Efficient sketch-based character modelling with primitive deformer and shape generator

Anonymous cvm submission

Paper ID ****

Abstract

How to efficiently create detailed 3D models from 2D sketches is an important problem. This paper will propose a new sketch-based and ordinary differential equation (ODE) driven modelling technique to tackle this problem. It consists of two main components: primitive deformer and shape generator. With such a technique, we first draw 2D silhouette contours of a 3D model. Then, we select proper primitives and align them with the corresponding silhouette contours. After that, we develop a sketch-guided and ODE-driven primitive deformer. It uses ODE-based deformations to deform the primitives to exactly match the generated 2D silhouette contours in one view plane and obtain a base mesh of a 3D model consisting of deformed primitives smoothly connected together with ODE-based blending surfaces. In order to add various 3D details, we develop a shape generator which uses 2D sketches in different view planes to define a local shape and employs ODE-driven deformations to create a local surface passing through all the sketches. The experimental results demonstrate that our proposed approach can create detailed 3D models from 2D sketches easily and quickly.

1. Introduction

Main-stream modelling approaches such as polygon and NURBS can create detailed 3D models. However, they require good knowledge and skills to use them, involve heavy manual operations, and take a lot of time to complete modelling tasks. In order to address these problems, various sketch-based modelling approaches have been developed in the past three decades [12].

Although template based modelling [10] has been proposed to create new 3D models by deforming template models to match 2D sketches, most of sketch-based modelling approaches create new 3D models directly from 2D sketches. Among them, many modelling approaches such as Teddy [7] and its descendants [11] follow a "sketch-rotate-sketch" workflow which requires users to sketch from a large number of different views. In spite of the limitation of creating rough shapes, the following limitations of these approaches have been reported in [6]: 1) users cannot match their input strokes to a guide image due to the view changes, 2) how to find a good view for a stroke is often difficult and time consuming.

In order to overcome the limitations caused by the "sketch-rotate-sketch" workflow, a new approach was proposed in [6]. With this approach, geometric primitives are first placed on a 2D image. After that, various annotations including same-lengths and angles, alignment, mirror symmetry, and connection curves are proposed to communicate higher level semantic information, and create 3D models through applying the annotations to the geometric primitives. Since all manual operations are in a fixed view, this approach can create very rough models only.

This paper will take the easiness and efficiency advantages of primitives in representing rough 3D models. It will use primitives to quickly obtain an initial 3D mesh. Then, these primitives are deformed into required shapes by fitting their silhouette contours to 2D sketches. Next, they are smoothly connected together to create a rough 3D model. Ordinary differential equations are widely used to describe various physical laws in scientific calculations and engineering applications. For example, a fourth-order ordinary differential equation is used to describe lateral bending of elastic beams in structural engineering. Therefore, ODE-driven modelling is physics-based and able to generate more realistic appearances and deformations [17]. In order to generate such physically realistic surfaces, we introduce ODE-based modelling to develop a primitive deformer and shape generator which deforms primitives or creates local shapes to match user's drawn sketches.

By integrating sketch-based modelling, primitive deformer, and shape generator, this paper will propose an easy and efficient modelling approach to quickly create detailed 3D models from 2D sketches and primitives. The main contributions made by our proposed approach are summarized below.

- We develop an efficient sketch-guided and ODE-

driven primitive deformer to create a base mesh. It can deform primitives to exactly match the generated silhouette contours. Compared to the existing methods, it automates shape manipulation, avoids tedious manual operations, can deform primitives to match the generated silhouette contours quickly, and is powerful in achieving different shapes of a same primitive.

- We develop a shape generator to add 3D details to the base mesh. Our proposed sketch-guided and ODE-driven shape generator can create a new local shape to match user's drawn sketches in different views quickly.

- We develop a modelling system consisting of primitive deformer and shape generator. With our developed system, detailed 3D models can be created easily and efficiently.

The rest of the paper is organized as follows. The related work is briefly reviewed in Section 2. The system overview of our proposed approach is presented in Section 3. The primitive deformer is examined in Section 4, and the shape generator is investigated in Section 5. Finally, the conclusions and future work are discussed in Section 6.

2. Related Work

The work proposed in this paper is related to sketch-based modelling, primitive-based systems, and ODE-based geometric processing. In what follow, we briefly review the existing work in these fields.

2.1. Sketch-based modelling

Over the past three decades, sketch-based-modelling (SBM) has been widely studied in the computer graphic research community [12]. They can be broadly divided into direct mesh generation and template-based mesh creation.
For direct mesh generation, several systems have been proposed to generate organic models. The surface inflation technique extrudes the polygonal mesh from the skeleton outwards and does a good job in modelling stuffed toys. One trend is to inflate freeform surfaces to create simple stuffed animals and other rotund objects in a sketch-based modelling fashion, such as [7, 11, 8]. The Teddy system [7] presents the pioneer work. It takes closed curves as inputs, finds their cordial axes as spline, then wraps the splines with the polygonal mesh. Later, FiberMesh [11] enriches the editing operations for the inflating base mesh. FiberMesh also presents two types of the control curves: smooth and sharp. A smooth curve constrains the surface to be smooth across it, while a sharp curve only places positional constraints with C0 continuity. Sharp control curves appear when operations like cutting, extrusion and tunnel take place. They also serve the creation of creases on surface. Based on the study from William[15], the SmoothSketch system addresses the problem of T-junction and cusp, which Teddy fails to solve.
For template-based mesh creation, an elegant technique for sketch-based modeling has been proposed in [10] to find precise correspondences and determine mesh deformation. By combining skeleton-based deformation and mesh editing, an efficient approach is proposed in [9] to quickly deform a 3D template model to fit user's drawn sketches.
Among various approaches of direct mesh generation from user's drawn sketches, a variety of sketch-based modelling tools are based on a ?sketch-rotate-sketch? workflow. Such a workflow requires users to draw sketches from many views, causing the difficulties in matching input strokes to a guide image and finding a good view [6].

2.2. Primitive-based systems

Unlike the inflating systems, primitives-based systems deconstruct the modelling task as a process of creating a certain set of geometry primitives and further editing on the primitives. The idea of assembling simple geometry primitives to form 3D models is very common in CSG (constructive solid geometry) modelling, the relevant researches including [13, 4]. Shtof et al. [13] introduces a snapping method which helps determining the position and core parameters of several simple geometry primitives. In [4], the authors provide the tools for generating a cylinder from only 3 strokes: the first two strokes define the 2D profile and the last stroke defines the axis along which the profile curve will sweep. Copies of the profile are not only perpendicularly aligned to the axis, but also resized to snap to the input outlines. However their work is only for man-made objects which simple sweeping surface can meet the quality requirements of the shapes. Structured Annotations for 2D-to-3D Modelling [6], on the other hand, focus on organic modelling. It is a system using two sets of the primitives. One is generalized cylinders, created by the input of a single open sketch stroke representing the spline, and then modified by using simple gestures such as tilting cross sections, scaling local radius, rotating symmetrical plane, and changing cap size. And the other is ellipsoid, generated according to the drawn closed ellipse sketch stroke. As the system?s name indicates, there are a set pf annotation tools to further editing the surface shape using the annotations such as same lengths, same angles, alignment and mirror symmetry.
Using geometric primitives to align 2D sketches provides an easy and efficient approach to generate rough base models. However, manipulating the generated rough base models in one image plane only is difficult to create detailed 3D models.

2.3. ODE-based geometric processing

Ordinary differential equations have been widely applied in scientific calculations and engineering analyses to describe the underlying physics. For example, fourth-order ordinary differential equations have been used to describe the lateral bending deformations of elastic beams. Introducing ODEs into geometric processing can create physically realistic appearances and deformations of 3D models. ODE-based sweeping surfaces were proposed in [17], ODE-based surface deformations were investigated in [18, 3], and ODE-based surface blending was examined in [16].

Although there have been some research studies on ODE-based geometric surface creation and deformations, how to use ODE-based modelling to deform geometric primitives and create new shapes from user?s drawn sketches has not been investigated.

The work given in this paper falls within the category of direct mesh generation from sketches. It integrates quick creation of primitive-based base meshes and efficient and realistic ODE-driven primitive deformations and shape generation from sketches in three orthotropic views to create detailed 3D models.

