EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2011)
T. Hammond and A. Nealen (Editors)

Sketch Express: Facial Expressions Made Easy

José Carlos Miranda[1] , Xenxo Alvarez[2] , João Orvalho[3] , Diego Gutierrez[4] , A. Augusto Sousa[5] , Verónica Orvalho[2]

[1]Instituto de Telecomunicações & Instituto Politécnico da Guarda - UDI, Portugal
[2]Instituto de Telecomunicações & Faculdade de Ciências da Universidade do Porto, Portugal
[3]Face in Motion, Portugal
[4]Universidad de Zaragoza, Spain
[5]INESC Porto & Faculdade de Engenharia da Universidade do Porto, Portugal

Abstract

Finding an effective control interface to manipulate complex geometric objects has traditionally relied on experienced users to place the animation controls. This process, whether for key framed or for motion captured animation, takes a lot of time and effort. We introduce a novel sketching interface control system *inspired in the way artists draw, in which a* stroke *defines the shape of an object and reflects the user's intention. We also introduce the* canvas*, a 2D drawing region where the users can make their strokes, which determines the domain of interaction with the object. We show that the combination of strokes and canvases provides a new way to manipulate the shape of an implicit volume in space. And most importantly, it is independent from the 3D model rig. The strokes can be easily stored and reused in other characters, allowing retargeting of poses. Our interactive approach is illustrated using facial models of different styles. As a result, we allow rapid manipulation of 3D faces on the fly in a very intuitive and interactive way. Our informal study showed that first time users typically master the system within seconds, creating appealing 3D poses and animations in just a few minutes.*

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism—Animation;

1. Introduction

Digital characters and animations are everywhere. Videogames and films drive the demand for powerful yet intuitive interfaces to allow animators to quickly produce final results. Usually animators work with rigged models: we can loosely define a generalized rig as a structured set of transformation and deformation elements that can modify an associated geometry by manipulating a set of controls. The rig can range from a simple bone system to a more sophisticated hierarchy of elements. As the complexity of the rig grows, creating the required different shapes and poses of the model by hand quickly becomes impractical. Thus, it is challenging for non-expert artists to master the manipulation and creation of rigs in a short period of time. User interfaces associated to the rig provide high-level controls masking most of the technical difficulties of the animation process, allowing for an easy manipulation, thus helping the user focus on the creative issues. While high-level rigs can simplify the animation process, encapsulating a set of control objects on a single element presents a particularly challenging problem: designing an interface that intuitively maps the manipulation of the control to the model deformation while increasing the rig usability. Our goal is to employ a sketch based user interface to control model deformations, so that complex shapes can be defined with just a freehand drawing.

In this paper we present a *facial sketching interface control system* based on simple strokes on a 2D canvas (see Figure 1). Sketching is an increasingly popular way to create, deform or control 3D models. It enables the user to create animations and free-form object deformations intuitively [IMT99] (e.g. designing a 3D model by drawing its silhouette), or to control the animation of a character [TBvdP07]. However, while most of these approaches act on the 3D mesh

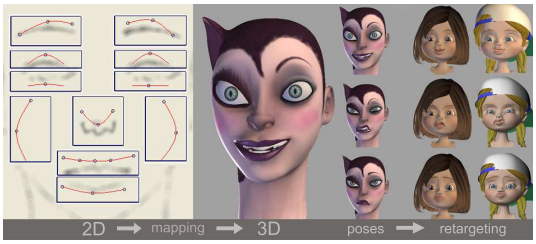

Figure 1: *System overview. left: drawing window where the user creates the 2D strokes; middle: 3D facial pose based on the strokes; right: retarget of facial poses.*

of the model, our method acts directly on the rig of the model, allowing for direct manipulation of the rig deformers. In addition, traditional manipulation techniques that rely on transform and attribute controls do not always follow the design of natural interfaces [Nor02], making their usability cumbersome for less experienced users.

Our solution allows the user to control an object without understanding the inner structure of the 3D model, and to intuitively create facial poses from scratch without manipulating complex rigs. Additionally, the poses can be retargeted to different models in a natural way, by storing the 2D strokes and later reusing them in a different model: *Sketch Once, Use Many*. As a result, we allow rapid animation in real time by sketching strokes on a virtual canvas.

Our work is inspired by Wobbrock et al. [WWL07] that shows how gesture-recognition, sketch-based systems allow for fast performance with little training. However, they are not intuitive as the user must draw in a specific way to make the system work properly. Also, while previous methods work directly on the 3D mesh [LCXS09], ours allows editing in 2D space, which is more intuitive than working directly on 3D models [SMND08].

