



# Physics-based Deformation of Subdivision Surfaces for shared Virtual Worlds<sup>\*,\*\*</sup>

Andreas Riffnaller-Schiefer<sup>a,\*</sup>, Ursula H. Augsdörfer<sup>a</sup>, Dieter W. Fellner<sup>a,b</sup>

<sup>a</sup>Institute of Computer Graphics and Knowledge Visualization, Graz University of Technology, Austria

<sup>b</sup>Fraunhofer IGD, TU Darmstadt, Germany

## ARTICLE INFO

### Article history:

Received 7 November 2017

**Keywords:** subdivision surfaces, isogeometric analysis, interactive, soft-body, web service

## ABSTRACT

Creating immersive interactive virtual worlds not only requires plausible visuals, but it is also important to allow the user to interact with the virtual scene in a natural way. While rigid-body physics simulations are widely used to provide basic interaction, realistic soft-body deformations of virtual objects are challenging and therefore typically not offered in multi user environments.

We present a web service for interactive deformation which can accurately replicate real world material behavior. Its architecture is highly flexible, can be used from any web enabled client, and facilitates synchronization of computed deformations across multiple users and devices at different levels of detail.

© 2018 Elsevier B.V. All rights reserved.

## 1. Introduction

The computation of physically accurate deformations of objects in response to user interaction is usually slow, which is why accurate soft-body deformations are often not available in interactive virtual reality (VR) environments. In VR, interaction and feedback need to be immediate to create a feeling of immersion. In the real world we are used to interacting with things, e.g. to assess material properties of an object upon how it responds to interaction. Therefore, having only rigid objects in VR reduces immersion since many real world objects are easily deformed.

Many virtual worlds are accessed by multiple users with various different devices ranging from mobile devices to powerful workstations, making consistent soft-body deformations even

harder due to differences in computing and rendering capabilities.

To provide realistic interaction with objects in VR environments for a range of devices with various capabilities, a client-server approach which makes use of recent advances in subdivision based analysis is proposed. Deformations are computed on a central server and results are available to multiple clients. Subdivision surfaces, i.e. smooth surfaces derived through iterative refinement from a coarse control polygon, are used to represent the geometry and to perform simulations. Bandwidth usage is kept low by sending only the coarse control polygon, and different clients can easily render different levels of detail according to their capabilities by adjusting the number of subdivision steps performed on the control polygon.

Many existing techniques to deform surfaces, discussed in Section 2, are either not based on physical principles, hard to synchronize or too slow to allow real-time interaction. In Section 3 we explain how to compute physically accurate deformations of surfaces represented by subdivision surfaces and how to accelerate the computation to achieve interactive rates for moderately sized meshes. This simulation is used as the basis for our main contributions:

- We provide physically accurate deformation of surfaces

\*<https://doi.org/10.1016/j.cag.2017.12.005>

\*\*© 2018. This manuscript version is made available under the CC-BY-NC-ND 4.0 license <http://creativecommons.org/licenses/by-nc-nd/4.0/>

\*Corresponding author

e-mail: [a.schiefer@cgv.tugraz.at](mailto:a.schiefer@cgv.tugraz.at) (Andreas Riffnaller-Schiefer),  
[u.augsdorfer@cgv.tugraz.at](mailto:u.augsdorfer@cgv.tugraz.at) (Ursula H. Augsdörfer),  
[d.fellner@igd.fraunhofer.de](mailto:d.fellner@igd.fraunhofer.de) (Dieter W. Fellner)

through a web service. We define a client-server architecture and an HTTP API, described in Section 4, to provide analysis results to clients, ranging from web applications to high end workstations.

- Interaction with virtual objects on the client side is discussed in Section 5, where we present a new fast and simple algorithm to pick arbitrary parameter locations on a subdivision surface.

We also look at simulation of different levels of detail of a surface and discuss possible solutions to the challenge of applying consistent forces and constraints across subdivision levels.

- We demonstrate the advantages of the proposed architecture in Section 6 with several applications for different use cases. Section 7 discusses the performance of the proposed architecture and shows that it is well suited for interactive use and synchronization between multiple clients.

We conclude the paper with a discussion of limitations of the proposed architecture and possible future research directions in Section 8.

This paper is based on and extends the conference paper [1]. Here, we expand the discussion of the limitations of a linear simulation and explain what defines a *small deformation* in Section 3.4. We add Section 5.2, for a discussion on simulation of different levels of detail of a surface, and how constraints and forces can be applied consistently across different subdivision levels. Further, we expand the use cases of the proposed architecture with web related technologies in Section 6.3. Regarding the choice of solver, we provide a comparison between the CPU based solver and the CUDA based solver in Section 7.2.

## 2. Related Work

Many existing techniques allow interactive deformation of surfaces. Good overviews can be found in [2] and [3]. The different approaches differ in physical accuracy and performance.

Simple mass spring systems and mesh processing based approaches are very fast, but they provide only a rough approximation of real physics. Because such methods are not based on the actual physical behavior of real materials they are sometimes difficult to tweak to get the expected results. To overcome this, methods like [4] attempt to derive the parameters for the mass spring system from an accurate simulation of a reference model. This allows mass spring systems to approximate the behavior of more realistic methods. However, because mass spring systems and mesh processing based approaches usually work directly with the dense mesh used for rendering, synchronizing the deformation to multiple users may require a lot of bandwidth, making it impractical to perform synchronization across a network.

There are highly accurate numerical methods, for example the finite element method (FEM), to solve partial differential equations describing various physical effects. FEM requires a geometry to be expressed in a dense polygon mesh for the simulation. Typically, the *simulation mesh* does not match exactly

the representation used for rendering which makes it difficult to translate simulation results to the geometry used for rendering in virtual worlds. Additionally, the same restriction as for dense triangle meshes apply, i.e. because simulation meshes are dense, they require a lot of bandwidth for synchronization.

For *isogeometric analysis* (IGA) [5], a variant of FEM, a simulation mesh is no longer necessary. In IGA the basis function of the geometry representation are also used for representing the domain and the solution space for the analysis. Therefore, design, rendering, and the finite element simulation are based on the same representation of the geometry. The concept of IGA has been extended to many 3D representations used in modeling/CAD, including NURBS [5], T-splines [6] and subdivision surfaces [7, 8, 9]. Having the same representation for both simulation and rendering eliminates the problem of translating simulation results to objects in virtual environments. Instead, simulation results can be directly exported to be rendered in e.g. VR applications.

With some restrictions, IGA can also be used for interactive applications. In [10] physics-based surface design tools based on IGA of Catmull-Clark subdivision surfaces have been integrated into a standard 3D modelling software. The approach described in [10] requires integrating the simulation into the client application and is therefore unsuitable for clients with limited capabilities and provides no support for multiple users. However, it demonstrates IGA based surface deformation using constraints and forces.

Other approaches like e.g. [11] or [12] do support sharing deformations with multiple users by using a client-server architecture, but are limited to mass spring simulation to approximate the physical behavior respectively a single shared object for all clients. Moreover, both approaches are based on custom network protocols which cannot be accessed e.g. behind corporate firewalls or from a web based application.

We extend ideas from [10] and present a client-server architecture for interactive simulations using IGA, that provides access to the simulation for any web enabled client and supports multiple users and deformable objects.