3. System Overview

Our proposed approach is composed of two main components: ODE-driven primitive deformer and shape generator. The ODE-driven primitive deformer is used to deform primitives to exactly match user?s generated 2D silhouette contours and blend the deformed primitives together to create a base mesh. The ODE-driven shape generator creates new local shapes from three different algorithms. They are local shape creation from: 1) two open silhouette contours in two different view planes, 2) one open and one closed silhouette contour in two different view planes, and 3) two open and one closed silhouette contours in three different view planes.

Taking the creation of a 3D female warrior model as an example, the modelling process using our proposed approach is demonstrated in Figure 1.

First, 2D character silhouette contours are generated. Users can draw their own silhouette contours directly or input their selected sketches. If the selected sketches are input, a further process may be required to extract the 2D silhouette contours from the input sketches. For the example demonstrated in Figure 1, a 2D female warrior sketch shown in Figure 1a is input. Then, users can extract 2D silhouette contours from the input sketch as shown in Figure 1b. After that, proper primitives are selected and placed to align with the corresponding silhouette contours through purely geometric transformations as shown in Figure 1c. Since the silhouette contours of the primitives do not match the generated 2D silhouette contours of the 2D female warrior, the primitive deformer developed from sketch-guided and ODE driven deformations described in Section 4 is applied to deform the primitives to exactly match the corresponding 2D silhouette contours as depicted in Figure 1d. Then, these deformed primitives are smoothly connected together with ODE-based surface blending to create a rough 3D base mesh shown in Figure 1e.

Once a 3D base mesh model is obtained, the shape generator described in Section 5 is employed to add 3D details to the 3D base mesh. These details include the detailed facial shape, two breasts, and thin sheets. The final detailed 3D female warrior model is shown in Figure 1f. In the following two sections, we will introduce in detail the primitive deformer and the shape generator, respectively. Some examples will be presented to demonstrate their applications.

4. Primitive deformer

As shown in Figure 1c, the base mesh of the female warrior without primitive deformations cannot exactly match the generated 2D silhouette contours of the female warrior. In order to tackle this problem, in this section, we develop a primitive deformer. In the subsections below, we first introduce the interface of the primitive deformer in Subsection 4.1. Then, we discuss the algorithm of the primitive deformer in Subsection 4.2.

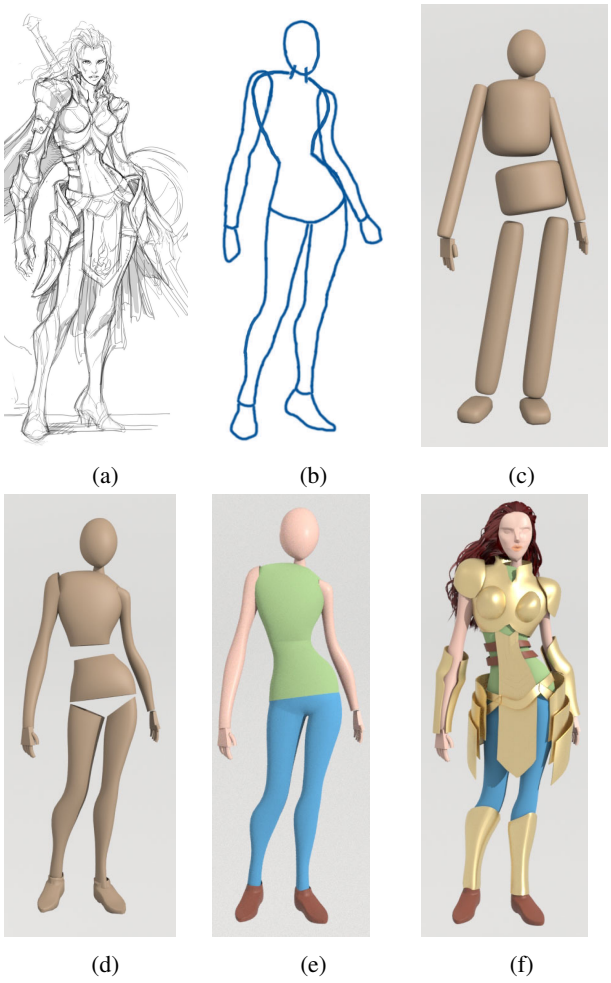
4.1. User interface of primitive deformer

The user interface of our developed primitive deformer uses four windows as shown in Figure 2. The upper left window is used to display 3D base mesh without primitive deformations in the front view. The upper right window is used to draw and edit 2D silhouette contours for these primitives, and the deformed primitives in the front view. The bottom left window is used to draw and edit 2D silhouette contours and deformed the primitives in the die view. The bottom right window is used to draw and edit cross section contours and deform 3D models to obtain realistic shapes.

Taking the left leg of the female warrior as an example Figure 2, the primitive of the left leg is shown in the upper left window. The user-generated 2D silhouette contours in the front view and side view plus the cross section contours are depicted in the top right, bottom left and bottom right windows. Our proposed primitive deformer described in Subsection 4.2 deforms the primitive to exactly match the generated 2D silhouette contours and cross section contours, create a realistic 3D leg model shown in the bottom left window.

4.2. Algorithm of primitive deformer

After 3D primitives have been placed and aligned with the generated 2D silhouette contours, these 3D primitives



(a) (b) (c)

(d) (e) (f)

Figure 1: Quick creation of a 3D female warrior model: (a) 2D female warrior sketch, (b) 2D silhouette contours, (c) base mesh without primitive deformations, (d) base mesh by deforming primitives to match generated 2D silhouette contours, (e) base mesh by adding blending surfaces to smoothly connect deformed primitives, (f) detail generation through local shape creation (Sketch by EngKit Leong)

should be deformed so that their 2D silhouette contours can match the generated 2D silhouette contours exactly. Here we use the example shown in Figure 3 to demonstrate the algorithm of our proposed primitive deformer and how it deforms a 3D primitive to match the 2D silhouette contours.

Figure 3a depicts a torso model of the female warrior which is represented with a cylinder. The 2D silhouette contour to be matched is also shown in the image. Figure 3b shows how the cylinder is deformed with the algorithm developed below to match the 2D silhouette contour exactly.

To tackle the above problem, we propose a sketch-guided

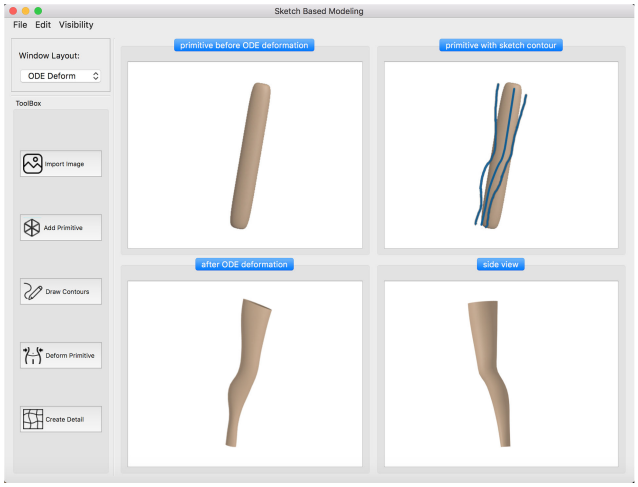


Figure 2: Interface for primitive deformer: (a) 3D base mesh of the left leg of the female warrior without primitive deformations, (b) 2D silhouette contours of the female warrior leg sketch and primitive, (c) and (d) front view and side view of the 3D left leg base mesh after primitive deformations

(a) (b)

Figure 3: Primitive deformer: a) female warrior torso represented with a cylinder and its 2D silhouette contour, b) deformed shape of the cylinder

and ODE-drive primitive deformer. It is developed from a simplified version of the Euler-Lagrange PDE (partial differential equation) which is widely used in physically-based surface deformations and briefly introduced below.

