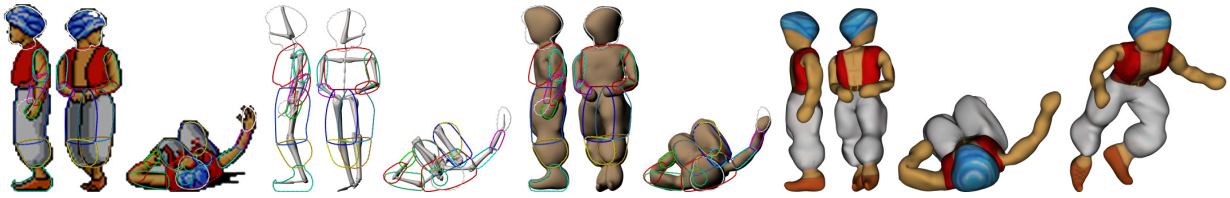
EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)

Volume 32 (2013), Number 2

ArtiSketch: A System for Articulated Sketch Modeling

Zohar Levi Craig Gotsman

Technion - Israel Institute of Technology



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

Figure 1: Modeling the Prince shape with ArtiSketch: (a) The contours in three 2D sketches are traced. Correspondence between contours in different frames are color coded. (b) The user poses the 3D skeleton to approximately fit each sketch. (c) ArtiSketch generates a 3D reconstruction of the articulated shape, which is posed to fit the sketches. (d) The 3D model painted. (e) A novel pose of the articulated shape (obtained by manipulating the skeleton).

Abstract

We present ArtiSketch - a system which allows the conversion of a wealth of existing 2D content into 3D content by users who do not necessarily possess artistic skills. Using ArtiSketch, a novice user may describe a 3D model as a set of articulated 2D sketches of a shape from different viewpoints. ArtiSketch then automatically converts the sketches to an articulated 3D object. Using common interactive tools, the user provides an initial estimate of the 3D skeleton pose for each frame, which ArtiSketch refines to be consistent between frames. This skeleton may then be manipulated independently to generate novel poses of the 3D model.

1. Introduction

Creating compelling 3D content is one of the biggest challenges in 3D graphics. Software tools for acquiring or modeling static geometry are prevalent, but require significant experience and expertise of the modeler, making the process time-consuming and costly. Thus the 3D industry is constantly on the lookout for ways to create more 3D content in less time using more intuitive interfaces.

Drawing on the success of traditional 2D sketch modeling, sketch-based 3D modeling is emerging as a possible alternative to the common freeform surface manipulation or other analytic modeling techniques. In a typical setup, such as in FiberMesh [NISA07], a user draws and manipulates 3D curves which are intelligently interpolated to form a surface. While reasonably effective, this type of interaction requires the modeler to sketch and operate in 3D, which is difficult in itself. The alternative is systems such as the popular Google SketchUp software and those proposed by Rivers et al. [RDI10] and Kraevoy et al. [KSvdP09] that enable artists, trained in traditional 2D sketching, to be able to continue to use a 2D paradigm, but harness it to the creation of static 3D models. In these systems the artist "models" by sketching the object silhouette from multiple views interactively or in advance, usually from at least two orthogonal orthographic views. Then a process of multi-view surface reconstruction is applied.

We propose ArtiSketch – a system that takes this methodology one step further, allowing novice users to take advantage of the abundance of existing 2D content, such as cartoon animations and sprites (see Fig. 2) to construct a 3D model. However, since this content usually depicts animated shapes, which means that inter-frame motion is present in the content and has to be accounted for, this is an obstacle for current systems that require that the object be rigid between views. We address this issue by assuming that the animation depicted by the content is articulated, i.e. piecewise-rigid. We also assume that the animation "imitate real-life", a concept coined by Disney [TJ87], which translates into object consistency between frames, i.e. the images obey physical camera rules. Alas, given a few frames (~3) of content that

follow these assumptions, in the form of a few simple contours, there is still information missing, namely the camera location and orientation in each of the frames. Without this crucial information, contours such as those in Fig. 1(a) are ambiguous, which makes the surface reconstruction problem intractable. We emphasize that in the systems mentioned previously, the camera positions are predetermined and the user is limited to sketching in these views only. Thus we are interested in the following question: Can the user somehow supply, through a convenient interface, an approximate initial camera transformation that is sufficient for the computer to fully synchronize the cameras and reconstruct the 3D object? To answer this question we start by proposing a novel way for the user to supply this information. We add to our setup a simple 3D skeleton that controls the dynamic parts of the 3D object sketched in the input frames. A skeleton is commonly used for manipulating, posing, and animating a 3D object. In our setup we require that the user manipulates the skeleton to achieve the inverse: Given a 2D sketch and a skeleton, pose the skeleton to imitate the pose of the 3D object depicted in the sketch. Adding this requirement, we encounter and solve a new problem, which we believe is interesting in its own right: Given a number of articulated sketches with initial skeleton poses, reconstruct a plausible 3D object whose silhouettes at those poses coincide with the sketches. In the rest of the paper we propose a solution to this problem, resulting in ArtiSketch, a system for articulated sketch modeling. ArtiSketch is not limited only to inputs which seemingly depict orthographic views, but can also handle perspective views, allowing for scaling of the shape between frames. Since we cannot expect from the user a precise placement of the skeleton, ArtiSketch uses a novel algorithm that synchronizes the camera views based on the user initial poses.

1.1. Contribution

We introduce a reconstruction problem which has not been treated before, and propose a solution in the form of our ArtiSketch system. This involves the following:

- An algorithm for solving for the virtual camera positions associated with each sketch in a given set, where an initial estimate for each is given through a skeleton pose.
- An algorithm for reconstruction of a single 3D triangle mesh consistent with multi-frame articulated silhouette data.

