The Shape Matching Element Method: Direct Animation of Curved Surface Models

TY TRUSTY, University of Toronto, Canada
HONGLIN CHEN, University of Toronto, Canada
DAVID I.W. LEVIN, University of Toronto, Canada

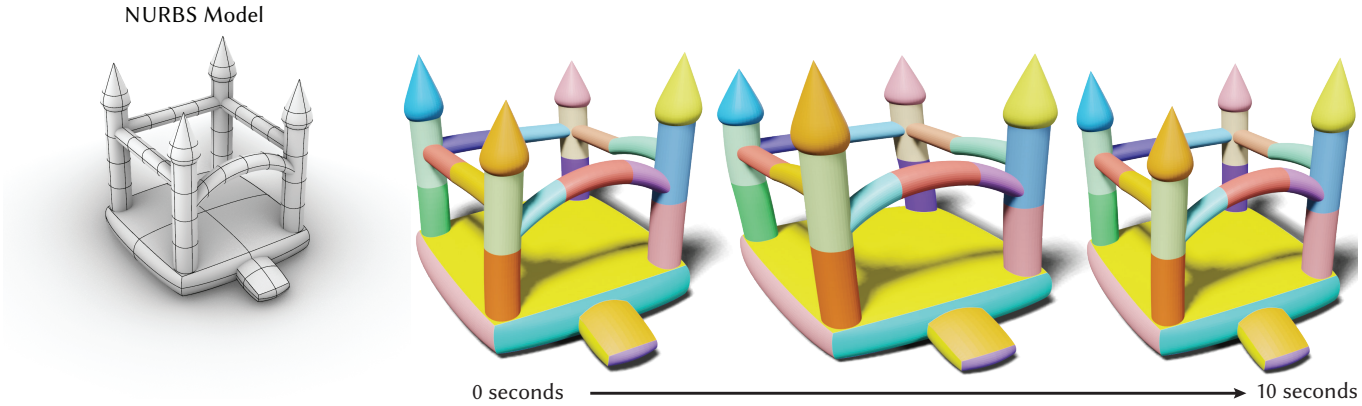

Fig. 1. Using the shape matching element method we can directly simulate this NURBS surface model of a bouncy castle as a volumetric elastic object without the need for volumetric meshing of any kind.

We introduce a new method for direct physics-based animation of volumetric curved models, represented using NURBS surfaces. Our technical contribution is the Shape Matching Element Method (SEM). SEM is a completely meshless algorithm, the first to simultaneously be robust to gaps and overlaps in geometry, be compatible with standard constitutive models and time integration schemes, support contact and frictional interactions and to preserve feature correspondence during simulation which enables editable simulated output. We demonstrate the efficacy of our algorithm by producing compelling physics-based animations from a variety of curved input models.

CCS Concepts: • **Computing methodologies** → **Physical simulation**;

Additional Key Words and Phrases: Physically-based animation, NURBS simulation, Meshless simulation, Lagrangian Mechanics

ACM Reference Format:
Ty Trusty, Honglin Chen, and David I.W. Levin. 2021. The Shape Matching Element Method: Direct Animation of Curved Surface Models. *ACM Trans. Graph.* 40, 4, Article 69 (August 2021), 14 pages. https://doi.org/10.1145/3450626.3459772

Authors' addresses: Ty Trusty, trusty@cs.toronto.edu, University of Toronto, 40 St. George Street, Toronto, ON, Canada, M5S 2E4; Honglin Chen, chl9797@cs.toronto.edu, University of Toronto, 40 St. George Street, Toronto, ON, Canada, M5S 2E4; David I.W. Levin, diwlevin@cs.toronto.edu, University of Toronto, 40 St. George Street, Toronto, ON, Canada, M5S 2E4.

0730-0301/2021/8-ART69 $15.00
https://doi.org/10.1145/3450626.3459772

1 INTRODUCTION

The consumption of geometric surface models by physics-based animation algorithms is fraught with difficulty. For volumetric objects, this process often involves identifying and discretizing the interior of the modelled object, typically either as a tetrahedral or hexahedral mesh. This procedure is both expensive and difficult, especially if the surface model is constructed from higher-order boundary representations, or if the volumetric discretization is required to be conforming or feature aligned. Removing explicit volumetric discretization from the physics-based-animation pipeline can avoid these difficulties and also provide a more unified modelling and simulation experience.

NURBS (Non-uniform Rational B-Splines) are a popular higher-order modelling primitive which are used for computer-aided design (CAD), computational fabrication and computer animation. NURBS primitives were the first geometric representation used for physics-based animation [Terzopoulos et al. 1987], yet, despite over three decades of research, animation of curved models, such as those built with NURBS, remains a challenge.

Isogeometric Analysis (IGA) is a physics simulation methodology that uses the control variables of the NURBS model as the degrees-of-freedom (DOFs) of the simulation itself. Unfortunately IGA approaches for volumetric objects still require background volumetric structures, typically regular grids that make satisfaction of boundary conditions difficult (which makes collision resolution hard) or more complicated cut-cell grids which introduce non-trivial root finding problems into the mix. Crucially, these simulation schemes typically assume models arise from engineering applications and meet tight geometric criteria such as that the mesh is manufacturable. These

inputs are much cleaner than those produced by a typical animation modeller.

We present the first truly meshless (no volumetric discretization required) algorithm for direct, nonlinear elastodynamic simulation of NURBs models. Our nonlinear elastodynamic simulation scheme requires only a boundary description of the object (we do not strictly require a solid model, see Fig. 1) and appropriate physical parameters. Because we explicitly use the NURBS boundary representation in the simulation, it is straightforward to handle Dirichlet and Neumann boundary conditions and to apply contact resolution. Crucially, because we broadly target animation and not necessarily simulation for engineering or manufacturing, we don't require that models satisfy the rigorous geometric requirements common for these applications.

Our approach draws inspiration from the recently developed Virtual Element Method (VEM) for solving partial differential equations on domains tiled with arbitrary polygons. We establish a connection between VEM and the well-known shape matching simulation algorithm [Diziol et al. 2011; Müller et al. 2005]. This allows us to derive equations of motion for an arbitrary curved surface model, made of NURBS, via Lagrangian Mechanics. Importantly, we show how to replace volumetric data structures for integration and blending weight computation with ray casting approaches which enables our meshless approach to elastodynamic physically-based animation.

2 RELATED WORK

Geometric modeling is a necessary precursor to physics-based animation, however connecting differing geometric representations for modelling and simulation often requires time-consuming, complex geometry processing pipelines. For instance, the popular tetrahedral finite element approach for simulating solid elastodynamics [Sifakis and Barbic 2012] requires robust algorithms for converting input surface geometry into a volumetric tetrahedral mesh. This is a difficult problem and while significant progress has been made, even the most robust volumetric methods [Hu et al. 2018] can be time consuming, fail to maintain correspondence between the input model and output simulation mesh, and don't work directly on curved surface representations such as NURBS.

For many physics-based animation tasks, it would be desirable to bypass volumetric meshing entirely and directly simulate the geometric model. An ideal approach would avoid meshing of any kind (no volumetric meshes or cut-cells), support continuum mechanics-type constitutive models and energies that have become standard in physics-based animation pipelines, be compatible with a wide range of time integration schemes and ensure that simulation output can be edited in the same modelling software used to create the input (important for post-processing). Finally, our method should put only moderate constraints on input model quality to facilitate ease-of-use.

Isogeometric Analysis [Cottrell et al. 2009] endeavors to perform simulation directly on the NURBS output from Computer-Aided Design (CAD) software. Initial attempts used NURBS surfaces to represent the medial surface of thin objects [Terzopoulos and Qin 1994]. Volumetric simulations relied on volumetric NURBS [Aigner et al. 2009] but were limited to a narrow class of geometries. Finite volume methods are applicable to more general geometries [Heinrich et al. 2012; Sevilla et al. 2008] but require a volumetric mesh to be generated. Modern approaches are constructed around the extended finite element method which enriches the standard finite element basis with discontinuous basis functions to improve boundary handling [Haasemann et al. 2011; Hafner et al. 2019; Legrain 2013; Safdari et al. 2015, 2016]. These methods typically start with an easy-to-generate structured volumetric mesh (tetrahedral or hexahedral), "cutting" the NURBS geometric model against it to enable boundary handling (such a mesh is called a *cut-cell* mesh). Like volumetric meshing, this cutting operation can be difficult and our ideal method would avoid it. Some cut-cell algorithms assume engineering/manufacturing quality input, which puts tight requirements on input models [Hafner et al. 2019]. Finally these approaches require additional mechanisms to ensure that simulation results lie inside the shape space of the input model's primitives, which increases the complexity.

Embedded Methods attempt to sidestep many of these issues by enclosing complex surface geometry inside a simulation coarse mesh [Muller et al. 2004]. However, producing an appropriate mesh embedding introduces a number of challenges since the coarse mesh's connectivity must mirror that of the embedded surface. This necessitates the use of complex hierarchical methods like that of Nesme et al. [2009], which themselves require the user to correctly intuit an appropriate ultimate grid resolution. Alternately one can introduce cut cell [Tao et al. 2019] or extended Finite Element [Hafner et al. 2019] approaches which bring with them complicated geometric operations. Furthermore when the DOFs do not lie on the boundary, methods such as Nitsche's Method [Nitsche 1971] are required for handling Dirichlet boundary conditions. Our method inherits shape connectivity directly from the boundary input, avoiding these difficulties entirely.

Shape Matching is a meshless approach to physics-based animation [Diziol et al. 2011; Müller et al. 2005] and geometry processing [Bouaziz et al. 2012] built around shape registration. The algorithm has been extended from volumetric triangle mesh input to cloth [Stumpp et al. 2008], to particles [Müller and Chentanez 2011] and even to visual geometry for video games [Müller et al. 2016]. Shape Matching is fast [Rivers and James 2007; Steinemann et al. 2008] and meshless, but state-of-the-art methods require additional modelling input to position simulation primitives [Müller and Chentanez 2011] or limit simulation primitives to be collections of convex polytopes [Müller et al. 2016]. Finally, Shape Matching is tightly coupled to the position-based dynamics (PBD) [Müller et al. 2007] time integration methodology. While incredibly performant, this approach is incompatible with the constitutive models that are popular for physics-based animation, as well as other time integration schemes. The popular Projective Dynamics algorithm [Bouaziz et al. 2014] enables a more flexible Shape Matching implementation but is still limited to a subset of constitutive models for elastic solids.

To alleviate these restrictions we turn to other meshless methods popular in computer graphics and engineering [Faure et al. 2011; Gilles et al. 2011; Liu et al. 1995; Martin et al. 2010; Müller et al. 2004]. These methods support more advanced constitutive models and integration schemes but often require background integration meshes, limit themselves to low order deformation functions and

lose the direct connection with modelling geometry. Like cut-cell-based, Isogeometric Analysis approaches, additional constraints must be added to ensure that simulation output can be represented by the input model. Given this, we conclude that there is no existing algorithm that meets our desiderata for success.

