The Shape Matching Element Method for Meshless Animation of NURBs Models

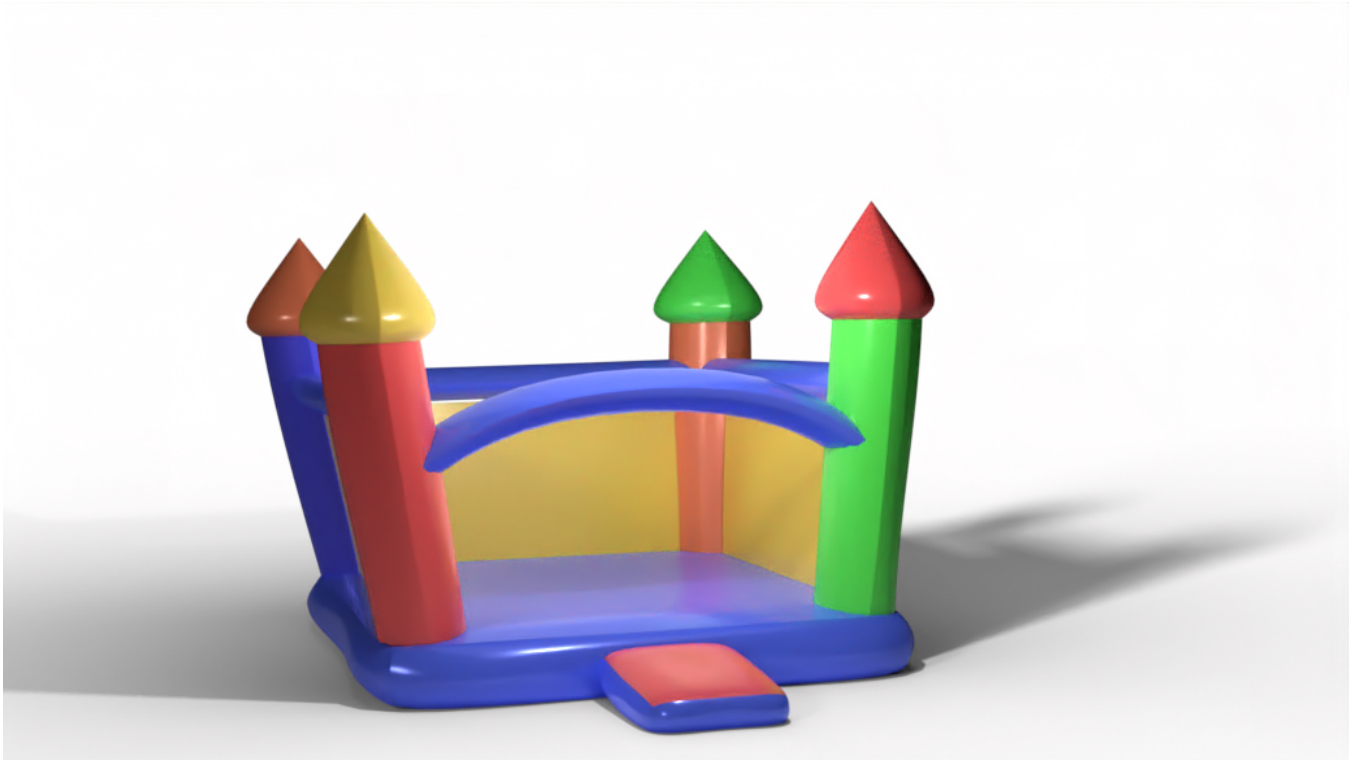

Figure 1: tmp

ACM Reference Format:
. 2021. The Shape Matching Element Method for Meshless Animation of NURBs Models. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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, yet, despite over three decades of research, animation of NURBS objects 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 difficult) 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 modeller.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

We present the first truly meshless (no volumetric discretization generated) 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) and approriate physical parameters. Because we explicitly use the NURBS boundary representation in the simulation its is straightforward to handle Dirichilet, Neumann boundary conditions and to apply contact resolution. Crucially because we broadly target animation not necessarily simulation for engineering or manufacturing we don't require that models satisfy the rigourous geometric requirements common for these applications

Our approach is an extension of the recently developed Virtual Element Method (VEM) for solving PDEs on domains tiled with arbitrary polygons. We establish a connection between VEM and the well-known shape matching simulation algorithm which enables us to derive equations of motion for an arbitrary NURBS model using Lagrangian Mechanics. Importantly, we show how to replace volumetric data structures for integration with ray casting approaches which enables our meshless approach to isogeometric elastodynamic physically-based animation of volumetric structures.

2 RELATED WORK

Geometric modeling is a necessary precursor to physics-based animation, however differing geometric representations for modelling and simulation often require 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 physically-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 cutcells), 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 mid-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 be generated. Modern approaches are constructed around the extended finite element method which enriches the standard finite element basis with discontinuous basis functions to enable improved 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 geomtric model against it to enable boundary handling (such a mesh is called a *cutcell* mesh). Like volumetric meshing, this cutting operation can be difficult and our ideal method would avoid it. Some cutcell 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, increasing complexity.

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 modeling 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 can to 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 support more advanced constitutive models and integration schemes but often require background integration meshes and lose the direct connection with modelling geometry. Like cutcell-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 NURBS based 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. 2014]. Our method is entirely meshless (requiring no volumetric meshes or cutcells) and is compatible with standard constitutive models and time integrators. It guarantees that simulation output is directly consumable by the input modeling 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 NURBS models for physics-based animation.

3 METHODS

Dave: We should probably put some pseudo code for the preprocessing and the runtime algorithms right up fron, along with a quick textual walk through of the method. Given a volumetric model composed of one or more NURBS surfaces, the surface of each surface is reconstructed using a set of control points $\mathbf{p}$ and their associated weights w. Following D-NURBS [Terzopoulos and Qin 1994], we take our degrees of freedom (DOF) to be control points of the surfaces and produce displacements directly on these control points at each simulation step. We note that we used a simplified version of D-NURBS in which the weights are held constant throughout the simulation and find this not to be limiting (feel like we'll need better justification than saying just this?). We represent the degrees of freedom for the i-th NURBS surface with m control points as $\mathbf{q}_i = \left[\mathbf{p}_1^T, \ldots, \mathbf{p}_m^T\right]^T \in \mathbb{R}^{3m}$. From this definition, we write the DOF for the entire model as $\mathbf{q} = \left[\mathbf{q}_i^T, \ldots, \mathbf{q}_n^T\right]^T$ where n is the number of NURBS surfaces for the model.

To evaluate the volumetric integrals in solid mechanics models, we require material points $\mathbf{X} \in \mathbb{R}^{3m}$ in the undeformed space that serve as integration points. Our shape matching element method enables us to evaluate the deformed positions of the material points strictly using positions defined on the boundary. Like in VEM, we fit a polynomial to each boundary primitive (NURBS in this case) describing its deformation. The fitting of these polynomials closely follows that of the Shape Matching [Müller et al. 2005] approach where we perform a least squares polynomial fit to the deformed boundary points. These boundary points are represented by sampled positions on each NURBS surface. Since each surface is associated with a single polynomial, we can construct a unique polynomial for some arbitrary point by blending these polynomials. Thus given some material point $\mathbf{X}_i$ we find it's corresponding deformed position $\mathbf{x}_i$ via a weighted combination of the polynomials. This permits us to build a general deformation map which we use to construct our equations of motion.

In the following sections, we will first review the simplified D-NURBS model used to form our generalized coordinates. We then present our shape matching algorithm for fitting polynomials to surfaces .... In section ... we next describe how to blend these surface polynomials to construct a general deformation map for material points. Section ... describes our strategy for selecting integration points in the volume through a raycasting approach. Finally, we formulate our equations of motions from a Lagrangian formulation.

4 D-NURBS MODEL

TODO: Most of these matrices should be paranthesis matrices We provide a brief overview of the D-NURBS formulation presented in [Terzopoulos and Qin 1994]. Our models in our method are composed of a collection of NURBS surfaces.

4.1 D-NURBS Formulation

A point on a NURBS surface is typically expressed in the following form

$$\mathbf{x}(u,v) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{p}_{i,j}w_{i,j}B_{i,k}(u)B_{j,l}(v)}{\sum_{i=1}^{n}\sum_{j=1}^{m}w_{i,j}B_{i,k}(u)B_{j,l}(v)}, \tag{1}$$

where where have a total of nm control points $\mathbf{p}$ and weights w. Using the B-spline basis functions and the weighted control points, this formula gives a map from our parametric coordinates u, v to their corresponding position on the surface in $\mathbb{R}^{3m}$. $B_{i,k}(u)$ is the B-spline basis function for the $i-th$ control point with degree $k-1$.

