Optimizing UI Layouts for Deformable Face-Rig Manipulation

JOONHO KIM, University of Toronto, Canada
KARAN SINGH, University of Toronto, Canada

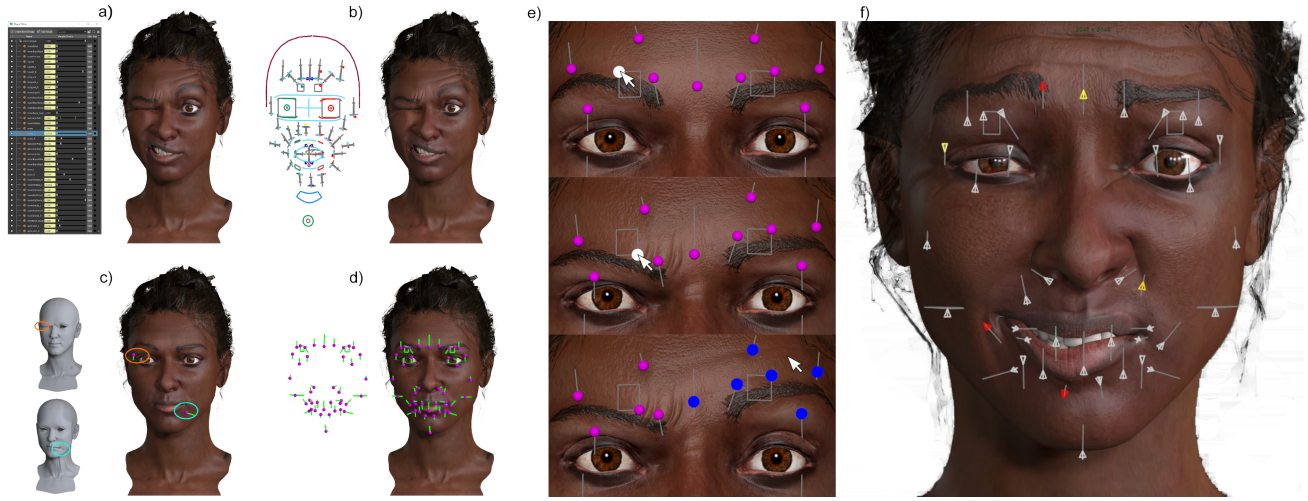


Fig. 1. Expressive deformable models like faces have many independently controlled rig parameters (a). Animators increasingly hand-craft in-situ UI layouts for such face-rigs (b). We distill their design choices into a set of layout principles (c). We then present an algorithm to produce optimal in-situ UI layouts that match animator expectation (d); enabling a direct on-shape deformation interface (e), where rig parameters can be both independently manipulated (cursor on white control, top to middle), and best-fit computed (blue controls) to match the direct manipulation of the shape (cursor on face, bottom) [Lewis and Anjyo 2010]. Our in-situ UI layouts can be further customized and refined by animators (f).

Complex deformable face-rigs have many independent parameters that control the shape of the object. A human face has upwards of 50 parameters (FACS Action Units), making conventional UI controls hard to find and operate. Animators address this problem by tediously hand-crafting in-situ layouts of UI controls that serve as visual deformation proxies, and facilitate rapid shape exploration. We propose the automatic creation of such in-situ UI control layouts. We distill the design choices made by animators into mathematical objectives that we optimize as the solution to an integer quadratic programming problem. Our evaluation is three-fold: we show the impact of our design principles on the resulting layouts; we show automated UI layouts for complex and diverse face rigs, comparable to animator hand-crafted layouts; and we conduct a user study showing our UI layout to be an effective approach to face-rig manipulation, preferable to a baseline slider interface.

CCS Concepts: • **Computing methodologies** → **Animation**; **UI**.

Additional Key Words and Phrases: facial animation, deformable rigs, UI.

Authors' addresses: Joonho Kim, University of Toronto, Toronto, Canada, joonho@dgp.toronto.edu; Karan Singh, University of Toronto, Toronto, Canada, karan@dgp.toronto.edu.

0730-0301/2021/8-ART172 $15.00
https://doi.org/10.1145/3450626.3459842

ACM Reference Format:
Joonho Kim and Karan Singh. 2021. Optimizing UI Layouts for Deformable Face-Rig Manipulation. *ACM Trans. Graph.* 40, 4, Article 172 (August 2021), 12 pages. https://doi.org/10.1145/3450626.3459842

1 INTRODUCTION

Character rigging is an art that provides a number of meaningful interactive controls (like strings of a puppet) with which to bring a character to life [Allen and Murdock 2008]. The creative flow and expressivity of a puppet-master (animator) is impacted not only by how the strings manipulate a puppet (control parameters), but by the relationship between the strings and the puppet-master (control UI and its layout). While much CG research has focused on the anatomic, geometric, and dynamic aspects of character rigging, research into the layout and interactive control of rig parameters is relatively less explored. This paper thus, introduces the problem of creating animator-optimal, in-situ, layouts of UI controls to facilitate the manipulation of a deformable character rig (Figure 1).

Motivation

Professional character animation rigs have hundreds of interactive controls. A Facial Action Coding System (FACS)-based face-rig [Ekman 1997] for example has upwards of 50 independently controlled Action Units (AUs) that define a facial expression. Modern face rigs often go well beyond FACS to model sub-muscular behavior, simulate aging, or represent families of faces [Seymour 2016, 2018], resulting in rigs with hundreds of parameters [Seymour 2019].

These rigs are animated by traditional keyframing [Navone 2020], performance capture [Seymour 2019; Tewari et al. 2017], or even speech audio [Edwards et al. 2020]. While rigs are increasingly driven by performance capture, the role of an animator remains as important as ever: the popular genre of stylized animation is still tediously keyframed [Navone 2020], and performance captured animation is still cleaned-up and embellished by animators working with interactive deformable rigs [Seymour 2018]. Film rigs are also ephemeral, with base rigs continually evolving to account for context specific corrections and shot customization [Li et al. 2013].

The default UI for such rigs in commercial animation software like *Maya* is a disparate and long scrollable list of deformation parameter sliders, only a fraction of which might be concurrently viewable (Figure 1(a)). Interacting with these sliders is woefully inadequate: knowing which AUs are needed to effect a desired change to the face is not obvious. These AUs must then be found by name in the slider list and manipulated while shifting visual focus between the 2D UI and the 3D face.

Manipulating such a large number of control parameters is challenging. Direct manipulation techniques such as [Lewis and Anjyo 2010] inverse-fit a configuration of rig parameter values, whose deformed shape best-matches a user manipulated target shape. Low-dimensional data-driven manifolds allow users to rapidly explore a meaningful subspace of the high-dimensional rig parameter space [Abdrashitov et al. 2020]. While animators appreciate the simplicity and efficiency of such approaches, they also demand independent control of rig parameters for the fine control needed to showcase their skills (similar to the complementary use of both Forward and Inverse Kinematics in skeletal animation) [Osipa 2010] (Figure 1(e)). As a result most riggers and animators meticulously hand-craft in-situ UI layouts to drive their character rigs (Figure 1(b)). Such in-situ/on-screen interfaces (Figure 2) are also increasingly prevalent in game engines, and mixed reality platforms: to support expressive character posing and animation for film, games, social media and avatar customization.

Each UI control in Figure 1(b) was hand-crafted in a 2D frontal view, designed to largely emulate the view-projected deformation trajectory[1] of a representative shape vertex, across extreme values of a corresponding rig parameter. The UI layout took the rigger about a day to create, intuitively picking a visually meaningful trajectory for each parameter, and laying it out respectful of symmetry, occlusion, and maximizing the use of space around the shape. Creating such a layout by hand gets significantly harder with 3D curve trajectories (Figure 16) and evolving rig parameters (Figure 12). Hand-crafted rig layouts are also brittle to layout re-targeting across characters with diverse proportions, topology, and rig parameters.

Our primary contribution is thus: an analysis of the artistic insights in creating hand-crafted face-rig UI layouts and, subsequently, an optimization algorithm that fully automates the creation of such UI layouts. Our resulting UI layouts have many applications: they validate manual layout design choices and can optimally complete a partially hand-crafted rig; provide an in-situ UI layout for one-off shot-specific/layered/corrective shapes common in performance capture; provide in-situ UI layouts for novices, and one-off or non-standard characters with stylized proportions (animal in Figure 18); and generalize to arbitrary parametric deformations (Figures 16).

[1]The blendShape weight parameters in Figure 1 produce linear trajectories, but in general this trajectory is a parametric 3D curve.

Problem Statement and Overview

We observe that the feel of direct manipulation can be provided by mapping UI handles to the deformation of a representative vertex of the shape. Defining an in-situ layout can thus be cast as a mapping $L : C \longrightarrow V$, where C is a set of rig parameters/controls, and V a set of 3D shape vertices. We then formulate the various aesthetic and usability properties that hand-crafted layouts imbibe (Figure 1(c)) as an extendable set of energy terms, defined for any given layout L. The optimal in-situ layout is thus found by computing the energy minimizing control to vertex mapping (Figure 1). While our approach is focused on face-rigs (Figure 17, 18), our technique is designed to be generally applicable to parametric deformations (Figure 16).

A review of related work (Section 2), is followed by a discussion of our overall problem space, and development of an extendable set of in-situ UI layout design principles (Section 3). Section 4 presents the details of our optimization algorithm. Section 5 presents a user study comparing our UI layout to a baseline slider interface, and artist hand-crafted rig UI layout. Further evaluation, limitations and directions for future work are presented in Section 6, 7.