1.2. Related Work

In this review of prior art we will not mention sketch-based methods that mainly propose new user interface tools to manipulate a 3D object, e.g. Teddy and its evolutions such as FiberMesh [NISA07] or the rigged version [BJD*12], but focus instead on more relevant methods that reconstruct a surface that fits a set of input sketches or silhouettes. A more comprehensive review of sketch-based methods can be found in [OSSJ09]. Kraevoy et al. [KSvdP09] described a system that uses contour sketches to design a new 3D surface. Their system deforms a template mesh such that its outline conforms with the sketches. They primarily use one orthographic view, and the heart of the algorithm is finding a correspondence between the contour vertices and the mesh silhouettes. This is done by dynamic programming based on proximity and normal difference. The template is then deformed to fit the sketches. One limitation of this method is that the template should be viewed through standard orthographic views (front, side, and top), otherwise the deformation will warp it unnaturally. Unlike our method, Kraevoy et al. [KSvdP09] do not require that the user segment the sketches and make a careful assignment of the contours to the model parts. Thus, the only way to prevent the corresponding curves from "sliding" to undesired areas of the template, is to make sure that the template is very close to the contours. Rivers et al. [RDI10] take a different approach, modeling each part separately without the need for a template. They let the user sketch each part of the model from different orthogonal orthographic views (front, side, top). The parts are then smoothed and combined using CSG operations, using a method to perform the 3D CSG operations as 2D CSG operations on the projection plane.

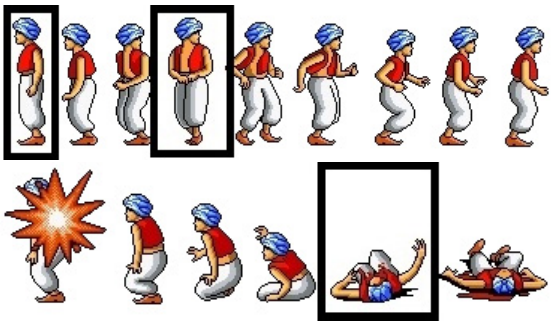


Figure 2: Prince sprites. Marked in black are the three frames from a 2D sequence that were used in the reconstruction of the prince in Fig. 1.

[KH06] is a system for inferring plausible 3D free-form shapes from visible-contour sketches. [SKSK09] describes an approach that consolidates and disambiguates sketched 2D curves into a 3D interpretation. More recently Xu et al. [XZZ*11] introduced an algorithm for 3D object modeling where the user is inspired by an object captured in a single photograph. The method leverages the wide availability of photographs for creative 3D modeling. Since using only one photo as a guide is not sufficient, they support the modeling process by limiting the model to a small set of 3D candidate models. Cashman et al. [CF12] showed that there is enough information in a collection of 2D images of certain object classes to generate a full 3D morphable model, even in the absence of surface texture. The model representation is a linear combination of subdivision surfaces, which they

fit to image silhouettes and any identifiable key points, using a combined continuous-discrete optimization strategy. Vlasic et al. [VBMP08] demonstrated a practical software system for capturing mesh animation details from multi-view video recordings. The system starts with a stream of silhouette videos and a rigged template mesh. At every frame, it fits a skeleton to the visual hull, deforms the template using Linear Blend Skinning (LBS), and adjusts the deformed template to fit the silhouettes.

1.3. System Outline

Before we provide all the details of ArtiSketch, we give here a succinct outline of the system functionality; see Fig. 1.

Input:

- A set of F sketches, which we will also refer to as frames. Each sketch contains a set of simple closed contours. Each contour is assumed to be the outline (silhouette) of a rigid *part* of an articulated 3D object in some pose, as viewed from a virtual viewpoint. Some of the frames may be missing some of the contours. For simplicity, we assume simple closed contours, but the system also allows for more general contours. These can be traced manually over an image, or extracted using appropriate methods.
- A skeleton in F (initial) poses - one for each sketch. Constructing the skeleton is a simple matter of pointing and clicking at joints positions to build a tree of bones. It is also possible to simply adjust existing standard skeletons of common articulated objects. The user poses the skeleton manually by transforming the skeleton joints.
- A correspondence between the set of contours and the skeleton joints.

Output:

- A triangle mesh surface bound to the skeleton. The mesh is such that the silhouettes of its LBS, as defined by the skeletal pose associated with each sketch, fits the contours of that sketch well.

ArtiSketch proceeds in the following steps:

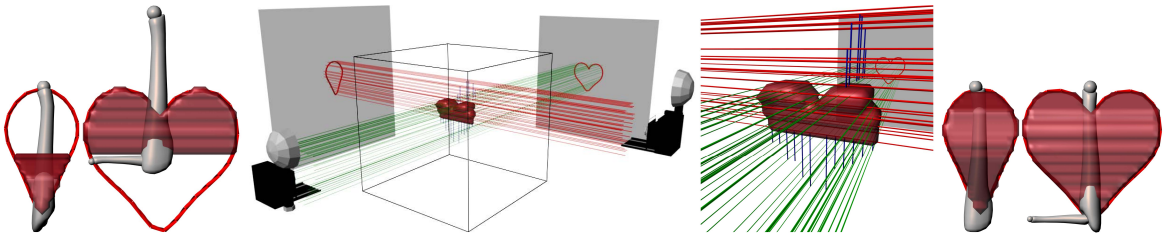
- Camera calibration: The system optimizes the skeleton pose in each frame to synchronize the camera views of each rigid part.
- Surface reconstruction: A variant of the Level Set Method (LSM) is used to reconstruct each rigid part of the shape.
- Volume reduction: Rim paths matching the sketches are frozen for each rigid part, and the rest of the surface is smoothed using a bi-Laplacian.
- Parts consolidation: The individual meshes are fused into a single final triangle mesh.

ArtiSketch is a complete system. As such, all of the steps mentioned above are necessary for the system operation. Some steps, such as the parts consolidation step, are heuristic in nature. Other steps rely heavily on previous work, such as LSM in the surface reconstruction step and dynamic programming in the volume reduction step. Thus we refer the reader to the prior work on the appropriate subjects, and focus our discussion only on the modifications that we have made. These heuristics and modifications, although essential to our system, are not notable on their own, and our main contribution is their integration to a working system, as stated in Section 1.1.