We illustrate our approach, implemented in Maya, with several facial models of distinct artistic styles: from photo-realistic to cartoon. Additionally, we have carried out an informal study that shows how users with little or very limited expertise on 3D modeling can quickly create complex poses with our system. Expert artists can also benefit from our approach, by using the system for rough, quick blocking out of poses for animation.

The main contributions of our work are: (1) a facial sketching interface control system, which allows manipulating large number of control objects of the rig through a single control curve, the stroke; (2) a method to automate the movement of the control objects of the rig on the Z-axis, which automatically maps the deformation from 2D to 3D; (3) a retargeting technique, which allows reusing the strokes in different models.

2. Related Work

Research on sketch-based interfaces dates back to the SketchPad System [Sut64], which allowed creation and manipulation of objects on a screen using a light-pen input. However, creating a consistent system that provides a natural interface that understands the user's intention and display the correct result is still an open issue. The challenge behind sketching is to interpret what the stroke must do. Most previous sketching approaches deal with the creation, editing and deformation of 3D models [ZHH96, IH03]. For a more thorough review on sketching, we refer the reader to the survey by Olsen et al. [OSSJ09, JS11]. Our approach differs from those as we use sketching to control the structure underneath the 3D model - the rig. In this section we focus on those techniques most closely related to our work: sketch-based interfaces as control systems for 3D model manipulation.

Today, we can find tools for direct 3D manipulation of models in commercial modeling packages, like Maya and Blender. But, few sketch-based systems have explored the use of strokes as a control interface and not just for modeling [NISA07, SSB08], as the design of the widgets do not follow the sketching paradigm [Hou92, IH01]. Another major problem is that sketch-based interfaces lose precision when trying to manipulate a 3D model, because they do not control directly the details of the mesh. This goes against the nature of freehand drawing. A study found out that users are not satisfied with imprecise controls [IH01]. Thus, it becomes necessary to provide a method that bridges the gap between sketch-based interfaces and traditional 3D model manipulation [CSH*92]. Traditional approaches rely on the user to create a rig to control a 3D model. A rig can be based on shapes, bones or a combination of both. Editing the rig directly quickly becomes impractical when the complexity of the rig grows. Osipa [Osi07] presented a facial control interface that provides a high level viewport to edit the model and animations, but it only supports shapes.

Our work gives special attention to the control interface for facial rigs. Using sketching to generate 3D facial expressions has been explored by several researchers. Facial sketching requires methods that can cope with subtle skin deformation. The uniqueness of each face makes facial synthesis so challenging. The smallest anomaly in the face shape, proportion, skin texture or movement is immediately detected and classified as incorrect. Chang et al. [CJ06] allow changing the shape of face by editing two curves: reference and target. The reference curve allows the user to select the facial features on the mesh and the target curve determines the new shape of the mesh. Lau et al. [LCXS09] build upon this concept but allow animators to use pre-recorded data to eliminate the unnatural expressions that can be generated due to ambiguous user input. Sucontphunt et al. [SMND08] allow creating facial expressions on a 3D model by manipulating the control points of a set of predefined curves on a 2D portrait.

The novelty of our work lies in the direct manipulation of the underlying rig structure. Despite the lose of precision on sketch-based interfaces, in our approach the quality of the deformations are constrained by the rig. Our system does not need additional curves or use pre-recorded data. Also, it is not limited to a set of predefined curves with control points, but allows the user to generate facial deformation with any stroke and change the stroke at any point.

3. System Overview

Manipulating by hand each component of the rig (joints, shapes, deformers...) quickly becomes impractical when complexity grows. We have created an application in Maya that allows controlling simple and complex rigs based on a new approach. We begin with a rigged 3D model and a canvas. The canvas serves as a simple animation control system where the model deformation is sketched on. The user draws free-form strokes on the canvas and the system automatically creates the correspondent deformation on the 3D model displayed on a separate, side-by-side window. A stroke is represented by a curve and the canvas is a 2D drawing region. The user can then indefinitely refine the strokes and save poses at anytime, which can then be used for blendshape or keyframe animation. The user can also store the strokes and re-used them in different 3D models that have the same rig structure.

There is an initial setup step that lets the user assign the correspondence between the joints of the 3D model to each canvas. Then, the user is able to start controlling the model through strokes on the pre-defined canvas. The user can sculpt, retarget and animate facial expressions.

4. Definition and Method

The method generates a NURBS curve for each 2D stroke the user draws on a canvas, and associates the stroke to the corresponding control objects of the rig to deform its mesh.

