GarmentCode: Programming Parametric Sewing Patterns

MARIA KOROSTELEVA, ETH Zurich, Switzerland
OLGA SORKINE-HORNUNG, ETH Zurich, Switzerland

| Early 1800 (Regency) | 1930s | Modern | Modern | 1950s | 1920s |

Fig. 1. Dress examples showcasing women's garment styles inspired by different epochs in fashion history. All are sampled from a single parametric garment configurator created with GarmentCode.

Garment modeling is an essential task of the global apparel industry and a core part of digital human modeling. Realistic representation of garments with valid sewing patterns is key to their accurate digital simulation and eventual fabrication. However, little-to-no computational tools provide support for bridging the gap between high-level construction goals and low-level editing of pattern geometry, e.g., combining or switching garment elements, semantic editing, or design exploration that maintains the validity of a sewing pattern. We suggest the first DSL for garment modeling – GarmentCode – that applies principles of object-oriented programming to garment construction and allows designing sewing patterns in a hierarchical, component-oriented manner. The programming-based paradigm naturally provides unique advantages of component abstraction, algorithmic manipulation, and free-form design parametrization. We additionally support the construction process by automating typical low-level tasks like placing a dart at a desired location. In our prototype garment configurator, users can manipulate meaningful design parameters and body measurements, while the construction of pattern geometry is handled by garment programs implemented with GarmentCode. Our configurator enables the free exploration of rich design spaces and the creation of garments using interchangeable, parameterized components. We showcase our approach by producing a variety of garment designs and retargeting them to different body shapes using our configurator. The library and garment configurator are available at https://github.com/maria-korosteleva/GarmentCode.

CCS Concepts: • **Computing methodologies** → **Shape representations**; *Shape inference*; *Parametric curve and surface models*; *Graphics file formats*; • **Applied computing** → *Media arts*.

Additional Key Words and Phrases: Garment Modeling, Sewing Patterns

Authors' addresses: Maria Korosteleva, maria.korosteleva@inf.ethz.ch, ETH Zurich, Zurich, Switzerland; Olga Sorkine-Hornung, sorkine@inf.ethz.ch, ETH Zurich, Zurich, Switzerland.

1 INTRODUCTION

"Instead of making people want what we made, we will make what they want". This motto is one of the motivations behind product configurators and other services that aim to create fashion products tailored to the needs of individual customers. In stark contrast, the fast-fashion industry focuses on mass-production of a variety of designs in standard sizes, relying on trends and statistics of body shapes. The customers are left to choose from available designs and standardized sizes, which often do not fit their body shape. This results in customer dissatisfaction, as well as the production of dead stock, which intensifies the negative impact of the garment industry on the climate [Bick et al. 2018]. Unfortunately, the creation of custom-made garments is an expensive, labor-intensive process, unattainable for most people. Recent years saw launches of services for made-to-order clothes of popular garment types, such as Amazon t-shirts [Amazon 2023] and Unspun jeans [Unspun 2023], offering a middle ground between the two extremes mentioned above. These services allow making some limited adjustments to pre-defined designs and producing garments on demand according to the customer's individual body shape. However, extending such services to general garment designs remains a challenge. Current production-grade tools are oriented towards creating single designs: tools like Clo3D [CLO Virtual Fashion 2022] do not support creating parametric garments. Existing research works on alternative garment modeling tools either do not take sewing patterns into consideration or do not come close to supporting the required complexity, see Tab. 1.

To facilitate the development of personalized clothing designs, we propose GarmentCode, a domain-specific language (DSL) for sewing pattern construction, adapting the principles of object-oriented programming to naturally allow parameterization and algorithmic support while efficiently handling the design complexity. First, GarmentCode embodies a hierarchical representation of garment sewing patterns with abstract components: the basic component of a garment is a panel, which is a 2D piece of fabric, and more involved, higher-level components can be composed and modified using a range of provided operators. Second, GarmentCode extends stitch definition to allow specification of connectivity between high-level components through semantic component interfaces. Stitch abstraction enables the interchangeability of components with the same semantic interfaces even if the underlying geometry of those differs. In turn, this interchangeability enables modularity: simplicity of integrating novel components into the system and easy garment construction from existing components (see an example in Sec. 5.2). Third, our method supports the construction of advanced garment features, such as gathers and darts, currently missing in existing large-scale datasets and modeling approaches [Bertiche et al. 2020; Heming et al. 2020; Korosteleva and Lee 2021]. An example of a garment program is given in Fig. 11.

Our conceptual framework enables the definition of rich parametric design spaces, which we demonstrate in our garment configurator. It allows the construction of a variety of garment styles, from simple tops, skirts, and pants, to more elaborate complex evening gowns (Figs. 1 and 10), while using a limited number of parametric components, and with support for adjusting the designs according to different body measurements. A key advantage of using a configurator is the automatic maintenance of a valid sewing pattern and the inherent interchangeability of components, enabling effortless design exploration without having to worry about low-level sewing constraints or deep expert knowledge of patternmaking. Another advantage is the ability to adapt garment designs by exposing intuitive, physically meaningful parameters, such as various body measurements and style parameters.

Limited versions of such configurators can be shared with end customers to let them adjust designs within the limits acceptable for fabrication or to dress custom virtual avatars in video games and metaverses. Such features would welcome users to become part of the creative process and would allow to better satisfy their individual tastes. More detailed design space definitions may assist designers in quickly obtaining starting sewing patterns for their creative exploration. Another envisioned application is the creation of parametric templates for synthetically generated garment design collections, which currently often suffer from limited variety and simplistic designs [Bertiche et al. 2020; Heming et al. 2020; Korosteleva and Lee 2021]. Such design datasets play an important role in different data-driven applications [Chen et al. 2022; Jiang et al. 2020; Korosteleva and Lee 2022; Wang et al. 2018], attracting much research interest in recent years.

Our implementation of GarmentCode and configurator is publicly available on GitHub[1].

Table 1. Comparison of GarmetCode with baseline systems. Here, "Definition" means the ability to specify a single sewing pattern, "Tools" refer to instruments supporting pattern construction, "Modular construction" enables specifying garment parts as independent modules and constructing new designs by part combinations, "Continuous" and "Categorical" are parameters with corresponding value types, while a "Dependent" parameter's value depends on other parameters, allowing for complex parameterization and resolution of parameter value conflicts. Commercial CAD tools like Clo3D [2022] do not support true parameterization while existing parametric garment systems [Korosteleva and Lee 2021] support only limited types of parameterization and do not provide any modeling tools; neither of them fully supports modularity. *Limited support.*

		CAD (Clo3D [2022])	Korosteleva and Lee [2021]	GarmentCode (ours)
Modeling	Definition	✓	✓	✓
	Tools	✓	✗	✓*
	Modular construction	✓*	✗	✓
Parameters	Continuous	✗	✓	✓
	Categorical	✗	✗	✓
	Dependent	✗	✗	✓

2 RELATED WORK

2.1 Garment modeling

Industry-grade tools for garment modeling, such as Clo3D [2022], rely on artists manually drawing and adjusting sewing pattern shapes. Such tools enable the creation of complex garments, but the design process is often tedious. Support for semantic parameterization is very limited: Clo3D provides only one pattern parameterized by body measurements – a bodice, and it is non-extensible with a fixed parameter set. The modular configurator is akin to categorical parameterization for component combination, but it offers a fixed set of component types (upper body, sleeve, collar, cuff) and has only one hierarchy level, and no other parameterizations are supported (Tab. 1). Alternative methods for garment modeling are a subject of ongoing research: for example, editing a garment model in 3D, and automatically readjusting [Bartle et al. 2016] or inferring a sewing pattern [Liu et al. 2018; Meng et al. 2012; Pietroni et al. 2022; Wang et al. 2009; Wolff et al. 2023] corresponding to the 3D garment design. Another inspiring line of work aims to maximally reduce the modeling effort by computing 3D garment models from designs sketches [Chowdhury et al. 2022; Fondevilla et al. 2021; Li et al. 2018; Wang et al. 2018], reconstruct sewing patterns from images [Jeong et al. 2015; Yang et al. 2018], or 3D capture [Bang et al. 2021; Chen et al. 2015; Hasler et al. 2007; Korosteleva and Lee 2022]. However, all these methods focus on producing a single garment. GarmentCode offers an alternative perspective on garment modeling, offering a design toolkit for *parametric* sewing patterns, which allow for fast and convenient exploration of a created design space. Moreover, it offers capabilities to explicitly condition the design on body measurements. Working in the space of sewing patterns

ensures controllable and fabrication-plausible garment designs at each stage of the construction process.

