ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2018
T. Beeler and N. Thuerey
(Guest Editors)

Volume 37 (2018), Number 8

Computational design of transformables

Ye Yuan[1] Changxi Zheng[2] Stelian Coros[3]

[1]Carnegie Mellon University, USA
[2]Columbia University, USA
[3]ETH Zürich, Switzerland

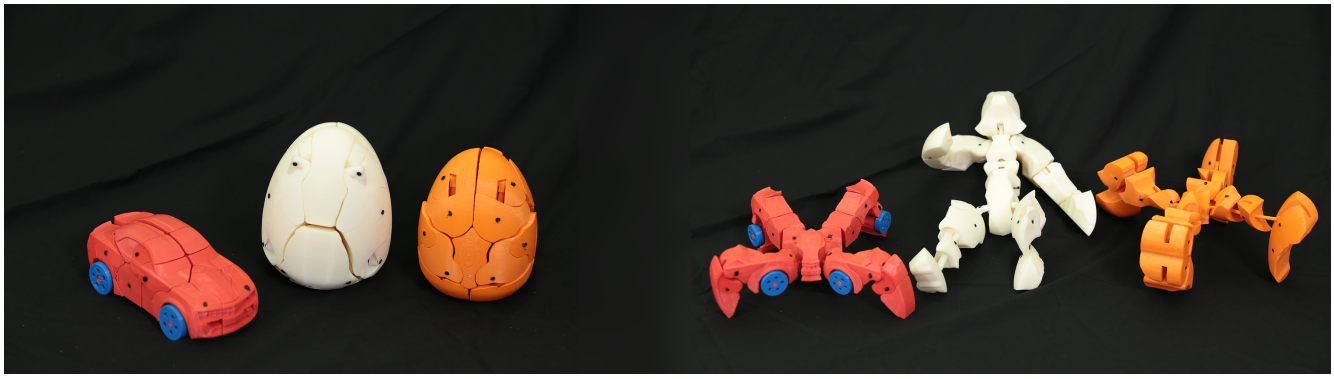

Figure 1: *We present a computational approach to designing transformables – physical characters that can take on vastly different forms with distinct functional capabilities. In* object mode*, these transformables are disguised as common items, such as a car or an egg. In* character mode*, they take on the form of a spider or a humanoid.*

Abstract

We present a computational approach to designing transformables, physical characters that can shape-shift to take on vastly different forms. The design process begins with a morphological description of an input character and a target object that it should transform into. Guided by a set of objectives that model the core attributes of desirable transformable designs, optimized embeddings are interactively generated. Intuitively, embeddings represent tightly folded character configurations that fit within the target object. From any feasible embedding, skin meshes are then generated for each body part of the character. The process for generating these 3D models is based on a segmentation of the target object, which is achieved through a growth-based model applied to a multiple level set representation of the transformable. A set of transformation-aware post-processing algorithms ensure the feasibility of the final designs. Building on this technical core, our computational design system provides many opportunities for users to inject their intuition and personal preferences into the process of creating transformables, while shielding them from tasks that are challenging and tedious. As a result, they can intuitively explore the vast space of design possibilities. We demonstrated the effectiveness of our computational approach by creating a variety of transformable designs, three of which we fabricate.

CCS Concepts

•***Computing methodologies*** → *Animation;*

1. Introduction

Thanks to numerous depictions in comic books, cartoons and movies, robots that change their shapes to perform different tasks have captured our imagination. Remarkably, these robots are not confined to digital media. The iconic *Transformer* action figure toys continue to entertain children and adults alike, and have even inspired the development of fully operational robots [Rus14]. Going beyond entertainment, multi-functional transformable devices hold considerable promise: with the ability to shape-shift, robots could be given the different locomotion modes required to traverse highly varied environments; fabricated in a tightly packed configuration, transformables can unfold to become functional when needed and shift back to their compact form to be stowed away or transported; disguised as everyday objects, household robots can pave the way to smart appliances whose mobile capabilities better accommodate the needs of their users.

As captivating as they are, transformable objects are very challenging to design. For shape-shifting action figures, for example, going from an initial vision to a working prototype is a laborious, time-consuming and expensive iterative process [Wag13]. In robotics, research into self-reconfigurable systems [YmSS*07] studies closely related challenges. To adapt their shape according to different tasks, reconfigurable robots rely on modular components that are simple and interchangeable. The discrete nature of modular robots, however, limits both their ability to seamlessly take on specific forms, and their overall motor capabilities. In contrast, the types of transformable characters considered in our work are custom-designed based on a description of their morphology and the target shape they should transform into.

Overview and contributions We present a novel computational approach to addressing challenges that are unique to the design of transformable physical characters. Starting from an input character skeleton and a target shape, our method can generate fabricable designs in a fully automated manner. To help account for hard to quantify considerations such as aesthetic or functional preferences, we also present a suite of interactive tools that allow non-expert users to guide the design process if desired. Our design system first creates an optimized, tightly-packed embedding of the character into the target shape. The objectives driving the optimization process are specifically formulated based on the requirements we define for desirable transformable designs. Based on the optimized folded pose, the target shape is decomposed into 3D models that are attached to the character's body and limbs. To create these 3D models, we propose automated and user-driven editing modes that operate on a highly-versatile multiple level set representation. A set of finishing algorithms that model the spatio-temporal transformation process are then applied. They post-process the generated geometry and create collision-free transition motions between the character and object modes.

We demonstrate the versatility of our computational approach by designing a diverse set of transformable characters. Under the user's guidance, each transformable takes just minutes to complete. As exemplified by our results, the space of possible designs is vast. Even for the same character morphology and target shape provided as input, drastically different transformables can be created. One of the key benefits of our work, therefore, is that it enables an efficient, user-guided exploration of various design choices. We validate the feasibility of the designs created with our system by fabricating several transformable prototypes.

2. Related Work

Fabrication-Aware Design Advances in digital manufacturing technologies are opening up a vast space of opportunities for physical artifacts with complex functional and aesthetic characteristics. Fueled by these advances, the graphics community has witnessed a surge in research efforts aimed at formalizing computational methods for fabrication-aware design of interesting classes of objects. Examples include design systems for objects whose mass distribution can be precisely controlled to enable them to stand, spin or float stably [PWLSH13, BWBSH14, MAB*15], paper airplanes and kites with optimized aerodynamics [UKSI14, MUB15], sound filters whose acoustic properties are intuitively prescribed [LLMZ16], and furniture pieces that are structurally stable [UIM12, KLY*14]. The unifying characteristic of many of these computational systems is that they aim to allow non-experts to easily explore the space of achievable results by automatically adjusting design parameters based on high-level notions of desired intent. In line with this philosophy, our work enables the design of transformables – physical characters that have the ability to shape-shift between vastly different forms.

Articulated Shape Design The design of articulated structures provides interesting opportunities and challenges that have attracted the attention of the research community. Calì et al. [CCA*12] proposed a method to generate fabricatable models that convert joint configurations of an animation rig to mechanically functional geometries. Concurrently, Bächer et al. [BBJP12] proposed a technique that uses skinning information associated with virtual characters to design the placement of mechanical joints and their properties. The design system described by Ureta et al. [UTZ16] further takes into account the range of motion when designing geometry for 3D printable mechanical joints. Several methods that design functional mechanisms and articulated linkage structures to create animated mechanical automata have also been proposed [CTN*13, CLM*13, TCG*14, BCT15]. Similar to these works, our goal is to automatically design functional geometry for mechanical joints and the rigid components that they connect. However, the artifacts that we create are specifically designed to enable a seamless transformation process between two very distinct shapes. One of these shapes aims to be nearly indistinguishable from a target object, while the other maintains the overall look of an articulated character.