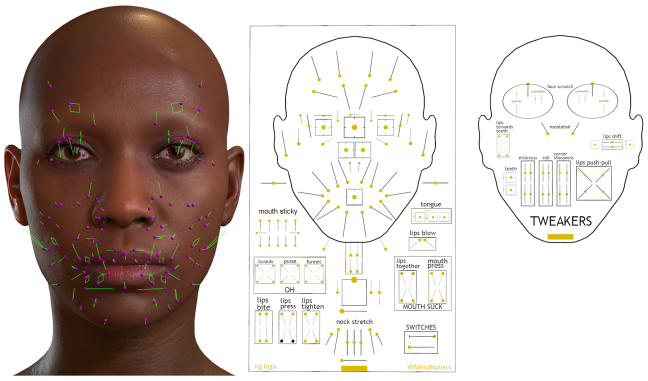


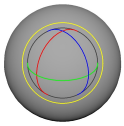
Fig. 2. Our rig UI layout alongside hand-crafted UI layout for the 24K vertex Metahuman with 142 parameters ©Epic Games, Inc.

2 RELATED WORK

The problem of rigging and motion control for interactive character animation is at least three decades old [Badler et al. 1990]. We focus on character rigging: the interactive set-up that allows animators to manipulate parameters that control the deformation of character geometry. The majority of rigging research concerns algorithms that actually deform a 3D model, based on configurations of rig parameters. Work relevant to this paper instead addresses the user interface between rig parameters and the animator, broadly classified under: deformation proxies and widgets, high-dimensional UIs, and direct manipulation of deformable shape.

Deformation Proxies and Widgets

UI widgets are visual 3D elements designed to provide an in-situ interactive interface to manipulating objects and aspects of a virtual scene [Bier 1987]. Widgets are typically hand-designed to capture the form/function of parameters they control, like a rotation arc-ball.



A variety of general frameworks for 3D widget design have been developed [Stevens et al. 1994], including approaches to interactively combine simple widgets into composite widgets for complex 3D object manipulation [Schmidt et al. 2008]. We automatically create such a composite widget for deformable shapes. Deformable objects are often interactively manipulated using proxy objects like lattices [Sederberg and Parry 1986] or wires [Singh and Fiume 1998], whose direct manipulation is intrinsically responsible for the resulting shape deformation. Our approach can augment such deformable proxies by automating the UI layout of various deformation parameters. While our optimization objectives are visual design and interaction based [Agrawala et al. 2011], we also draw inspiration from research on computing optimal marker configurations to accurately capture facial deformations [Le et al. 2013].

High-Dimensional User Interfaces

Exploring high-dimensional design spaces in a creative context is a challenging task. Data-driven techniques, often aimed at novice users, reduce the dimensionality of the parameter space [Kry et al. 2002], employ design galleries of good representative examples [Brochu et al. 2010; Gibson et al. 1997], or use crowd-sourced visualizations usually to aid novice users [Koyama et al. 2014; Talton et al. 2009]. Recently data-driven manifolds of deformable shape [Bailey et al. 2020] have also been used in the context of rapidly posing expressive faces [Abdrashitov et al. 2020]. Expert animators are not intimidated by a large number of parameters, as long as their interface is streamlined for efficient interaction that maintains their creative flow. Hand-crafted in-situ rig UI layouts (Figure 3) exemplify such an interface. This paper is the first to automatically create such rig UI.

Direct Manipulation of Deformable Shapes

Sketch-based and direct manipulation alternatives can streamline spatially goal-directed animation [Gleicher 1992] and reduce the tedium of sequentially manipulating individual parameters of a high-dimensional rig. Inverse Kinematics techniques (IK) for constrained goals, like walking or grasping [Parent 2012], or aesthetic line-of-action sketching [Guay et al. 2013] position multiple joint parameters of a deformable character using high-level animator input. Facial domain systems like Face-Poser [Lau et al. 2009] use screen-space input as 2D point, stroke, and curve constraints with a deformation prior learned from a prerecorded facial expression database. Other approaches [Miranda et al. 2011; Sucontphunt et al. 2008] use 2D drawing and curve-editing to drive 3D marker-based facial expressions. Performance capture of the face or body [Williams 1990] is a popular approach to driving a 3D deformable shape from markers or video of an actor. Much of this work is not animator-centric, but techniques such as Weise et al. [2011]; Zell et al. [2017] can fit performance capture to the parameters of a deformable rig. In this spirit direct manipulation methods [Cetinaslan and Orvalho 2018; Lewis and Anjyo 2010] compute rig parameters from the manipulation of vertices on a 3D face, with varying degrees of local spatial control [Tena et al. 2011]. Direct manipulation techniques perfectly complement our work, for the same reason that both inverse and forward kinematics are necessary for animation. Figure 1(e) and accompanying video shows our combined interface, where selecting a UI element controls specific parameters (top, middle), and selecting the shape elsewhere sets parameters best-fit to direct manipulation.

3 DESIGN PRINCIPLES

Direct, in-situ interfaces keep users immersed and focused on their creative task in a 3D scene. Our general problem space is vast, with many design dimensions, including:

Rig parameter structure and mapping: clusters and hierarchies of rig parameters, and their mapping to UI elements can provide a trade-off between visual clutter and coarse-to-fine parameter interaction. Additionally, parameter structuring can greatly reduce the computational complexity of layout optimization.

UI element form: the geometric shape of a UI element/widget is both: an interaction handle for its rig parameter(s), and a visual proxy for the resulting deformation (for eg. the straight sliders convey a linear blendShape trajectory, and the eyelash/chin curves convey the deformation due to *squint*/*jawOpen* parameters in Figure 1(b)).

UI element function: the interaction workflow with UI elements can address design issues ranging from UI clutter, to providing visual feedback and control over deformation properties and constraints.

UI element layout: a successful layout, for a given mapping of rig parameters to UI elements, should be optimized for visual meaning, aesthetic, and ease of interaction.

While we use parameter clustering to aid layout optimization and show three different rig parameter to UI element mappings, this paper remains largely focused on UI layout, leaving detailed explorations of other design aspects to future work.

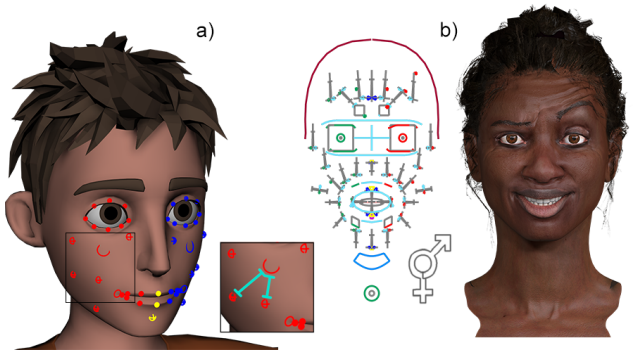


Fig. 3. Hand-crafted rig UI layouts: the control UI is a mix of manipulation handle shapes, line/curve sliders, and patches in 3D (a) or 2D (b). The layouts are symmetric, well spread in space, minimally occluded and correspond directly to the shape deformation. The layouts can be placed on (a) or off the face (b) and UI elements for global rig parameters like gender, are typically placed beside the rig UI (b). Ray Character Rig by CGTarian ©UAB MOCAP.LT.

We distill an understanding of animator intuition in rig UI creation from conversations with animators and an analysis of hand-crafted rigs (Figure 3 and near a dozen proprietary professional rigs). In all in-situ rigs the manipulation handle of a parameter was aligned to at least approximately track the deformation of a point on the face resulting from manipulating that parameter. In many instances 2D handles matched the deformation projected in a front view (Figure 7, 17).

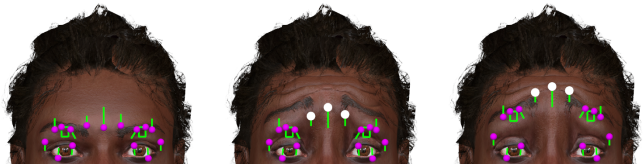


Fig. 4. UI elements on the eyebrows (left) can remain static (middle) or interactively adapt (right) to parameter manipulation (white controls).

3.1 Feeling of Direct Manipulation

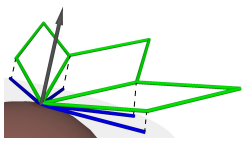
We thus formalize the UI layout problem as a mapping $L : C \longrightarrow V$, where C is a set of m rig parameters $c_1..c_m$ and V the set of n 3D model vertices $v_1..v_n$. Let $P = \{p_1..p_m\}$ be a configuration of the rig parameter values, where p_j lies between min_j and max_j values for control parameter c_j. Let $l_j \in \{1..n\}$ be the vertex index that c_j is mapped to under some layout $L = \{l_1..l_m\}$, and $d_j(t)$ the parametric 3D trajectory of vertex l_j, as it deforms under some parameter configuration P (for eg. the default parameter values) with only c_j changing from $t = min_j$ to $t = max_j$. For the blendShape weight parameters seen in Figure 1, this trajectory d_j is a straight line (in general $d_j(t)$ is a parametric 3D curve).