2.2 Garment modeling at scale

The rise of deep learning sparked an interest in data-driven techniques in many domains, including various tasks related to garments, such as virtual try-on, modeling, and neural simulation, hence creating a unique demand to generate garment designs at scale to be used in synthetic datasets for training. Some of the works in the area [Bertiche et al. 2020; Chen et al. 2015] rely on the combinatorial effect of constructing designs from a set of sub-components, such as multiple options for sleeves, upper body, and lower body garments. In these works, the combinations are performed on 3D geometry, which may result in physically implausible 3D models. These approaches do not provide corresponding sewing patterns. Other works [Jiang et al. 2020; Korosteleva and Lee 2021; Wang et al. 2018] rely on sampling designs from a set of custom parametric sewing pattern templates, varying the continuous style parameters like length and width of garment elements. Korosteleva and Lee [2021] provide a framework for describing garment templates, but do not allow defining discrete parameters, or parameter dependencies, nor support for base sewing pattern description, as indicated in Tab. 1, rendering it hard to use for complex designs.

GarmentCode combines continuous and discrete approaches to design variation into one framework, allowing creating garment design templates with interchangeable components and flexible parameterizations of style and body shape. Different helper operators (Sec. 3.6) support the construction process to reduce the workload.

2.3 Garment retargeting

The process of retargeting a garment from one body shape to another is usually performed manually, with leading industry tools like Clo3D [CLO Virtual Fashion 2022] providing support in storing (arbitrary) displacements, specified by designers for each vertex of garment panels individually, for each size. However, a number of research works have been exploring automatic solutions to this problem. Optimization-based approaches [Ait Mouhou et al. 2022; Brouet et al. 2012; Fondevilla et al. 2021; Lee and Ko 2018; Lee et al. 2013; Wang 2018] transform the garment geometry to reproduce design parameters such as fit, proportionality and overall design shape on a new body model, offering impressive results. In recent years, data-driven approaches for transferring garments across different shapes [Bertiche et al. 2020; Shi et al. 2021; Tiwari et al. 2020], or both shapes and poses [Corona et al. 2021; Lal Bhatnagar et al. 2019; Ma et al. 2020; Santesteban et al. 2021] have gained popularity. Most of these works represent garments as a displacement map over a body model, which helps disentangle design from body shapes, while [Corona et al. 2021] utilizes an implicit function as a more general approach to describe various garment styles. Unlike the approaches mentioned above, GarmentCode embeds the retargeting capability already at the sewing pattern modeling stage, which both reduces the need for manual editing and allows for controllable results reflecting the intention of the designer, which is not offered by the automatic methods mentioned above. The work of Wang et al. [2003] provides control over the design transfer optimization

process by allowing to relate design and body feature points in 3D. Some recent works [Jin et al. 2023; Wang et al. 2022] present case studies of one or two garment templates implemented with body shape parametrizations. Although they do not propose DSL-like tools to support the implementation of a general garment, these works demonstrate an interest in the fashion field in programmable sewing patterns. Works like [Umetani et al. 2011; Vidaurre et al. 2020; Wang et al. 2018; Yang et al. 2018] build their methods around parameterized base garments. Our work contributes towards a unified framework for creating such designs.

2.4 Procedural modeling and CAD DSL

Coding shapes like programs is not a novel idea. There are a number of programming languages developed for traditional solid CAD (Featurescript [Onshape 2023], OpenSCAD [Kintel, Marius and Wolf, Claire 2023]) with rich toolkits supporting shape definition (e.g., collections of standard shapes), editing (e.g., extrusion, boolean operations), and parameterization. Procedural modeling methods for buildings [Haegler et al. 2010; Müller et al. 2006; Schwarz and Müller 2015], city landscapes [Birsak et al. 2022; Parish and Muller 2001] and plants [Aono and Kunii 1984; Lane and Prusinkiewicz 2002; Lindenmayer 1968; Makowski et al. 2019; Oppenheimer 1986] provide tools (e.g., shape grammars) to code highly parametric generative models of the target objects in a variety of styles, with built-in editing options (e.g., varying the number of floors and windows in a building). Similar approaches emerge for furniture [Jones et al. 2020; Pearl et al. 2022], providing an interesting new angle on reconstruction problems. Due to the unique coupling of the 2D base representation with highly deformable behavior in 3D, garment engineering presents its own challenges and requires targeted modeling tools, but the research in this direction is limited. In addition to works on synthetic garment datasets described above, research on procedurally generated knitting instructions [Jones et al. 2022] computes machine-knittable patterns of given garment models. GarmentCode aims to fill the gap in the procedural generation of sewing pattern designs. We "translate" some of the tools of traditional CAD DSL to the garment domain, e.g., our edge loop definition for panels mirrors the structure of parametric boundary representations, and component copy operations are akin to linear and circular patterning in CAD. At the same time, we introduce component abstraction, tools for component stitching, and 2D-3D coupling, unique to garments.

3 ARCHITECTURE

3.1 Overview

Approaching sewing pattern modeling with a programming-based paradigm, especially when built upon the basis of existing general-purpose programming languages like Python, immediately provides a number of benefits, such as performing auxiliary computations (examples in Sec. 3.6.3 and Appendix A), free-form parametrization of geometry, and leveraging existing libraries built by the community. None of these benefits are available in existing design representations, be it visual [CLO Virtual Fashion 2022] or text-based parametric approaches [Korosteleva and Lee 2021]. However, specifying pattern geometry as a program without the support of structures and tools designed to handle garment-specific properties is tedious and
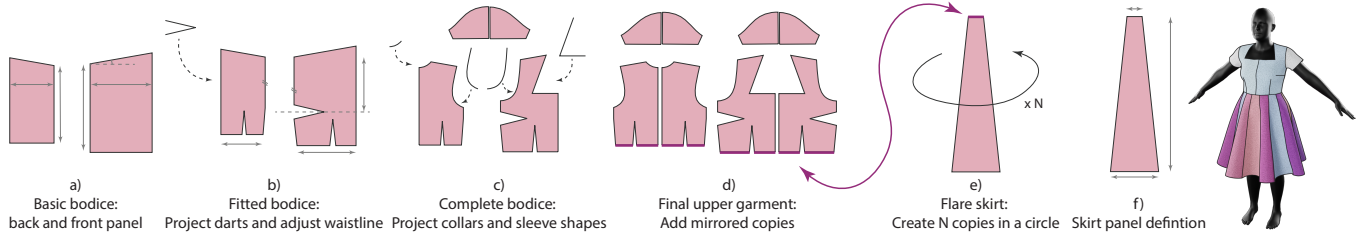
Fig. 2. Construction of a fitted bodice component and a skirt for 1950s dress style pattern. Dashed arrows denote projection operators; gray arrows show some of the body-related and stylistic parameters of presented components; thick lines on d) and e) show the interfaces of upper garment and skirt components.
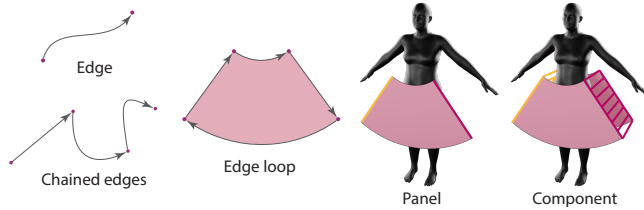

Fig. 3. Overview of the elements of a GarmentCode architecture. Highlighted edges on panels correspond to chosen interfaces. The body model serves as a positioning reference. Here and in the figures below we use SMPL female average body model [Loper et al. 2015], unless otherwise specified.

inefficient, akin to the dictionary-based specification of the existing text-based approach [Korosteleva and Lee 2021], requiring explicit definitions of each panel vertex and cross-referencing individual edges for stitch specification ("flat pattern representation"), with no support for element reuse beyond basic language capabilities.

Hence, the goal of GarmentCode is to provide a domain-specific language for specifying parametric sewing patterns and allowing easy reuse of defined garment elements to compose new garments as modular programs, enabling programming efficiency and complexity management. Specifically, we are bringing the principles of encapsulation and abstraction from the object-oriented programming (OOP) paradigm to garment construction. OOP has proven to be extremely efficient when it comes to building large complex systems across application areas, and we would like to leverage that efficiency when representing the complexity of garment design spaces. A *panel*, a stitched combination of panels, or a higher-level combination of components all define a garment *component* object, which encapsulates its particular geometry and only exposes an abstract semantic *interface*, implemented as a subset of edges of panels that comprise the component. Interfaces of two individual components can be connected together (*abstract stitch*) to form a higher-level component, and any components that implement the same set of interfaces (in our implementation, interfaces identified by the same names) can be used interchangeably regardless of the differences in their encapsulated geometry, enabling modular construction.