2. Camera Calibration

A common way to reconstruct 3D shapes from multiple silhouettes is using the so-called *visual hull*, which is the largest volume consistent with the given silhouettes. However, good visual hull reconstructions require silhouettes from many angles (at least 8), that the shape be static, and that the camera parameters are known for each silhouette. ArtiSketch is able to produce good results based on only three sketches on the average, due to our additional assumption of smoothness and minimalism of the surface. Given the sketches, ArtiSketch also needs to determine the camera parameters. Moreover since we allow articulated motion, we need to determine the parameters of a separate camera per rigid part for each frame. Equivalently, ArtiSketch uses only one static "real" camera for all the frames and for all the parts, and moves instead the parts rigidly in front of the camera using a skeleton. However, it is more natural to talk about virtual cameras, one per rigid part for each frame. The virtual camera for a part in an input sketch sees a single contour of a single rigid part, which is assigned (by the user) to a skeleton joint. The virtual camera position is set to the transformed position of the real camera. The transformation which is applied to the real camera is the inverse of the joint transformation from the first frame to the current frame. The joint transformation is calculated from the poses of the skeleton hierarchy. Setting the skeleton for each frame is a simple matter of translating the skeleton's root joint, and then rotating the individual joints. The user sets the initial pose of the skeleton for each frame.

We describe a calibration algorithm which, given the initial poses and optional constraints on the joints rotation and root translation, calibrates the skeleton (i.e. the virtual cameras) to maximize the consistency between the silhouette of the shape derived from the sketch and the contour present in the sketch. This procedure is related to existing algorithms for calibrating a camera based on silhouette data alone. These are either RANSAC-based for finding a correspondence of frontier points and epipolar tangents [FSPK04, SPM04, WC04], or based on minimizing a visual hull-dependent energy – an energy that cannot be differentiated and thus cannot be efficiently optimized - using Powell's derivative-free algorithm [HSC07]. We should note that there are not many calibration algorithms that are based on silhouette data alone, and most algorithms rely on image texture. Our procedure is a fast ICP-based approach, where



(a) (b) (c)

Figure 3: Heart sketch calibration. (a) Two (red) 2D sketches of a heart shape provided by the user. The user has inaccurately positioned the (gray) skeleton, so that there appears to be a vertical translation between the two contours. As a result, the (red) visual hull is incomplete. (b) The scene before the camera calibration, with the virtual cameras, and the ray correspondences marked by blue vertical lines. (c) Correctly positioned skeletons and the resulting visual hull projected onto the two frames after camera calibration.

each global iteration step minimizes a differentiable energy. We cannot guarantee the stability or convergence of this process, and similarly to ICP, our method is sensitive to the initial "guess", which should be close enough to the required minimum. Nevertheless, where ArtiSketch is concerned we expect from the user reasonable initial poses. We conducted a small user study to test this expectation. The users were able to pose a skeleton successfully, such that ArtiSketch converged to a local minimum and achieved good view synchronization. On the average it took a novice user up to five minutes per frame to pose the skeleton, and a professional 3D animator up to two minute. Both test groups found the task easy.

2.1. Calibrating Two Cameras

Most popular methods for constructing a visual hull are voxel-based [EBBN05]. The idea is to discretize the region of space which the object lies in with a cubic voxel grid, and "carve" away voxels that are not contained in the intersection of the generalized "cones" defined by each of the silhouette contours. Thus each surviving voxel defines a correspondence between rays of the different views intersecting that voxel, and, by definition, there is one ray per view for each voxel.

As opposed to the conventional "volume carving", ArtiSketch must also determine the relative position of the virtual cameras associated with each contour in each sketch. We start by considering the case of a single rigid object (one contour in each frame) and two sketches, which result in two virtual cameras. One camera, A, is fixed and the other camera, B, is to be calibrated relative to A. Call an A-ray (B-ray, resp.) a ray emanating from A (B, resp.) through its contour in the image plane. As mentioned before, if the two cameras are calibrated (i.e. positioned correctly in space relative to each other), an A-ray should intersect (i.e. correspond to) a B-ray at some voxel. If not, some A-rays will not intersect any B-rays, and this part of the surface would be missing and not conform with the sketches. Thus the objective is to maximize the number of intersections between A-rays and B-rays, or more specifically, minimize the Hausdorff distance between the two sets of rays. When the cameras are perfectly calibrated, and the silhouette contours are exact, each A-ray should intersect a B-ray and vice-versa.

We propose a new calibration algorithm that consists of two steps, which are alternated similarly to the well-known Iterated Closest Point (ICP) algorithm [BM92]:

- Find correspondences between A-rays and B-rays, where the rays are discretized by the voxels.
- Find a camera transformation that optimizes an energy based on the correspondences.

2.2. Ray Correspondence

The first step is to find a correspondence between A-rays and B-rays. A finite number of A-rays and B-rays are generated by projecting the center of each voxel of the discretized space onto a discrete 2D pixel grid of the sketch, and determining if it falls on the contour. Our experiments showed that it is better to exclude rays that fall inside the shape (as well as outside); namely, we match only rays that contribute to the surface of the object. We say that a voxel is *hit* by camera A if an A-ray intersects the voxel, i.e. projects its center onto a contour pixel in the sketch. An A-ray *corresponds* to a B-ray if they intersect voxels in close proximity (ideally the same voxel). This correspondence is represented by these two voxels, defining the distance between the two rays (ideally zero). In practice, each voxel contains a vector of flags, one flag per camera, where each flag indicates whether the voxel is hit by the respective camera.

We compute correspondences between A-rays and B-rays by traversing the voxels hit by A, and searching a spatial data structure (k-d tree) for the closest voxel hit by B. When found, the centers of these two voxels are projected with the respective cameras, and the two pixels which the respective projections fall into define a correspondence between the A-ray and the B-ray. This correspondence, i.e. the identity of the two voxels which led to the correspondence, is stored in

the 2D pixel array describing the A-rays, only if the cost - calculated with our distance-based cost function - of the two voxels is lower than the cost between the previous voxels that were stored there (the array is initialized to no voxels with infinite cost). The same is done for the voxels hit by B, resulting in an A-ray corresponding to each of the B-rays. See Fig. 3 for an illustration of this method. In the scenario depicted there, the user has mistakenly positioned the skeleton in two contours with a vertical offset, and the camera calibration procedure detects and corrects this.