Direct manipulation dictates that the UI element E_j for any control c_j should be shaped by d_j. The UI elements in Figure 1,2,17 are mostly 3D line sliders, but can also conform in shape to the region being deformed (for eg. eyelash/lip/chin in Figure 1(b)). We thus need to support general element shapes and trajectories, like parametric curves, patches, and fans (Figure 5), and our goal is to find a layout L, whose UI element shape and trajectory optimize certain functional and aesthetic criteria.

Note that the trajectory $D = \{d_1, ..d_m\}$ for any layout L, also depends on the parameter values P. Conceptually we can either find an optimal layout for some default parameter configuration, or constantly recompute an optimal layout as parameter values P change. Animator consultations and HCI principles [Gajos et al. 2006] confirm that persistent layouts become familiar and more predictable for users. Optimal layouts, like hand-crafted layouts, are thus only computed once, for some neutral/default or user-defined parameter configuration P. While the layout mapping remains fixed, all UI elements should deform to remain attached to the interactively manipulated shape (Figure 4), to provide a feeling of direct manipulation.

3.2 UI Elements

Figure 5 shows various linear UI elements. A *single-parameter slider* independently manipulates a rig parameter. A *two-parameter patch* allows simultaneous 2-DOF manipulation of two parameters. A *multi-parameter fan* (inset), extends a patch to allow simultaneous 2-DOF manipulation of adjacent fan parameters. The trajectory tangents (green) for the parameters at the common layout vertex are projected (blue) onto the tangent plane at the vertex, and the radial angular order of the projected tangents in this plane, defines the adjacency of parallelogram patches (angle $\leq 180^o$) of the fan.



The rig parameters that define multi-parameter elements typically deform the same region somewhat orthogonally, such as a *jaw-open-close* and *jaw-left-right*. While multiple rig parameters are used to define patch and fan UI elements, such elements should be represented by a single representative control in the layout C, so that the element can be created at a common vertex in the optimal layout L. Below we thus assume each control c_j to map to a unique UI element E_j.

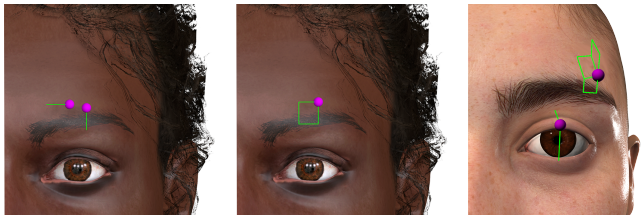


Fig. 5. Rig UI layouts can be optimized for arbitrary UI elements, such as the sliders, patches and fans shown. Further, our algorithm can automatically group rig parameters with large overlapping regions of deformation, to define 2-parameter patches, and multi-parameter fans.

3.3 UI Layout Design Objectives

We observe that the following extendable set of objectives guide the functional and aesthetic quality of a UI layout. We discuss these objectives for single-parameter sliders. For a multi-parameter UI element (patch or fan), we set the objective to be the worst objective value computed for each of its mapped parameters independently.

Maximal Displacement

The visual impact of manipulating a rig parameter is best observed in regions where it invokes large deformations. Functionally, larger deformation trajectories also produce larger UI elements that are easier to manipulate with better resolution. A user wishing to *squint* an eye in Figure 1 for eg., would expect to tug directly on a UI element that tracks a vertex on the eyelid, instead of a visually confusing and unusable zero-length UI element attached to say, a vertex on the chin. As a down-side, a large UI element may disproportionately occupy space and interfere with the 3D shape and UI elements of other rig parameters. We thus model this objective for a candidate layout L as a function of the arc-length of its deformation trajectories D.

Minimal Inter-Element Overlap

The UI elements in hand-crafted layouts (Figure 3) are spatially well-spread. Elements that intersect or have significant spatial overlap are unattractive, visually cluttered and difficult to select unambiguously. We thus aim to minimize inter-element overlap (or alternately maximize inter-element distance).

Figure 6 shows the trade-off between the use of maximal displacement, and minimal inter-element overlap in our approach.

Element-Model Occlusion

It is visually desirable for UI elements to remain largely un-occluded by the 3D shape during interaction (even though they can always be drawn un-occluded as an overlay). Locally, we thus favour trajectories for control c_j whose tangent d'_j at the default configuration, aligns well with the 3D vertex normal for the layout vertex l_j. This alignment can be measured as a function of the dot product of the vectors. We can also address element-model occlusion in a view

dependent fashion by limiting candidate layout vertices, to those that are visible from a given view in the default configuration. Finally if a view-dependent 2D layout is desired, the optimal layout can further be computed after view-projecting elements in 2D, to penalize strongly foreshortened UI elements (Figure 7).

Symmetric Elements

As characters are inherently symmetric, most rigs have parameter sets that deform shape symmetrically (for eg. a *left-squint* and *right-squint* control). Such symmetric controls are readily inferred automatically [Mitra et al. 2012], or can be user annotated. UI elements for symmetric parameters should convey that symmetry visually. Given controls c_j, c_k subject to some symmetry transform S, we enforce precise symmetry by removing c_k from the layout optimization, but retaining its UI element as part of c_j (i.e. the UI element for c_j is defined as both E_j and its symmetric element $S(E_j)$). This ensures that the optimal l_j will account for both UI elements it is responsible for creating, and that the UI elements for c_j and c_k are symmetric. The above trivially generalizes to multi-fold symmetry by optimizing the UI layout of one representative control c_j for each set of symmetric controls.

Symmetric Deformations

Characters often have individual parameters c_j that deform the shape symmetrically (for eg. a *brow-raise* control that wrinkles the forehead symmetrically). Such controls should visually lie along their plane or axis of symmetry. We can enforce this by constraining the set of candidate layout vertices for l_j to those that lie precisely on the plane or axis of symmetry. In the absence of such vertices, the hard constraint can be replaced with a function that penalizes the distance of a candidate vertex l_j, from the plane or axis of symmetry.

Global Deformations

While most rig parameters are manifested as localized shape deformations, some rig parameters, such as one to customize gender may deform all vertices of the shape significantly. Placing a UI element for such a control locally on the shape can be misleading, and the UI elements for such controls, such as the gender symbols in Figure 3(a), are often presented as global and off-shape.

4 LAYOUT OPTIMIZATION ALGORITHM

The search for a layout (mapping m rig controls to n shape vertices) L^*, that optimizes the design objectives in Section 3.3 is an instance of an integer quadratic programming problem, known to be NP-Hard. We first describe the quadratic programming formulation below, and then look at various pre-processing steps in Section 4.2, that reduce the computational complexity, and make the optimization practical for state-of-the-art industrial rigs (Figure 2, 12).

4.1 Integer Quadratic Programming

We represent a candidate layout L using an $n \times m$ binary matrix $\mathbf{X}$ where $x_{ij} = 1$ iff control c_j maps to vertex v_i in L (i.e. l_j is v_i). Each column j of $\mathbf{X}$ is thus all zero elements and a single 1 in the row for vertex l_j. We interchangeably write matrix $\mathbf{X}$ as a concatenation of rows into an $n*m$ column vector $\vec{x} = [x_{11}, x_{12}...x_{nm-1}, x_{nm}]^\mathsf{T}$. We use x_{ij} equivalent to $x_{(i-1)*n+j}$, and x_k as equivalent to $x_{row(k)col(k)}$, where $col(k) = (k-1)\%m + 1; row(k) = (k - col(k))/n + 1$.

The maximal displacement objective for a given rig parameter configuration, is also encoded as an $n \times m$ column vector $\vec{d}$, where

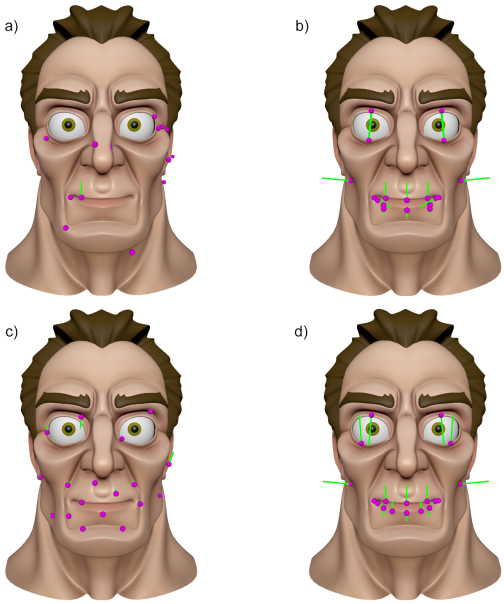


Fig. 6. Design Choices: a random control to vertex mapping produces visually meaningless and unusable UI layout (a); a layout based on maximal displacement can result in visually cluttered and occluded UI elements (b); a layout based on minimal inter-element overlap creates well spread but small UI elements (c); our approach jointly optimizes displacement, overlap, occlusion and symmetry criteria (d). Rig by ©Antony Ward (3D World)

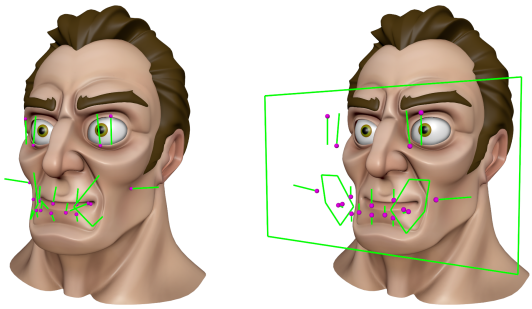


Fig. 7. UI layouts can be optimized in 3D, or 2D from a given view.

d_k represents the arc-length of the deformation trajectory of vertex $v_{row(k)}$ controlled by UI element $E_{col(k)}$.