## 3. Deformation of Subdivision Surfaces

To define smooth surfaces of arbitrary topology using just a small number of control points, a common representation used in the entertainment industry are subdivision surfaces. A subdivision surface is the smooth limit surface of a recursive refinement operation, the subdivision algorithm, infinitely applied to a coarse polygon mesh, the control mesh. A subdivision surface is fully defined by a coarse control polygon and a subdivision algorithm. Many subdivision algorithms exist. The Catmull-Clark subdivision [13] is the de-facto standard for modelling in the entertainment industry and is available in most major modelling applications. Using Catmull-Clark, each subdivision step splits all faces into  $n$  quads, where  $n$  is the number of corners in the face, by inserting new points in the middle of each face and edge [13]. In practical use this algorithm is only applied a limited number of times, until the surface appears smooth when rendered. The coarse control polygon can be efficiently sent via

a network. Each application or client can apply the subdivision algorithm to the coarse control mesh to derive a smooth subdivision surface. By adjusting the number of subdivision steps applied to the control mesh, each client can render the surface at a different level of detail (LOD), thus adapting the LOD to its capabilities.

### 3.1. Thin Shells

Typically, 3D objects in CAD or the entertainment industry are represented as surfaces, rather than volumes. To compute the response of a surface to environmental impact we perform an isogeometric *thin shell* simulation on a Catmull-Clark subdivision surface. Thin shells are structures where one dimension (the thickness) is very small compared to the other two. Such structures are very common in the automotive and aerospace industries, but also in everyday life in the form of e.g. objects made of metal sheets or thin plastic materials.

Thin shells are particularly suited for IGA because their geometry is usually defined by their middle surface, rather than a volumetric representation. This matches how such structures are typically modelled in a VR environment. Although we will interact with a range of different geometries in VR, thin shell structures cover a large array of geometries.

The control points of the subdivision surface are the only degrees of freedom (DOF) for the isogeometric thin shell simulation. They define both the initial surface geometry as well as the deformed configuration. All computations for the simulation are performed on the subdivision limit surface defined by the control points and their corresponding basis functions. The number of DOF can be easily increased without changing the surface geometry, by simply subdividing the control mesh. For regular regions of a Catmull-Clark surface, the basis functions employed to represent the design as well as the solution space, are uniform B-splines of degree 3. The basis functions around extraordinary vertices, where the topology differs from a regular grid, can be evaluated as described by Stam [14]. For more details on computing isogeometric thin shell analysis using subdivision surfaces, we refer the reader to [7, 15, 8, 16, 9].

### 3.2. Forces

The response of a surface to environmental impact is determined by the physical material of the object. For the thin shell simulation, an isotropic material is defined by three values: the thickness  $t$  of the material, its Young's modulus  $E$  and its Poisson's ratio  $\nu$ . For example, to simulate a sheet of aluminum with a thickness of  $1\text{mm}$ , the material is defined by

$$t = 0.001\text{m} \quad E = 6.9 \cdot 10^{10}\text{N/m}^2 \quad \nu = 0.32$$

Using the material definition, an element stiffness matrix can be constructed for each face in the subdivision control mesh [7]. These element matrices are then assembled into the complete stiffness matrix  $\mathbf{K}$ , which is of size  $3n \times 3n$  for  $n$  control points in the subdivision control mesh.

Forces causing the surface to deform can be applied to all DOF, i.e. to the individual  $x, y, z$  components of the control points of the subdivision surface. The simplest form is to apply a force to a single control point. In contrast to classical

FEM, which typically employs linear basis functions, for IGA this does not define a true point force concentrated to a single point of the Catmull-Clark subdivision surface. Instead, the force spreads over all basis functions which are non-zero at that control point. If a force is applied to a region, or the whole object, the force is distributed over all control points which have their support within the region of interest. The vector of forces  $\mathbf{f}$  for all DOF is used as the right hand side of the equation for the thin shell simulation:

$$\mathbf{K}\mathbf{u} = \mathbf{f} \quad (1)$$

Solving Equation 1 yields the unknown displacements  $\mathbf{u}$ . Adding the displacements  $\mathbf{u}$  to the initial DOF, i.e. the control point positions, results in new control point positions, namely those of the deformed surface. The smoothly deformed limit surface can then be computed from the new control mesh by employing the subdivision algorithm.

### 3.3. Constraints

To be able to solve Equation 1, we need to specify the boundary conditions for the problem, i.e. we need to apply some constraints to the system of equations. The physical interpretation of this requirement is that without any constraints the object is free floating in space, and any applied force will move it infinitely. Therefore, in three dimensions, at least six DOF need to be constrained to prevent any rigid body translation and rotation. The simplest constraints just prescribe some unknown displacement  $u_i$  to zero, enforcing that DOF  $i$  does not change. Similarly, an unknown  $u_i$  can also be prescribed to a nonzero value, imposing a particular displacement at DOF  $i$ . These constraints can be directly enforced by modifying corresponding rows and columns in Equation 1.

More complex constraints can be defined as linear combinations of unknowns, e.g.:

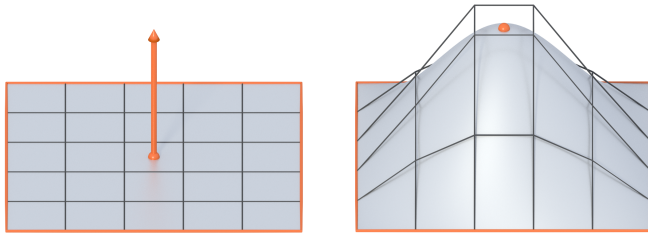
$$au_i + bu_j + cu_k = d \quad (2)$$

Constraints of the form shown in Equation 2 can be added to the system of equations using the *Lagrange multiplier* method, resulting in an augmented system of equations

$$\begin{bmatrix} \mathbf{K} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} \quad (3)$$

where  $\mathbf{A}$  is a matrix with the coefficients of the left hand sides of all Lagrange multiplier constraints, i.e.  $a, b, c$  from Equation 2, and  $\mathbf{b}$  contains the right hand sides of these constraints, i.e.  $d$  from Equation 2. The unknowns  $\boldsymbol{\lambda}$  are the *Lagrange multipliers* and can be interpreted as forces required to satisfy the constraints.

Such constraints are very useful for IGA with subdivision surfaces. Because Catmull-Clark is an approximating subdivision scheme, the displacement of the limit surface cannot be directly controlled by the displacement of a single control point. To directly constrain the displacement of the subdivision limit surface we define the linear combination of control points which compute the limit point position. The linear combinations are derived from the Catmull-Clark subdivision basis



**Fig. 1. A plate with a fixed boundary:** The subdivision limit surface is displaced with a single Lagrange multiplier constraint. The coefficients of the constraint are defined by the subdivision basis functions for the point on the surface.

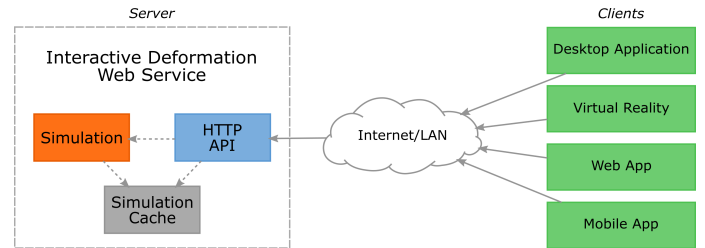
functions, as discussed in Section 3.1. These linear combinations can in turn be used to define Lagrange multiplier constraints to enforce certain displacements of the limit surface. Fig. 1 shows how the limit surface can be exactly displaced using a single Lagrange multiplier constraint. This can be applied to any point on the surface, independent of the actual control points. The control points are automatically moved by the thin shell simulation so that the constraint is satisfied.