We would like our generalized coordinates to be the control points of the model so that the dynamics operates directly on the original representation. Therefore, we would like to rewrite equation 1 in the form

$$\mathbf{x}(u,v) = \mathbf{J}(u,v)\mathbf{q}, \tag{2}$$

where $\mathbf{q} = \left[\mathbf{p}_{1,1}^T, \ldots, \mathbf{p}_{i,j}^T, \ldots, \mathbf{p}_{n,m}^T\right]^T \in \mathbb{R}^{3nm}$ is the set of nm control points in the overview I just write n control points, but here i'm writing nm for a single NURBS surface arranged as a column vector. $\mathbf{J} \in \mathbb{R}^{3x3nm}$ is the NURBS jacobian that maps (u,v) coordinates to world positions in $\mathbb{R}^3$.

The NURBS jacobian, $\mathbf{J}$, is a matrix composed of horizontally concatenated 3x3 blocks where the (i,j)-th block is $\frac{\partial \mathbf{x}}{\partial \mathbf{p}_{i,j}}$ for the (i,j)-th control point. $\frac{\partial \mathbf{x}}{\partial \mathbf{p}_{i,j}}$ is a diagonal 3x3 matrix where each diagonal entry $N_{i,j}(\mathbf{u})$ takes the form

$$N_{i,j}(u,v) = \frac{w_{i,j}B_{i,k}(u)B_{j,l}(v)}{\sum_{i=1}^{n}\sum_{j=1}^{m}w_{i,j}B_{i,k}(u)B_{j,l}(v)}. \tag{3}$$

Thus the NURBS jacobian for a single (u,v) pair is written as

$$\mathbf{J}(u,v) = \left[\frac{\partial \mathbf{x}}{\partial \mathbf{p}_{1,1}} \ldots \frac{\partial \mathbf{x}}{\partial \mathbf{p}_{i,j}} \ldots \frac{\partial \mathbf{x}}{\partial \mathbf{p}_{n,m}}\right]. \tag{4}$$

This formulation is a simplification of the full D-NURBS described in [Terzopoulos and Qin 1994] the weights of the NURBS are fixed throughout the simulation. Permitting the dynamics to modify weights would likely improve the expressiveness of the kinematics, but we find fixing the weights affords sufficiently expressive results and improves performance. The performance improvement is a result of having fixed $N_{i,j}(u,v)$ entries over the course of the simulation. The full D-NURBS model requires rebuilding the jacobian and consequently the mass matrices at each time step.

4.2 Generalized Coordinates for a NURBS model

The above formulation describes developing the map from a single (u,v) coordinate to its corresponding position on the surface. In

our simulation we represent the entire boundary of the NURBS model, so we require samples along each surface. We emphasize that we only require an initial selection of (u, v) coordinates across each surface and the dynamics of the simulation modifies the set of control points $\mathbf{q}$ to modify the world space maps $\mathbf{x}(u, v) = \mathbf{J}(u, v)\mathbf{q}$.

Let's say for a NURBS surface on the model the DOF column vector is $\mathbf{q}_i$, which is the subset of control points in the generalized coordinates for the i-th surface. For a given surface with m pairs of $\mathbf{u} = (u, v)$ coordinates sampled in parametric space, we write the jacobian for the i-th surface as (should I use something like $\hat{\mathbf{J}}$ instead?)

$$\mathbf{J}_i = \left[\mathbf{J}(\mathbf{u})_1^T \dots \mathbf{J}(\mathbf{u})_m^T\right]^T. \tag{5}$$

Then if $\mathbf{x}_i$ is the set of m world space positions for the i-th surface, we may write $\mathbf{x}_i = \mathbf{J}_i\mathbf{q}_i$. Finally, given the full set of generalized coordinates $\mathbf{q} = \left[\mathbf{q}_i^T, \dots, \mathbf{q}_n^T\right]^T$ with n surfaces, we may write jacobian $\mathbf{J}$ for the entire model as a block diagonal matrix:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & & & & \\ & \ddots & & & \\ & & \mathbf{J}_i & & \\ & & & \ddots & \\ & & & & \mathbf{J}_n \end{bmatrix}. \tag{6}$$

With this equation, the full set of world positions of the NURBS model may be written as $\mathbf{x} = \mathbf{J}\mathbf{q} \in \mathbb{R}^{3N}$ where N is the total number of (u, v) coordinates.

5 SHAPE MATCHING

The key feature of our method is its unique deformation map $\phi(\mathbf{X}) = \mathbf{x}$ mapping some arbitrary undeformed material point $\mathbf{X}$ to its corresponding deformed position in space $\mathbf{x}$. We develop this similarly to the VEM approach in that we say the function we intend to reproduce may be some highly nonlinear one, but it may be approximated accurately by a polynomial. Thus we seek some ϕ function where $\phi^* \approx \phi$ where ϕ^* is the true solution. In 1-D this means we wish to find a function of the form $\phi(X) = c_1 + c_2X + c_3X^2 + \dots + c_nX^{n-1}$ where each c_i is an unknown polynomial coefficient and the X^i terms are elements of the polynomial basis.

5.1 The Shape Matching Element

In typical FEM in graphics, we make use of tetrahedral or triangular elements with a piecewise-linear function space. In such a case our functions may be evaluated as a weighted combination of the nodal values of an element: $f(\mathbf{x}) = \sum_{i=0}^{n} f_i\phi_i(\mathbf{x})$ where n is the number of nodal values and ϕ_i is the i-th barycentric coordinates. Thus in this case we have the simple deformation map $\mathbf{x}(\mathbf{X}) = \sum_{i=0}^{n} \mathbf{x}_i\phi_i(\mathbf{X})$. However, our method contrasts this typical approach in that our elements are more general and take the form of any arbitrary boundary primitive, such as NURBS, and also that the function space in non-linear.

In our algorithm, a single shape matching element is represented by a NURBS surface, a center of mass, and polynomial describing its current deformation. We construct these polynomials via a shape matching strategy. With the addition of the center of mass, we

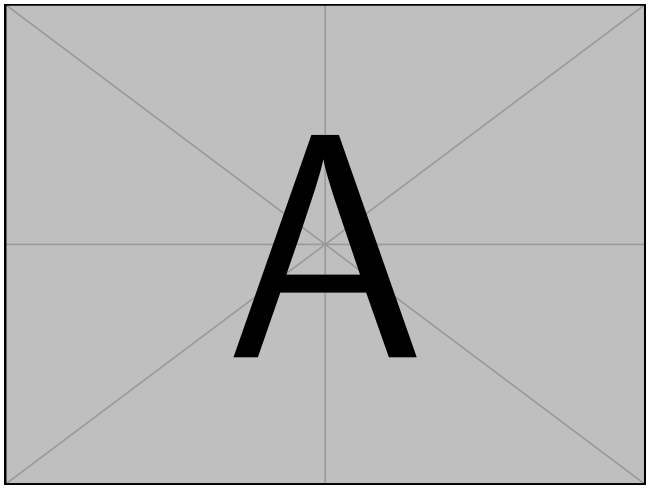

Figure 2: Figure showing shape matching to NURBS

describe the shapes deformation about this point and thus the monomial basis in the case of a linear polynomial is $\left[\mathbf{1}^T(\mathbf{X} - \bar{\mathbf{X}})^T\right]$ where $\mathbf{1}^T$ is a constant 1 column vector and $\bar{\mathbf{X}}$ is the center of mass for the element. To simplify the following formulas, we collect the non-constant terms of the monomial basis into a single vector $\mathbf{m}(\mathbf{X}-\bar{\mathbf{X}})$. For example if $\mathbf{X} = (X, Y, Z)$, $\bar{\mathbf{X}} = (\bar{X}, \bar{Y}, \bar{Z})$, and $\mathbf{q} = \mathbf{X}-\bar{\mathbf{X}}$ with a quadratic polynomial we would have

$$m(\mathbf{q}) = (q_1, q_2, q_3, q_1^2, q_2^2, q_3^3, q_1q_2, q_2q_3) \in \mathbb{R}^k, \tag{7}$$

where k is the size of the monomial basis. Next we arrange this basis into a matrix of the form

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} \mathbf{m}(\mathbf{q}) & & \\ & \mathbf{m}(\mathbf{q}) & \\ & & \mathbf{m}(\mathbf{q}) \end{bmatrix} \in \mathbb{R}^{3x3k}. \tag{8}$$

With this definition of the monomial basis, we may reconstruct the deformed position $\mathbf{x}$ for some material point $\mathbf{X}$ with

$$\mathbf{x}(\mathbf{X}) = \begin{bmatrix}\mathbf{M}(\mathbf{X} - \bar{\mathbf{X}}) & \mathbf{I}\end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{t} \end{bmatrix}, \tag{9}$$

where $\mathbf{c} \in \mathbb{R}^{3k}$ is the set of polynomial coefficients for the non-constant terms, $\mathbf{t} \in \mathbb{R}^3$ is the constant term coefficients, and $\mathbf{I} \in \mathbb{R}^{3x3}$ is an identity matrix. The reason for separating out the constant term becomes more evident when we examine case where we have multiple elements.

