Project Report – An Intuitive Framework for Real-Time Freeform Modeling Implementation

Authors of the original paper: Mario Botsch & Leif Kobbelt

Implementation by: Yissachar Abraham & Manor Zvi, cs236329, Technion, Israel

Project Github:
https://github.com/ManorZ/cs236329

Presentation and theoretical explanations:
https://github.com/ManorZ/cs236329/blob/project/An%20Intuitive%20Framework%20for%20Real-Time%20Freeform%20Modeling%20-%20Presentation.pptx

Introduction

This report summarizes our work to implement the 2004 paper "An Intuitive Framework for Real-Time Freeform Modeling Implementation".
Our implementation is written in Python and provides a GUI-based application for real-time manipulation of triangle meshes.

The fundamental problem this work thrives to solve is that freeform deformation of meshes is an extremely high-dimensional problem. The designer explores the problem space using only clicking & dragging 2D positions on the screen.
Therefore, an exact (in terms of deformation outcome) yet abstract (in terms of UI) control metaphor is needed.
Moreover, we require this controlling metaphor to be fast (real-time!), so the designer could perform multiple online steps, observe some visual feedback, and decide on the next step during the design process.

In our implementation and UI design, we followed the principles defined in the paper, both *mathematically* and in terms of user *control metaphors*.

From the *control metaphors* perspective, our application allows the user to control the deformation via a 9-DoF manipulator object, defined visually in the application by dragging vertices and selecting areas. The 9 DoF are Space (3), Pitch (3) & Roll (3).
The manipulation process is as follows:

1. The user selects an area of interest in the mesh, called support, only in which the deformation takes place.
2. The user decides a characteristic shape of the deformation: smoothness vs. stiffness
3. Lastly, the user selects a handle – a smaller area contained inside the support region, and drag it, rotate it, and scale it.

These 3 steps all together define our abstract UI and allows the user to interact with the algorithm easily, intuitively, and fast.

Mathematically, we perform the mesh deformation by minimizing a certain energy function with boundary constraints. For the practical implementation on a triangle mesh we use the discretization of the Laplace-Beltrami operator:

$$\Delta(p_i) := \frac{2}{A(p_i)} \sum_{p_j \in N(p_i)} (\cot\alpha_{ij} + \cot\beta_{ij})(p_j - p_i)$$

For a group of mesh vertices $P = [p_1 \dots p_n]^T$ it can be written in a matrix form as:

$$\Delta = M^{-1}L$$

Where M is a diagonal matrix of vertex barycentric area: $M = 2A_{bary}(p_i)$ and L is defined as follows:

$$l_{ij} = \begin{cases} \cot(\alpha_{ij}) + \cot(\beta_{ij}): & i \neq j, edge\ ij\ exists \\ -\sum_{N_1(p_i)} \cot(\alpha_{ik}) + \cot(\beta_{ik}): & i = j \\ 0: & otherwise \end{cases}$$

Raising the Laplacian to different orders control 'flavors' of deformation. Δ^1 deforms the surface like a membrane by minimizing the area of the deformed section and Δ^2 minimizes surface bending and Δ^3 minimizes surface curvature.

The minimization problem has the structure of constraints Euler-Lagrange Equation:

$$\Delta^k S(x) = 0 \quad : \quad x \in \Omega \backslash \delta\Omega$$
$$\Delta^j S(x) = b_j(x) \quad : \quad x \in \delta\Omega,\ j < k$$

But we don't need to solve this equation for all the vertices of the mesh (remembering we are aiming for real-time execution), but only for the support (defined by the user) of the deformed surface. Therefore, the problem becomes easier to solve:

$$\begin{bmatrix} L_1 & L_2 \\ 0 & I \end{bmatrix} \begin{bmatrix} s \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix},$$

Where: s are support vertices, b are boundary vertices, L_1, L_2 are slicing of the Laplacian, taking only rows acting on the support vertices and boundary vertices. It leads to a sparse equation:

$$L_1 s = -L_2 b.$$

Where L_1 is PSD and therefore the equation can be solved for s using LDL Decomposition or just matrix inversion using any numeric library – we use simple 'Pythonic' matrix inversion, for its simplicity and good-enough performance on our laptops.