To summarize, the GarmentCode architecture allows pattern description through the following basic types: *Edge*, *EdgeSequence*, *Component*, *Panel* (as a special type of Component), and *Interface*, as illustrated in Fig. 3. Supporting the process, a variety of tools are implemented: a factory for typical edge sequences (e.g., dart shape), specification of curved edges with target properties, projecting an

open edge sequence on a corner or an edge, copy operators, normal evaluation for automatic right/wrong side definition, placing stitched components next to each other. To support downstream processing, GarmentCode also implements unfolding the abstract stitch definition into a flat pattern representation and serialization of patterns into files.

Below we describe the elements that comprise our architectural approach in detail.

3.2 Building blocks

3.2.1 Component. A component is an abstract class providing a framework to describe a compound garment or a garment element and holds some component processing methods (serialization, rotation, translation, mirroring, etc.). Any component should contain the following attributes:

- A set of subcomponents;
- Stitches – a list of stitching rules describing how the subcomponents should be connected (see Sec. 3.3).
- A set of interface objects that describe how other components can connect to this one (see Sec. 3.3.1).

Apart from specifying these attributes, a component construction process may contain instructions for modifying subcomponents, e.g., projecting an interface – a selection of edges from subcomponent's panels – of one component onto another, or smart copies, as demonstrated in Fig. 2.

3.2.2 Panel. A panel is a "leaf" component with special structure, so that it can act as a subcomponent, but also specify the panel geometry. Following the work of Korosteleva and Lee [2021], GarmentCode defines a panel as a closed piecewise smooth curve represented as a sequence of directed edges organized in a loop, as well as 6D placement parameters (rotation and translation). The latter is needed to correctly place the panel around the body but defaults to zero translation and rotation, and can be left to be set by higher-level components. These attributes define a panel component, in addition to the standard component attributes, namely interfaces and stitches. Note that a panel may contain stitches between its own edges, e.g., if the panel contains darts, as in the fitted bodice panel in Fig. 2.

3.2.3 Edges. An edge is an elementary building block of panels in a sewing pattern. Every edge describes an oriented curve segment. Specifically, GarmentCode supports straight line segments, circular arcs, and quadratic and cubic Bézier splines as edges. This set is

flexible and representative enough to model a variety of panel geometries, while ensuring smoothness and computational feasibility.

Edges are represented by their start and end vertices as attributes (the vertex coordinates are defined in 2D). Bézier curves additionally hold the coordinates of the their control points, while circular arcs store the signed distance of the midpoint of the arc to the straight line connecting the start and end vertices. Building upon the ideas of [Korosteleva and Lee 2021], all controls are specified in a local coordinate system of an edge: the straight segment connecting the edge endpoints is used as the unit horizontal axis, and the left perpendicular is the vertical axis. Such relative representation is invariant to edge translation and rotation, and preserves the curvature with uniform scaling, allowing to perform these operations on all types of edges only though vertex manipulation.

Working with edges is supported by a variety of routines. For simplicity of use, GarmentCode supports conversion of internal representations from and to standard ones: absolute control point coordinates for Bézier curves, and for circular arcs, the standard three-point representation or the desired radius with flags indicating one of the four arc options.

3.2.4 Edge sequences. An edge sequence specifies an ordered list of edges. An edge sequence used in panel definition must have all its edges chained one after another and into a loop, but other types of edge sequences might be used in other contexts, for example, in interfaces edges may not be chained together nor form a loop. To manipulate edge sequences conveniently, GarmentCode implements them as a variation of Python list type and thus supports indexing and slicing, appending, removals, and insertions, as well as domain-specific geometry manipulation methods: translation, rotation, scaling, and reflection around an arbitrary axis.

3.3 Stitches

The GarmentCode approach to stitch representation is one of the key elements that enable modular component construction. We aim to keep the stitching abstracted from the internal structure of individual components and rather reflect a semantic connection between high-level components. This allows substituting one component in the connection by another with the same semantic meaning despite differences in underlying geometry – enabling the interchangeability property. This behavior is realized by defining an abstract stitch as a connection between component-defined interfaces instead of a connection between panels' edges, as in flat sewing pattern representations. For example, in composite garments, the bottom of a bodice connects to the top of an under-waist garment. The same abstract stitch would describe a connection of a fitted bodice to a flared skirt, as in a 1950s dress, or a basic straight bodice to pants in a jumpsuit (see Fig. 7 for patterns of these examples).

3.3.1 Interface. An interface describes how and where a particular component can be connected with. An interface contains a collection of edges of panels that can connect to another component in an abstract stitch. An interface can be constructed directly as a subset of panel edges (usually in panel components), or from reusing or combining interfaces of subcomponents. Thus, a single interface can contain multiple edges from multiple different panels (in contrast

to 1-1 edge stitches in [Korosteleva and Lee 2021]). One component may have several interfaces.

In addition to stitches, interfaces may be useful for other purposes. For example, in sleeves, an interface specifies a projecting shape for correct modification of the bodice panel, which differs from the shape of the sleeve panel edges themselves. In Fig. 4, an Armhole is part of a Sleeve and defines the projecting shape, and Fig. 5 shows the differences and the projection result up close.

3.3.2 Stitching rule. In our implementation, an abstract stitch is specified simply as a pair of interfaces, wrapped in a stitching rule object. The wrapper encapsulates the processing of the stitch flattening (see below), performed at stitch declaration time.

3.3.3 Flattening stitch representation. While abstract stitches are convenient for modeling, downstream processing tasks like simulation usually require a flat representation of stitches as edge-to-edge connection instructions. Unpacking the hierarchy of interfaces is straightforward, however, oftentimes one or both interfaces participating in a stitch contain multiple edges. Breaking such a stitch down to edge-to-edge instructions requires additional processing. To perform this conversion, GarmentCode automatically generates additional vertices on the underlying panels to match the number of edges in the two connecting interfaces. Once the subdivision is completed, the set of resulting one-to-one stitches can then be used as a flattened representation. At the current stage of development, we use a simplifying assumption that each edge participates in no more than one stitch.

The process of generating the needed vertices is as follows. First, we note that the total length of edges in the interfaces on either side may not match, e.g., in stitches with gather, as in the Regency dress in Fig. 1. To accommodate for non-matching lengths of interfaces, the edge lengths are represented as fractions of the total lengths of an interface instead of being used directly. The fractions from one of the interfaces are projected onto another interface to generate additional vertices whenever the projections fall outside of existing ones with some tolerance, and then the process is repeated in the other direction. The edge sequences in the interfaces are assumed to be aligned in the expected connection order.

Our stitch flattening algorithm is rather straightforward, and similar algorithms are likely to be behind many-to-many stitch features in commercial visual CAD like CLO3D [CLO Virtual Fashion 2022].

3.4 Serialization

Serialization denotes a conversion of a GarmentCode hierarchical component into a flat sewing pattern representation that can then be passed on to downstream tasks such as cloth simulation. The process is fairly straightforward: GarmentCode recursively converts all panels involved in component construction into a text representation and then gathers them into one file, together with flattened stitching instructions. In this work, we serialize component instances into the open-source JSON file format introduced in [Korosteleva and Lee 2021], compatible with their draping pipeline.
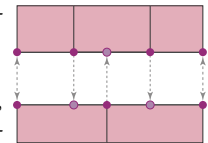
3.5 Parameterization format

While the opportunity for defining parameters emerges naturally from using a programming-based paradigm, GarementCode provides configuration formats to support this feature further. In GarmentCode, we propose separating body and style parameters into two sets, with body parameters containing the measurements of the current avatar, and style parameters specifying not only the particular values but also the possible value ranges and type (numerical, boolean, or categorical) for each parameter, to allow for design sampling.

Since some of the useful body measurements can be derived from others, GarmentCode provides an abstract class for loading body configuration files, with a separate method for the specification of formulas for derived parameters, which can be implemented in the application through a subclass. For example, in our prototype garment configurator, we use the waist level from the ground for positioning the garments, which is calculated from the body height, height of the head, and the usual waistline measurement from the nape of the neck to the waist. The calculation of derived design parameters, as well as the handling of parameter value conflicts, are usually component-specific and hence should be defined by the creators at the appropriate level.

3.6 Helpers

We propose severalroutines designed to simplify the construction of sewing patterns. It is worth noting that the presented algorithms, when needed, were chosen for their simplicity while providing reasonable quality, however, better or more efficient solutions are likely to exist. All calculations are performed at evaluation time.

3.6.1 Typical edge sequences. We implement shortcuts to create typical edge sequences, which include creating a loop of straight edges from a set of vertices, edge subdivision for all edge types (adding vertices inside an existing edge specified by relative distances), and creating a triangular dart shape, specified by desired dart width and depth.