### 3.4. Interactive Simulation

Having the basic building blocks for a thin shell simulation available, a deformation of a subdivision surface can be computed by constructing the stiffness matrix  $K$  and applying the desired forces and constraints to the system of equations. Unfortunately, this is generally too slow for interactive use of the simulation, with building the stiffness matrix being the most time consuming part. However, as already observed in [10], if the goal is to apply different forces and constraints to the same initial subdivision surface, the stiffness matrix stays the same and has to be computed only once for the initial surface. Therefore, to use the isogeometric thin shell simulation interactively, the stiffness matrix is precomputed for the initial geometry and subsequently used with different sets of forces and constraints to compute deformed surfaces. This computes a linear approximation to the solution of the partial differential equations describing the physical deformation. Therefore, given enough DOF to represent the solution, this results in highly accurate deformations as long as the overall deformations are small. Unfortunately, what exactly is considered a small deformation cannot be easily checked. The linear simulation assumes that the resulting stress and strain are proportional [17]. This implies that the resulting stress must be below the yield-strength of the simulated material. However, to check this, one needs to perform a non-linear simulation. For the use case of interactive deformation in virtual worlds, however, the linear simulation also provides plausible deformations in many cases where overall deformations are large. One notable exception are large rotational deformations which can cause artifacts where the surface area increases unrealistically, as shown in [2].

## 4. Server

To provide the thin shell simulation from Section 3 to different applications running on different types of devices, we de-



**Fig. 2. Isogeometric analysis web service architecture:** Using the HTTP API, many different clients have access to the thin shell simulation to compute deformations.

fine an HTTP API to access the simulation running on a central server. This web-based simulation service can be accessed from any web enabled client, ranging from apps running on mobile devices to VR applications running on high end workstations. This separation of the simulation and the client allows for flexible use cases. For example, the simulation may run on a powerful server to be accessed by multiple mobile devices, or, the simulation can run on the same computer as the client to minimize latency for high performance VR applications.

### 4.1. Server Architecture

Fig. 2 shows the architectural overview of our isogeometric thin shell simulation web service. The server itself consists of three main parts:

- the thin shell simulation,
- an in-memory cache to store results, and
- the web server providing the HTTP API.

Each of these parts can be easily extended or replaced. For example, in addition or as a replacement for the thin shell simulation another back-end to compute the deformation could be added. Similarly, other APIs can be added to access the simulation, for example a WebSocket-based API to allow for bi-directional communication. However, this paper focuses only on the HTTP API to access the isogeometric thin shell simulation.

The thin shell simulation is responsible for precomputing the stiffness matrix, as explained in Section 3.4, and to compute the final deformation for a given set of forces and constraints. The precomputed matrix and computed deformation results are stored in the in-memory cache for fast access. The API provides easy access to this for clients, as detailed in the following section.

### 4.2. HTTP API

To access the simulation service, an API based on the HTTP protocol is provided. The main advantage of the HTTP protocol is that it is available almost everywhere. This enables using the web service from any client, ranging from web applications running in a browser, or mobile applications running on tablets or phones to VR applications displayed in a CAVE or head mounted display (HMD). Another advantage is that distributed clients can seamlessly connect to the web service over

the Internet, without requiring special rules, e.g. for firewalls, because the HTTP protocol is allowed in most cases, which might not be true for a custom protocol based on UDP or TCP.

The API is kept intentionally simple for ease of use and is split into two parts: the core API to compute deformations, and optional additional functionality that is useful in different applications.

#### 4.2.1. Core API

A small core API is enough to provide the minimum functionality to compute deformations for single user applications, where synchronization between multiple clients is not required. It consists of the following HTTP endpoints:

POST /geometries?namespace= to upload a new geometry resource to the web service. The request data includes the geometry and material parameters. This returns a unique geom\_id for the geometry, based on a hash of its optional namespace and the geometry and material data. This allows multiple clients to share the same geometry. If the clients work in the same namespace, the geom\_id of a particular geometry is equal for all clients. Therefore, all clients share the same geometry and its deformation results. At the same time, using different values for the optional namespace enables clients to upload the same geometry, but treat them as different resources.

Once the geometry is uploaded, the server starts precomputing the stiffness matrix, as described in Section 3.4. After this precomputation is performed, deformations can be computed quickly.

POST /geometries/{geom\_id}/results to compute a deformation for the geometry with the given geom\_id. All data needed to compute the deformation, like constraints and forces, is sent in the body of the POST request. This endpoint returns the displacements of the DOF together with a monotonically increasing res\_id value, starting at 1.

If the precomputed stiffness matrix for this geometry is not yet available in the cache, this endpoint blocks until the pre-computation finishes and then computes the deformation.

#### 4.2.2. Additional Functionality

The core API to compute deformations can be extended with additional functionality on the server, which is required for example to synchronize deformations across multiple clients and to implement more complex use cases. In the following, several optional endpoints are presented that provide additional useful features for the simulation service. Many additional, application specific API endpoints are possible and can be easily added.

GET /geometries and

GET /geometries/{geom\_id} to list and download subdivision control meshes and material data for geometries uploaded by a different client.

GET /geometries/{geom\_id}/results and

GET /geometries/{geom\_id}/results/{res\_id} to list all computed deformation results for the given geometry and to download individual results. These endpoints can be used, among other things, to implement undo functionality into clients.

GET /geometries/{geom\_id}/latest\_result?have= can be used to get the latest deformation result for the given geometry. The optional parameter have can be used by the client to indicate which res\_id it already has. The res\_id 0 is special in that it indicates that the client only has the initial, undeformed geometry. If there is a newer result, with a higher ID, available, the server returns the deformed geometry and res\_id of the latest result. If the client already has the latest result, the server can keep the request open and send a response once a new result is available. This minimizes the number of round trips between client and server. Alternatively, the server could also return an empty response to the client if keeping the request connection open is not desired or not supported by the server.

GET /geometries/{geom\_id}/

limit\_comb?u=&v=&face-index= is used to get the linear combination of control points defining the limit point for a certain parameter location (u, v) on the face with index face-index in the subdivision control mesh. This returns the linear combinations for the limit position as well as for the tangents at the given location. This can be used by the client to get linear combinations needed e.g. for Lagrange multiplier constraints of the limit surface. Using this API, the client does not need to implement complex evaluations of the subdivision surface itself.

## 5. Client

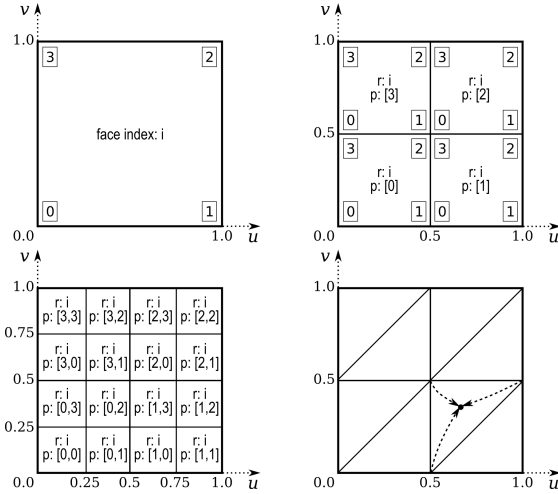
The HTTP API presented in Section 4 can be used by any client that can send HTTP requests to access the simulation. In particular, this allows e.g. web browsers, mobile applications and most game engines to access and use the simulation web service.

For simple use cases the client needs to provide only the geometry data for a Catmull-Clark control mesh and a set of forces and constraints to compute a deformed surface. However, in some cases, e.g. to apply constraints to the subdivision limit surface, more information is required.

### 5.1. Catmull-Clark Parameter Estimation of Ray Intersections

As discussed in Section 3.3, the linear combination of control points that defines a limit position is needed to apply a constraint directly on the limit surface. To compute these linear combinations, either within the client application or with the API from Section 4.2.2, the (local) parameter location of the desired point on the corresponding subdivision patch is required. Fig. 3 summarizes a simple approach to estimate the parameter values of the limit point using a slightly extended implementation of Catmull-Clark subdivision on the client. This method is similar to some pre-tessellation methods to evaluate Catmull-Clark surfaces for ray tracing [18]. Here, the tessellation is used to estimate the parameter location of a ray intersecting the surface, e.g. for user interaction or collision detection, and not to evaluate the limit surface itself.