To treat the boundaries of the support region with the statistic and handle regions a matrix D was introduced to the computation in order to interpolate the mesh smoothly to the boundary. This is relevant for the second and third orders. As described in the paper [1] and [2]. This parameter manipulates the higher-order Laplacian definition. Where $\lambda \in [0, k-1]$

$$\Delta^k = \Delta(\lambda_{k-1}(p) * \Delta_{k-1})$$

For first and second order

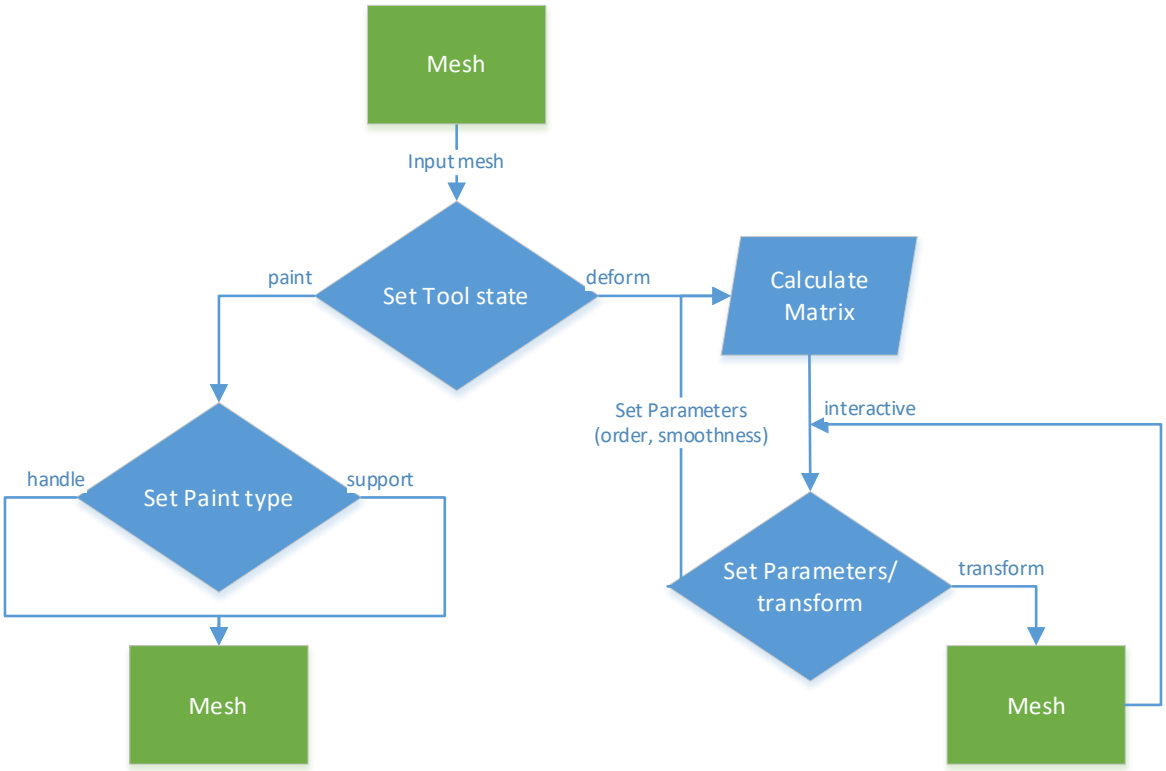
$$\Delta^2 = M^{-1}LDM^{-1}L$$

$$\Delta^3 = M^{-1}LDM^{-1}LDM^{-1}L$$

Where M is the diagonal matrix of vertex areas, L is the Laplacian Operator, and D is the λ operator

Implementation

User interface:

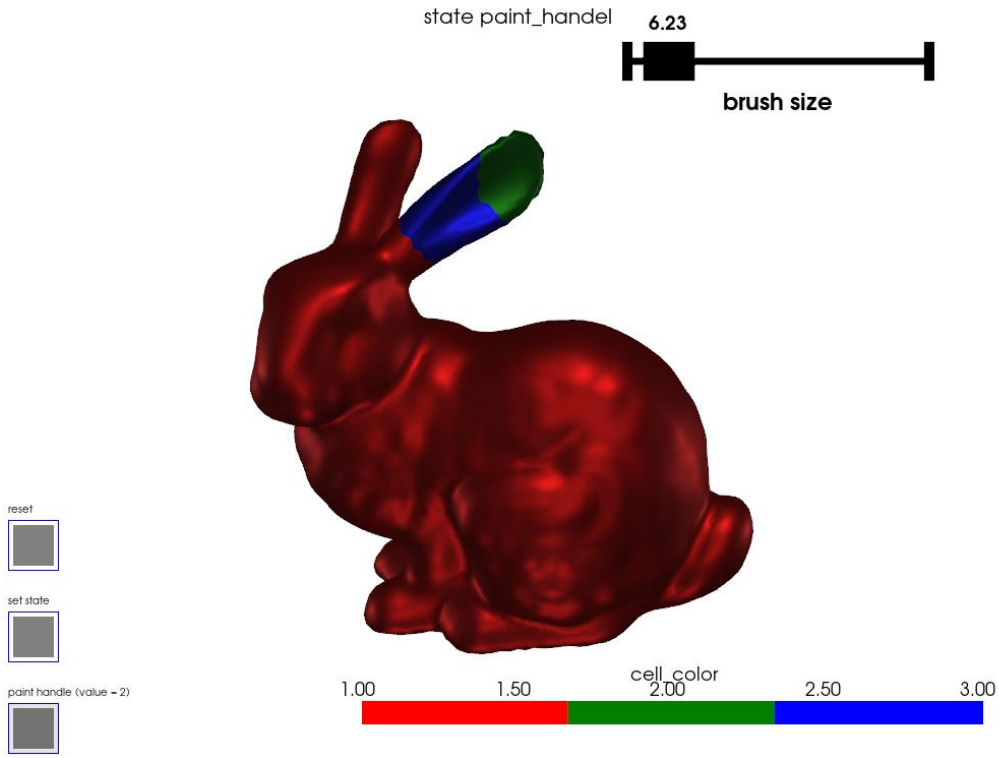


Input: triangular mesh

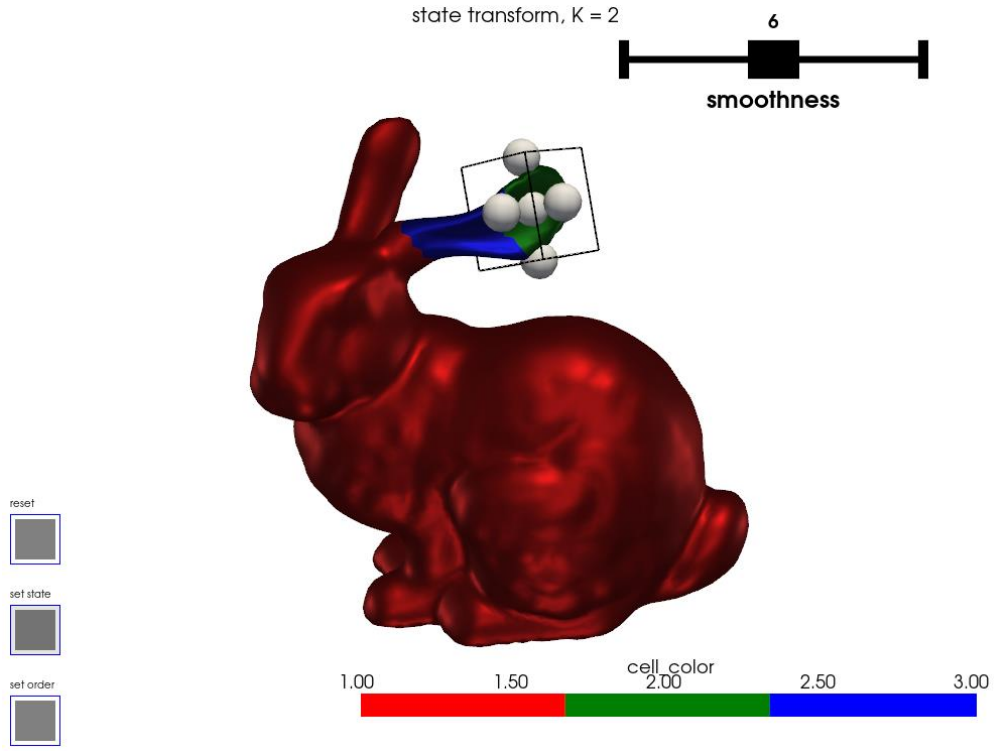
1. Paint Mode: the user selects regions on the mesh to paint upon
   - Static Region (Red) is the default color of the mesh, where no region is painted
   - Handle Region (Green) is the region witch the handle will control.
   - Support region (Blue) is the region where the mesh will deform based on the edge conditions defined by the handle and static regions

   Once the user paints the support and handles regions the deformation mode of the tool is enabled

2. Deform Mode: A handle is constructed around the handle region and the transformation matrix is calculated.
   - The user can interactively transform the handle and in effect, the support region will follow using the transformation matrix.
   - The user can adjust the parameters of the transformation matrix, which will be recalculated and applied.
     1. The Order (K), the order of the transformation can be toggled between k = 1 – 3
        For k = 1 the support region will be deformed by minimum energy, k = 2 transformed by minimum bending, and k =3 by minimum curvature.
     2. The Smoothness, can be manipulated using a slider that defines the measure of smoothness between the support region and edges (static and handle regions)