Reconfigurables Objects with seamlessly reconfigurable structures are captivating to observe in action. Recognizing the technical challenges that arise in designing such objects, a number of research projects focused on reconfigurables have been proposed. For example, Li et al. [LHAZ15] proposed a computational approach to designing furniture items that fold compactly when not in use. The design system proposed by Zhou and colleagues [ZSMS14] generates mechanical structures that can fold a pre-specified shape into a box. The structure of the mechanism, which consists of unit cubes interconnected with joints, is designed solely to produce a successful folding process, and is therefore not likely to be otherwise functional. In contrast, our approach takes as input the morphological design of a character, whose range of motion is preserved. Furthermore, we pair the computational tools we develop with intuitive editing tools that allow users to inject their intuition and personal preferences into design process. Most closely related to our mission is the recent work of Huang and colleagues [HCLC16], who explored the problem of creating transformables for animation purposes. Our work shares their vision, but the efficient representations and mathematical models we employ, the user-controllable segmentation method we propose, and our fabrication-oriented methodology fundamentally differentiates our approach from theirs.

Our work also bears resemblance to previous methods that focus on the design of 3D puzzles. However, rather than creating disjoint, interlocking blocks [LFL09, XLF*11, SFCO12], our method generates articulated structures. The arbitrary length of the kinematic

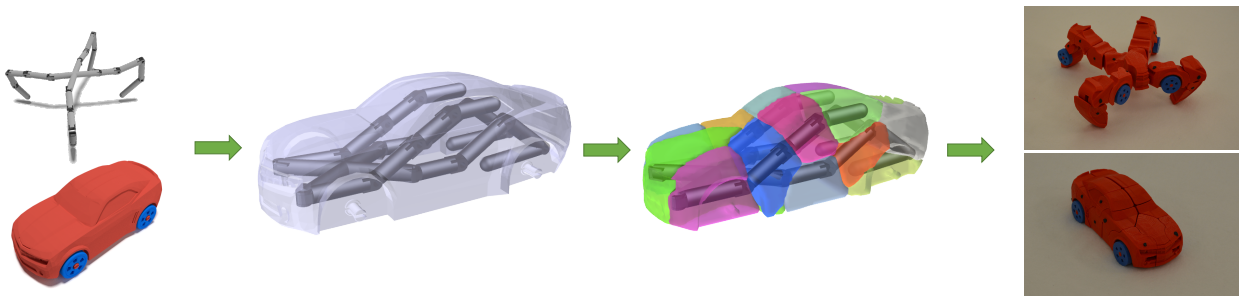

Figure 2: ***Overview of our design system*** *for transformable characters. (a) Our system takes as input the morphological descriptions (skeleton) of a character and an object mesh. (b) The embedding stage fits the skeleton tightly inside the object mesh with considerations of bulkiness and collisions. The combination of efficient embedding optimization and IK-based user interface allows users to explore the large design space. (c) The segmentation stage uses a collision-aware multiple level set-based growth model to partition the project. Intuitive editing tool is provided for users to direct the segmentation. (d) The finishing stage unions skeletal and skin meshes together to create a fabricatable design.*

chains in these articulated structures, the ability of our designs to transform between a functional character and an arbitrary object, as well as the conceptually different approach that we base our computational method on, are also in contrast to recent work on creating twisty puzzles [SZ15].

The computational system described by Garg et al. [GJG16] is also related to our work. This design system takes as input a choreography, which consists of a set of objects and their spacetime motion trajectories. The goal of our method is to generate precisely this type of choreography for transformables. Consequently, we adopt several techniques from their work to post-process our designs. Although not focused on fabrication, the recent work of Won and Lee [WL16], where multiple virtual characters choreograph their poses such as to cast a specific shadow pattern, inspired the approach we developed to optimize folded character configurations that best fit inside the target object.

Mesh Segmentation Image and geometry segmentation have been studied extensively by computer vision and graphics researchers; a recent survey on well-established techniques can be found in [Sha08]. Thanks to their ability to robustly handle topological changes and efficiently implement collision detection queries, level set-based approaches (e.g. [VC02, GF05]) are particularly well-suited for our work. The method we use to generate geometric models for the transformable's body parts is inspired by the recent work of Yao et al. [YCL*15]. While their application domain is very different – segmentation and packing to handle constraints due to limited print volumes – we share concepts required to partition a volumetric object using an evolving multiple level set model. However, we introduce an additional technique that allows users to intuitively control the evolution of the target object's segmentation.

3. Design Process Overview

Figure 2 provides an overview of our computational approach to designing transformables. The input to our method consists of a character Ψ and a target shape Θ (Figure 2-a). The kinematic structure of the input character is defined by a hierarchical arrangement of joints and rigid body parts, or *bones*. We let $M = (\beta_1, \cdots, \beta_m)$ be a *morphology vector* that stores a scaling parameter for each bone. The morphology vector provides a convenient way of adapting the character's body proportions during the design process, if desired. When β_i is set to 1, bone i has the dimension specified by the input character. The geometry of each bone, which we refer to as its *skeletal mesh*, is procedurally created based on the known positions and orientations of the joint geometries, using a strategy similar to [MTN*15].

A pose, or configuration, of the transformable is defined by a vector $S = (\mathbf{p}, \mathbf{o}, \alpha_1, \cdots, \alpha_n)$, which stores the position and orientation of the root as well as an angle value for each of its joints. The initial pose of the character, which we call *character mode*, is denoted by S_R. Motions are represented by a set of states S_{t_i}, where t_i denotes a discrete time index.

Our goal is to enable the input character to seamlessly transform into the target shape. To this end, we first compute an optimized *embedding* $E = (S_O, M)$. The embedding stores a *folded* character pose, S_O, and a morphology vector. Intuitively, the folded pose, which we call *object mode*, corresponds to a configuration of the character that fits inside the target shape (Figure 2-b). The morphology vector can either be optimized concurrently with the folded pose, or it can be kept fixed. Although the choice is left entirely to the user, our experiments show that adapting the morphology of the character based on the target shape often results in higher quality embeddings. This is because by growing the input skeleton as much as possible while concurrently optimizing the character's folded pose, the overall size of the skin meshes relative to the bones they are attached to is minimized. The resulting designs in both character and object modes are therefore as compact as possible, as shown in Fig 4. Section 4 describes our technical approach to generating optimized embeddings based on functional and aesthetic considerations.

While adopting an optimized folded pose, the character will fit inside the target object. However, given the generic geometry of its skeletal meshes, it can only create a poor depiction of this shape. As a next step, we therefore generate high-fidelity *skin meshes* for each

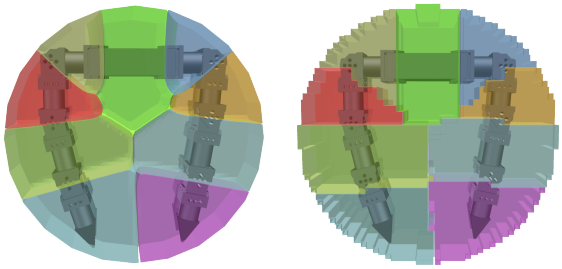

Figure 3: *(a) High-quality segmentation generated using the multiple level set-based growth model. (b) Approximate segmentation generated using the voxel-based flood-fill approach. The latter approach is hundreds of times faster than the former.*

body part of the character (Figure 2-c). We create these geometric models with an approach inspired by character skinning techniques used in the field of computer animation [JDKL14]. Starting from an optimized embedding, the target shape Θ is partitioned into a set of components that are to be assigned to each bone. As detailed in Section 5, we employ a versatile multiple level set representation to generate this segmentation. Starting from the location of each bone in the character's folded configuration, a growth-based model generates overlap-free, smooth skin meshes that partition the target shape based on geodesic distances. Leveraging this model, we also propose intuitive editing modes that afford explicit control in shaping the skin meshes according to user preferences or functional requirements. With the resulting skin meshes attached to its bones, the character then becomes indistinguishable from the target shape when in object mode.