During each subdivision step, the index i of the original root face (Fig. 3 top-left) in the control mesh, from which each subdivided face originated, is stored as r. Additionally, for each



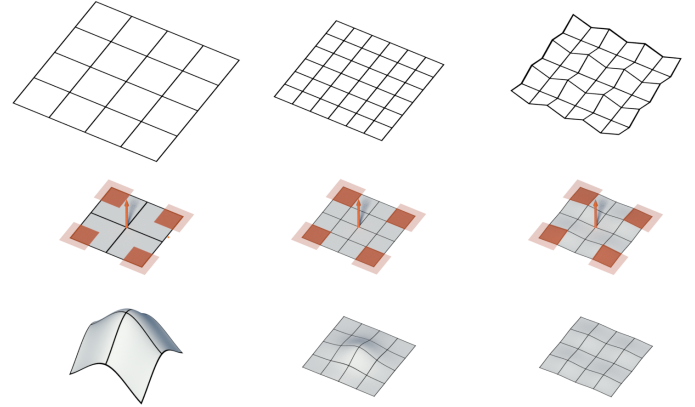
**Fig. 3.** Parameter estimation for a ray intersection with a tessellated Catmull-Clark subdivision surface: New faces created during subdivision (top-right, bottom-left), store an index  $r$ , referencing the corresponding root face  $i$  (top-left), and the path  $p$  from the root to the given face. In each subdivision step the corresponding corner index of the parent face, displayed in boxes, is appended to the path. The path is used to determine the parameter range of a given quad, consisting of two triangles. The final parameter value is a linear interpolation of the triangle corner parameters based on the barycentric coordinates of the intersection (bottom-right).

face in the subdivided mesh the index of the corresponding corner vertex in its parent face, i.e. 0, 1, 2 or 3 for quads, is stored. This index is stored for every subdivision level and defines the path  $p$  from the root face to a given face in the quadtree of subdivided faces (Fig. 3 top-right and bottom-left). So, a face at subdivision level 2 stores two values, e.g. [2, 1] for the right most face just above the vertical center. For rendering, each quad is split into two triangles (Fig. 3 bottom-right), each of which refers to the same information stored for the quad. This information requires one additional index and 2 bits per subdivision level of additional storage for each subdivided face.

To find the control face and parameter location of a ray intersecting the Catmull-Clark surface, the intersection is first computed with the triangle mesh used for rendering. If there is an intersection, the previously stored information for that triangle/quad is retrieved. From path  $p$  the possible range of parameter values can be derived, i.e. the face with path [2, 1] contains the  $(u, v)$  parameter domain from  $(0.75, 0.5)$  to  $(1.0, 0.75)$ . The first path item 2 limits the  $u$  domain to  $[0.5 \dots 1.0]$  and the  $v$  domain also to  $[0.5 \dots 1.0]$ . The second item 1 then restricts these domains further, for  $u$  to  $[0.75 \dots 1.0]$  and for  $v$  to  $[0.5 \dots 0.75]$ .

An approximation of the final parameter value can be computed as a linear interpolation of the corresponding range boundaries based on the barycentric coordinates of the ray intersection with the triangle (Fig. 3 bottom-right). The higher the subdivision level of the mesh used for intersection testing, the more accurate the approximation of the parameter values.

The face index in the control mesh used for evaluation is found by using the stored index  $r$  of the selected face. The linear combination of control points defining the limit position at the approximated parameter location are derived from the



**Fig. 4.** Simulation of different levels of detail of a subdivision surface can lead to very different results. The top row shows three control meshes defining similar surfaces, from left to right: the initial coarse control mesh, a subdivided control mesh derived from the coarse control mesh, and a subdivided control mesh with added details not present in the coarse mesh. The corresponding surfaces are shown in the center row, together with forces and constraints highlighted in orange. The corners are fixed in place while a force is applied at the center of each surface. The results in the bottom row, computed using the same forces and constraints for each surface, are very different. Either due to different DOF (left, center), where more DOF improve the accuracy of the result. Or due to different surface geometry with different stiffness properties (center, right).

Catmull-Clark basis functions. Because the simulation service already needs to perform such evaluations for the isogeometric thin shell analysis, see Section 3.1, it can also provide functionality to compute these linear combinations, as described in Section 4.2.2. However, the face index and parameter location always need to be provided by the client.

## 5.2. Level of Detail Simulation

One advantage of subdivision surfaces is that the level of detail for the visualization can be easily adapted by rendering a different subdivision level of the surface. Therefore, objects close to the virtual camera can be rendered at a high subdivision level, resulting in many triangles to represent a smooth surface, while objects far away can be rendered at a low subdivision level or even without subdivision.

Subdivision can also be applied to increase the DOF of the surface for analysis. Providing surfaces with more DOF to the simulation leads to more accurate and detailed deformation results. However, the number of DOF also directly affects the time needed to compute the deformation, as discussed later in Section 7. Therefore, there is always a tradeoff between accuracy and computation time.

To achieve a fast response time but eventually also get a very accurate result, a parallel simulation scheme can be used. The basic idea is to perform the simulation on multiple subdivision levels of the surface at the same time. The result for the coarse mesh can be computed quickly and can therefore be presented to the user immediately. Meanwhile, a variant of the same surface with more DOF takes longer to compute, but provides the user with a more accurate deformation result, once it is available.

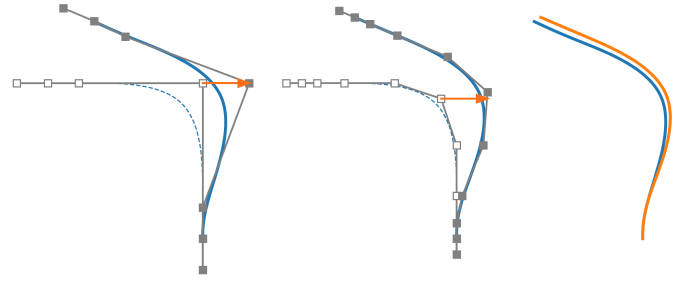


The parallel simulation can be controlled entirely by the client, without modifying or extending the HTTP API from Section 4.2. However, it assumes that the server can handle multiple requests simultaneously, to compute several deformations in parallel. To be able to simulate multiple subdivision levels of a surface, the client uploads each level as a separate geometry to the server. In other words, instead of just uploading the initial control mesh of an object, the client uploads the initial control mesh and additionally also subdivides this control mesh and uploads the resulting subdivided geometry as a separate object to the server. It is important to note that both control meshes define the exact same subdivision limit surface, on which all simulations are based. While simply subdividing the control mesh for the simulation could also be performed by the server, often, additional details are added to the subdivision surface for visualization on the client, e.g. using displacement maps. The client can also add these additional visualization details as geometric details to the subdivided control mesh by displacing the control points. This way, the details in the geometry are part of the simulation and can therefore lead to a more accurate deformation result.

Fig. 4 highlights the differences in results and problems that can arise from different levels of detail and additional geometric details. The coarse mesh on the left results in a very large deformation, because it has too few DOF to accurately represent the solution for the given forces and constraints. The subdivided control mesh in the center, describing exactly the same surface, results in a small, but visible deformation. However, the subdivided mesh with added details on the right, having the same number of DOF, results in almost no displacement at all. This is because the slightly rippled surface is much stiffer than the perfectly flat surface. While typically the differences are not as extreme as in this example, it is important to consider the amount of DOF and the influence of geometric details on the structural stability of surfaces when simulating multiple levels of detail.

Another challenge with this approach is to find corresponding constraints and forces for the subdivided control meshes. Often, the client only has constraints and forces for the initial control mesh, e.g. because only the coarse initial control polygon is shown in the user interface for manipulation, but a more accurate, higher resolution simulation result of the limit surface should eventually be visualized. In that case, the client needs to derive constraints and forces for the higher subdivision level, which has different DOF, from the user interaction with the initial control points. Having the corresponding constraints and forces, the client can then separately request deformation results for each subdivision level from the server.

