Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2017)
P. Sander and M. Zwicker (Editors)

sLayer: a System for Multi-Layered Material Sculpting

C. Calabrese[1], M. Fratarcangeli[2] and F. Pellacini[1]

[1]Sapienza University of Rome
[2]Chalmers University of Technology

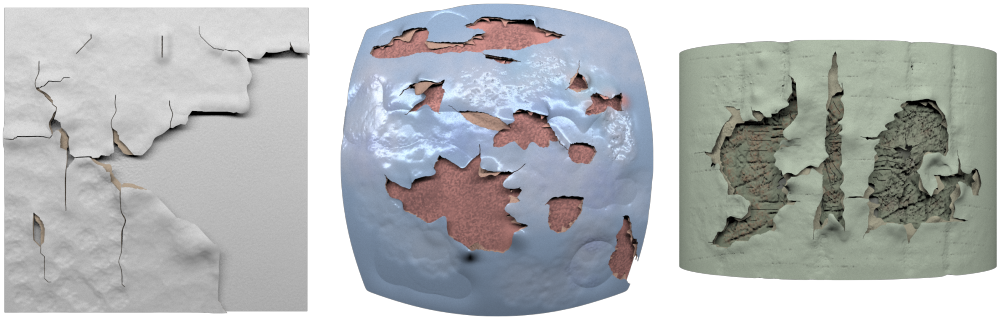

Figure 1: *Sculpts created with* sLayer*, our interactive sculpting tool.* Left. *Peeled plaster (640K triangles).* Middle. *Rusted metal (525K triangles).* Right. *Tree bark (981K triangles).*

Abstract

Many real world materials have a stratified structure, composed by the proximity and the interaction of multiple highly-detailed layers. Example of these materials are peeling paint, old tree bark and rusted metals. While digital sculpting is particularly well-suited to model these aged surfaces, the interaction between layers is not accounted for. We present a system for sculpting multi-layers materials where collision between layers are handled interactively while brushing meshes that scales up to the million of polygons necessary to model aged surfaces. We do so by observing that if the average mean edge length is maintained constant throughout the modeling session, we can use a single data structure, namely a uniform grid, to accelerate all the sculpting operations. We present a brush rasterization pipeline that uses this data structure for multi-layer editing. We also show that by adding a few interface tools for layer creation and selection, we can create detailed surface similar to real-world ones. To the best of our knowledge, our work is the first to show sculpting of highly-detailed, multi-layered materials in real-time.

1. Introduction

Modeling Multi-Layered Surfaces. Digital sculpting is widely used in our industry to model highly detailed natural objects and characters [Ble16, ZBr16, Pai16]. The models have complex materials with fine details in both local deformation and reflectance. Many real-world materials though present a stratified structure, composed by many thin layers, rich of cavities and detached components. To digitally sculpt these objects, an artist would have to manually avoid interpenetration while radically deforming the shape and connectivity of each highly-tessellated layer. Furthermore, reflectivity editing would have to be considered too, since most of these surfaces have different reflectivities in close proximity.

System Desiderata. To be more specific, we are interested in a system targeting surfaces with small details, in the order of a millimeter in length. This roughy translates to meshes in the order of a million polygons for tabletop-sized objects. We consider materials with multiple layers, where each layer can be formed by multiple disconnected patches. To maintain detailed meshes during modeling, we want to support edge refinement operations, as typically done in modern sculpting toolsets. During editing, sculpting tools act on the material as a whole, handling any number of layers while performing large but consistent deformations of the layers stack. To remain consistent, we want our system to automatically avoid interpenetration between of layers. We want to handle interpenetration in real-time, while the artist moves the digital brush, to produce useful feedback during modeling. Finally, as we deform

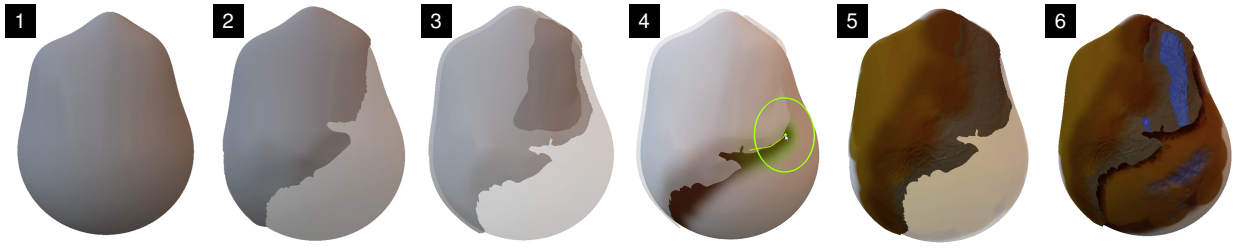

Figure 2: *Sequence of editing operations. 1) a base layer. 2) and 3) patches are drawn as separate layers on the existing surface. 4) the gap in-between the layers is deformed, including collision handling, and painted. 5) and 6) materials are painted directly on the layers with per-vertex BRDFs.*

the surface, we also want to define the BRDF at each location, since surface deformations like scratches and bumps are often correlated with reflectance changes. To the best of our knowledge, existing solutions, including commercial systems, address only single aspects, and typically fail when handling million polygon shapes with interpenetration avoidance during editing.