The process of creating skin meshes as summarized above is not informed by the motions the character will have to perform during the transformation process or while performing functional tasks such as walking. While this approximation allows us to devise a computationally-efficient numerical solution, it does prevent the character from performing any meaningful tasks. We therefore post-process the skin meshes with a set of algorithms that are aware of motion requirements, as explained in Section 6. The end result is a functional, 3D-printable transformable character (Figure 2-d). As a final step, a motion sequence $(S_R, \cdots, S_{t_i}, \cdots, S_O)$ between character mode and object mode is also generated.

4. Embedding

Given an initial character Ψ and an object mesh Θ, our goal is to find an embedding $E^* = (S_O^*, M^*)$ that is optimal with respect to objectives capturing both aesthetic and functional aspects of the design:

$$E^* = arg\,min_E \quad L_{\text{compact}} + L_{\text{skin}} + L_{\text{skeleton}} + L_{\text{protrude}}, \qquad (1)$$

where the individual objective terms will be defined shortly.

As the most basic requirement for an optimization procedure, each objective must evaluate a different characteristic of the transformable's design. Ideally, they would be functions of the final skin meshes generated by our design system. Unfortunately, while the model we develop in the next section produces a high-quality segmentation of the target shape, it is too slow to employ in the inner loop of our optimization process. Furthermore, it is impossible to predict how users will choose to shape the individual skin meshes. For these reasons, we resort to a computationally efficient algorithm that instead approximates the segmentation of the target shape.

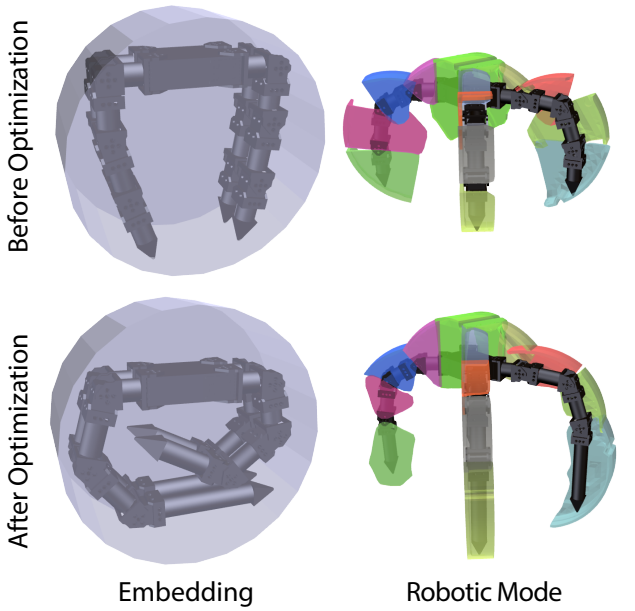

Figure 4: *Segmentation of target object with an initial (top) and optimized embedding (bottom). After the optimization, the skin meshes conform better to the character's bones, and are therefore more aesthetically pleasing.*

To efficiently estimate a segmentation Ω based on an embedding E, we adapt the method of [DdL13] to our problem setting. Briefly, we begin by voxelizing the target object. Each voxel stores a *visited* flag, which is initialized to *false*, as well as the index of the character bone that it is closest to. Using a flood-fill approach, we create a voxel *front* that initially consists of the set of voxels overlapping with the character's skeleton meshes. Each voxel in this initial front is marked as belonging to the bone whose skeletal mesh it overlaps with. At each iteration of the flood-fill algorithm, we create a new front that includes all unvisited voxels neighboring the old front. For each voxel added to the new front, the index of the closest bone is set according to which voxel in the old front it neighbored. The process terminates when no unvisited voxels remain. This simple algorithm fully partitions the volume occupied by the target mesh. While providing only a coarse approximation of the final segmentation, our experiments demonstrate that it leads to comparable optimized embeddings, but hundreds of times faster. Figure 3 compares the segmentation obtained using the flood-fill algorithm against the higher-quality result obtained with the growth-based method described in the following section.

With an efficient way to approximate the segmentation Ω, we now turn our attention to the objectives that drive the process of generating optimized embeddings.

Compactness objective Based on a geodesic distance measure, Ω

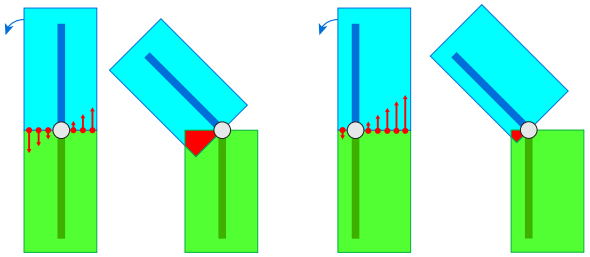

Figure 5: *(a) The highlighted points are sampled on the interface defined by the skin-meshes of parent and child bones; arrows indicate their instantaneous velocities, relative to the joint's range of motion, as calculated in the skin collision objective. Arrows pointing towards the skin-mesh of the parent bone indicate collisions as the joint moves from* $\theta_{\min}$ *to* $\theta_{\max}$*. The larger the arrows are, the more severe the collisions will be. (b) For this embedding, the arrows pointing towards the parent are fewer and shorter, which corresponds to fewer collisions in the unfolded configuration.*

maps points in space to the bone of the character they will be assigned to. The set of all points assigned to bone i therefore constitute its skin mesh. As an intuitive interpretation, for a given embedding, the compactness objective quantifies the degree to which the corresponding skin meshes *conform* to the character's bones. Informally, the bulkier these skin meshes are, the larger the value of the compactness objective will be:

$$L_{\text{compact}}(E) = \sum_{i=1}^{K} D(\mathbf{x}_i, \Omega(E, \mathbf{x}_i))^2, \tag{2}$$

where $\{\mathbf{x}_i\}_{i=1}^{K}$ is a set of points uniformly-sampled on the surface of the skin-meshes and $D(\mathbf{x}, b)$ is the function that calculates the distance from point $\mathbf{x}$ to bone b. As illustrated in Figure 4, optimizing an embedding according to the compactness objective leads to the character being posed such that it uniformly occupies the volume defined by the target object. Consequently, the resulting segmentation is more even, and the skin meshes more aesthetically pleasing.

Skin collision objective Skin meshes are generated by partitioning, based on geodesic distances, the entire volume occupied by the target shape. Consequently, as illustrated in Figure 5, they typically collide with each other while the transformable is performing the motions it was designed for (see Figure 5). The post-processing algorithms described in Section 6 address this shortcoming. Nevertheless, the embedding plays an important role in just how much the geometry of the skin meshes need to be adapted to eliminate self-collisions. To make our optimization process aware of this fact, we define another objective that operates on pairs of bones $(b_c, b_p)_i$ that are connected by joint J_i. For each such pair, the objective estimates how much the skin meshes of the two bones will collide with each other while the joint goes through its full range of motion:

$$\begin{aligned} L_{\text{skin}}(E) = \sum_{i=1}^{n} \sum_{\mathbf{p} \in P_i} [&\max\left(\langle \theta_{\min} \mathbf{L}_i \times \mathbf{p}, \mathcal{N}(\mathbf{p}) \rangle, 0\right) \\ &+ \max\left(\langle \theta_{\max} \mathbf{L}_i \times \mathbf{p}, \mathcal{N}(\mathbf{p}) \rangle, 0\right)]. \end{aligned} \tag{3}$$

As illustrated in Figure 5, P_i is a set of points sampled at the interface between the skin meshes corresponding to joint J_i's child and parent bones. $\mathcal{N}(\mathbf{p})$ represents the surface normal at a point $\mathbf{p}$ that lies on this interface, and we use moving least squares on sampled interface points neighboring $\mathbf{p}$ to estimate the normal. The segmentation Ω is used to estimate both P_i and $\mathcal{N}(\mathbf{p})$. The world coordinates joint axis is denoted by $\mathbf{L}_i$, and $[\theta_{\min}, \theta_{\max}]$ represent J_i's range of motion relative to the embedding angle. The term $\mathbf{L} \times \mathbf{p}$ outputs the velocity of point $\mathbf{p}$ due to infinitesimal changes in J_i's joint angle. Scaling this term by $\theta_{\min}$ and $\theta_{\max}$ therefore provides a first-order approximation of how much point $\mathbf{p}$ moves as the joint goes through its full range of motion. Projecting this estimate onto the normal direction of the interface then measures the expected severity of self-collisions. The example shown in Figure 5 illustrates the benefits of minimizing the skin collision objective. For the same range of motion of the joint, the optimized embedding results in skin meshes that collide significantly less. Consequently, as discussed in Section 6, the skin meshes resulting from the optimized embedding will need to undergo smaller changes during the post-processing steps.