Our algorithm takes the Shape Matching approach as inspiration, but rather than follow the PBD formalism, we interpret Shape Matching as a Virtual Element Method (VEM) [Beirão da Veiga et al. 2014; Veiga et al. 2012]. Virtual Elements are an extension of mimetic finite differences [Brezzi et al. 2005; Lipnikov et al. 2014] to weak-form variational problems. VEM relaxes the mesh generation requirements of the finite element method by enabling the solution of partial differential equations on domains partitioned with arbitrary polytopes [De Goes et al. 2020]. The solution inside each polytope is approximated using a polynomial function of a specified order. VEM typically assumes that the boundary of the the problem domain is described by a piecewise linear complex which makes its standard formulation incompatible with our curved input geometry.

Contributions

In this paper we develop a new Shape Matching-based, Virtual Element Method which is directly compatible with NURBS input geometry rather than piecewise linear surfaces. Furthermore, we improve the expressivity of the VEM basis by using a shape blending approach inspired by algorithms for skinning [Jacobson et al. 2014a]. Our method is entirely meshless (requiring no volumetric meshes or cut-cells) and is compatible with standard constitutive models and time integrators. It guarantees that simulation output is directly consumable by the input modelling software and can ingest models which include large gaps, intersections and disconnected primitives without additional user input. These features mean that our algorithm is the first truly meshless approach for the direct simulation of curved surface, NURBS models in physics-based animation.

3 OVERVIEW

The input to the Shape Matching Element Method (SEM) is a NURBS *boundary representation* of a volumetric object, along with a set of physical parameters (density, constitutive model, model parameters, etc.). The boundary representation is composed of one or more (not necessarily explicitly connected) NURBS surface primitives that we will call *parts*. The output is an elastodynamic simulation. SEM consists of preprocessing and runtime simulation phases (Alg. 1, Alg. 2) and at no point do we need to generate a volumetric mesh of the input. In the *preprocessing* stage we use raycasting to find quadrature points and weights for volumetric integration, as well as to compute part blending parameters. Finally we construct local shape matching operators and the mass matrix for our problem.

At runtime we use standard time integration schemes to time step the system. This requires us to evaluate the energy, gradient and Hessian (implicit only) which we do at the previously computed quadrature points. For collisions we use penalty forces and apply them as external forces during integration (though other methods could be used).

Algorithm 1 Shape Matching Element Preprocessing

procedure PREPROCESS(*model*)
  // Get initial DOF and sample boundary ▷ Section 4
  $\mathbf{q} \leftarrow$ GetControlPoints(*model*)
  $\hat{J}, S \leftarrow$ SampleNURBS(*model*) ▷ Section 5
  $\mathbf{x^0} \leftarrow S\hat{J}\mathbf{q}$
  // Generate material points inside the model
  $\mathcal{X} \leftarrow$ RaycastQuadrature(*model*) ▷ Section 6.3
  // Set of blending weights for each $\mathbf{X} \in \mathcal{X}$
  $\mathcal{W} \leftarrow$ BlendingWeights($model, \mathcal{X}, \alpha$) ▷ Section 6.1
  $\bar{\mathcal{X}} \leftarrow$ DeformationOrigins($\mathcal{X}, w$) ▷ Section 6.2
  // Create projection operator
  $\Pi \leftarrow$ ShapeMatching($\mathbf{q}, \hat{J}, S, \bar{\mathcal{X}}$) ▷ Section 5
  // Form mass matrix and error hessian, respectively
  $M \leftarrow$ MassMatrix($\mathcal{X}, \bar{\mathcal{X}}, \mathcal{W}, \Pi$) ▷ Section 7.2
  $M_\epsilon \leftarrow$ ErrorMatrix($\mathbf{x^0}, \bar{\mathcal{X}}, \mathcal{W}, \Pi$) ▷ Section 7.6
end procedure

4 GEOMETRIC MODEL

Our algorithm acts on objects composed of multiple NURBS surfaces (Fig. 2). The three-dimensional position, $\mathbf{x}$, of any point on a

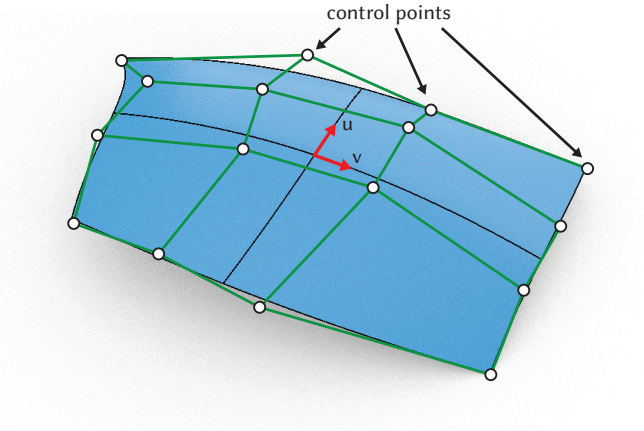

Fig. 2. A cubic NURBS patch with 16 control points.

NURBS surface can be written as

$$\mathbf{x}(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{m} \phi_{ij}(u,v)\,\mathbf{q}_{ij}, \tag{1}$$

where u and $v \in \mathbb{R}$ are the coordinates in the 2D parametric domain, n and m are the number of control points in the u and v directions, $\mathbf{q}_{ij} \in \mathbb{R}^3$ are the control points and $\phi_{ij}(u,v) \in \mathbb{R}$ are the NURBS

basis functions given by

$$\phi_{ij}(u,v) = \frac{\omega_{i,j}\beta_{ik}(u)\beta_{jl}(v)}{\sum_{r=1}^{n}\sum_{s=1}^{m}\omega_{r,s}\beta_{rk}(u)\beta_{sl}(v)},$$

where β_{ik} (resp. β_{jl}) is the i^{th} (j^{th}) B-spline basis of order k (l) and $\omega_{i,j}$ is the NURBS weight.

Exploiting linearity with respect to the control points, we can express this mapping as

$$\mathbf{x}(u,v) = \underbrace{\begin{pmatrix}\phi_{11}(u,v)\,I & \dots & \phi_{nm}(u,v)\,I\end{pmatrix}}_{J(u,v)} \underbrace{\begin{pmatrix}\mathbf{q}_{11}\\ \vdots \\ \mathbf{q}_{nm}\end{pmatrix}}_{\mathbf{q}}, \tag{2}$$

where $J(u,v)$ is the NURBS Jacobian, I is the 3×3 identity matrix and $\mathbf{q}$ is the vector of generalized coordinates [Lanczos 2012] The Shape Matching Element Method will follow from this kinematic description.

5 SHAPE MATCHING

We begin by describing shape matching to a single NURBS part. Given two different configurations of the same part (specified by the control points), shape matching computes a polynomial deformation that best aligns them.

Let $\mathbf{q^0}$ be the initial control points of the part, provided by the modeler, and $\mathbf{q}$ be a vector of modified control point values. We can compute the undeformed (or reference) position of any point on our part using $\mathbf{X}(u,v) = J(u,v)\,\mathbf{q^0}$ and the corresponding deformed (world space) position as $\mathbf{x}(u,v) = J(u,v)\,\mathbf{q}$.

We can define a polynomial deformation from $\mathbf{X} = \begin{pmatrix}X & Y & Z\end{pmatrix}^T$ to $\mathbf{x}$ as

$$\mathbf{x}(\mathbf{X}) = \underbrace{\begin{bmatrix}\mathbf{p}(\mathbf{X}-\bar{\mathbf{X}})^T & \mathbf{0} & \mathbf{0} & 1 & 0 & 0\\ \mathbf{0} & \mathbf{p}(\mathbf{X}-\bar{\mathbf{X}})^T & \mathbf{0} & 0 & 1 & 0\\ \mathbf{0} & \mathbf{0} & \mathbf{p}(\mathbf{X}-\bar{\mathbf{X}})^T & 0 & 0 & 1\end{bmatrix}}_{P(\mathbf{X})} \begin{bmatrix}\mathbf{c}\\ \mathbf{t}\end{bmatrix}, \tag{3}$$

where $\mathbf{p}(\mathbf{X})^T$ is the polynomial basis vector (e.g., $\begin{bmatrix}X & Y & Z & X\cdot Y & X\cdot Z & Y\cdot Z & X\cdot X & Y\cdot Y & Z\cdot Z\end{bmatrix}$), $\bar{\mathbf{X}}$ is an a priori chosen origin around which to deform, $\mathbf{c}$ are the coefficients for the non-constant part of the polynomial and $\mathbf{t}$ is the translation of the deformation origin. We can compute the shape matching deformation by solving [Müller et al. 2005]

$$\begin{bmatrix}\mathbf{c}\\ \mathbf{t}\end{bmatrix}^* = \arg\min_{\mathbf{c},\mathbf{t}} \int \left\| P(\mathbf{X}) \begin{bmatrix}\mathbf{c}\\ \mathbf{t}\end{bmatrix} - J\mathbf{q}\right\|_2^2 dudv, \tag{4}$$

where we have suppressed the dependence of $\mathbf{X}$ and J on u and v for the sake of brevity.

We discretize the shape matching energy and minimize by solving

$$\hat{P}^T S\hat{P}\begin{bmatrix}\mathbf{c}\\ \mathbf{t}\end{bmatrix} = \hat{P}^T S\hat{J}\mathbf{q}. \tag{5}$$

Here $\hat{P}$ (resp. $\hat{J}$) is a $3s\times 3(k+1)$ (resp. $3s\times 3nm$) matrix where s is the number of quadrature points, k is the number of terms in the shape matching polynomial and nm is the number of NURBS control points. $\hat{P}$ (resp. $\hat{J}$) is constructed by stacking the evaluations of $P(\mathbf{X}(u,v))$ (resp. $J(u,v)$) at each quadrature point. S is a diagonal, $3s\times 3s$ matrix of integration weights.

Quadrature points are selected so that $\hat{J}$ is of sufficient rank required to represent all control points. We achieve this by sampling over each knot span defined by the B-splines, and find it effective to use the order of the NURBS as the number of quadrature points sampled in each knot span. Positions and weights of the quadrature points in each span are chosen using Gauss-Lobatto quadrature.

Relationship to the Virtual Element Method. For a non-planar NURBS part, a sufficient number of quadrature points will make Eq. 5 well-posed, allowing us to write: $\begin{bmatrix}\mathbf{c}\\ \mathbf{t}\end{bmatrix}^* = \Pi\mathbf{q}$ where

$$\Pi = \left(\hat{P}^T S\hat{P}\right)^{-1}\hat{P}^T S\hat{J},$$

is a projection from the space of control points onto the space of polynomials. This projection serves an identical purpose (and is mathematically equivalent up to choice of metric) to the projection operator used in VEM [Beirão da Veiga et al. 2014]. VEM uses a single polynomial projection per polyhedral element to solve differential equations on complex volumetric meshes. We interpret shape matching as converting our NURBS part into a single Virtual Element suitable for elastodynamics simulation. Unfortunately, a single polynomial deformation space will not be suitably expressive for even moderately complex models. Previous work [Müller et al. 2005; Rivers and James 2007] used additional lattices or partitions of the reference space to increase the number of polynomials in the shape matching. In contrast, our meshless Shape Matching Element Method works directly on the NURBS geometry itself.