***sLayer*: Material Editing System.** In this paper, we present *sLayer*, a system for sculpting high-detailed multi-layered materials. In our system, we make one simplifying assumption, that edge length is roughly constant for all polygons. The key insight of our work is that this assumption leads to a relatively simple system design that uses a single data structure to support all major sculpting operations, namely rendering, mouse picking, brush rasterization, collision handling, and remeshing by edge refinement. In particular, we represent material layers as index triangle meshes, with per-vertex normals, tangents and BRDFs, and only additionally maintain vertex–triangle and triangle–triangle adjacencies. To accelerate all operations, we use a regular grid represented as a hash table. This data structure is fast to update, scalable and allows most operations to be trivially parallelized. Throughout the paper we will motivate how we reached this design choices, and show how *sLayer* supports all major editing operations interactively and scalably.

***sLayer* Editing Tools.** While most of our efforts were focused on the underlying sculpting system, creating the results shown in this paper would not have been possible without a few, relatively simple, editing tools that augment the typical scultping toolset to handle multi-layer materials. In particular, we augment sculpting brushes with the ability to modify any number of layers concurrently, and introduce tools for creating new layers easily, by cutting and duplicating other layers. Example of these operations are shown in Fig. 2.

Results. Through this paper, we show detailed materials modeled by us entirely with *sLayer*, in between 20 and 90 minutes. We tested our system with surfaces of a up to a million polygons, shown in Fig. 1, demonstrating the scalability of our approach. The supplemental video contains short screen capture of the modeling sessions. For most results, we started from a real photograph, to test how our system can let us model real-world detailed, organic and non-organic, surfaces.

Contributions. Many single components of our system can be already found in the literature. In our opinion, the main contribution of our work is the overall system design, and its motivations, where we show how to combine these components in a scalable, interactive system for sculpting detailed real-world materials that were not previously possible. To the best of our knowledge, this particular system design has not been previously explored, nor we know of any other tool with the same performance and scalability.

2. Related Work

To motivate the design choices at the core of our system, we first review prior methods that address some of the corcerns treated in *sLayer*.

There are many different methods for modeling highly-tesselated single surfaces that exibit rich material details. For example, [BGF15, ZGZJ16] rely on contructive solid geometry, in [SVJ15] a shrinking and inflating approach is used to track intersections during surface layer crafting, while [PBFJ05, ZHW*06] make use of geometry instancing. Even though the final output model is accurate and rich of subtle details, the computational cost of these techniques is not suitable for the real-time editing of triangulated mesh with up to million polygons, as achieved in our system.

There is a rather wide choice of interactive sculpting tools assembling different parts and features to obtain novel surfaces (e.g., [SS08, SS10, CKGK11]). *sLayer* shares similarities with [SCCS13], a free-form sculpting tool for *closed* surfaces able to handle remeshing on the fly. While it preserves sharp features, that system does not support the intersections between *thin* layers not high detailed surfaces, which is expressly our goal. Furthermore, being based on arbitrary triangulated meshes, *sLayer* can handle any orientable layer, not only closed ones. This is necessary for materials such as peeling paint. In another interactive, multi-layered sculpting tool presented in [DPS15], successive layers are defined according to the strokes of a brush in a sketch-based interface. Every newly defined surface extrudes exactly the shape of the underlying layers limiting the expressiveness of the tool, the possibility to further deform their shapes, and the overall surface details. Additionally, such system does not allow to change the connectivity of the meshes for tearing, peeling and shredding of the layers while sculpting, often found in real-world aged materials, which instead we support.

Most of these tools allow for changes in the topology of the mesh with effective support for shape deformations, but no previous method can handle many thin overlapping layers, supporting their

composition in shape and reflectivity. This allows us to model a category of materials which is not possible with other methods.

We want to stress that *sLayer* does not address a particular geometrical or modeling problem. Rather, it is a system of interconnected components, designed around few cores principles, which together enable the real-time editing of detailed multi-layered materials. Other frameworks, addressing very different problems but with a similar design philosophy in mind, are becoming increasingly popular in different contexts of the Computer Graphics, e.g. [FH11, MMCK14, BSL*16].

3. Multi-layer Material Editing System

Surface Representation. We seek to model detailed materials with multiple overlapping non-continuous layers. Each layer is an indexed triangle mesh that might have any number of disconnected patches. Vertex data includes position, normals and BRDF parameters. To handle strong abrasions, we impose no constraints on the edits of each layers besides non-interpenetration. So patches from different layers might interleave or change order in the physical stack. This is necessary for materials like peeling paints. All operations in our system are applied to individual vertices and triangles, with no special handling between different layers. In fact, layers are only used to allow artists to selectively apply operations in a manner similar to image editing applications. This allows to model any multi-layer materials without restriction imposed by the layers composition.

