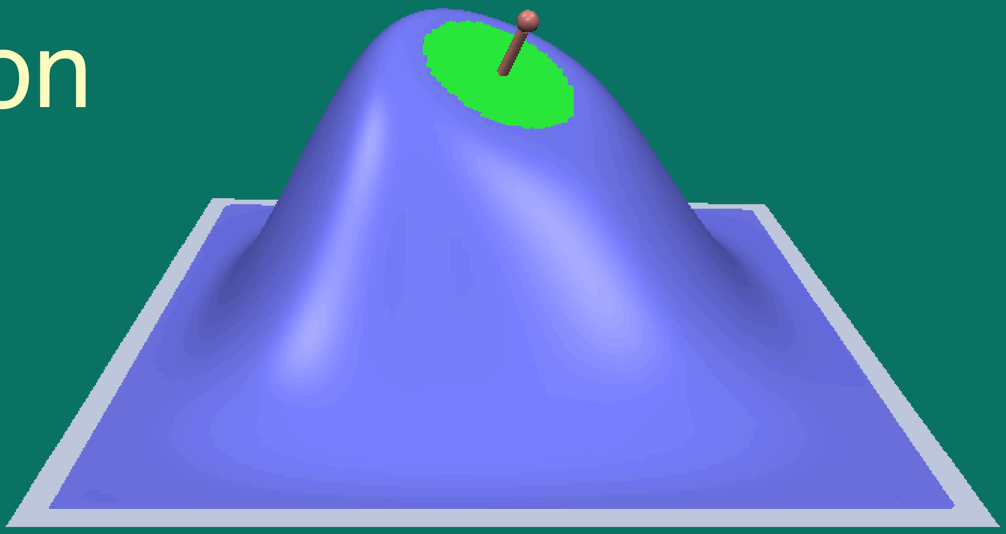
Real-Time Shape Editing using Radial Basis Functions

Mario Botsch, Leif Kobbelt
RWTH Aachen

Boundary Constraint Modeling

- Prescribe irregular constraints
  - Vertex positions
- Constrained energy minimization
  - ➡ Optimal fairness
- Euler-Lagrange PDE
  - ➡ Solve (bi- or tri-) Laplacian system per frame



Differential Constraint Modeling

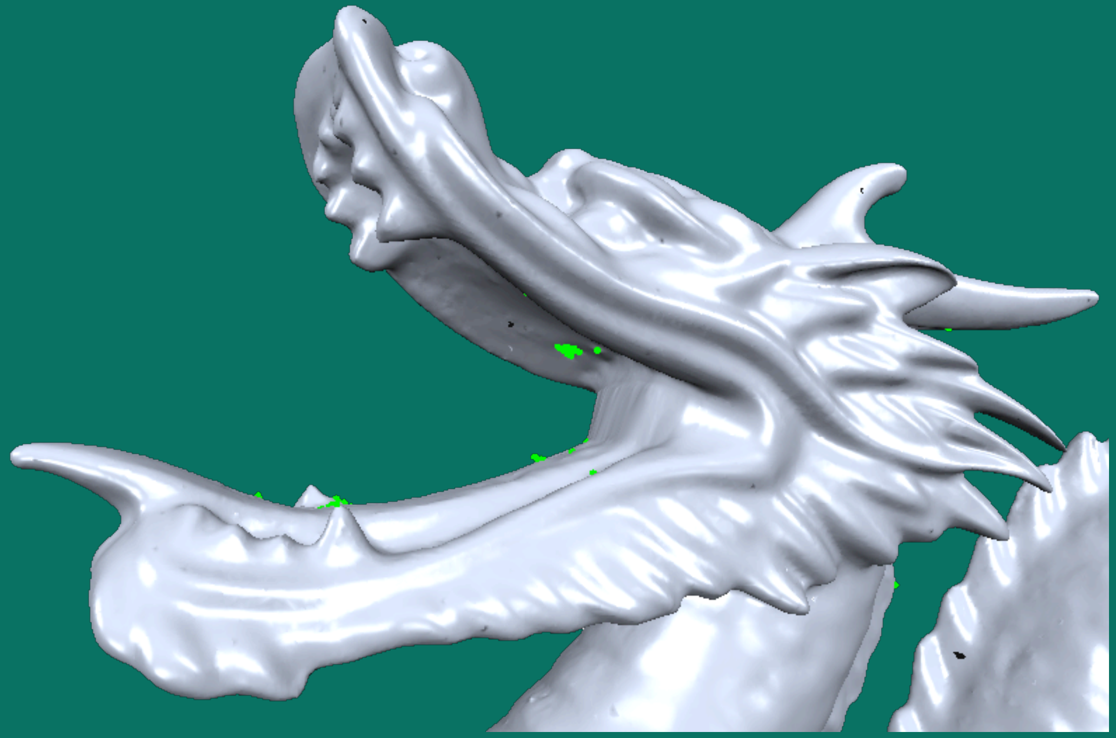
- Prescribe differential constraints
  - Laplace coordinates
  - Poisson gradient editing
  - Laplacian editing
  - Deformation gradients

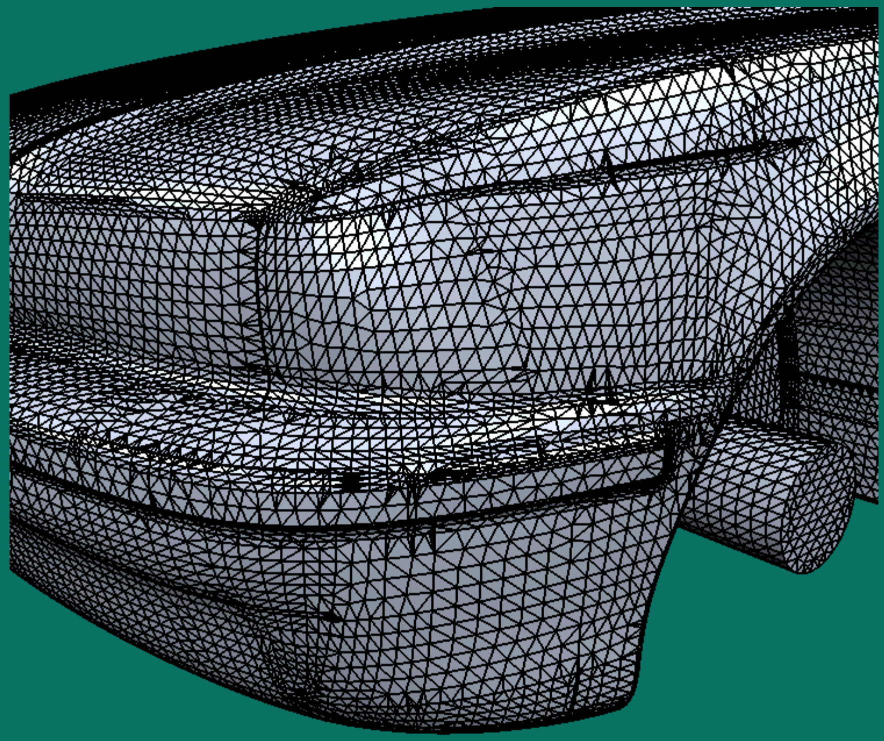
➡ Solve (least squares) Laplacian systems

Surface-Based Deformation

Problems with

- Highly complex models
- Topological inconsistencies
- Geometric degeneracies





Space Deformation

1. Control. Prescribe (irregular) constraints:

$$\mathbf{c}_i \mapsto \mathbf{c}_i'$$

2. Fitting. Smoothly interpolate constraints by a displacement function *in space*:

$$\mathbf{d} : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \quad \text{with} \quad \mathbf{d}\left(\mathbf{c}_i\right) = \mathbf{c}_i'$$

3. Evaluation. Displace all points:

$$\mathbf{p}_i \mapsto \mathbf{d}\left(\mathbf{p}_i\right) \qquad \forall \mathbf{p}_i \in \mathcal{S}$$

How to interpolate?

- Represent deformation by RBFs

$$\mathbf{d}(\mathbf{x}) = \sum_j \mathbf{w}_j \cdot \varphi(\|\mathbf{c}_j - \mathbf{x}\|) + \mathbf{p}(\mathbf{x})$$

- Well suited for scattered data interpolation
  - Smooth interpolation
  - Irregular constraints

Which basis function?

- Triharmonic RBF $\varphi(r) = r^3$
  - ➡ C^2 boundary constraints
  - ➡ High fairness *(energy minimization)*
- Globally supported RBF
  - ➡ Works well for *irregular* constraints
  - ➡ But linear systems are *dense*

Which basis function?

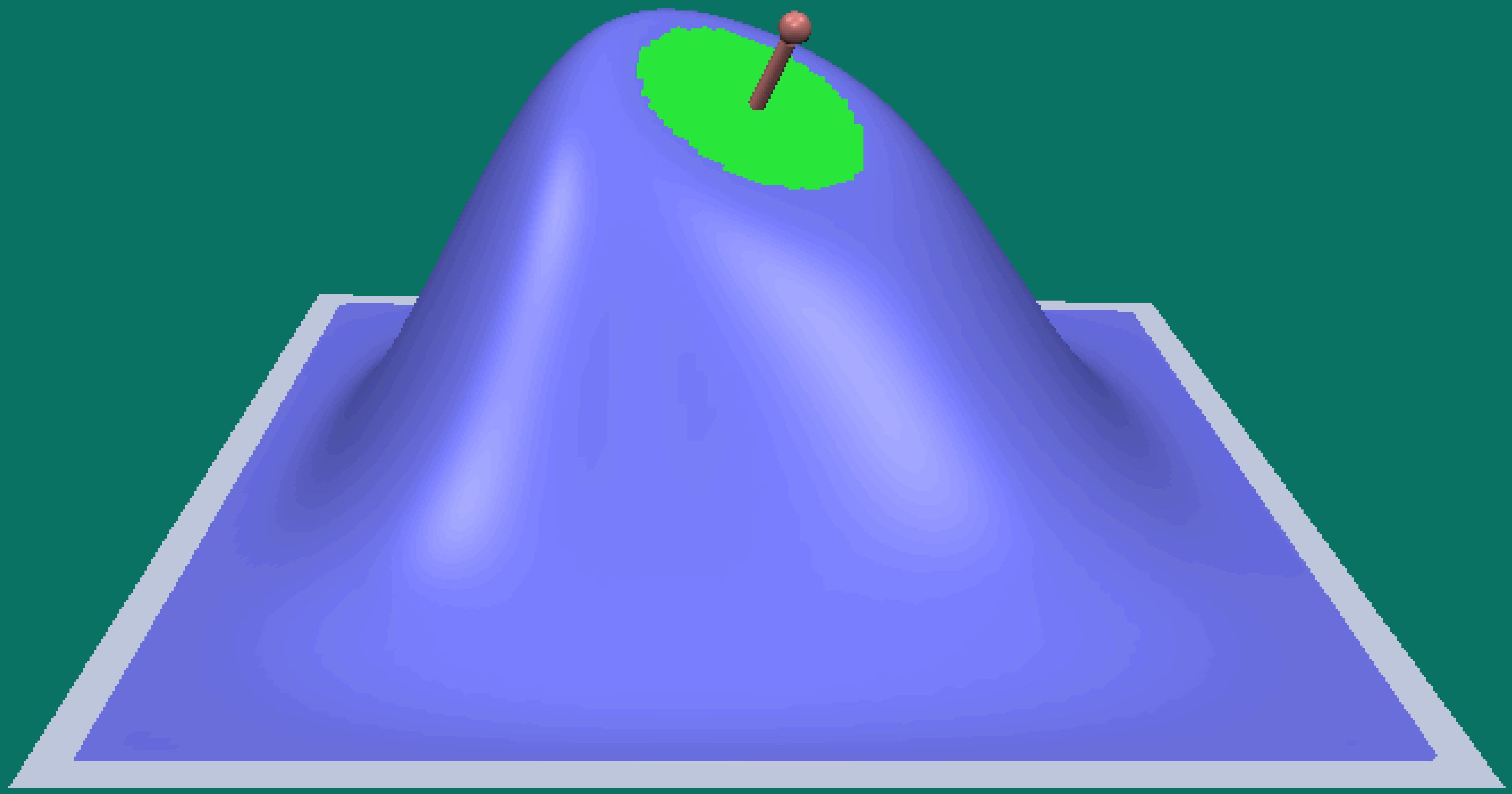
- Compactly supported functions...
  - are more efficient *(sparse systems)*
  - but yield inferior fairness
- Don’t trade quality for efficiency!
  - ➡ Use triharmonic functions
  - ➡ Accelerate involved computations