Skeleton collision objective Aiming for tightly packed poses, the character's bones can easily end up intersecting one-another. To ensure the feasibility of the design, we must prevent this from happening. We therefore enclose each bone in a capsule and penalize collisions between each pair of bones not directly connected through a joint. Let P be this set of bone pairs. The collision objective is defined as:

$$L_{\text{skeleton}}(E) = \sum_{(b_i, b_j) \in P} \max(R_i + R_j - G(b_i, b_j), 0)^2, \tag{4}$$

where R_i, R_j are the capsule radii of bones b_i, b_j, and G is the function computing the distance between bone b_i and b_j. For all pairs of bones that are directly linked, we penalize the angle between them if it goes outside the $[\alpha_{\min}, \alpha_{\max}]$ interval that encodes the maximum allowed range of motion. The bounds of this interval are specified by the type of joint used to fabricate the physical prototypes. Note that $[\theta_{\min}, \theta_{\max}]$ in Eq. (3) also represents the range of motion of a joint, but it uses the embedding angle as 0, while $[\alpha_{\min}, \alpha_{\max}]$ uses joint's default angle as 0. **Protrusion objective** To ensure a successful transformation process, the character's bones must be fully confined to the interior of the target shape when in object mode. This requirement is captured by the following objective:

$$L_{\text{protrude}}(E) = \frac{1}{M} \sum_{i=1}^{M} \max(\hat{\phi}(\mathbf{s}_i), 0)^2, \tag{5}$$

where $\hat{\phi}$ is the level set function of the target object and $\{\mathbf{s}_i\}_{i=1}^{M}$ are points sampled on the character's skeletal meshes. In our implementation, all level set functions are negative inside the shape. If $\mathbf{s}_i$ is outside the grid where $\hat{\phi}$ is defined, we replace $\hat{\phi}(\mathbf{s}_i)$ with a large positive number.

Optimization To initialize the optimization, the root of the character is placed at the center of the bounding box of the input mesh. If symmetry is imposed, the symmetry axes of the input mesh and the character will be aligned. The optimization process searches for an optimized embedding by minimizing the objective function (1), a weighted sum of the four constituent sub-objectives we describe above. As weights, we use 10^3 for L_{protrude} and L_{skeleton}, and 1 for L_{skin} and L_{compact} for all our examples. If the morphology vector is

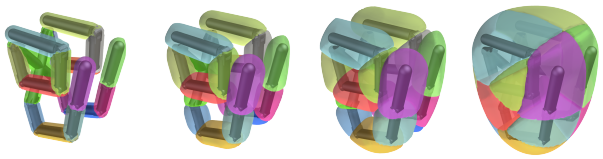

Figure 6: *Starting from the character's skeletal meshes (a), the skin mesh for each bone is grown until the entire volume occupied by the target shape is filled (b-d).*

also being optimized, we add a small regularizer that is proportional to the amount by which each bone's scaling factor deviates from 1. Furthermore, if the user desires a symmetric solutions, then we reparameterize the folded pose and morphology vector using just half the parameters, which are then automatically mirrored when evaluating each objective.

To generate optimized embeddings, we employ the powerful non-linear solver CMA-ES [HO96]. We note that due to the highly non-convex nature of the embedding task, the optimization landscape features many local minima. However, for our problem domain, this is not necessarily an undesirable attribute. In fact, local minima enrich our design space. Consider, for example, the transformable shown in Figure 1. One can certainly envision other ways in which the spider-like character can be made to transform into the target car shape. For example, it could be facing upside down, or its front and hind legs could be folded symmetrically. Each corresponding embedding would represent a local minimum, but it would nevertheless correspond to an interesting design.

To allow users to explore different variations of their design, we provide them with the means to explicitly influence the embedding process. Through a familiar click-and-drag editing mode powered by inverse kinematics, the user can adjust the pose of the character at any time. The optimization, which is efficient enough to run at interactive rates, then starts from this user-provided configuration to generate an optimal embedding. We found that this user-guided optimization strategy is very effective, as it leverages human insight and intuitively allows personal preferences to be accounted for.

5. Geometry Generation

Starting from an embedding $E^\star$, our next goal is to decompose the target object into skin meshes that are to be associated with the transformable's body parts. To accomplish this task, we adopt a versatile multiple level set representation due to its simple definition of constructive solid geometry operations, its ability to gracefully handle topological changes, and the efficient implementation of collision detection queries that it enables. To begin, we define a multiple level set vector $\Phi = (\phi_0, \phi_1, \cdots, \phi_m)$, where ϕ_0 is the level set of the outer domain of the target object, and ϕ_i is the level set representing the skin mesh of bone b_i. As shown in Figure 6 (a), ϕ_i is initialized with the level set corresponding to bone b_i's skeletal geometry. Our strategy for decomposing the target object is to grow each skin mesh gradually, but without overlaps, until the entire volume occupied by the target shape is filled. We note that for all our examples the grid step size of all level sets is 0.01.

Level set growth To evolve the surface of each level set ϕ_i, we define a scalar velocity field $v_i = v_0 - c\tilde{\kappa}_i$, where v_0 is a nominal growth speed, $\tilde{\kappa}_i$ is a function of the mean curvature, and c is a scaling factor. Including the $\tilde{\kappa}_i$ term in computing the velocity field ensures that high-frequency geometric features erode over time. This term is defined as:

$$\tilde{\kappa}_i = \max(|\kappa_i| - \kappa_0, 0) \cdot |\kappa_i|/\kappa_i, \tag{6}$$

where κ_i is the mean curvature of the level set and κ_0 is a threshold. Intuitively, with this definition of $\tilde{\kappa}_i$, geometric features begin to smooth out only if they become too sharp [YCL*15]. Given the scalar velocity field v_i, the evolution of the skin mesh is governed by the *Level Set Equation*:

$$\dot{\phi} + v_i|\nabla\phi| = 0. \tag{7}$$

To solve this PDE, we couple an upwinding scheme for gradient evaluation with an explicit time integrator. Thanks in part to the level set correction step that modulates the growth process, as outlined below, we have not observed any stability problems with this numerical solution when a reasonably-sized time step is used.