An alternative could have been to consider a volumetric representation, such as [WR16]. The main concern with this representation is that precise collision detection during deformation would likely be troublesome at interactive rates. Furthermore, to reach the desired detail level, a hierarchical volumetric data structure would likely be needed, with the burden of updating it efficiently during stroke application.

Brush Application. The main operation performed in a sculpting system is the application of brush strokes. Brush application is summarized in Alg. 1 for a standard sculpting system (e.g., [AWC06, SCCS13, DPS15]). Each brush stroke is represented as an appropriately sampled polyline of 3D positions. For each of these positions, we apply a deformation by selecting the vertices within the brush influence, and apply a weighted linear transform to each of them. In a standard brushing application, we now move the vertex positions to the newly computed ones. After each brushed stroke position, we remesh the parts of the surface that have moved to keep a good surface sampling. Remeshing might also be performed after processing each polyline vertex for higher accuracy.

Our system is built on the same brush application pipeline with the major difference that vertex movement are constrained by collisions between layers and self-collision within the same layer. To handle collisions, we determine set of vertices which collide with the surface *after* computing their predicted location. Since the polylines are sampled very finely, we consider the vertex trajectories to be linear for each polyline stroke. For all non-colliding vertices, we update the positions as in standard sculpting. For colliding vertices, we compute a safe position that is intersection free and update the vertex position with this one. This is also illustrated in Alg. 1.

forall the *stroke positions* **p do**
  // select influenced vertices $\{\mathbf{v}_i\}$ tracking all updated in $\mathcal{S}$
  $\{\mathbf{v}_i\} \leftarrow select(\mathbf{p}, radius)$; $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{v}_i\}$
  // compute deformed positions by applying brush transform
  forall the *selected vertices* $\mathbf{v}_i$ **do** $\mathbf{v}'_i \leftarrow transform(\mathbf{v}'_i, M)$
  // update positions
  if *no collision handling (standard sculpting)* **then** $\mathbf{v}_i \leftarrow \mathbf{v}_i$
  if *collision handling (*sLayer*)* **then**
    // compute collisions
    $C \leftarrow collisions(\{\mathbf{v}_i \rightarrow \mathbf{v}'_i\}, layers)$;
    forall the *non-colliding vertices* $\mathbf{v}_i \notin C$ **do** $\mathbf{v}_i \leftarrow \mathbf{v}'_i$
    // collision response
    forall the *colliding vertices* $\mathbf{v}_i \in C$ **do**
      $\hat{\mathbf{v}}_i \leftarrow collide(\mathbf{v}_i \rightarrow \mathbf{v}'_i, layers)$ // safe position
      $\mathbf{v}_i \leftarrow \hat{\mathbf{v}}_i$
    end
  end
end
// remesh triangles adjacent to all updated vertices
$remesh(\mathcal{S})$

Algorithm 1: Brush application with and without collision handling.

Reflectance Editing. We edit reflectance by painting BRDF parameters per vertex. The editing toolset does not depend on the specific BRDF model. In our preview renderer we support microfacet BRDF with the Phong distribution. Material applications is similar to the application of deformation, and in fact integrated with it. At each brush stroke, BRDF parameters are a weighted sum between the layer BRDFs and the brush one. To do so, we use a linearized BRDF model so that linear weighted sums maintain their semantic. In our case, we choose the *AppIm* [DRCP14] linear parametrization that works well when editing microfacet BRDFs and geometry together. We support the standard brush texturing interface, which we augment with full BRDFs, that allows us to transfer detailed materials on the surface just like stencils.

Edge Length Assumption. In our system, we make the fundamental design choice of maintaining the mean edge length constant during the editing sessions. While this is not strictly necessary, it has the main advantage of maintaining a well-behaved triangulation while ensuring that deformations are sampled uniformly over the mesh surface. At the same time, the required remeshing operation is computationally significant for detailed meshes. For this reason, many sculpting systems do not provide such a feature. In our case though, this choice has the major advantage of simplifying significantly collision detection. In a way, we shift computation from collision handling to remeshing and we believe this tradeoff works really well.

One way to think about the mean edge length is that it measures the resolution of the material we can represent, since both local geometry and BRDFs are sampled on vertices. Based on this observation we make a second major design choice and set the accuracy for collision handling to be of the same order as the mean edge length. A higher accuracy would not be useful since the layers are sampled at the edge length resolution anyway.