Overview

- Introduction
- RBF Modeling Setup
- Incremental Least Squares Solver
- Precomputed Basis Functions
- GPU Implementation
- Results

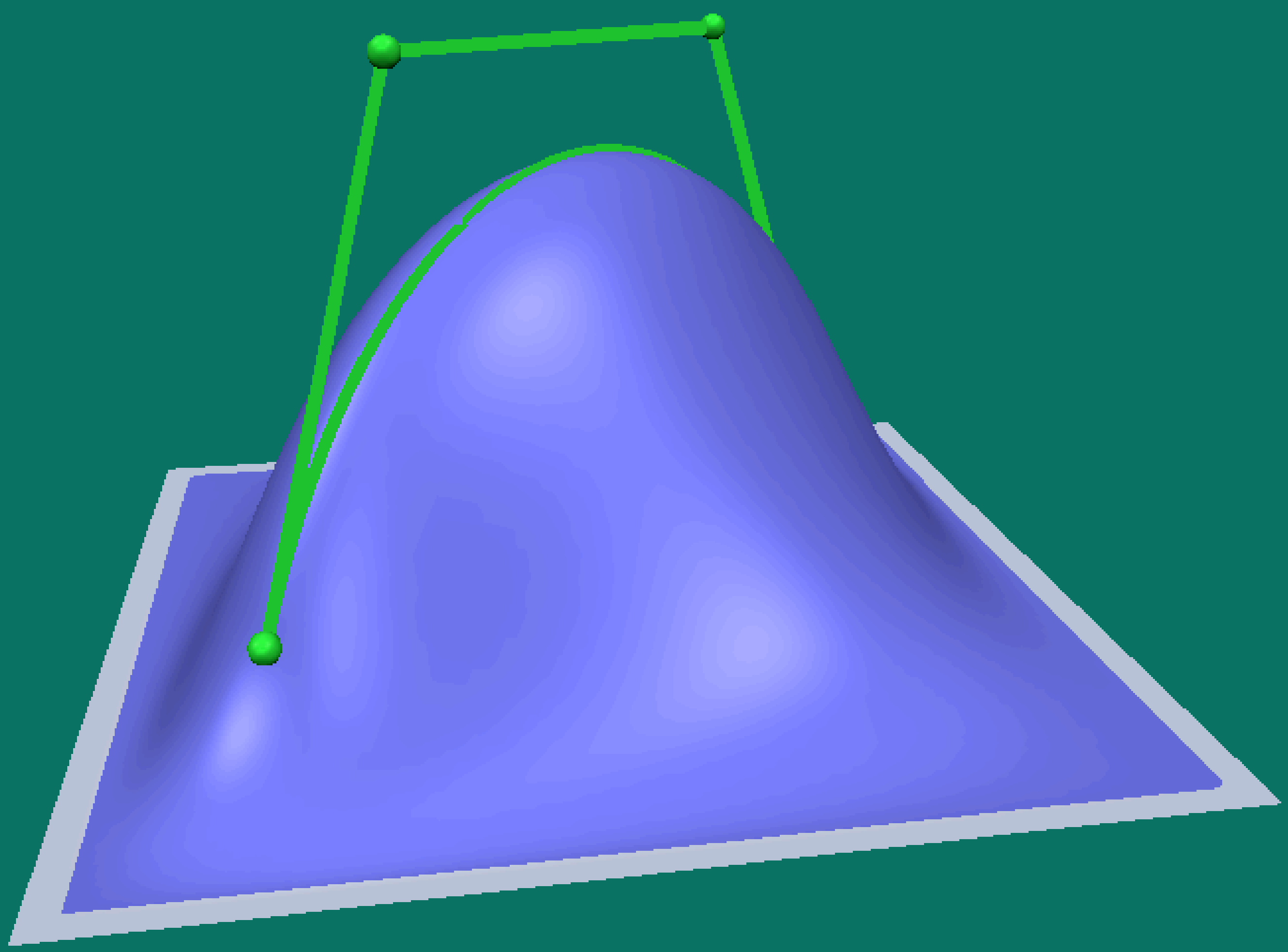
Handle Metaphor

- Affinely transformed control handle
  - Fixed vertices $\mathbf{f}_i \mapsto \mathbf{f}_i$
  - Handle vertices $\mathbf{h}_i \mapsto \mathbf{h}'_i$



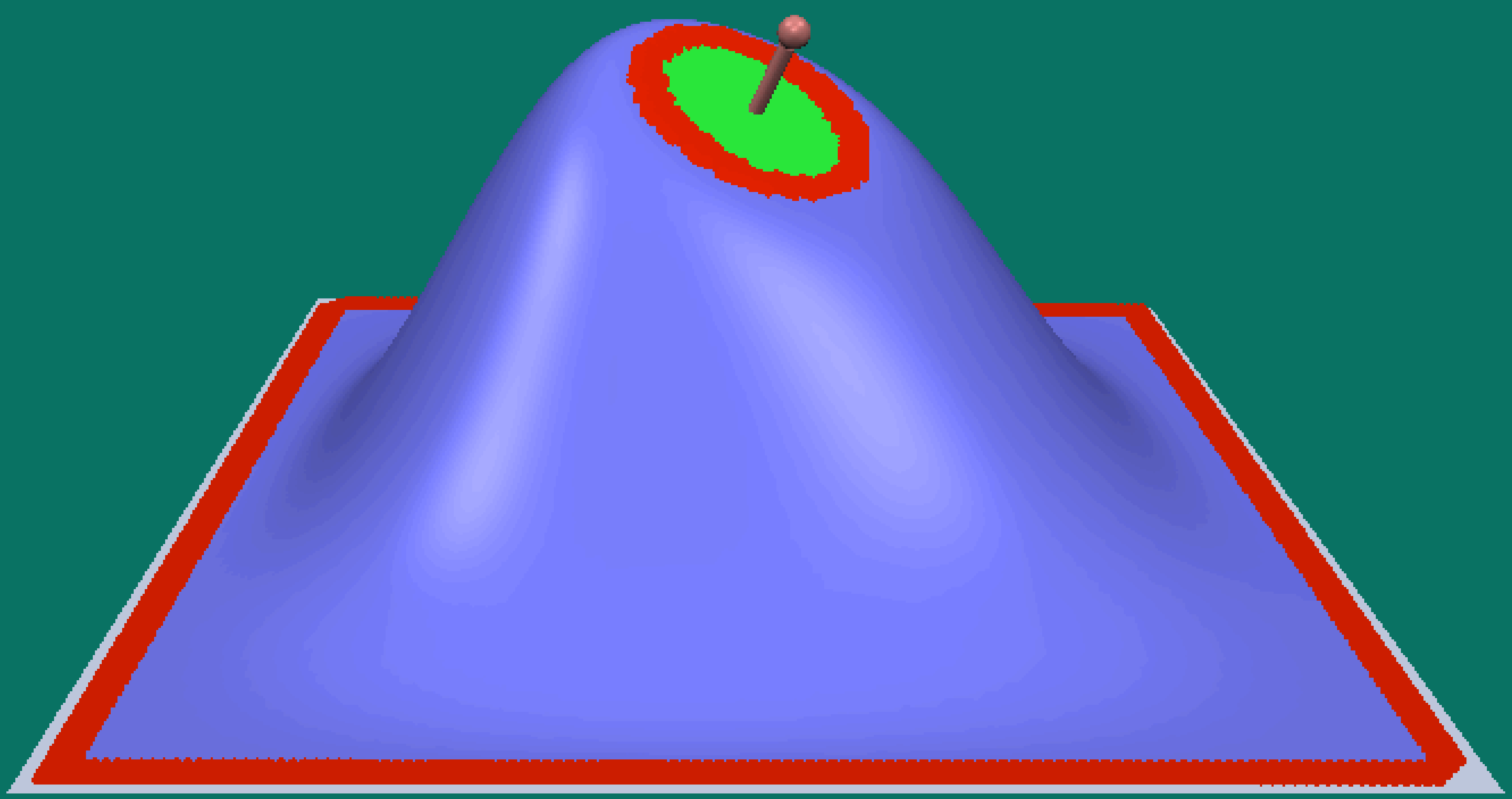
Curve Metaphor

- Deform spline curve <u>on</u> the surface
  - Fixed vertices $\mathbf{f}_i \mapsto \mathbf{f}_i$
  - Curve points $\mathbf{c}\left(t_i\right) \mapsto \mathbf{c}'\left(t_i\right)$



C^2 Boundary Constraints

- Three rings of constrained points
- Finite difference approximation to exact C^2 constraints



RBF Fitting

- Place *m* centers at *m* constraints

$$\{\mathbf{c}_i\} \;=\; \{\mathbf{f}_i\} \cup \{\mathbf{h}_i\}$$

- Solve *m×m* system for weights $\{\mathbf{w}_j\}$

$$\Phi \cdot W \;=\; \begin{pmatrix} F \\ H' \end{pmatrix}$$

  - Rows *i* ⇔ constraints
  - Columns *j* ⇔ basis functions

Overview

- Introduction
- RBF Modeling Setup
- Incremental Least Squares Solver
- Precomputed Basis Functions
- GPU Implementation
- Results

RBF Fitting

- Computation time should depend on...
  - *deformation* complexity
  - *not* *surface* complexity

- Simple deformation, complex surface?
  - Not all *m* basis functions needed
  - Solve up to error tolerance

Incremental RBF Fitting

1. Start with a few basis functions only
2. Iteratively refine approximation
   1. Add one basis function
   2. Recompute fitting
   3. Break if error < tolerance

which one?

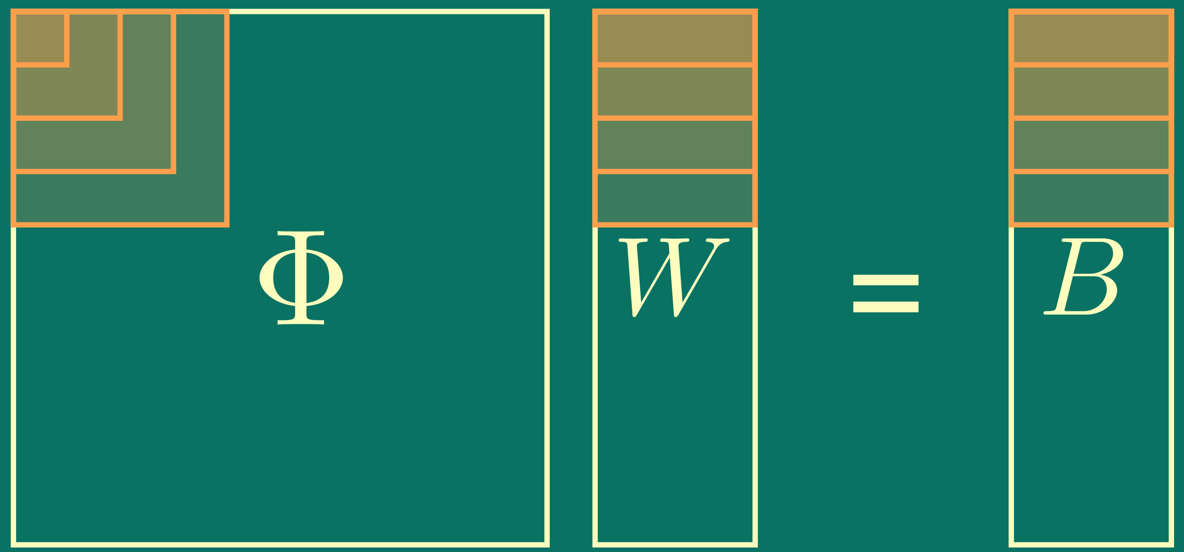
how to re-fit efficiently?

how to check efficiently?