The remaining step in defining a single element is to describe how to solve for the polynomial coefficients. For a single element, we have n material points $(\mathbf{X}_1 \dots \mathbf{X}_n)$ and their corresponding deformed positions $(\mathbf{x}_1^* \dots \mathbf{x}_n^*)$. As these elements are defined on a boundary, these are both known throughout the simulation. Therefore we may form a fitting problem to solve for the polynomial coefficients that minimize the error $\sum_i^n ||\mathbf{x}_i^* - \mathbf{X}_i||^2$. We minimize

this error in a least squares sense where we first form the system

$$\begin{bmatrix} \mathbf{M}(\mathbf{X_1}-\bar{\mathbf{X}}) & \mathbf{I} \\ \vdots & \\ \mathbf{M}(\mathbf{X_i}-\bar{\mathbf{X}}) & \mathbf{I} \\ \vdots & \\ \mathbf{M}(\mathbf{X_n}-\bar{\mathbf{X}}) & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{t} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^* \\ \vdots \\ \mathbf{x}_i^* \\ \vdots \\ \mathbf{x}_n^* \end{bmatrix}, \tag{10}$$

and then in the more compact form

$$[\mathbf{A} \quad \mathbf{S}] \begin{bmatrix} \mathbf{c} \\ \mathbf{t} \end{bmatrix} = \mathbf{b}, \tag{11}$$

where $\mathbf{A}$ is the block containing the monomial matrices of the system, and $\mathbf{S}$ is the block of identity matrices. Next, to reduce the degrees of freedom of this system we let the deformed center of mass (the constant term coefficients), t, be equal to the mean of the deformed positions influenced by this center of mass. In the case of one element we have $t = \frac{1}{n}\sum_i^n x_i^*$. We describe the details on center of mass selection in section insert here. In matrix form we have

$$t = \underbrace{\begin{bmatrix} \frac{1}{n} & & \cdots \frac{1}{n} & & \\ & \frac{1}{n} & \cdots & \frac{1}{n} & \\ & & \frac{1}{n} \cdots & & \frac{1}{n} \end{bmatrix}}_{T} \mathbf{b}. \tag{12}$$

(spacing on this looks bad....). Finally, this leaves us with the system

$$\mathbf{Ac} = (\mathbf{I} - \mathbf{ST})\mathbf{b}. \tag{13}$$

Like in the original Shape Matching [Müller et al. 2005], we solve for the coefficients by minimizing the error in a least squares sense, resulting in

$$\begin{aligned} \mathbf{c} &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T(\mathbf{I}-\mathbf{ST})\mathbf{b} \\ &= \mathbf{Lb} \end{aligned}. \tag{14}$$

This $\mathbf{L}$ matrix thus gives us our file ingredient in the element, coefficients defining the polynomial. We emphasize that $\mathbf{L}$ is constant throughout the simulation, therefore as $\mathbf{b}$ varies in the simulation, $\mathbf{Lb}$ gives us a new set of polynomial coefficients as $\mathbf{b}$ changes.

5.2 Shape Matching with Multiple Elements

In a typical simulation we will have multiple shape matching elements where the boundary of each element is defined by a different NURBS surface. Therefore for the entire model each shape has a boundary, a polynomial describing its deformation, as well as an associated center of mass. These centers of mass may not be distinct, meaning that a single center of mass may be shared among multiple surfaces. This sharing of center of masses is typical and in some cases required (reference center of mass section that describes how for planar surfaces we need to share COM to get a COM out of plane). This leads to a new definition of the center of mass where it is the mean of all the boundary points associated with it.

In defining the shape matching fitting problem, we will form a system similar to equation 11. In this case where we have n shape elements our set of unknown coefficients is $\hat{\mathbf{c}} = \begin{bmatrix}\mathbf{c}_1^T \dots \mathbf{c}_i^T \dots \mathbf{c}_n^T\end{bmatrix}^T \in \mathbb{R}^{3kn}$ where $\mathbf{c}_i$ corresponds to the unknown polynomial coefficients for the i-th shape element. The set of boundary positions is similarly written as $\hat{\mathbf{b}} = \begin{bmatrix}\mathbf{b}_1^T \dots \mathbf{b}_i^T \dots \mathbf{b}_n^T\end{bmatrix}^T \in \mathbb{R}^{3N}$ where N is the total number of boundary points across all elements. If we have m centers of mass we write the set of polynomial constant terms as $\hat{\mathbf{t}} = \begin{bmatrix}\mathbf{t}_1^T \dots \mathbf{t}_i^T \dots \mathbf{t}_m^T\end{bmatrix}^T \in \mathbb{R}^{3m}$. With these terms, we write the polynomial fitting system for the entire model as

$$\begin{pmatrix} \mathbf{A}_1 & & & & \mathbf{S}_1 \\ & \ddots & & & \vdots \\ & & \mathbf{A}_i & & \mathbf{S}_i \\ & & & \ddots & \vdots \\ & & & \mathbf{A}_n & \mathbf{S}_n \end{pmatrix} \begin{bmatrix} \hat{\mathbf{c}} \\ \hat{\mathbf{t}} \end{bmatrix} = \hat{\mathbf{b}}, \tag{15}$$

where $\mathbf{A}_i$ is the monomial basis matrix for the i-th shape element, and $\mathbf{S}_i$ is a selection matrix that extracts the j-th center of mass which is associated with the i-th shape. We simplify this as

$$\begin{pmatrix} \hat{\mathbf{A}} & \hat{\mathbf{S}} \end{pmatrix} \begin{bmatrix} \hat{\mathbf{c}} \\ \hat{\mathbf{t}} \end{bmatrix} = \hat{\mathbf{b}}. \tag{16}$$

Like in the single element case, we set the $\hat{\mathbf{t}}$ to be equal to the mean of a set of boundary positions, but in this case each $\mathbf{t}_j$ is equal to the mean of only the boundary positions associated with the j-th center of mass. The definition of the values of $\hat{\mathbf{t}}$ is similar to 12:

$$\hat{\mathbf{t}} = \underbrace{\begin{bmatrix} \mathbf{T}_1 \\ \vdots \\ \mathbf{T_m} \end{bmatrix}}_{\hat{\mathbf{T}}}, \tag{17}$$

where the entries each $\mathbf{T}_i \in \mathbb{R}^{3N}$ take a similar form, but are only nonzero for columns corresponding to associated boundary vertices.

Finally, to solve for the full set of non-constant term coefficients, $\hat{\mathbf{c}}$, we solve in the same manner as equation 14 where we have

$$\begin{aligned} \hat{\mathbf{c}} &= (\hat{\mathbf{A}}^T\hat{\mathbf{A}})^{-1}\hat{\mathbf{A}}^T(\mathbf{I}-\hat{\mathbf{S}}\hat{\mathbf{T}})\hat{\mathbf{b}} \\ &= \mathbf{L}\hat{\mathbf{b}} \end{aligned}. \tag{18}$$

5.3 Hierarchical fitting

A problem may emerge in the shape matching process where the system may be underdetermined. To account for this we solve for the coefficients in a hierarchical manner where we initially solve for the linear terms and at each step solve the coefficients for the next higher order of deformation. The produces a set of smaller systems which are guaranteed to be determined probably need to justify this more. The intuition behind this can be described by the *hierarchical ordering principle*, which argues that the behavior of a system is typically dominated by lower order effects cite something here.. To describe our fitting strategy, we begin with the following shape matching system for a single element

$$[\mathbf{A} \quad \mathbf{S}] \begin{bmatrix} \mathbf{c} \\ \mathbf{t} \end{bmatrix} = \mathbf{b}. \tag{19}$$