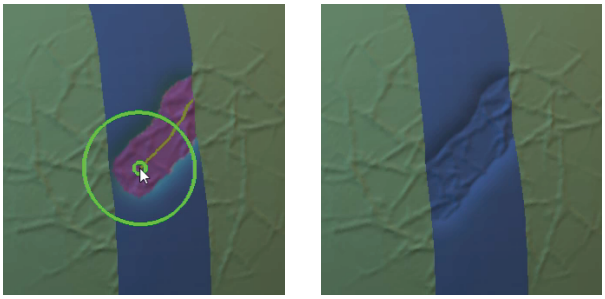

Figure 3: *Example of collisions handling during user interaction. The blue layer is pushed against the green one, picking up details from the lower features.*

Collision Handling. Collision detection and response are well studied problems in our field with many known solutions with tradeoffs that differ substantially in terms of data representation, scalability and accuracy. Choosing which tradeoffs are acceptable depends stricktly on the application domain [TKH*05, YKH*10].

One major difference between collision handling solutions is whether they use a mesh data structure or an alternate geometric representation, e.g. signed distance fields [XB14] or [MCKM15]. Since we need to handle million of polygons that are updated dynamically in real-time, we excluded these solutions that work typically well for static object or lower polygon ones. We settle on performing self-intersection directly on the mesh representation, using a broad-phase/narrow-phase design.

Grid-Based Broad Phase. In the broad phase, we maintain a spatial partitioning data structure that allows us to prune collision tests. Many of these data structures have been proposed, ranging from grid cells, object or axis aligned bounding boxes and n-polytopes [Eri04]. All these solutions have tradeoffs between pruning accuracy, build times and the type of meshes that they can handle well. We chose a regular grid, stored as a hash table [THM*03], since it can be very efficiently updated, even in parallel [ASA*09]. A uniform grid becomes efficient only when the edge length is homogeneoues, which in our case we ensure during remeshing. In our particular context, the sides of the grid cells are set to the average edge length of input surface and we insert vertex references in the grid itself. This ensure that the number of reference for each grid cell remain small and can be quickly updated during interaction. On the other hand, this only allows us to test vertex movements of at most the mean edge length, to ensure that all triangles are properly considered for intersection.

Proximity-Only Narrow Phase. In the narrow phase, the tradeoff depends on whether proximity queries are sufficient, or more complex continuous collision detection is necessary. The decision here depends on the amount of movement for each brush application and the desired precision. In our case, we observe that the accuracy required during collision handling, namely the mean edge length, is relatively large compared to the amount of vertex movement during each rasterization step. The reason for this is that sculpting systems require a high stroke polyline resolution to ensure that the brush stroke is well sampled. This implies that only relatively small movements happen when handling each polyline location. For these

```
// rebuild the grid G
G ← rebuild_grid(layers)
forall the brush strokes do
    forall the stroke positions p do
        // split deformation due to p into steps of maximum
        edge length e_l
        forall the edge length steps do
            // select influenced vertices using the grid
            {v_i} ← select(p, radius, G); S ← S ∪ {v_i}
            // compute deformed positions
            forall the selected verts v_i do
            v'_i ← transform(v'_i, M)
            // insert updated positions in the grid
            G ← insert(G, {v'_i})
            // compute possible vertex–triangle pairs with the
            grid
            forall the selected vertices v_i do
                C̃ ← C̃ ∪ broad_phase(v_i, G)
            end
            // compute actual collision and their locations
            forall the possibly colliding vertices v_i ∈ C̃ do
                C ← C ∪ narrow_phase(v_i, layers)
                v^c_i ← collision_location(v_i, layers)
            end
            forall the non-colliding vertices v_i ∉ C do
            v_i ← v'_i
            // push colliding vertices an edge-length e_l away
            forall the colliding vertices v_i ∈ C do
                v_i ← clamp(v_i → v^c_i, e_l)
            end
            // update the grid by removing old vertices
            G ← remove(G, {v_i})
        end
    end
    // remesh triangles adjacents to all updated vertices
    S' = remesh(S)
    // insert updated positions in the grid
    G ← insert(G, S')
end
```

Algorithm 2: *sLayer*'s brush application pipeline.

reasons, we used standard vertex–triangle proximity queries [Eri04]. For more accuracy we could include edge–to–edge tests without changing the overall system design, but in our current prototype we found them not necessary. This is common in many animation applications too. Finally, we considered the use of continuous collision detection and tested the solution presented in [Wan14]. This solution though did not perform significantly better in our testing then the simple one, so we settle on the simplest design.

***sLayer* Brush Application.** Alg. 2 shows the *sLayer* brush application pipeline. Before each stroke, we build a regular grid with all vertices inserted. We choose to fully rebuild the grid since large deformations might have happened during the last user interaction. Then, for each stroke location, we perform standard stroke application aided by the grid. Since we insert just the vertices into the grid,

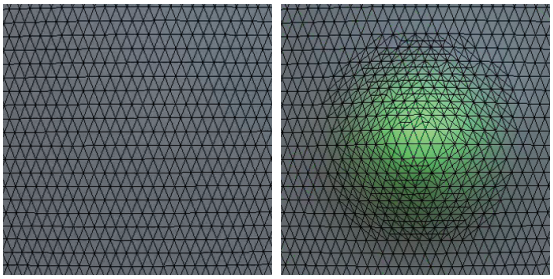

Figure 4: *Remesh example.* Left. *Initial surface.* Right. *Surface remeshed after deformation.*

rather than whole triangles, we handle each stroke location with multiple steps where the maximum vertex translation is the edge length, i.e. the grid cell side.