Carr et al. SG 2001

Exactly interpolate *n* chosen constraints

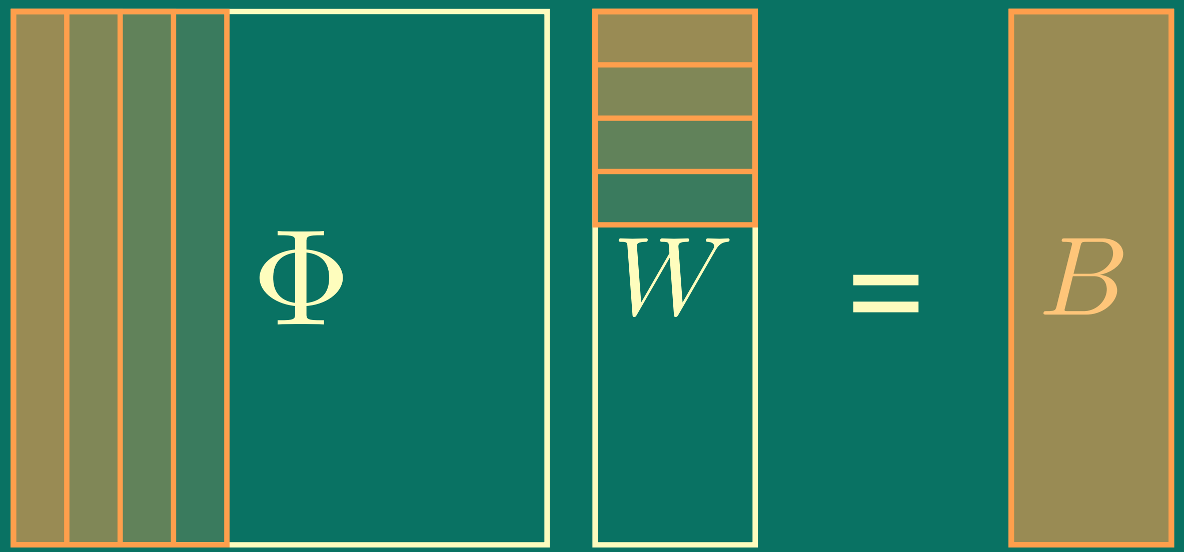
- Solve upper *n×n* block
- `for` *n* `= 1 to` *m* `do`



Incremental Least Squares

Compute optimal L^2 approximation

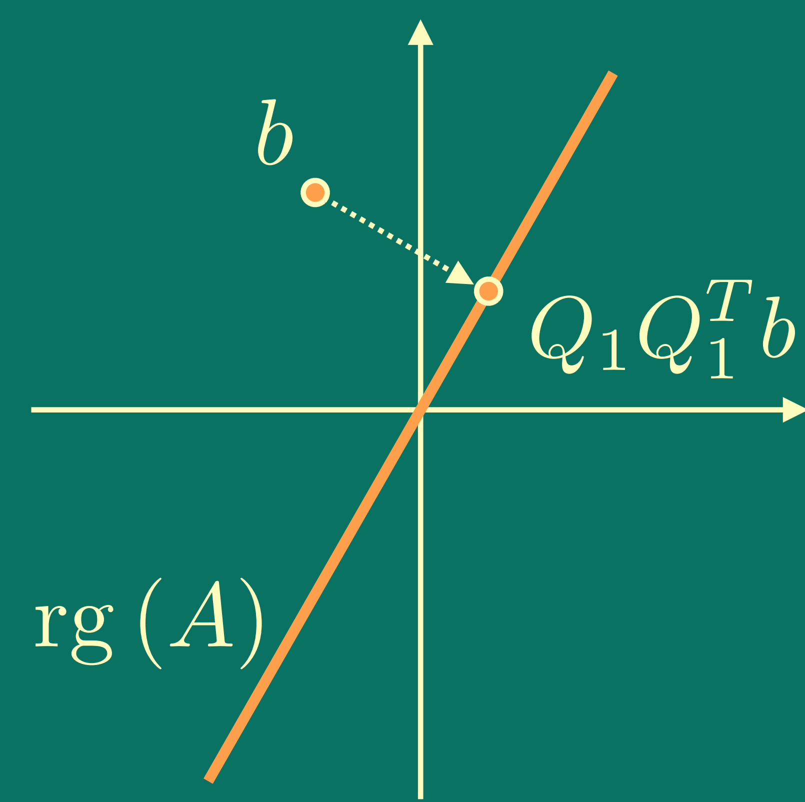
- Solve left *m×n* block *(least squares)*
- `for` n `= 1 to` m `do`



Least Squares QR Method

- Overdetermined system $Ax = b$

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}$$



- Least Squares solution

$$Rx = Q_1^T b$$

- L² error

$$\|b - Ax\| = \|Q_2^T b\|$$

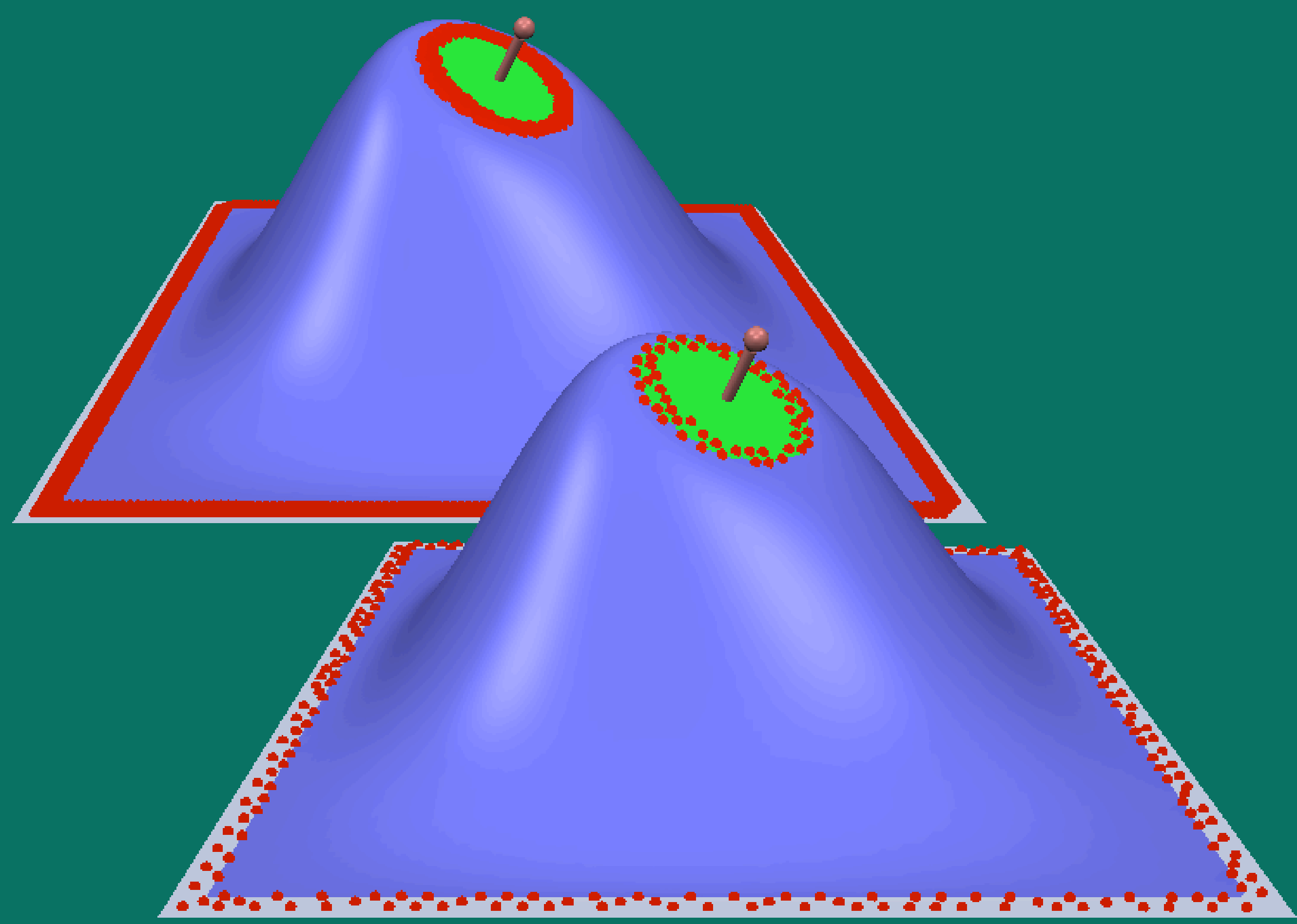
Incremental QR Solver

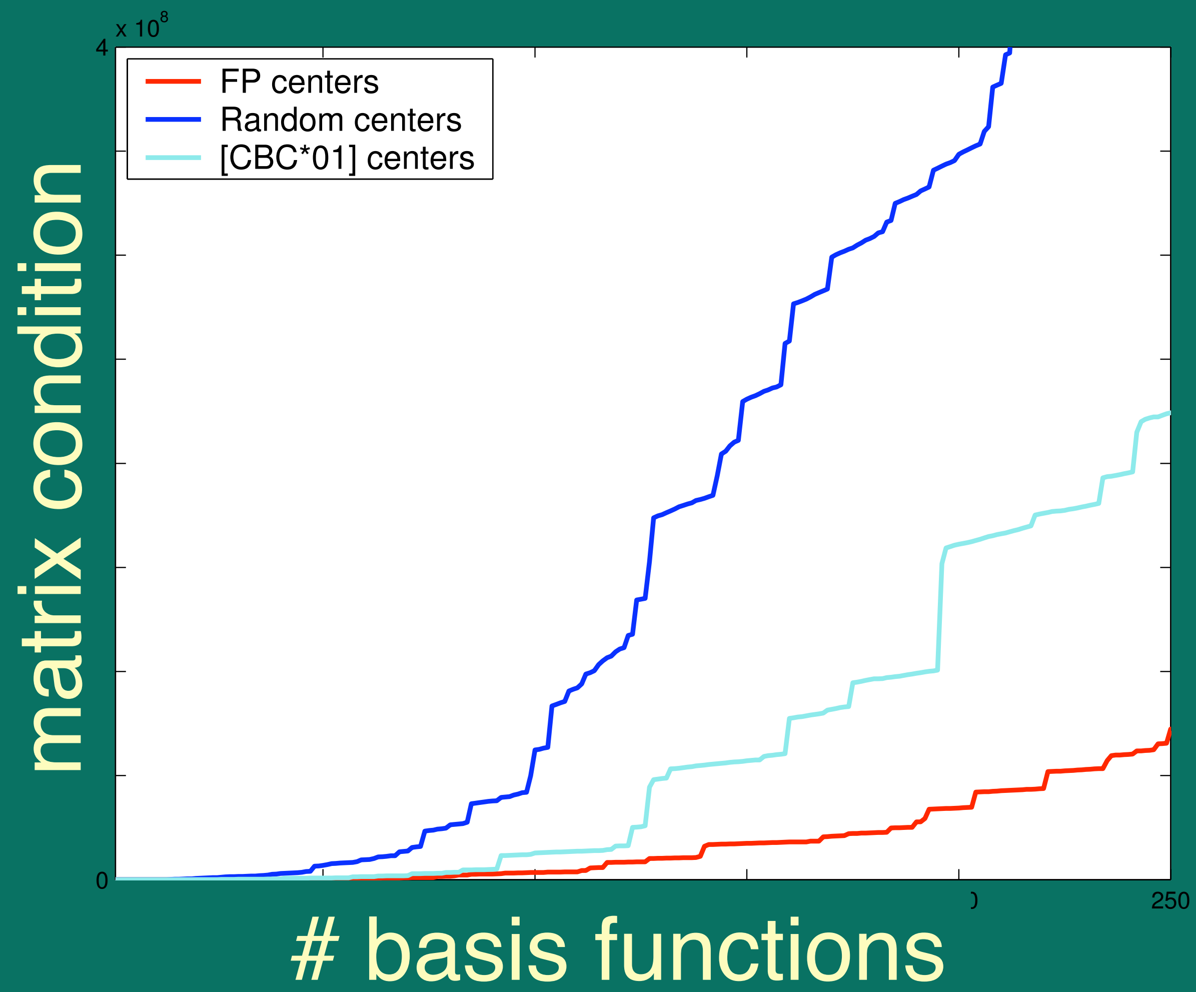
- In each iteration...
  - add one more basis function
  - add one more column
  - do one QR iteration *(Householder)*
- Slight adjustment of standard QR
  - Iterate until error < tolerance
  - Then solve $Rx = Q_1^T b$
  - Comes at *no* performance penalty!