3.6.2 Defining curve edges. In practice, curved edge design is often driven by certain requirements, rather than by the placement of control points. GarmentCode implements two optimization-based routines to specify Bézier curves:

- Specifying a quadratic Bézier curve with the highest point at a particular location, used e.g. in pencil skirts and pants to correctly hug the hips in a manner transferable across different bodies;
- Smoothly adjusting a cubic Bézier curve to match the desired tangent directions at the edge ends while preserving the curve lengths, used in defining inverted sleeve opening shapes for curve-based sleeves.

The second routine is described in Appendix A as part of the sleeve inversion algorithm.

3.6.3 Projection operators. Oftentimes when there is a need to connect two panels together, we wish the interface on one panel to be in

On an edge   On a corner
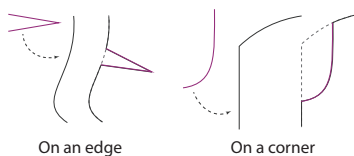
stylistic or functional correspondence with the geometry of the other, e.g. when connecting a sleeve panel with a bodice panel. Describing and maintaining such correspondence across independent panels and components would break the encapsulation principle, so instead GarmentCode offers projection operators to transfer the shape defined in a one-panel component onto another panel. The transferred shape may be part of a panel geometry, or be a *construction geometry* defined specifically for use in projection, implemented as an interface. Such a solution ensures that either of the panels can be easily modified or substituted, without the need to manually accommodate the changes in the second panel, supporting our target plug-and-play garment construction style. An example use of projections is demonstrated in Fig. 2.

GarmentCode supports projection on an edge (injection) and projection on a corner for an arbitrary open chained edge sequence as projection shape, and edge or pair of chained edges, correspondingly, as target shapes. Both types support the use of curves in projection shapes, as well as in the target edges, and hence are formulated as optimization problems.

In both cases, the goal is to find the points on the target that align with the "opening" (the first and the last vertices in the edge sequence) of projection shapes. The target edges are then split at the found points, and the projection shape is injected in between the found points into the target edge sequence, while the leftover "cuts" are removed. What differs between the two projection types is the process of finding the injection points. Projection on a corner relies on finding the points in the curve parameterization space (for straight edges and circular arcs we use arc length parameterization) whose 2D positions correspond to the projection shape opening:

$$\underset{t_1, t_2}{\arg\min} \|(e_1(t_1) - e_2(t_2)) - proj\_vec\|^2,$$

$$\text{s.t. } 0 \le t_1 \le 1, \ 0 \le t_2 \le 1, \quad (1)$$

where e_1, e_2 are the edges of the target corner, t_1, t_2 are values in curve parameterization space, and $proj\_vec = proj.end - proj.start$ is the vector describing the projecting shape opening.

Projection on edges also acts in curve parameterization space and finds two points that accommodate the projecting shape and are equidistant from the target point of injection $e(t)$:

$$\underset{t_1, t_2}{\arg\min}(\|e(t + t_1) - e(t - t_2)\| - \|proj\_vec\|)^2 +$$

$$+ (\|e(t + t_1) - e(t)\| - \|e(t - t_2) - e(t)\|)^2,$$

$$\text{s.t. } 0 \le t_1 \le 1, \ 0 \le t_2 \le 1, \quad (2)$$

where e is the target edge, t is the requested placement of the projecting shape t_1, t_2 are the shifts, all specified in respective curve parameterizations, and $proj\_vec = proj.end - proj.start$ is the vector describing projecting shape opening.

In case of projection on an edge, the projecting shape is automatically rotated such that $proj\_vec$ aligns with the estimated insertion vector $(e(t + t_1) - e(t - t_2))$, following the edge direction. The inserted shape may be reflected over the insertion vector following a user-specified parameter to appear on the other side. Rotation alignment in projection on the corner is left to be specified externally to ensure design flexibility: within certain limits, different rotations of
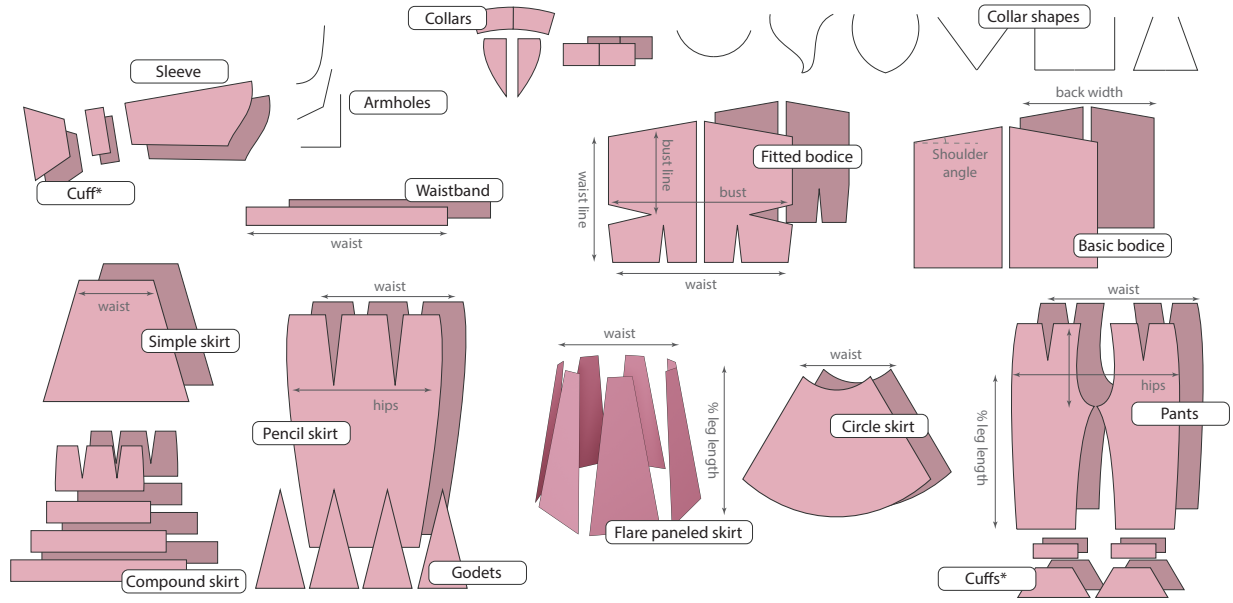
Fig. 4. Samples of garment components that we design using GarmentCode (stitches are not depicted for clarity). Each component's appearance is conditioned on style parameters and body measurements, some of which are illustrated with gray arrows. *Cuffs for sleeves and pants are the same component, shown twice to demonstrate potential use in both cases.*

the same shape projected on a corner would produce different but equally valid results.

3.6.4 Smart copy. The garment designs often exhibit reflection symmetry w.r.t. the sagittal plane (left-right symmetry), so describing only one of the halves often suffices. To support such a design shortcut, GarmentCode provides a mirroring operator, which reflects a component over a vertical line, including its shape and the location (used in our upper garment components, showcased throughout the paper). For other types of repetitive designs, GarmentCode provides a distribution of components copied along a line or a circle, an example use of which is demonstrated in Fig. 11.

3.6.5 Panel normal. The notion of the right and wrong sides of a fabric is represented as the direction of the panel normal (positive orientation corresponding to the "right", usually outwards-facing side). The normal direction is defined by the counterclockwise traversal of edges. GarmentCode can automatically update the edge loop traversal such that the right side of the fabric points outside of the body, under the assumption that a body model is aligned with the axes in the coordinate basis in which the panel's 6D placement is specified. To do so, we evaluate the position of the panel center-of-mass (COM), and for each edge determine whether it traverses the COM on the right or on the left using the cross product of the edge vector from its end to start vertex with the vector from COM to the edge start vertex. The normal is then the prevalent direction of those cross products among the panel edges. If the normal is not oriented outwards of the center of the body, the edge loop is reversed to flip the normal. Large curvature arcs in curve edges may interfere with this process, so we use their linear approximation. We find the extremal points of the curve (furthest away from the straight line

of an edge) and use them as new panel vertices, connecting them by straight edges.

In our implementation, the panel normal is adjusted upon every update to the panel's placement, ensuring correct normals throughout. However, updating the normals of all involved panels at the time of component serialization may also suffice.

3.6.6 Placement support. GarmentCode simplifies the task of correctly manipulating the component placement. First, all placement modifications performed at component level are automatically propagated to subcomponents while preserving their relative placement (that was set on the lower levels of the hierarchy). Second, GarmentCode provides a helper to adjust the translation of one component so that its chosen interface is aligned with another component's interface in 3D, which can be used to align components by their stitches. The helper simply evaluates the 3D centers-of-mass of the edge sequences described by the interfaces, and the modified component's translation is updated by the location difference on the two COMs. The modified component may additionally be shifted outward of the *component* COM to create a gap between two interfaces.