Inter-element overlap is encoded as a symmetric matrix $\mathbf{Q}$ of size $nm \times nm$ where q_{rs} is the inter-element overlap $\phi(dist(x_r, x_s))$, where $dist(x_r, x_s)$ computes an inter-element distance between element $E_{col(r)}$ mapped to vertex $v_{row(r)}$, and element $E_{col(s)}$ mapped to vertex $v_{row(s)}$, and ϕ is a sigmoid function.

We calculate the inter-element distance to capture the difficulty of selecting a point on a UI element due to spatial interference with another element. We thus first discretely sample any x_r (uniform parametric sampling): the UI element $E_{col(r)}$ placed in 3D at shape vertex $v_{row(r)}$ is sampled at u points $W_r = \{w_r^1 ... w_r^u\}$. We then compute the shortest distance from every point in W_r to the point sample set W_s for x_s, and vice versa. The overall average of this

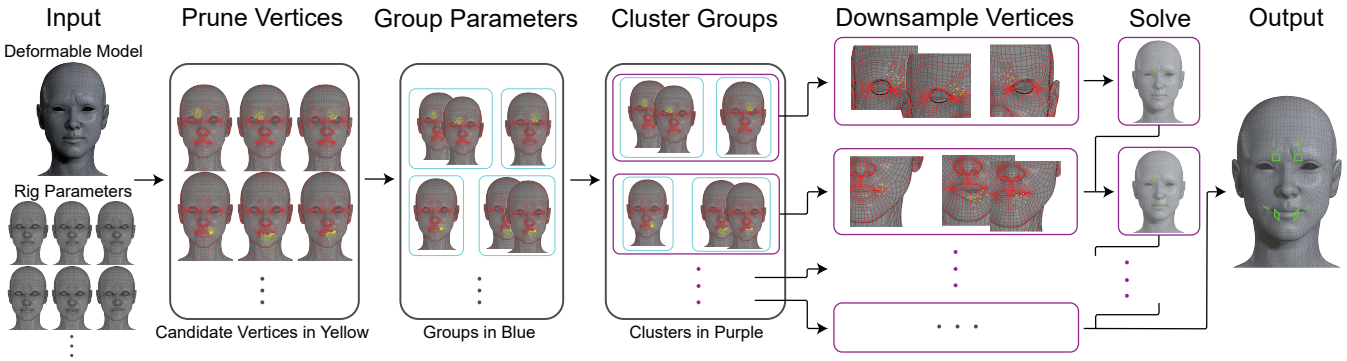


Fig. 8. Algorithm Overview: model vertices are first pruned based on design criteria to a candidate set of layout vertices for each rig parameter; the parameters are then grouped and clustered based on the overlap and separation of their candidate vertices; candidate vertices in each cluster are then sampled and used to compute an optimal layout using integer quadratic programming.

shortest distance is a symmetric distance measure $dist(x_r, x_s)$. This distance is then inverted to capture overlap using a function $\phi(z)$:

$$\phi(z) = a - \frac{a}{1 + e^{s(z-\tau)}} \tag{1}$$

where a is the maximum overlap error, τ is the distance for z which gives $a/2$ overlap, and s scales how quickly the overlap penalty vanishes with increasing distance. In our implementation we used values $a = 5$, τ = average edge length between adjacent shape vertices, and $s = 2$.

We cast these objectives as Equation 2 in a quadratic program:

$$\min_{\vec{x}} \quad \frac{1-\lambda}{2}\vec{x}^{\mathsf{T}} Q \vec{x} - \lambda d^{\mathsf{T}} \vec{x} \tag{2}$$

$$\text{s.t.} \quad \mathbf{A}\vec{x} = \vec{1} \tag{3}$$

$$\mathbf{B}\vec{x} \leq \vec{1} \tag{4}$$

$$\vec{x}_{1..nm} \in 0, 1 \tag{5}$$

The relative importance of large displacements vs. inter-element overlap in equation 2 can be controlled using λ (default $\lambda = 0.85$). Additional design objectives can be similarly added to equation 2 as quadratic or linear functions of $\vec{x}$. The goal of equation 3 is to ensure that each of the m controls gets mapped to exactly one of the n vertices. $\mathbf{A}$ is thus an $m \times nm$ matrix. Row j of $\mathbf{A}$ thus needs to isolate all variables x_{ij} for $i \in \{1, ..n\}$. Row j of $\mathbf{A}$ is thus all 0's except for n 1's every m elements, starting at index j. Inequality 4 ensures that each of the n vertices is mapped to by at most one of the m controls. $\mathbf{B}$ is thus an $n \times nm$ matrix. Row i of $\mathbf{B}$ is filled with 0's except for a block of m 1's starting at column $i*n$. Equation 5 ensures that all variables in our quadratic program are binary. We found a quadratic program to be significantly faster and space efficient than a linear program that needs nm^2 additional variables to capture the inter-element overlap.

Figure 9 further shows how our algorithm parameters can impact the optimal rig UI layout, by varying the contributions of inter-element overlap, and occlusion culling (particularly useful when many parameters are concentrated in local regions of high occlusion like lip corners).

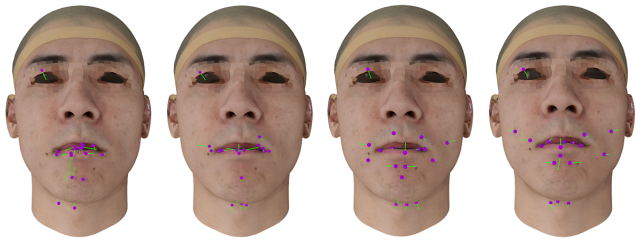


Fig. 9. FaceScape #393 ©The CITE LAB. Left to right: λ = 1 (maximal displacement), λ = 0.85 (default), λ = 0.4 (higher inter-element penalty), occlusion culling.

4.2 Optimization Pre-processing

We now discuss four independent grouping and pruning strategies that can radically reduce the number of variables we input to our quadratic layout optimization above.

Pruning Layout Vertex Candidates

Our optimization variables x_{ij} consider all vertices v_i as layout mapping candidates l_j for parameter c_j, yet many of these variables would produce undesirable UI elements, and can be omitted from the pool of candidates. We thus use our design principles based on the following criteria, to produce a pruned set of candidate vertices and associated variables $\vec{x}_{pru}$:

- *Minimally deformed vertices*: We prune variable x_{ij} if the length of v_i's deformation trajectory for c_j is below a threshold (default set to the avg. length of all v_i with non-zero trajectories). Deformation controls c_j for which the minimally deformed vertices are a small fraction ($< 10\%$) of model vertices are termed global. Such controls, like the gender control in Figure 3(a), are often laid out adjacent to the shape, and can optionally be removed from the optimization.
- *Occluded Vertices*: We prune variables x_{ij} whose deformation trajectory for parameter j is largely occluded by the model. We can also prune all x_{ij}, that are invisible from a given view.
- *Symmetric Deformations*: For any parameter j that symmetrically deforms the shape, we can prune away all variables x_{ij}, whose vertices v_i do not lie on the plane/axis of symmetry.

Multi-Parameter Grouping

Rig parameters that deform very similar regions of the model are often complementary and may benefit from UI-elements like patches and fans (Figure 5) that provide concurrent multi-parameter control (Section 3.2). To detect possible parameter groupings of size k (for a k-parameter fan, default $k = 2$ for a patch element), we initialize a set of groups $G = [g_1..g_m]$, associated with with each rig parameter $c_1..c_m$ and a vertex set $g_j.v$ that comprises candidate layout vertices for c_j after pruning. We then iteratively combine control groups g_i and g_j, that together can be represented by a k-parameter fan (i.e. $|g_i \cup g_j| \leq k$), and whose Jacaard index J, where $J = \frac{|g_i.v \cap g_j.v|}{|g_i.v \cup g_j.v|}$ is the largest (i.e. we pick the i, j pair with he biggest $J > \epsilon$ (default $\epsilon = 0.85$). We combine the two control groups by setting $g_i = g_i \cup g_j$ and removing g_j from G. Likewise, $\vec{x}_{pru}$ is reduced to only retain variables corresponding to vertices $g_i.v \cap g_j.v$ for combined controls $\in (g_i \cup g_j)$. The final set of parameter groups in G, with its corresponding associated variables termed $\vec{x}_{grp}$, can be further reduced by clustering and down-sampling below, or serve as input to the quadratic optimization in Section 4.1.

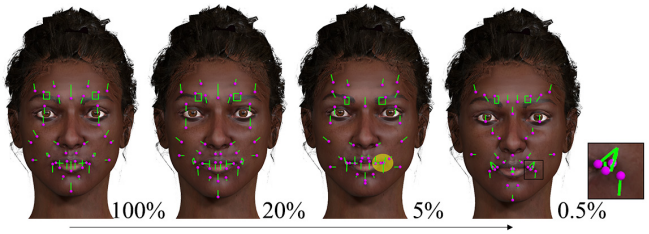


Fig. 10. UI layouts resulting from weighted random down-sampling of vertices. The UI layout created using 20% of the model vertices is qualitatively similar to the optimal layout (100%). Quality degrades with visual clutter at 5% and inter-element overlap at 0.5% down-sampling.