We will show the hierarchical fitting on a polynomial with a set of coefficients for the linear terms, $\mathbf{c}_{\text{lin}}$, and the coefficients for the quadratic terms $\mathbf{c}_{\text{quad}}$. Similarly, we also have $\mathbf{A}_{\text{lin}}$ and $\mathbf{A}_{\text{quad}}$ which are the blocks of $\mathbf{A}$ corresponding to the columns for the linear

and quadratic monomial terms, respectively. First we construct a system for the linear terms:

$$[\mathbf{A}_{\text{lin}} \quad \mathbf{S}] \begin{bmatrix} \mathbf{c}_{\text{lin}} \\ \mathbf{t} \end{bmatrix} = \mathbf{b}, \tag{20}$$

which leads to the following expression for the linear coefficients

$$\begin{aligned} \mathbf{c}_{\text{lin}} &= (\mathbf{A}_{\text{lin}}^T \mathbf{A}_{\text{lin}})^{-1} \mathbf{A}_{\text{lin}}^T (\mathbf{I} - \mathbf{ST})\mathbf{b} \\ &= \mathbf{L}_{\text{lin}} \mathbf{b} \end{aligned}. \tag{21}$$

where $\mathbf{T}$ is defined in equation 12. For the quadratic system, we subtract linear term estimate of the solution and solve for the coefficients that fit this residual:

$$\begin{bmatrix} \mathbf{A}_{\text{quad}} & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{c}_{\text{quad}} \\ \mathbf{t} \end{bmatrix} = \mathbf{b} - \mathbf{A}_{\text{lin}} \mathbf{c}_{\text{lin}}. \tag{22}$$

Using the solution for the linear coefficients in 21, we arrive at the solution for the quadratic coefficients:

$$\begin{aligned} \mathbf{c}_{\text{quad}} &= (\mathbf{A}_{\text{quad}}^T \mathbf{A}_{\text{quad}})^{-1} \mathbf{A}_{\text{quad}}^T ((\mathbf{I} - \mathbf{ST})\mathbf{b} - \mathbf{A}_{\text{lin}} \mathbf{c}_{\text{lin}}) \\ &= (\mathbf{A}_{\text{quad}}^T \mathbf{A}_{\text{quad}})^{-1} \mathbf{A}_{\text{quad}}^T (\mathbf{I} - \mathbf{ST} - \mathbf{A}_{\text{lin}} \mathbf{L}_{\text{lin}})\mathbf{b} \\ &= \mathbf{L}_{\text{quad}} \mathbf{b} \end{aligned}. \tag{23}$$

The final form for the full $\mathbf{L}$ matrix is thus given by

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{\text{lin}} \\ \mathbf{L}_{\text{quad}} \\ \mathbf{T} \end{bmatrix} \tag{24}$$

The recursive nature of this process permits a more general description for arbitrary orders, which we write in algorithm TODO.

6 THE DEFORMATION MAP

At this point we have shown how to fit a polynomial describing the deformation of each surface. With these polynomial coefficients coupled with their associated centers of mass, we can construct an estimate of some arbitrary material point $\mathbf{X}$ using the deformation described by the surface. This lends itself to a general definition for the deformed position of a point as a weighted sum of the polynomials, which yields a unique polynomial for the point. Thus we arrive at a definition for the deformed position, $\mathbf{x}$, of a material point $\mathbf{X}$ in the undeformed space:

$$\mathbf{x}(\mathbf{X}) = \sum_i^n w_i(\mathbf{X}) \left[\mathbf{M}(\mathbf{X} - \bar{\mathbf{X}}_j)\mathbf{c}_i + \mathbf{t}_j \right], \tag{25}$$

where we have n surfaces, and w_i and $\mathbf{c}_i$ are the weights and the coefficients for the i-th surface, respectively. The terms $\bar{\mathbf{X}}_j$ and $\mathbf{t}_j$ the undeformed and deformed centers of mass, respectively, that are associated with the i-th surface.

6.1 Computing the Blending Weights

All that remains for the deformation map is to compute blending weights. Our choice of the weighting function is guided by the simple idea that that for a given material point its weight should be highest to whichever surfaces it is closest. This idea lends itself to a simple distance-based weighting method. The first step in this process is to compute *distance weights*. For a given point $\mathbf{X}$ and a set of n we wish to compute a set of distance weights to each surface $\theta(\mathbf{X}) = (\theta_1, \ldots, \theta_i, \ldots, \theta_n)$.

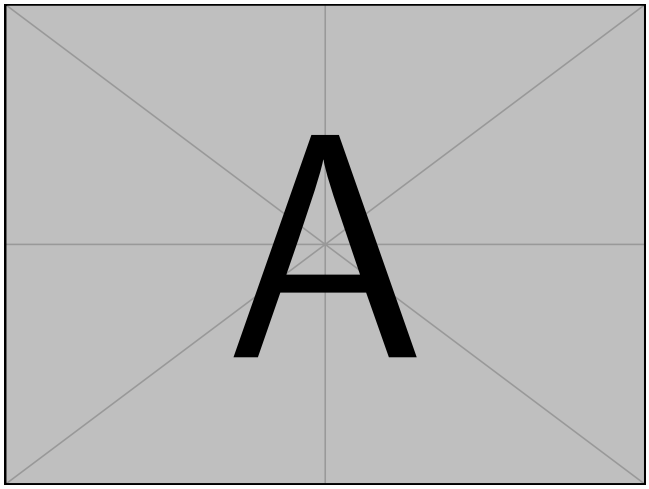

Figure 3: Figure showing how weight calculation works (left) ray casting plus result (right) after correcting for partition of unity

For the i-th surface, we first find the distance, d_{primary}, from the point $\mathbf{X}$ to the closest point, $\mathbf{p}$, on the surface. Next we wish to satisfy the condition that the distance weight is highest on the surface and it decays towards zero as it approaches other surfaces. To achieve this, we shoot a ray from $\mathbf{p}$ towards $\mathbf{X}$ and check for intersections between this ray and the other $n-1$ surfaces. If this ray hits another surface, we let d_{total} be the distance from $\mathbf{p}$ to the closest ray hit $\mathbf{q}$. These two distances enable us to write a simple linear function to compute the *distance weight*:

$$\theta_i = \max(1.0 - \frac{d_{\text{primary}}}{\min(d_{\text{total}}, D)}, 0.0), \tag{26}$$

where D is a fixed cutoff distance parameter. We see that this distance weight is the result of interpolating along this ray from $\mathbf{p}$ to $\mathbf{q}$ (or a fixed point that depends on D). Importantly, this gives a result where the distance weight to surface i is equal to 1 if it lies on i and 0 if it lies on another surface. More elaborate schemes may be developed for computing distances, such as using the Boundary Element Method, but we find this simple linear function to perform well for our purposes.

Now that we have a set of distances weights **theta**, we seek to form a final set of *blending weights* $\mathbf{w}(\mathbf{X}) = (w_1, \ldots, w_i, \ldots, w_n)$ to blend the polynomials of the n shape elements. In constructing these blending weights, we require that they adhere to a set of conditions. We require that (1) each weight is non-negative, (2) the sum of the weights is equal to 1 (*partition of unity*), (3) and that if a point lies on a surface i, it will have a weight of 1 to i and consequently zero to the others. To produce these blending weights we form a quadratic program that satisfies these constraints:

$$\begin{aligned} \mathbf{w}(\mathbf{X}) = \min_{\mathbf{w}} \quad & \mathbf{w}^T \Theta(\mathbf{X}) \mathbf{w} \\ \text{s.t.} \quad & 0 \le w_i \le 1 \\ & \sum_i^n w_i = 1 \end{aligned} \tag{27}$$

where

$$\Theta(\mathbf{X}) = \text{diag}\left(\frac{1}{\theta_1}, \frac{1}{\theta_2}, \ldots, \frac{1}{\theta_n}\right) \tag{28}$$

We see that through the use of linear constraints we satisfy conditions (1) and (2). Satisfying the third condition (3) is achieved through using the distance weights. Intuitively, we see that for some small θ_i, $\frac{1}{\theta_i}$ will be very large, encouraging the quadratic program to assign a small weight for w_i. Conversely, we see the opposite effect for large values of θ_i.