4 APPLICATION: GARMENT CONFIGURATOR

We apply the GarmentCode architecture described in Sec. 3 to build a garment configurator. The configurator allows selecting and assembling various high-level components and interactively manipulating their parameters, displaying the resulting sewing pattern. We design a collection of parametric garment components: various styles of skirts, such as flare, godet, pencil, gather, and compound, with the flare skirt implemented with two different topologies; bodice (i.e., components covering the torso, which can be fitted or loose), pants, sleeves (with optional cuffs), and different collar shapes. See Fig. 4.
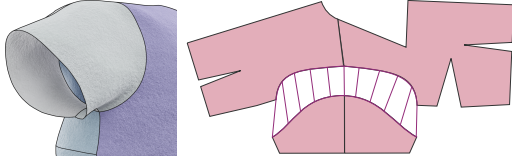
Fig. 5. Connecting a sleeve to a bodice. The edges on the sleeve side should create a concave shape compared to the sleeve opening shape on the bodice, so that the sleeve curves away from the body when stitched (see details in Appendix A). Our sleeve component defines a bodice opening shape as an additional interface for correct projection and connection.
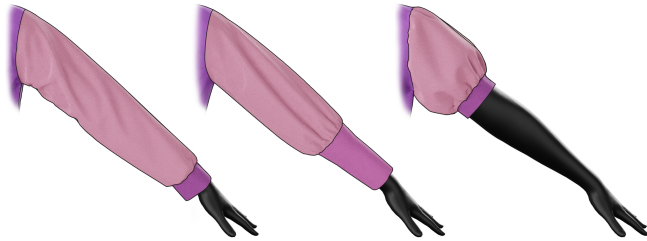

Fig. 6. Sleeve styles created from the same sleeve component by varying sleeve length, cuff length, and gather parameters

Many of these components can represent garments on their own, but an additional meta-component enables combining those elements into complex dresses (Fig. 1) and jumpsuits (Fig. 7). Components are parameterized w.r.t. body measurements (bust size, waist length, etc.), and style features (like lengths of elements). Most of the style parameters are defined to depend on body measurements or each other (e.g., collar width is bound between neck width and shoulder size), which additionally enables body retargeting and ensures pattern validity.

A selection of garments sampled from our parametric template is presented in Fig. 10. An example garment program is provided in Fig. 11. Flexible implementation of our GUI allows any garment programs with the recommended parameterization format to be loaded as GUI with little-to-no tweaks.

4.1 Example construction process

Here we describe an example process of constructing a dress in a 1950s style with GarmentCode. This dress requires a definition of a flare skirt, fitted bodice, collars, and sleeves. The process is illustrated in Fig. 2.

A fitted bodice is created to accommodate the natural body curvature and accentuate the waistline; darts are employed to create this effect. Since we assume that the body is left-right symmetric, we start building this piece by defining two quadrilateral panels representing one half of the front and the back of the bodice, following the body measurements (Fig. 2, a). The front panel is wider and longer to accommodate the extra curvature on the chest. We add darts (Fig. 2, b) by projecting a triangular shape onto the side of the front panel, such that the side with a dart is the same length as the back side, and the bottom of the front and back panel, and

removing some length from the side as well. The bottom length of both pieces equals half of the body waist measurement.

We combine the bodice component with other components to create an upper garment. First, we define a collar shape and project it onto the inner corners of the front and back panels. Secondly, we take a sleeve component and project a corresponding opening shape onto the outside corners of the front and back panels (Fig. 2, c; Fig. 5). Applying the projection operator makes the bodice design agnostic to the types of sleeves and collars used in this step, hence these components are easily replaceable with other designs. The next step is simply to mirror the upper garment component created so far, as described in Sec. 3.6.4. The bottom edges of the final four panels are designated as the interface of this upper component (Fig. 2, d).

We approximate a 1950s-style flare skirt with trapezoid panels replicated multiple times and distributed around the body (Fig. 2, e). The same shape can be achieved with two or even one panel – a section of a circle, but we show a more complex option to demonstrate the capabilities of GarmentCode. The tops of the panels should follow the body waist size, and the bottoms are wider, creating a flare effect. The sequence of the top edges of the skirt panels is designated to be a skirt interface. The final step is to connect the bottom of the upper garment with the top of the skirt, which is done automatically (see Sec. 3.3).

5 EVALUATION

5.1 Element parameterization

The power of parameterization in garment design can be well demonstrated on a sleeve example Fig. 6. The same building block can produce various sleeve silhouettes, from modern streetwear style to a vintage balloon sleeve, merely by varying a few parameters. The GarmentCode representation and parameterization open up an easy way for experimenting with designs based on human-readable parameters, rather than editing sewing patterns at the low level.

5.2 Handling complexity

The convenience of manipulating garments through hierarchical component structure is well-demonstrated through an example of a compound skirt (Fig. 8). Having the base skirts (pencil, flare, gather) and their corresponding parameter spaces defined, creating a compound skirt that uses existing components as layers is trivial: it requires an initialization of the base skirt (hugging the hips), and multiple copies of the skirt type used for the different levels, initializing their size based the bottom size of the one above, connecting their tops to bottoms of previous ones, and using the placing helper for correct alignment. With the base skirts implemented, extending our component library with the compound skirt component takes just 50 lines of code (see SKIRT_LEVELS.PY in supplementary code). Thanks to the component abstraction, it seamlessly integrates into the library: it can be used in place of other skirts or pants by simply adding its class name to the list of supported components in the meta-component parameter range for bottoms types.

5.3 Body retargeting

Our garment components are parameterized by body measurements, which makes it easy to fit a garment design on a different body.

Fig. 7. Retargeting garments conditioned on body measurements across different body shapes.
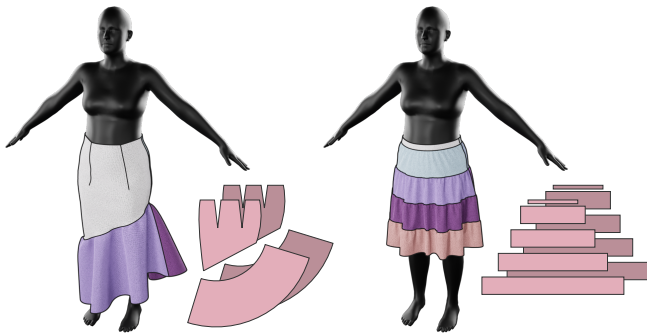

Fig. 8. Two design samples from a compound skirt component. With trivial implementation, this component allows combinations of existing skirts and their parametrizations, resulting in complex garment styles.

The parameterizations are introduced in the areas of garments that tightly hug the body, e.g., the waist of flare skirts and pants. The fitted bodice component is fully specified by body measurements (waist and bust circumferences, bust line, waistline, and back width) since its purpose is to accentuate the body curves. Some style parameters are made dependent on the body measurements, e.g., the length of a flare skirt is specified as a fraction of leg length and varies according to the wearer's height.

To demonstrate body retargeting, we take several body shape samples from SMPL [Loper et al. 2015] and manually acquire their body measurements. Fig. 7 shows the retargeting results. A 1950s-style dress recognizable hourglass silhouette relies on a proper fit of the tailored bodice component. The correct fit of the dress is fully preserved across large body shape variations thanks to the semantic encoding of GarmentCode components. The skirt length varies with the leg length, hence leaving approximately the same part of the leg uncovered in different body shapes. The tight-fitting pants in the jumpsuit and the pencil skirt in the strapless dress successfully adapt to different body shapes and proportions. The bottom of the pencil skirt is parameterized relative to the hip size, enabling the preservation of the defining upside-down triangular silhouette across all body models.

5.4 Reproducing a real-world pattern

To evaluate the patterns created with GarmentCode, we reproduce one of the professional garment patterns from Mood Fabrics [2020] by adjusting the parameters of the demo configurator, see Fig. 9. We observe that the overall 3D shape and design intention are well reproduced. However, a number of details vary. Our panel definition excludes inner loops such as diamond darts, resulting in an excessive stitch in the waist area. Without the support for fold lines (both in our system and in the downstream simulator), the sleeve panel needs to be defined as two. Since we currently do not support stitching
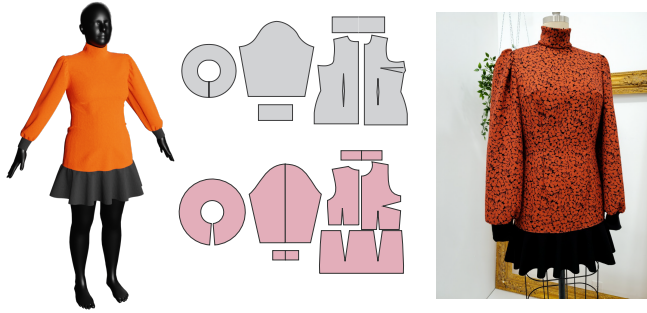
Fig. 9. Reproducing a production sewing pattern "Birch dress" of MoodFabrics [2020]. The MoodFabrics pattern is in grey (top) with the final garment on the right (photo provided by MoodFabrics), and ours is in pink (bottom) with the final garment on the left. Some discrepancies occur simply due to different design decisions and body sizes, while others highlight the limitations of the base GarmentCode architecture, as discussed in Sec. 5.4.

multiple layers of fabric, we can only model one-sided cuff and turtle neck components, whereas in the original pattern these are double-sided, with half of the panel folded inside.