As discussed in [1], the main requirement for physically-based surface deformations is an elastic energy which considers locally stretching for solid objects plus bending for two-manifold surfaces called thin-shells. When a surface $\mathbb{S} \subset \mathbb{R}^3$ parameterized by a function $\mathbf{P}(\mathbf{u}, \mathbf{v}) : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{S} \subset \mathbb{R}^3$ is deformed to a new shape $\mathbb{S}'$ through adding a displacement vector $\mathbf{d}(\mathbf{u}, \mathbf{v})$ to each point $\mathbf{P}(\mathbf{u}, \mathbf{v})$, the change of the first and second fundamental $I(u, v), \Pi(u, v) \in \mathbb{R}^{2\times 2}$ forms in differential geometry [5] yields a measure of stretching and bending descried by [14]

$$E_{shell}(S') = \int_{\Omega} k_s \|I' - I\|_F^2 + k_b \|\Pi' - \Pi\|_F^2 dudv \quad (1)$$

Where ,I',Π' are the first and second fundamental forms of the surface $\mathbf{S}'$, $\|.\|$ indicates a (weighted) Frobenius norm, and the stiffness parameters k_s and k_b are used to control the resistance to stretching and bending.
Generating a new deformed surface requires the minimization of the above equation which is non-linear and computationally too expensive for interactive applications. In order to avoid the nonlinear minimization, the change of the first and second fundamental forms is replaced by the first and second order partial derivatives of the displacement function $\mathbf{d}(\mathbf{u}, \mathbf{v})$ [**?**, **?**], i. e.,

$$\tilde{E}_{shell}(d) = \int_{\Omega} k_s(\|d_u\|^2 + \|d_v\|^2) + k_b(\|d_{uu}\|^2 + 2\|d_{uv}\| + \|d_{vv}\|^2)dudv \tag{2}$$

where $d_x = \frac{\partial}{\partial x}$ and $d_{xy} = \frac{\partial^2}{\partial x \partial y}$. The minimization of the above equation can be obtained by applying variational calculus which leads to the following Euler-Lagrange PDE

$$-k_s \triangle d + k_b \triangle^2 d = 0 \tag{3}$$

where $\triangle$ and $\triangle^2$ are the Laplacian and the bi-Laplacian operator, respectively.

$$\begin{aligned} \triangle d &= div \nabla d = d_{uu} + d_{vv} \\ \triangle^2 d &= \triangle(\triangle d) = d_{uuuu} + 2d_{uuvv} + d_{vvvv} \end{aligned} \tag{4}$$

Using the sketched 2D silhouette contours shown in Figure **????** to change the shape of the primitive can be transformed into generation of a sweeping surface which passes through the two sketched 2D silhouette contours. The generator creating the sweeping surface is a curve of the parametric variable u only, and the two silhouette contours are trajectories. If Equation (3) is used to describe the generator, the parametric variable in Equation (3) drops, and we have $d_{vv} = 0$ and $d_{vvvv} = 0$. Substituting and into Equation (3), we obtain the following simplified version of the Euler-Lagrange PDE (3)which is actually a vector-valued ordinary differential equation

$$k_b \frac{\partial^4 d}{\partial u^4} - k_s \frac{\partial^2 d}{\partial u^2} = 0 \tag{5}$$

As pointed out in [2] , the finite difference solution of ordinary differential equations is very efficient, we here investigate such a numerical solution of Equation (4).

For a typical node shown in Figure 4, the central finite difference approximations of the second and fourth order derivatives can be written as [3]

$$\begin{aligned} \frac{\partial^2 d}{\partial u^2}|_i &= \frac{1}{\triangle u^2}(d_{i+1} - 2d_i + d_{i+1}) \\ \frac{\partial^4 d}{\partial u^4}|_i &= \frac{1}{\triangle u^4}[6d_i - 4(d_{i-1} + d_{i+1}) + d_{i-2} + d_{i+2}] \end{aligned} \tag{6}$$

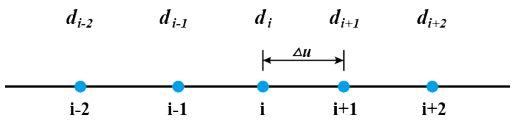
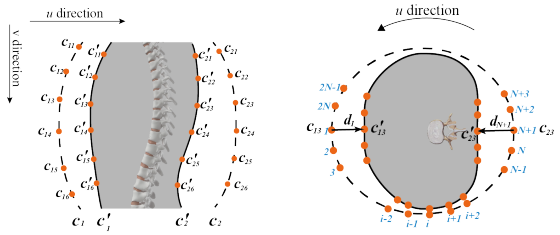


Figure 4: Typical node i for the finite difference approximations of derivatives

Introducing Equation (6) into Equation (5), the following finite difference equation at the representative node i can be written as:

$$\begin{aligned} (6k_b + 2k_s h^2)d_i + k_b d_{i-2} + k_b d_{i+2} \\ -(4k_b + k_s h^2)d_{i-1} - (4k_b + k_s h^2)d_{i+1} = 0 \end{aligned} \tag{7}$$

For character models, the 3D shape defined by two silhouette contours is closed in the parametric direction u as indicated in Figure 5b. Therefore, we can extract some closed curves each of which passes through the two corresponding points on the two silhouette contours. Taking the silhouette contours in Figure 5b as an example, we find two corresponding points $\mathbf{c_{13}}$ and $\mathbf{c_{23}}$ on the original silhouette contours $\mathbf{c_1}$ and $\mathbf{c_2}$, and two corresponding points $\mathbf{c'_{13}}$ and $\mathbf{c'_{23}}$ on the deformed silhouette contours $\mathbf{c'_1}$ and $\mathbf{c'_2}$ as shown in Figure 5b. Then we extract a closed curve $\mathbf{c}(\mathbf{u})$ passing through the two corresponding points $\mathbf{c_{13}}$ and $\mathbf{c_{23}}$ from the 3D model in Figure 5a and depicted it as a dashed curve in Figure 5(b). Assuming that the deformed shape of the closed curve $\mathbf{c}(\mathbf{u})$ is $\mathbf{c}'(\mathbf{u})$, the displacement difference between the original closed curve and deformed closed curve is $\mathbf{d}(\mathbf{u}) = \mathbf{c}'(\mathbf{u}) - \mathbf{c}(\mathbf{u})$. Our task is to find the displacement difference $\mathbf{d}(\mathbf{u})$ and generate the deformed curve $\mathbf{c}'(\mathbf{u}) = \mathbf{d}(\mathbf{u}) + \mathbf{c}(\mathbf{u})$.



(a) side view (b) top view

Figure 5: Finite difference nodes for local shape manipulation from sketches in different view planes and the deformed 3D finger models

In order to use the finite difference method to find the displacement difference $\mathbf{d}(\mathbf{u})$, we uniformly divide the closed curve into $2N$ equal interval as indicated in Figure 5. The

displacement difference at node 1 and node **N** is known, i. e. $\mathbf{d_1} = \mathbf{c'_{13}} - \mathbf{c_{13}}$ and $\mathbf{d_{N+1}} = \mathbf{c'_{23}} - \mathbf{c_{23}}$.

When we write the finite difference equations for the nodes 2, 3, $2N-1$ and $2N$, the node 1 will be involved, and we have $\mathbf{d_1} = \mathbf{c'_{13}} - \mathbf{c_{13}}$. The finite difference equations at these points can be derived from Equation (7). Substituting $\mathbf{d_1} = \mathbf{c'_{13}} - \mathbf{c_{13}}$ into these equations, we obtain the finite difference equations for the nodes 2, 3, $2N-1$ and $2N$

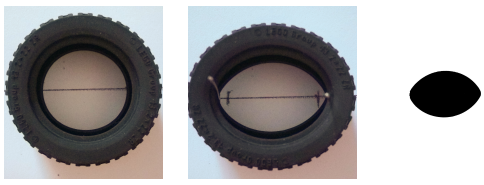
When we write the finite difference equations for the nodes $N-1$,N, $N+2$ and $N+3$, the node $N+1$ will be involved, and we have $\mathbf{d_{N+1}} = \mathbf{c'_{23}} - \mathbf{c_{23}}$. Once again, the finite difference equations at these points can be derived from Equation (7). Substituting $\mathbf{d_{N+1}} = \mathbf{c'_{23}} - \mathbf{c_{23}}$ into these equations, we obtain the finite difference equations for the nodes N-1, N, N+2 and N+3

For all other nodes 3, 4, 5,..., $N-3$, $N-2$ and $N+4$,$N+5$,..., $2N-3$, $2N-2$, the finite difference equations are the same as Equation (7). For these nodes, there are $2N-5$ finite difference equations. Plus the 8 finite difference equations at node 2, 3, $N-1$, N , $N+2$, $N+3$, $2N-1$ and $2N$, we get $2N-2$linear algebra equations which can be solved to determine the unknown constants $\mathbf{d_2}$, $\mathbf{d_3}$, ...,$\mathbf{d_{N-1}}$, $\mathbf{d_N}$, $\mathbf{d_{N+2}}$, $\mathbf{d_{N+3}}$,..., $\mathbf{d_{2N-1}}$, and $\mathbf{d_{2N}}$. Adding the $\mathbf{d_i}$ ($i = 1, 2, \ldots, 2N-1, 2N$) to the original curve $\mathbf{c(u)}$, we obtain the deformed curve $\mathbf{c'(u)}$, and depict it as a solid curve in Figure 5b. Repeating the above operations for all other points on the two silhouette contours, we obtain all deformed curves. These curves describe a new 3D deformed shape.