Rig Parameter Clustering

The optimization space of candidate control layouts for n model vertices and m rig controls grows exponentially with m. We note however, that many pairs of controls locally deform spatially distinct regions of the model, and their respective UI elements are unlikely to overlap. Partitioning the controls into such spatial clusters that can be independently optimized can thus greatly improve efficiency, with little detriment to the overall UI layout. For example, UI elements for controls that deform the eyes and mouth are spatially distant in Figure 1, 11, and could thus be optimized separately.

We thus implement a simple single-link clustering [Murtagh and Contreras 2012] on the rig control parameters where the distance metric between two clusters of parameters U and V:
$Dist(U, V) = \sum_{c_u \in U} \sum_{c_v \in V} \sum_r \sum_s dist(x_{ru}, x_{sv})$,
where $x_{ru}, x_{sv} \in \vec{x}_{grp}$. In other words the distance between two clusters U, V is the total inter-element distance between all candidate vertices for all pairs of controls belonging to U and V respectively.In practice, replacing the inter-element distance $dist(x_{ru}, x_{sv})$ by the distance between vertices $||v_r - v_s||$ produces similar results with better efficiency. Let $S = \{S_1..S_o\}$ be the resulting clusters that partition the set of rig controls. Consequently, variables $\vec{x}_{grp}$ are partitioned into $\vec{x}^1_{grp}..\vec{x}^o_{grp}$, where $x_{ij} \in \vec{x}^k_{grp}$ iff $c_j \in S_k$.

Figure 11 shows the clustering of rig controls for different values of o. We use a default of $o = 2$ to capture the typical upper/lower half split on faces. We then compute the UI layout of each cluster of controls in sequence, starting with the largest cluster. The UI elements computed for former control clusters contribute fixed inter-element distances as constraints for subsequent clusters and thus impact the optimal layout when optimizing latter clusters. As shown in Figure 11, increasing the number of clusters can substantially improve efficiency, with minimal impact on the optimality of the control UI layout.

Vertex Down-sampling

High-end production models such as Figure 12 have upwards of 100K vertices. Rig deformations however, are mostly smooth, and regions of neighboring vertices tend to have similar deformation trajectories. We thus note that lower resolution proxies with a fraction of the original shape vertices, can produce near optimal UI layouts more efficiently (Figure 10). There are a number of ways to down-sample the shape including random vertex sampling and model decimation that retains correspondence to a subset of the original shape vertices [Botsch et al. 2010]. Unlike typical mesh simplification that optimizes geometric fidelity to the original model, our sample importance criteria for a vertex favours large displacements (Equation 2). We thus perform a weighted random sampling on all $\vec{x}$ variables $\vec{x}^k_{grp}$ in each cluster $k \in\in \{1..o\}$ above. The weights for each variable are associated with the displacement d of its corresponding UI element. Our final set of variables down-sampled from $\vec{x}^k_{grp}$ is termed $\vec{x}^k_{dsp}, k \in\in \{1..o\}$. Figure 10 shows that 5K+ vertex shapes can be down-sampled 5-10x before there is noticeable visual degradation in the control UI layout.

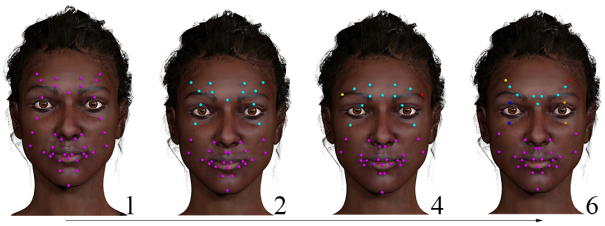


Fig. 11. Partitioning rig parameters significantly improves performance, while preserving the visual quality of the UI layout. Computation time (seconds) for 1 (164.43), 2 (13.6), 4 (11.41), and 6 (9.95) clusters.

4.3 Implementation Details

We implemented our approach using a quadratic programming solver in C++ using IBM CPLEX Optimizer library and control our models in Autodesk Maya®2020. The results shown in the paper were generated on a a Windows 10 machine with an Intel Core i5 3.6GHz processor and NVIDIA®GeForce GTX 1080.

5 USER STUDY

We conducted a within subjects user study with 12 (8 amateur, 4 professional) participants (contacted via email), to evaluate the user experience of our UI layouts. All participants had access to and experience with Autodesk Maya®, in which the study was conducted. Participants were identified as amateur/professional, specific to their

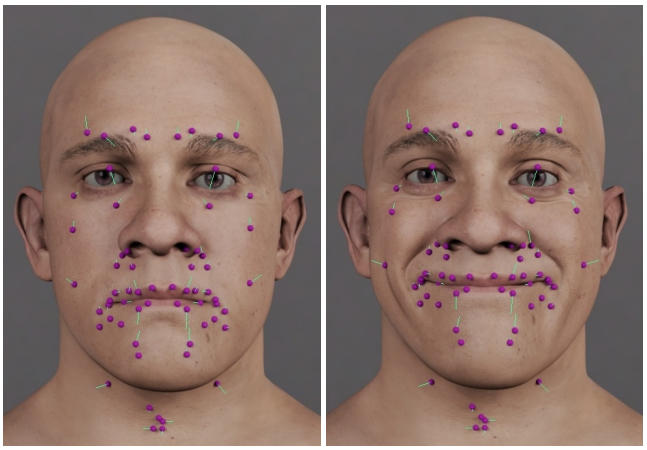


Fig. 12. UI layout for a 85K vertex, 65 control Allan Henry face-rig ©Weta Digital with neutral (left) and posed parameters (right).

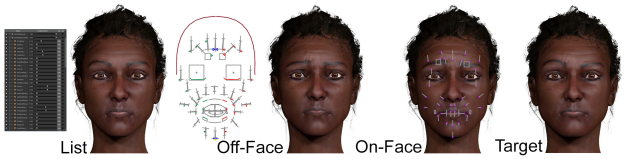


Fig. 13. Expressions set by user *P8* using the three compared interfaces, for the common target expression.

experience animating 3D face-rigs. While our in-situ layouts are designed to be used with direct manipulation [Lewis and Anjyo 2010] within a single interface (Figure 1(e)), our study disabled direct manipulation to focus on a principled comparison of independent parameter control: between a baseline slider *list*, an *off-face* animator created gold-standard layout, and our *on-face* automatic UI layout, for the model in Figure 13. We note that the artist layout and ours, could theoretically be interchangeably used as *on* or *off* face layouts.

5.1 Experiment Setup and Protocol

The study presented a subject with a posed target face, adjacent to a neutral face that was interactively manipulated by the subject using one of the above three interfaces, to visually match the target, until satisfied (Figure 13). Each participant was presented with the same sequence of 15 faces to match (3 interfaces * 5 faces). The interfaces were tested in order (one of 6 permutations of *list*, *off-face* and *on-face*), fully counter-balanced across the 12 subjects. 12 of the 15 faces were unique and of varying complexity (activating 1-13 rig parameters) spread across the three interfaces tested. One expression (13 activated parameters) was repeated for all the 3 interfaces (Figure 13). Participants were presented with the faces side-by-side in a frontal view, but were free to manipulate the camera, as desired.

The study instructions and testing itself were driven from a UI control panel within Maya. Each interface test started with 2 simple expressions (1-2 rig parameters) followed by 3 more complex expressions (6+ rig parameters). The study was run uncontrolled, and completed on average in 28 minutes by participants remotely in Maya. After the study participants filled-out an online questionnaire on their experience using the three interfaces (Figure 14).

5.2 Analysis

All subjects had some familiarity with Maya, and were able to complete the study without any reported difficulty. Our summary finding was that qualitatively, the overall user experience of the on-screen interfaces *off-face* and *on-face* (both with avg. 4 out of 5) was much better than the slider *list* (avg. 2.25 out of 5) (Figure 14(left)). Our *on-face* interface also scored highly (avg. 4.83 out of 5) on important questions like "The system gave me a feeling of directly manipulating the face?" (Figure 14(right)). Quantitatively, the study data corroborated user response: users were significantly slower with the slider *list* (46.87% of their total time on avg.), compared to the artist created *off-face* (25.62%), and our *on-face* interface (27.51%) (Figure 15(right)); while accuracy in matching target faces was similar, since users manipulated the face until satisfied. The *list* was marginally, but not consistently more accurate than the on-screen interfaces, possibly due to finer control resolution of the longer *list* sliders (Figure 15(left)).

Quantitative

We measure expression accuracy in both rig parameter space (wt. error sq.) and model space (vtx. error sq.), as the squared difference between the values set by the user and the target expression. For reference, we report these error values as normalized relative to the difference between the target and neutral expression. In other words, the relative vertex error squared for a user set expression u, target t and neutral n is: $\sum_v(||v_u - v_t||^2)/\sum_v(||v_n - v_t||^2)$. The rig parameter weight error is formulated similarly. Rig parameter weight errors better reflect perceptual changes in expression, but these weight errors can sometimes be misleading due to redundancy in rig parameter space (i.e. a user may create a similar visual expression to the target by using a different rig parameter that produces a similar visual change in expression). Time spent on an interface is also normalized as a $\%_{age}$ of total time taken by a user (Figure 15).