Other differences are not dictated by the architectural limitations, but simply by the choices made when designing our example garment components. Our sleeve element uses smoother curves compared to the original pattern, resulting in a mismatch. The difference in lengths between the waist and the hip is distributed between the darts and the skirt sides differently, resulting in misaligned darts. Remaining variations (e.g. side dart width) are simply due to differences in body sizes between our body model and the standard sizing used in the original pattern. The showcased limitations provide inspiration for further development of the GarmentCode system.

6 DISCUSSION

We introduced a new framework for representing and designing parameterized garments. Our architecture encourages composing garments as hierarchical structures with interchangeable parametric components akin to configurable puzzle pieces. This approach enables exponential growth of design possibilities whenever a new component is added to the collection, expanding the design space, which can be easily explored through semantic parameters with little manual overhead, or sampled when constructing design datasets.

We demonstrated how our framework can be employed to create parametric garment design templates suitable for product configurators or in design samplers for synthetic garment datasets. Our templates offer extended design spaces, garment transfer across different body shapes, and produce valid sewing patterns for each instance, which can be passed on to a physics-based simulator or adapted for fabrication.

Our presented system changes the paradigm of garment construction to programming-based, which does not follow the traditional design workflow, and presents other challenges like the need for explicit specification of vertex coordinates in panels, which may be an obstacle for industry adoption. However, the successful cases of embracing programming in the creative domains, such as solid modeling CAD/DSL systems [Kintel, Marius and Wolf, Claire 2023; Onshape 2023], procedural tools for plans [Makowski et al. 2019],

buildings [Müller et al. 2006], or city landscapes [Parish and Muller 2001], and even such widespread fields as web design, give us reason to believe that fashion creators might be willing to acquire the needed programming skills to access unique features GarmenCode provides. Pairing programming-based parametric construction with a visual tool for specifying panel and edge geometry could be an interesting avenue for future work.

Creating a new design tool for garment construction is an ambitious and complex goal. GarmentCode aims to demonstrate the potential of our idea and provides a solid proof-of-concept implementation, but it is not all-encompassing. GarmentCode could be expanded with additional helpers to improve the toolkit: readjusting the edge shape after dart insertion for a smooth connection, adding a curved dart calculator, adding rotation alignment to the placement by stitches, etc. On the architecture level, the simplified definition of a panel does not allow specification of internal loops, hence GarmentCode cannot seamlessly represent panels with holes (Fig. 9). The architecture could also be extended to incorporate elements that are sewn on top of a fabric piece, such as pockets and flounces. Likewise, GarmentCode currently has limited support for sharp folds, and more tools are needed to efficiently specify and assemble pleats and smocking patterns. We also wish to further accommodate the differences between sewing patterns for garment fabrication vs. simulation: for example, merging excessively fragmented panels in the final pattern to reduce the number of stitches needed (e.g., removing the central stitch for front and back panels). Representing different stitch appearances in the spirit of [Rodríguez and Cirio 2022] is also an interesting direction to explore.

Finally, there is a considerable variation in sewing pattern geometry even within the basic garment elements, which is not fully represented in our implemented garment components, making it difficult to reproduce real garment designs merely by varying the semantic parameters of the demo configurator (Fig. 9). An additional engineering effort is required to accommodate such variations of real-world patterns.

GarmentCode achieves several considerable advancements through a simple architecture. It is evident that the problem of garment construction is under-explored, and we hope our work will inspire further research on computational support of this important engineering problem.

ACKNOWLEDGMENTS

Fig. 10. A selection of design samples from our parametric garment template. The segmentation corresponds to panels and stitches in the respective sewing patterns.

```
46  class ThinSkirtPanel(pyp.Panel):
47      """One panel of a panel skirt"""
48
49      def __init__(self, name, top_width=10, bottom_width=20, length=70) -> None:
50          super().__init__(name)
51
52          # define edge loop
53          self.flare = (bottom_width - top_width) / 2
54          self.edges = pyp.esf.from_verts(
55              [0,0], [self.flare, length], [self.flare + top_width, length], [self.flare * 2 + top_width, 0],
56              loop=True)
57
58          # w.r.t. top left point
59          self.set_pivot(self.edges[0].end)
60
61          self.interfaces = {
62              'right': pyp.Interface(self, self.edges[0]),
63              'top': pyp.Interface(self, self.edges[1]),
64              'left': pyp.Interface(self, self.edges[2])
65          }
66
```

```
310 v class SkirtManyPanels(pyp.Component):
311      """Round Skirt with many panels"""
312
313 v    def __init__(self, body, design) -> None:
314          super().__init__(f'{self.__class__.__name__}_{design["flare-skirt"]["n_panels"]["v"]}')
315
316          waist = body['waist']    # Fit to waist
317
318          design = design['flare-skirt']
319          n_panels = design['n_panels']['v']
320
321          # Length is dependent on length of legs
322          length = body['hips_line'] + design['length']['v'] * body['leg_length']
323
324          flare_coeff_pi = 1 + design['suns']['v'] * length * 2 * np.pi / waist
325
326          self.front = ThinSkirtPanel('front', panel_w:=waist / n_panels,
327                                       bottom_width=panel_w * flare_coeff_pi,
328                                       length=length )
329          self.front.translate_to([-waist / 4, body['waist_level'], 0])
330          # Align with a body
331          self.front.rotate_by(R.from_euler('XYZ', [0, -90, 0], degrees=True))
332          self.front.rotate_align([-waist / 4, 0, panel_w / 2])
333
334          # Create new panels
335          self.subs = pyp.ops.distribute_Y(self.front, n_panels, odd_copy_shift=15)
336
337          # Stitch new components
338 v        for i in range(1, n_panels):
339              self.stitching_rules.append((self.subs[i - 1].interfaces['left'], self.subs[i].interfaces['right']))
340
341          self.stitching_rules.append((self.subs[-1].interfaces['left'], self.subs[0].interfaces['right']))
342
343          # Define the interface
344 v        self.interfaces = {
345              'top': pyp.Interface.from_multiple(*[sub.interfaces['top'] for sub in self.subs])
346          }
347
```

Fig. 11. Example of a garment program written with GarmentCode, showcasing a skirt used in our 1950s dress example. The ThinSkirtPanel class defines a single trapezoid-shaped panel. SkirtManyPanels creates an instance of the panel according to style parameters (desired length, number of panels, and flare, where 1 sun = full circle skirt), and body measurements (waist to condition the top opening, and hips height to condition the length). The panel is then placed, and its copies are distributed on a circle around the body (using distribute_Y() operator) and stitched. The top edges of all panels constitute the interface of a final skirt.

REFERENCES

Abderrazzak Ait Mouhou, Abderrahim Saaidi, Majid Ben Yakhlef, and Khalid Abbad. 2022. 3D garment positioning using Hermite radial basis functions. *Virtual Reality* 26, 1 (2022), 295–322.

Amazon. 2023. Made for you custom t-shirt. https://www.amazon.com/Made-for-You-Custom-T-shirt/dp/B08HPZ3RHR.

Masaki Aono and Tosiyasu L. Kunii. 1984. Botanical Tree Image Generation. *IEEE Computer Graphics and Applications* 4, 5 (1984), 10–34. https://doi.org/10.1109/MCG.1984.276141

Seungbae Bang, Maria Korosteleva, and Sung-Hee Lee. 2021. Estimating Garment Patterns from Static Scan Data. *Computer Graphics Forum* 40, 6 (2021), 273–287. https://doi.org/10.1111/cgf.14272 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14272

Aric Bartle, Alla Sheffer, Vladimir G. Kim, Danny M. Kaufman, Nicholas Vining, and Floraine Berthouzoz. 2016. Physics-driven pattern adjustment for direct 3D garment editing. *ACM Transactions on Graphics* 35, 4 (2016), 50–1. https://doi.org/10.1145/2897824.2925896

Hugo Bertiche, Meysam Madadi, and Sergio Escalera. 2020. CLOTH3D: Clothed 3D Humans. In *European Conference on Computer Vision*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Virtual, 344–359. https://doi.org/10.1007/978-3-030-58565-5_21

Rachel Bick, Erika Halsey, and Christine C Ekenga. 2018. The global environmental injustice of fast fashion. *Environmental Health* 17, 1 (2018), 92.