Which centers to choose?

"Farthest point sampling" of RBF centers

- ➡ Linearly independent columns
- ➡ Good matrix condition



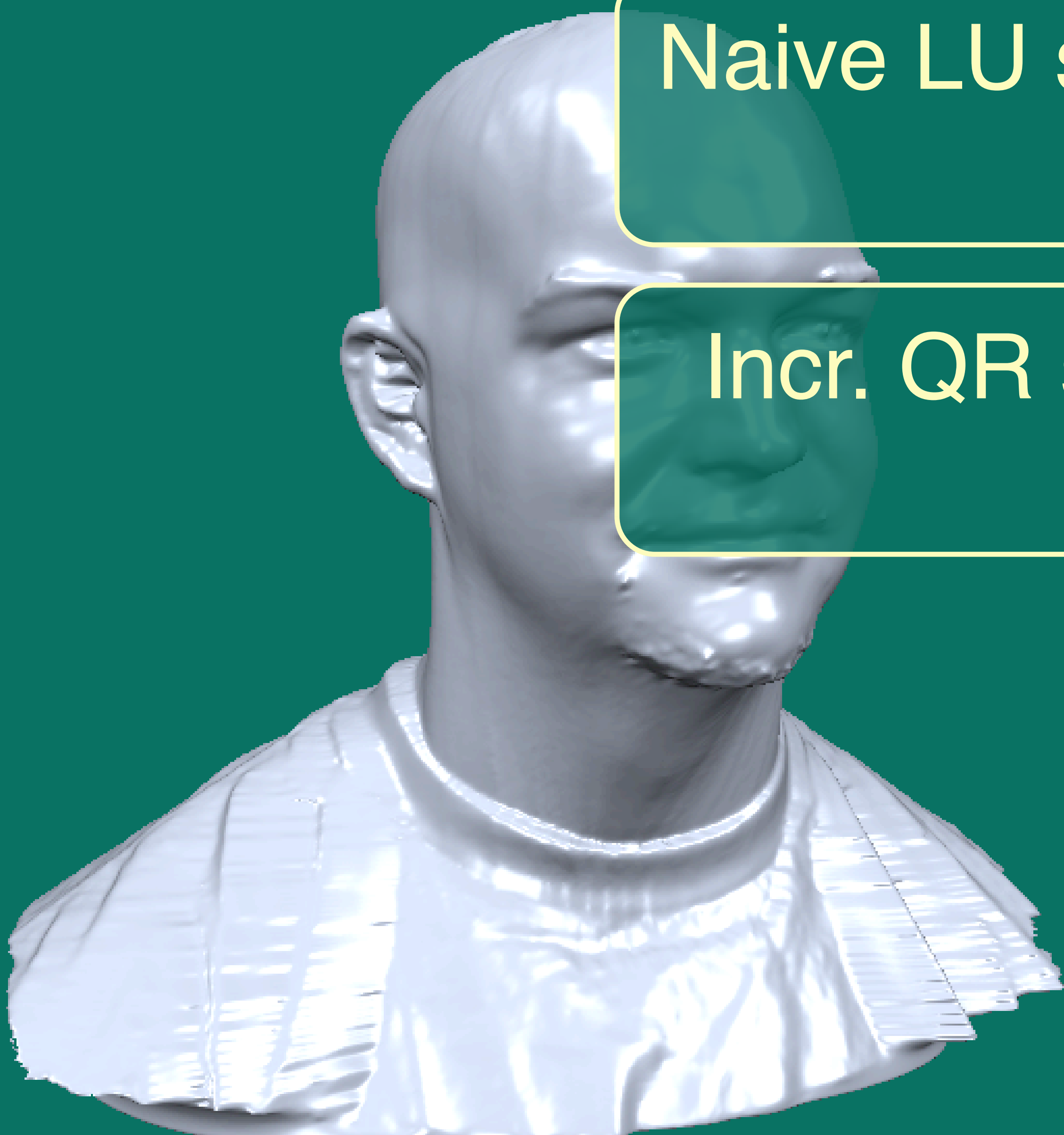


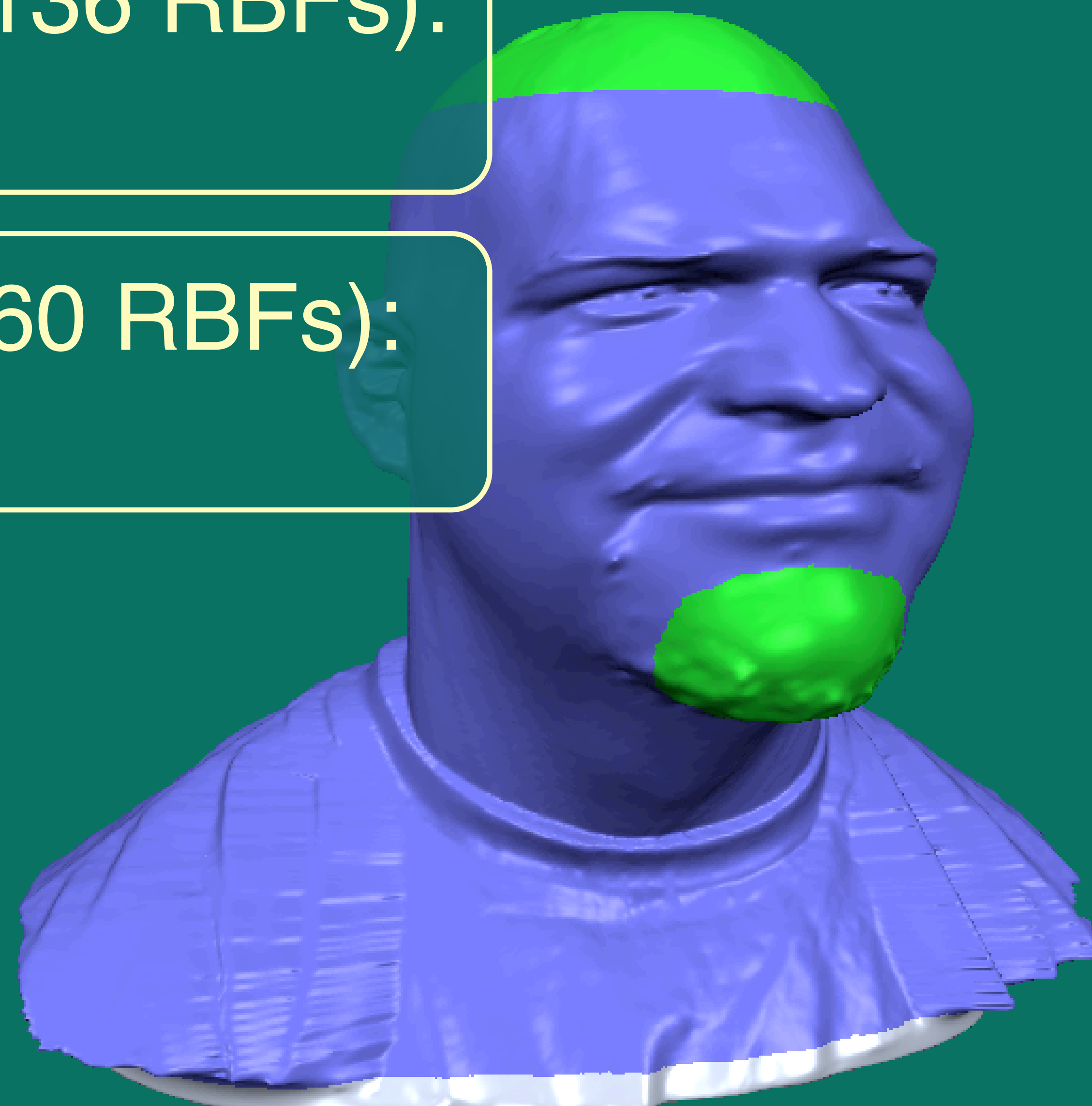
Surface Deformation

270k blue vertices, 4136 constraints

Naive LU solver (4136 RBFs):
153s

Incr. QR solver (160 RBFs):
1.99s





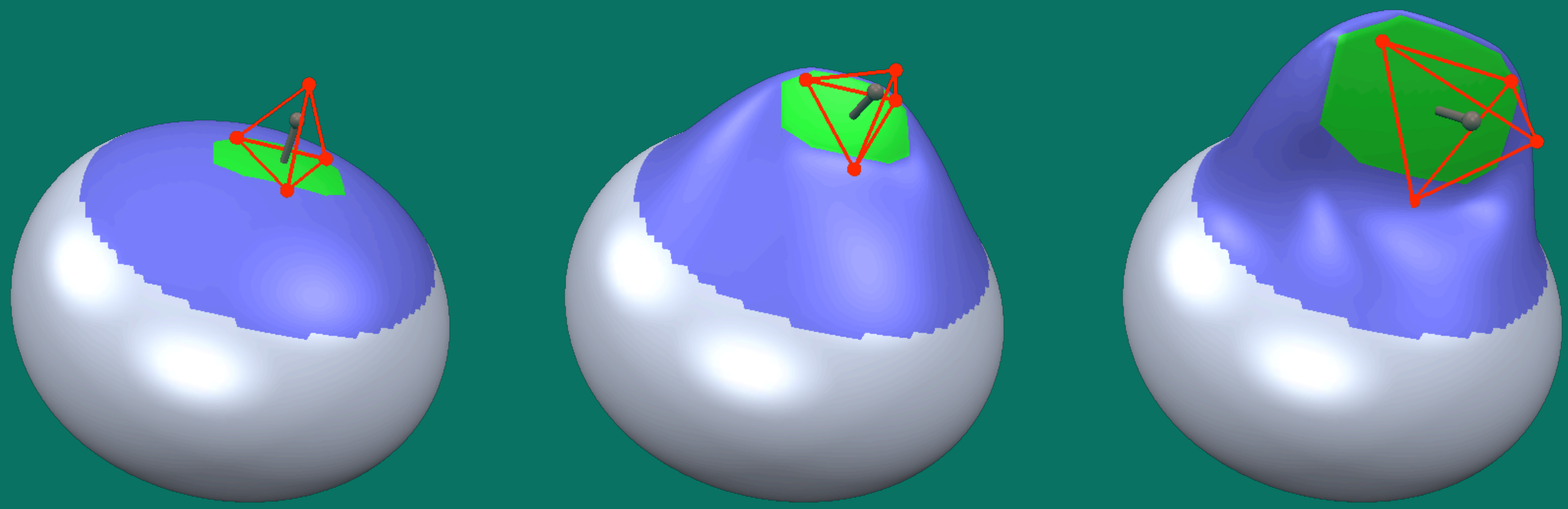
Overview

- Introduction
- RBF Modeling Setup
- Incremental Least Squares Solver
- Precomputed Basis Functions
- GPU Implementation
- Results

Precomputed Basis Functions

- Affine coordinate system for handle

$$H \;=\; M\;(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})^T \;=:\; M\,C$$