6 SHAPE MATCHING ELEMENT METHOD

Each input object is composed of a set of boundary representations, $\mathcal{B}$, one for each part $\mathcal{b}\in\mathcal{B}$. Our kinematic mapping is a blended construction [Andrews et al. 2016; Jacobson et al. 2014b] which we write as

$$\mathbf{x}(\mathbf{X}) = \sum_{\mathcal{b}_i\in\mathcal{B}} w_i(\mathbf{X})\,P_i(\mathbf{X})\,\Pi_i\mathbf{q}_i, \tag{6}$$

where $w_i(\mathbf{X})$, $P_i(\mathbf{X})$, Π_i and $\mathbf{q}_i$ are the blending weights, polynomial basis, projection operator and control points for the i^{th} part respectively.

Multiple Part Projection

As in §5 we construct each Π_i via shape matching. It is tempting to use an identical polynomial basis matrix, $P(\mathbf{X})$ for all parts, $\mathcal{b}_i$, but recall that constructing $P(\mathbf{X})$ requires the selection of a deformation origin $\bar{\mathbf{X}}$ (Eq. 3). Using the same origin for all polynomials limits the expressivity of the kinematic model [Jacobson and Sorkine 2011]. Instead we choose a set of points $\bar{\mathcal{X}}$ ($0 < |\bar{\mathcal{X}}| < |\mathcal{B}|$) to act as deformation origins (§6.2). For each part we now construct $P_i(\mathbf{X})$ using its associated deformation origin $\bar{\mathbf{X}}_j\in\bar{\mathcal{X}}$, where each origin can be shared between multiple parts.

Attempting to use $P_i(\mathbf{X})$ in Eq. 5 on a per-part basis is problematic as it can become ill-posed, especially for planar boundary representations. To alleviate this problem we invoke the *hierarchical ordering*

principle, which argues that the behavior of a system is typically dominated by lower order effects [Li et al. 2006]. Mechanically, this principle suggests that the amount of motion modelled by increasingly high-order polynomials is, itself, decreasing. This implies that we can achieve acceptable shape matching results by performing shape matching hierarchically – starting with the constant term and then fitting the remaining deformation with progressively higher-order expressions.

For a given part, $\mathcal{b}_i$, with associated origin $\bar{\mathbf{X}}_j$ we first estimate the constant term, $\mathbf{t}_i$ as the centroid of all parts associated with $\bar{\mathbf{X}}_j$. Next we fit the linear part of the deformation by performing linear shape matching to $\mathcal{b}_i$. This can be repeated for polynomials of arbitrary high order, for instance here we apply this approach to quadratic shape matching:

$$
\begin{aligned}
\mathbf{t}_i &= \sum_{\mathcal{b}_k \in \bar{\mathbf{X}}_j} \underbrace{\frac{1}{\sigma_k} \mathbb{1}^T S_k \hat{J}_k}_{B_{ik}} \mathbf{q}_k \\
\mathbf{c}_i^1 &= \underbrace{\left(\hat{P^1}^T S_i \hat{P^1}\right)}_{A_i^1}{}^{-1} \left(\underbrace{\hat{P^1}^T S_i \hat{J}_i}_{B_i^1} \mathbf{q}_i - \underbrace{\hat{P^1}^T S_i \mathbb{1}}_{T_i^1} \mathbf{t}_i \right) \\
\mathbf{c}_i^2 &= \underbrace{\left(\hat{P^2}^T S_i \hat{P^2}\right)}_{A_i^2}{}^{-1} \left(\underbrace{\hat{P^2}^T S_i \hat{J}_i}_{B_i^2} \mathbf{q}_i - \underbrace{\hat{P^2}^T S_i \hat{P^1}}_{D_i^2} \mathbf{c}_i^1 - \underbrace{\hat{P^2}^T S_i \mathbb{1}}_{T_i^2} \mathbf{t}_i \right),
\end{aligned}
\tag{7}
$$

where $\sigma_k = \sum_{\mathcal{b}_k \in \bar{\mathbf{X}}_j} \mathrm{Tr}\, S_k$ is the normalization constant for the k^{th} part, P^l contains only the monomial basis of order l, $\hat{P^l}$ is the stacked evaluation of this matrix at the s quadrature points and $\mathbf{c}_i^l$ are the corresponding coefficients. Finally, S_i is the diagonal integration weight matrix and $\mathbb{1} \in \mathbb{R}^{3s \times 3}$ is a matrix of stacked 3×3 identity matrices – one for each quadrature point.

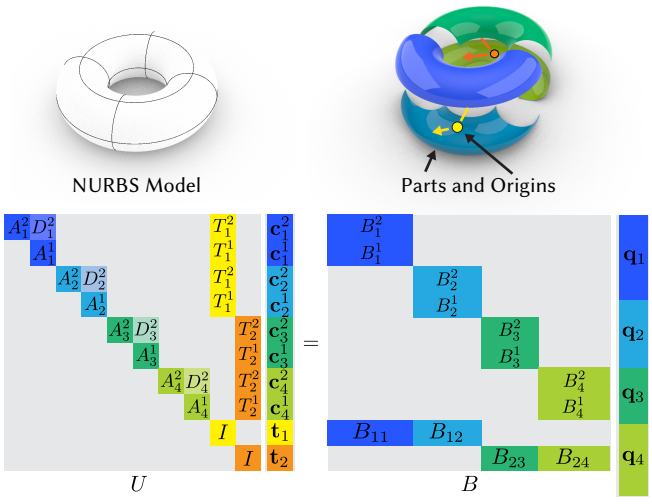

Fig. 3. A simple, four part model with two deformation origins and the corresponding matrix equation for the quadratic coefficients.

When assembled for all parts, Eq. 7, yields a block upper triangular matrix U and a sparse matrix B (Fig. 3) which can be used to efficiently compute the projection operator for the entire object $\Pi = U^{-1}B$. The structure of U and B ensures that Π only couples objects which share deformation centers, which implies a sparse Π. The per-part projection operators Π_i correspond to blocks of rows of Π. Coupling via the deformation origins ensures our method will reproduce rigid and linear deformations in regions around these points, while higher order terms help compensate for more local deformations.

Note that hierarchical shape matching is not guaranteed to find the optimal polynomial fit and potentially risks overfitting to low-order terms. However we find that it provides qualitatively reasonable results and avoids much of the dense matrix arithmetic and singular value decompositions of alternative methods such as the Moore-Penrose inverse or QR factorization.

6.1 Meshless Blending Weight Computation

Our kinematic map (Eq. 6) requires blending weights in order to homogenize our per-part representation. Specifying weights manually would be antithetical to our goal of providing a seamless translation from surface representation to physics-based animation. Automatic weight computation is well-studied (see [Wang et al. 2015], [Jacobson et al. 2011] and [Faure et al. 2011]), but typically requires solving an optimization problem that itself relies on a volumetric discretization. This can be avoided by using stochastic methods [Sawhney and Crane 2020] but these do not yet support the boundary conditions we require for our problem.

Ideally our shape functions would both partition unity and obey the Kronecker delta property (the i^{th} blending function is one at i^{th} part and zero on all other parts). These properties ensure that our simulation can properly represent translation and that the solution stored at each part is the actual deformed position. Additionally we would like our weights to be sparse both for performance reasons (leads to sparse matrices) and for expressivity (leads to locality of deformation).

While partition of unity and sparsity are straightforward to enforce, the overlapping and intersecting parts (Fig. 17) often encountered in non-engineering models leads to a contradiction — how can the blending functions be both one and zero at the points of overlap? Our solution is to allow weights to include bounded discontinuities at these points and it is these discontinuities that are difficult to enforce with previous methods.

Given a set of reference space points $\mathcal{X}$ inside our model, we want to evaluate $w_i\left(\mathbf{X}_j\right)$, where w_i is the weight function associated with the i^{th} part, $\mathcal{b}_i$, and $\mathbf{X}_j \in \mathcal{X}$. For each $\mathbf{X}_j$ we find the nearest point on $\mathcal{b}_i$, $\mathbf{X}^*$. We cast a ray from $\mathbf{X}^*$, towards $\mathbf{X}_j$ and find the intersection with the nearest part, $\mathbf{X}^h$.

Using these two distances we define a linear *distance weight* using the well-known ReLU function:

$$
\theta_i = \max\left(1.0 - \frac{d_{\text{primary}}}{\min(d_{\text{total}}, \alpha)}, 0.0\right), \tag{8}
$$

where α is a distance cutoff parameter, d_{primary} is the primary ray length and d_{total} is the total ray length. This distance weight (Fig. 4) is the result of interpolating along the ray from $\mathbf{X}^*$ to $\mathbf{X}^h$ (or a fixed

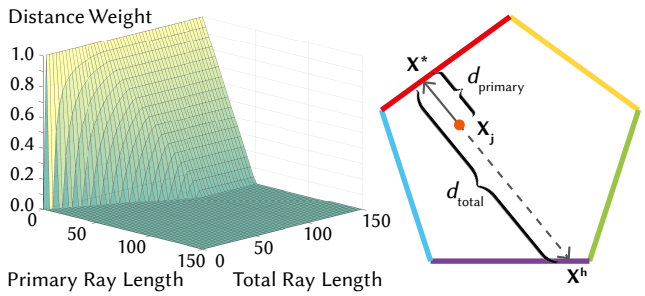

Fig. 4. Distance weight as a function of primary ray length d_{primary} and total ray length d_{total} with cutoff distance 50 (left). Here we use different colors to denote different parts (right).

point that depends on α). These weights obey the Kronecker delta property for all parts but do not partition unity which we repair using a post-normalization step. For non-planar surfaces, $\mathbf{X}^h$ may lie on the same part as $\mathbf{X}^*$. For points inside closed periodic surfaces, it is critical that the weights equal one in this case, so we set $\theta_i = 1$.

At each $\mathbf{X}_j$ we want to compute $\mathbf{w}(\mathbf{X}_j) = \big(w_1(\mathbf{X}_j), \ldots, w_n(\mathbf{X}_j)\big)$, where $\sum_{i=1}^{n} w_i(\mathbf{X}_j) = 1$, and each $w_i \geq 0$, which we enforce by solving a local optimization problem:

$$\begin{aligned} \mathbf{w}(\mathbf{X}_j) = \arg\min_{\mathbf{w}} \quad & \mathbf{w}^T \Theta(\mathbf{X}_j)\mathbf{w} \\ \text{s.t.} \quad & 0 \leq w_i \leq 1 \; \forall i \\ & \sum_i^n w_i = 1 \end{aligned} \tag{9}$$

where $\Theta(\mathbf{X}) = \text{diag}\left(\frac{1}{\theta_1}, \frac{1}{\theta_2}, \ldots, \frac{1}{\theta_n}\right)$.

This post-process ensures that our blending weights partition unity and are non-negative (Fig. 5). For any point on a part that is non-intersecting, the weights will satisfy the Kronecker delta property since the inverse distance weights imply that only a single Θ_i will have a non-infinite weight. For points that lie on more than one part, the weights associated with the intersecting parts will be equal and all others will be zero This implies that the deformed positions of these intersecting points will be an average, which is a plausible solution and ensures continuity. Finally, the distance cutoff α allows us to control the sparsity of the blending weights. We employ a simple heuristic to select α by searching for one that produces a set of blending weights meeting a desired sparsity target $T \in (0, 1)$, which is defined as the ratio of non-zeros to total number of weights. This yields a unitless, easy to tune parameter.