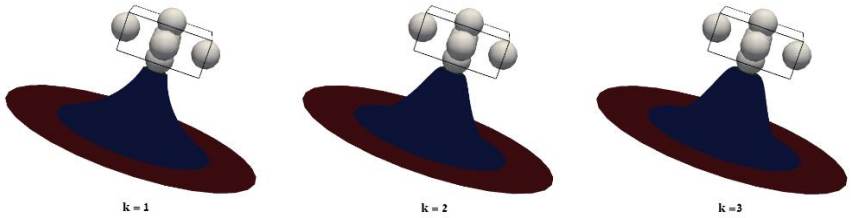


Paint Mode, Green is the handle region Blue the support and Red static

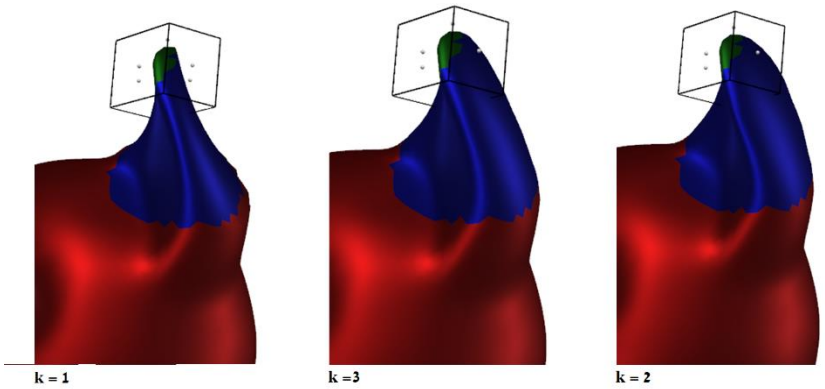


Deform Mode, The handle widget is set to transform the handle region and in effect deform the support region