Figure 13 shows the complex common expression posed in block order (*list*, *on-face*, *off-face*) by user *P*8. The actual time and $\%_{age}$ time spent on each interface in block order was (223s, 119s, 80s) and (17.5%, 9.3%, 6.3%), and the relative weight and vertex sq. errors were (1.0, 1.01, 1.34) and (0.43, 0.47, 0.57), respectively. These statistics might suggest a learning effect with users getting quicker with each block, and that accuracy depends on time spent with the interface. This however, was not a general trend: for eg., the same statistics for *P*12 (*on-face*, *off-face*, *list*) were (146s, 153s, 186s), (9.9%, 10.4%, 12.6%), (0.35, 1.36, 1.45) (0.33, 0.9, 0.62). In general for the common expression, overall participants spent 36% and 12% more time using *list* and *off-face*, than our interface.

Judging, from user comments, the off-screen parameter *list* takes consistently longer to find and manipulate than on-screen controls ("moving back and forth between the face and the slider GUI takes time" *P*8, "slider attribute editor interface does not scale, need to hunt for controls in a scroll list and controls are sometimes not in list close to similar ones" *P*6, "when working with an animated face in a dynamic scene it is better to have controls close at hand", *P*7).

The marginally better accuracy of the slider *list* could be due to resolution ("one advantage of the slider interface is that it does seem to provide finer resolution when interacting with the controls" *P*7). The resolution of on-screen controls can be made finer by zooming

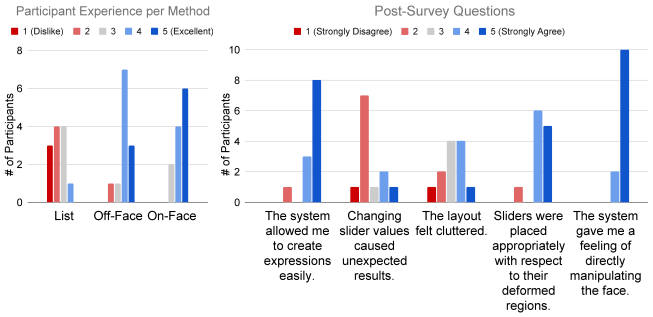


Fig. 14. User Study: overall experience with the three interfaces (left); questions pertaining to the *on-face* UI (right).

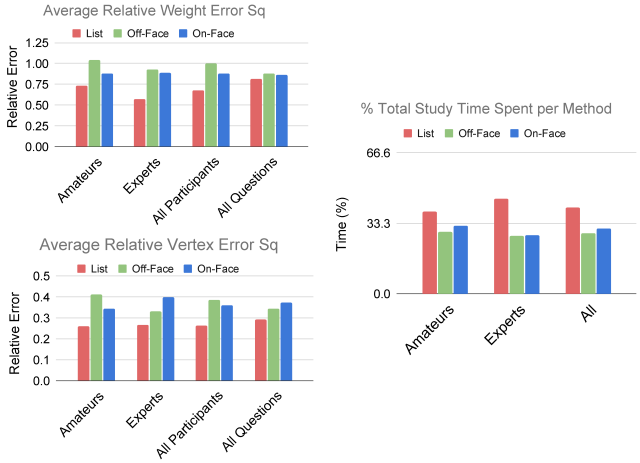


Fig. 15. User Study: avg. relative weight|vertex sq. error (left top|bottom) and % time the users spent on the three interfaces (right).

into the model but we feel users did not manipulate the camera much ("...this fixed model and camera test setting" *P*7).

Qualitative

Figure 14(left) shows that our interface was well-received, relative to the *list* and *off-face* interfaces. The responses to questions (below) focused on our interface were also largely positive Figure 14(right):

– The system allowed me to create expressions easily.
– Changing slider values caused unexpected results.
– The layout felt cluttered.
– Sliders were placed appropriately with respect to their deformed regions.
– The system gave me a feeling of directly manipulating the face.

We also received a wealth of both positive and critical feedback:

General (Positive)

– "I really liked the on-the-face slider a lot." *P*4.
– "Am not an expert on faces but such an interface (on-face) can be very useful for any on-object parameter interaction." *P*11.
– "The on-face patches are very cool and it would be great to see more such coupled controls". "Maybe this grouping can even be dynamic, where I select two or more controls..." *P*8.

Direct Manipulation (Positive)

– "On-face sliders are great and direct" *P*6.
– "The on-face sliders feels very direct because all the sliders move and stay stuck to the face. It would be cool to augment the on-face slider interaction with directly pulling on parts of the face." *P*7.
– "Precise and local control with direct face grabbing is a problem, and the on-face sliders complement direct manipulation beautifully. I would love to see them work together." *P*8.
  Our layouts are indeed designed and already implemented to be used in conjunction with direct manipulation (Figure 1(e)).

Face specific (Critical)

– "It would be useful to show the area of influence when you select the control to give a sense of what area is affected (similar to skin weights map). Or the inverse: clicking anywhere on the mesh would highlight which controls have influence on that area." *P*5.
– "All 3 interfaces could benefit from symmetric control toggle." *P*6.

Model occlusion (Critical)

Many users found that *on-face* visually obscured evaluating the manipulated expression, suggesting off-face transport, and UI element toggles/transparency as solutions:

– "Perhaps working on a duplicate copy of the face will be better? So your manipulations feel direct, but you can still see the actual result well." *P*2 (also *P*1).
– "It would be useful to be able to move the off-face controls on-face and vice versa for the on-face sliders." *P*7.
– "It felt the most natural to use the on-the-face slider layout, but I wish I could toggle the visibility of the sliders in this format." *P*3 (also suggested by *P*5, *P*10 and *P*6 actually "added a toggle to hide/show them").
– "On-face sliders need to be very transparent but if not visually occluding can save time keeping the focus on the face." *P*11. (also *P*8 did likewise "I made the on-face slider shader very transparent so I could grab them and evaluate the facial pose simultaneously.").

6 RESULTS AND DISCUSSION

Direct Shape|Parameter Control Interface

While the user study (Section 5) only compared independent parameter control using our *on-face* UI layout with an *off-face* hand-crafted UI layout and a baseline slider *list*, our layouts provide a homogeneous interface to complementary direct-shape and direct-parameter manipulation (see video and Figure 1(e)).

Gallery of Rig UI layouts

Figures 1,2,17 show a number of rigs, for which our approach produced UI layouts that matched the aesthetic and functional choices made by artists in hand-crafting UI layouts for the rig. Figures 9,12,18 show further UI layouts for a number of face rigs for which we had no hand-crafted comparison.

Table 1 shows the input vertex and control complexity, algorithmic parameters, and computation times, for all the face-rigs shown in the paper. While our core optimization algorithm (Section 4.1) can be exponential in complexity, the pruning, grouping, clustering and down-sampling stages in (Section 4.2), allow us in practice to compute rig UI layouts for high-end face rigs with over a hundred rig

Animator critique

We consulted a professional animator to critique the rig layouts in Figures 17, 18. He found our rig UI layouts to be compelling, and was easily able to refine our layout aesthetics in Figure 1(f). He further commented on various aspects of our problem space, opining that our interface was (or could be):

discoverable and could be manipulated instinctively, without instruction or labels, in contrast to a baseline list of parameter names;
immersively *direct* keeping focus on the face in context with UI elements that tracked the deformed shape during manipulation (Figure 4(right) and video);
aesthetic, and produced well-spaced layouts, respecting symmetry and occlusion handling;
localizable, in that the UI layout of a parameter could be forced to lie in proximity to scene element, for eg., the UI element for a joint related rig parameter should be mapped to the shape in the vicinity of the joint Figure 16(right) (also see future work);
customizable, in that animators should be able to provide a partial rig UI layout for some parameters and let our algorithm automate the others, which we handle by simply pre-constraining the variables pertaining to the hand-crafted controls.

General Deformable Rigs

Conceptually, our approach is rig agnostic. Shapes can be driven by skeletal skinning, blendShapes, or other geometric/data-driven deformers, as long as rig parameters can be independently sampled to construct the piece-wise linear deformation trajectories of rig vertices, required by our algorithm (Section 4). For example, Figure 16 (left) illustrates a simple rig UI layout from sampling rotation parameters of the shoulder (green), elbow (red) and wrist (blue) skeletal joints, and Figure 16 (right) a pose-based UI element.

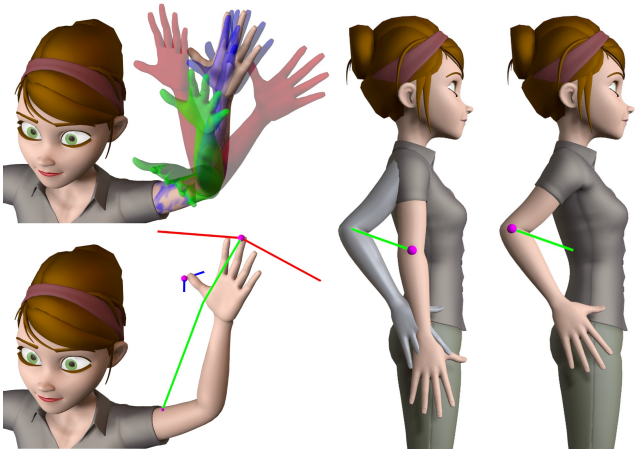


Fig. 16. Pose-Pose control using rig UI layouts.

Limitations and Future Work

Being an early investigation into automated in-situ UI generation for 3D deformable models, we only touch upon some aspects of the problem space, like automating the design of the visual form and function of UI elements/widgets (Section 3). The focus of our work, optimizing rig UI layout also has inherent limitations.