We emphasize again that a ray is discretized by all the voxels that it hits, and they are all projected onto the same pixel in the 2D array that holds the best correspondence. This pixel is in fact the ray representative. For example, in our setup the distance between two rays is the distance between the pair of voxels that are closest to each other, one from each set of voxels discretizing each ray. Thus, in the end, we obtain correspondences between pairs of rays, represented by single voxels. Note that this is not the same as what would have been obtained were we simply trying to match pairs of voxels in the first place without using the 2D pixel arrays that represent the rays.

In the general case where we need to calibrate $n_{cam} > 2$ cameras, the cost function gives more weight to voxels having more hits (from different cameras). The objective is to prefer correspondence from the center of the rays intersection:

$$cost(v_A, v_B) = dist(v_A, v_B) + \lambda(n_{cam} - n_{hits}(v_B))$$

where v_A and v_B are voxels of A and B respectively, n_{cam} is the number of sketches, $n_{hits}(v)$ is the number of cameras hitting v, and the scalar λ is a weighting factor. Note that this cost function is not symmetric.

2.3. Optimal Camera Transformation

First we consider the simple case of one rigid part, two cameras, and no constraints. Given the list of voxel correspondences from the previous section, we convert it to a list of 3D point correspondences, where a voxel is represented by its center. The optimal rigid camera transformation then reduces to a simple Procrustes problem: Given two corresponding sets of N points $P = \{p_i\}$, the centers of the voxels hit by B, and $Q = \{q_i\}$, the centers of the corresponding voxels hit by A, we seek a rigid transformation $T \in SE(3)$ that consists of a rotation R and a translation t:

$$T(x) = Rx + t, \;\; R \in SO(3), \;\; t \in \mathbb{R}^3$$

that minimizes

$$E_{one\_joint} = \sum_{i=1}^{N} ||T(p_i) - q_i||^2 \;\; .$$

We now consider the case of more than one rigid part (but still two cameras). In our setup the camera is static, and we have a skeleton with degrees of freedom corresponding to root translation and joint rotations. We have corresponding point sets P_j and Q_j associated with the j'th joint, each having N_j points, $\{p_{ji}\}$ and $\{q_{ji}\}$, and we need to compute the best transformation T_j between them. Therefore if we have J joints, we have to solve J coupled Procrustes problems, minimizing the energy:

$$E_{skeleton} = \sum_{j=1}^{J} \sum_{i=1}^{N_j} ||T_j(p_{ji}) - q_{ji}||^2$$

T_j is a transformation for a joint from one pose to another, which is the joint transformation in the second pose multiplied by the joint inverse transformation in the first pose. Each transformation of a joint in a pose is evaluated by multiplying the local rotations of joints that precedes it in the hierarchy, and adding root translation. Thus the variables are the local rotations (parametrized by Euler angles) and root translation of the second pose. Except for the root transform, the translation in the other joints is zero since the bones length is constant. The coupling between the Procrustes problems results from higher-level joints in the skeleton inheriting the sequence of transformations of the lower-level joints, and adding just a rotation to them. The transformation of the corresponding virtual camera of a joint is then T_j^{-1}.

This energy is for two sketches (frames), and we now describe how to work with multiple frames. We start with the first frame, whose skeleton pose (virtual cameras) is fixed throughout the process. We incrementally add sketches, one by one. Each new sketch is optimized independently against the rest of the sketches, namely, only the joints in this sketch move relative to the previous sketches. A list of correspondences is made between the current sketch and all other sketches in both directions (for each pair of sketches, A and B, we find correspondence from A to B and from B to A), and transformations describing the movement of the new points relative to the union of points from all previous poses is computed.

If we allow arbitrary translation and rotation between cameras, we may obtain undesirable transformations, such as the trivial identity transformation. To make the process more robust, we constrain the possible rotations and the root translation from the "intersection centroid" - defined as the centroid of the voxels hit by all cameras. We optimize the energy using the Augmented Lagrangian method [NW06]. This is an off-the-shelf optimizer, which can solve a constrained problem, and requires the derivatives of the objective function and the constraints.

2.4. Perspective Views and Camera Dolly

Our system is not limited to orthographic views, It can handle perspective views as well, which allows to account for

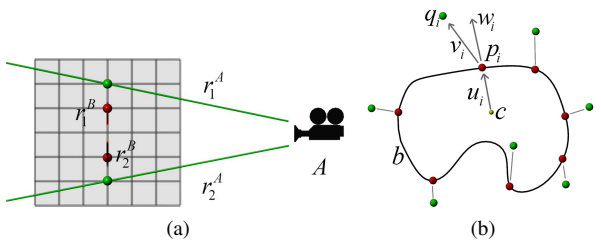


Figure 4: Camera dolly. (a) Camera B view. The two red points are voxel centers that were selected to represent the B-rays, and the two green points are the centers of the voxel representatives of the A-rays. The angle between each pair of camera rays in the image is the same. B needs to dolly backward for the B-rays to intersect the A-rays. (b) In this scene, all the forces except one attract the points outward, thus B should dolly backward.

variation in the object size between frames using a camera dolly parameter. Since our scheme uses closest voxels to represent corresponding rays, in most cases the corresponding voxels will have the same depth coordinate w.r.t. the camera, and thus cannot express the dolly movement when transforming one voxel to another. We illustrate this point by the following example, which is illustrated in Fig. 4(a). Consider a scene viewed by two orthogonal perspective cameras A and B. Both cameras see the same sketch: two points sitting on a vertical line, except that B was initially positioned closer to the grid than A. Since the cameras are perspective and B is closer, the two B-rays that hit the two points on the image plane are enclosed by the two A-rays. The correspondence between the rays would be: (r_1^A, r_1^B) and (r_2^A, r_2^B) at voxels which reside on a vertical line (in reality the closest point would not reside on a vertical line, but due to voxel discretization it will). Since the two correspondence pairs would apply the same force (moving points as a result of energy minimization, can be viewed as applying force to the points), the camera optimization, as defined previously, would leave B in place and would not be able to detect a dolly backward. To rectify this, we use a specialized univariate optimization to compute a separate dolly parameter.