Simple constraints of single DOF, as mentioned in Section 3.3, are difficult to apply exactly on a subdivided control mesh. The reason is, that the influence of a single DOF of the coarse control mesh on the surface is spread to multiple DOF in the subdivided mesh. If we spread the constraint itself in a similar way, multiple neighboring DOF are constraint, instead of just one. Unfortunately, this not only results in the desired displacement, defined by the constraint, but also prevents any rotation of the surface, as all neighboring DOF are constraint,



**Fig. 5.** The same constraint, a displacement of one x-coordinate - visualized with an orange arrow, is applied to different subdivision levels of the same curve (left, center), of which the bottom end is fixed. Even though the DOF of the curves are different, the resulting deformation is very similar. In the comparison on the right, the blue result is the constraint applied to the original control polygon and the orange result is for the subdivided curve. By minimizing the energy caused by the deformation, the simulation automatically moves nearby control points in a similar way as the constraint DOF. Therefore, the limit surface moves approximately as specified by the single displacement constraint, independent of the number of DOF.

which is not desired.

Interestingly, we found that in many cases a good approximation of the constraint on the coarse control mesh is to just apply the same constraint to the corresponding control point of the subdivided control mesh, as demonstrated in Fig. 5. As the thin shell simulation tries to minimize the deformation energy required to satisfy the constraints, control points nearby constrained DOF are automatically moved similar to the defined constraint to avoid deforming the surface, if possible. And due to the Catmull-Clark subdivision rules, moving a control point and its one-ring neighborhood directly relates to the movement of the corresponding region on the subdivision limit surface. Therefore, neglecting the influence of other constraints and forces, a single displacement constraint causes approximately the same displacement of the limit surface, independent of the subdivision level it is applied to.

Forces applied at DOF are also valid on the subdivided control mesh, but the area of influence is reduced because the support of each DOF on the limit surface shrinks with each subdivision level. To spread the force to the same region as on the initial coarse control mesh, the force vectors can be subdivided according to the Catmull-Clark subdivision rules in the same way as the control points are subdivided. This way, a force applied to a single DOF in the coarse control mesh is split into multiple forces, applied to the corresponding DOF and its one-ring neighborhood in the subdivided control mesh. The subdivision rules ensure that the total force applied to the surface stays constant.

Finally, constraints defined directly for the limit surface, as explained in Section 3.3 and visualized in Fig. 1, are also valid for subdivided surfaces, because they are defined for a certain position on the subdivision limit surface, independent of the control points. However, the linear combination of DOF that define the limit constraint need to be computed separately for each control mesh, as shown in Section 5.1. Limit constraints always ensure that surfaces displace exactly the desired distance, independent of the control mesh. Therefore, they are the



Fig. 6. Synchronizing soft-body deformations across multiple devices: Both the workstation and the tablet run the same application connected to the simulation web service. Deformations are immediately synchronized between both devices.

preferred way to apply constraints from direct user interaction.

## 6. Applications

This section presents several applications using the proposed simulation web service in various environments for different use cases.

### 6.1. Multi Client

A challenge of multi-client VR environments is the correct and efficient synchronization of the different clients. This especially includes the synchronization of soft-body deformations. If a dense polygon mesh is used for deformation, transferring the deformed surface can require a lot of bandwidth. On the other hand, if each client computes the deformations independently, inconsistencies can appear due to different timing or computing capabilities of the devices.

Using the central simulation web service based on Catmull-Clark subdivision surfaces to synchronize deformed surfaces across multiple devices overcomes these issues. First, only the subdivision control mesh needs to be synchronized between clients, using less bandwidth. And second, all clients get the exact same deformation because it is only computed once by the server for any given geometry. Each client then performs Catmull-Clark subdivision on the control mesh to derive the smooth limit surface for visualization. This allows each client to render different levels of detail of surfaces without inconsistencies, because the limit surface is fully defined by the control mesh and the subdivision scheme.

Fig. 6 shows two devices, a workstation and a tablet, running the same application connected to the central simulation server. Surface deformations caused by user interaction are immediately synchronized between both devices, using the `latest_result` API endpoint described in Section 4.2.2.

Currently, the applied forces and constraints are not synchronized between clients, only the deformation results are shared. Therefore, deformations requested from one device override

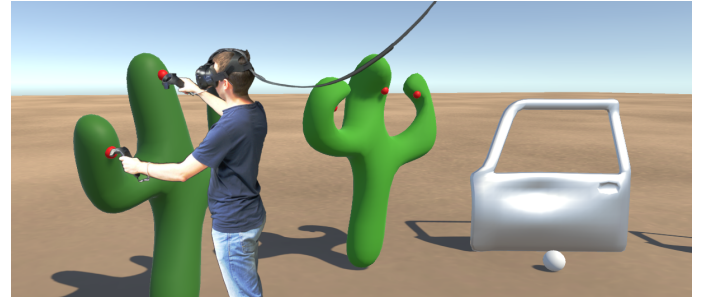


Fig. 7. Subdivision surfaces can be interactively deformed in an intuitive way using hand controllers and a head mounted display. Users interact directly with the subdivision limit surface and can define constraints, visualized with small red spheres, by pushing and pulling the surface. Additionally, users can also pick up and move rigid body objects, like the white ball in the scene, which can also cause deformations by colliding with surfaces.

previous results requested on other devices, as usually only the last deformation result is shown. The server and the API could be extended to also store and synchronize the constraints and forces provided by the requesting client together with the deformation result. This would allow multiple clients to deform an object alternately. A difficulty with this approach are race conditions due to simultaneous deformation requests by multiple clients. However, synchronizing constraints and forces is currently not implemented and we therefore have a *master* client for each object who defines the applied constraints and forces.

One use case for this type of synchronization is rendering multiple projections of the same scene on multiple devices, as it is usually done in a CAVE [19] environment or on a large tiled display [20]. Using the approach described in this paper, it is ensured that all projections render the exact same soft-body deformation, controlled by a single master device.

### 6.2. High Performance VR

Current state-of-the-art computer graphics techniques can be used to create immersive virtual worlds that *look* almost real, however these worlds do not *feel* real because the interaction with objects is very limited. Many real world objects are not rigid and users expect to be able to deform them. To improve realism, interactive soft-body deformation of virtual objects can be provided using the simulation web service described in this paper. However, visualizing such virtual worlds using immersive head mounted displays like the HTC Vive or Oculus Rift requires high frame rates and low latency processing of user inputs.

To ensure high performance and low latency of the VR application, the deformation service can run on the same high end workstation as the visualization. It does not slow down the rendering loop of the visualization because it runs in a separate process and network latency is minimized by running the server and the client on the same computer.

In a prototype application for the HTC Vive HMD, shown in Fig. 7, interactive deformation has been implemented using a simple push/pull interface using the tracked hand controllers. Users can simply grab and move a surface directly with their



hands, similar to the real world. Grabbing the surface adds a constraint on the subdivision limit surface which is updated when the user moves the controller. To compute the selected point, a ray based on the position and orientation of the users hand is intersected with the triangulated mesh used to render the subdivision surface. The face index in the subdivision control mesh and the parameter location of the intersection is then computed as discussed in Section 5.1. The deformed surface is asynchronously computed by the simulation web service and, once the result is available, updated in the visualization to interactively reflect the user input. The grabbed surface point therefore exactly follows the users hand movement, while the rest of the surface deforms according to its material properties. Applying the constraint on the limit surface uses the `limit_comb` API endpoint described in Section 4.2.2 and allows the user to not only change the position of the surface but also to control the tangent plane at the grabbed surface point.

However, as explained in Section 3.3 multiple DOF of the surface need to be constrained to prevent any rigid body motion. As a simple intuitive metaphor to fix virtual objects in place, all parts of an object touching the floor are automatically constrained to the floor at that position.