7 SELECTING CENTERS OF MASS

TODO. waiting to implement this before I start writing how we do it

8 RAYCASTING QUADRATURE

The last step before we may discuss the dynamic simulation is to describe our strategy for generating material points $\mathbf{X}$ in the undeformed model. Our volume integration strategy is based on [Khosravifard and Hematiyan 2010] which we briefly describe in the following section

8.1 Review of the Meshless Method for Domain Integral Evaluation in [Khosravifard and Hematiyan 2010]

This paper addresses the integration of volume integrals of the form $\mathbf{I} = \int_\Omega f(\mathbf{x}) d\Omega$ where $\mathbf{x} = (x_1, x_2, x_3)$. The authors show that through application of the Green-Gauss theorem, this integral may be rewritten as 2D integral over a yz plane where at each position along the plane we evaluate a line integral along the x-axis. To evaluate these line integrals, a raycasting procedure is performed to find a set of intersection along the line. These intersection define integration domains in which we sample 3D integration points. As a result of integrating over the 2D plane and raycasting at each 2D point, we produce an integration rule over the entire domain that can converge to the true integral value.

In our approach we modify the raycasting procedure by allowing integration over intersecting NURBS surfaces. It is common in NURBS modeling to produce models in which many surfaces overlap. To account for this we augment the raycasting intersection step to use an approach common in Constructive Solid Geometry (CSG). For the set of intersection intervals, we instead take the union over all intervals to yield a new set of intervals handling intersecting solids. We find that this extension serves as a simple and practical approach to support inexact models (not sure what to right here). We note that this solution is not perfect and making this intersection step more robust serves as future work.

9 DYNAMIC SIMULATION

With our definition of the deformation map as well as a strategy for integrating the volume, we now have the necessary ingredient to perform dynamic simulation. We follow the Variational Mechanics approach that defines a pair of kinetic and potential energies, and we apply the Euler-Lagrange equation to develop our equations of motion.

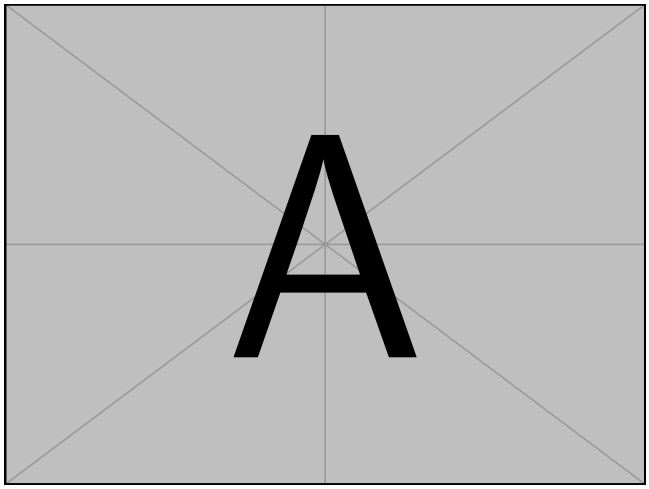

Figure 4: Figure showing how raycasting quadrature works (top), convergence plot for volume integration (bottom

Before we define these energies we first would like to simplify the form of the deformation map presented in equation 25. We may write $\mathbf{x}(\mathbf{X})$ in matrix form as

$$\mathbf{x}(\mathbf{X}) = \underbrace{\left[w_1\mathbf{M}(\mathbf{X}-\bar{\mathbf{X}}_j)\ldots w_n\mathbf{M}(\mathbf{X}-\bar{\mathbf{X}}_j)\middle|\sum_i^{n_1} w_1^i\mathbf{I}\cdots\sum_i^{n_m} w_m^i\mathbf{I}\right]}_{\mathbf{Y}(\mathbf{X})}\begin{bmatrix}\hat{\mathbf{c}}\\ \hat{\mathbf{t}}\end{bmatrix}, \tag{29}$$

I'm not sure how to write this matrix form for x(X), especially on finding the best notation for the center of mass indexing. So this above equation is temporary. where we have n shape elements, m centers of mass. We simplify the above equation and use the following form

$$\mathbf{x}(\mathbf{X}) = \mathbf{Y}(\mathbf{X})\mathbf{L}\hat{\mathbf{b}}, \tag{30}$$

where we make use of equation 18 to rewrite this in terms of the boundary points $\hat{\mathbf{b}}$. As one final step, we rewrite this again in terms of the control points of our NURBS surfaces:

$$\mathbf{x}(\mathbf{X}) = \mathbf{Y}(\mathbf{X})\mathbf{L}\mathbf{J}\mathbf{q}, \tag{31}$$

where we use the fact that $\hat{\mathbf{b}} = \mathbf{J}\mathbf{q}$. not a fan of this, perhaps we change $\hat{b}$ to something else... This gives us an expression for the deformed position in terms of our generalized coordinates, $\mathbf{q}$.

9.1 Generalized Inertia

The kinetic energy for a model may be expressed as

$$T = \frac{1}{2}\int_\Omega \rho\dot{\mathbf{x}}^T\dot{\mathbf{x}}d\Omega, \tag{32}$$

where Ω is the 3D integration domain, ρ is the density, and $\dot{\mathbf{x}}$ is the velocity for some point in the domain. Using the deformation map

defined in 31, we may rewrite this kinetic energy as

$$\begin{aligned} T &= \frac{1}{2}\int_\Omega \rho \dot{\mathbf{q}}^T \mathbf{J}^T \mathbf{Y}(\mathbf{X})^T \mathbf{Y}(\mathbf{X})\, d\Omega \\ &= \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{J}^T \left[\int_\Omega \rho \mathbf{Y}(\mathbf{X})^T \mathbf{Y}(\mathbf{X}) d\Omega\right] \mathbf{J}\dot{\mathbf{q}}. \\ &= \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}\dot{\mathbf{q}} \end{aligned} \tag{33}$$

The second line is a result of the fact that all terms except for $\mathbf{Y}(\mathbf{X})$ are spatially constant, allowing us to move these outside the volume integral, which enables fast assembly of the mass matrix, $\mathbf{M}$.

9.2 Error Energy

Over the course of the simulation we may see the true boundary values deviate from their positions estimated by the polynomial. This is most notable in cases where the degree of the polynomial on a NURBS is higher than the degree of the B-spline surface. To address this we augment our kinetic energy with a term to account for this error. The error for a boundary point $\mathbf{x}_i^*$ that is known to lie on the limit surface of the NURBS can be written as

$$\begin{aligned} \epsilon_i &= ||\mathbf{x}(\mathbf{X}_i) - \mathbf{x}_i^*|| \\ &= ||(\mathbf{Y}(\mathbf{X}_i)\mathbf{L} - \mathbf{S_i})\hat{\mathbf{b}}||' \end{aligned} \tag{34}$$

where $\mathbf{S_i}$ extracts $\mathbf{x}_i^*$ from $\hat{\mathbf{b}}$. We modify this error further to be expressed in terms of the control points. We note that $\mathbf{x}^*(u,v)$ yields the position on the NURBS surface whereas $\mathbf{x}(\mathbf{X})$ is the deformed value estimated by the polynomial fitting. Furthermore $\mathbf{X}$ may be written as a function of the parametric coordinates $\mathbf{X}(u,v)$ as its position on the surface is produced using the initial set of control points representing the undeformed model. This leads to an energy potential over the NURBS surface in parametric space

$$T_E = \iint (\mathbf{x}(\mathbf{X}(u,v)) - \mathbf{x}^*(u,v))^T (\mathbf{x}(\mathbf{X}(u,v)) - \mathbf{x}^*(u,v)) du dv. \tag{35}$$

The discrete form of this written in terms of the control points may be written as

$$\begin{aligned} T_E &= \sum_i^N \dot{\mathbf{q}}^T \mathbf{J}^T (\mathbf{Y}(\mathbf{X}_i)\mathbf{L} - \mathbf{S_i})^T (\mathbf{Y}(\mathbf{X}_i)\mathbf{L} - \mathbf{S_i})\mathbf{J}\dot{\mathbf{q}} \\ &= \dot{\mathbf{q}}^T \mathbf{J}^T \left[\sum_i^N (\mathbf{Y}(\mathbf{X}_i)\mathbf{L} - \mathbf{S_i})^T (\mathbf{Y}(\mathbf{X}_i)\mathbf{L} - \mathbf{S_i})\right] \mathbf{J}\dot{\mathbf{q}}' \\ &= \dot{\mathbf{q}}^T \mathbf{M}_E \dot{\mathbf{q}} \end{aligned} \tag{36}$$

where N is the total number of boundary points and $\mathbf{M}_E$ is our *error mass matrix*.