– In-production rigs can have highly localized and/or redundant parameters that cause inevitable UI crowding (Figure 12). While penalizing inter-element overlap can partly alleviate this problem (Figure 9), the problem can also be addressed by dynamically spreading the UI upon interaction, using layouts with diagrammatic arrows [Agrawala et al. 2011], or by a multi-level user interface.

Model	Vertex Count	Controls	Clusters	Sample %	**Time (s)**
Dakar Valley Girl	5064	47	2	100/20/5	249/45/11
Head Rig	1186	26	2	20	<1
Louise	5034	28	2	20	3
Mery	12942	28	2	20	3
Nico	27434	28	2	20	19
Dan	5706	23	2	20	62
Clara	5064	45	2	20	15
Prague Boy	5064	47	2	20	18
Pidoras	5064	25	2	20	62
Angela	4002	22	2	20	<1
Allen Henry	85753	65	4	10/5/1	642/243/108
Metahuman	24049	142	4	5/1	553/162
FaceScape #393	16437	20	2	20	210

Table 1. Shape vertex and control count, algorithmic parameters, and computation times, for all the face-rigs as shown.

– Rig parameters controlling subtle deformations, even when mapped to their maximally displaced vertex, may produce tiny UI elements that have poor resolution for interaction.
– Global rig parameters and those controlling largely hidden regions of a face like the tongue, may not be ideally suited to our proposed on-face UI layout.
– Sometimes UI elements enable constrained parameter manipulation such as the focus point cross-hair of an eye-gaze controller. While we do not design such UI elements, artists can hand-craft UI elements for some rig parameters, and use our approach to optimize the remaining rig UI layout, subject to a partial rig UI.
– Some rig parameters, like a sculpting tool deformation radius, are better conveyed by a circular UI element and not a UI element that tracks the deformation trajectory of a vertex of the shape.

There are many avenues for future work. Our multi-parameter groupings at present are simple: either user specified, or simply based on overlapping spatial regions of influence. Inferring data-driven (rig + animation data) grouping, correlation, and independence between rig parameters is subject to future work. Our user study participants further suggested dynamic parameter groupings, and suggested that our approach can form the building block of a more complex posing system where multiple rig parameters can be posed and controlled in-situ using our rig UI layouts (Figure 16). Another interesting avenue for research is customizing the UI elements of the rig parameters to be visually representative of the region of the shape they deform, or other rig entities like bones or joints (for eg. the hand-crafted jaw or eyelid UI in Figure 1(b)). Finally, while our approach can conceptually define a UI layout for any deformable rig, such as Figure 16, our claimed contributions to the UI layout of face-rigs, remain to be validated for general deformable rigs.

7 CONCLUSION

Inspired by animator hand-crafted UI layouts, and increasing on-screen UI elements seen in games, mixed reality, and graphics applications, we propose the novel problem of automating rig parameter UI layouts. We then present an automated solution to creating such UI layouts for deformable rigs. Our approach seamlessly integrates into current animation pipelines, in game engines and commercial software, like Autodesk Maya. Character rigs form the basis for expressive animation. We believe that automatically streamlining the interface between the rig and the animator will positively impact the

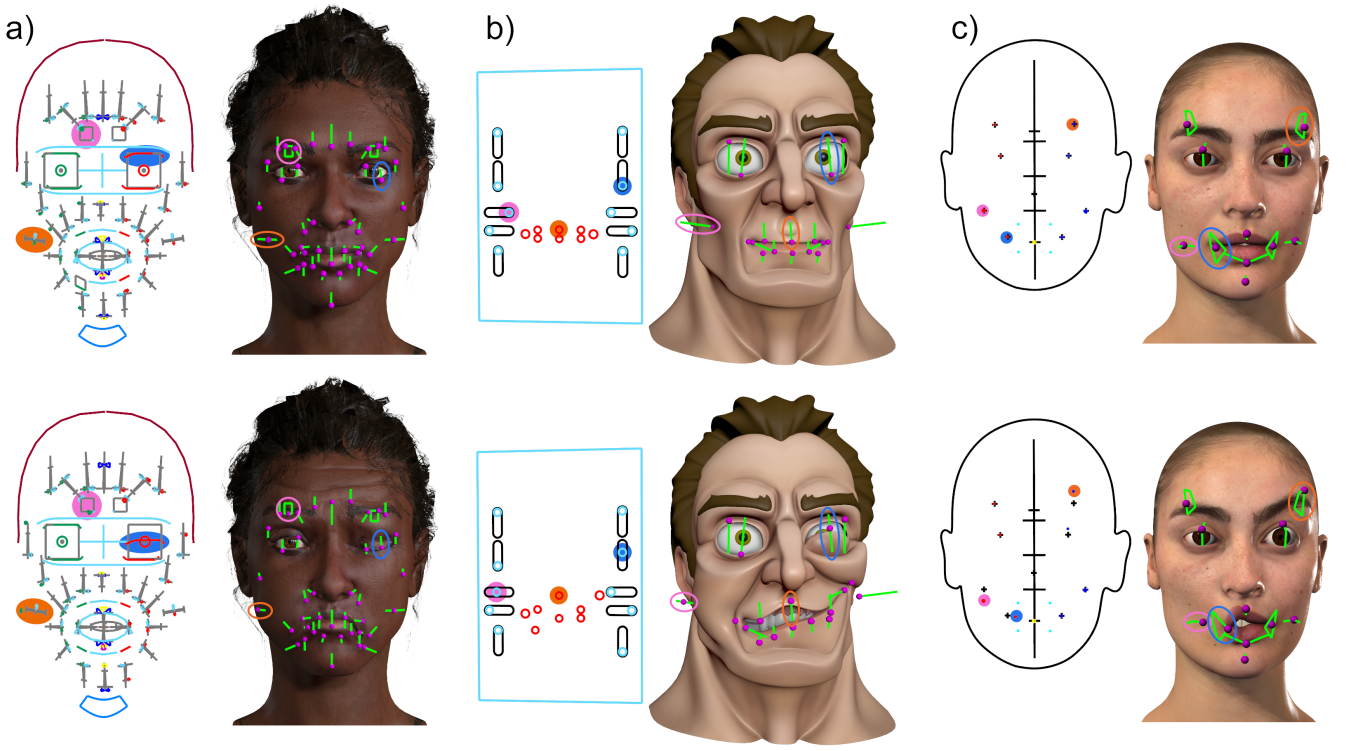


Fig. 17. Hand-crafted layouts (left) compared to our rig UI layouts (right). a) Dakar Valley Girl ©Chris Landreth b) Rig by ©Antony Ward (3D World) c) Animatable Digital Double of Louise by Eisko© (www.eisko.com)

throughput and quality of animation produced. Beyond character animation, we hope our work will inspire future research in automatic creation of in-situ user interfaces.

Acknowledgments

We would like to thank Chris Landreth (JALI Research) and Steve Cullingford (Weta Digital) for providing us with facial rigs, animations, and invaluable feedback. We are also grateful to our user study participants, and to Metahuman (Epic Games), CGTarian, Antony Ward (3D World), CITE LAB, Eisko, Mery Project, and Chad Vernon for face rigs. This research was supported by NSERC.

REFERENCES

Rinat Abdrashitov, Fanny Chevalier, and Karan Singh. 2020. Interactive Exploration and Refinement of Facial Expression Using Manifold Learning. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 778–790. https://doi.org/10.1145/3379337.3415877

Maneesh Agrawala, Wilmot Li, and Floraine Berthouzoz. 2011. Design Principles for Visual Communication. *Commun. ACM* 54, 4 (April 2011), 60–69. https://doi.org/10.1145/1924421.1924439

Eric Allen and Kelly L. Murdock. 2008. *Body Language: Advanced 3D Character Rigging* (pap/cdr ed.). SYBEX Inc., USA.

Norman I Badler, Brian A Barsky, and David Zeltzer. 1990. *Making Them Move: Mechanics, Control & Animation of Articulated Figures*. Routledge.

Stephen W. Bailey, Dalton Omens, Paul Dilorenzo, and James F. O'Brien. 2020. Fast and Deep Facial Deformations. *ACM Trans. Graph.* 39, 4, Article 94 (July 2020), 15 pages. https://doi.org/10.1145/3386569.3392397

Eric Allan Bier. 1987. Skitters and jacks: interactive 3D positioning tools. In *Proceedings of the 1986 workshop on Interactive 3D graphics*. ACM, 183–196.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. CRC press.

Eric Brochu, Tyson Brochu, and Nando de Freitas. 2010. A Bayesian interactive optimization approach to procedural animation design. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 103–112.

Ozan Cetinaslan and Verónica Orvalho. 2018. Direct Manipulation of Blendshapes Using a Sketch-Based Interface. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology* (Poznań, Poland) *(Web3D '18)*. Association for Computing Machinery, New York, NY, USA, Article 14, 10 pages. https://doi.org/10.1145/3208806.3208811

Pif Edwards, Chris Landreth, Mateusz Popławski, Robert Malinowski, Sarah Watling, Eugene Fiume, and Karan Singh. 2020. JALI-Driven Expressive Facial Animation and Multilingual Speech in Cyberpunk 2077. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks* (Virtual Event, USA) *(SIGGRAPH '20)*. Association for Computing Machinery, New York, NY, USA, Article 60, 2 pages. https://doi.org/10.1145/3388767.3407339

Rosenberg Ekman. 1997. *What the face reveals: Basic and applied studies of spontaneous expression using the Facial Action Coding System (FACS)*. Oxford University Press, USA.