Precomputed Basis Functions

- Affine coord. system for handle
  - ➡ H depends linearly on C
- Fitting: pseudo inverse
  - ➡ W depends linearly on H
- Evaluation: matrix multiplication
  - ➡ P depends linearly on W

➡ P depends linearly on C

Precomputed Basis Functions

- In terms of displacements:

$$P' = P + B\,\delta C$$

- Simplifies fitting & evaluation to weighted sum of 4 displacements

- Works for curve metaphor as well
  - Curve points are affine combination of control points

Surface Deformation

270k blue vertices, 4136 constraints

Naive LU solver (4136 RBFs):
153s

Incr. QR solver (160 RBFs):
1.99s

Basis precomp.: 7.5s
CPU per-frame: 0.27s

Overview

- Introduction
- RBF Modeling Setup
- Incremental Least Squares Solver
- Precomputed Basis Functions
- GPU Implementation
- Results

GPU Implementation

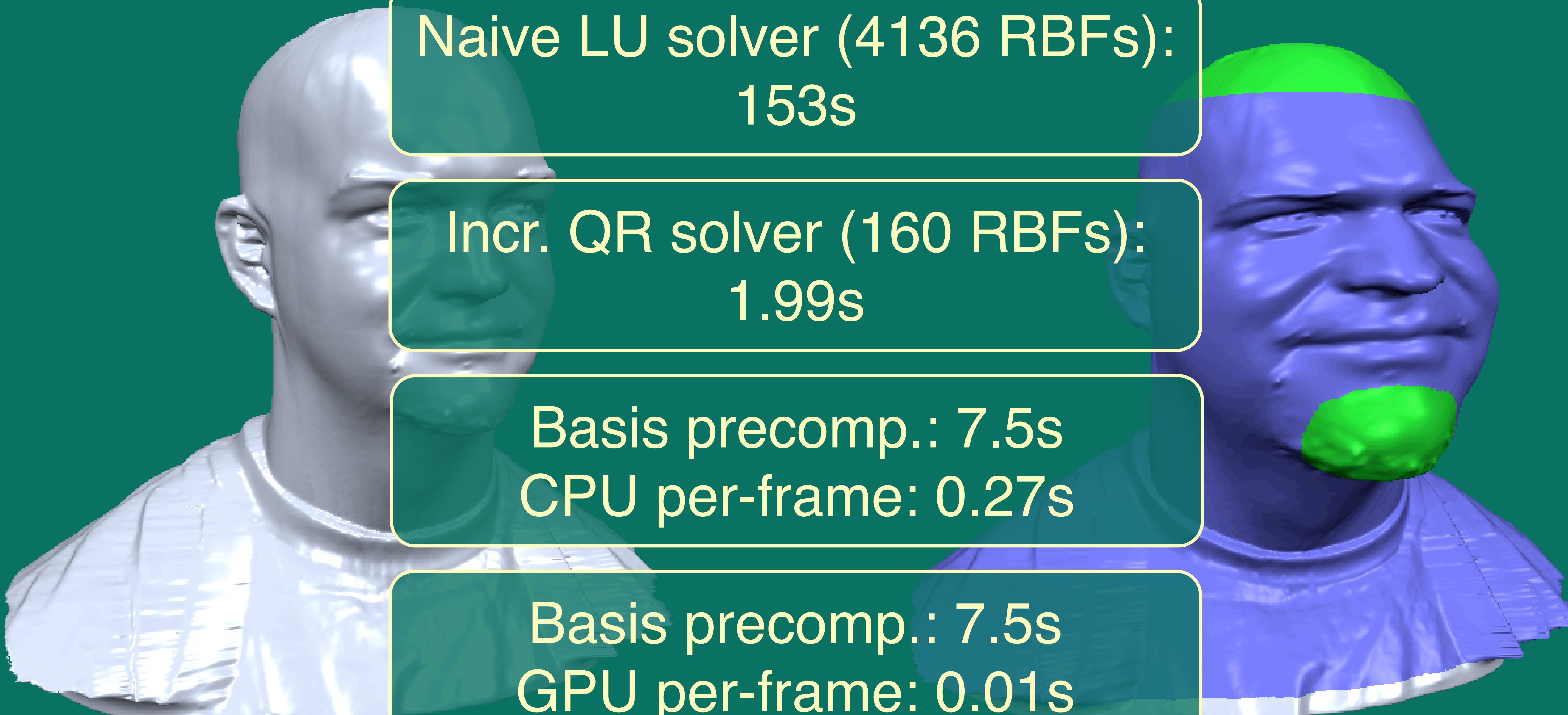
- Analytic space deformation
  - Transform points $\mathbf{p}'_i = \mathbf{d}(\mathbf{p}_i)$
  - Transform tangents $\mathbf{t}'_i = J_{\mathbf{d}}(\mathbf{p}_i)\,\mathbf{t}_i$
  - Transform normals $\mathbf{n}'_i = J_{\mathbf{d}}(\mathbf{p}_i)^{-T}\,\mathbf{n}_i$
- Precompute basis functions for
  - Deformation $\mathbf{d}(\cdot) \Rightarrow B$
  - Jacobian $J_{\mathbf{d}}(\cdot) \Rightarrow B_x, B_y, B_z$
  - Requires 16 floats per vertex

GPU Implementation

- Each point is handled individually
  - ➡ Easily computed in vertex shader
- Now all geometry data is static
  - ➡ Store in video memory
- Only affine frame changes
  - ➡ Global shader variable (12 floats)

Surface Deformation

270k blue vertices, 4136 constraints



Naive LU solver (4136 RBFs):
153s

Incr. QR solver (160 RBFs):
1.99s

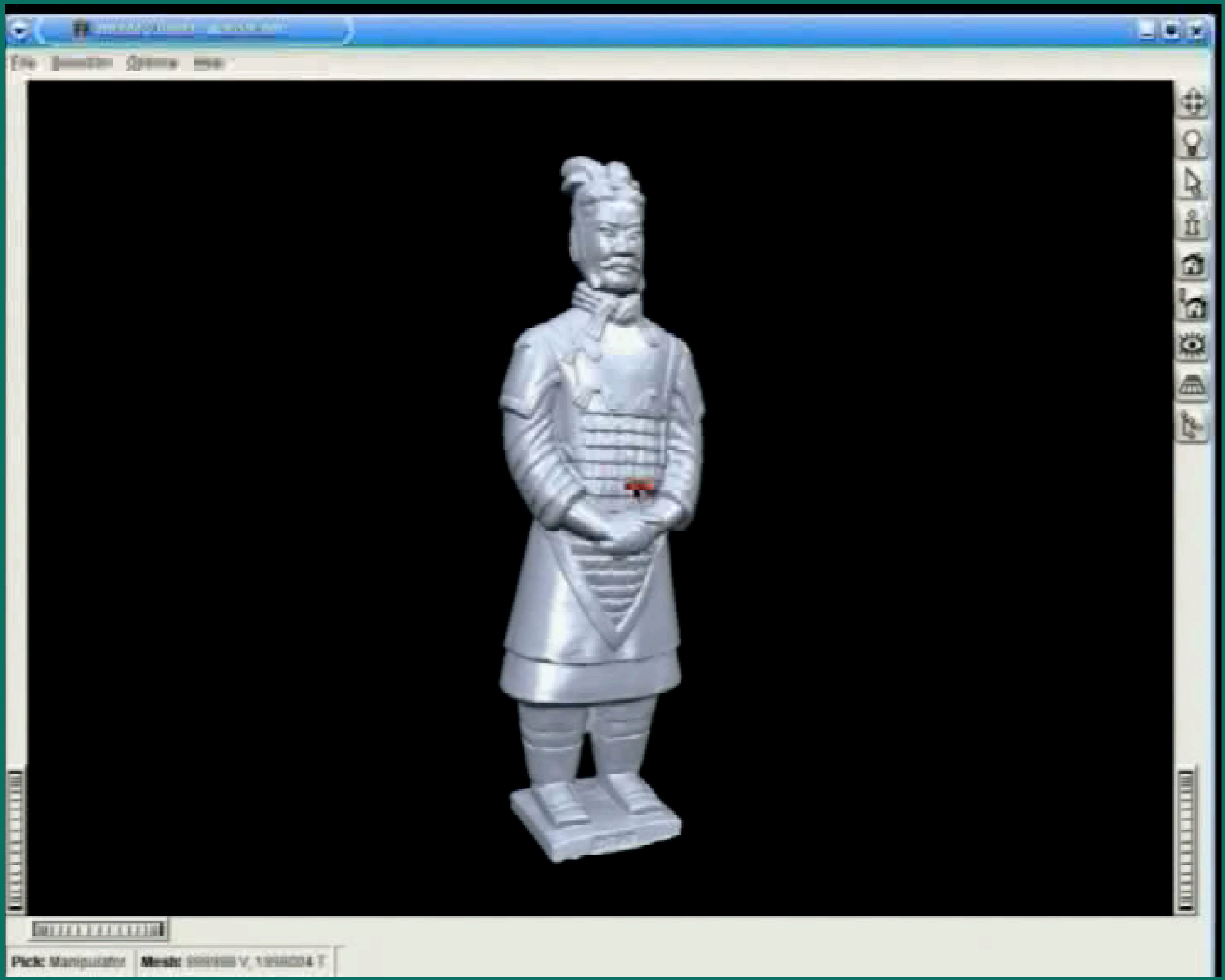
Basis precomp.: 7.5s
CPU per-frame: 0.27s

Basis precomp.: 7.5s
GPU per-frame: 0.01s

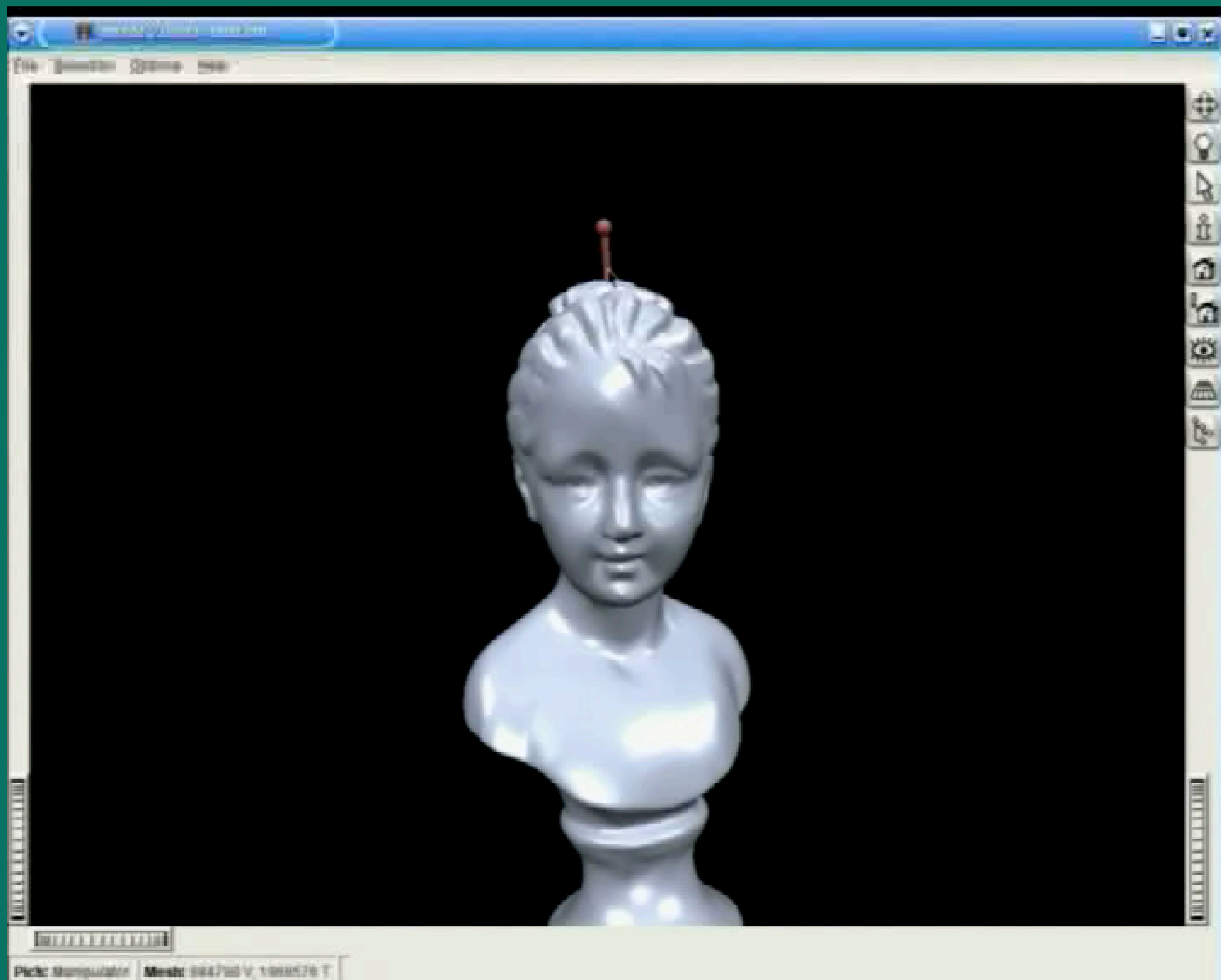
Overview

- Introduction
- RBF Modeling Setup
- Incremental Least Squares Solver
- Precomputed Basis Functions
- GPU Implementation
- Results