9.3 Generalized Forces

We write our potential energy as

$$V = \int_\Omega \psi(\mathbf{F}(\mathbf{X})) d\Omega. \tag{37}$$

To evaluate this energy (as well as its derivatives), we require a definition for the deformation gradient, $\mathbf{F}(\mathbf{X})$:

$$\mathbf{F}(\mathbf{X}) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \frac{\partial \mathbf{Y}(\mathbf{X})}{\partial \mathbf{X}} \mathbf{L}\hat{\mathbf{b}}. \tag{38}$$

In the expression for the deformed position $\mathbf{x}(\mathbf{X})$ we see that the dependence on $\mathbf{X}$ only exists in $\mathbf{Y}(\mathbf{X})$, making $\mathbf{L}$ and $\hat{\mathbf{b}}$ constant

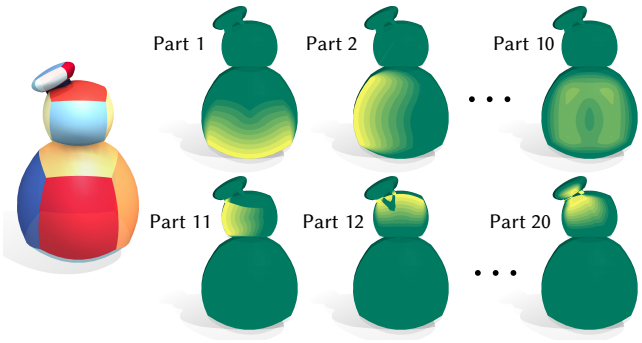

Figure 5: Our distance weights decay smoothly from 1.0 (yellow) to 0.0 (green) when moving away from its closest surface. Here we visualize the distribution of the distance weights (with cutoff distance 5.0) corresponding to each part.

terms when compute $\mathbf{F}$. Full details on how to compute $\mathbf{F}$ and its derivative $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ may be found in the appendix. (I just wrote this because I'm too tired to right the d etails right now :p. Now that we have an expression for the deformation gradient, we discretize our potential energy as follows

$$V = \sum_i^m \psi(\mathbf{F}(\mathbf{X}_i) A_i, \tag{39}$$

where m is the number of integration points and A_i is the volume for the i-th integration point.

Like in the case of the kinetic energy, we also introduce an error force potential, which takes the simple form

$$V_E = \mathbf{q}^T \mathbf{M}_E \mathbf{q} \tag{40}$$

9.4 Time Integration

We have now provided all the necessary ingredient to produce equations of motion for deformable bodies, which we derive via Euler-Lagrange equation. We begin with the Lagrangian

$$L = (T + T_E) - (V + V_E) \tag{41}$$

Then from the Euler-Lagrange equation we have:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{p}}} + \frac{\partial L}{\partial \mathbf{p}} = 0 \tag{42}$$

todo: not totally sure what to write here since we're experimenting with both linearly implicit backward euler as well as Newton's method soon

10 RESULTS

Dave: we should show the nurbs model in rhino/fusion for every example, maybe an exploded view as well

REFERENCES

Martin Aigner, Christoph Heinrich, Bert Jüttler, Elisabeth Pilgerstorfer, Bernd Simeon, and Anh-Vu Vuong. 2009. Swept Volume Parameterization for Isogeometric Analysis. 19–44. https://doi.org/10.1007/978-3-642-03596-8_2

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. https://doi.org/10.1142/S021820251440003X arXiv:https://doi.org/10.1142/S021820251440003X

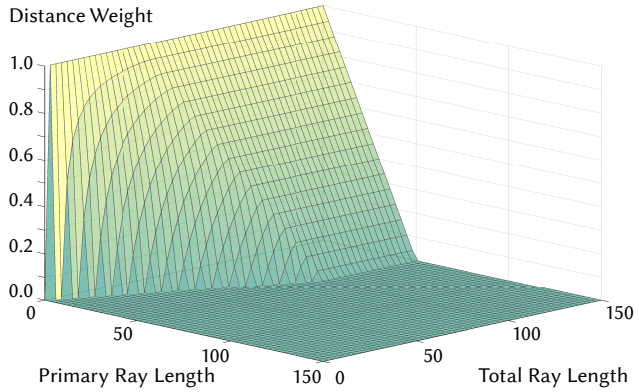

Figure 6: We plot the distance weight with respect to the primary and total ray length. Here the cutoff distance is 50.

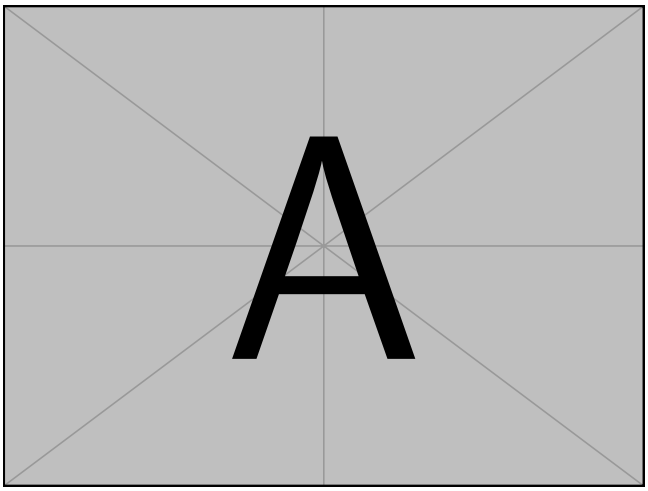

Figure 7: Take an intesting geometry (maybe that geometry processing cacture), deform it, do our shape matching and then reconstruct the deformed pose. Do this for a bunch of poses (show's kinematic model is good).

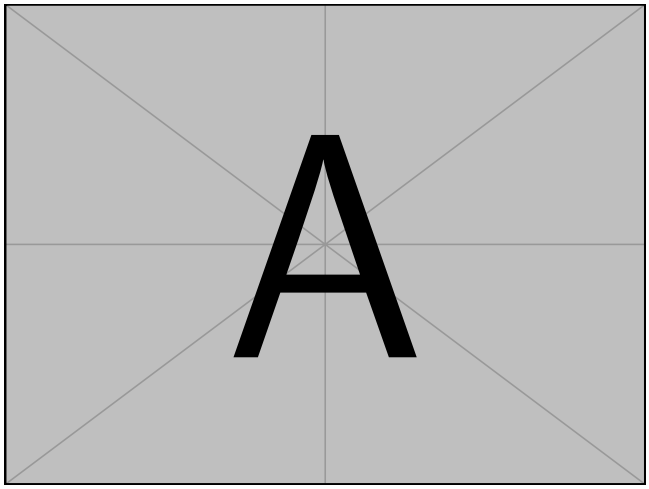

Figure 8: 2D examples (left) rigidly transform boundary and show that solving the static problem gives us rigid motion on the interior (right) isotropically scale boundary and show solving static problem gives us constant strain on the interior

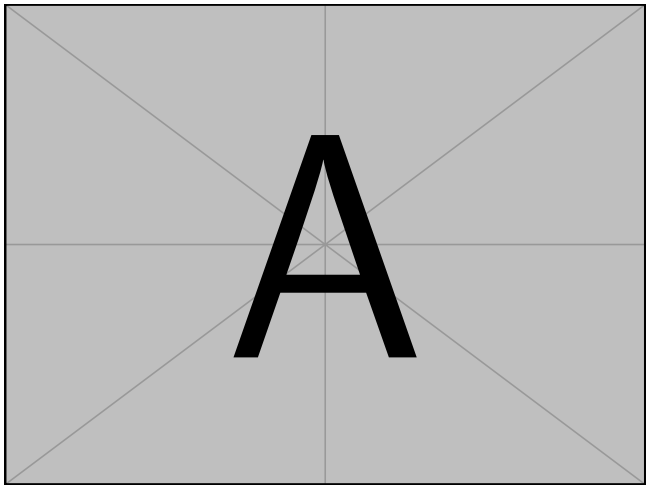

Figure 9: (left) high res FEM cantileavered bar (right) several simulations using more and more surface patches plus center of mass. Need to show beam is just unattached patches

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. https://doi.org/10.1111/j.1467-8659.2012.03171.x

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. https://doi.org/10.1145/2601097.2601116

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). https://doi.org/10.1142/S0218202505000832

J. Cottrell, Thomas Hughes, and Yuri Bazilevs. 2009. *Isogeometric Analysis: Toward integration of CAD and FEA.* https://doi.org/10.1002/9780470749081.ch7

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. https://doi.org/10.1145/3386569.3392389

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* (Vancouver, British Columbia, Canada) *(SCA '11)*. Association for Computing Machinery, New York, NY, USA, 237–246. https://doi.org/10.1145/2019406.2019438

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. https://doi.org/10.1145/2010324.1964968

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. https://doi.org/10.1145/1944846.1944855

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. https://doi.org/10.1002/nme.3120 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.3120

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. https://doi.org/10.1145/3355089.3356576

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. https://doi.org/10.