Level set correction Using the method outlined above, each skin mesh is evolved independently. Collisions between them are therefore to be expected. We use a level set projection method [LSSF06] to modify the evolving segmentation such that collisions are resolved as soon as they occur. The method is based on the observation that if the multiple level set is a unit segmentation of space, and each level set corresponds to a signed distance field, then at any point $\mathbf{p}$, the two closest level sets ϕ_{s1}, ϕ_{s2} satisfy $\phi_{s1}(\mathbf{p}) + \phi_{s2}(\mathbf{p}) = 0$. If the sum of signed distances from $\mathbf{p}$ to the two level sets is less than zero, then the two level sets intersect one-another. If it is greater than zero, then there are voids between them. Since we aim to have empty space be filled in through the incremental growth process, we only apply the projection step where level sets are found to intersect. The projection operation is simple: For every grid point $\mathbf{p}$ used to define Φ, let the two closest level sets be ϕ_{s1}, ϕ_{s2}. If $e = \phi_{s1}(\mathbf{p}) + \phi_{s2}(\mathbf{p}) < 0$, then we shift the value of both $\phi_{s1}(\mathbf{p})$ and $\phi_{\mathbf{p}}$ by $-e/2$. We use a similar strategy to ensure that no level set ϕ_i overgrows the target shape. To this end, we resolve collisions between ϕ_i and ϕ_0 as before, except that we shift the value of $\phi_i(\mathbf{p})$ by $-e$ and keep $\phi_0(\mathbf{p})$ intact. Before applying the projection step, we apply the fast marching method [Set95] to ensure that each level set corresponds to a signed distance field.

Why growth? It is tempting to consider using the level set projection method to directly fill out the entire empty space when segmenting the target object. However, our growth-based model has several important advantages. First, it achieves a segmentation that is based on geodesic distances, rather than an Euclidean metric. As noted in the character skinning literature [DdL13, JDKL14], this feature is crucial in avoiding artifacts when concave target shapes are provided as input. Furthermore, because the segmentation process is incremental, we have observed it is better behaved numerically than our early experiments that relied solely on level set projections. Last, our growth-based method allows us to seamlessly incorporate user input in the skin mesh generation process.

Mesh transfer The segmentation generated through level set growth is unaware of the motions that the transformable will need to perform. To deal with this shortcoming, we first address collisions between each pair of bones that are connected by a joint

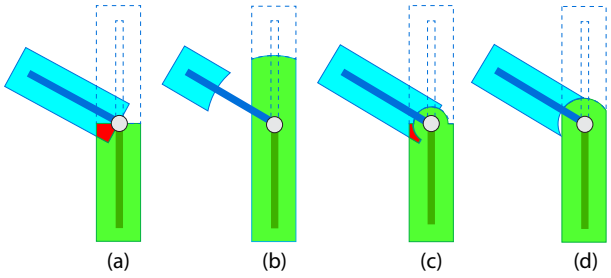

Figure 7: *(a) After the geometry generation process described in Section 5, collisions exist between the skin meshes of bones connected by a joint. (b) A large transfer radius R is overly-conservative. Although it resolves collisions, it alters the appearance of the skin meshes too much. (c) If R is too small, collisions are reduced, but they still exist. (d) The smallest transfer radius R that resolves collisions.*

since their relative motions are already determined. As shown in Figure 7 (a), collisions between the original skin meshes of a joint's child and parent bones are likely to be significant once its full range of motion is considered. To resolve these collisions, we developed a technique that we call *mesh transfer*. In effect, this step adjusts the segmentation that was produced with our growth-based method such that it becomes collision-aware. This is achieved by transferring over the collision-causing geometry from one bone to the other, in a way that does not affect the overall shape of the transformable when it is in object mode.

Due to fabrication-imposed constraints, the hinge joints used in transformables only have a single degree-of-freedom. Consequently, we can parameterize the geometry to be transferred over with the aid of a cylindrical shape. The origin of this cylinder coincides with the position of the joint. It is aligned with the axis of the joint and has a radius R. Using level set operations, the transfer operation is defined as:

$$\phi_{\text{transfer}} = \max(\phi_{\text{cyl}}(R), \phi_{\text{child}}) \quad (8)$$

$$\phi_{\text{child}} = \max(\phi_{\text{child}}, -\phi_{\text{transfer}}(R)) \quad (9)$$

$$\phi_{\text{parent}} = \min(\phi_{\text{parent}}, \phi_{\text{transfer}}(R)). \quad (10)$$

Figure 7 illustrates the effect of the mesh transfer step for a range of values of the cylinder's radius: as R increases, collisions vanish. However, if R is too large, the changes to the original segmentation become needlessly intrusive. It is worth noting that the mesh transfer operation is performed when the transformable is posed in object mode. Consequently, for any value of R, large or small, the net volume occupied by the skin meshes does not change. As a second remark, the skin collision objective that guides the generation of optimized embeddings promotes the use of small transfer radii, and therefore smaller changes to the segmentation during this post-processing step.

The optimal transfer radius $R^\star$, one that is as small as possible yet eliminates all collisions, depends non-trivially on the shape of the skin meshes that the transfer operation is applied to, as well as on the range of motion of the joint. While we cannot compute $R^\star$ analytically, a simple bisection search algorithm is very effective in finding it: We evaluate the quality of any candidate transfer radius

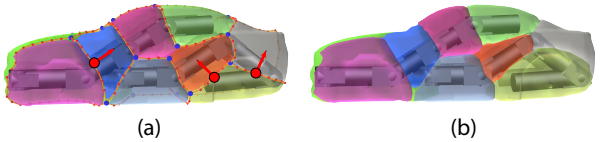

Figure 8: *Before editing, the wheel sockets are fragmented into several parts. (a) Through the editing tools we provide, users can edit the segmentation by dragging point handles on the skin mesh interfaces. (b) After editing, each wheel socket is contained by a single skin mesh.*

R using:

$$L_{\text{Collision}} = \sum_{i=1}^{T'} \text{LevelSetCollision}(\tilde{\phi}_{\text{child}}, \tilde{\phi}_{\text{parent}}, S_i), \quad (11)$$

where $\tilde{\phi}_{\text{child}}, \tilde{\phi}_{\text{parent}}$ are the skin meshes of the parent and child bone after the transfer operation, $\{S_i\}_{i=1}^{T'}$ are sampled character configurations across the joint's range of motion, and LevelSetCollision is a function that evaluates the net overlap between the two level sets when they are positioned relative to each other according to S_i. Starting with a conservative value of R, we iteratively increase it if collisions are detected, or decrease it otherwise, as typical done in interval halving methods. Once the search procedure converges, the skin meshes are finalized using the steps outlined in Eq. (8). Users can choose to transfer geometry from child to parent, or vice versa, to further direct the design of their transformable.

User interaction An important goal of our work is to keep users involved in the design process. This allows personal preferences and intuition to be taken into account when creating transformables. We provide several ways in which users can control the segmentation of the target object. First, at any time, users can inhibit the growth process for any selected skin mesh. This feature is important if users would prefer some of the resulting skin meshes to be more compact than others. To implement this feature, the scalar velocity field that controls the growth process of a skin mesh is set to 0, and the level set projection operator is modified to no longer change the values of the associated level set during the correction step. Users can also remove entirely any subset of skin meshes. The remaining ones then simply grow to fill in the resulting void. The contrast between some body parts of the transformable having only skeletal meshes, while others feature prominent skin meshes, can lead to interesting designs as shown in our results. Our design system also provides an editing mode that allows users to explicitly manipulate the shape of any skin mesh as described next.

Editing skin meshes To provide direct control over the shape of the skin meshes, we propose an intuitive editing mode that integrates seamlessly with our growth-based model. At a high-level, this editing mode translates user inputs into a change of the scalar velocity field that guides the evolution of the level sets. Figure 8 illustrates the editing mode. Users are provided with a visualization of the interfaces between level sets. These interfaces act as handles that the user can drag along the surface of the target object. As the shape of the interfaces changes due to the user actions, the geometry of the skin meshes follows suit.