In each step, we quickly select affected vertices by rasterizing a sphere into the grid, centered at the brush location and with radius equal to the brush radius; we pick all the vertices for the intersected cells. We compute the new vertex location by applying the brush deformation with a standard method. We then insert the new vertex location in the grid, in order to track all new vertex positions during intersection handling, and find all vertex–triangle pairs that are closer than the mean edge length, pruning the tests first with the grid. All vertices that have no collision are simply moved to the transformed locations. For the colliding ones, we place the vertices one edge length away from the collided triangle in the direction of motion. This can be considered as our collision response. We finally remove the old vertex locations from the grid. To update the grid data structure efficiently, for both insertion and deletion, we maintain a list of vertex-to-cell correspondences and swap references when possible. This minimizes memory allocation during editing.

It should be noted that different collision responses could have been employed in this context and recent works address this issue particularly in the context of shape modelling, e.g. [HPSZ11]. We chose to simply lock the vertex, rather than perform a physically-sound response, since we want to maintain user edits exactly and introduce no deformation which is not directly controllable by the brush.

Remeshing. Among the many known remeshing schemes (see [AUGA08] for a survey), we peform isotropic remeshing using the approach described in [DVBB13] based on successive iterations of mesh edge improvements followed by vertex repositioning. Fig. 4 shows an example. We chose this method based on its performance and relative simplicity.

The remeshing method requires adjacency information that would be provided by a complex data structure such as half-edges. Our concern with maintain such data structures it that they add complexity to the system, but do not improve any other part of the pipeline. For this reason, we settle on augmenting our index triangle meshes with explicit adjacency stored for vertex–vertex, vertex–face and face–face and reconstruct all other adjacies from here.

Discussion. In the end, our material editing system remains relatively simple given the combination of three design choices. First, the constant mean edge choice was crucial in our system design. It allowed us to use a single, easily updatable, grid for vertex select and broad-phase collision. In the narrow phase, the assumption translated into using simple proximity queries and a trivial collision response, like shown in Fig. 3. The tradeoff is to use more complex remeshing that it is forced to be executed once per stroke, rather than one for stroke's polyline location. Second, representing layers as indexed triangle meshes, with possibly disconnected patches, allows us to represent all orientable surface configurations while still leading to a fast and reliable collision handling, when combined with constant edge length. Both of these were surprising for us and in fact we prototype other more complex solutions first, but found this combination to be the key for a reliable and scalable system. Finally, we chose to avoid separate texture and vertex sampling for reflectance. This makes sure that reflectance edits are easily aligned with geometry deformation just like they are on real-world surfaces. Obviously, a mesh simplification method that used textures for reflectance and normal maps for geometry might be applied once the model is completed to achieve faster reproduction during rendering.

4. Results

We implemented *sLayer* as a multicore CPU application and gather testing results using a Intel i7-6700k CPU with 4 physical cores running at 4.0 Ghz, and 16 GB of RAM.

Editing Tools. Fig. 5 shows an example sequence created with the editing tools in our interface. The bulk of the modeling was done with brushes similar to [Ble16]'s ones, augmented with collision handling. Material painting was implemented as in *cSculpt* [CSTP16]. The major additional difference is that the artists can select which layer group the brush is active on, leaving the others undeformed, but still participating in collision handling. For layer creation, we used three different operations. The simplest one is the offsetted duplication of selected parts of a layer, similar to [SVJ15]. The second one is to draw a closed stroke onto a surface and create a new layer from it using discrete coons patches. The last one is to cut a layer along a stroke. While high fidelity cuts can be implemented [MSF*15], we use a simpler implementation that snaps the stroke to existing triangle edges, similarly to [NvdS04], given our short edge length and the fact that vertices and then remodeled with brushing.

Materials and Performance. The main strengths of *sLayer* are its interactivity and its scalability. This enabled us to model complex materials, as depicted in Fig. 1 and Fig. 6, where we demonstrate the expressiveness of our approach.

We choose three stratified materials of different nature: a piece of peeled plaster and rusted metal, each one composed by more than half a million of polygons, and a portion of tree bark. The final model of this latter is composed by approximately 940 thousands of triangles; during the modeling session, before deleting some of the fragments of the bark, we reached 1.2 millions of polygons.