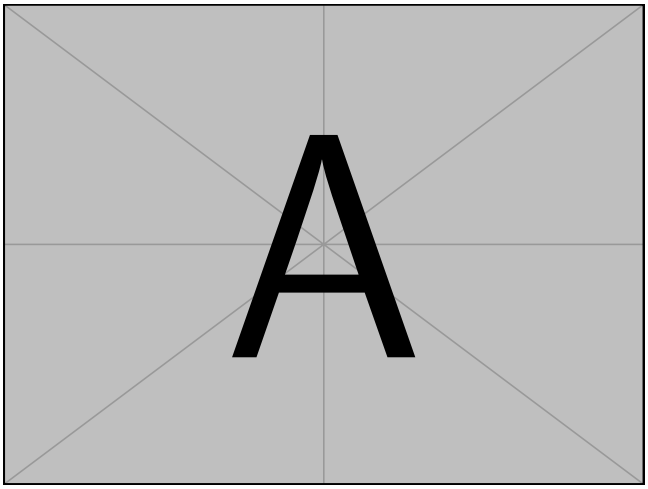

Figure 10: X shape and show arms of X move independently

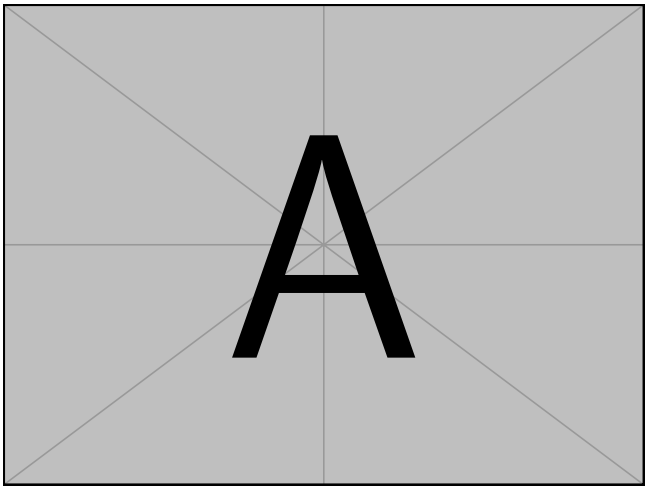

Figure 11: Coffee mug with overlapping handle (top) row of models with increasing overlap (middle) single weight image showing behaviour in the overlapping region, (bottom) simulation result for each one

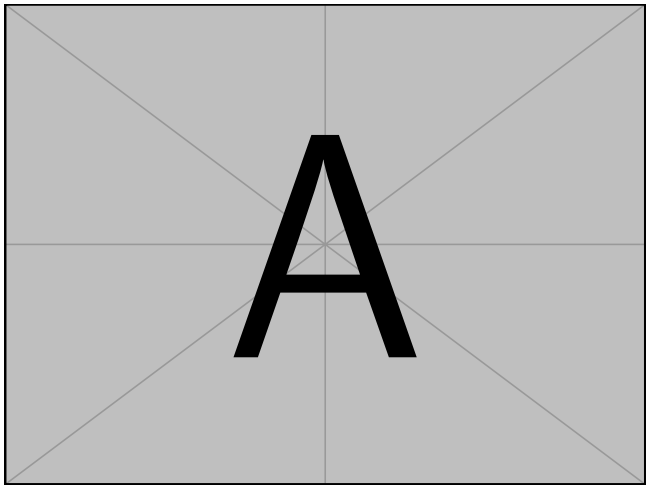

Figure 12: Big image simulating the rocket with all different combinations of things

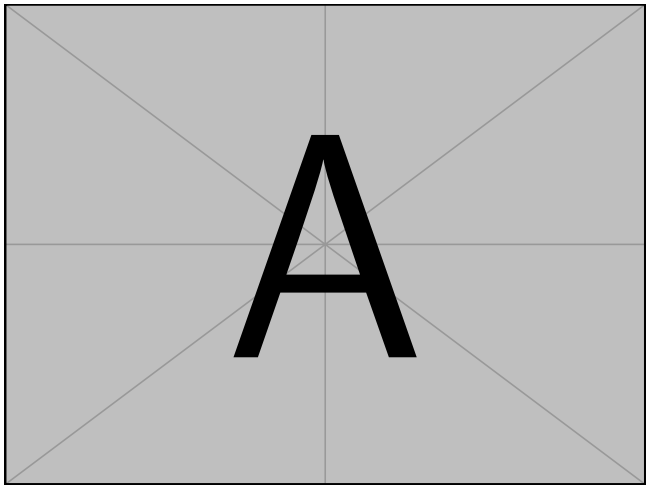

Figure 13: Big image simulating the rocket with all different combinations of things

1016/j.matcom.2012.03.008

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. https://doi.org/10.1145/3197517.3201353

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

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. https://doi.org/10.1016/j.enganabound.2009.07.008

Grégory Legrain. 2013. A NURBS enhanced extended finite element approach for unfitted CAD analysis. *Computational Mechanics* 52 (04 2013). https://doi.org/10.1007/s00466-013-0854-7

Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. 2014. Mimetic Finite Difference Method. *J. Comput. Phys.* 257 (Jan. 2014), 1163–1227. https://doi.org/10.1016/j.jcp.2013.07.031

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. https://doi.org/10.1002/fld.1650200824 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.1650200824

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. https://doi.org/10.1145/1778765.1778776

Matthias Müller and Nuttapong Chentanez. 2011. Solid Simulation with Oriented Particles. In *ACM SIGGRAPH 2011 Papers* (Vancouver, British Columbia, Canada) *(SIGGRAPH '11)*. Association for Computing Machinery, New York, NY, USA, Article 92, 10 pages. https://doi.org/10.1145/1964921.1964987

Matthias Müller, Nuttapong Chentanez, and Miles Macklin. 2016. Simulating Visual Geometry. In *Proceedings of the 9th International Conference on Motion in Games* (Burlingame, California) *(MIG '16)*. Association for Computing Machinery, New York, NY, USA, 31–38. https://doi.org/10.1145/2994258.2994260

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 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. https://doi.org/10.1145/1073204.1073216