Sketching Representation. We define a stroke S as an ordered set of points $\{s_0,..,s_{n-1}\}$, which is drawn onto a 2D region we call canvas C. Each canvas can support multiple strokes S, which are stored as parametric NURBS curves N. Each curve N is parameterized with t edit points, where t is the total number of joints associated to that canvas during the setup (see Figure 2). We generate the curve N by chosing the edit points ep_i by distance along the total number of points, n, of the original stroke S:

$$ep_i = S\left(\frac{i*(n-1)}{t-1}\right) \qquad i = 0,..,t-1 \qquad n > t \qquad (1)$$

Deformation Window. The shape and position of a stroke affects the associated control object in the 3D model. Thus, the combination of stroke and canvas becomes the effective controller of a region of the 3D model. This model is a 3D face displayed on a separate window, which we call deformation window W. The deformation window W shows in real time the correspondent deformation of the 3D face.

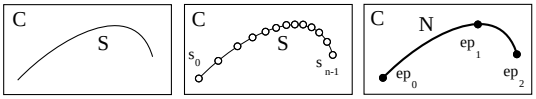

Figure 2: *left: input stroke S in canvas C; middle: sequence of point samples $\{s_0,..,s_{n-1}\}$ that define S. Notice that the samples are spaced irregularly, depending on the drawing speed; right: creation of the NURBS curve with three edit points (ep_0, ep_1, ep_2). This curve corresponds to a region rigged with three joints.*

To compute the deformation window W, we calculate the minimum bounding rectangle (envelope) of the joints initial positions, obtaining the minimum and maximum position values r_0 and r_1 (see Figure 3 first row). To allow the joints to move "outward" of their initial position, but also to support exaggeration of deformations, we expand the window by a displacement value d (for the x and y coordinates). The points w_0 and w_1 are the deformation window limits, which are defined as $w_0 = r_0 - d$, $w_1 = r_1 + d$. Figure 3 second row shows a close-up of the mouth region with different displacement values and consequently different deformation window.

4.1. Mapping Method

Based on these definitions, we additionally define a 2D domain as a 2-tuple (S,C), where S represents the stroke and C is the canvas that contains it. Similarly, a 3D domain is defined as a 2-tuple (R,W), where R represents the rig of the 3D model (joints in our case), and W is the deformation window. The relationship between the 2D and 3D domains defined by the mapping function M, determines the correspondence between the tuples (S,C) and (R,W) (see Figure 3 third row). We split the mapping in two stages: first, using a simple affine mapping, we compute the xy-coordinates of the joints to obtain the XY plane deformation; then, through ray casting techniques, we find the value on the z coordinate to obtain the deformation along the Z axis.

Stage 1: XY-Plane Deformation The method starts by computing the xy-coordinates of the joints of the rig. The rectangular window of the canvas C is mapped to the corresponding rectangular window of the deformation window W, by means of axis-aligned, non-uniform scaling. The method computes the mapping between the curve edit points ep_i and the correspondent joints of the associated region. The new position of the joint j_i on the XY-plane is defined by $j_i = a * ep_i + b$, with $i = 0,..,t-1$, where $a = (w_1 - w_0)/(c_1 - c_0)$ and $b = w_1 - c_1 * a$. The points c_0 and c_1 define the limits of the canvas.

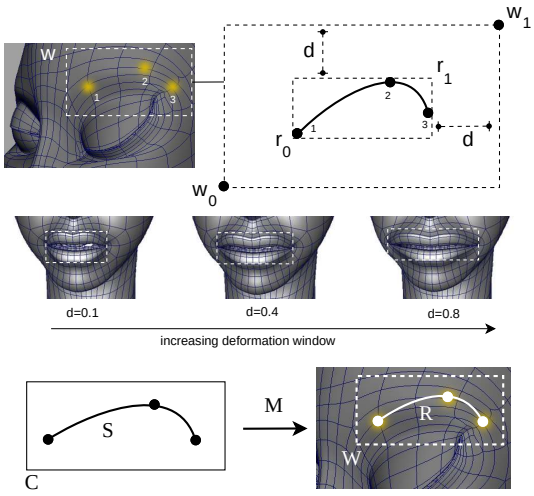

Figure 3: *first row: deformation window W where the joints movement take place, the initial position of the joints are represented by highlighted circles; The value d represents the displacement to expand the deformation area; second row: close-up of the mouth region with different d values on the x coordinate; Notice that the deformation window increases from left to right; third row: represents the mapping M between the canvas C and the stroke S with the deformation window W and the rig R;*

Stage 2: Z Axis Deformation So far, the method has only changed the x and y coordinates of the joints, but to obtain a 3D deformation it needs to change the values on the Z axis. We consider two different deformations in the Z axis: deformation over the surface $Z_{tangent}$ and deformation in *normal* direction of the surface Z_{normal} (see Figure 4).