The optimization proceeds as follows: We find a correspondence between P and Q as described above. P are voxels hit by B, the camera that we want to dolly, and Q are the voxels hit by A that correspond to those in P. We project P and Q onto B's image plane. Denote by b the 2D contour of B's sketch that the set P projects onto, and denote b's centroid by c. We compare the forces that attract P to the inside of b, towards c, to the forces that attract P to the outside. Define the vectors $v_i = q_i - p_i$, $u_i = p_i - c$. Project v_i on u_i:

$$w_i = \frac{< u_i, v_i > u_i}{||u_i||^2}, \;\; sign(w_i) = sign(< u_i, v_i >) \; .$$

If $\sum sign(w_i)||w_i|| > 0$, we dolly the camera a step backward, otherwise we dolly it a step forward; see Fig 4(b). We keep the new camera position only if it reduces the total energy $E_{skeleton}$. We described here a basic line search of a constant step size in the view direction. We perform the dolly computation step after every few iterations of the ICP-based algorithm.

3. Surface Reconstruction

Now that we know the parameters of each virtual camera, we proceed to the problem of rigid object reconstruction from the 2D contours as seen by these cameras, known as the shape-from-silhouette problem. We have already mentioned the visual hull reconstruction method, but the results it generates are too coarse, and many views are needed to obtain good results. The coarseness stems from the simplicity of the algorithm, and its inability to incorporate other energies such as one that regulates the surface smoothness. Additionally, the visual hull carving method is sensitive to inaccuracies in the silhouettes of the object. It would blindly ignore parts of the object where not all the camera rays intersect. We would prefer a more sophisticated method that is capable of approximating the object when not all the sketches are consistent. Common stereo vision methods may be used to solve this problem, but we require a method that does not depend on photo-consistent texture for finding the correspondence, and can cope well with silhouette data alone.

We used a variant of the Level Set Method (LSM), which takes the Eulerian approach, and progresses a front implicitly in a higher dimensional grid. An alternative solution to this problem, which takes the Lagrangian approach, is the snake [XP98], which directly manipulates the surface vertices. We chose to use the LSM, since it overcomes some serious limitations of the snake, such as the need to remesh the evolving surface in order to preserve density, the inability to cope with topological changes, and the inability to prevent self-intersections. Also, calculating differential properties on a snake is less accurate and leads to instabilities in its evolution. We use a LSM that combines variants of energies used in two popular methods. The first is the region-based Chan-Vese [CV01], and the second is the edge-based Geodesic Active Contour (GAC) [CKSS96] using a GVF [XP98] as an external force. The GAC by itself allows the user to draw unstructured sketches (containing open contours), while the Chan-Vese requires a simple closed contour that can be filled to create two separate colored regions. We found the Chan-Vese energy to be quite robust in the setup where the sketches are restricted to simple closed contours, since it indirectly enforces the sketches bi-color photo consistency (background / foreground). In the Appendix we provide the energy formulation that we used for the LSM.

4. Volume Reduction

The surface obtained by the procedure of the previous section, although of a better quality, is still very similar to the naive visual hull, in the sense that it is still pursuing the maximal volume contained in the intersection of the rays. This is

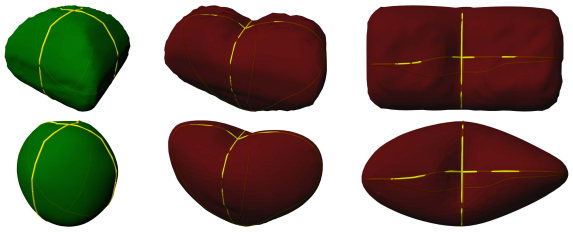


Figure 5: Volume reduction. (Left) On the top is the visual hull of two perpendicular orthographic cameras viewing circular contours. The result is an intersection of two cylinders. The bottom is the result after volume reduction - a sphere. (Middle) Visual hull vs. volume reduction result for the two heart contours of Fig. 3. (Right) Top view of the shapes in the middle.

usually not what the user is looking for; see Fig. 5. Thus we improve the result by finding the shape whose outline fits the sketches and whose surface minimizes our smoothness energy (implying a minimal surface). The general approach to this type of problem is to select rim paths (silhouette points) on the surface, freeze them, and let the rest of the surface evolve using mean curvature flow to reach a minimal surface area. Rivers et. al [RDI10] follows the naive approach of freezing some random silhouette points. But one can easily see that if these random points happen to fall outside the minimal surface, the resulting surface would not be minimal and quite deformed. In the heart example in Fig. 5, if we do not make a smart selection such as the two contour paths, the result would be a deformed heart.

To find the rim paths, we use a method based on dynamic programming, similar to [KSvdP09]. Our cost for a path is determined by the vertices it passes through. The cost of a vertex v on the path is

$$C_{silhouette} + C_{proximity} + C_{geodesic} + C_{barycenter} + C_{normal} \ ,$$

where we omitted the weighting constants for clarity.

The silhouette cost measures how far v is from being a silhouette point: $< n_v, d >^2$, where n_v is the normal at v, and $d = v - eye$, where eye is the camera position.

The proximity cost measures the distance between the vertex projection and a sketch point: $||\pi(v) - p||^2$, where $\pi(v)$ is the projection of v, and p is the closest sketch point in image space.

The geodesic cost penalizes long edges on the path since we strive for a short path: $||v - v_{prev}||^2$, where v_{prev} is the vertex preceding v on the path.

The barycenter cost penalizes vertices that are far from the shape: $||v - c||^2$, where c is the shape barycenter. This keeps the paths close to each other with a common barycenter if possible. This should promote smaller volume.