Krzysztof Z Gajos, Mary Czerwinski, Desney S Tan, and Daniel S Weld. 2006. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces*. 201–208.

Sarah Gibson, Paul Beardsley, Wheeler Ruml, Thomas Kang, Brian Mirtich, Joshua Seims, William Freeman, Jessica Hodgins, Hanspeter Pfister, Joe Marks, et al. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. (1997).

Michael Gleicher. 1992. Integrating constraints and direct manipulation. In *Symposium on Interactive 3 D Graphics: Proceedings of the 1992 symposium on Interactive 3 D graphics*, Vol. 1992. 171–174.

Martin Guay, Marie-Paule Cani, and Rémi Ronfard. 2013. The line of action: an intuitive interface for expressive character posing. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–8.

Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. 2014. Crowd-powered parameter analysis for visual design exploration. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 65–74.

Paul G Kry, Doug L James, and Dinesh K Pai. 2002. Eigenskin: real time large deformation character skinning in hardware. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 153–159.

Manfred Lau, Jinxiang Chai, Ying-Qing Xu, and Heung-Yeung Shum. 2009. Face poser: Interactive modeling of 3D facial expressions using facial priors. *ACM Transactions*

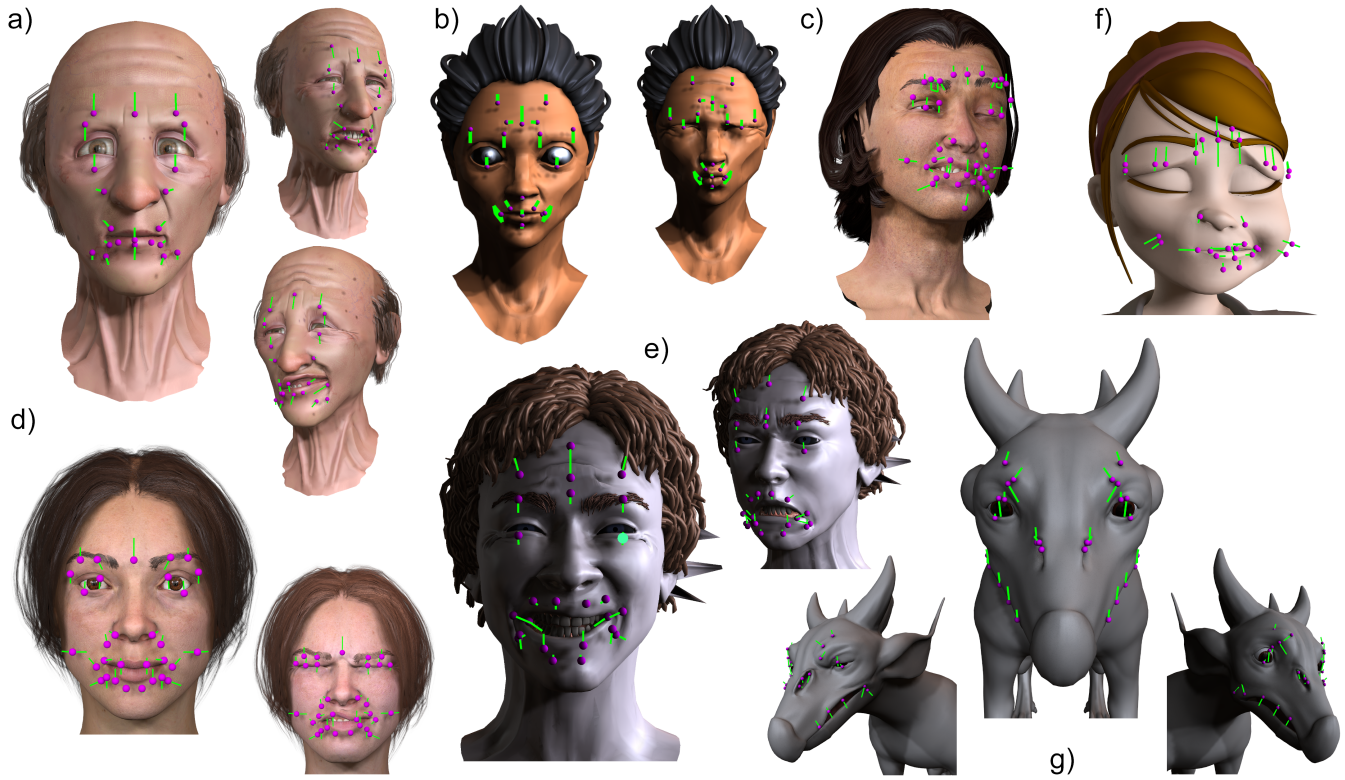


Fig. 18. Rig UI layouts on models for which no hand-crafted comparison is available. a) Dan b) Angela c) Prague Boy d) Clara e) Pidoras all by ©Chris Landreth f) Mery Project by ©{José Manuel García Alvarez and Antonio Francisco Méndez Lora} g) Nico by ©Chad Vernon (chandmv).

on Graphics (TOG) 29, 1 (2009), 3.

Binh H. Le, Mingyang Zhu, and Zhigang Deng. 2013. Marker Optimization for Facial Motion Acquisition and Deformation. *IEEE Transactions on Visualization and Computer Graphics* 19, 11 (Nov. 2013), 1859–1871. https://doi.org/10.1109/TVCG.2013.84

John P Lewis and Ken-ichi Anjyo. 2010. Direct manipulation blendshapes. *IEEE Computer Graphics and Applications* 30, 4 (2010), 42–50.

Hao Li, Jihun Yu, Yuting Ye, and Chris Bregler. 2013. Realtime Facial Animation with On-the-Fly Correctives. *ACM Trans. Graph.* 32, 4, Article 42 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2462019

José Carlos Miranda, Xenxo Alvarez, João Orvalho, Diego Gutierrez, A Augusto Sousa, and Verónica Orvalho. 2011. Sketch express: facial expressions made easy. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*. ACM, 87–94.

Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. 2012. Symmetry in 3D Geometry: Extraction and Applications. In *EUROGRAPHICS State-of-the-art Report*. https://doi.org/10.1111/cgf.12010

Fionn Murtagh and Pedro Contreras. 2012. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery* 2, 1 (2012), 86–97. https://doi.org/10.1002/widm.53 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.53

Victor Navone. 2020. Facial Animation for Feature Animated Films: Animating stylized facial expressions. *https://www.thegnomonworkshop.com/tutorials/facial-animation-for-feature-animated-films* (2020).

Jason Osipa. 2010. *Stop Staring: Facial Modeling and Animation Done Right* (3rd ed.). SYBEX Inc., USA.

Rick Parent. 2012. *Computer animation: algorithms and techniques*. Newnes.

Ryan Schmidt, Karan Singh, and Ravin Balakrishnan. 2008. Sketching and composing widgets for 3d manipulation. In *Computer graphics forum*, Vol. 27. Wiley Online Library, 301–310.

Thomas W Sederberg and Scott R Parry. 1986. Free-form deformation of solid geometric models. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 151–160.

Mike Seymour. 2016. Put your (digital) game face on. (2016). https://www.fxguide.com/fxfeatured/put-your-digital-game-face-on/

Mike Seymour. 2018. Making Thanos Face the Avengers. (2018). https://www.fxguide.com/fxfeatured/making-thanos-face-the-avengers/

Mike Seymour. 2019. Bebyface in Bebylon. (2019). https://www.fxguide.com/fxfeatured/bebyface-in-bebylon/

Karan Singh and Eugene Fiume. 1998. Wires: a geometric deformation technique. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 405–414.

Marc P Stevens, Robert C Zeleznik, and John F Hughes. 1994. An architecture for an extensible 3D interface toolkit. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*. 59–67.

Tanasai Sucontphunt, Zhenyao Mo, Ulrich Neumann, and Zhigang Deng. 2008. Interactive 3D facial expression posing through 2D portrait manipulation. In *Proceedings of graphics interface 2008*. Canadian Information Processing Society, 177–184.

Jerry O Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. 2009. Exploratory modeling with collaborative design spaces. *ACM Transactions on Graphics-TOG* 28, 5 (2009), 167.

J. Rafael Tena, Fernando De la Torre, and Iain Matthews. 2011. Interactive Region-Based Linear 3D Face Models. *ACM Trans. Graph.* 30, 4, Article 76 (July 2011), 10 pages. https://doi.org/10.1145/2010324.1964971

Ayush Tewari, Michael Zollöfer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Perez, and Theobalt Christian. 2017. MoFA: Model-based Deep Convolutional Face Autoencoder for Unsupervised Monocular Reconstruction. In *The IEEE International Conference on Computer Vision (ICCV)*.

Thibaut Weise, Sofien Bouaziz, Hao Li, and Mark Pauly. 2011. Realtime performance-based facial animation. In *ACM transactions on graphics (TOG)*, Vol. 30. ACM.

Lance Williams. 1990. Performance-driven facial animation. In *ACM SIGGRAPH Computer Graphics*, Vol. 24. ACM, 235–242.

Eduard Zell, JP Lewis, Junyong Noh, Mario Botsch, et al. 2017. Facial retargeting with automatic range of motion alignment. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 154.