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*

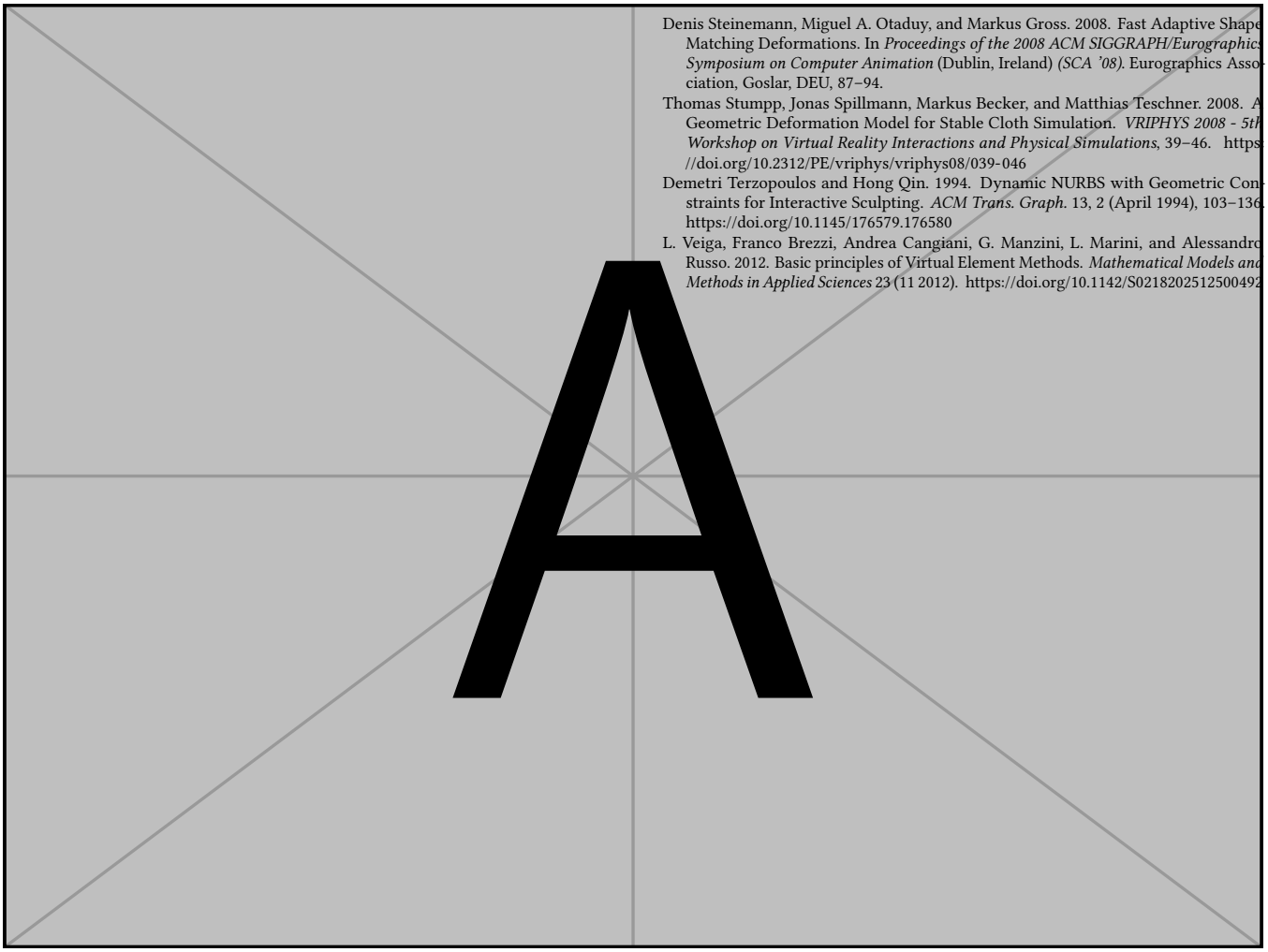

Figure 14: Avocado to show heterogenous materials

SIGGRAPH/Eurographics Symposium on Computer Animation (Grenoble, France) *(SCA '04)*. Eurographics Association, Goslar, DEU, 141–151. https://doi.org/10.1145/1028523.1028542

Alec R. Rivers and Doug L. James. 2007. FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. In *ACM SIGGRAPH 2007 Papers* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 82–es. https://doi.org/10.1145/1275808.1276480

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. https://doi.org/10.1002/nme.4852 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.4852

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. https://doi.org/10.1016/j.jcp.2016.05.004

Ruben Sevilla, Sonia Mendez, and Antonio Huerta. 2008. Nurbs-enhanced finite element method (NEFEM). *Internat. J. Numer. Methods Engrg.* 76 (10 2008), 56–83. https://doi.org/10.1002/nme.2311

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* (Los Angeles, California) *(SIGGRAPH '12)*. Association for Computing Machinery, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

Denis Steinemann, Miguel A. Otaduy, and Markus Gross. 2008. Fast Adaptive Shape Matching Deformations. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Dublin, Ireland) *(SCA '08)*. Eurographics Association, Goslar, DEU, 87–94.

Thomas Stumpp, Jonas Spillmann, Markus Becker, and Matthias Teschner. 2008. A Geometric Deformation Model for Stable Cloth Simulation. *VRIPHYS 2008 - 5th Workshop on Virtual Reality Interactions and Physical Simulations*, 39–46. https://doi.org/10.2312/PE/vriphys/vriphys08/039-046

Demetri Terzopoulos and Hong Qin. 1994. Dynamic NURBS with Geometric Constraints for Interactive Sculpting. *ACM Trans. Graph.* 13, 2 (April 1994), 103–136. https://doi.org/10.1145/176579.176580

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). https://doi.org/10.1142/S0218202512500492

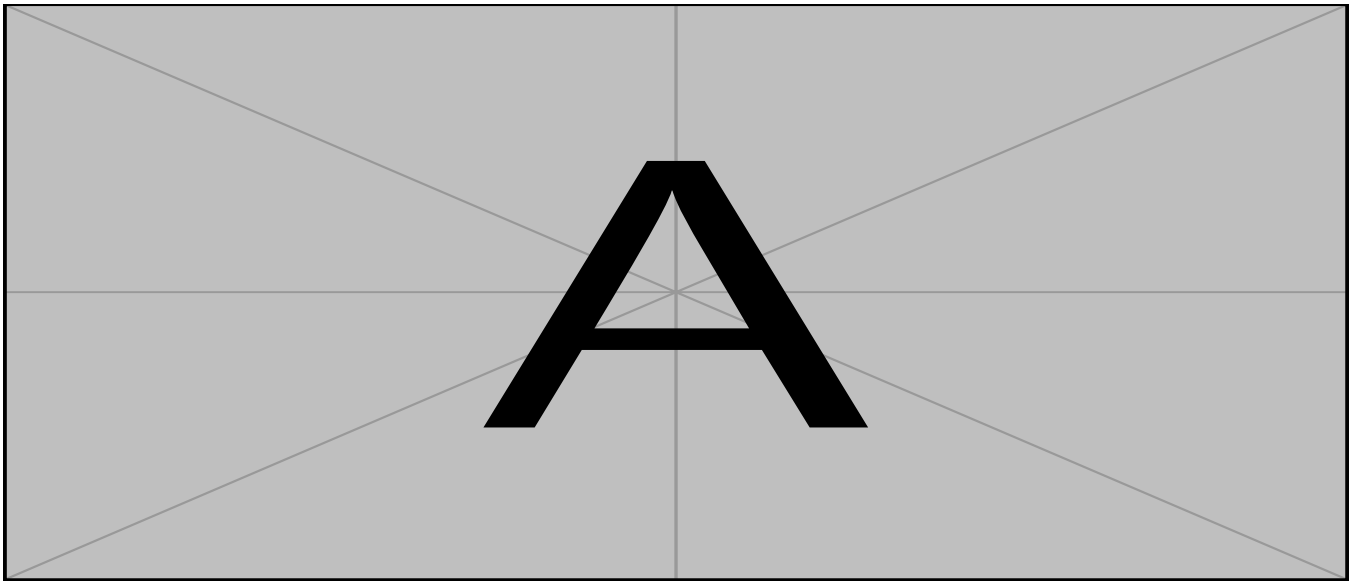

Figure 15: sequence of frames from staypuft simulation

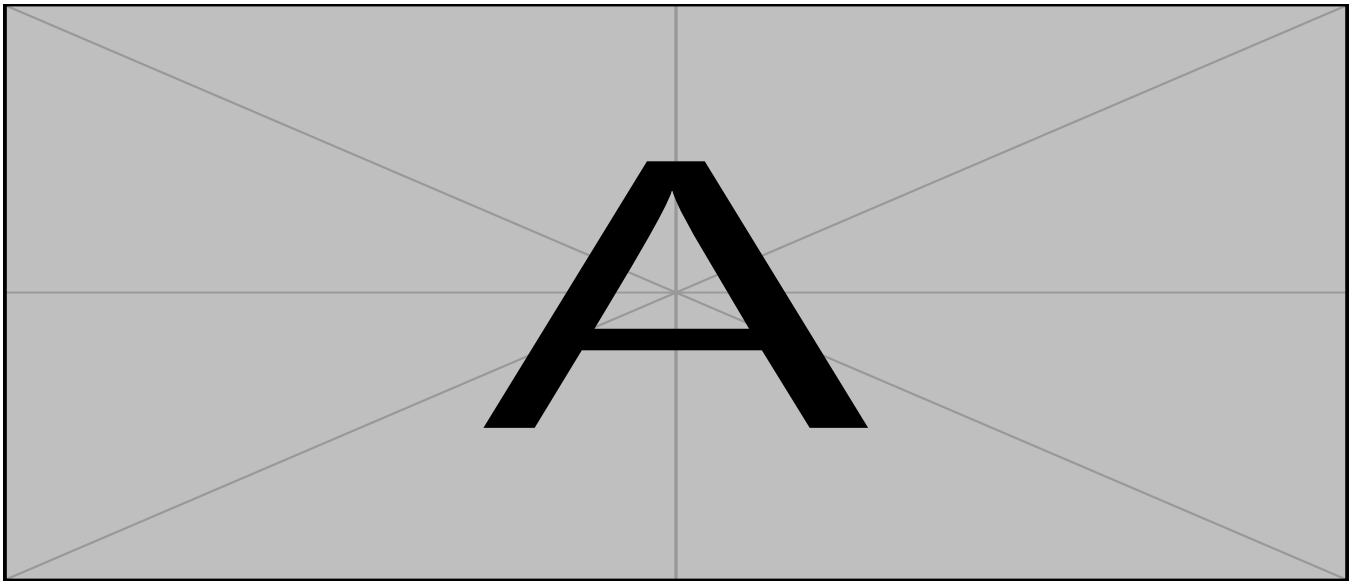

Figure 16: sequence of frames from F1 car simulation