Michael Birsak, Tom Kelly, Wamiq Para, and Peter Wonka. 2022. Large-Scale Auto-Regressive Modeling Of Street Networks. *arXiv preprint arXiv:2209.00281* (9 2022). http://arxiv.org/abs/2209.00281

Remi Brouet, Alla Sheffer, Laurence Boissieux, and Marie-Paule Cani. 2012. Design preserving garment transfer. *ACM Transactions on Graphics* 31, 4 (7 2012), 1–11. https://doi.org/10.1145/2185520.2185532

Xipeng Chen, Guangrun Wang, Dizhong Zhu, Xiaodan Liang, Philip H S Torr, and Liang Lin. 2022. Structure-Preserving 3D Garment Modeling with Neural Sewing Machines. In *Advances in Neural Information Processing Systems*.

Xiaowu Chen, Bin Zhou, Feixiang Lu, Lin Wang, Lang Bi, and Ping Tan. 2015. Garment modeling with a depth camera. *ACM Transactions on Graphics* 34, 6 (2015), 1–12. https://doi.org/10.1145/2816795.2818059

Pinaki Nath Chowdhury, Tuanfeng Wang, Duygu Ceylan, Yi-Zhe Song, and Yulia Gryaditskaya. 2022. Garment Ideation: Iterative View-Aware Sketch-Based Garment Modeling. In *International Conference on 3D Vision*.

CLO Virtual Fashion. 2022. *Clo3D.* https://clo3d.com/en/

Enric Corona, Albert Pumarola, Guillem Alenya, Gerard Pons-Moll, and Francesc Moreno-Noguer. 2021. SMPLicit: Topology-Aware Generative Model for Clothed People. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Virtual, 11875–11885.

Amelie Fondevilla, Damien Rohmer, Stefanie Hahmann, Adrien Bousseau, and Marie Paule Cani. 2021. Fashion Transfer: Dressing 3D Characters from Stylized Fashion Sketches. *Computer Graphics Forum* 40, 6 (2021), 466–483. https://doi.org/10.1111/cgf.14390

Simon Haegler, Peter Wonka, Stefan Müller Arisona, Luc Van Gool, and Pascal Müller. 2010. Grammar-based encoding of facades. *Computer Graphics Forum* 29, 4 (2010), 1479–1487. https://doi.org/10.1111/j.1467-8659.2010.01745.x

Nils Hasler, Bodo Rosenhahn, and Hans Peter Seidel. 2007. Reverse engineering garments. In *International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*. Springer, Berlin, Heidelberg, 200–211. https://doi.org/10.1007/978-3-540-71457-6_19

Zhu Heming, Cao Yu, Jin Hang, Chen Weikai, Du Dong, Wang Zhangye, Cui Shuguang, and Han Xiaoguang. 2020. Deep Fashion3D: A Dataset and Benchmark for 3D Garment Reconstruction from Single Images. In *The European Conference on Computer Vision (ECCV)*. Springer International Publishing, 512–530.

Moon-Hwan Jeong, Dong-Hoon Han, and Hyeong-Seok Ko. 2015. Garment capture from a photograph. *Computer Animation and Virtual Worlds* 26, 3-4 (2015), 291–300. https://doi.org/10.1002/cav.1653

Boyi Jiang, Juyong Zhang, Yang Hong, Jinhao Luo, Ligang Liu, and Hujun Bao. 2020. BCNet: Learning Body and Cloth Shape from a Single Image. In *Computer Vision – ECCV 2020*. Springer International Publishing, Cham, 18–35. https://doi.org/10.1007/978-3-030-58565-5_2

Peng Jin, Jintu Fan, Rong Zheng, Qing Chen, Le Liu, Runtian Jiang, and Hui Zhang. 2023. Design and Research of Automatic Garment-Pattern-Generation System Based on Parameterized Design. *Sustainability* 15, 2 (1 2023), 1268. https://doi.org/10.3390/su15021268

Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. 2022. Computational Design of Knit Templates. *ACM Transactions on Graphics* 41, 2, Article 16 (dec 2022), 16 pages. https://doi.org/10.1145/3488006

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis. *ACM Transactions on Graphics* 39, 6 (11 2020), 1–20. https://doi.org/10.1145/3414685.3417812

Kintel, Marius and Wolf, Claire. 2023. *OpenSCAD.* http://openscad.org/

Maria Korosteleva and Sung-Hee Lee. 2021. Generating Datasets of 3D Garments with Sewing Patterns. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1. Virtual.

Maria Korosteleva and Sung-Hee Lee. 2022. NeuralTailor: Reconstructing Sewing Pattern Structures from 3D Point Clouds of Garments. *ACM Transactions on Graphics* 41, 4 (2022). http://arxiv.org/abs/2201.13063

Bharat Lal Bhatnagar, Garvita Tiwari, Christian Theobalt, and Gerard Pons-Moll. 2019. Multi-Garment Net: Learning to Dress 3D People from Images. In *The IEEE International Conference on Computer Vision (ICCV)*. IEEE, Seoul, Republic of Korea, 5420–5430.

Brendan Lane and Przemyslaw Prusinkiewicz. 2002. Generating Spatial Distributions for Multilevel Models of Plant Communities. In *Graphics interface*. 69–87.

Norris Lee and MoodFabrics. 2020. THE BIRCH DRESS – FREE SEWING PATTERN. https://www.moodfabrics.com/blog/the-birch-dress-free-sewing-pattern/.

Wonseop Lee and Hyeong Seok Ko. 2018. Heuristic misfit reduction: A programmable approach for 3D garment fit customization. *Computers and Graphics (Pergamon)* 71 (4 2018), 1–13. https://doi.org/10.1016/j.cag.2017.10.004

Yongjoon Lee, Jaehwan Ma, and Sunghee Choi. 2013. Automatic pose-independent 3D garment fitting. *Computers and Graphics (Pergamon)* 37, 7 (2013), 911–922. https://doi.org/10.1016/j.cag.2013.07.005

Minchen Li, Alla Sheffer, Eitan Grinspun, and Nicholas Vining. 2018. FoldSketch: Enriching garments with physically reproducible folds. *ACM Transactions on Graphics* 37, 4 (8 2018), 1–13. https://doi.org/10.1145/3197517.3201310

Aristid Lindenmayer. 1968. Mathematical Models for Cellular Interactions in Development. *Journal of Theoretical Biology* 18, 3 (1968), 280–299.

Kaixuan Liu, Xianyi Zeng, Pascal Bruniaux, Xuyuan Tao, Xiaofeng Yao, Victoria Li, and Jianping Wang. 2018. 3D interactive garment pattern-making technology. *CAD Computer Aided Design* 104 (11 2018), 113–124. https://doi.org/10.1016/j.cad.2018.07.003

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-moll, and Michael J Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Transactions on Graphics* 34, 6 (2015), 1–16. https://doi.org/10.1145/2816795.2818013

Qianli Ma, Jinlong Yang, Anurag Ranjan, Sergi Pujades, Gerard Pons-Moll, Siyu Tang, and Michael J Black. 2020. Learning to Dress 3D People in Generative Clothing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Virtual, 6469–6478. https://cape.is.tue.mpg.de.

Miłosz Makowski, Torsten Hädrich, Jan Scheffczyk, Dominik L Michels, Sören Pirk, and Wojtek Pałubicki. 2019. Synthetic Silviculture: Multi-scale Modeling of Plant Ecosystems. *ACM Transactions on Graphics* 38, 4 (2019). https://doi.org/10.1145/3306346.3323039

Yuwei Meng, Charlie C.L. Wang, and Xiaogang Jin. 2012. Flexible shape control for automatic resizing of apparel products. *CAD Computer Aided Design* 44, 1 (2012), 68–76. https://doi.org/10.1016/j.cad.2010.11.008

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*. Association for Computing Machinery, Boston, Massachusetts, 614–623.

Onshape. 2023. *FeatureScript.* https://cad.onshape.com/FsDoc/

Peter E Oppenheimer. 1986. Real time design and animation of fractal plants and trees. *ACM SIGGRAPH Computer Graphics* 20, 4 (1986), 55–64. https://doi.org/10.1145/15886.15892

Yoav I H Parish and Pascal Muller. 2001. Procedural Modeling of Cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 301–308.