Our first task is to generate a set of points $Q_{i,j}$ that lie on the surface of the target object and delineate the interface between two

neighboring level sets ϕ_i and ϕ_j. We obtain $Q_{i,j}$ by merging points from Q_i and Q_j that are sufficiently close. Here, Q_i represents the contour of ϕ_i along the surface of the target shape, ϕ_0. Q_j is defined analogously. To compute this contour, we begin with a seed point $\mathbf{q}_0$ that is added to Q_i. This seed point is found by repeatedly projecting a point $\mathbf{q}$ onto ϕ_i and ϕ_0 until convergence. Starting from a point $\mathbf{q}_t \in Q_i$, the next point along the contour can be found by tracing along the mesh surface in a direction tangent to $\phi_i(\mathbf{q}_t)$:

$$\mathbf{q}_{t+1} = P_{\phi_0}(P_{\phi_i}(\mathbf{q}_t + \text{Normalize}(\nabla\hat{\phi_0}(\mathbf{q}_t) \times \nabla\phi_i(\mathbf{q}_t)) \cdot h)). \quad (12)$$

where P_ϕ projects points on the surface of level set ϕ, and h controls the spacing between consecutive points in Q_i. The process terminates when the next candidate point to add to Q_i is sufficiently close to $\mathbf{q}_0$.

Once the points in $Q_{i,j}$ are generated, they act as an editing handle for the user. Briefly, the user can select any point $\mathbf{q}_t \in Q_{i,j}$ and move it to a new location on the surface of the target object. Using a soft-selection approach, points in $Q_{i,j}$ neighboring $\mathbf{q}_t$ follow. A fall-off function inversely proportional to distance from $\mathbf{q}_t$ is used to compute their relative displacements. The user input therefore results in a new set of points, $\bar{Q}_{i,j}$. Our next task is to modify level sets ϕ_i and ϕ_j such that $\bar{Q}_{i,j}$ becomes their new interface. In the following, we present the approach we use to modify ϕ_i, with the understanding that an equivalent method is applied to ϕ_j. The new interface $\bar{Q}_{i,j}$ imposes a set of constraints that must be satisfied at the next iteration of the growth process:

$$\phi_i^{t+1}(\mathbf{q}) = 0, \quad \forall \mathbf{q} \in \bar{Q}_{i,j}. \quad (13)$$

To satisfy these constraints, we modify the velocity field governing the level set evolution described by Eq. (7). In particular, given the simple nature of our time integrator, we compute the velocity u_i that would satisfy the constraints:

$$u_i(\mathbf{q}) = -\phi_i^t(\mathbf{q})/\delta t, \quad \forall \mathbf{q} \in \bar{Q}_{i,j}, \quad (14)$$

where $\phi_i^t(\mathbf{q})$ is the current value of the level set at $\mathbf{q}$, and δt is the time step used to integrate the level set equation forward in time. To ensure that $u_i(\mathbf{q})$ takes on the correct values for all points in $\bar{Q}_{i,j}$, we must compute appropriate velocities at the grid points used to represent ϕ_i. With $\mathbf{q}$ expressed through trilinear interpolation of grid points $\mathbf{v}$, the relationship we seek is:

$$\sum_{\mathbf{v} \in \text{Vert}(\mathbf{q})} w_{\mathbf{v}}(\mathbf{q}) u_i(\mathbf{v}) = u_i(\mathbf{q}), \forall \mathbf{q} \in \bar{Q}_{i,j}, \quad (15)$$

where $\text{Vert}(\mathbf{q})$ is the set of grid points of the voxel that contains $\mathbf{q}$, and $w_{\mathbf{v}}(\mathbf{q})$ are trilinear interpolation weights associated with each grid point $\mathbf{v} \in \text{Vert}(\mathbf{q})$. In matrix form, this equation takes on the standard form, $A\mathbf{x} = \mathbf{b}$, where the unknown vector $\mathbf{x}$ contains the velocity field u_i for the underlying grid points. We obtain a solution to this equation using regularized least-squares, $\mathbf{x} = (A^T A + \lambda I)^{-1} A^T \mathbf{b}$, where λ is set to be 1. With the new velocity terms computed, all that remains is to update the final velocity field that controls the growth process for ϕ_i:

$$v_i = v_0 - c\tilde{\kappa}_i + u_i. \quad (16)$$

For points in $Q_{i,j}$ that are unaffected by user edits, the new velocity term evaluates to zero and the typical geometry generation process is unaffected. However, for all other points, the evolution of the level sets is biased such that the resulting skin meshes take on the profile delineated by $\bar{Q}_{i,j}$. Due to the regularized solver we employ to compute u_i, and because users often provide incremental changes to the interface $\bar{Q}_{i,j}$, we continuously compute updates to the velocity field using Eq. (16). $Q_{i,j}$ and $\bar{Q}_{i,j}$ are updated with each iteration. The computational overhead of this update step is negligible, so users are provided with a seamless interactive experience. Note that every time the user edits the segmentation, mesh transfer operation will be applied to resolve any incurred collisions. In this way, the user can always make edits on the segmentation that is used for fabrication.

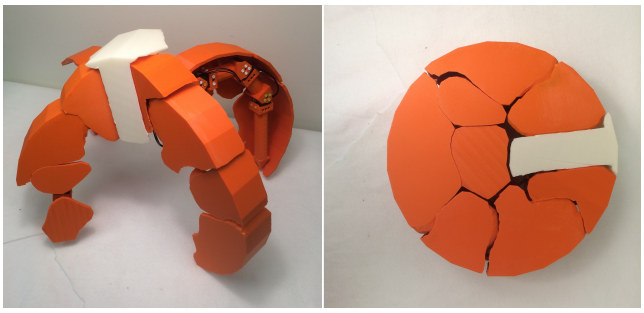

Figure 9: *Extension: A self-transforming robot prototype with motors installed at the joints.*

6. Design Finishing

Folding sequence generation The mesh transfer step outlined above ensures there are no collisions between the skin meshes of child-parent bone pairs. Nevertheless, collisions between body parts that are not directly connected by a joint can still occur. Our design system therefore searches for a sequence of folding states, $(S_R, \cdots, S_{f_i}, \cdots, S_O)$, between character and object mode (S_R and S_O, respectively) that minimizes collisions. The physics-based approach presented by Zhou and his colleagues [ZSMS14] could be used for this purpose. However, because we cannot guarantee that a collision-free folding sequence exists for any transformable design, we resort to a greedy algorithm instead. Because of the reversibility of the folding sequence, the algorithm aims to find an unfolding sequence, $(S_O, \cdots, S_{f_i}, \cdots, S_R)$. Starting from the object mode S_O, if there are unfolded joints, each time we choose one joint to unfold all the way that causes least amount of collisions. We use Eq. (11) to evaluate collisions between the character's skin and skeletal meshes. For all our examples, after the user-guided geometry generation step, the best folding sequence is found within 1 minute of computation.

Final carving If the folding sequence that is generated is not collision-free, we apply the level-set based carving method in [GJG16] to ensure that the final design of the transformable is functional. Importantly, we carve the geometry of each skin mesh to resolve collisions with other skin meshes, as well as with the character's skeletal meshes.

7. Results

We used our computational method to create a variety of transformable designs, as seen in Figure 10. To validate our results, we