Using raycasting to compute blending weights imbues our method with similar advantages to Sawhney and Crane [2020] (allowing us to handle intersecting geometry, geometry with gaps and to perform constructive solid geometry operations on the fly), but allows us to use more general weighting functions and apply the appropriate behavior for intersecting parts. The price we pay is that our weights are not C^0 continuous but rather contain bounded jumps. However, these jumps occupy an infinitesimal percentage of the volume of our object, and are easy to avoid when integrating physical quantities, leaving our simulation unaffected.

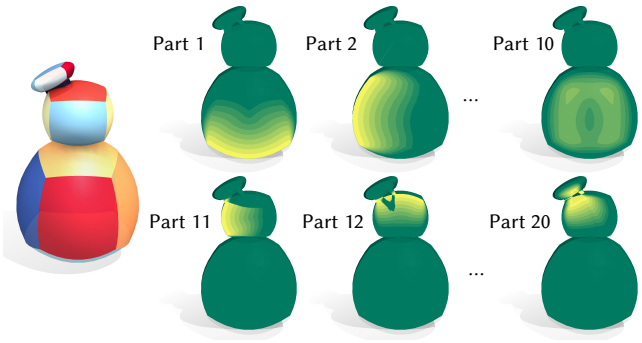

Fig. 5. Our blending weights decay smoothly from 1.0 (yellow) to 0.0 (green) when moving away from its closest surface. Here we visualize the distribution of the distance weights (with cutoff distance 5.0) corresponding to each part.

6.2 Choosing Deformation Origins

Each part, $\mathcal{b}_i$, must be associated with a single deformation origin, $\bar{\mathbf{X}}_j$, and each origin is computed as the centroid of the set of parts with which it is associated. While we constrain a part to be paired with only one deformation origin, an origin may be associated with any number of parts.

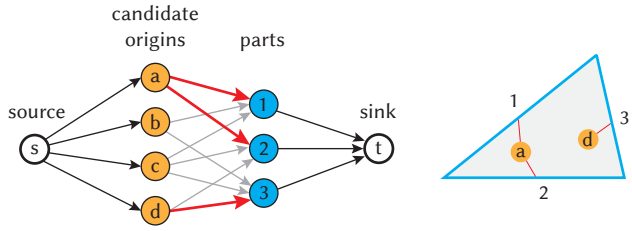

Fig. 6. Example structure for the network flow selection of deformation origins with a shape consisting of three parts. Red arrows indicate each part's origin association.

Candidate deformation origins are selected by clustering quadrature points (themselves computed in §6.3). We group quadrature points by the combination of parts influencing each point (non-zero blending weight (§6.1)). Each group now represents a unique part combination and the centroids of these groups become deformation origin candidates. To assign deformation origins to parts we use an integer network flow approach (Fig. 6).

Consider a directed bipartite graph $G = (V = A \cup B, E)$ where each node in A corresponds to a deformation origin candidate and each node in B represents a part. E is the set of edges connecting nodes in A to B where each origin has an edge to every part with which it attempts to associate. We augment G to form $G' = (V' = A \cup B \cup \{s, t\}, E')$ where V' is the new set of nodes now containing a source, s, and sink, t. E' is the augmented set of nodes connecting s to each candidate in A and connecting each part in B to the sink t. Each edge $(i, j) \in E'$ has an associated flow f_{ij} and capacity constraint u_{ij}, and each edge in E also has a cost c_{ij}. The principal constraint for this network flow problem is the conservation constraint, meaning

that the net flow for all nodes in V is zero. Each edge from B to t is assigned a capacity of 1, which enforces that any part node in B receives flow from a single edge, and as a consequence each part may only be associated with a single deformation origin. The last step is to specify the costs on the edges, E, between the deformation origin candidates and the parts.

Given a quadrature point group, and an edge from the associated deformation origin to b_i, we set the edge cost to be the sum, over all quadrature points in the group of the blending weights associated with b_i. This cost penalizes heavily "connected" origin candidates, encouraging sparsity during origin selection. This amounts to minimizing the following integer linear program:

$$\begin{aligned} \mathbf{f} = \arg\min_{\mathbf{f}} \quad & \mathbf{c}^T\mathbf{f} \\ \text{s.t.} \quad & A_{\text{cons}}\mathbf{f} = 0 \\ & f_i \in \mathbb{Z}, \forall i \\ & 0 \le f_i \le u_i \end{aligned} \tag{10}$$

where $\mathbf{f}$ and $\mathbf{c}$ are the edge flow and cost values, respectively, assembled into vectors, and A_{cons} forms the set of constraints satisfying the flow conservation constraint. We solve this using the dual simplex solver in MATLAB.

Note that this approach yields compactly supported weight functions due to the cut-off parameter (Eq. 8). Depending on this value, we obtain a range of system matrix sparsity patterns – from fully dense to block sparse (Fig. 8). Importantly, unlike other boundary-only approaches, our method is not guaranteed to produce dense system matrices.

6.3 Meshless Integration

We integrate physical quantities over our undeformed object using raycasting based quadrature [Khosravifard and Hematiyan 2010].

The key idea behind raycasting quadrature is to transform a domain integral of the form $I = \int_\Omega f(X,Y,Z)d\Omega$ to an easier form that enables us to select quadrature points in an entirely meshless manner. The first transformation is to convert the domain of the integral to an *auxiliary domain*, Ω_R on which integration is easy to compute. In our case we take this domain to be an axis-aligned bounding box around our model, and rewrite integration as $I = \int_{\Omega_R} g(X,Y,Z)d\Omega_R$ where g evaluates to zero outside of Ω, but equals f inside Ω. Next we apply the Divergence theorem to rewrite our integral as

$$\int_{\Omega_R} g(X,Y,Z)d\Omega_R = \int_{\Gamma_R}\left(\int_{X_1}^{X_2} g(\alpha,Y,Z)d\alpha\right)\mathbf{n}d\Gamma_R. \tag{11}$$

where Γ_R is the boundary of the bounding box with normals $\mathbf{n}$ and X_1 and X_2 represent the minimum and maximum values of the box along the x-axis. With this form, all we require to evaluate the outer integral is to integrate over the y-z plane located at X_1. With this simple form, the normal term may be excluded as it always equals 1 in the case of an axis-aligned bounding box. Finally, raycasting is used to evaluate the line integral along the x-axis. From here, we find all the intersection intervals where the ray is inside Ω, and select quadrature points uniformly along this interval. Thus, this transformation from the original volume integral into an integral over a y-z plane plus axis-aligned rays enables us to integrate the volumes of our NURBS models in a completely meshless approach.

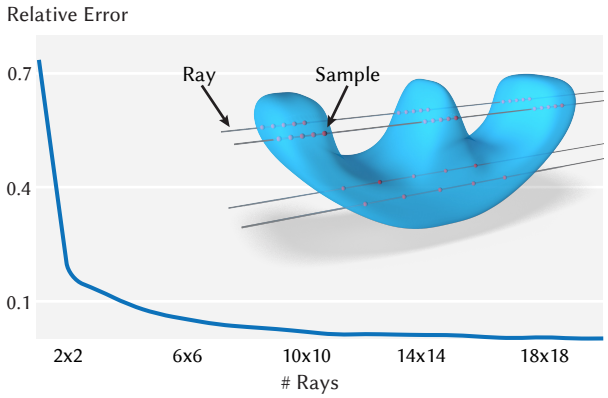

Fig. 7. When integrating physical quantities over the volumetric domain of an object, our raycasting quadrature is robust and converges to the groundtruth result as we increase the number of rays in use.

We make a minor modification to this procedure to handle common geometric pathologies encountered in NURBS modelling, such as overlapping surfaces, by performing CSG unions while raycasting [Roth 1982]. The output of this raycasting procedure is a set of quadrature points x and weights v that we use for integration and also as input to our deformation origin extraction (§6.2) and blending weights computation (§6.1) Fig. 7 shows the convergence of this quadrature approach when used to compute the volume of a NURBS model. In principle we could apply detail-aware sampling methods [Wang et al. 2012] to improve the robustness of raycasting quadrature even further, but we found it unnecessary for the examples in this paper.

7 PHYSICS-BASED ANIMATION

Having defined the individual components of the SEM method, we now briefly detail how we apply it to the problem of physics-based animation, specifically elastodynamics. The advantage of having a unified kinematic description (Eq. 6) is that we can directly apply Lagrangian Mechanics [Lanczos 2012] to arrive at the pertinent equations of motion:

$$M\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}) + \mathbf{f}_g + \mathbf{f}_{\text{ext}},$$

where $M \in \mathbb{R}^{n\times n}$ is the mass matrix, $\mathbf{q} \in \mathbb{R}^n$ is the stacked vector of surface control points, $\ddot{\mathbf{q}} \in \mathbb{R}^n$ is the generalized accelerations, $\mathbf{f} \in \mathbb{R}^n$ is the vector of internal forces, $\mathbf{f}_g \in \mathbb{R}^n$ are body forces such as gravity and $\mathbf{f}_{\text{ext}} \in \mathbb{R}^n$ are external forces such as those due to contact. We use penalty springs [McAdams et al. 2011a] to resolve contact, and the approach of Bridson et al. [2002] for approximating frictional effects. Because SEM yields equations of motion in the standard form, it is compatible with any standard integration scheme, though in this submission we limit ourselves to Implicit Euler.

7.1 Generalized Velocity

We can rewrite Eq. 6 in matrix form as

Algorithm 2 Shape Matching Element Simulation Loop

procedure SIMULATE(*model*)
  while *simulating* **do**
    $\begin{bmatrix}\mathbf{c}\\ \mathbf{t}\end{bmatrix} \leftarrow \Pi\mathbf{q}$
    for all $i \leftarrow 1$ to $|\mathcal{X}|$ **do**
      $\mathbf{F}_i \leftarrow \frac{\partial \mathbf{F}}{\partial c} c$ // deformation gradient
      $\frac{\partial V}{\partial c} \leftarrow \frac{\partial V}{\partial c} + \frac{\partial \mathbf{F}}{\partial c}^T \frac{\partial \psi}{\partial \mathbf{F}}(\mathbf{F}_i) \cdot \text{volume}_i$
      $\frac{\partial^2 V}{\partial c^2} \leftarrow \frac{\partial^2 V}{\partial c^2} + \frac{\partial \mathbf{F}}{\partial c}^T \frac{\partial^2 \psi}{\partial \mathbf{F}^2}(\mathbf{F}_i) \frac{\partial \mathbf{F}}{\partial c} \cdot \text{volume}_i$
    end for
    $K \leftarrow \Pi^T \frac{\partial^2 V}{\partial c^2} \Pi$
    $\mathbf{f}(\mathbf{q}) \leftarrow \Pi^T \frac{\partial V}{\partial c} + M_\epsilon \mathbf{q}$
    $\mathbf{f}_{\text{total}} \leftarrow \mathbf{f}(\mathbf{q}) + \mathbf{f}_g + \mathbf{f}_{\text{ext}}$
    $\dot{\mathbf{q}} \leftarrow \text{TimeIntegration}(K, \mathbf{f}_{\text{total}}, \dot{\mathbf{q}}, \hat{J}, M, M_\epsilon)$
    $\mathbf{q} \leftarrow \mathbf{q} + h\dot{\mathbf{q}}$ // h is timestep size
  end while
end procedure

$$\mathbf{x}(\mathbf{X}) = \underbrace{N(\mathbf{X}) W \Pi}_{\Gamma(\mathbf{X})} \mathbf{q}, \tag{12}$$

where $N \in \mathbb{R}^{3\times n} = \left[P_1(\mathbf{X}) \dots P_m(\mathbf{X})\right]$, n is of size $3(km + |\bar{\mathcal{X}}|)$ with k being the number of terms in each polynomial, m is the number of parts, and $|\bar{\mathcal{X}}|$ is the number of deformation origins. W is the diagonal matrix of blending weights, Π is the global projection operator and $\Gamma(\mathbf{X})$ is the kinematic Jacobian.