The normal cost measures how well the projection of the normal at v matches the normal to the sketch contour: $-(1 + < \pi(n_v), n_p >)^2$, where n_p is the normal at p.

After finding the rim points we trim the rest of the surface. Rivers et al. [RDI10] use the Lagrangian approach, whose limitations were discussed in Section 3. We tried an approach that freezes the voxels where the rim points coincide, and continue the LSM process with curvature flow alone. But for thin surfaces, such as the prince's shoes (Fig. 1), the anchored rim points degenerated into disconnected "islands". Another obstacle that both methods face is numerical instability, which becomes more acute considering our next requirement: When smoothing a surface we would prefer an energy similar to the "roundness" preserving Willmore flow, which requires a fourth degree differential, and is known to be unstable even for LSM. Therefore instead of the LSM we adopted the approach of Botsch and Kobbelt [BK04], solving a bi-Laplacian equation to reconstruct the surface, using the rim points as anchors:

$$\Delta^2 x = 0 \ \ s.t. \ \ x_i = v_i \ \ , \ \ i \in I_{anchors} \qquad (1)$$

which proved to be more robust.

A few notes on the volume reduction step. In order for the rim paths detection to be robust, it is necessary for the initial surface that was produced in the previous section to have silhouettes that fit the sketches, and to be smooth (crucial for the normal cost). Thus the robust LSM is used, and alternatives such as a simple initialization of a template mesh (e.g. a sphere) or simply the visual hull, would not have been good enough. After the rim paths detection, the only hints that the bi-Laplacian needs are the surface topology and the rim points as anchors.

5. Parts Consolidation

If we did not allow the designer the freedom to specify standalone parts in the sketches, or more specifically, contours inside the area of the shape (as opposed to curves on the 2D shape outline only), we could have used one global LSM step to capture all the parts. But in order to give the user this extra freedom, we run a LSM step and a volume reduction step for each rigid part independently. This allows for costs in the volume reduction step, such as C_{normal}, to match between the 2D sketch and points inside the final shape volume - points which now sit on the surface of a rigid part that was constructed independently. As a consequence, the result of the previous steps is a set of disconnected meshes; see Fig. 6. The gaps between the meshes occur at the skeleton joints, where the transformation is not strictly rigid anymore. Thus the cameras may not agree at these places, and the surface is not reconstructed there. To "stitch" together these parts we use the following heuristic, which proved to be quite robust:

- Place a sphere at each joint. The sphere radius is taken to be 60% of the diameter of the surface that envelops the bone, represented as a line segment between a joint and its child joint. The diameter is computed as follows. Consider the set of vertices in a sphere of an arbitrary radius, which are close to a plane that intersects the center of the

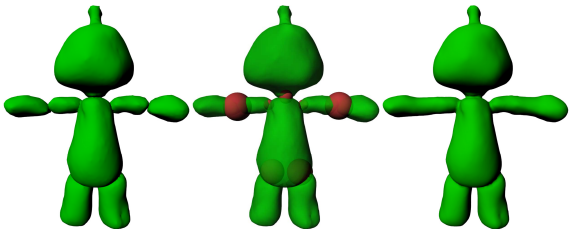


Figure 6: Parts consolidation of the Alien model. From left to right: Independent parts, adding spheres, and the consolidation result.

bone, and has a normal parallel to the bone segment. The diameter of this set is defined to be the diameter of the surface. In the rare cases that the user wants to improve the sphere parameters, our system allows the user to manually "tweak" the spheres.

- Perform boolean union of the spheres with the mesh parts.
- Set as variables all the vertices inside the volume of the previous spheres. Fix the rest of the surface, and solve a bi-Laplacian equation for these variables, as in Eq. 1.

To allow for an easier placement of the spheres, such that they would not overlap unnecessary regions, the user adds a zero frame, where the skeleton is in a da-Vinci pose (i.e. extended limbs) without any assigned contours, and on this pose we perform the consolidation step.

6. Experimental Results

Implementation For the LSM implementation we used the method of Iwashita et al. [IKTH04]. This method uses stencils to quickly estimate the tube around the propagating front. We bound the final mesh to the skeleton for LBS posing, using *Pinocchio* [BP07] to define the skin weights.

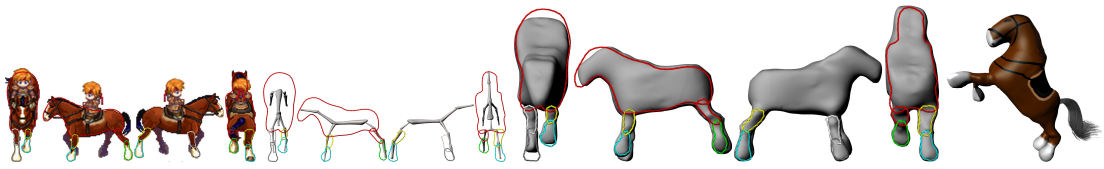
Performance We ran our experiments with ArtiSketch on a laptop computer, having an Intel i7 2.2GHz processor and 8GB RAM. The calibration step between three sketches takes up to ten seconds. In the reconstruction step, for most models we used a grid size of 100^3 for the LSM. It takes twenty seconds to propagate a level set to detect a part from three sketches. The algorithm we used can be parallelized over the GPU, and adding an initialization step of the level set to the visual hull, which requires a smaller grid and less steps to converge, can easily reduce the run-time to less than five seconds per rigid part. For the consolidation step, the rim paths detection takes up to ten seconds, and the smoothing step that involves solving a bi-Laplacian equation, featuring in both the volume reduction and the consolidation step, takes up to ten seconds as well.