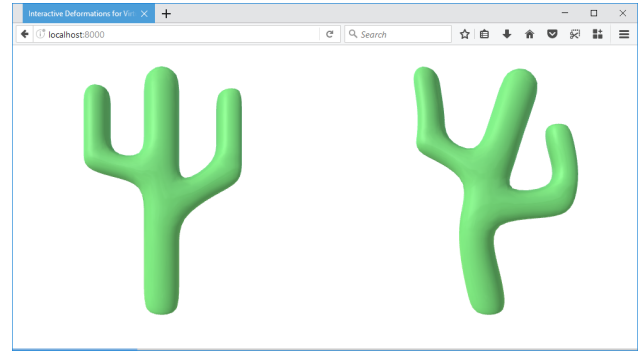
For additional feedback, vibrations of the controller are used to let the user *feel* the deformation. The more deformation the interactions cause, the stronger the vibrations of the controller.

In addition to direct user input also interaction of the soft-body deformation with traditional rigid body physics has been implemented. This lets users deform virtual surfaces by e.g. throwing rigid objects on them. Once a collision with a rigid body object is detected, the parameter location of the collision point can be computed similar to how it is done for the user interaction. But instead of constraints, forces derived from the colliding objects are applied to the surface. Each surface will respond to collisions according to its material properties, improving realism.

### 6.3. Interactive Collaborative Web Based Tools

A big advantage of the HTTP based API described in Section 4.2 is that it can be accessed from any client that can use the HTTP protocol. This also includes web browsers, which use HTTP natively to transfer data from web servers. Together with other features of modern web browsers like fast JavaScript implementations, WebGL [21] and more recently WebVR [22], they can be used to create interactive visualizations of virtual worlds or virtual objects. Users can easily access such visualizations on the web without installing any special software besides a web browser. Using the presented deformation web service now also enables users to interactively, and possibly collaboratively, deform objects in a web browser, as shown in Fig. 8. Providing simulation tools for web based applications has many use cases, for example in games, product customization or engineering.

One possible use case is for example collaborative design and evaluation. To collaboratively evaluate designed objects, plugins for modelling or CAD software can send the subdivision geometry directly from the modelling application to the simulation server. A web based application allows to share access to



**Fig. 8.** The simulation server can also be accessed from web applications running in a browser. On the left, a cactus model is shown in its initial state. The deformed surface, after applying some constraints, is shown on the right. Scripts running in the browser control the simulation parameters based on user interaction and efficiently render the surfaces in real time using WebGL.

these models with other users and to interactively perform simulations. This enables users to discuss design ideas and to simulate and analyse structural properties. This use case is equally applicable to designing virtual worlds as it is to rapid prototyping of objects that are manufactured for the real world.

Different interfaces can also be combined, as the only requirement for collaboration is that all users access the same simulation server. Therefore, one user can, for example, control simulation parameters with a web based interface on a mobile device, while the simulation results are visualized in a VR environment. Meanwhile, a designer can create and edit geometry in a CAD application and seamlessly upload new geometry to the simulation server for other users to simulate.

This type of application makes use of all of the additional APIs defined in Section 4.2.2. Depending on the application, also more APIs can be defined on the server, e.g. to compute stresses in the material of a simulated object intended for manufacturing or to compute an accurate non-linear simulation non-interactively for engineering analysis.

## 7. Performance





This section discusses several performance related topics of the presented simulation web service. All timing measurements were performed on a workstation with an Intel Core i7-3820 CPU running at 3.6 GHz and 12 GB of main memory.

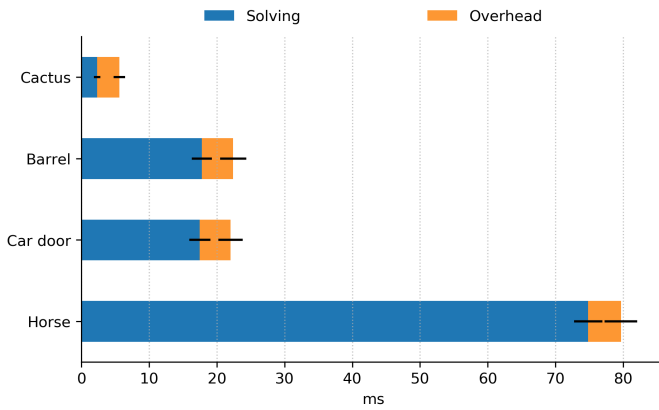
### 7.1. Precomputation

For computing a thin shell deformation, the most time consuming part is the derivation of the stiffness matrix. As noted in Section 3.4, this has to be done only once per object to compute linear simulation results and can be cached for subsequent uses of the model.

Table 1 lists the time needed to perform this initial precomputation for various models. Generally, the computation time depends on the number of faces in the subdivision control mesh, i.e. the number of elements in terms of FEM. However, it also depends on the number of extraordinary vertices (EVs), vertices with a valence other than four, in the control mesh. Regions

**Table 1. Time needed to precompute the stiffness matrix and to solve the system of equations for various models**

				
Model	Cactus	Barrel	Car door	Horse
Control Vertices	60	210	218	472
thereof EVs	32	8	24	64
Faces	58	208	218	470
Precomputation	14 s	7 s	20 s	82 s
Solving	2 ms	18 ms	18 ms	75 ms

**Fig. 9. Average latency and standard deviation (visualized as black bars) for deforming various objects: The biggest part, solving the system of equations, is performed using a CPU based sparse matrix solver for smaller meshes and using a GPU based solver for the *Horse* model. The remaining overhead includes request processing and network latency.**

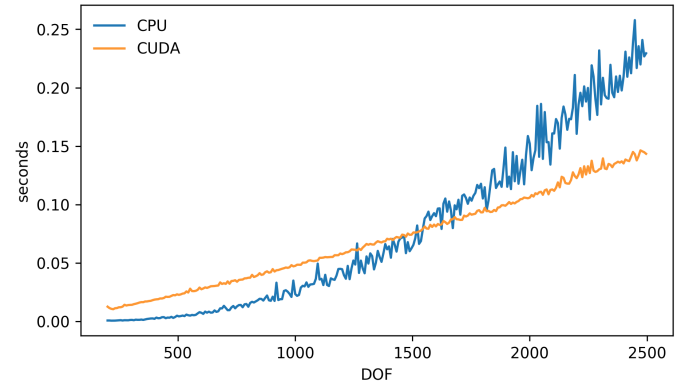
of the Catmull-Clark subdivision surface containing EVs cannot be evaluated as cubic B-splines but require a more complex evaluation [14]. Usually, the precomputation takes between a few seconds and a few minutes, depending on the complexity of the subdivision control mesh.

Once the stiffness matrix has been either precomputed or loaded from a cache, a deformed version of the surface can be computed quickly, depending on the size of the system of equations, by applying forces and constraints and solving the resulting system of equations from Equation 1 or Equation 3.

## 7.2. Latency

It is very important for interactive applications to minimize the time between user interaction and visualization of the corresponding changes. This applies especially to VR applications. High latency limits immersion and can cause nausea for users of the VR application [23].

