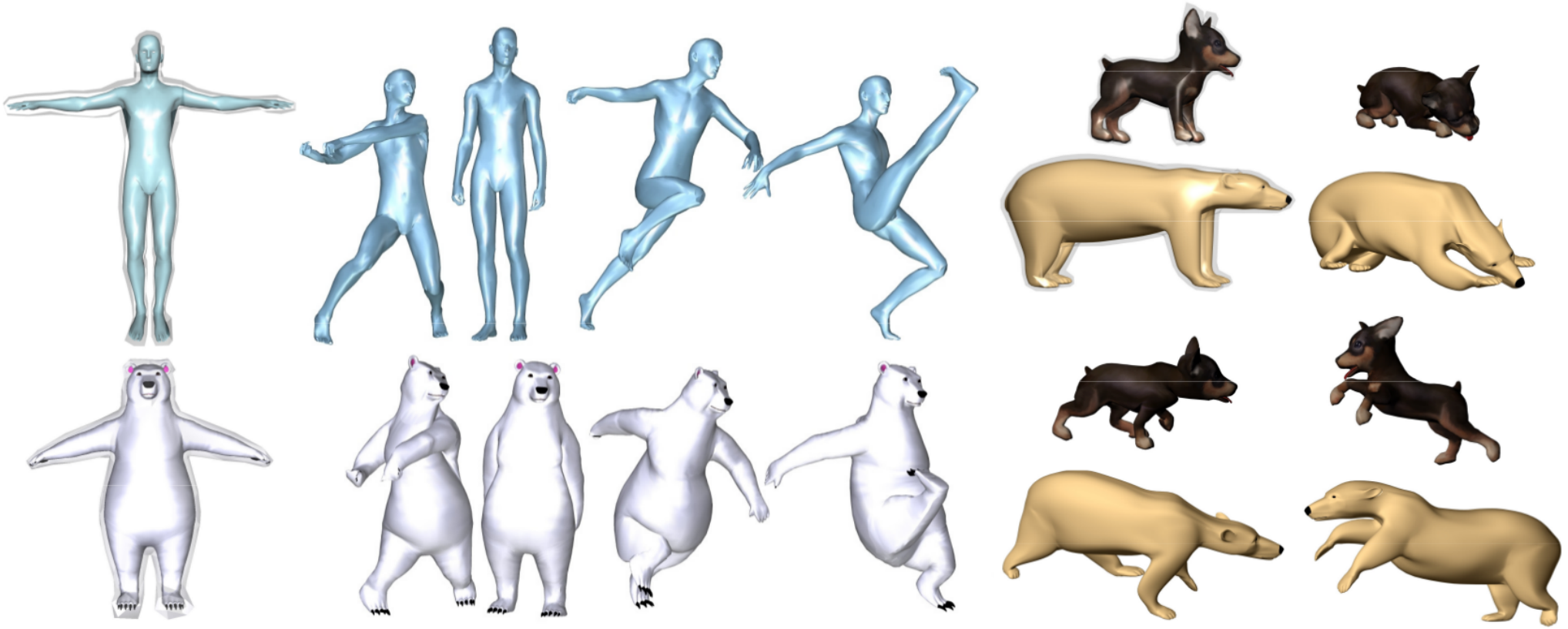
Automatic cages (cheap algo)



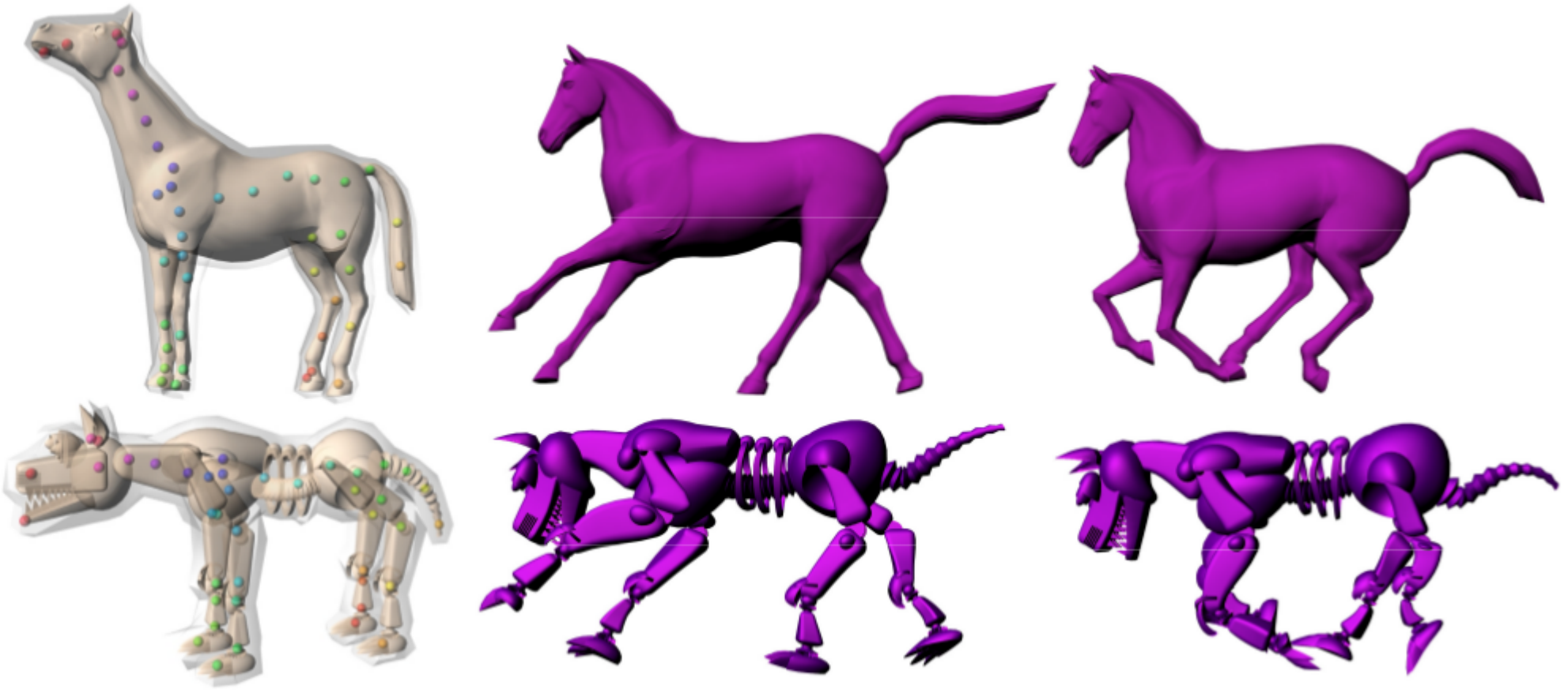
Spatial deformation transfer



Spatial deformation transfer



Spatial deformation transfer



Tough Challenges

- Cages for every shape :
  - Main problem : intersecting shapes
- Cages by artists :
  - Local density ?
  - Local distance to shape ?
  - Desired topology ?
- Cages made of quads :
  - Need to compute new sets of coordinates