The velocity of a point on the object is then

$$\mathbf{v}(\mathbf{X}) = \Gamma(\mathbf{X}) \dot{\mathbf{q}}, \tag{13}$$

where $\Gamma(\mathbf{X})$ is the kinematic Jacobian and $\dot{\mathbf{q}}$ are the generalized velocities of the system.

7.2 Mass Matrix

We derive the mass matrix for our system from the definition of kinetic energy:

$$T = \frac{1}{2}\dot{\mathbf{q}}^T \underbrace{\Pi^T \int_\Omega \rho W(\mathbf{X})^T N(\mathbf{X})^T N(\mathbf{X}) W(\mathbf{X}) \, d\Omega \Pi}_{M} \dot{\mathbf{q}}. \tag{14}$$

7.3 Deformation Gradient

The deformation at a point $\mathbf{X}$ in the reference space is given by

$$\frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \sum_{i=1}^{m} \left(w_i \frac{\partial P_i(\mathbf{X})}{\partial X} \Pi_i \mathbf{q}_i + \frac{\partial w_i}{\partial X} P_i(\mathbf{X}) \Pi_i \mathbf{q} \right), \tag{15}$$

For physics-based animation, we find that it is sufficient to assume the blending weights are constant in the region around quadrature points, allowing us to discard the second term.

7.4 Constitutive Models

Our algorithm supports arbitrary elastic constitutive models. In this submission we focus on hyperelastic models [Sifakis and Barbic 2012] wherein the potential energy stored by the object, per unit volume, is given by the strain energy density function $\psi(F)$, and its internal forces and stiffness are the negative gradient and Hessian respectively. Evaluating and applying constitutive models of this form requires the ability to (1) evaluate the deformation gradient over the reference domain of the model and (2) integrate the resulting constitutive properties. SEM provides both features and thus support for such models.

7.5 External Forces

We compute the force of gravity and apply external forces in the standard way. Gravity is defined as $\mathbf{f}_g = \int_\Omega \rho \mathbf{g} d\Omega$, where $\mathbf{g}$ is the acceleration due to gravity acting on the object. We leverage our meshless quadrature (§6.3) for integration.

External point forces are applied using the Jacobian transpose technique which specifies that the generalized force, $\mathbf{f}_{\text{ext}}$, that results from a point load, applied at $\mathbf{X}$ is given by $\mathbf{f}_{\text{ext}} = \Gamma^T \mathbf{f}(\mathbf{X})$, where $\mathbf{f}$ is the point load itself. Using the Jacobian transpose allows us to easily apply all manner of external forces including collision and frictional forces.

7.6 Error Energy

Like all VEM-type methods, ours requires an error energy to ensure its consistency [Beirão da Veiga et al. 2014]. A natural choice is to adapt the error energy specified by Müller et al. [2005] to penalize deviations between the pose prescribed by $\mathbf{q}$ and that prescribed by polynomial basis.

The total error at the boundary of the object can be written as

$$\epsilon = \int_{\partial\Omega} ||\Gamma(\mathbf{X})\mathbf{q} - J(\mathbf{X})\mathbf{q}||_2^2 d\mathbf{X}, \tag{16}$$

which is a quadratic.

We augment our physical system with the quadratic penalty energy $\gamma\epsilon$ and use it to derive additional force and Hessian terms. In practice we find that choosing γ to be the Young's Modulus of the elastic solid provides good behavior in all cases. If the polynomial basis can exactly match the deformation of the boundary representation, this error naturally elides to zero.

In the next section we show a plethora of physics-based animations created using our approach.

8 RESULTS AND DISCUSSION

We implemented SEM using a combination of MATLAB and C++ using both GPToolbox [Jacobson et al. 2018] and Bartels [Levin 2020] for geometry processing and constitutive models respectively. Benchmarks were performed using a MacBook Pro with an Intel i5 2.3GHz processor, 16GB of RAM and an Intel Iris Plus Graphics 655 GPU.

Table 1 shows the size of all our examples along with performance statistics and relevant parameters. We also plot the sparsity patterns of several stiffness matrices in our examples in Fig. 8. Note that the individual parts of all models are *not connected*, and no continuity constraints or constructive solid geometry operations have been applied. Rather, the SEM approach implicitly couples the boundary parts together to allow for seamless volumetric simulation. We use quadratic deformation for all the animation results, except the rocket (stiff) and starship which use linear deformation. Models were

Table 1. Performance and parameters for the Shape Matching Element Method on all examples. All wall-clock timings are reported in seconds, physical parameters are reported with appropriate units. ρ is the applied density, **E** is the Young's Modulus and ν is the Poisson's ratio. **Sample** is the time taken to sample and construct $\hat{J}$ and $P(\hat{\mathbf{X}})$, **Quad** is the time taken to generate quadrature points via raycasting, **Weights** is the time to compute blending weights, **Build** Π is the time to build the projection operator, **Step** is the average time required to step the simulation (not including collision detection) and **Nnz** is the ratio of nonzeros to the total number of elements in the stiffness matrix. We use linearly implicit time integration on all examples and thus the number of step per frame is 1 in the simulation.

Example	|q|	Material	ρ(kg/m^3)	E(Pa)	ν	Sample(s)	Quad(s)	Weights(s)	Build Π(s)	Step(s)	Nnz(%)
Rocket (soft)	648	Rubber	$1.27e^3$	$7e^6$	0.40	$2.83e^{-2}$	$5.62e^{-3}$	$5.58e^{-1}$	$6.83e^{-2}$	$1.26e^0$	100
Rocket (stiff)	648	Steel	$8e^3$	$2e^{11}$	0.32	$2.56e^{-2}$	$5.97e^{-3}$	$4.59e^{-1}$	$3.48e^{-2}$	$9.99e^{-2}$	100
Starship	750	Steel	$8e^3$	$2e^{11}$	0.32	$6.78e^{-2}$	$1.52e^{-2}$	$8.50e^{-1}$	$9.52e^{-2}$	$6.95e^{-1}$	100
Beam (twist)	756	Rubber	$1e^3$	$1e^6$	0.45	$1.97e^{-1}$	$2.25e^{-2}$	$1.35e^0$	$5.16e^{-1}$	$3.95e^0$	46.94
Lamppost	942	Rubber	$3e^3$	$1e^6$	0.40	$1.28e^{-1}$	$1.27e^{-2}$	$1.89e^0$	$1.07e^0$	$2.22e^0$	56.38
Tire	1404	Rubber/Steel	$2e^3/3e^3$	$1e^6/7e^9$	0.47/0.35	$1.74e^{-1}$	$1.32e^{-2}$	$1.82e^0$	$6.79e^{-1}$	$1.28e^1$	92.5
Chicken	1773	Jelly	$1.27e^3$	$1e^4$	0.47	$9.19e^{-1}$	$2.31e^{-2}$	$3.38e^0$	$1.91e^0$	$1.08e^1$	100
Coffee mug (soft)	1800	Jelly	$1e^3$	$1e^4$	0.47	$4.18e^{-1}$	$2.87e^{-2}$	$3.77e^0$	$2.08e^0$	$1.19e^1$	100
Coffee mug (stiff)	1800	Rubber	$1e^3$	$4e^7$	0.40	$4.15e^{-2}$	$1.17e^{-2}$	$6.86e^{-1}$	$1.09e^{-1}$	$5.00e^{-1}$	81.42
Castle	1872	Jelly	$1.27e^3$	$2e^3$	0.45	$4.84e^{-1}$	$1.44e^{-2}$	$1.52e^0$	$5.78e^{-1}$	$8.02e^0$	100
Grumpy	2010	Jelly	$1e^3$	$1e^4$	0.40	$2.92e^{-1}$	$1.23e^{-2}$	$2.29e^0$	$2.89e^0$	$6.02e^0$	84.15
Astronaut (soft)	3609	Rubber	$1.27e^3$	$1e^6$	0.47	$2.47e^0$	$6.71e^{-2}$	$1.13e^1$	$1.09e^1$	$1.41e^1$	49.04
Squid	4347	Jelly	$1e^3$	$2e^4$	0.40	$4.48e^{-1}$	$2.47e^{-2}$	$1.54e^1$	$4.48e^1$	$6.73e^0$	25.47

created using Autodesk Fusion 360 [Autodesk 2021] and Rhinoceros 3D 7 [McNeel and Associates 2021]. Please see the supplemental video for more animation results.

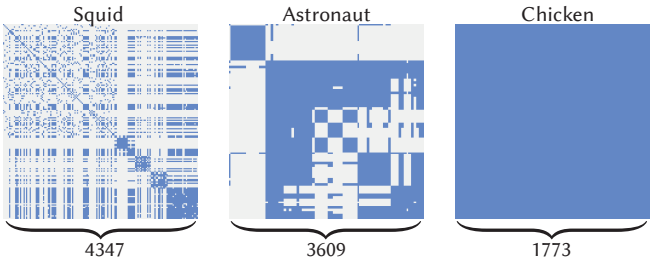

Fig. 8. Our stiffness matrices can be sparse, while the corresponding matrix has to be dense in the traditional Boundary Element Method [James and Pai 1999]. Sparsity of our stiffness matrix is primarily determined by weight sparsity (number of non-zero weights at a given quadrature point) and polynomial order.

For raycasting operations and rendering we rely on Embree [Wald et al. 2014] and Blender [Blender 2020], and triangulate the NURBS surfaces, which only represents an implementation simplification. Our core algorithm is not dependent on triangulating the boundary, and an ideal alternative would be using a path tracer that directly accepts NURBS geometry. For collision detection and handling, for simplicity we use the triangulation to detect inter-penetrations at each iteration, and attempt to move the collided vertex out of its collision surface using a spring-like penalty force, similar to [McAdams et al. 2011b; Xian et al. 2019]. An ideal alternative would be adopting the implicit representation of the surfaces for collision detection and resolution [Buffet et al. 2019; Weber and Gornowicz 2009].

For handling trimmed NURBS surfaces, we augment our sampling of the surfaces by using raycasting integration in 2D. Trimmed NURBS may be defined by a boundary curve in the parametric space, so we raycast to find intersection intervals in the UV space on which quadrature points are generated. For rendering Trimmed NURBS, we discretize the the boundary curve as a polyline and use triangle [Shewchuk 1996, 2002] to form a boundary conforming triangulation. The vertices of this triangulation serve as fixed set of high resolution UV samples for constructing a render mesh.