With the primitive deformer developed above, we deform the primitive of the 3D base mesh shown in Figure 1c, obtain a deformed 3D base mesh, and depicted it in Figure 1d. It is clear that deformed primitives have matched the generated 2D silhouette contours exactly.

The above ODE-driven primitive deformer can create more realistic shapes as demonstrated in Figure 6 and discussed below.

By comparing the red, pink, and green curves, the pink curve is much closer to the red curve than the green curve. It indicates that ODE-driven deformation gives a more realistic appearance.



(a) (b) (c)

Figure 6: Deformation comparison: (a). unreformed toy tyre with an inner diameter 3 cm, (b). deformed toy tyre by moving the horizontal diameter to 3.6 cm, (c) comparison of the inner curves: black-unreformed, red-deformed, pink-ODE-driven deformation, and green-ellipse

When a cross section contour will be used to modify the cross section shape, the influence range of the cross section contour is first specified to generate two boundary curves. The cross section contour and the two boundary curves are parameterized to find the corresponding points. If only position continuity is required, the three corresponding points: one is on each of the two boundary curves and one is on the cross section contour are taken to be the nodes of the finite difference calculations and introduced into Equation(7) to reduce three unknown constants. If both position and tangent continuities are required, the zeroed first partial derivatives at the boundary curves are introduced into Equation(7) to reduce two unknown constants.

With the above ODE-driven deformations, realistic cross section shapes are created. The bottom right window of Figure 2 shows such an example.

With the surface blending method proposed in [16], a smooth transition surface is created between two adjacent deformed primitives. The boundary curves and first partial derivatives are determined from the deformed primitives. Figure 1e gives an example where the disconnected primitives shown in Figure 1d are smoothly connected together.

5. Shape generator

In order to add 3D details to 3D base mesh created from the primitive deformer, we develop a shape generator. In what follows, we first introduce the user interface of our developed shape generator in Subsection5.1. Then we discuss the algorithms of the shape generator in Subsections 5.2.

5.1. User interface of shape generator

The user interface of our developed detail generator uses different windows. They are used to respectively treat local model creation from sketches in different view planes, and 2D image-based detail generation. For local model creation, four windows are used to help provide depth information and control the shape of 3D local details to be created. As shown in Figure 7, the two top windows and left bottom window allow users to sketch 2D silhouette contours from two or all of three orthotropic views: front view, side view, and top view in the local coordinates of a local shape, or project a 3D model to the three orthotropic views to modify them. The right bottom window is used to display the created 3D models.

Taking the breast model as an example, we first sketch a 2D silhouette contour in the front view window as indicted in the upper left part in Figure 7. The sketched silhouette contour is also displayed in the side view window and top view window, which locate at the upper right part of the Figure 7 and lower left part of the Figure 7, respectively. Then one

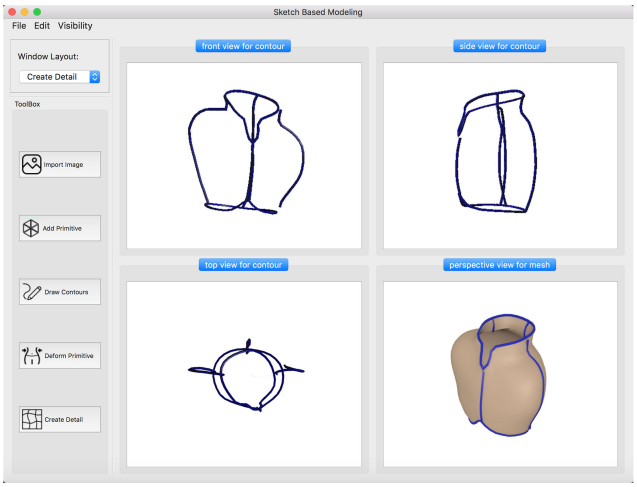


Figure 7: Interface for sketching three orthotropic silhouette contours and viewing generated 3D models

can manipulate the 2D silhouette contours in all three views to your heart's content. Once a 3D model is created from the two or three sketched silhouette contours, it is displayed in the right bottom window of Figure 7.

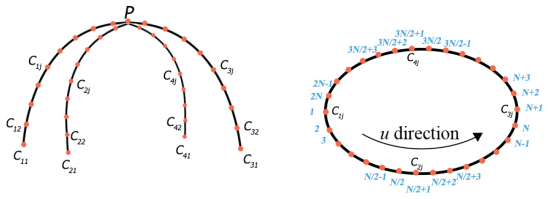
For 2D image-based detail generation, 2D images are input into the left window. After carrying out the calculations of shape-from-shading, the generated 3D models are displayed in the right windows. Users can input a whole character model into the window and perform the operations of adding the generated models to the whole character model.

5.2. local shape creator

Although a 3D regional model can be created from two sketched silhouette contours in the same view plane as discussed in Subsection 4.2, the shape changes in the depth direction may not be our expected ones. In order to allow users to control shape changes in the depth direction and smoothly connect primitives together, we develop a new local shape creator to create a 3D model from two or three silhouette contours in different view planes. It is developed from the four algorithms which can create a model from two open sketches, one open and one closed sketches, two open and one closed sketches, and two closed curves, respectively. In the following four subsections, we will investigate these four algorithms.

5.2.1 Algorithm for two open silhouette contours in two different view planes

For creation of a local 3D model from two open silhouette contours in two different view planes, we use the intersecting point $\mathbf{p}$ to segment each of the two sketched 2D silhouette contours obtained from the front view and the side view into two curves and denote all the four segmented curves with $\mathbf{c_1}$, $\mathbf{c_2}$, $\mathbf{c_3}$ and $\mathbf{c_4}$. Then we find the four corresponding points from the four curves, and denote them with $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$, $\mathbf{c_{3j}}$ and $\mathbf{c_{4j}}$, respectively as shown in Figure 8. Since the four points are on the two sketched 2D silhouette contours in the front view and the side view, their coordinate values are known.



(a) side view (b) top view

Figure 8: Finite difference nodes for the algorithm for two open silhouette contours in two different view planes

The 3D model to be created from the two sketched 2D silhouette contours can be regarded as a sweeping surface whose generator is a closed curve $\mathbf{c(u)}$ shown in Figure 8b passing through the four corresponding points $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$, $\mathbf{c_{3j}}$ and $\mathbf{c_{4j}}$, and shown in Figure 8(a) and (b). We uniformly divide the domain of the parametric variable u corresponding to the closed curve (generator) $\mathbf{c(u)}$ into $2N$ equal intervals. The nodes corresponding to $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$, $\mathbf{c_{3j}}$ and $\mathbf{c_{4j}}$ are 1, $N/2+1$, $N+1$ and $3/2N+1$, respectively as demonstrated in Figure 8(b). Here, the selection of N should ensure that is an even number and $N \geqslant 4$.

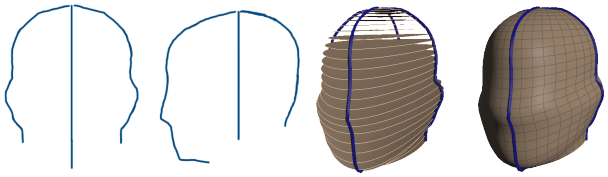
The finite difference equations for the nodes 4, 5, ..., $N/2-3$, $N/2-2$; $N/2+4$, $N/2+5$, ..., $N-3$, $N-2$; $N+4$, $N+5$, ..., $3N/2-3$, $3N/2-2$; $3N/2+2$, $3N/2+3$, ..., $2N-3$, and $2N-2$ are the same as Equation (7).