During our tests, the tool always remained interactive (> 10fps) as demonstrated by the data reported in Table 1, including when a portion of a layer is deformed and collides with another layer, and remeshing is carried out on the fly. We are not aware of any existing

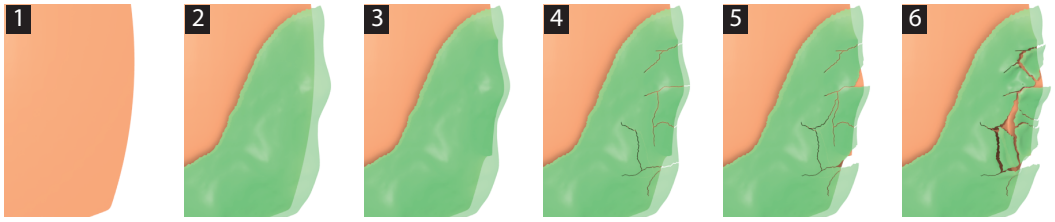

Figure 5: *Representative editing interactions offered by our system. From left to right: (1) plain surface; (2) adding patches; (3) deforming bottom layers against the one on top; (4) adding cuts on the top surface; (5) deforming and deleting some detached components (6) final result. Images renderer with raytracing.*

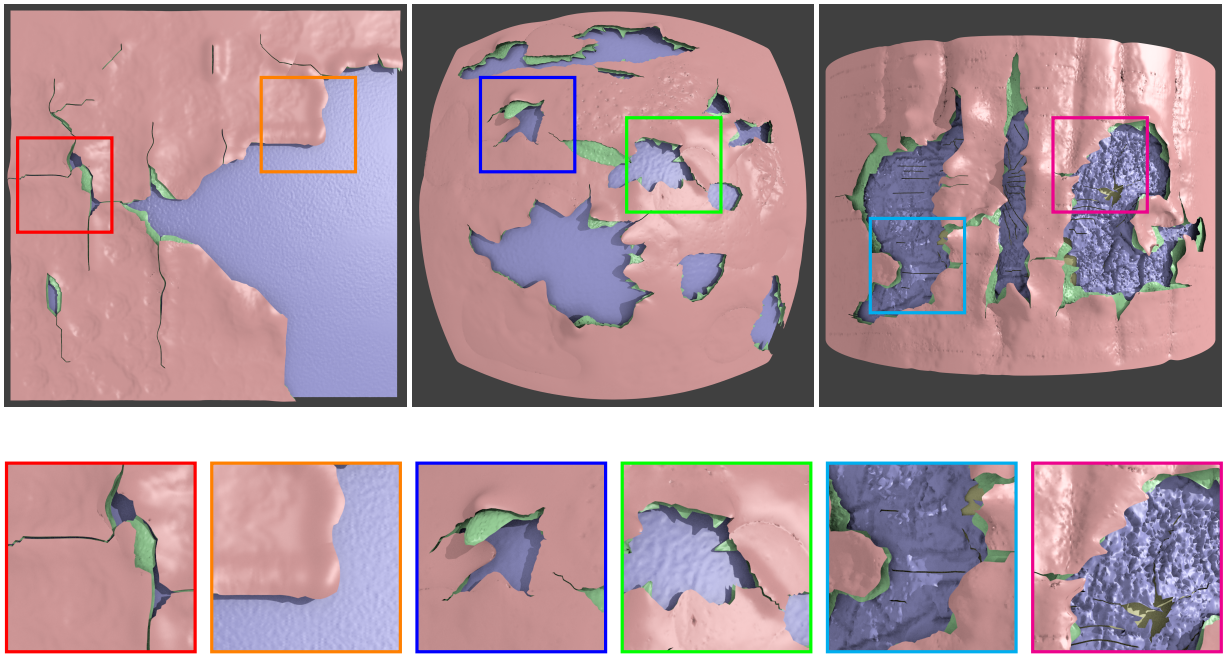

Figure 6: *Rendering of the test materials to highlight the high frequency details. In the bottom row, some details are magnified 2X.*

sculpting tool able to deliver such complex cases interactively, and *sLayer* handles this very well.

5. Conclusions

In this paper we present *sLayer*, a system for sculpting editing multi-layer materials capable of handling interactively detailed, million-polygons, surfaces with collision handling and remeshing. In the future, we plan to investigate how to model subsurface scattering with a brush like interface including the creation of holes and cracks within the volume.

6. Acknowledgments

This work has been partially funded by MIUR (project DSurf), Sapienza University of Rome and Intel Corporation. Marco Fratarcangeli was supported in part by the Swedish Research Council (project number: 2015-05345).