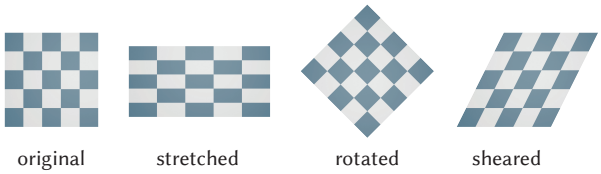

Fig. 9. 2D patch test. By applying affine transformations to the boundary of the original undeformed model (left), we show that solving the static problem gives rigid motion for rotation and constant strain for shearing and stretching (right).

We first validate the physical plausibility of our method using a small 2D patch test (Fig. 9). Here the test object is a square made of four edges (not joined at the corners) simulated using linear polynomials with a single deformation center. We apply a battery of boundary conditions and resolve the deformation of the element by minimizing the elastic potential. We note that SEM is able to represent rigid motions, as well as shearing and anisotropic stretching. This implies that, with sparse weights, SEM can resolve these motions locally, leading to physically plausible simulation results.

We demonstrate *qualitative* convergence of SEM with respect to linear tetrahedral finite elements when increasing the number of patches in use. In Fig. 11, we compare the static configurations

of identical cantilevered beams, simulated using a Neohookean constitutive model (Young's Modulus = 0.001 GPa, Poisson's Ratio = 0.45). We further perform a basic *quantitative* comparison against FEM by measuring the deflection of the beam at equilibrium as we increase the number of patches along the beam. In Fig. 10, the deflection of the SEM beams is reported as a percentage of the FEM beam's deflection.

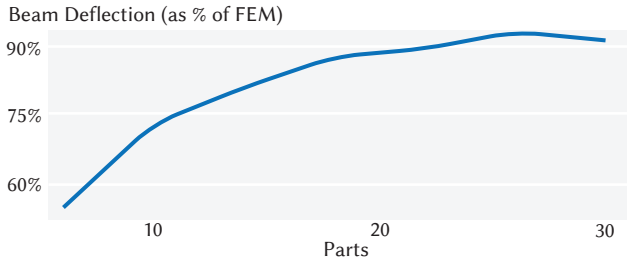

Fig. 10. The SEM beam deflection is plotted as a percentage of the FEM beam's deflection when the beam is at equilibrium. As we subdivide the SEM beam, its deflection converges to the FEM beam's deflection.

We use SEM with quadratic polynomials for this test and observe that our SEM simulation, made of 26 independent NURBS patches, shows good agreement with FEM. Each subdivision of the NURBS beam enables more complicated kinematics, but very few surface elements are needed to produce compelling results.

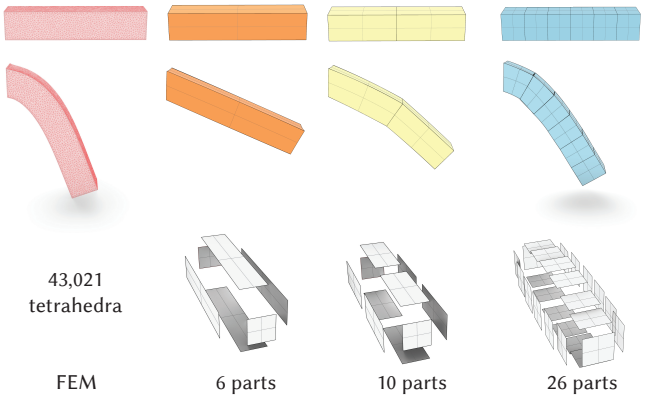

Fig. 11. Our SEM simulation is able to qualitatively converge to a high resolution FEM simulation result (left) as we increase the number of surfaces in the model (right).

Additionally, we perform a twisting beam experiment on a 22 part beam model. Despite the relatively small number of NURBS patches, we see in Fig. 12 that the quadratic polynomials permit complex deformation.

Fig. 13 shows a scalability study in which a unit cube, composed of 6 parts, is extended by appending 4 more parts at a time. Note that our algorithm scales similarly to standard high-order finite element approaches due to the compact support of our weight functions.

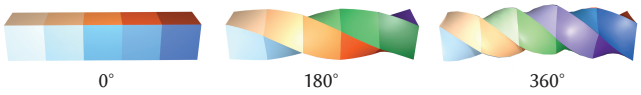

Fig. 12. With the left end of the beam held fixed, we show the result of rotating the right end of a 22 part beam up to 360 degrees.

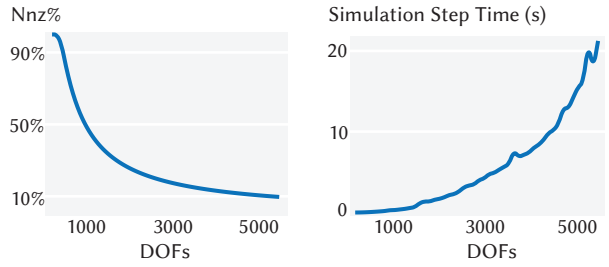

Fig. 13. Here we progressively add more degrees of freedom to a beam by iteratively inserting 4 part sections, extending the beam in the x-axis direction at each step. On the left we report the number of non-zeros as a percentage of the total number of elements in the stiffness matrix (Nnz%). The right plot shows the average runtime of a simulation step as the number of degrees of freedom increases.

We also show that our raycasting weight computation is able to create shape-aware output. Fig. 14 and Fig. 15 show that manipulating parts that are nearby but separated will behave in an appropriately independent fashion. Our grumpy model is capable of extending its leg without causing unrealistic deformations in the plant foot, and our octopus model shows independent motion of all eight limbs upon contact with the ground.

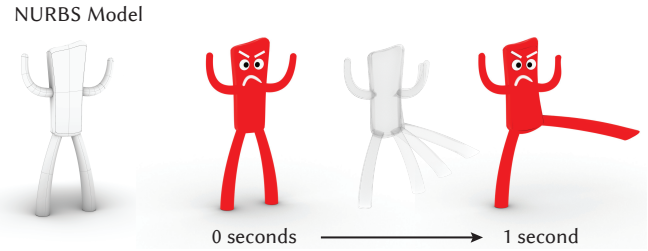

Fig. 14. Our raycasting weight computation produces shape-aware blending weights. Here the locality of our blending weights allows the two legs of the grumpy model to move independently.

By virtue of its meshless nature, SEM is robust to a wide range of challenging models with large gaps and disconnected primitives. Fig. 16 shows frames from a simulation of a rubber chicken. Note that the chicken model itself features large gaps between the individual NURBS parts. Despite the lack of explicit connectivity, the SEM blending weights have the effect of implicitly enforcing connectivity at these seams.

SEM is also robust to intersections in modelling input. Fig. 17 shows simulations of two jelly coffee mugs. The top row shows a

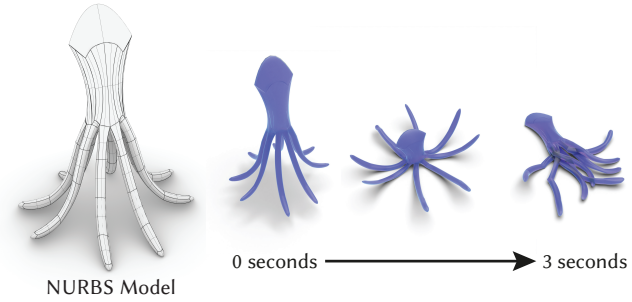

Fig. 15. We simulate a falling octopus, and observe the 8 limbs moving independently. Self-collision is not accounted for here.

model with negligible overlap, whereas the bottom row shows the result of a careless modeller who has deeply embedded the handle of the mug in the body in order to attach it, creating a large area of overlap. In both cases SEM produces a plausible physically-based animation without requiring additional model clean-up.

Since the equations of motion for SEM are derived using a general elastic potential, it is theoretically capable of supporting arbitrary elastic constitutive models. Fig. 18 shows a rocket ship simulated with both high stiffness (steel) and low stiffness materials (rubber). In both cases intuitive and visually pleasing results are created wherein the trajectories correctly reflect the desired material characteristics.

SEM also supports the simulation of objects made up of heterogeneous materials. By specifying different material properties inside nested parts we are able to simulate a drag racing wheel with a metal hub and soft rubber tread (Fig. 19).

For downstream applications, the output of SEM simulation is itself editable in a NURBS modeling program (see Fig. 20). This can allow artists to easily post-process animations using the same tools in which they were created.

Finally we show a few additional examples, highlighting the ability of SEM to handle complex simulations involving deformation, contact and friction. In Fig. 21 we show two astronauts in jaunty space suits collide in zero gravity. SEM gracefully allows bounded discontinuities in simulation output to allow for rich motion. This is similar to Discontinous Galerkin approaches but in SEM this behavior emerges from the kinematic model, rather than from additional flux terms [Kaufmann et al. 2009]. Our soft enforcement of part connectivity at boundaries is a result of the weighting function attributing equal weight values to equidistant parts. This equal weighting induces elastic forces that attempt to enforce continuity. In our examples, these discontinuities only introduce minor artifacts affecting the rendering, but this may be addressed using a separate render mesh or by using a representation with explicit connectivity.

In Fig. 1 we directly simulate the NURBS surface model of a bouncy castle under a periodic wind force. In Fig. 22 we show an animation of the Space-V starship landing on Mars in a graceful way. This example shows the benefit of NURBS modeling: a relatively small number of primitives can represent a complex shape. SEM successfully handles the stiff materials, complicated geometry and non-trivial collisions in this scene.

9 FUTURE WORK AND CONCLUSIONS

We have presented the Shape Matching Element Method (SEM), the first completely meshless approach to direct simulation of curved surface models, made from NURBS primitives. Our approach is unique in its ability to infer volumetric shape from surface only input, including input with intersecting geometry between parts and other defects common to non-engineering models. We believe that SEM is a significant improvement over standard physics-based animation pipelines. As evidence of this, the authors submit that many of the examples in this paper were constructed ourselves (since the standard graphics menagerie is not available as NURBS models). Modelling, cleaning, meshing and simulating would've been a burdensome experience without SEM's ability to leap directly from (often hastily) constructed models to physics-based animations.

SEM, as it's presented here, is in its infancy and we believe there are many exciting areas of future work to explore. We are very excited to couple SEM to machine learning approaches for design, parameter estimation and Real2Sim applications. One of the cumbersome elements in using finite element simulation for such problems is the need for robust, differentiable volumetric meshing [Gao et al. 2020]. SEM removes this bottleneck entirely, providing a direct mapping from geometric input to physics-driven output. We believe SEM will enable simpler and more robust algorithms for physics-based ML and allow the application of such algorithms to a much broader class of problems.

SEM itself has much room for improvement. First, while we focus on NURBS surfaces here, the only part of SEM that is NURBS specific is the shape matching operation. We believe there is potential to allow mixed models (models which include polygonal meshes, particles, subdivision surfaces and NURBS) by extending the range of shape matching operations used by the algorithm. The shape matching operation itself could be improved to be material-aware (to better handle heterogeneous materials) or to be robust to noisy data (allowing direct simulation of scanned data). In future work we also intend to explore using robust solvers or orthogonal bases for the shape matching step itself. It is also interesting to consider the relationship between our error term and the orthogonality constraint proposed by Zhang et al. [2020]. Their approach could be used to remove this term from SEM entirely. Finally, in this paper we have focused on applying SEM to physics-based animation, and there is a significant amount of additional work needed to extend SEM reliably into engineering applications.