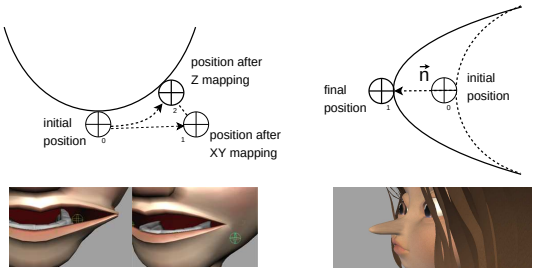

Figure 4: *left: $Z_{tangent}$ deformation over the surface; the lips must follow the contours of the face and not stick out in front of the cheek. right: Z_{normal} deformation in normal direction of the surface; useful if we just want to protrude the nose of Pinocchio.*

The $Z_{tangent}$ deformation is based on ray casting, and it is a z-adjustment over the surface while the user changes the xy-coordinates (see Figure 5). To deform the Z axis we start by duplicating the original 3D model mesh. We call the original mesh Active mesh A_m and the duplicate Reference mesh R_m. We hide R_m and use it only as a local coordinate reference. The mesh that will be deformed is A_m. In A_m, the joint j is moved from its initial position j_0 to position j_1 by the XY deformation part of the method (stage 1). To change the joint z coordinate and get to the final joint position over R_m, the method calculates the auxiliary point p_{aux} in front of the mesh by adding the joint position j_1 to its normal vector $\vec{n}$ ($p_{aux} = j_1 + \vec{n}$). Then, it casts a ray $\vec{r}$ from p_{aux} in the inverse normal direction. The intersection point between $\vec{r}$ and R_m is j_2, the final position of the joint with the z coordinate computed.

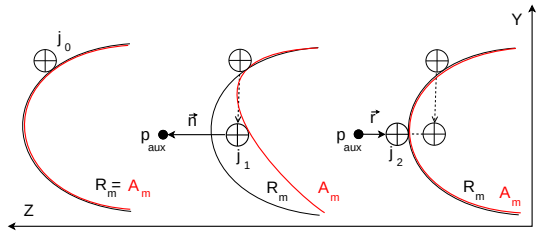

Figure 5: *$Z_{tangent}$ deformation; left: j_0 is the initial position of the joint; middle: j_1 is the position after the xy-mapping, which is not tangent to R_m; right: the final joint position j_2 is calculated according to j_1, its normal vector $\vec{n}$, point p_{aux} and the ray $\vec{r}$ cast from it.*

The Z_{normal} deformation is defined for all those situations when all the user wants is to modify the model in the Z axis and leave the x and y coordinates unchanged (see Pinocchio's nose in Figure 4).

4.2. Collision Detection

Collision detection continues to be a fundamental problem in computer animation and modeling [KHI*07, SPO10]. Finding collisions accurately is key to achieve realistic behavior of 3D model deformations. Next, we provide the details of our collision detection implementation that uses ray casting. The method considers collisions between polygons of different meshes and between polygons of the same mesh (see Figure 6).

We start with two meshes, A_m that is moving towards C_m, and a joint j associated to mesh A_m (see Figure 7). A stroke moves j from position j_0 to position j_1, which defines vector $\vec{s}$ representing the stroke direction. The method casts a ray $\vec{r}$ from j_1 in the direction of $\vec{s}$. The intersection point between $\vec{r}$ and C_m is p_i. The method also takes into account a collision distance defined by the user. This distance determine a collision region around C_m where sketching is not possible. This is necessary to avoid unwanted artifacts, caused by the polygons of A_m around joint j intercepting the surface of C_m. Ray $\vec{r}$, p_i and the collision distance define the collision point

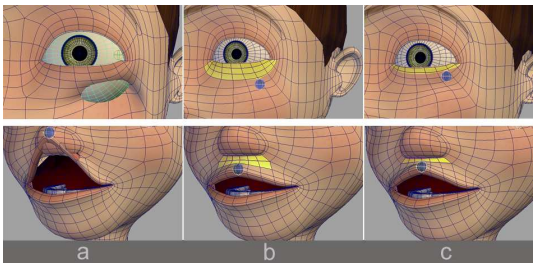

Figure 6: *top row: collision detection between polygons of different meshes (skin-eye); bottom row: collision detection between polygons of the same mesh (skin-skin); From left to right, a) without collision detection; b) and c) collision detection with different collision distances; the yellow areas are collision regions where the joint (blue circle) is not allowed to go.*

p_c. When j reaches p_c the method detects a collision and stops the movement of the mesh. Then, to be able to detect if a new stroke (represented by vector $\vec{s'}$) is a valid action, we define a collision vector $\vec{c}$ as the inverse of $\vec{s}$ ($\vec{c} = -\vec{s}$). If the angle θ between $\vec{s'}$ and $\vec{c}$ is more than 90°, the stroke is invalid because it would take j into the collision region (red vector $\vec{s'}$ in Figure 7). Otherwise the sketching can continue (blue vector $\vec{s'}$ in Figure 7).