The finite difference equations for the nodes 2, 3, $N/2-1$, $N/2$, $N/2+2$, $N/2+3$, $N-1$, N, $N+2$, $N+3$, $3N/2-1$, $3N/2$, $3N/2+2$, $3N/2+3$, $2N-1$, and $2N$ can be obtained by introducing the corresponding one of $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$, $\mathbf{c_{3j}}$ and $\mathbf{c_{4j}}$ into Equation (7) to replace d_1 , $d_{N/2+1}$, d_{N+1} and $d_{3N/2+1}$ accordingly.

Putting all obtained finite difference equations together, we obtain all the unknown constants $\mathbf{d_2}$, $\mathbf{d_3}$, ..., $\mathbf{d_{N/2-1}}$, $\mathbf{d_{N/2}}$, $\mathbf{d_{N/2+2}}$, $\mathbf{d_{N/2+3}}$, ..., $\mathbf{d_{N-1}}$, $\mathbf{d_N}$, $\mathbf{d_{N+2}}$, $\mathbf{d_{N+3}}$,..., $\mathbf{d_{3N/2-1}}$, $\mathbf{d_{3N/2}}$, $\mathbf{d_{3N/2+2}}$, $\mathbf{d_{3N/2+3}}$, ..., $\mathbf{d_{2N-1}}$ and d_{2N} . They together with the known four points $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$, $\mathbf{c_{3j}}$ and $\mathbf{c_{4j}}$ are used to define the generator.

Repeating the above treatment for all other points on the four curves $\mathbf{c_1}$, $\mathbf{c_2}$, $\mathbf{c_3}$ and $\mathbf{c_4}$, we obtain the generators at the other positions. With these generators, we create local 3D shapes.

A head model is used to demonstrate the above method

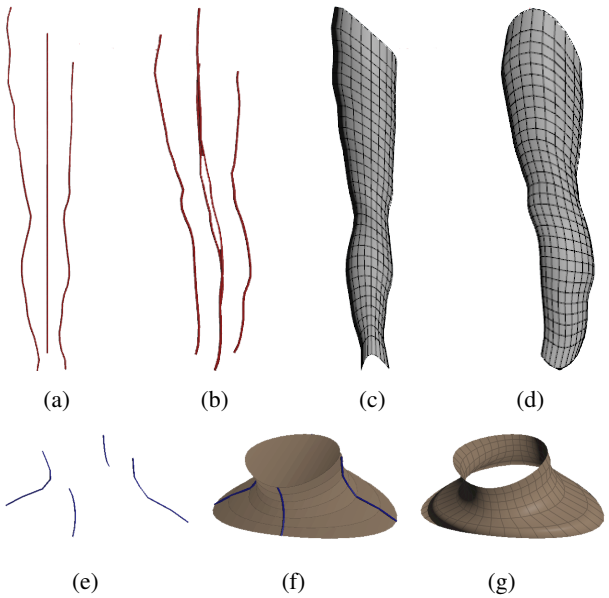


(a) (b) (c) (d)

Figure 9: Primitive generator: a) 2D silhouette contours of the male head in front view, b)-2D silhouette contours of the male head in front view, c) cross sections of the male head d) lofting the cross sections into a male head mesh

Figure 9. Figure 9a and Figure 9b show two open head silhouette contours in two different view plans. Figure 9c shows the cross section for each discrete parameter u_i. Figure 9d depicted the created 3D head model.

The above method can be directly extended to deal with local shape creation from four disconnected silhouette contours. The silhouette contours of the leg model in the front and side views are shown in Figure 10a and Figure 10b, respectively. The corresponding four points on the four silhouette contours are used to define a cross-section curve. All of these cross section curves are used to generate the 3D leg model with the front and side views in Figure 10c and Figure 10d, respectively. Another example given in Figure 10 is a neck generated in the same fashion where Figure 10e shows the 4 disconnected contours, Figure 10f shows the cross-sections for each discrete u and finally the mesh is in Figure 10g



(a) (b) (c) (d)

(e) (f) (g)

Figure 10: leg model creation from two open silhouette contours in two different view planes

5.2.2 Algorithm for one open and one closed silhouette contours in two different view planes

For creation of a local 3D model from one open and one closed silhouette contours in two different view plans as shown in Figure 11, the intersecting points $\mathbf{p_1}$ and $\mathbf{p_3}$ between the open silhouette contour and the closed silhouette contour divide the closed silhouette contour into two curves. Then we find the middle points $\mathbf{p_2}$ and $\mathbf{p_4}$ of the two curves. These four points $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$ and $\mathbf{p_4}$ divide the closed silhouette contour into four curves $\mathbf{c_1}$, $\mathbf{c_3}$, $\mathbf{c_4}$ and $\mathbf{c_6}$. Next, we find the middle point $\mathbf{p}$ of the open silhouette contour which divides the open silhouette contour into two curves $\mathbf{c_2}$ and $\mathbf{c_5}$. With this treatment, the creation of the local 3D detail model is changed into creation of two sweeping surfaces: one is defined by the three curves $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$, and the other is defined by the three curves $\mathbf{c_4}$, $\mathbf{c_5}$ and $\mathbf{c_6}$.

Since the creation process of the two weeping surfaces is the same, we take the creation of the sweeping surface defined by the three curves $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$ to demonstrate the process.

For each of the three curves $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$ we uniformly divide it into $2N$ equal intervals, and use $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$ and $\mathbf{c_{3j}}(\mathbf{j = 1, 2, \ldots, J})$ to respectively indicate the nodes on the three curves.

The sweeping surface can be generated by sweeping a generator $\mathbf{c(u)}$ passing through the points$\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$ and $\mathbf{c_{3j}}$. Based on this consideration, the creation of the sweeping surface is transformed into determination of the generator at different positions along the curve.

In order to create the generator $\mathbf{c(u)}$ passing the points $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$ and $\mathbf{c_{3j}}$, we uniformly divide the domain of the parametric variable u corresponding to the generator $\mathbf{c(u)}$ into 2N equal intervals, and get the nodes $1, 2, \ldots, 2N-1$, $2N$ and $2N+1$. Since the nodes 1, $N+1$, and $2N+1$ are on the curves $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$, their values are known, i. e.$\mathbf{d_1 = c_{1j}}$, $\mathbf{d_{N+1} = c_{2j}}$ and $\mathbf{d_{2N+1} = c_{3j}}$. When writing the finite difference equations for the node 2 and the node $2N$, the node 0 beyond the boundary node 1 and the node $2N+2$ beyond the boundary node $2N+1$ will be involved. They can be determined below.

If the created local 3D model is to be smoothly connected to another 3D model, we can obtain a closed curve close to but larger than the closed silhouette contour from another 3D model. Then, the vector-valued first derivative $\mathbf{T_{1j}}$ on the curve $\mathbf{c_1}$ and $\mathbf{T_{3j}}$ on the curve $\mathbf{c_3}$ can be calculated from the closed curve and the closed silhouette contour.

If the created local 3D model is to be connected to another 3D model with positional continuation only, the vector-valued first derivative $\mathbf{T_{1j}}$ on the curve $\mathbf{c_1}$ and $\mathbf{T_{3j}}$

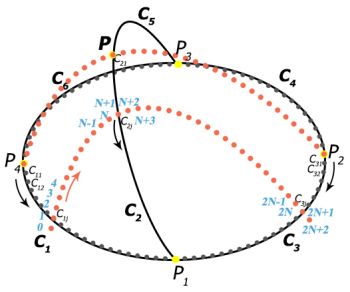


Figure 11: Finite difference nodes for Algorithm for one open and one closed silhouette contours

on the curve $\mathbf{c_3}$ can be estimated from the three points $\mathbf{c_{1j}}$, $\mathbf{c_{2j}}$ and $\mathbf{c_{3j}}$ by first constructing a curve passing through the three points and then calculating the vector-valued first derivative of the constructed curve at the points $\mathbf{c_{1j}}$ and $\mathbf{c_{3j}}$. Once the vector-valued first derivatives $\mathbf{T_{1j}}$ and $\mathbf{T_{3j}}$ are obtained. The nodes 0 and $2N + 2$ can be determined by the following finite difference approximation.

$$\mathbf{T_{1j}} = \frac{\mathbf{d_2} - \mathbf{d_0}}{2\triangle u}, \mathbf{T_{3j}} = \frac{\mathbf{d_{2N+2}} - \mathbf{d_{2N}}}{2\triangle u} \tag{8}$$

Solving the above equation, we obtain the nodes 0 and $2N+2$ by

$$\begin{aligned} \mathbf{d_0} &= \mathbf{d_2} - 2\mathbf{T_{1j}}\triangle u \\ \mathbf{d_{2N+2}} &= \mathbf{d_{2N}} + 2\mathbf{T_{3j}}\triangle u \end{aligned} \tag{9}$$

The finite difference equations for nodes 4, 5, ..., $N - 3$, $N - 2$ and $N + 4$, $N + 5$, ..., $2N - 3$, $2N - 2$ are the same as Equation (7).