Fig. 9 shows the total latency, from request until the result is available at the client, for deforming different objects. The stiffness matrix for each object was already precomputed before. The values show the average latency observed during an interactive modelling session creating approximately 300 deformations for each object using varying combinations of vertex constraints, Lagrange constraints and forces. The black error

**Fig. 10. A GPU based solver based on the *cuSOLVER* CUDA library improves performance for larger systems of equations, but has a higher overhead compared to a CPU based solver. Therefore, the solver is dynamically chosen based on the DOF of the simulated mesh.**

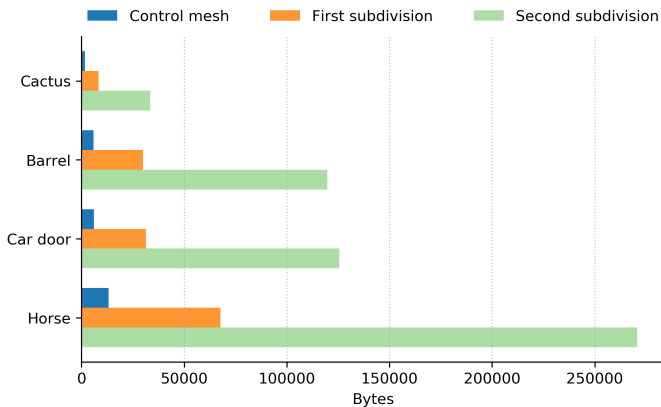
bars visualize the standard deviation of the measured timings. Most of the time is spent solving the linear system of equations on the server. This also includes building the deformed result by adding the computed deformations to the initial control points. The effect of additional Lagrange constraints, which increase the size of the system of equations, is small compared to the overall solving time. The remaining overhead includes processing the HTTP requests on the client and server and transferring the data over the network.

The systems of equations are either solved with a standard CPU based sparse matrix solver or using a GPU based solver based on the *cuSOLVER* CUDA library, depending on the number of DOF. Using the GPU based solver has higher setup costs and therefore slower performance for solving the equations resulting from smaller meshes. A comparison of the two solvers for random systems of equations with varying DOF is shown in Fig. 10. At around 1400 DOF, the GPU based solver tends to be as fast as the CPU based solver in most cases. For even larger systems of equations the GPU based solver is significantly faster. Therefore, for larger meshes with more than approximately 1400 DOF, like the *Horse* model, we use the GPU based solver, while for smaller meshes the CPU based solver is used.

The perceived latency for the user depends on the frame rate of the visualization. For desktop applications, typically rendering 60 frames per second (FPS), the deformed result for a moderately sized mesh will be visualized 1-2 frames after user interaction. This delay of up to 33 ms is acceptable and usually not noticed by the user. The *Horse* mesh has a latency of 5 frames at 60 FPS, or 83 ms, which is already noticeable but still allows interactive deformation.

## 7.3. Bandwidth

One of the challenges for synchronizing soft-body deformations over a network is the required bandwidth. In addition to object position, velocity, etc., which also need to be synchronized in rigid-body physics common in multiplayer games, soft-body deformations also need to synchronize each vertex of the



**Fig. 11. Size of initial mesh data in bytes: The subdivision control mesh requires significantly less bandwidth than the triangulated dense meshes used for rendering. By taking advantage of isogeometric analysis, the control mesh can be directly used to compute deformations.**

mesh.

For rendering, the subdivision control mesh is typically subdivided at least once or twice before triangulation to get a smooth looking surface. By taking advantage of IGA based on subdivision surfaces, only the subdivision control mesh is sent to the server when uploading new geometry. Other mesh based deformation approaches mentioned in Section 2 require synchronizing the complete mesh that is visualized to get accurate and consistent deformation results across different clients, because all DOF are required to compute a deformation result.

As a baseline, Fig. 11 compares the required bandwidth in bytes for transferring the initial control mesh and the triangulations of the first and second subdivision level of that mesh respectively. This assumes that vertices are defined using three single precision floating point values and  $n$ -sided faces are defined by  $n$  32-bit unsigned integer indices. Using the subdivision control mesh, instead of the denser meshes used for rendering, saves a significant amount of bandwidth.

When sending the resulting deformed mesh data back to the client, only the vertices of the subdivision control mesh have to be transmitted. This also results in similar savings compared to a dense triangulated mesh. Therefore, the network overhead of transferring the result, included in the timings in Fig. 9, is very low compared to the time needed to solve the system of equations.

For all mesh based deformation techniques, including our subdivision based approach, many existing methods for 3D mesh compression could be used to further reduce the required bandwidth for the mesh upload [24].

## 8. Conclusion and Future Work

We present a flexible client-server architecture of a simulation web service to interactively compute and synchronize deformations of moderately complex subdivision surfaces. The defined HTTP API provides this simulation to all web enabled

devices, enabling physically accurate deformation also in resource constrained environments like mobile devices or web applications.

Users can directly interact with the subdivision limit surface, while only the control mesh needs to be synchronized. By using subdivision surfaces, we can also simulate different levels of detail of a surface. To do so, we show how to apply constraints and forces defined on a coarse control mesh to a finer subdivision level of the same surface.

The web service architecture has many different use cases, e.g. realistic deformations for interactive virtual worlds, interactive physics-based product customization or collaborative analysis of designs. Using the presented architecture, different types of clients, like web applications, mobile devices or CAD applications, can seamlessly interact with the same geometry.

One remaining limitation of the presented approach is that only a linear thin shell simulation can be performed interactively. The linear simulation results in physically accurate deformations as long as the overall deformations are small. For larger deformations this approach also yields plausible results in many cases, but may lead to distortion artifacts as demonstrated in [2]. An accurate simulation of larger deformations necessitates a non-linear simulation, which requires repeated updating of the stiffness matrix, which is too time consuming for interactive use cases. Further, the topology of the subdivision surface used for deformation cannot change as this also requires recomputing the stiffness matrix.

Another limitation are complex subdivision meshes with many control points, where computations are too slow for interactive deformation. Even neglecting the long precomputation time to setup the stiffness matrix, solving the system of equation alone is today still too time consuming on an average computer. The *Horse* example can still be deformed interactively, as shown in Table 1 and Fig. 9, but solving the system of equations for significantly larger meshes is too slow. We will therefore explore whether a more efficient approach to solving the equations on the GPU, e.g. as presented in [25], might enable deformation of larger models at interactive rates.

As the presented architecture is very flexible, the web service could easily be extended to include other types of simulations by adapting the simulation computation at the server. A possible future extension are also volumetric representations, e.g. based on an extension of Catmull-Clark to solids [26].

## References

- [1] Riffnaller-Schiefer, A, Augsdörfer, UH, Fellner, DW. Interactive physics-based deformation for virtual worlds. In: 2017 International Conference on Cyberworlds (CW). 2017, p. 88–95.
- [2] Nealen, A, Müller, M, Keiser, R, Boxerman, E, Carlson, M. Physically based deformable models in computer graphics. Computer Graphics Forum 2006;25(4):809–836.
- [3] Botsch, M, Sorkine, O. On linear variational surface deformation methods. IEEE Transactions on Visualization and Computer Graphics 2008;14(1):213–230.
- [4] da Silva, JP, Giraldo, GA, Apolinário Jr., AL. A new optimization approach for mass-spring models parameterization. Graphical Models 2015;81(Supplement C):1 – 17.
- [5] Hughes, T, Cottrell, J, Bazilevs, Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comp Methods Appl Mech Engrg 2005;194:4135–4195.