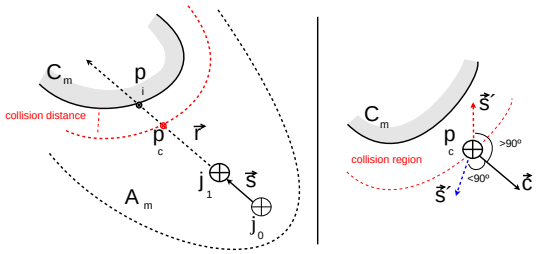

Figure 7: *Collision detection between different meshes. left: collision point p_c is defined by mesh C_m, joint j, stroke vector $\vec{s}$, ray $\vec{r}$ cast in the same direction and the collision distance; right: if the angle between the new stroke ($\vec{s'}$) and collision vector $\vec{c}$ (inverse of $\vec{s}$) is more than 90° the action is invalid; if it is less than 90° the sketching resumes.*

To detect self-collisions the method needs to consider another parameter, *collision height*. The stroke $\vec{s}$ is tangent to the surface of mesh A_m, so collisions between polygons of the mesh occur constantly. We extend the method by using an auxiliary point p_{aux}, calculated by adding a normal vector with collision height magnitude to joint position j_1. Then it is from p_{aux} that we cast the ray $\vec{r}$ to find p_c.

5. Facial Sketching Application

We illustrate the flexibility of our method to build a sketching interface control system for facial animation. It is implemented as a plug-in for Maya 2010, chosen for prototyping and testing purposes. Our system neatly maps the mental concept of a face as a collection of parts. Each part is a facial region, such as, the brows, eyes, nose, cheeks and mouth. We start with a drawing window composed of several canvases, which are depicted as boxes on the background generic face image. Each canvas represents a different facial region of the 3D face model. These regions are enabled every time the user draws a stroke, which automatically deforms the 3D face mesh, creating facial poses. Aditionally, the poses can be transfered to different models using our expression retargeting process. We refer the reader to the accompanying video.

Our system is composed of four modules: Setup, Facial Posing, Expression Retargeting and Facial Animation.

Setup It is the initial step, where the user associates the joints of the rig model to the corresponding canvas. We have implemented a wizard interface to assist the user on the setup process to hide the complexity of 3D modeling and animation. The wizard is prepared so novice users can also configure the system. Additionally, the correspondance between the rig and the canvas can be saved in a XML setup file. Thus, characters with the same rig can use the same file. For example, in a production with 10 different characters that share the same rig we only need to do the setup once. If we have a new character with a different rig, we either do the setup process or apply a rig retargeting technique [OZS08, KMML10] and then use the stored setup file. Now, the 3D model is ready for the creation of facial poses, retargeting and animation.

Facial Posing To create a pose the user need to deform the face model. Deforming the mesh of the face model is a very straightforward and interactive process. First, the user draws a stroke on a canvas. Then the stroke is mapped into the 3D model, which automatically deforms the correspondent facial region of the character. The user can continue drawing strokes on different canvases to generate new deformations. The user can also modify an existent stroke (tottally or partially) and alter the shape of the mesh in real-time. The displacement value d can also be changed in real time using two slide bars on the interface (for the x and y coordinates). If we increase the displacement value d, we are expanding the deformation window W. This is crucial for certain models, like cartoon style characters, in which it is often necessary to exaggerate certain regions of the face. Thus, if we decrease d we shrink the deformation window. Figure 8 shows two strokes around the mouth, converted into two NURBS, and the resulting deformation on the face model mesh with displacement values $d = 0.1$ and $d = 0.8$ for the x coordinate.

Expression Retargeting By facial expression retargeting we mean the action of transferring the facial pose from a

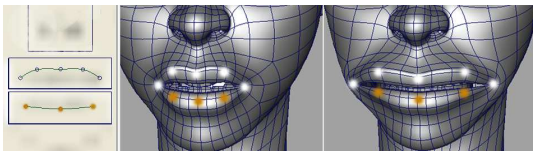

Figure 8: *Close-up of the mouth region, notice how the mouth deformation mimic the shape of strokes (NURBS curve). left: Two strokes around the mouth, upper canvas has a NURBS with five edit points as the rig has five joints and lower canvas has a NURBS with three edit points as the rig has three joints. Joints are represented by white and yellow circles; middle: mouth deformation with $d = 0.1$; right: mouth deformation with $d = 0.8$.*

source to a target model. In our system to retarget facial poses we store in a *pose* file the strokes the user draw in the canvases. For each canvas we save the edit points ep_i used to generate the curve, and the displacement value d used to compute the deformation window W. Then, these data can be reused in different characters as long as they share the same rig. After the curves are loaded into the new 3D model the user can use the sketching tool to delete, modify and redraw the curves to create new poses, or simply modify the d value to adjust the proportions of the deformation.