References

[ASA*09] Alcantara D. A., Sharf A., Abbasinejad F., Sengupta S., Mitzenmacher M., Owens J. D., Amenta N.: Real-time parallel hashing on the gpu. In *SIGGRAPH Asia '09* (2009), pp. 154:1–154:9. URL: http://doi.acm.org/10.1145/1661412.1618500, doi:10.1145/1661412.1618500. 4

[AUGA08] Alliez P., Ucelli G., Gotsman C., Attene M.: *Recent Advances in Remeshing of Surfaces*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 53–82. URL: http://dx.doi.org/10.1007/978-3-540-33265-7_2, doi:10.1007/978-3-540-33265-7_2. 5

[AWC06] Angelidis A., Wyvill G., Cani M.: Sweepers: Swept deformation defined by gesture. *Graphical Models 68*, 1 (2006), 2–14. URL: http://dx.doi.org/10.1016/j.gmod.2005.08.002, doi:10.1016/j.gmod.2005.08.002. 3

[BGF15] Barki H., Guennebaud G., Foufou S.: Exact, robust, and efficient regularized booleans on general 3d meshes. *Computers & Mathematics with Applications 70*, 6 (2015), 1235–1254. URL: http://dblp.uni-trier.de/db/journals/cma/cma70.html#BarkiGF15. 2

[Ble16] Blender: Blender. http://www.blender.org/, 2016. 1, 5

[BSL*16] Bernstein G. L., Shah C., Lemire C., Devito Z., Fisher M., Levis P., Hanrahan P.: Ebb: A dsl for physical simulation on cpus

Model	Layers	Vertices	Faces	Strokes	Time for Brush Application					
					Selection	Deformation	Grid Update	Collision	Remesh	Total
Peeled plaster	3	323K	640K	279	4.71%	0.31%	1.83%	66.37%	8.19%	0.11s
Tree Bark	4	496K	981K	6557	3.66%	0.30%	1.26%	68.13%	3.98%	0.13s
Rusted Metal	3	265K	525K	1100	4.80%	1.85%	2.29%	73.07%	7.00%	0.10s

Table 1: *Statistics gathered over the modeling sessions of the materials in Fig. 1 and Fig. 6. All values are averaged over all brush strokes.*

and gpus. *ACM Trans. Graph. 35*, 2 (2016), 21:1–21:12. URL: http://doi.acm.org/10.1145/2892632, doi:10.1145/2892632. 3

[CKGK11] Chaudhuri S., Kalogerakis E., Guibas L., Koltun V.: Probabilistic reasoning for assembly-based 3d modeling. In *SIGGRAPH '11* (2011), pp. 35:1–35:10. URL: http://doi.acm.org/10.1145/1964921.1964930, doi:10.1145/1964921.1964930. 2

[CSTP16] Calabrese C., Salvati G., Tarini M., Pellacini F.: csculpt: A system for collaborative sculpting. *ACM Trans. Graph. 35*, 4 (2016), 91:1–91:8. URL: http://doi.acm.org/10.1145/2897824.2925956, doi:10.1145/2897824.2925956. 5

[DPS15] De Paoli C., Singh K.: Secondskin: Sketch-based construction of layered 3d models. *ACM Trans. Graph. 34*, 4 (2015), 126:1–126:10. URL: http://doi.acm.org/10.1145/2766948, doi:10.1145/2766948. 2, 3

[DRCP14] Di Renzo F., Calabrese C., Pellacini F.: Appim: Linear spaces for image-based appearance editing. *ACM Trans. Graph. 33*, 6 (2014), 194:1–194:9. URL: http://doi.acm.org/10.1145/2661229.2661282, doi:10.1145/2661229.2661282. 3

[DVBB13] Dunyach M., Vanderhaeghe D., Barthe L., Botsch M.: Adaptive Remeshing for Real-Time Mesh Deformation. In *Eurographics 2013* (2013). URL: https://hal.archives-ouvertes.fr/hal-01295339, doi:10.2312/conf/EG2013/short/029-032. 5

[Eri04] Ericson C.: *Real-Time Collision Detection*. CRC Press, Inc., Boca Raton, FL, USA, 2004. 4

[FH11] Foley T., Hanrahan P.: Spark: Modular, composable shaders for graphics hardware. In *SIGGRAPH '11* (2011), pp. 107:1–107:12. URL: http://doi.acm.org/10.1145/1964921.1965002, doi:10.1145/1964921.1965002. 3

[HPSZ11] Harmon D., Panozzo D., Sorkine O., Zorin D.: Interference-aware geometric modeling. *ACM Trans. Graph. 30*, 6 (2011), 137:1–137:10. URL: http://doi.acm.org/10.1145/2070781.2024171, doi:10.1145/2070781.2024171. 5

[MCKM15] Müller M., Chentanez N., Kim T.-Y., Macklin M.: Air meshes for robust collision handling. *ACM Trans. Graph. 34*, 4 (2015), 133:1–133:9. URL: http://doi.acm.org/10.1145/2766907, doi:10.1145/2766907. 4

[MMCK14] Macklin M., Müller M., Chentanez N., Kim T.-Y.: Unified particle physics for real-time applications. *ACM Trans. Graph. 33*, 4 (2014), 153:1–153:12. URL: http://doi.acm.org/10.1145/2601097.2601152, doi:10.1145/2601097.2601152. 3