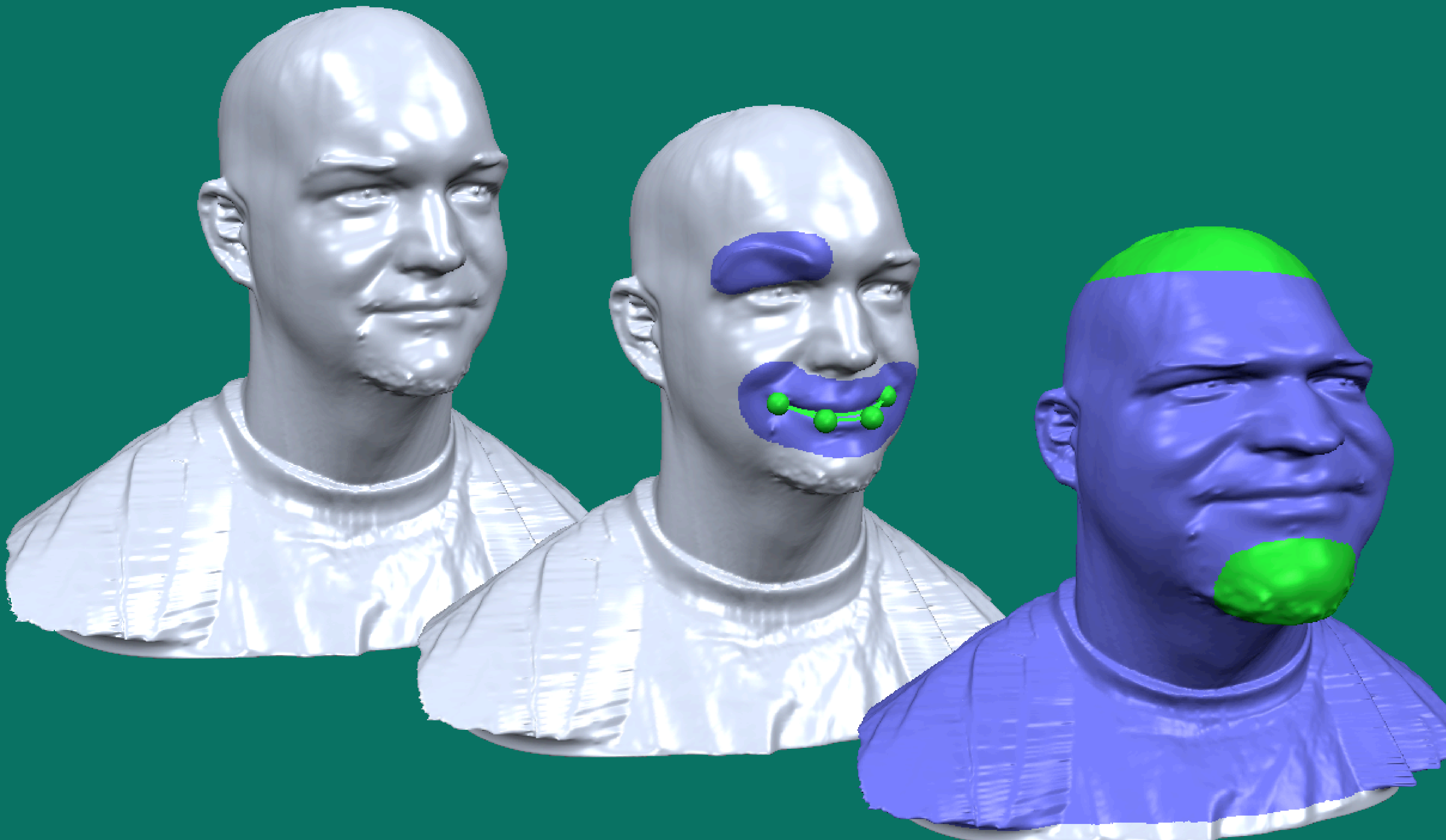
1M vertices, 355k active vertices



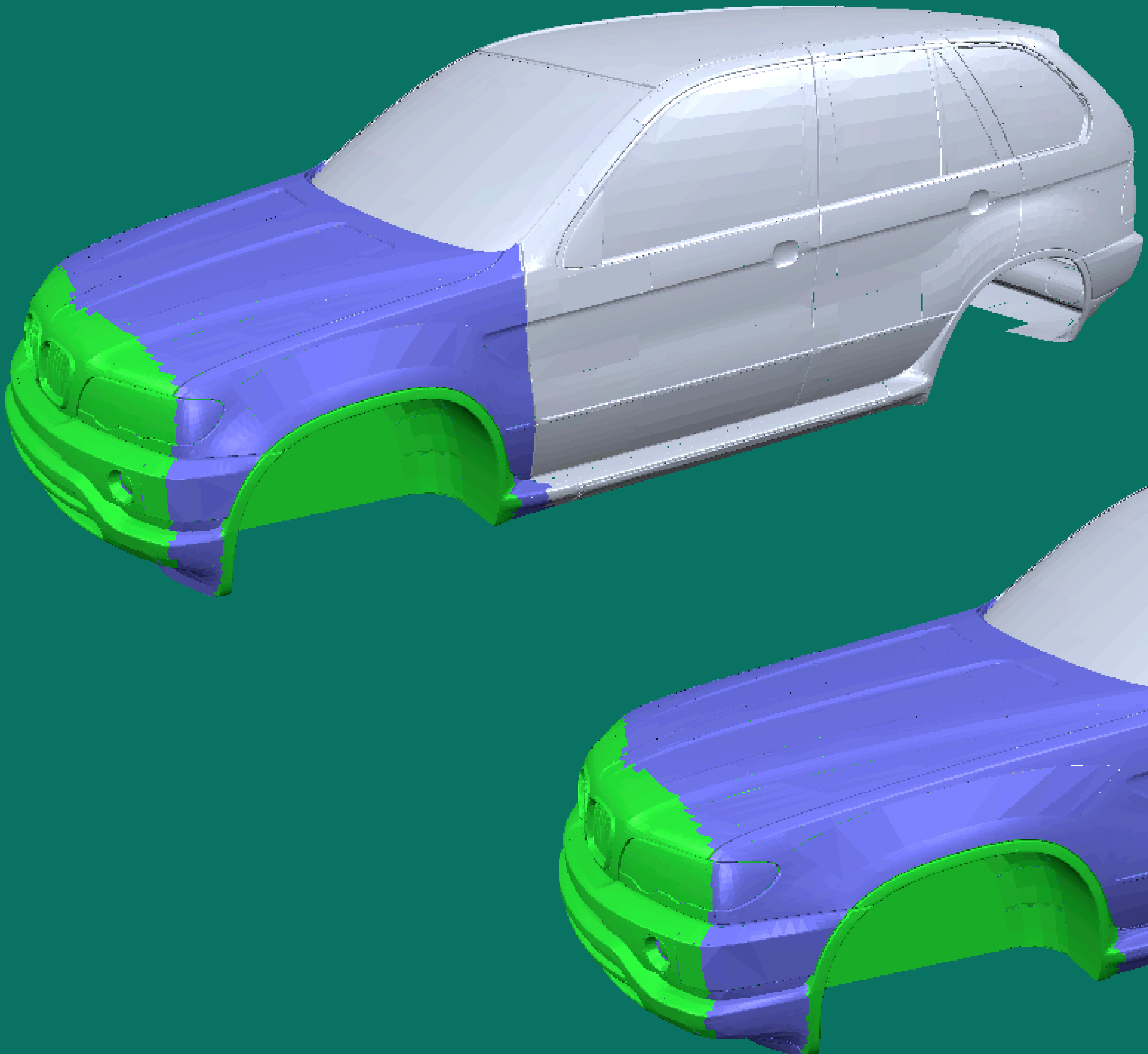
984k vertices, 880k active vertices



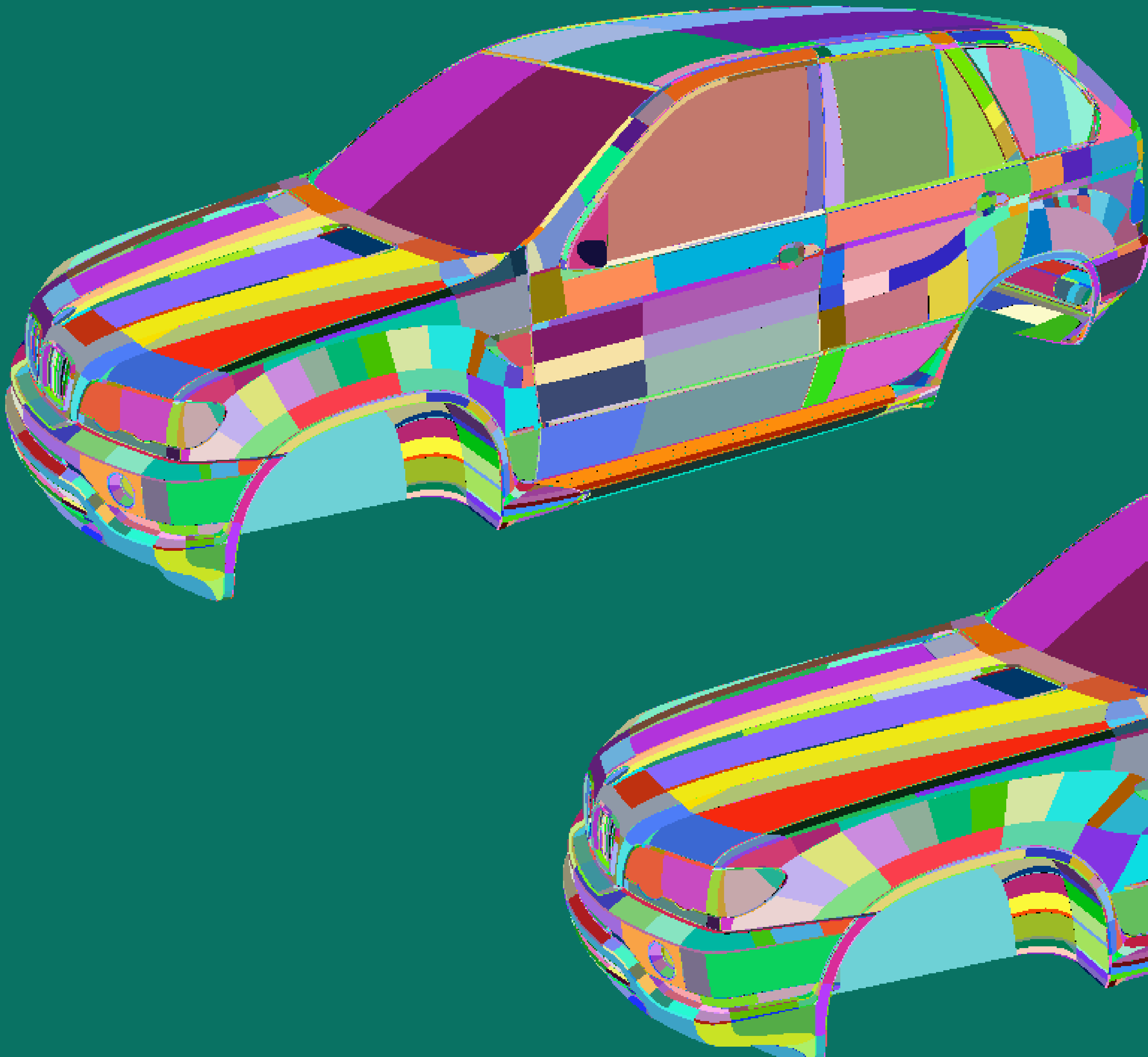
Local & Global Deformations



"Bad Meshes"