Facial Animation As an add-on to our sketching applicatin we allow the user to create facial animations using the traditional keyframe technique. Our system was designed having in mind the usability and direct manipulation. After creating all the facial poses the user is able to generate animations by interpolation of poses.

6. Results and Discussion

We have presented a sketching system that allows easy and interactive prototyping of facial poses by drawing strokes on a virtual canvas. The system allows creating facial poses on the fly. It supports non-hierarchy rigs, which makes it very convenient to control 3D models with complex structures, like a face. Our system can handle symmetric and asymmetric characters and is independent of the underlying rig structure. In film and videogame productions, artists are often given one base model to make all new faces (shapes). Also, it is common that afterwards they are asked to create new shapes or modify the existing ones. Currently, the artists would need to fine tune the model by directly modifying the geometry or editing existing controls to reflect the new face, which can take a lot of time. Our method allows artists to redefine the new shape by simply drawing a new pose. Figure 9 shows several facial poses we created with our sketching system. All the characters share the same rig structure, therefore the setup was created only for one model. The model has 29 joints and it took about one minute to setup the rig to the 11 canvases.

One important feature of our system is the *retargeting* of curves. Figure 10 illustrates the retargeting of facial poses between different models. The first column shows the strokes in the canvases; column 2 shows the source model with the pose created with the strokes from column 1, while columns 3 to 7 show the result of the retargeting process. Each row represents a different source model and shows how the retargeting results vary between characters but retain consistency. Notice that when the source and target characters have similar facial proportions the retargeting result is accurate. But, if the facial proportions vary significantly the retargeting process may produce exaggerated deformations. Figure 10 row 4 shows an example of how the deformation in the mouth breaks for extreme poses. The user can fix the pose by modifying the curves or simply by adjusting the displacement value d (see Figure 10 row 5).

Testing and Validation We conducted an informal study focusing on the usability and performance of the system. This informal test involved one group of six graduate students and another group of eight undergraduate students, both with computer graphics background, plus one digital artist and one professional Technical Director. We gave the participants three facial expressions and asked them to recreate each expression, first using our sketching system and then by directly manipulating the rig, both embedded in Maya. Participants used a mouse and a tablet PC, and repeated the test three times; we recorded how long it took them to create the poses each time. This measurement allowed us to roughly estimate the learning curve. There were no precision requirements and they were encouraged to experiment the potential of the system (e.g. creating poses, retargeting) after they finished the three expressions. Note that participants were only instructed with a brief explanation of what they where asked to do, i.e. recreate the facial poses.

The result of this study suggests that our system is indeed likely to be useful for rapid creation of facial poses. Our *expert* participants, the Technical Director and the digital artist, which regularly use 3D animation packages, took roughly less than one minute to complete one pose using our system and an average of five minutes to create the same pose using the individual controls of the rig. We see this as a very positive result as they have years of training using Maya. However, the experts mentioned that to keep precision and speed while sketching, they rather use the strokes with visible edit points (ep), as they map directly to the joints of rig structure and can be tuned individually.

The participants with *average* 3D animation skills (group of graduate and undergraduate students), meaning that they were familiar with 3D animation concepts but do not regularly use 3D animation software, were roughly as efficient as the expert participants: it took them an average of 1.60 minutes to create one pose. The quality of the poses created by this group were of similar quality with the poses created by the expert participants. We believe these are very positive re-

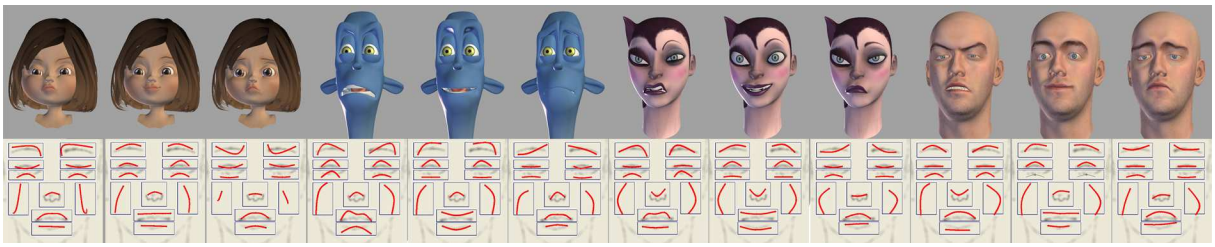

Figure 9: *Facial poses created using our sketching system*

sults. Note that, additionally, these participants did not need instructions to use our tool, but in contrast needed extensive explanations on how to manipulate the rig directly in Maya. It took them about 10 minutes to create a pose in Maya. They said it was difficult to move around the 3D space to manipulate the rig, while sketching in 2D felt much more natural.