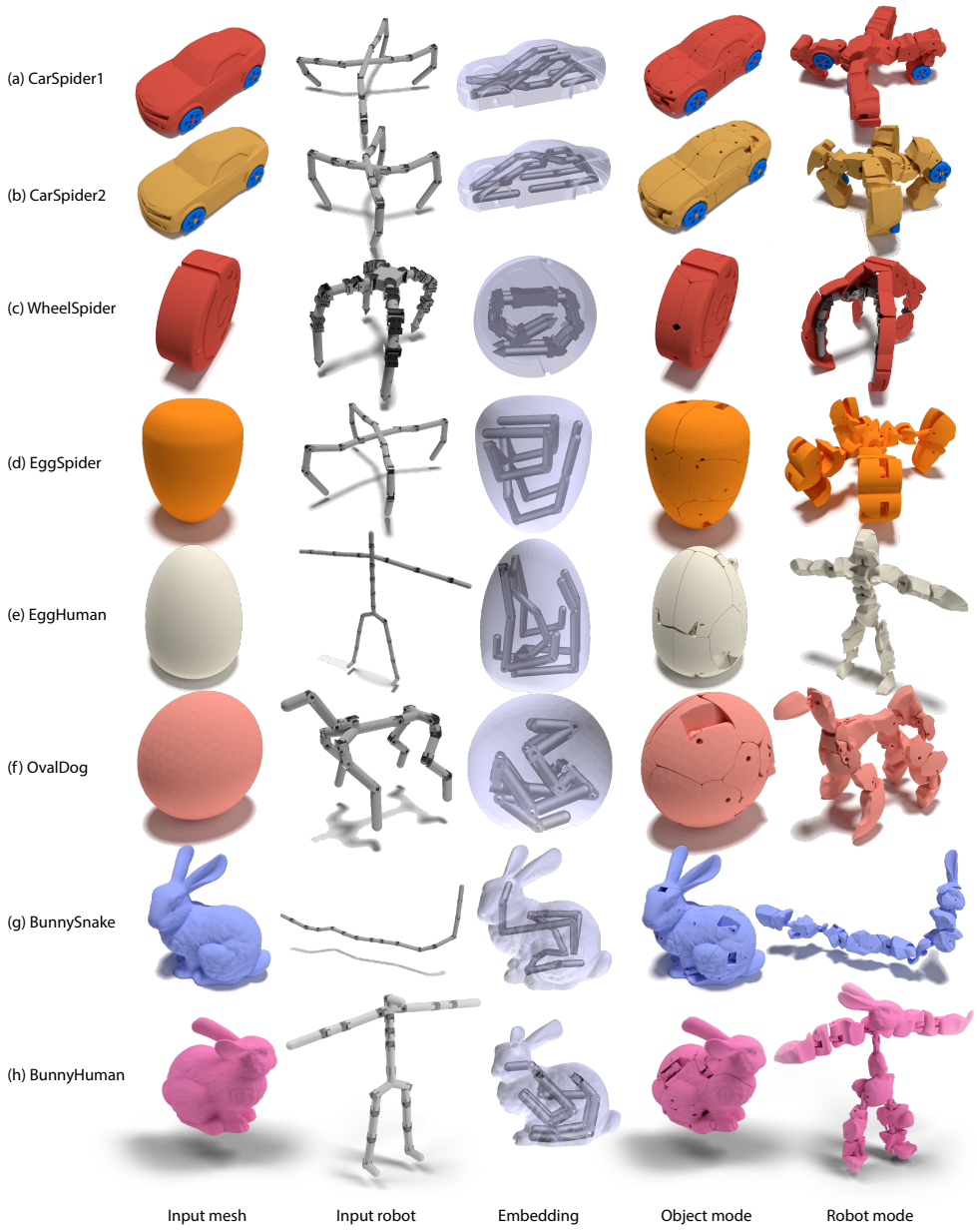

Figure 10: *A gallery of transformable designs. Their dynamic folding and unfolding processes are shown in the supplemental video.*

Example	#Bone	Grid Size	Time (s)			
			Embedding	Segmentation	Folding Opt.	Finishing
CarSpider1	17	23×18×50	x	x	32.0	127.1
CarSpider2	17	23×18×50	x	x	36.7	132.4
WheelSpider	17	26×41×41	13.0	4.3	12.0	148.1
EggHuman	20	44×54×44	34.5	13.0	50.6	262.6
EggSpider	17	41×50×41	23.5	10.0	24.7	177.4
OvalDog	19	30×37×37	15.1	4.2	13.7	153.9
BunnySnake	15	45×54×54	31.2	18.6	15.7	279.6
BunnyHuman	20	27×33×34	x	11.1	14.1	117.1

Table 1: *Timing information for automatically generated examples. x indicates that user interaction is involved.*

fabricated three transformable action figures (Figure 1). The transformation process for both simulated and fabricated prototypes is best seen in the supplementary video. We run our design system on a standard PC with 2.7 GHz Intel Core i7 processor.

Our design system enables functional transformable designs to be generated either fully automatically, or under the guidance of the user. The results shown in Fig 10 (d,e,f,g), for example, were created without any user intervention. These results show that our method does not depend on user input to create valid segmentations or collision-free designs. For the examples shown in Fig 10 (a-c), the embedding stage was user-guided, as shown in the supplementary video. As can be seen, within about a minute, the user guides

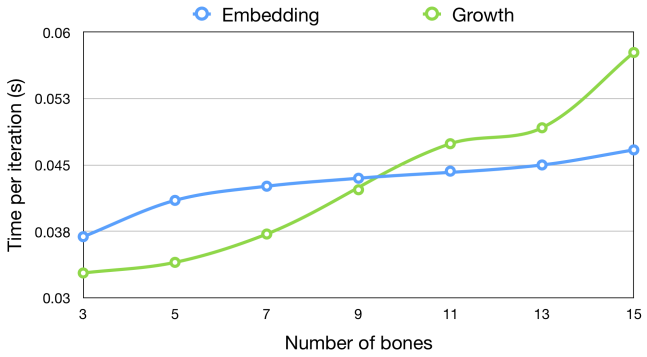

Figure 11: *Timing information for embedding and growth-based segmentation.*

the system into generating three different ways of embedding the skeleton into the car object. Each one of these is valid and corresponds to a different local minimum of the embedding optimization process. As demonstrated also by the result shown in Fig 10 (h), where the head of the character is purposefully aligned with the head of the bunny, this interactive tool enables users to effectively create the transformable designs they envision.

For the results shown in Fig 10(a, b) we further used our interactive, user-guided segmentation editing method, which is also shown in the results video. The goal here is to efficiently edit the resulting skin meshes such that the wheel section is not split up into multiple parts. If the users would so choose, they could also use this editing mode to align the segmentation boundaries with specific seams in the target model. Such considerations are largely aesthetic, which is why they are left to the user.

As our results indicate, the space of possible transformable designs is vast. The same input character can be designed to transform into different target shapes (Figure 10(a,d) and (e,h)). Conversely, characters with unmistakably different morphologies can seamlessly transform into the same target shape (Figure 10(d,e) and (g,h))). Even for the same input character and target shape pair, our computational approach can easily create, under the guidance of the user, distinctly different transformable designs (Figure 10(a,b)).

To generate final skin meshes for the tranformables, we can use two strategies, that each start with CSG operations between the level sets of the skin meshes and the target object. This step produces high-quality 3d models that capture all the surface details featured by the input shape. What is different, however, is that our system can either generate volumetric models, or shells. The volumetric models can be directly attached to the character's skeletal meshes through CSG operations. The shells can be attached to the underlying skeleton through lightweight support structures. Given the segmentation of the input shape, it is relative easy for us to manually create the support structures that won't interfere with other parts. Each strategy has its own advantages. Transformables with volumetric skin meshes are structurally stable when in object mode, which could be important if they are designed to be easily stowed away or transported, for example. Shell skin meshes, on the other hand, are much lighter, and are therefore less likely to hinder the functional capabilities of the character. Furthermore, if the shells attach to the character skeleton through snap-on or magnetic connectors, then they can be easily replaced. This would allow the same character to enrobe different skins in order to transform into different types of objects. It is worth noting that, although not demonstrated, our design system can easily support this type of application by simply keeping the character's morphology vector fixed while optimizing embeddings.

Self-transforming robotic device As an exciting extension that we wish to pursue in future work, our method can be applied to designing fabricatable robotic devices that shape-shift to perform different tasks. Towards this application, we fabricated a wheel robot (Figure 9) using our design platform. However, it highlights an important consideration that is not yet addressed. The robot's servomotors are not strong enough to carry the full weight of the skin meshes. Ideally, actuation limitations should therefore be taken into account at design time. To bypass actuation constraints, we further tested our designs in a physically-simulated environment. As the results video shows, in our black-box physically simulated environment, transformables can transition from object mode to character mode, walk successfully, and then change back into object mode. The walking motions were generated using the method proposed by Megaro and his colleagues [MTN*15].