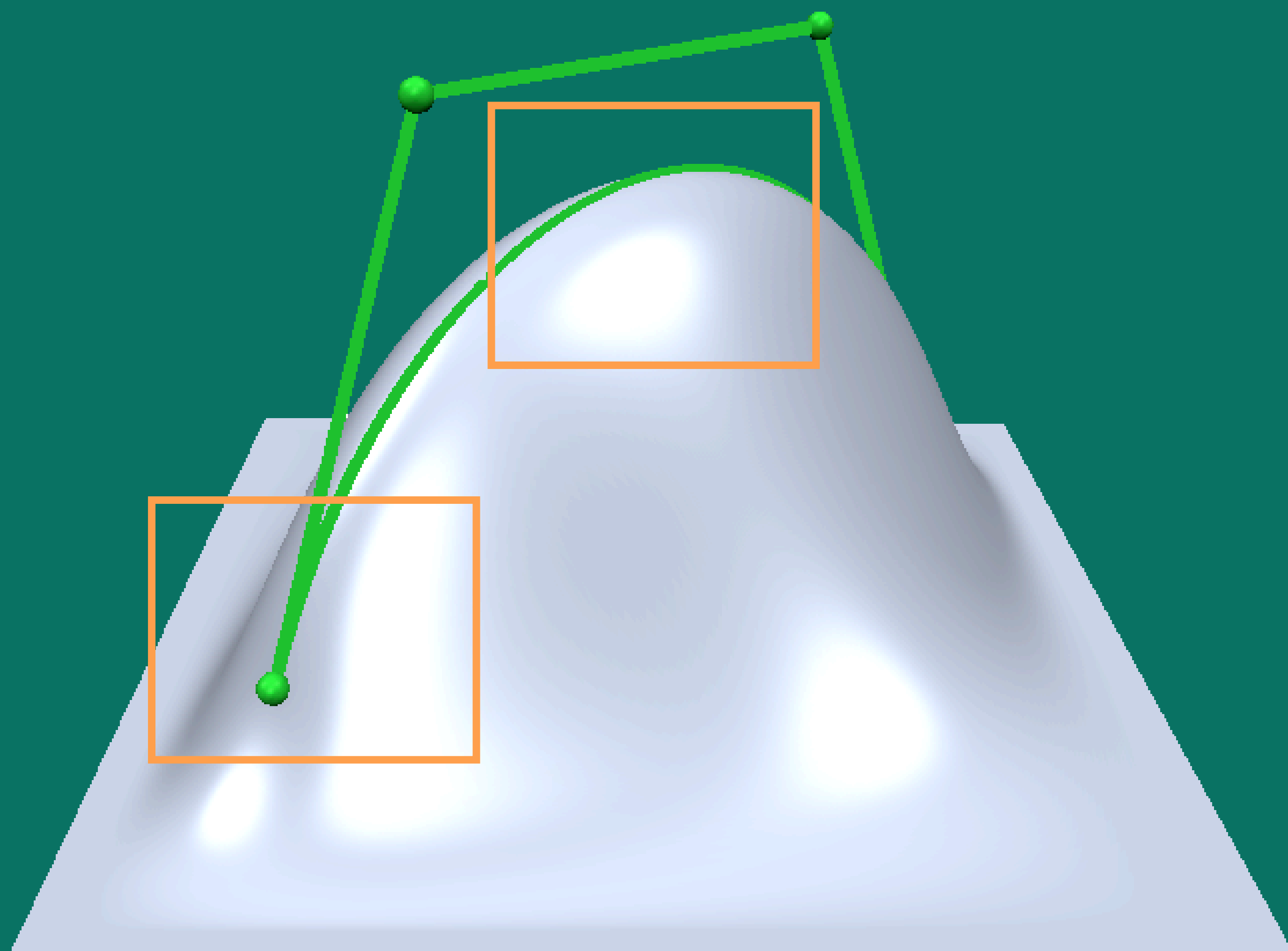
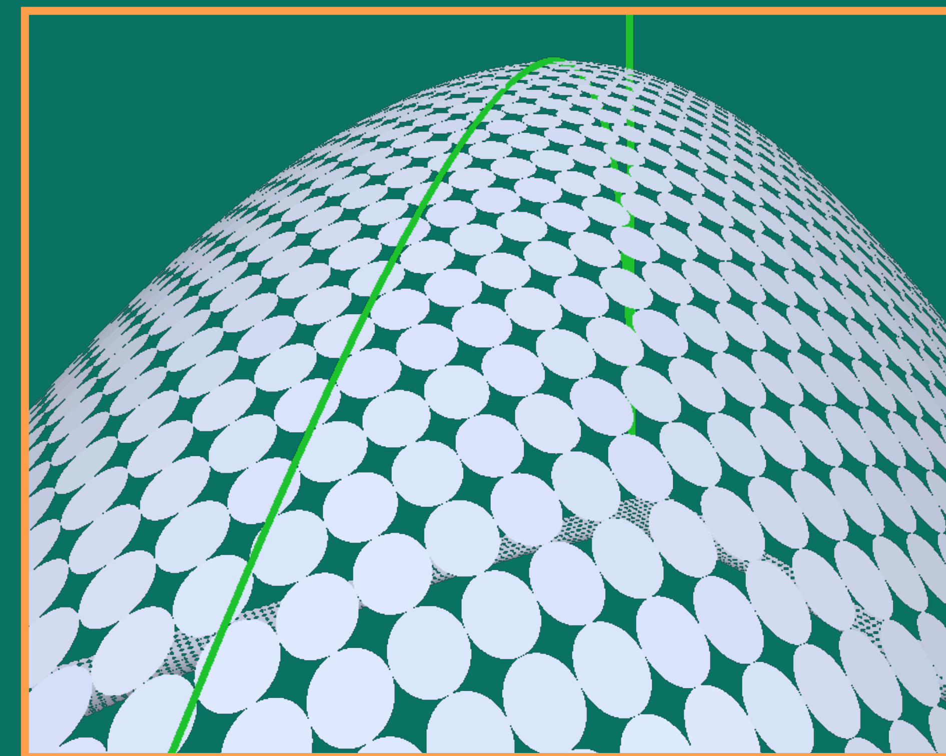
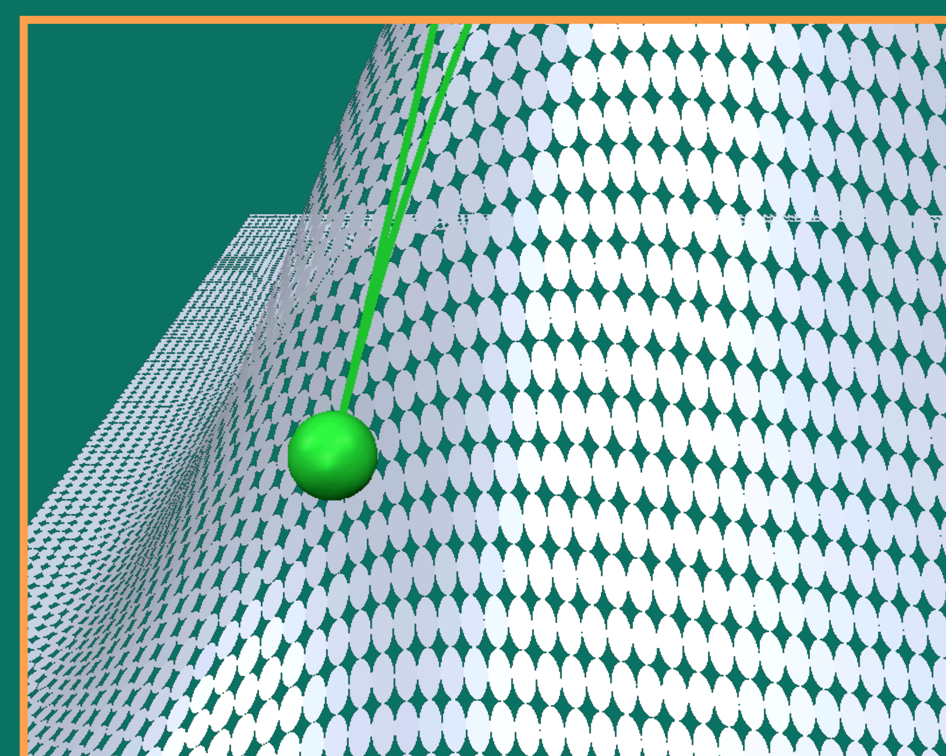
“Bad Meshes”



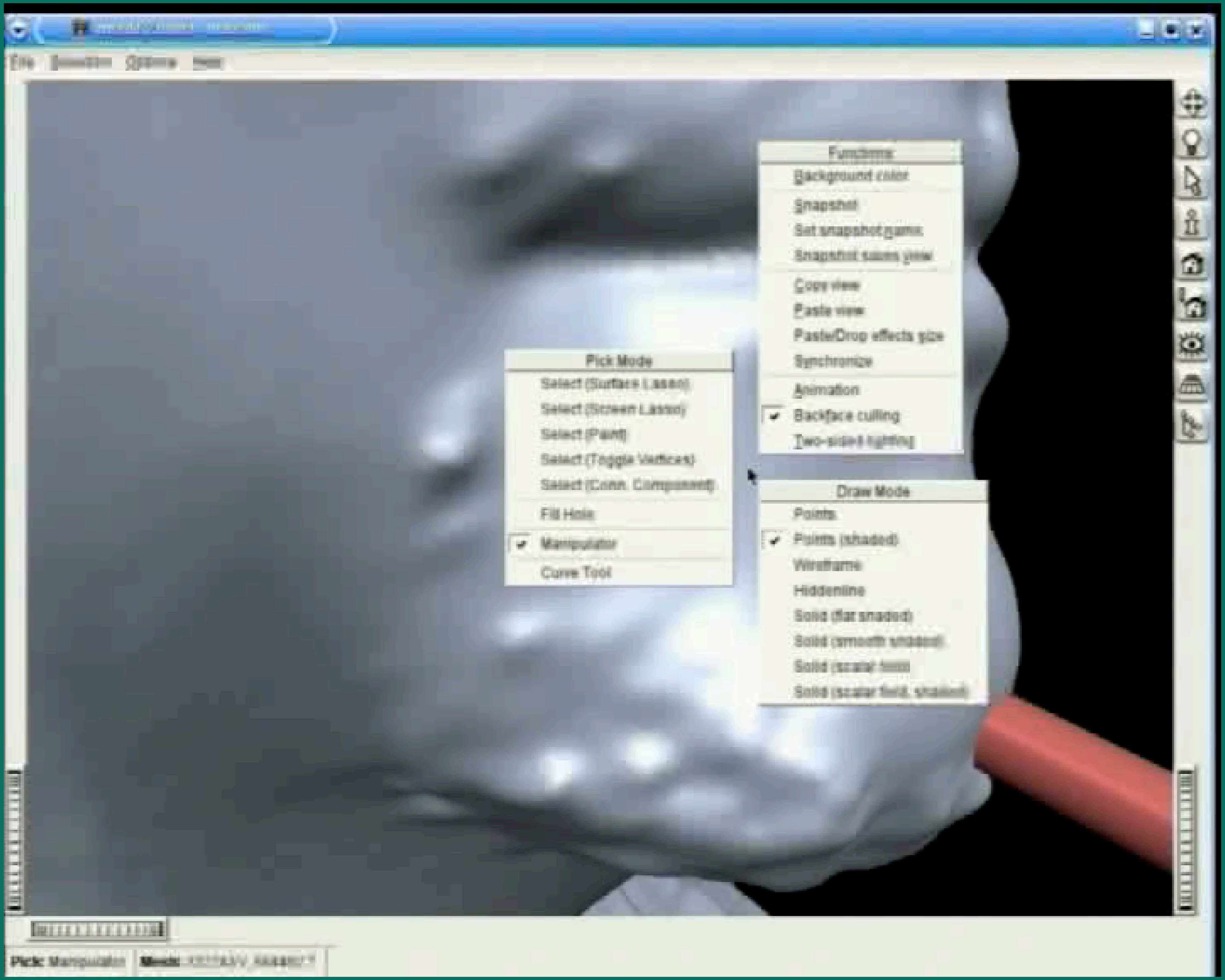
- 3M triangles
- 10k components
- Not oriented
- Not manifold