The user study presented is not meant to be an exhaustive, formal evaluation, but it reveals relevant information about the usability of our system. We can conclude that non-experts were able to create facial poses in a very short amount of time, without any training period, leveraging the intuitive sketching process, similar to hand-drawing. We expect to perform a thorough test to measure the precision, speed and usability of the system and make it available to the artist community to gather extensive feedback.

Limitations Our system is not free of limitations. We are bounded by the limitations of the underlying rig, so that if its quality is low, our mapping algorithm will still succeed but the results of the deformation may not be satisfactory. Also, our current version of the system does not detect nor fix unexpected strokes, a feature we expect to add soon. Last, to limit the behavior of the model it is necessary to rely on the constraints originally built into the rig; it could be possible to define a separate constraint system for facial animation to work on top of the constraints of the rig.

Future Work A possible future extension is to map motion capture data to the curves and then retarget these curves to different characters. We also aim to generalize the method to control other kind of models like hands or ropes. We will extend our retargeting method to support transfering strokes to different rigs. Last, we intend to support different control objects like wires and lattice, which is actually more of an implementation issue. The sketching system we present can become part of different type of applications like educational learning tools, fast modeling prototyping and game interface. It can also be integrated in a variety of devices like digital tables, mobile phones and iPad.

Acknowledgements

This work is partially supported by Instituto de Telecomunicações, Fundação para a Ciência e Tecnologia (SFRH/BD/46588/2008), the projects LIFEisGAME (ref: UTA-Est/MAI/0009/2009), VERE (ref: 257695) and Golem (ref.251415, FP7-PEOPLE-2009-IAPP). We specially thank Andrew Tucker for the two girls cartoon models.

References

[CJ06] CHANG E., JENKINS O. C.: Sketching articulation and pose for facial animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics SCA* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, Eurographics Association, pp. 271–280. 2

[CSH*92] CONNER B. D., SNIBBE S. S., HERNDON K. P., ROBBINS D. C., ZELEZNIK R. C., VAN DAM A.: Three-dimensional widgets. In *Proceedings of the 1992 symposium on Interactive 3D graphics* (1992), I3D '92, ACM, pp. 183–188. 2

[Hou92] HOUDE S.: Iterative design of an interface for easy 3-d direct manipulation. In *Proceedings of the ACM SIGCHI'92* (1992), pp. 135–142. 2

[IH01] IGARASHI T., HUGHES J. F.: A suggestive interface for 3d drawing. In *Proceedings of the 14th annual ACM symposium on User interface software and technology* (2001), UIST '01, ACM, pp. 173–181. 2

[IH03] IGARASHI T., HUGHES J. F.: Smooth meshes for sketch-based freeform modeling. In *Proceedings of the 2003 symposium on Interactive 3D graphics* (2003), I3D '03, ACM, pp. 139–142. 2

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), SIGGRAPH '99, pp. 409–416. 1

[JS11] JORGE J., SAMAVATI F. F.: *Sketch-based Interfaces and Modeling*. Springer-Verlag New York Inc, May 2011. 2

[KHI*07] KOCKARA S., HALIC T., IQBAL K., BAYRAK C., ROWE R.: Collision detection: A survey. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on* (2007), pp. 4046 –4051. 4

[KMML10] KOMOROWSKI D., MELAPUDI V., MORTILLARO D., LEE G. S.: A hybrid approach to facial rigging. In *ACM SIGGRAPH ASIA 2010 Sketches* (2010), pp. 42:1–42:2. 5

[LCXS09] LAU M., CHAI J., XU Y.-Q., SHUM H.-Y.: Face poser: Interactive modeling of 3d facial expressions using facial priors. *ACM Trans. Graph. 29* (December 2009), 3:1–3:17. 2