The Finite Element method took over 40 years to mature to its current state and to become the preeminent tool in physics-based animation. We hope that this is the beginning of a similar, exciting journey for SEM. Motivated by this sentiment, and to encourage future research on SEM, the authors will release our SEM implementation under a permissive license as well as all models created for this submission.

ACKNOWLEDGMENTS

This work is funded in part by NSERC Discovery (RGPIN-2017-05524), Connaught Fund (503114), CFI-JELF Fund, Accelerator (RGPAS-2017-507909), and the Canada Research Chairs Program. We thank Abhishek Madan, Otman Benchekroun, Sarah Kushner, Michael Tao,

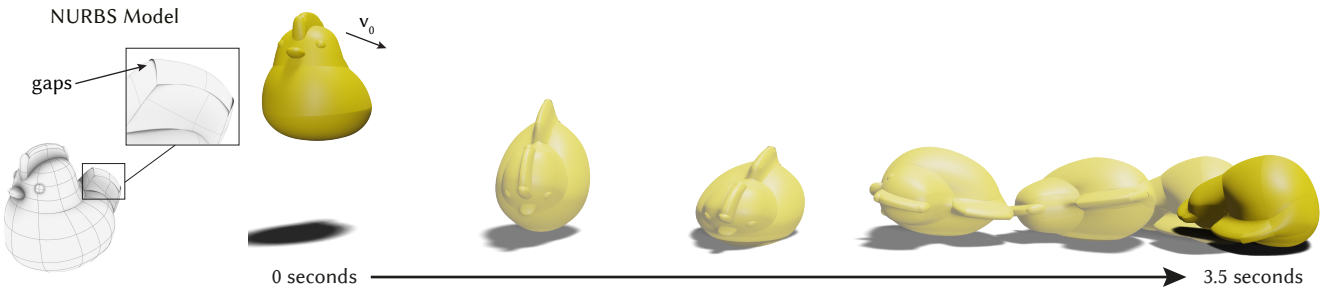

Fig. 16. Simulation of a chicken model with gaps between surfaces.

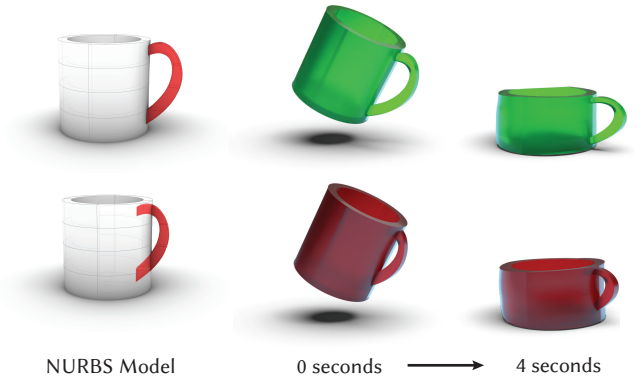

Fig. 17. Our SEM is robust to overlapping regions and self-intersections. Here the simulation result of the coffee mug with overlapping handle (bottom) still matches its corresponding counterpart without negligible overlaps (top).

Nicholas Vining, and Christopher Batty for proofreading; anonymous reviewers for their helpful comments and suggestions.

REFERENCES

Martin Aigner, Christoph Heinrich, Bert Jüttler, Elisabeth Pilgerstorfer, Bernd Simeon, and Anh-Vu Vuong. 2009. Swept Volume Parameterization for Isogeometric Analysis. 19–44.

Sheldon Andrews, Marek Teichmann, and Paul G. Kry. 2016. Blended Linear Models for Reduced Compliant Mechanical Systems. *IEEE Tran. on Visualization and Computer Graphics (TVCG)* 22, 3 (2016), 1209–1222.

Autodesk. 2021. *Autodesk Fusion 360.* https://www.autodesk.ca/en/products/fusion-360/

L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo. 2014. The Hitchhiker's Guide to the Virtual Element Method. *Mathematical Models and Methods in Applied Sciences* 24, 08 (2014), 1541–1573.

Online Blender. 2020. Blender - a 3D modelling and rendering package. http://www.blender.org

Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. 2012. Shape-Up: Shaping Discrete Geometry with Projections. *Comput. Graph. Forum* 31, 5 (Aug. 2012), 1657–1667.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014), 11 pages.

Franco Brezzi, Konstantin Lipnikov, and Valeria Simoncini. 2005. A family of mimetic finite difference methods on polygonal and polyhedral meshes. *Mathematical Models and Methods in Applied Sciences* 15 (04 2005).

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Trans. Graph.* 21, 3 (July 2002), 594–603.

Thomas Buffet, Damien Rohmer, Loïc Barthe, Laurence Boissieux, and Marie-Paule Cani. 2019. Implicit Untangling: A Robust Solution for Modeling Layered Clothing. *ACM Trans. Graph.* 38, 4, Article 120 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3323010

J. Cottrell, Thomas Hughes, and Yuri Bazilevs. 2009. *Isogeometric Analysis: Toward integration of CAD and FEA.*

Fernando De Goes, Andrew Butts, and Mathieu Desbrun. 2020. Discrete Differential Operators on Polygonal Meshes. *ACM Trans. Graph.* 39, 4, Article 110 (July 2020), 14 pages.

R. Diziol, J. Bender, and D. Bayer. 2011. Robust Real-Time Deformation of Incompressible Surface Meshes. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '11).* ACM, New York, NY, USA, 237–246.

François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K. Pai. 2011. Sparse Meshless Models of Complex Deformable Solids. *ACM Trans. Graph.* 30, 4, Article 73 (July 2011), 10 pages.

Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2020. Learning Deformable Tetrahedral Meshes for 3D Reconstruction. In *Advances In Neural Information Processing Systems.*

Benjamin Gilles, Guillaume Bousquet, Francois Faure, and Dinesh K. Pai. 2011. Frame-Based Elastic Models. *ACM Trans. Graph.* 30, 2, Article 15 (April 2011), 12 pages.

G. Haasemann, M. Kästner, S. Prüger, and V. Ulbricht. 2011. Development of a quadratic finite element formulation based on the XFEM and NURBS. *Internat. J. Numer. Methods Engrg.* 86, 4-5 (2011), 598–617.

Christian Hafner, Christian Schumacher, Espen Knoop, Thomas Auzinger, Bernd Bickel, and Moritz Bächer. 2019. X-CAD: Optimizing CAD Models with Extended Finite Elements. *ACM Trans. Graph.* 38, 6, Article 157 (Nov. 2019), 15 pages.

Ch. Heinrich, B. Simeon, and St. Boschert. 2012. A finite volume method on NURBS geometries and its application in isogeometric fluid–structure interaction. *Mathematics and Computers in Simulation* 82, 9 (2012), 1645 – 1666.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4, Article 60 (July 2018), 14 pages.

Alec Jacobson et al. 2018. gptoolbox: Geometry Processing Toolbox. http://github.com/alecjacobson/gptoolbox.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph. (proceedings of ACM SIGGRAPH)* 30, 4 (2011), 78:1–78:8.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. 2014a. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses.*

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and J. P. Lewis. 2014b. Skinning: Real-Time Shape Deformation (Full Text Not Available) *(SIGGRAPH '14).* ACM, 1 pages.

Alec Jacobson and Olga Sorkine. 2011. Stretchable and Twistable Bones for Skeletal Shape Deformation. *ACM Trans. Graph. (proceedings of ACM SIGGRAPH ASIA)* 30, 6 (2011), 165:1–165:8.

Doug L. James and Dinesh K. Pai. 1999. ArtDefo: Accurate Real Time Deformable Objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99).* 65–72.

Peter Kaufmann, Sebastian Martin, Mario Botsch, and Markus Gross. 2009. Flexible simulation of deformable models using discontinuous galerkin fem. *Graphical Models* 71, 4 (2009), 153–167.

Amir Khosravifard and Mohammad Rahim Hematiyan. 2010. A new method for meshless integration in 2D and 3D Galerkin meshfree methods. *Engineering Analysis with Boundary Elements* 34, 1 (2010), 30 – 40.

Cornelius Lanczos. 2012. *The variational principles of mechanics.* Courier Corporation.

Grégory Legrain. 2013. A NURBS enhanced extended finite element approach for unfitted CAD analysis. *Computational Mechanics* 52 (04 2013).

David I.W. Levin. 2020. Bartels: A lightweight collection of routines for physics simulation. https://github.com/dilevin/Bartels.

Xiang Li, Nandan Sudarsanam, and Daniel D Frey. 2006. Regularities in data from factorial experiments. *Complexity* 11, 5 (2006), 32–45.

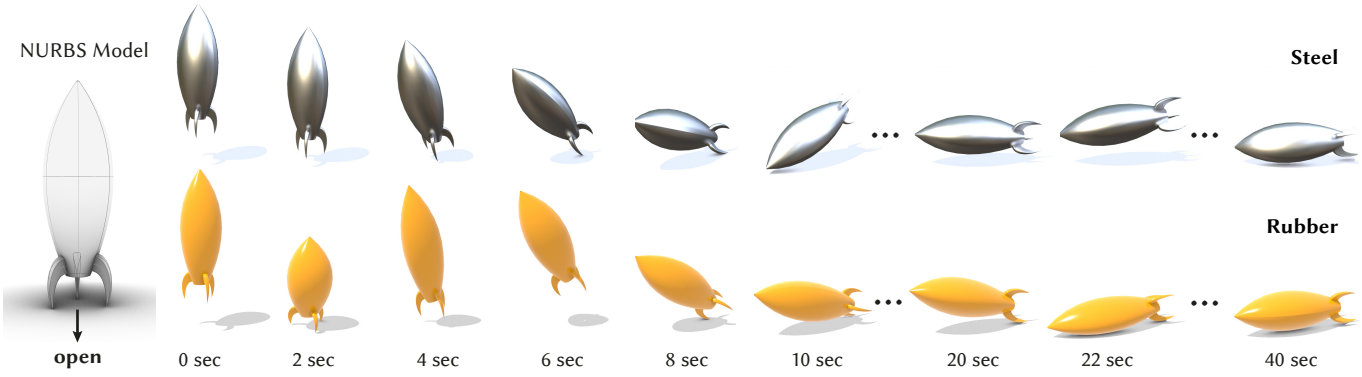

Fig. 18. SEM is able to produce animation using a wide variety of material parameters. Here we simulate both a steel and rubber rocket ship. To make this even more challenging, the rocket model is an open surface (at the bottom), but a plausible animation is still generated.

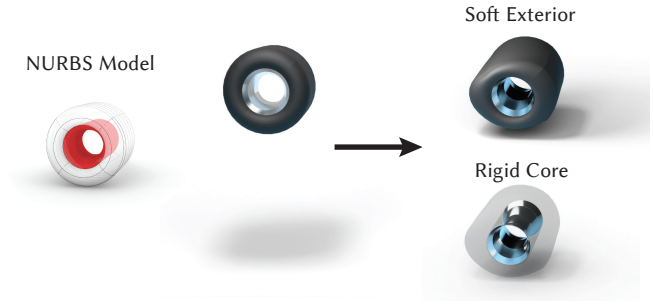

Fig. 19. Our SEM is directly applicable to the simulation of objects with heterogenous materials.