[MSF*15] Manteaux P.-L., Sun W.-L., Faure F., Cani M.-P., O'Brien J. F.: Interactive detailed cutting of thin sheets. In *Motion in Games '15* (2015), pp. 125–132. URL: http://doi.acm.org/10.1145/2822013.2822018, doi:10.1145/2822013.2822018. 5

[NvdS04] Nienhuys H.-W., van der Stappen A. F.: *A Delaunay Approach to Interactive Cutting in Triangulated Surfaces*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 113–129. URL: http://dx.doi.org/10.1007/978-3-540-45058-0_8, doi:10.1007/978-3-540-45058-0_8. 5

[Pai16] Painter S.: Substance painter. https://www.allegorithmic.com/products/substance-painter, 2016. 1

[PBFJ05] Porumbescu S. D., Budge B., Feng L., Joy K. I.: Shell maps. In *SIGGRAPH '2005* (2005), pp. 626–633. URL: http://doi.acm.org/10.1145/1186822.1073239, doi:10.1145/1186822.1073239. 2

[SCCS13] Stanculescu L., Chaine R., Cani M.-P., Singh K.: Sculpting multi-dimensional nested structures. *Computers and Graphics 37*, 6 (2013), 753–763. URL: https://hal.inria.fr/hal-00865552, doi:10.1016/j.cag.2013.05.010. 2, 3

[SS08] Schmidt R., Singh K.: Sketch-based procedural surface modeling and compositing using surface trees. *Computer Graphics Forum 27*, 2 (2008), 321–330. URL: http://dx.doi.org/10.1111/j.1467-8659.2008.01129.x, doi:10.1111/j.1467-8659.2008.01129.x. 2

[SS10] Schmidt R., Singh K.: Meshmixer: An interface for rapid mesh composition. In *SIGGRAPH '10 Talks* (2010), pp. 6:1–6:1. URL: http://doi.acm.org/10.1145/1837026.1837034, doi:10.1145/1837026.1837034. 2

[SVJ15] Sacht L., Vouga E., Jacobson A.: Nested cages. *ACM Trans. Graph. 34*, 6 (2015), 170:1–170:14. URL: http://doi.acm.org/10.1145/2816795.2818093, doi:10.1145/2816795.2818093. 2, 5

[THM*03] Teschner M., Heidelberger B., Mï£¡ller M., Pomerantes D., Gross M. H.: Optimized spatial hashing for collision detection of deformable objects. In *VMV* (2003), Ertl T., (Ed.), Aka GmbH, pp. 47–54. URL: http://dblp.uni-trier.de/db/conf/vmv/vmv2003.html#TeschnerHMPG03. 4

[TKH*05] Teschner M., Kimmerle S., Heidelberger B., Zachmann G., Raghupathi L., Fuhrmann A., Cani M.-P., Faure F., Magnenat-Thalmann N., Strasser W., Volino P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (2005), 61–81. URL: http://dx.doi.org/10.1111/j.1467-8659.2005.00829.x, doi:10.1111/j.1467-8659.2005.00829.x. 4

[Wan14] Wang H.: Defending continuous collision detection against errors. *ACM Trans. Graph. 33*, 4 (2014), 122:1–122:10. URL: http://doi.acm.org/10.1145/2601097.2601114, doi:10.1145/2601097.2601114. 4

[WR16] Wrenninge M., Rice M.: Volume modeling techniques in the good dinosaur. In *SIGGRAPH '16* (2016), pp. 63:1–63:1. URL: http://doi.acm.org/10.1145/2897839.2927397, doi:10.1145/2897839.2927397. 3

[XB14] Xu H., Barbič J.: Continuous collision detection between points and signed distance fields. *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS 2014* (2014). 4

[YKH*10] Yoon S.-e., Kim Y. J., Harada T., Kim Y. J., Yoon S.-e.: Recent advances in real-time collision and proximity computations for games and simulations. In *SIGGRAPH Asia '10 Courses* (2010), pp. 22:1–22:110. URL: http://doi.acm.org/10.1145/1900520.1900542, doi:10.1145/1900520.1900542. 4

[ZBr16] ZBrush: Zbrush. http://pixologic.com//, 2016. 1

[ZGZJ16] Zhou Q., Grinspun E., Zorin D., Jacobson A.: Mesh arrangements for solid geometry. *ACM Trans. Graph. 35*, 4 (2016), 39:1–39:15. URL: http://doi.acm.org/10.1145/2897824.2925901, doi:10.1145/2897824.2925901. 2

[ZHW*06] Zhou K., Huang X., Wang X., Tong Y., Desbrun M., Guo B., Shum H.-Y.: Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3 (2006), 690–697. URL: http://doi.acm.org/10.1145/1141911.1141942, doi:10.1145/1141911.1141942. 2