Ofek Pearl, Itai Lang, Yuhua Hu, Raymond A. Yeh, and Rana Hanocka. 2022. GeoCode: Interpretable Shape Programs. *arXiv* (12 2022). http://arxiv.org/abs/2212.11715

Nico Pietroni, Corentin Dumery, Raphael Guenot-Falque, Mark Liu, Teresa Vidal-Calleja, and Olga Sorkine-Hornung. 2022. Computational Pattern Making from 3D Garment Models. *ACM Transactions on Graphics* 41, 4 (2022). https://www.youtube.com/watch?v=7b27T8rh7SU

Alejandro Rodríguez and Gabriel Cirio. 2022. True Seams: Modeling Seams in Digital Garments. *ACM Transactions on Graphics* 41, 4 (7 2022). https://doi.org/10.1145/3528223.3530128

Igor Santesteban, Nils Thuerey, Miguel A Otaduy, and Dan Casas. 2021. Self-Supervised Collision Handling via Generative 3D Garment Models for Virtual Try-On. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 11763–11773.

Michael Schwarz and Pascal Müller. 2015. Advanced procedural modeling of architecture. *ACM Transactions on Graphics* 34, 4 (7 2015). https://doi.org/10.1145/2766956

Min Shi, Yukun Wei, Lan Chen, Dengming Zhu, Tianlu Mao, and Zhaoqi Wang. 2021. Learning a shared deformation space for efficient design-preserving garment transfer. *Graphical Models* 115, February (2021), 101106. https://doi.org/10.1016/j.gmod.2021.101106

Garvita Tiwari, Bharat Lal Bhatnagar, Tony Tung, and Gerard Pons-Moll. 2020. SIZER: A Dataset and Model for Parsing 3D Clothing and Learning Size Sensitive 3D Clothing. http://arxiv.org/abs/2007.11610

Nobuyuki Umetani, Danny M Kaufman, Takeo Igarashi, and Eitan Grinspun. 2011. Sensitive Couture for Interactive Garment Modeling and Editing. *ACM Transactions on Graphics* 30, 4 (2011), 90.

Unspun. 2023. Custom made jeans. https://www.unspun.io/.

Raquel Vidaurre, Igor Santesteban, Elena Garces, and Dan Casas. 2020. Fully Convolutional Graph Neural Networks for Parametric Virtual Try-On. *Computer Graphics Forum* 39, 8 (9 2020), 145–156. https://onlinelibrary.wiley.com/doi/full/10.1111/cgf.14109

Charlie C.L. Wang, Yu Wang, and Matthew M.F. Yuen. 2003. Feature based 3D garment design through 2D sketches. *CAD Computer Aided Design* 35, 7 (6 2003), 659–672. https://doi.org/10.1016/S0010-4485(02)00091-X

Huamin Wang. 2018. Rule-free sewing pattern adjustment with precision and efficiency. *ACM Transactions on Graphics* 37, 4 (2018), 1–13. https://doi.org/10.1145/3197517.3201320

Jin Wang, Guodong Lu, Weilong Li, Long Chen, and Yoshiyuki Sakaguti. 2009. Interactive 3D garment design with constrained contour curves and style curves. *CAD Computer Aided Design* 41, 9 (9 2009), 614–625. https://doi.org/10.1016/j.cad.2009.04.009

Tuanfeng Y. Wang, Duygu Ceylan, Jovan Popović, and Niloy J. Mitra. 2018. Learning a shared shape space for multimodal garment design. *ACM Transactions on Graphics* 37, 6 (12 2018), 1–13. https://doi.org/10.1145/3272127.3275074

Zhujun Wang, Xuyuan Tao, Xianyi Zeng, Yingmei Xing, Yanni Xu, Zhenzhen Xu, Pascal Bruniaux, and Jianping Wang. 2022. An Interactive Personalized Garment Design Recommendation System Using Intelligent Techniques. *Applied Sciences (Switzerland)* 12, 9 (5 2022). https://doi.org/10.3390/app12094654

Katja Wolff, Philipp Herholz, Verena Ziegler, Frauke Link, Nico Brügel, and Olga Sorkine-Hornung. 2023. Designing Personalized Garments with Body Movement. *Computer Graphics Forum* (2 2023). https://doi.org/10.1111/cgf.14728

Shan Yang, Zherong Pan, Tanya Amert, Ke Wang, Licheng Yu, Tamara Berg, and Ming C. Lin. 2018. Physics-inspired garment recovery from a single-view image. *ACM Transactions on Graphics* 37, 5 (11 2018), 1–14. https://doi.org/10.1145/3026479
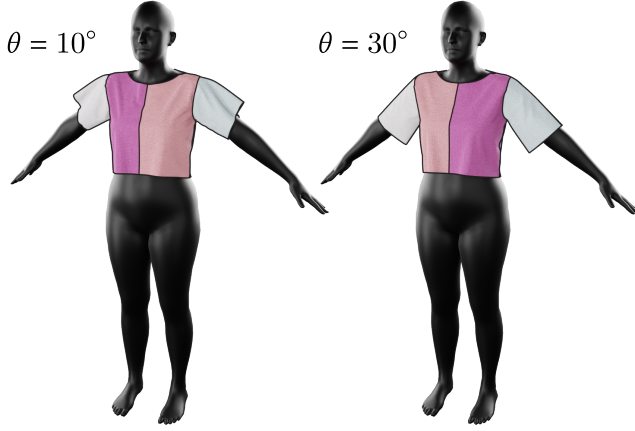
Fig. 12. T-shirts designed with different rest angles of sleeves (denoted θ) draped in the same pose. Note the differences in how well the fabric follows the arm angle.

A   CONSTRUCTING CURVED ELEMENTS

One of the advantages of the programming-based paradigm of garment modeling is the ability to utilize computational tools for defining garment elements that are difficult to specify correctly by hand, e.g., when it comes to manipulating smooth curves. Here we elaborate on one such example to complement the experiments presented in the paper.

A.1   Inverting sleeve opening

As shown in Fig. 5, the connection between the sleeve and a chosen bodice block is non-trivial, requiring the shape of the sleeve to be an "inverse" of the shape of the sleeve opening on the bodice in order to correctly wrap around the arm. In addition to these constraints, the shape of the inverse connection on the sleeve defines the rest angle of the sleeve in 3D: the arm angle at which there are neither folds nor tensions in the garment fabric. Smaller angles allow putting arms up more easily and are thus good for activewear, while bigger angles create fewer folds in the armpit zone when the arms are down, hence more suitable for officewear, as shown in Fig. 12.

For sleeve openings based on curves, defining sleeve edges correctly is especially challenging, since the inversion should preserve the length of the edge while following the desired rest angle and

maintaining smoothness of connection of sleeve panels. Garment-Code helps with this task.

Our process assumes that the projecting shape for a bodice is defined as a cubic Bézier curve with both control points on one side of the edge. The first step is then to create an initial guess for the inverted sleeve shape: the control point towards the end of the edge is flipped to the other side (flipping the y-coordinate in its relative representation), and the edge direction is aligned with an axis perpendicular to zero angle sleeve rest shape. In our implementation, the x-axis corresponds to a fully horizontal sleeve, and the perpendicular is the vertical direction. The edge is then rotated by a desired rest angle θ.

The second step is an optimization process, in which the edge extension and new positions of curvature control points are optimized s.t. the length of the curve is preserved, while the curve tangents at the endpoints are aligned with the downward direction at the top and the desired sleeve angle at the bottom. The first condition ensures a smooth connection at the top

between the front and back sleeve panels, while the second one enables the inversion effect and supports the chosen rest angle. The function to minimize is as follows:

$$E(c_1, c_2, s) = (\|e(c_{\text{start}}, c_1, c_2, c_{\text{end}} + s \cdot v)\| - l)^2 +$$
$$+ \left\|T_0(e(c_{\text{start}}, c_1, c_2, c_{\text{end}} + s \cdot v)) - T_0^*\right\|^2 +$$
$$+ \left\|T_1(e(c_{\text{start}}, c_1, c_2, c_{\text{end}} + s \cdot v)) - T_1^*\right\|^2 +$$
$$+ \lambda \left(C_{\max}(e(c_{\text{start}}, c_1, c_2, c_{\text{end}} + s \cdot v))\right)^2,$$

where c_1, c_2 are the cubic Bézier control points, $c_{\text{start}}, c_{\text{end}}$ are initial edge endpoints, $e(c_{\text{start}}, c_1, c_2, c_{\text{end}})$ is a curved edge with given endpoints and control points, s is the scaling factor of the edge vector $v = c_{\text{end}} - c_{\text{start}}$, the $T_0(\cdot)$, $T_1(\cdot)$ functions evaluate the curve tangent at the start and the end point of the edge curve, with T_0^*, T_1^* being the target tangent values as described above. Finally, $C_{\max}(\cdot)$ evaluates the maximum curvature of the edge and is used to regularize the curve smoothness.

The given process produces a correct sleeve inversion for sleeve openings of arbitrary size and for a desired sleeve rest angle, allowing us to define both as garment style parameters, as is done on our prototype garment configurator. The optimization process is included in the core GarmentCode as an operator.