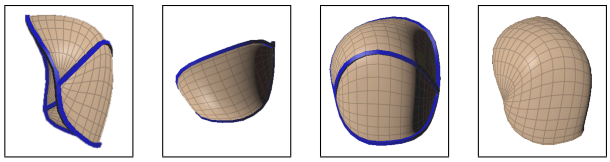
The finite difference equations for the nodes 2, 3, $N - 1$, N , $N + 2$, $N + 3$, $2N - 1$ and $2N$ can be obtained by introducing the above equation and $\mathbf{d_1} = \mathbf{c_{1j}}$, $\mathbf{d_2} = \mathbf{c_{2j}}$ and $\mathbf{d_3} = \mathbf{c_{3j}}$ into Equation (7).

Solving the finite difference equations for all the inner nodes 2, 3, ..., $2N - 1$ and $2N$, we obtain all the unknown constants d_2 , d_3 , ..., d_{2N-1} and d_{2N}. They together with the three known points c_{1j} , c_{2j} and c_{3j} are used to create the generator.

Using the same treatment, we can obtain the generator at the other positions. From these obtained generators, a local detail 3D model is created and depicted in Figure 12b.

5.2.3 Algorithm for two open and one closed silhouette contours in three different view planes

By introducing depth information, users can have more power to control the shape of the 3D model to be created as demonstrated by the algorithms given in Subsections 5.2.1 and 5.2.2. Users can further control the shape of 3D models by using one or more silhouette contours in other view planes. In this subsection, we discuss construct a 3D detail model from two open and one closed silhouette contours in three orthotropic view planes.



(a) front (b) top (c) side (d) perspective

Figure 12: creation of one open and one closed silhouette contours in four different view planes

As shown in Figure 13, the task of creating a local 3D detail model passing through two open and one closed silhouette contours can be transformed into constructing 4 sweeping surfaces encircled by the curves $\mathbf{c_1c_6c_5}$, $\mathbf{c_2c_7c_6}$, $\mathbf{c_3c_8c_7}$, and $\mathbf{c_4c_5c_8}$ respectively.

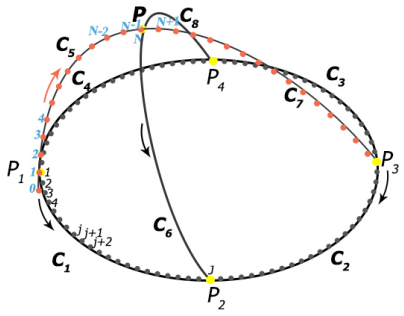


Figure 13: Finite difference nodes for Algorithm for two open and one closed silhouette contours

Since the construction algorithm for the 4 sweeping surfaces is the same, we take the sweeping surface defined by the three curves $\mathbf{c_1}$, $\mathbf{c_6}$ and $\mathbf{c_5}$ to demonstrate the construction algorithm. This algorithm will require reconstruction of the curves $\mathbf{c_5}$ and $\mathbf{c_6}$ to get additional information called sculpting forces for generating the sweeping surface.

In order to reconstruct the curves $\mathbf{c_5}$ and $\mathbf{c_6}$ accurately, we modify the vector-valued ordinary differential equation (5) by introducing a sculpting force $\mathbf{f}$ and obtain

$$k_b \frac{\partial^4 d}{\partial u^4} - k_s \frac{\partial^2 d}{\partial u^2} = f \tag{10}$$

Accordingly, the finite difference equation for the above ordinary differential equation becomes

$$(6k_b + 2k_s h^2)d_i + k_b d_{i-2} + k_b d_{i+2} \\ -(4k_b + k_s h^2)d_{i-1} - (4k_b + k_s h^2)d_{i+1} = \triangle u^4 f_i \quad (11)$$

The reconstruction algorithm for the curve $\mathbf{c_6}$ is exactly same as that of the curve $\mathbf{c_5}$. Here we take the reconstruction of the curve $\mathbf{c_5}$ to demonstrate the algorithm.

Similar to the previous treatment, we uniformly divide the domain of the parametric variable u for the curve $\mathbf{c_5(u)}$ into $N-1$ equal intervals. The vector-valued first derivatives of the curves $\mathbf{c_5(u)}$ at the node 1 and N can be determined from the curve $\mathbf{c_5(u)}$ and indicated by $\mathbf{T_{5,0}}$ and $\mathbf{T_{5,1}}$.

When writing the finite difference equations for the inner nodes 2, and $N-1$, the node 0 beyond the boundary node 1 and the node $N+1$ beyond the boundary node N will be involved. We determine the nodes 0 and N+1 from the vector-valued first derivatives $\mathbf{T_{5,0}}$ and $\mathbf{T_{5,1}}$ through

$$\mathbf{T_{5,0}} = \frac{\mathbf{d_2} - \mathbf{d_0}}{2\triangle u} \\ \mathbf{T_{5,1}} = \frac{\mathbf{d_{N+2}} - \mathbf{d_N}}{2\triangle u} \quad (12) \\ \mathbf{d_0} = \mathbf{d_2} - 2\mathbf{T_{5,0}}\triangle u \\ \mathbf{d_{N+1}} = \mathbf{d_{N-1}} + 2\mathbf{T_{5,1}}\triangle u$$

For each of the inner nodes 2, 3, ..., $N-2$, $N-1$, we can write a finite difference equation. Since the coordinate values for all the nodes on the curve $\mathbf{c_6}$ are known, we can calculate the sculpting force $\mathbf{f_i}(i = 2, 3, \ldots, N-2, N-1)$ from these equations,. We denote these sculpting forces as $\mathbf{f_{5,i}} = \mathbf{f_i}(i = 2, 3, \ldots, N-2, N-1)$. With the same method, we can obtain the sculpting forces $\mathbf{f_{5,i}} = \mathbf{f_i}(i = 2, 3, \ldots, N-2, N-1)$ acting on the curve c_6.

The curves $\mathbf{c_5}$ and $\mathbf{c_6}$ can be regarded as generators. The generation of the sweeping surface defined by the three curves $\mathbf{c_1}$, $\mathbf{c_6}$ and $\mathbf{c_5}$ is to sweep the generator from the curve $\mathbf{c_5}$ to the curve $\mathbf{c_6}$ along the curve $\mathbf{c_1}$ and the point $\mathbf{p}$. In order to determine the shape of the generator at different positions along the curve $\mathbf{c_1}$, we uniformly divide the curve $\mathbf{c_1}$ into J equal intervals and obtain the nodes $j = 1, 2, 3, \ldots, J-2, J-1, J$ where $j = 1$ and $j = J$ are the intersecting points between the curves $\mathbf{c_5}$ and $\mathbf{c_1}$ and between $\mathbf{c_6}$ and $\mathbf{c_1}$. Then we determine the shape of the generator between the node $j(j = 2, 3, \ldots, J-2, J-1, J)$ and the point $\mathbf{p}$.