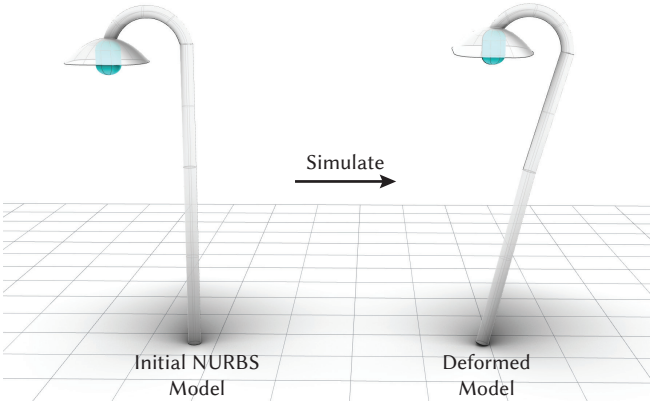

Fig. 20. Here we show the output of an SEM animation loaded into Rhinoceros 3D 7 for additional editing.

Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. 2014. Mimetic Finite Difference Method. *J. Comput. Phys.* 257 (Jan. 2014), 1163–1227.

Wing Kam Liu, Sukky Jun, and Yi Fei Zhang. 1995. Reproducing kernel particle methods. *International Journal for Numerical Methods in Fluids* 20, 8-9 (1995), 1081–1106.

Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. 2010. Unified Simulation of Elastic Rods, Shells, and Solids. *ACM Trans. Graph.* 29, 4, Article 39 (July 2010), 10 pages.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011a. *Efficient Elasticity for Character Skinning with Contact and Collisions.* ACM, New York, NY, USA.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011b. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph.* 30, 4, Article 37 (2011), 12 pages.

Robert McNeel and Associates. 2021. *Rhinoceros 3D.* https://www.rhino3d.com/

Matthias Müller and Nuttapong Chentanez. 2011. Solid Simulation with Oriented Particles. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11).* ACM, New York, NY, USA, Article 92, 10 pages.

Matthias Müller, Nuttapong Chentanez, and Miles Macklin. 2016. Simulating Visual Geometry. In *MIG.* ACM, 31–38.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *J Vis Commun Image R* 18, 2 (2007), 109–118.

Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. 2005. Meshless Deformations Based on Shape Matching. *ACM Trans. Graph.* 24, 3 (July 2005), 471–478.

M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. 2004. Point Based Animation of Elastic, Plastic and Melting Objects. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Eurographics Association, 141–151.

Matthias Muller, Matthias Teschner, and Markus Gross. 2004. Physically-Based Simulation of Objects Represented by Surface Meshes. In *Proceedings of the Computer Graphics International (CGI '04).* IEEE Computer Society, USA, 26–33.

Matthieu Nesme, Paul G. Kry, Lenka Jeřábková, and François Faure. 2009. Preserving Topology and Elasticity for Embedded Deformable Models. *ACM Trans. Graph.* 28, 3, Article 52 (2009), 9 pages.

Joachim Nitsche. 1971. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 36 (1971), 9–15.

Alec R. Rivers and Doug L. James. 2007. FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07).* ACM, New York, NY, USA, 82–es.

Scott D Roth. 1982. Ray casting for modeling solids. *Computer Graphics and Image Processing* 18, 2 (1982), 109 – 144.

Masoud Safdari, Ahmad R. Najafi, Nancy R. Sottos, and Philippe H. Geubelle. 2015. A NURBS-based interface-enriched generalized finite element method for problems with complex discontinuous gradient fields. *Internat. J. Numer. Methods Engrg.* 101, 12 (2015), 950–964.

Masoud Safdari, Ahmad R. Najafi, Nancy R. Sottos, and Philippe H. Geubelle. 2016. A NURBS-based generalized finite element scheme for 3D simulation of heterogeneous materials. *J. Comput. Phys.* 318 (2016), 373 – 390.

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains. *ACM Trans. Graph.* 39, 4 (2020).

Ruben Sevilla, Sonia Mendez, and Antonio Huerta. 2008. Nurbs-enhanced finite element method (NEFEM). *Internat. J. Numer. Methods Engrg.* 76 (10 2008), 56–83.

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Ming C. Lin and Dinesh Manocha (Eds.). Lecture Notes in Computer Science, Vol. 1148. Springer-Verlag, 203–222. From the First ACM Workshop on Applied Computational Geometry.

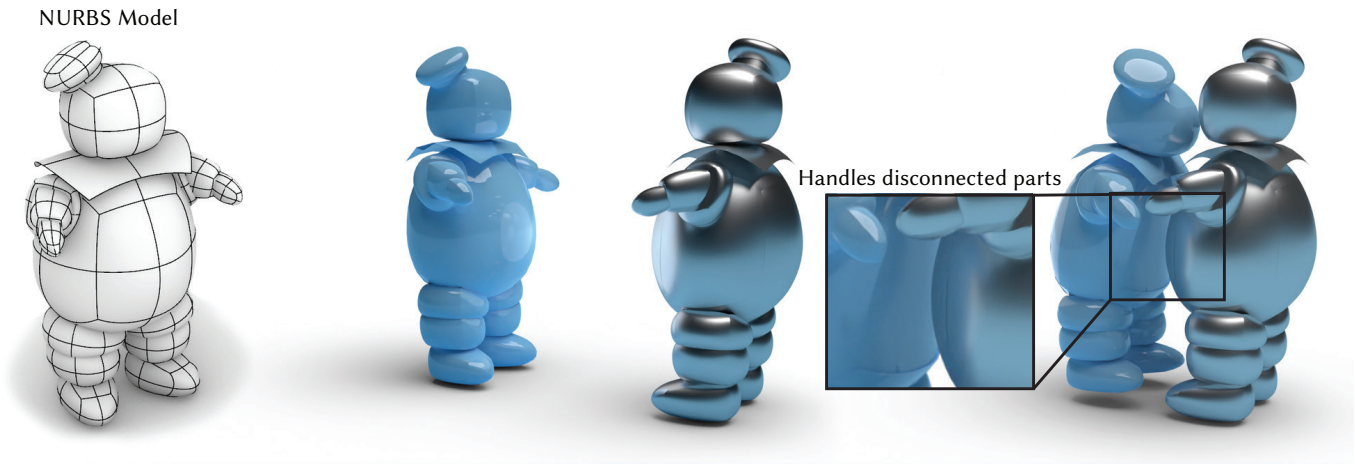

Fig. 21. Two astronauts, in jaunty spacesuits collide while spacewalking in zero-g. SEM correctly handles severely disconnected parts by correctly estimating elastic energy even in discontinuous regions.

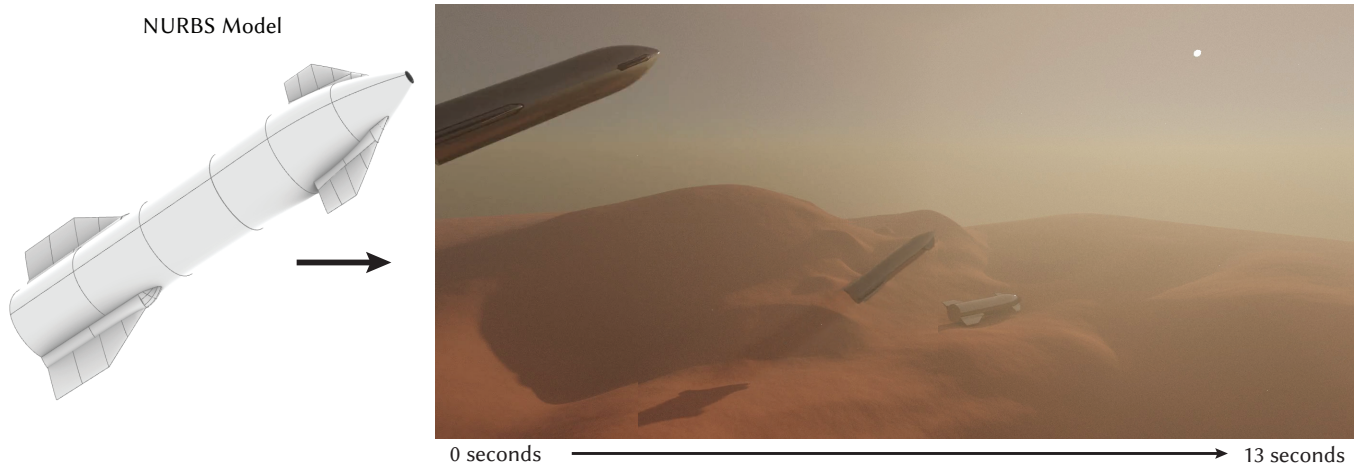

Fig. 22. Our SEM is robust to stiff materials, complicated geometry and non-trivial collisions. Here we show the Space-V starship makes a graceful landing on the rugged surface of Mars.

Jonathan Richard Shewchuk. 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry* 22, 1 (2002), 21 – 74. 16th ACM Symposium on Computational Geometry.

Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses*. ACM, Article 20, 50 pages.

Denis Steinemann, Miguel A. Otaduy, and Markus Gross. 2008. Fast Adaptive Shape Matching Deformations. In *ACM/Eurographics Symposium on Computer Animation*. Eurographics Association, 87–94.

Thomas Stumpp, Jonas Spillmann, Markus Becker, and Matthias Teschner. 2008. A Geometric Deformation Model for Stable Cloth Simulation. *VRIPHYS 2008*, 39–46.

Michael Tao, Christopher Batty, Eugene Fiume, and David Levin. 2019. Mandoline: Robust Cut-Cell Generation for Arbitrary Triangle Meshes. *ACM Trans. Graph.* (2019).

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically Deformable Models. 21, 4 (1987), 205–214.

Demetri Terzopoulos and Hong Qin. 1994. Dynamic NURBS with Geometric Constraints for Interactive Sculpting. *ACM Trans. Graph.* 13, 2 (April 1994), 103–136.

L. Veiga, Franco Brezzi, Andrea Cangiani, G. Manzini, L. Marini, and Alessandro Russo. 2012. Basic principles of Virtual Element Methods. *Mathematical Models and Methods in Applied Sciences* 23 (11 2012).

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.* 33, 4, Article 143 (July 2014), 8 pages.

Bin Wang, François Faure, and Dinesh K. Pai. 2012. Adaptive image-based intersection volume. *ACM Trans. Graph.* 31, 4 (2012), 97:1–97:9.

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear Subspace Design for Real-Time Shape Deformation. *ACM Trans. Graph.* 34, 4, Article 57 (July 2015), 11 pages.

Andrew J. Weber and Galen Gornowicz. 2009. Collision-Free Construction of Animated Feathers Using Implicit Constraint Surfaces. *ACM Trans. Graph.* 28, 2, Article 12 (2009), 8 pages.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Trans. Graph.* 38, 6, Article 162 (2019), 13 pages.

Jiayi Eris Zhang, Seungbae Bang, David I.W. Levin, and Alec Jacobson. 2020. Complementary Dynamics. *ACM Trans. Graph.* (2020).