8. Limitations and Future Work

We presented a computational approach to designing transformable robots that shape-shift in order to take on vastly different forms. Our method allows users to drive the design process by injecting their own intuition and personal preferences into the transformables they create. To validate our computational models, we designed a variety of simulated and physical prototypes.

Our method is not without limitations. We would like, for example, to investigate more powerful ways of creating motion sequences for the transformation process. Ideally, this step would be tightly integrated with the embedding optimization and skin mesh generation algorithms, in order to reduce or eliminate altogether the need to post-process the designs. Higher-level goals could also be considered while planning transformation sequences. For example, we would like to ensure that the robot does not lose balance while transforming to and from object mode. Designing the skin meshes such that they enhance the functionality of the robot is also an interesting direction for future work. For example, they can be designed to house electromechanical components and wiring, they can be optimized to be lightweight, yet sufficiently strong to bear load, or they can be made to give the transformation process a more puzzle-like feel.

References

[BBJP12] BÄCHER M., BICKEL B., JAMES D. L., PFISTER H.: Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31*, 4 (2012), 47–1. 2

[BCT15] BÄCHER M., COROS S., THOMASZEWSKI B.: Linkedit: Interactive linkage editing using symbolic kinematics. *ACM Trans. Graph. 34*, 4 (July 2015), 99:1–99:8. URL: http://doi.acm.org/10.1145/2766985, doi:10.1145/2766985. 2

[BWBSH14] BÄCHER M., WHITING E., BICKEL B., SORKINE-HORNUNG O.: Spin-It: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 33*, 4 (2014), 96:1–96:10. 2

[CCA*12] CALÌ J., CALIAN D. A., AMATI C., KLEINBERGER R., STEED A., KAUTZ J., WEYRICH T.: 3d-printing of non-assembly, articulated models. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 130. 2

[CLM*13] CEYLAN D., LI W., MITRA N. J., AGRAWALA M., PAULY M.: Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 186. 2

[CTN*13] COROS S., THOMASZEWSKI B., NORIS G., SUEDA S., FORBERG M., SUMNER R. W., MATUSIK W., BICKEL B.: Computational design of mechanical characters. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 83. 2

[DdL13] DIONNE O., DE LASA M.: Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2013), SCA '13, ACM, pp. 173–180. URL: http://doi.acm.org/10.1145/2485895.2485919, doi:10.1145/2485895.2485919. 4, 6

[GF05] GIBOU F., FEDKIW R.: A fast hybrid k-means level set algorithm for segmentation. In *4th Annual Hawaii International Conference on Statistics and Mathematics* (2005), Hawaii, USA, pp. 281–291. 3

[GJG16] GARG A., JACOBSON A., GRINSPUN E.: Computational design of reconfigurables. *ACM Transactions on Graphics (TOG) 35*, 4 (2016). 3, 8

[HCLC16] HUANG Y.-J., CHAN S.-Y., LIN W.-C., CHUANG S.-Y.: Making and animating transformable 3d models. *Computers & Graphics 54* (2016), 127–134. 2

[HO96] HANSEN N., OSTERMEIER A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on* (1996), IEEE, pp. 312–317. 6

[JDKL14] JACOBSON A., DENG Z., KAVAN L., LEWIS J.: Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses* (2014). 4, 6

[KLY*14] KOO B., LI W., YAO J., AGRAWALA M., MITRA N. J.: Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics (TOG) 33*, 6 (2014), 217. 2

[LFL09] LO K.-Y., FU C.-W., LI H.: 3d polyomino puzzle. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 157:1–157:8. URL: http://doi.acm.org/10.1145/1618452.1618503, doi:10.1145/1618452.1618503. 2

[LHAZ15] LI H., HU R., ALHASHIM I., ZHANG H.: Foldabilizing furniture. *ACM Transactions on Graphics, (Proc. of SIGGRAPH 2015) 34*, 4 (2015). 2

[LLMZ16] LI D., LEVIN D. I., MATUSIK W., ZHENG C.: Acoustic voxels: Computational optimization of modular acoustic filters. *ACM Trans. Graph. 35*, 4 (2016). doi:10.1145/2897824.2925960. 2

[LSSF06] LOSASSO F., SHINAR T., SELLE A., FEDKIW R.: Multiple interacting liquids. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 812–819. 6

[MAB*15] MUSIALSKI P., AUZINGER T., BIRSAK M., WIMMER M., KOBBELT L.: Reduced-order shape optimization using offset surfaces. *ACM Trans. Graph 34*, 4 (2015), 102. 2

[MTN*15] MEGARO V., THOMASZEWSKI B., NITTI M., HILLIGES O., GROSS M., COROS S.: Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 216. 3, 10

[MUB15] MARTIN T., UMETANI N., BICKEL B.: Omniad: data-driven omni-directional aerodynamics. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 113. 2

[PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., SORKINE-HORNUNG O.: Make It Stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 32*, 4 (2013), 81:1–81:10. 2

[Rus14] RUSSON M.-A.: Real transformer robot, 2014. Accessed on Jan 1, 2017. 1

[Set95] SETHIAN J. A.: A fast marching level set method for monotonically advancing fronts. In *PROC. NAT. ACAD. SCI* (1995), pp. 1591–1595. 6

[SFCO12] SONG P., FU C.-W., COHEN-OR D.: Recursive interlocking puzzles. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 128:1–128:10. URL: http://doi.acm.org/10.1145/2366145.2366147, doi:10.1145/2366145.2366147. 2

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. In *Computer graphics forum* (2008), vol. 27, Wiley Online Library, pp. 1539–1556. 3

[SZ15] SUN T., ZHENG C.: Computational design of twisty joints and puzzles. *ACM Trans. Graph* (2015). 3

[TCG*14] THOMASZEWSKI B., COROS S., GAUGE D., MEGARO V., GRINSPUN E., GROSS M.: Computational design of linkage-based characters. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 64. 2

[UIM12] UMETANI N., IGARASHI T., MITRA N. J.: Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4 (2012), 86–1. 2

[UKSI14] UMETANI N., KOYAMA Y., SCHMIDT R., IGARASHI T.: Pteromys: interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph. 33*, 4 (2014), 65–1. 2

[UTZ16] URETA F. G., TYMMS C., ZORIN D.: Interactive Modeling of Mechanical Objects. *Computer Graphics Forum* (2016). doi:10.1111/cgf.12971. 2

[VC02] VESE L. A., CHAN T. F.: A multiphase level set framework for image segmentation using the mumford and shah model. *International journal of computer vision 50*, 3 (2002), 271–293. 3

[Wag13] WAGNER K.: Autobots assembled: How transformers come to life, 2013. Accessed on Jan 1, 2017. 2

[WL16] WON J., LEE J.: Shadow theatre: discovering human motion from a sequence of silhouettes. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 147. 3

[XLF*11] XIN S., LAI C.-F., FU C.-W., WONG T.-T., HE Y., COHEN-OR D.: Making burr puzzles from 3d models. In *ACM SIGGRAPH 2011 Papers* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 97:1–97:8. URL: http://doi.acm.org/10.1145/1964921.1964992, doi:10.1145/1964921.1964992. 2

[YCL*15] YAO M., CHEN Z., LUO L., WANG R., WANG H.: Level-set-based partitioning and packing optimization of a printable model. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 214. 3, 6

[YmSS*07] YIM M., M. SHEN W., SALEMI B., RUS D., MOLL M., LIPSON H., KLAVINS E., CHIRIKJIAN G. S.: Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics Automation Magazine 14*, 1 (March 2007), 43–52. doi:10.1109/MRA.2007.339623. 2

[ZSMS14] ZHOU Y., SUEDA S., MATUSIK W., SHAMIR A.: Boxelization: folding 3d objects into boxes. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 71. 2, 8