Experiments Input frames from an animation should be chosen such that the view directions would be different as possible, optimally orthogonal; see Fig. 2. Similar to visual hull algorithms - the more exact input frames the system has - the more detailed the output will be. But if the sketches are inconsistent, as in our scenario, adding frames can make the result worse. Our experiments showed that approximately three frames sufficed. Thin objects, or very small parts such as hands, usually can be reconstructed using only two frames. Some of ArtiSketch's results can be seen in Figs. 1, 7-11. The (a) sub-figure shows the contours traced over the original 2D images, as provided by the user. The contours are color coded, such that contours in different frames corresponding to the same joint (same rigid part) share the same color. Note that not all the joints in a skeleton need to be assigned to contours in all the frames, e.g. Fig. 7, and some joints are not assigned at all, and are used only to forward a transformation to their children, or as orientation indicators. The (b) sub-figure shows the skeleton pose for each frame that was passed to the surface reconstruction step. The (c) sub-figure shows the final reconstructed mesh with the given contours superimposed. The (d) sub-figure shows the final model with texture and small accessories, either in one of the input poses that were used for the reconstruction or in a novel pose, which was produced manually by transforming the skeleton joints. As expected, discrepancies between the contours result in a surface with an outline that does not match the contours perfectly, e.g. the red vest of the prince in Fig. 1. In the horse model in Fig. 7, we tested ArtiSketch on a relatively large and more complicated shape, the horse torso. As evident in some of the figures, the outline of the model misses parts of the contour, implying that the torso should have been partitioned to smaller and simpler parts, where calibration should prove easier. The accompanying video demonstrates the ArtiSketch process and a small animation of sample results.

7. Conclusion

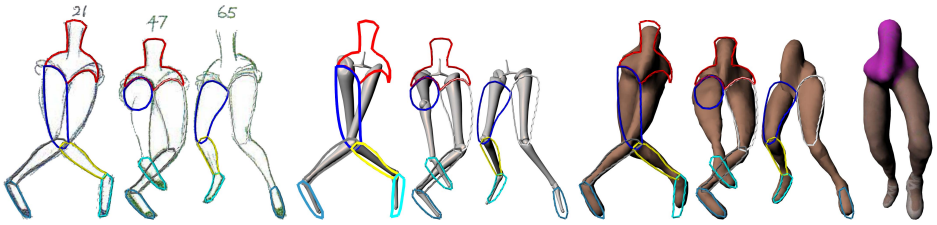
We have presented ArtiSketch, a system that reconstructs an articulated 3D object from a few sketches, with the help of a posed skeleton. ArtiSketch's main limitation is that the input sketches must depict piecewise-rigid, as opposed to soft-body, deformation. The sketches should also obey basic camera rules, and preserve the proportion of the parts, so that a skeleton can be fitted. This means that ArtiSketch will not be able to handle inputs with exaggerated deformations and unnatural movements between the sketches, such as cartoon characters that squash and stretch in nonphysical ways. Still ArtiSketch is an improvement over previous methods [KSvdP09, RDI10] that can handle only one rigid part from orthographic views.

As future research, we would like to incorporate symmetry information, which is present in many real-life shapes. It would also be interesting to test ArtiSketch on real (e.g. wildlife) movies, as opposed to synthetic cartoon inputs. For this purpose an energy term should be added that takes advantage of the photo-consistency between frames. Then Ar-



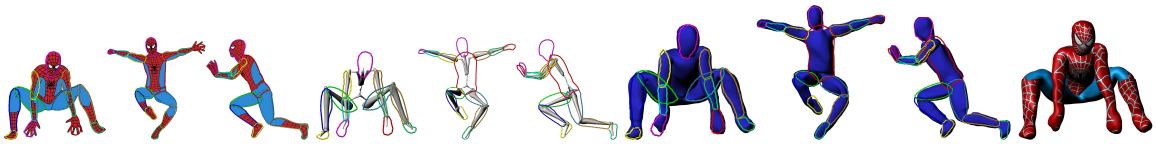
(a) (b) (c) (d)

Figure 7: Horse.



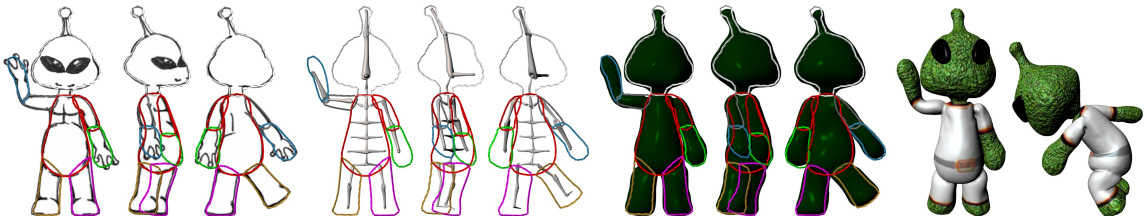
(a) (b) (c) (d)

Figure 8: Dancer legs



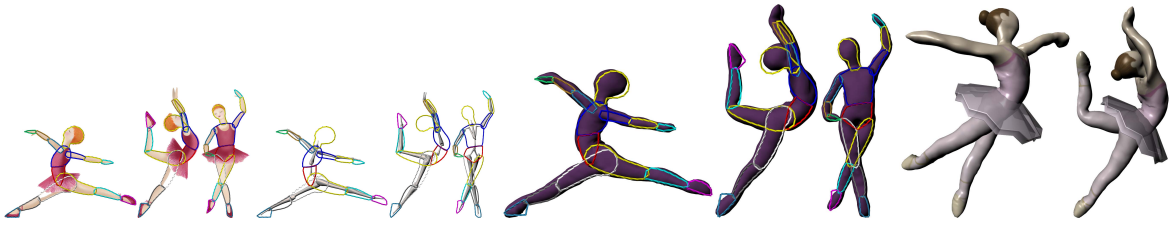
(a) (b) (c) (d)

Figure 9: Spiderman



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

Figure 10: Alien



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

Figure 11: Ballet dancer

tiSketch should texture the final mesh based on the input frames.

References

[BJD*12] BOROSÁN P., JIN M., DECARLO D., GINGOLD Y., NEALEN A.: Rigmesh: automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph.* (2012).

[BK04] BOTSCH M., KOBBELT L.: An intuitive framework for real-time freeform modeling. In *SIGGRAPH* (2004).

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3D shapes. *IEEE Trans. PAMI 14*, 2 (1992), 239–256.