- [6] Bazilevs, Y, Calo, V, Cottrell, J, Evans, J, Hughes, T, Lipton, S, et al. Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics and Engineering* 2010;199:229–263.
- [7] Cirak, F, Ortiz, M, Schröder, P. Subdivision surfaces: A new paradigm for thin-shell finite element analysis. *Int J Numer Meth Eng* 2000;47(12):2039–2072.
- [8] Wawrzinek, A, Hildebrandt, K, Polthier, K. Koiter's thin shells on Catmull-Clark limit surfaces. In: *Proceedings of the Vision, Modeling, and Visualization Workshop 2011*. 2011, p. 113–120.
- [9] Riffnaller-Schiefer, A, Augsdörfer, UH, Fellner, DW. Isogeometric shell analysis with NURBS compatible subdivision surfaces. *Applied Mathematics and Computation* 2016;272, Part 1:139 – 147. Subdivision, Geometric and Algebraic Methods, Isogeometric Analysis and Refinability.
- [10] Riffnaller-Schiefer, A, Augsdörfer, UH, Fellner, DW. Isogeometric analysis for modelling and design. In: Bickel, B, Ritschel, T, editors. *EG 2015 - Short Papers*. The Eurographics Association; 2015, p. 17–20.
- [11] Lin, S, Narayan, RJ, Lee, YS. Hybrid client-server architecture and control techniques for collaborative product development using haptic interfaces. *Computers in Industry* 2010;61(1):83 – 96.
- [12] Tang, Z, Yang, Y, Guo, X, Prabhakaran, B. On supporting collaborative haptic interactions with physically-based 3d deformations. In: *2010 IEEE International Symposium on Haptic Audio Visual Environments and Games*. 2010, p. 1–6.
- [13] Catmull, E, Clark, J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design* 1978;10(6):183–188.
- [14] Stam, J. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In: *Proceedings of SIGGRAPH 1998*. 1998, p. 395–404.
- [15] Green, S. Multilevel, subdivision-based, thin shell finite elements: development and an application to red blood cell modeling. Ph.D. thesis; University of Washington; 2003.
- [16] Barendrecht, PJ. Isogeometric analysis with subdivision surfaces. Master's thesis; Eindhoven University of Technology: Eindhoven, The Netherlands; 2013.
- [17] Slaughter, W. *The Linearized Theory of Elasticity*. Birkhäuser Boston; 2012. ISBN 9781461200932.
- [18] Benthin, C, Boulos, S, Lacewell, D, Wald, I. Packet-based ray tracing of Catmull-Clark subdivision surfaces. Tech. Rep.; SCI Institute, University of Utah; 2007.
- [19] Cruz-Neira, C, Sandin, DJ, DeFanti, TA. Surround-screen projection-based virtual reality: The design and implementation of the cave. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93; ACM; 1993, p. 135–142.
- [20] Hereld, M, Judson, IR, Stevens, RL. Introduction to building projection-based tiled display systems. *IEEE Computer Graphics and Applications* 2000;20(4):22–28.
- [21] Marrin, C. WebGL Specification. Khronos WebGL Working Group; 2011.
- [22] Vukicevic, V, Jones, B, Gilbert, K, Wiemeersch, C. WebVR. World Wide Web Consortium; 2016.
- [23] Allison, RS, Harris, LR, Jenkin, M, Jasiobedzka, U, Zacher, JE. Tolerance of temporal delay in virtual environments. In: *Virtual Reality*, 2001. *Proceedings*. IEEE. IEEE; 2001, p. 247–254.
- [24] Maglo, A, Lavoué, G, Dupont, F, Hudelot, C. 3D mesh compression: Survey, comparisons, and emerging trends. *ACM Comput Surv* 2015;47(3):44:1–44:41.
- [25] Weber, D, Bender, J, Schnoes, M, Stork, A, Fellner, D. Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum* 2013;32(1):16–26.
- [26] Burckhart, D, Hamann, B, Umlauf, G. Iso-geometric finite element analysis based on Catmull-Clark subdivision solids. *Computer Graphics Forum* 2010;29(5):1575–1584.

## Appendix A. Transferred Data

The exact data sent and received for each API endpoint listed in Section 4.2 can be considered an implementation detail. Depending on the use case, different data formats are possible. For example, compressed binary data could be used to save bandwidth, or JSON encoded structures for ease of use in web based

applications. In the following, the binary format used for comparisons in Fig. 9 and Fig. 11 is shown in detail for the relevant core API endpoints from Section 4.2.1. The datatypes are abbreviated as u32 for an unsigned 32 bit integer and f32 for a 32 bit single precision floating point number.

Using `POST /geometries` to upload new geometry, the subdivision control mesh is transmitted as raw binary data with a small header. The header contains the number of vertices  $n$ , the number of faces  $m$ , and the material parameters Young's Modulus  $E$ , Poisson Ratio  $\nu$  and thickness  $t$ . Following the header, the control points and quad face indices are transmitted:

Header	u32 $n$	u32 $m$	f32 $E$	f32 $\nu$	f32 $t$
Vertices	f32 $x_1$	f32 $y_1$	f32 $z_1$		
	...	...	...		
	f32 $x_n$	f32 $y_n$	f32 $z_n$		
Faces	u32 $a_1$	u32 $b_1$	u32 $c_1$	u32 $d_1$	
	...	...	...	...	
	u32 $a_m$	u32 $b_m$	u32 $c_m$	u32 $d_m$	

The response from the server is the unique `geom_id` transmitted as a sequence of 16 bytes.

Computing a new deformation with `POST /geometries/{geom_id}/results` requires the applied forces and constraints. Following the definitions in Section 3, clients can send a combination of  $v$  vertex displacement constraints,  $l$  Lagrange constraints and  $f$  force vectors:

u32 $v$	<i>vertex constraints data</i>
u32 $l$	<i>Lagrange constraints data</i>
u32 $f$	<i>force data</i>

The *vertex constraints data* consists of a sequence of  $dx_i, dy_i, dz_i$  displacement constraint values and a sequence of  $mask_i$  and  $index_i$  values:

Constraints	f32 $dx_1$	f32 $dy_1$	f32 $dz_1$
	...	...	...
	f32 $dx_v$	f32 $dy_v$	f32 $dz_v$
Masks & indices	u32 $mask_1$	u32 $index_1$	
	...	...	
	u32 $mask_v$	u32 $index_v$	

Constraint  $i$  is applied to control point  $index_i$  of the control mesh, according to  $mask_i$ . The mask can be used to only enforce the constraint for certain coordinates, e.g. only for the  $x$  and  $z$  coordinates of the control point.

The *Lagrange constraints data* contains  $p$  control point indices  $c_i$ ,  $3p$  coefficients  $cx_i, cy_i, cz_i$  for all DOF of these control points, and 3 constraint values  $cvx, cvy, cvz$  for each of the  $l$  Lagrange constraints:

#Coefficients	u32 $p_1$		
Indices	u32 $c_{1,1}$	...	u32 $c_{1,p_1}$
	f32 $cx_{1,1}$	f32 $cy_{1,1}$	f32 $cz_{1,1}$
Coefficients	...	...	...
	f32 $cx_{1,p_1}$	f32 $cy_{1,p_1}$	f32 $cz_{1,p_1}$
Values	f32 $cvx_1$	f32 $cvy_1$	f32 $cvz_1$
<hr/>			
#Coefficients	u32 $p_l$		
Indices	u32 $c_{l,1}$	...	u32 $c_{l,p_l}$
	f32 $cx_{l,1}$	f32 $cy_{l,1}$	f32 $cz_{l,1}$
Coefficients	...	...	...
	f32 $cx_{l,p_l}$	f32 $cy_{l,p_l}$	f32 $cz_{l,p_l}$
Values	f32 $cvx_l$	f32 $cvy_l$	f32 $cvz_l$

Each of these Lagrange constraints adds 3 rows and columns to the system of equations, one for each component  $x, y, z$  of the affected control points, as described in Section 3.3.

Finally, *force data* is a sequence of three floats  $fx_i, fy_i, fz_i$  for the forces and a sequence of control point indices  $fp_i$ , to which the forces are applied:

Forces	f32 $fx_1$	f32 $fy_1$	f32 $fz_1$
	...	...	...
	f32 $fx_f$	f32 $fy_f$	f32 $fz_f$
Indices	u32 $fp_1$	...	u32 $fp_f$

After the deformation is computed, the server returns a monotonically increasing `res_id` and the updated control point positions for the deformed subdivision surface:

Result ID	u32 $res\_id$		
	f32 $x_1$	f32 $y_1$	f32 $z_1$
Positions	...	...	...
	f32 $x_n$	f32 $y_n$	f32 $z_n$

The client uses the updated control points together with the unchanged face indices to visualize the deformed subdivision limit surface.