With the same treatment, we divide the domain of the parametric variable u for the generator between the node j and the point $\mathbf{p}$ into $N-1$ equal intervals (the node j on the curve $\mathbf{c_1}$ is the node 0 on the generator between the node j and the point $\mathbf{p}$). When sweeping the curve $\mathbf{c_5}$ along the curve $\mathbf{c_1}$ to the curve $\mathbf{c_6}$, the sculpting force $\mathbf{f_{5,i}}$ acting at the node i of the curve is gradually changed to the sculpting force $\mathbf{f_{6,i}}$ acting at the node i of the curve $\mathbf{c_6}$. Here we use a linear interpolation to describe the gradual change and obtain the sculpting force $\mathbf{f_i}$ below acting at the node i of the generator between the node j and the point $\mathbf{p}$

$$\mathbf{f_i} = \mathbf{f_{5,i}} + \frac{L_j}{L}(\mathbf{f_{6,i}} - \mathbf{f_{5,i}}) \quad (13)$$

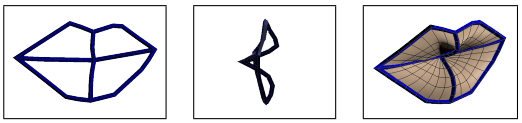
where L_j is the length from the point $\mathbf{p_1}$ to the node j and L is the length from the point $\mathbf{p_1}$ to the point $\mathbf{p_2}$. When sweeping the curve $\mathbf{c_5}$ along the curve $\mathbf{c_1}$ to the curve $\mathbf{c_6}$, the vector-valued first derivatives of the curve $\mathbf{c_5}$ at the points $\mathbf{p}$ and $\mathbf{p_1}$ are also gradually changed to the vector-valued first derivatives of the curve $\mathbf{c_6}$ at the points $\mathbf{p}$ and $\mathbf{p_2}$. The same linear interpolation is used to describe such a gradual change and determine the vector-valued first derivatives $\mathbf{T_0}$ and $\mathbf{T_1}$ of the generator at the node j and the point $\mathbf{p}$.

$$\mathbf{T_0} = \mathbf{T_{5,0}} + \frac{L_j}{L}(\mathbf{T_{6,0}} - \mathbf{T_{5,0}}) \\ \mathbf{T_1} = \mathbf{T_{5,1}} + \frac{L_j}{L}(\mathbf{T_{6,1}} - \mathbf{T_{5,1}}) \quad (14)$$

Having known the sculpting forces at all the inner nodes 2, 3, ..., $N-2$, $N-1$, and the coordinate values at the boundary nodes 1 and N and the nodes 0 and $N+1$ beyond the boundary nodes, we can write the finite difference equations for all the inner nodes where the finite difference equations for the nodes 4, 5,..., $N-4$, $N-3$ are the same as Equation (10) with the sculpting force $\mathbf{f_i}$ being calculated by Equation (13)

Solving all the finite difference equations, we obtain all the unknown constants $\mathbf{d_2}$, $\mathbf{d_3}$, ..., $\mathbf{d_{N-2}}$ and $\mathbf{d_{N-1}}$. They together with the two known points $\mathbf{d_1}$ and $\mathbf{p}$ are used to create the generator.

With the same method, we obtain all the curves of the generator at the positions $j = 1, 2, 3, \ldots, J-2, J-1, J$. They are used to create the sweeping surface shown in Figure 13.



(a) front (b) side (c) perspective

Figure 14: mouth model creation from two open and one closed silhouette contours in three different view planes

With the above method, we draw a closed lip contour and two open curves shown in Figure 14a and 14a. The two open curves divide the mouth into four regions. One surface is created for each of the four regions. Figure 14c depicts the mouth model consisting of the four surfaces.

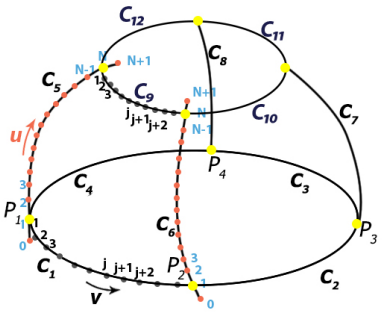
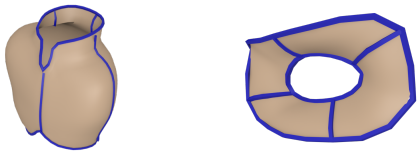


Figure 15: Finite difference nodes for Algorithm for four open and two closed silhouette contours

The above method can be also extended to deal with the situations where the two open curves do not intersect. For such situations, the intersecting point of the two open curves becomes the upper closed curve as indicated in Figure 15, and a sweeping surface is encircled by four curves. With this extension, we created a 3D vest model and an eye socket model depicted it in Figure 16a and Figure 16b, respectively.



(a) vest (b) eye socket

Figure 16: vest and eye socket creation from four open and two closed silhouette contours

5.2.4 Algorithm for patch surfaces

We introduce a new physic-based ODE-driven algorithm for making patch surfaces. When a patch is surrounded by other surfaces with shared borders, in order to maintain the C1 continuity(the left and right tangent are equal) between these surfaces, our method will extract a curve from the adjacent surfaces as a virtual border. One can see from the Figure **??**, a virtual border $\mathbf{c_5c_6}$ is a little bit far away beyond the actual border $\mathbf{c_3c_4}$. Taken the virtual border's coordinate values into the constraint matrix formation, solve this linear equation and the solution to that equation is the position of every vertex. We will explain how to form this constraint matrix more detailed later in this subsection. This virtual border can also be provided by users if there is no adjacent surfaces around but still user want to have more control over the surface shape near the border. Unlike in the typical mesh optimisation problems where C1 continuity is required on the joint border, our algorithm enables users to refine the border areas with or without the existence of adjacent patches. In addition, compared to the typical way of solving the border continuity optimisation[19], our method is computational cheaper because though both required forming a constraint matrix as a coefficient matrix of a linear equation, instead of getting the displacements of the control points and then computing the spline surface out of the modified control points, our method get the vertex positions of the final parametric surface straight away.

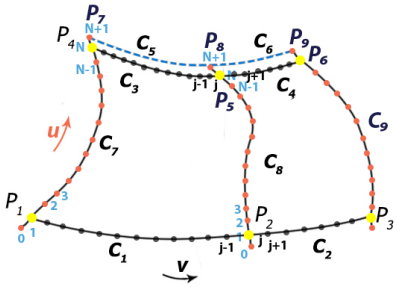


Figure 17: Finite difference nodes for Algorithm for patch surfaces

As demonstrated in Figure **??**, a curve network is constituted of $\mathbf{c_1c_2}$, $\mathbf{c_3c_4}$ in the u direction, and $\mathbf{c_7}$, $\mathbf{c_8}$, $\mathbf{c_9}$ on the v direction. Above $\mathbf{c_3c_4}$ there is a virtual curve $\mathbf{c_5c_6}$ which helps to define the shape near the patch border $\mathbf{c_3c_4}$. The sweeping surface is composed of surface encircled by the curves $\mathbf{c_7c_1c_8c_5}$, and $\mathbf{c_8c_2c_9c_4}$ respectively.
Since the construction algorithm for the 2 sweeping surfaces is the same, we'll take the sweeping surface defined by the four curves $\mathbf{c_1}$, $\mathbf{c_8}$, $\mathbf{c_5}$ and $\mathbf{c_7}$ to demonstrate the construction algorithm. We uniformly divide the domain of the parametric variable u for the curve $\mathbf{c_7(u)}$ into $N-1$ equal intervals. But since there is a virtual border $\mathbf{c_5}$ above $\mathbf{c_3}$, the end point of $\mathbf{c_5}$ extend $\mathbf{c_7}$ and contributes a nodes $N+1$ to $\mathbf{c_7}$. Same applies to $\mathbf{c_8}$. On the v direction, we uniformly divided parametric variable v domain into $j-1$ equal intervals. We will calculate sculpting forces from the positional information of curves $\mathbf{c_7}$ and $\mathbf{c_8}$, the algorithm to get the sculpting force is given in Subsections 5.2.3.
The vector-valued first derivatives of the curves $\mathbf{c_7(u)}$ at the node 1 can be determined from the curve $\mathbf{c_7(u)}$ and indicated by $\mathbf{T_{5,0}}$, as that described in Subsection 5.2.3 and 5.2.2. Since we want to make sure the two tangents of border curve $\mathbf{c_3}$ from both sides equal to each other, for every parametrc u, the vector-valued first derivatives at node N should conform with the constraint listed below.

$$\frac{\mathbf{Node_{N+1}} - \mathbf{Node_N}}{\widehat{Node_{N+1}Node_N}} = \frac{\mathbf{Node_N} - \mathbf{Node_{N-1}}}{\widehat{Node_NNode_{N-1}}} \quad (15)$$

So that the vector-valued first derivatives $\mathbf{T_{5,1}}$ can be determined by the position of point P_7 and P_4 as described in Equation (16).