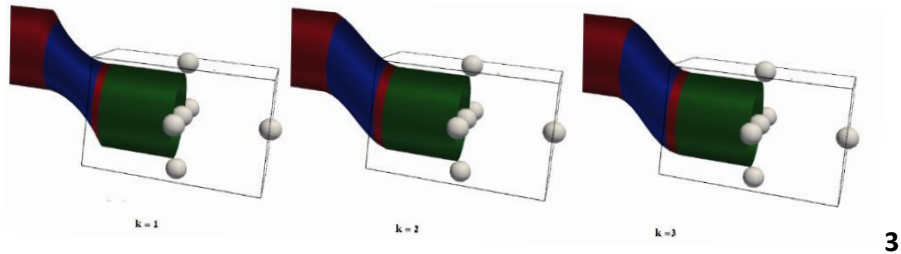
Comparisons with Original Paper



1



2



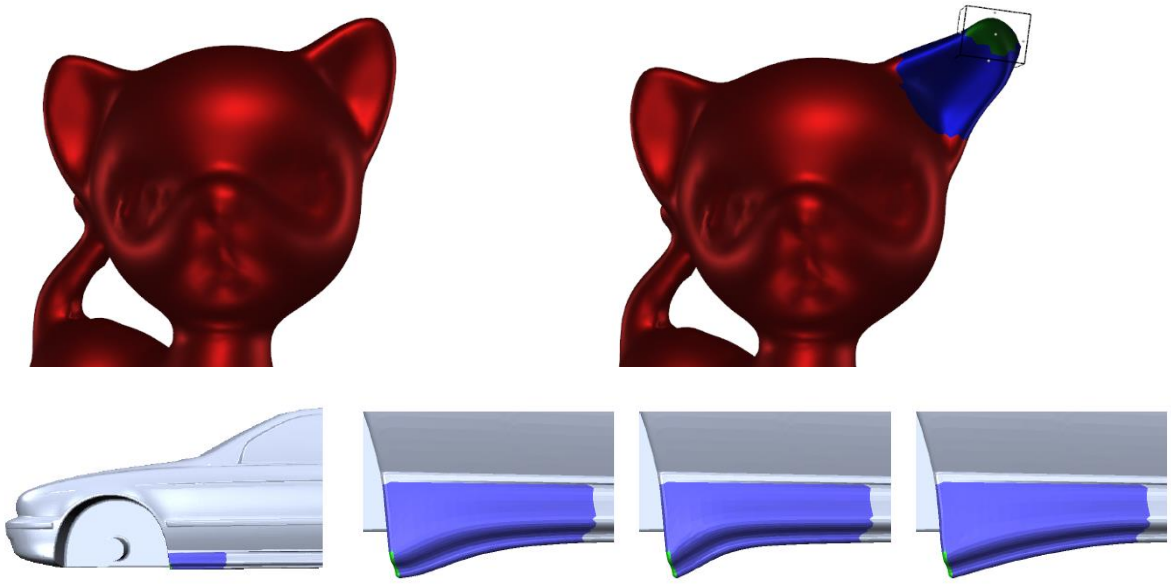
3

4

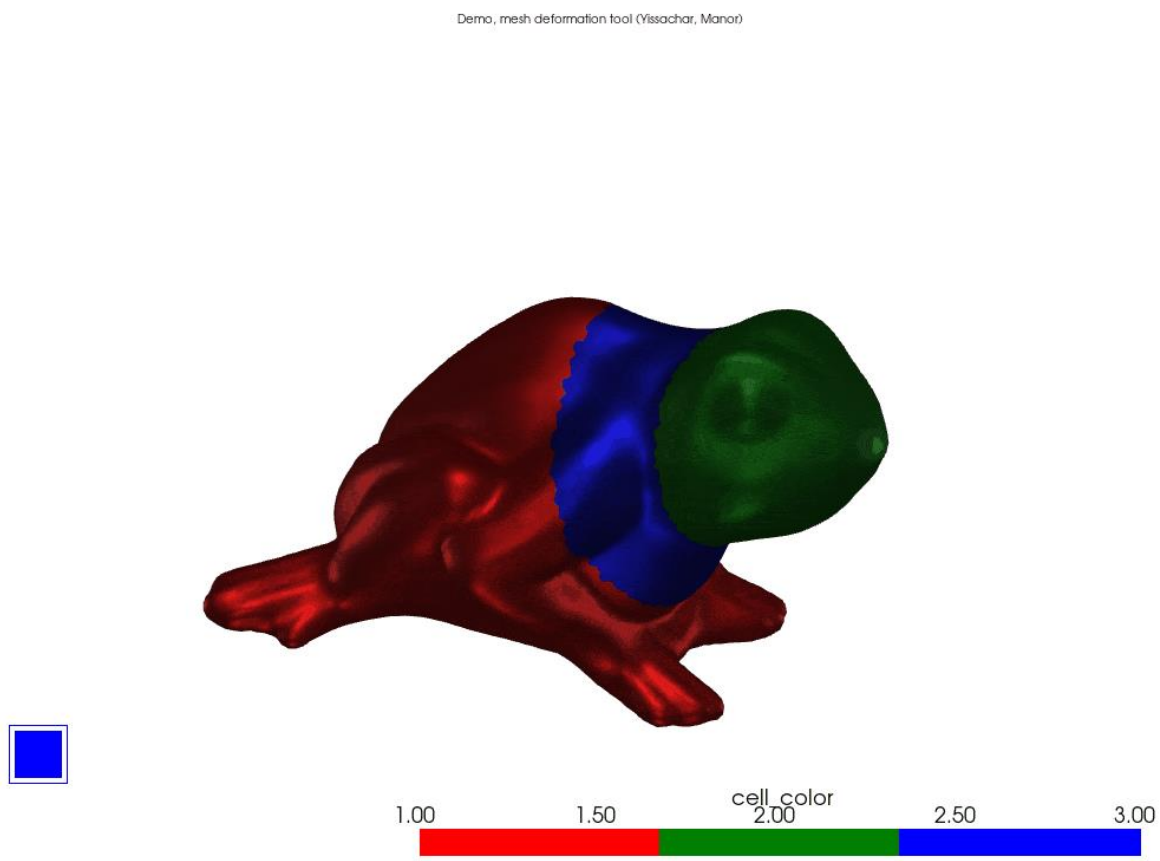
In the examples above the mesh manipulation is performed using orders k = 1, 2 and 3. For the fist order the minimum energy, second minimum bending and third minimum curvature are visually obvious. An additional parameter is the smoothness, which defines how smooth is the mesh on the edge conditions. This is obtained by manipulating the weights of the Laplace operator.

Image 4 is illustrated in the paper

Results



1. Above plot using our implementation. 2. Plot from the paper [1], illustrating the manipulation on a mesh while maintaining static regions.



A link to a demo where the tool animates a mesh

(https://raw.githubusercontent.com/ManorZ/cs236329/project/data/plots_mesh_tool/mesh_demo.gif)

References:

1. Botsch, Mario, and Leif Kobbelt. "An intuitive framework for real-time freeform modeling." *ACM Transactions on Graphics (TOG)* 23.3 (2004): 630-634. \Discrete di_erential-geometry operators for triangulated

2. Kobbelt, Leif, et al. "Interactive multi-resolution modeling on arbitrary meshes." *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998.