Point-Based Models

- Transform splat axes by Jacobian
- Integrates seamlessly into GPU rendering methods

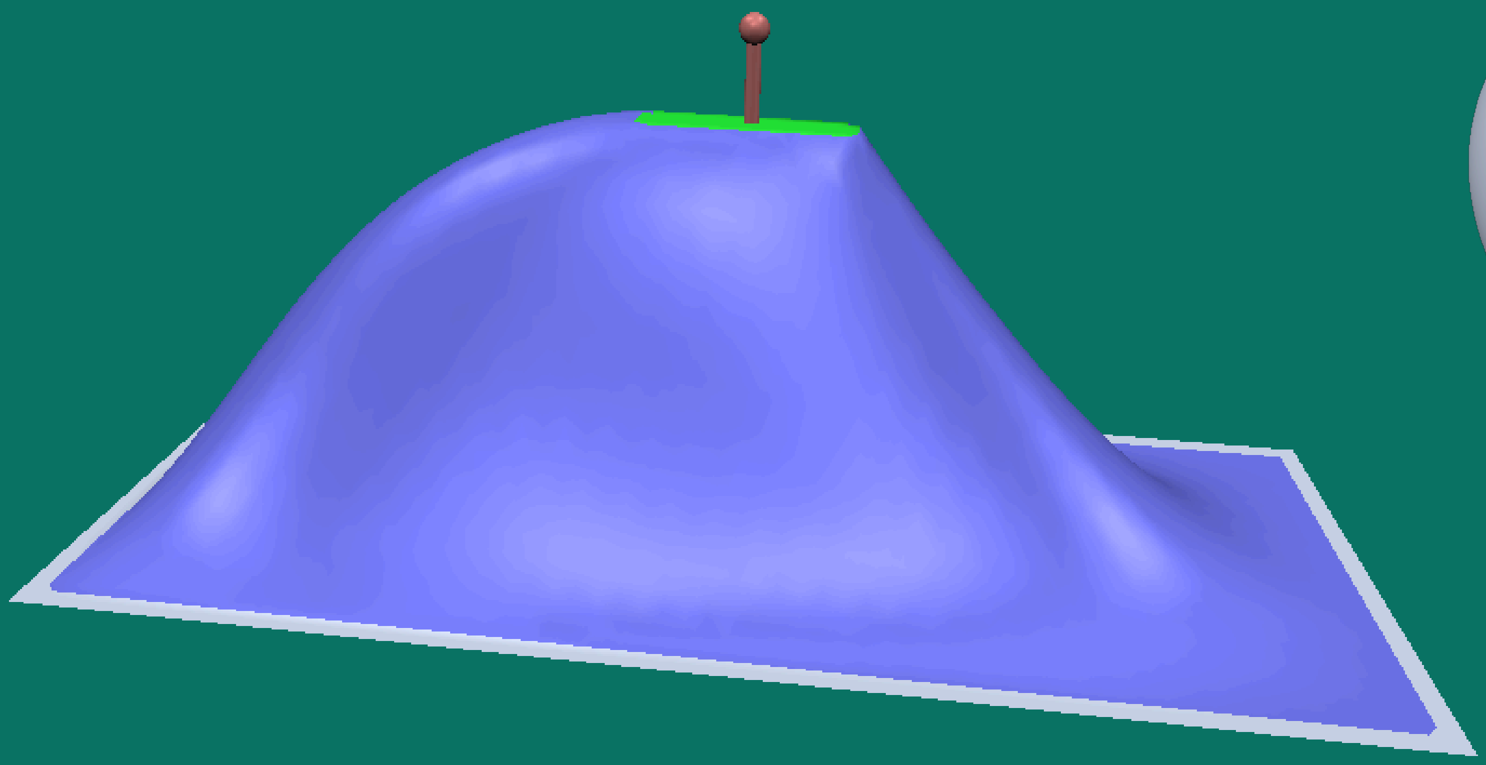
Point-Based Models

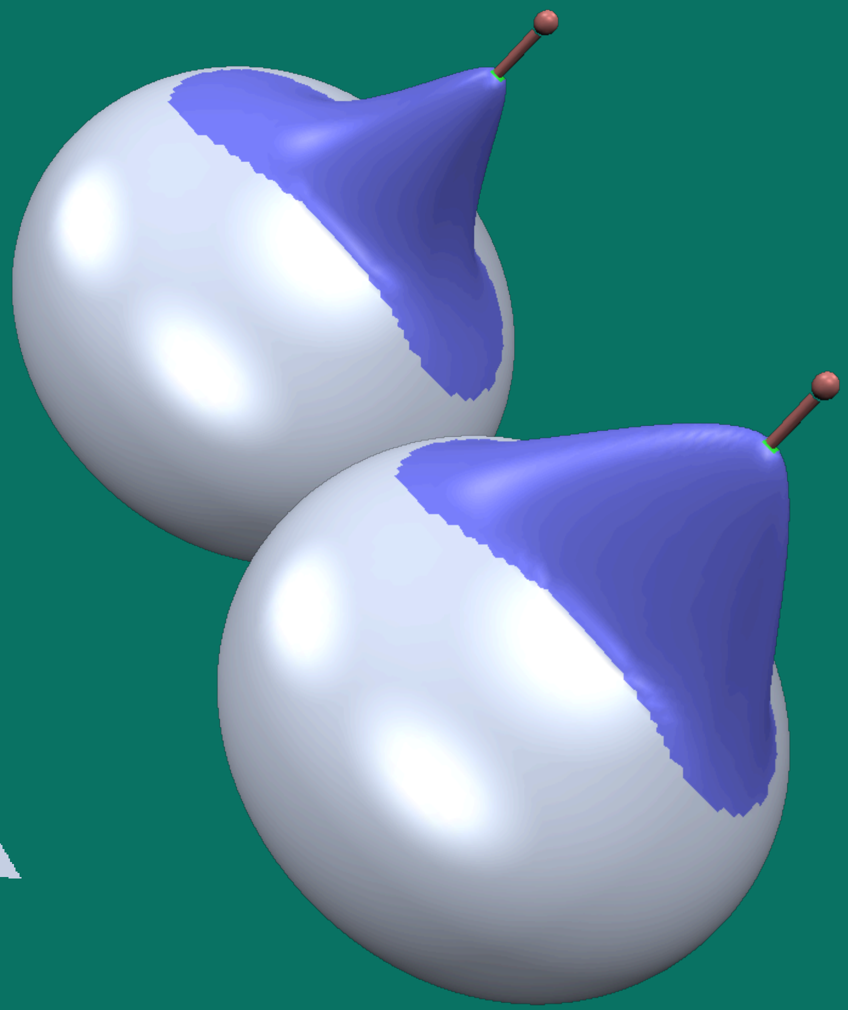


Comparison

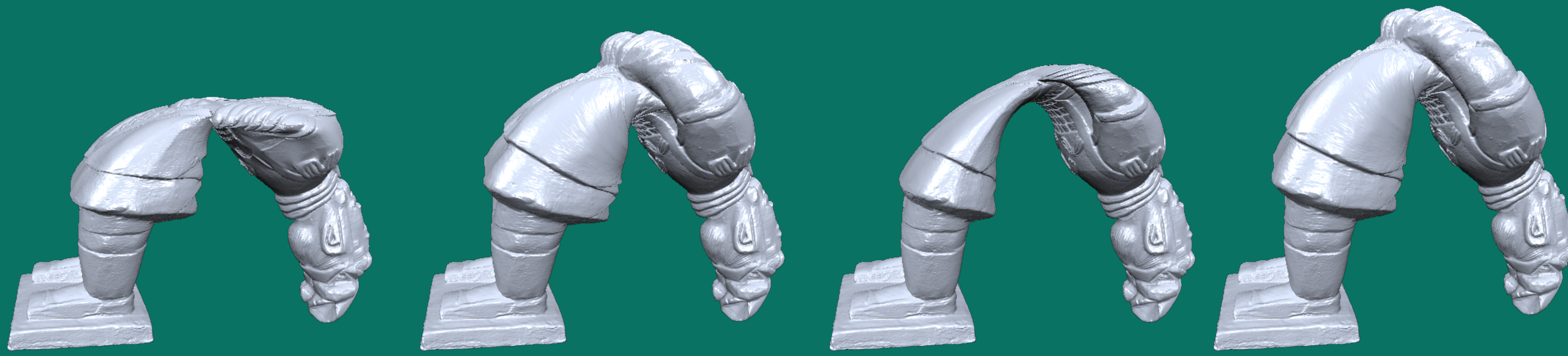
Surface-based methods offer more control

- Segment-wise boundary continuity
- Geodesic anisotropy

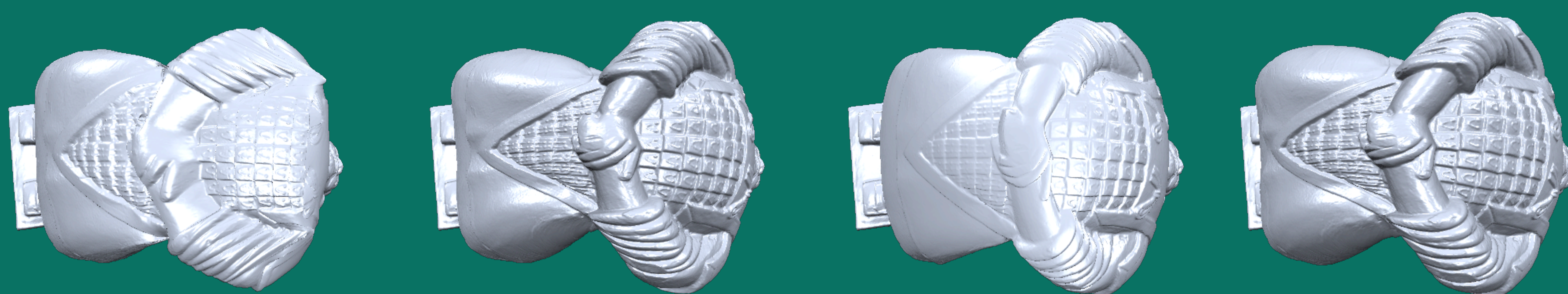




Comparison



Surface Freeform | Surface Multires | Space Freeform | Space Multires



Conclusion

- Triharmonic RBF space deformation
  - Robust & efficient
  - High fairness
- Acceleration techniques
  - Incremental QR solver
  - Precomputed basis functions
  - GPU implementation
- Real-time editing at 30M vertices/sec