[BP07] BARAN I., POPOVIC J.: Automatic rigging and animation of 3D characters. *ACM Trans. on Graph. 26*, 3 (2007), 72.

[CF12] CASHMAN T. J., FITZGIBBON A. W.: What shape are dolphins? Building 3D morphable models from 2D images. *IEEE Trans. PAMI 99*, PrePrints (2012).

[CKSS96] CASELLES V., KIMMEL R., SAPIRO G., SBERT C.: 3D active contours. *Lecture Notes in Control and Information Sciences 219* (1996), 43–49.

[CV01] CHAN T. F., VESE L. A.: Active contours without edges. *IEEE Image Processing 10*, 2 (2001), 266–277.

[EBBN05] EROL A., BEBIS G., BOYLE R. D., NICOLESCU M.: Visual hull construction using adaptive sampling. In *Proc. IEEE Workshops on App. of Comp. Vision* (2005), pp. 234–241.

[FK99] FAUGERAS O., KERIVEN R.: Variational principles, surface evolution, PDE's, level set methods and the stereo problem. *IEEE Trans. on Image Processing 7* (1999), 336–344.

[FSPK04] FURUKAWA Y., SETHI A., PONCE J., KRIEGMAN D.: Structure and motion from images of smooth textureless objects. In *ECCV* (2004), pp. 287–298.

[HSC07] HERNANDEZ C., SCHMITT F., CIPOLLA R.: Silhouette coherence for camera calibration under circular motion. *IEEE Trans. PAMI 29* (2007), 343–349.

[IKTH04] IWASHITA Y., KURAZUME R., TSUJI T., HASEGAWA T.: Fast implementation of level set method and its realtime applications. In *Sys., Man and Cyber.* (2004).

[Ker02] KERIVEN R.: *A variational framework for shape from contours*. Tech. rep., Ecole Nationale des Ponts et Chaussees, CERMICS, France, 2002.

[KH06] KARPENKO O. A., HUGHES J. F.: SmoothSketch: 3D free-form shapes from complex sketches. *ACM Transactions on Graphics 25*, 3 (2006), 589–598.

[KSvdP09] KRAEVOY V., SHEFFER A., VAN DE PANNE M.: Modeling from contour drawings. In *Proc. Eurographics Sym. on Sketch-Based Inter. and Modeling* (2009), pp. 37–44.

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3D curves. *ACM Trans. Graph. 26*, 3 (2007), 41.

[NW06] NOCEDAL J., WRIGHT S.: *Numerical optimization*. Springer, 2006.

[OSSJ09] OLSEN L., SAMAVATI F. F., SOUSA M. C., JORGE J. A.: Sketch-based modeling: A survey. *Comp. Graph.* (2009).

[RDI10] RIVERS A., DURAND F., IGARASHI T.: 3D modeling with silhouettes. In *SIGGRAPH* (2010), pp. 109:1–109:8.

[SKSK09] SCHMIDT R., KHAN A., SINGH K., KURTENBACH G.: Analytic drawing of 3D scaffolds. In *SIGGRAPH Asia* (2009), pp. 149:1–149:10.

[SPM04] SINHA S. N., POLLEFEYS M., MCMILLAN L.: Camera network calibration from dynamic silhouettes. *IEEE Comp. Vis. and Pattern Recognition 1* (2004), 195–202.

[TJ87] THOMAS F., JOHNSTON O.: *Disney Animation: The Illusion of Life*. Abbeville Press, 1987.

[VBMP08] VLASIC D., BARAN I., MATUSIK W., POPOVIĆ J.: Articulated mesh animation from multi-view silhouettes. In *SIGGRAPH* (2008), pp. 97:1–97:9.

[WC04] WONG K.-Y. K., CIPOLLA R.: Reconstruction of sculpture from its profiles with unknown camera positions. *IEEE Trans. on Image Processing 13* (2004), 381–389.

[XP98] XU C., PRINCE J. L.: Snakes, shapes, and gradient vector flow. *IEEE Trans. on Image Processing 7*, 3 (1998), 359–369.

[XZZ*11] XU K., ZHENG H., ZHANG H., COHEN-OR D., LIU L., XIONG Y.: Photo-inspired model-driven 3d object modeling. In *SIGGRAPH* (2011), pp. 80:1–80:10.

8. Appendix: LSM Formulation

The energy of a geodesic active surface [CKSS96] for capturing the edges of an intensity field I defined on a 3D surface S is

$$E(S) = \int\int_S g(|\nabla I|)da, \;\; g(t) = \frac{1}{1+t^p}$$

where p is a constant scalar. To minimize this energy, its Euler-Lagrange equation, which results in a surface evolution, is $S_t = g(H)n - \langle \nabla g, n \rangle n$, where n is the surface normal, and H is twice the mean curvature of S. The resulting LSM formulation is

$$\phi_t = div(g\frac{\nabla\phi}{|\nabla\phi|})|\nabla\phi|$$

where S is the zero level set of ϕ. We will now formulate the case of evolving a surface using its apparent contour as an external force inside a 2D image I [Ker02, FK99]:

$$E_{edge}(S) = \int\int_S h(S)g(|\nabla I(\pi(S))|)da$$

π is the camera projection, and h is a smooth indicator function that tests if a point is a silhouette. In a similar way we formulate an evolving surface using its apparent contour and the region-based Chan-Vese energy:

$$\begin{aligned} E_{region}(S) \;\; = \;\; & \int\int_S [\hat{h}(\pi(S))(I(\pi(S)) - c_1)^2 \\ & + (1 - \hat{h}(\pi(S)))(I(\pi(S)) - c_2)^2]da \end{aligned}$$

for colors c_1 inside the contour and c_2 for the background; $\hat{h}$ is a smooth function indicating whether the point is inside the region enclosed by the contour. The LSM formulation is

$$\phi_t = I(\pi(S)) - c_2)^2 - I(\pi(S)) - c_1)^2 \;\; .$$

The edge-based and the region-based formulations are combined using a weighting constant.