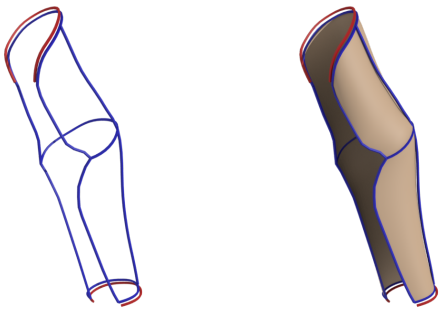
$$\mathbf{T_{5,1}} = \frac{\mathbf{P_7} - \mathbf{P_4}}{\widehat{P_7P_4}} \approx \frac{\mathbf{P_7} - \mathbf{P_4}}{|P_7P_4|} \tag{16}$$

When writing the finite difference equations for the inner nodes 2, and $N-1$, the node 0 beyond the boundary node 1 and the node $N+1$ beyond the boundary node N will be involved. We can use the method given in Subsection 5.2.3 to determine the vector-valued first derivatives $\mathbf{T_{5,0}}$. Hence, we can write equations for nodes 0 and $N+1$ Equation (17)

$$\begin{aligned} \mathbf{d_0} &= \mathbf{d_2} - 2\mathbf{T_{5,0}}\triangle u \\ \mathbf{d_N} &= \mathbf{d_{N-1}} + \mathbf{T_{5,1}}\triangle u \end{aligned} \tag{17}$$

Also, we need to calculate the sculpting forces for all the nodes on the curve $\mathbf{c_7}$ and $\mathbf{c_8}$, denoting as $\mathbf{f_{7,i}} = \mathbf{f_i}(i = 2, 3, \ldots, N-2, N-1)$ and $\mathbf{f_{8,i}} = \mathbf{f_i}(i = 2, 3, \ldots, N-2, N-1)$. Then we can calculate the sculpting force for each inner nodes 2, 3, ..., $N-2$, $N-1$, denoting as $\mathbf{f_i}(i = 2, 3, \ldots, N-2, N-1)$ from these equations by linear interpolating $\mathbf{f_{7,i}}$ and $\mathbf{f_{8,i}}$ with the same treatment as described in section 5.2.3. Now, for each of the inner nodes 2, 3, ..., $N-2$, $N-1$, we can write a finite difference equation just as that in Subsection 5.2.3.

An example of the above method is given in Figure 18, where the curves in red are virtual curves and the curves in blue are the actual curves forming a patch surface.



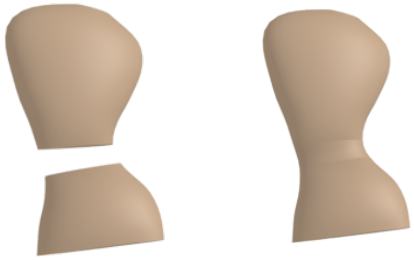
(a) contours (b) mesh

Figure 18

5.2.5 Algorithm for two closed curves

This algorithm is to create smooth transition surfaces between two disconnected primitives. Users can interactively draw two closed curves on two disconnect primitives or use two planes to intersect the two primitives to obtain two boundary curves of the transition surface. In order to obtain smooth transition between the two primitives and the transition surface, the tangents at the two boundary curves are obtained from the two primitives. With the two boundary curves and the tangents at the two boundary curves as the boundary conditions, the ODE-based surface blending method introduced in [16] is used to create the smooth transition surface which smoothly connects two primitives together. Figure 19 gives such an example.

By adding local shapes and smooth transition surface, the base mesh shown in Figure 1d is changed into that depicted in Figure 1e.



(a) detached primitives (b) add transition surface

Figure 19: Creation of smooth transition surface between primitives

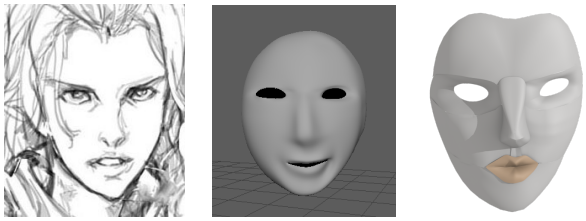
6. Conclusions and Future work

Compared to traditional polygon modelling technology, our method has these advantages:

- quicker in modelling complicated part of character models
- easier for beginner
- more precise in term of detail creation more precise in term of detail creation, for example the face. Figure 20a is the reference sketch, the Figure 20b shows the work from an amateur modeller did the traditional face modelling in Autodesk Maya 2018, and the Figure 20c shows the work from the same person, but did the modelling with our system.

Acknowledgement

This work is supported by the funding to be detailed after the anonymous review.



(a) (b) (c)

Figure 20: a. sketch of a face as a reference image; b. modelling in traditional polygon 3D software package; c. modelling in our sketch-based-modelling system

References

[1] M. Botsch and O. Sorkine. On linear variational surface deformation methods. *IEEE transactions on visualization and computer graphics*, 14(1):213–230, 2008. 4

[2] E. Chaudhry, S. Bian, H. Ugail, X. Jin, L. You, and J. J. Zhang. Dynamic skin deformation using finite difference solutions for character animation. *Computers & Graphics*, 46:294–305, 2015. 5

[3] E. Chaudhry, L. You, X. Jin, X. Yang, and J. J. Zhang. Shape modeling for animated characters using ordinary differential equations. *Computers & Graphics*, 37(6):638–644, 2013. 3, 5

[4] T. Chen, Z. Zhu, A. Shamir, S.-M. Hu, and D. Cohen-Or. 3-sweep: Extracting editable objects from a single photo. *ACM Transactions on Graphics (TOG)*, 32(6):195, 2013. 2

[5] M. P. Do Carmo, G. Fischer, U. Pinkall, and H. Reckziegel. Differential geometry. In *Mathematical Models*, pages 155–180. Springer, 2017. 4

[6] Y. Gingold, T. Igarashi, and D. Zorin. Structured annotations for 2d-to-3d modeling. In *ACM Transactions on Graphics (TOG)*, volume 28, page 148. ACM, 2009. 1, 2

[7] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999. 1, 2

[8] O. A. Karpenko and J. F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, July 2006. 2

[9] I. K. Kazmi, L. You, X. Yang, X. Jin, and J. J. Zhang. Efficient sketch-based creation of detailed character models through data-driven mesh deformations. *Computer Animation and Virtual Worlds*, 26(3-4):469–481, 2015. 2

[10] V. Kraevoy, A. Sheffer, and M. van de Panne. Modeling from contour drawings. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based interfaces and Modeling*, pages 37–44. ACM, 2009. 1, 2

[11] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM transactions on graphics (TOG)*, 26(3):41, 2007. 1, 2

[12] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, 2009. 1, 2

[13] A. Shtof, A. Agathos, Y. Gingold, A. Shamir, and D. Cohen-Or. Geosemantic snapping for sketch-based modeling. In *Computer graphics forum*, volume 32, pages 245–253. Wiley Online Library, 2013. 2

[14] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *ACM Siggraph Computer Graphics*, volume 21, pages 205–214. ACM, 1987. 4

[15] L. R. Williams and D. W. Jacobs. Stochastic completion fields: A neural model of illusory contour shape and salience. *Neural computation*, 9(4):837–858, 1997. 2

[16] L. You, H. Ugail, B. Tang, X. Jin, X. You, and J. J. Zhang. Blending using ode swept surfaces with shape control and c1 continuity. *The Visual Computer*, 30(6-8):625–636, 2014. 3, 6, 12

[17] L. You, X. Yang, M. Pachulski, and J. J. Zhang. Boundary constrained swept surfaces for modelling and animation. In *Computer Graphics Forum*, volume 26, pages 313–322. Wiley Online Library, 2007. 1, 3

[18] L. You, X. Yang, X. Y. You, X. Jin, and J. J. Zhang. Shape manipulation using physically based wire deformations. *Computer Animation and Virtual Worlds*, 21(3-4):297–309, 2010. 3

[19] X. Zhang, Y. Wang, M. Gugala, and J.-D. Müller. Geometric continuity constraints for adjacent nurbs patches in shape optimisation. *ECCOMAS-2016*, 2016. 11