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26* (July 2007). 2

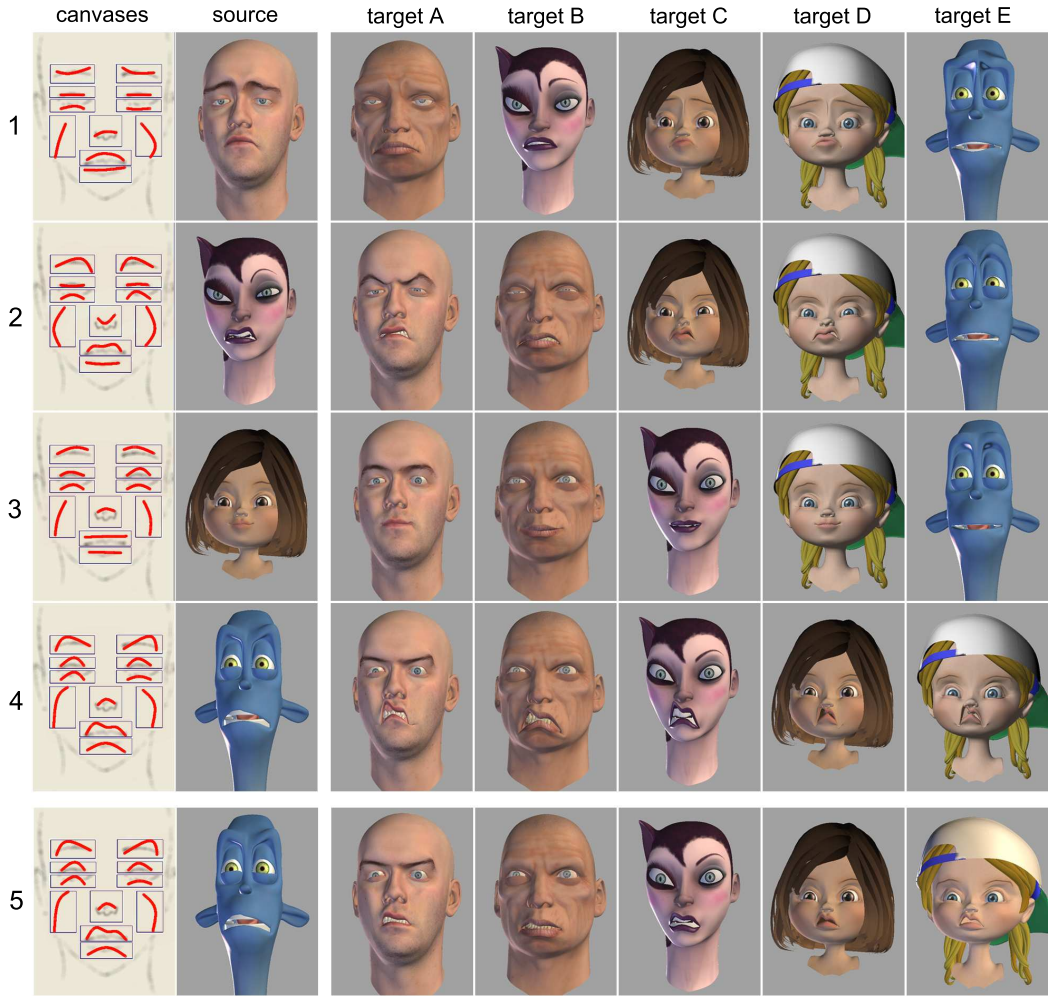

Figure 10: *Retargeting. Rows 1 to 4 show the retargeting of characters with different facial proportions. Row 4 is the most extreme case of retargeting as the source model proportions are very different from the rest of the characters. Thus, the expression on the target characters breaks the mesh (see mouth region). Row 5 fixes the poses by adjusting the deformation window.*

[Nor02] NORMAN D. A.: *The design of everyday things*, 1. basic paperback ed., [nachdr.] ed. Basic Books, 2002. 2

[Osi07] OSIPA J.: *Stop Staring: Facial Modeling and Animation Done Right*. SYBEX Inc., Alameda, CA, USA, 2007. 2

[OSSJ09] OLSEN L., SAMAVATI F., SOUSA M., JORGE J.: Sketch-based modeling: A survey. *Computers & Graphics 33* (2009), 85–103. 2

[OZS08] ORVALHO V. C., ZACUR E., SUSIN A.: Transferring the rig and animations from a character to different face models. *Computer Graphics Forum 27*, 8 (Dec. 2008), 1997–2012. 5

[SMND08] SUCONTPHUNT T., MO Z., NEUMANN U., DENG Z.: Interactive 3d facial expression posing through 2d portrait manipulation. In *Proceedings of graphics interface 2008* (2008), GI '08, pp. 177–184. 2

[SPO10] SCHVARTZMAN S. C., PÉREZ A. G., OTADUY M. A.: Star-contours for efficient hierarchical self-collision detection. In *ACM SIGGRAPH 2010 papers* (2010), pp. 80:1–80:8. 4

[SSB08] SCHMIDT R., SINGH K., BALAKRISHNAN R.: Sketching and composing widgets for 3d manipulation. *Computer Graphics Forum 27*, 2 (2008), 301–310. 2

[Sut64] SUTHERLAND I. E.: Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop* (1964), DAC '64, ACM, pp. 329–346. 2

[TBvdP07] THORNE M., BURKE D., VAN DE PANNE M.: Motion doodles: an interface for sketching character motion. In *ACM SIGGRAPH 2007 courses* (2007), SIGGRAPH '07, ACM. 1

[WWL07] WOBBROCK J. O., WILSON A. D., LI Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of UIST '07* (2007), ACM, pp. 159–168. 2

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: an interface for sketching 3d scenes. In *ACM SIGGRAPH 1996* (1996), SIGGRAPH '96, ACM. 2