Volume-encoded UV-maps

Marco Tarini*
Università dell'Insubria, Varese *and* ISTI-CNR, Pisa

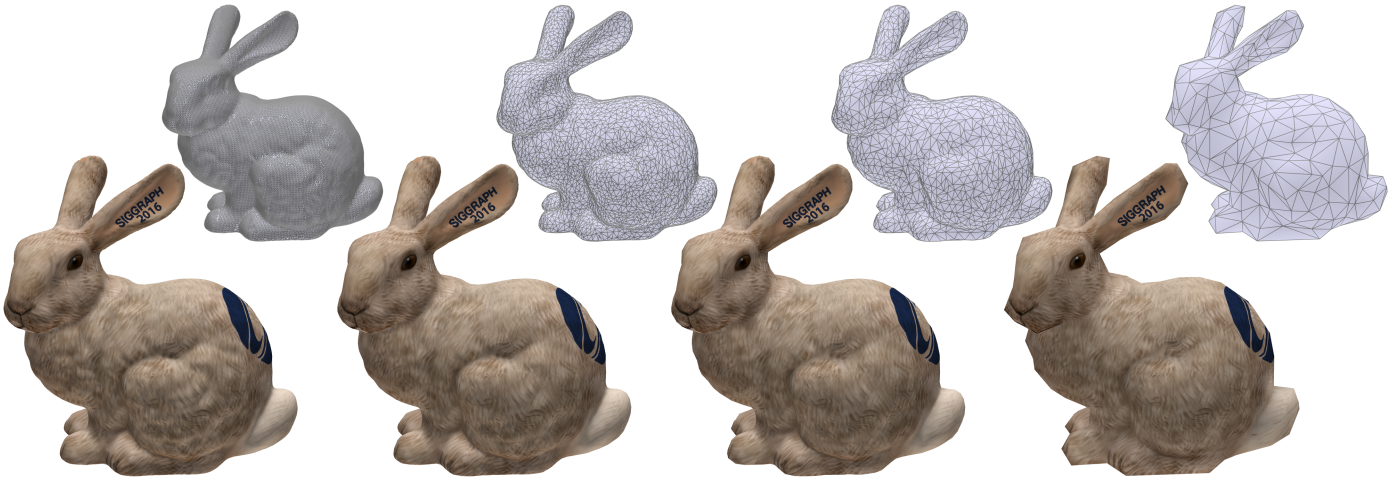

Figure 1: *The same volume-encoded UV-map is used to apply the same 2D texture (in Fig. 2) over all levels of a LoD pyramid. The models (64K, 10K, 1K, and 650 faces) are produced with an unconstrained automatic simplification tool [Cignoni et al. 2011], unaware of the UV-map, and consist in "naked" meshes, with no uv associated to vertices.*

Abstract

UV-maps are required in order to apply a 2D texture over a 3D model. Conventional UV-maps are defined by an assignment of uv positions to mesh vertices. We present an alternative representation, volume-encoded UV-maps, in which each point on the surface is mapped to a uv position which is solely a function of its 3D position. This function is tailored for a target surface: its restriction to the surface is a parametrization exhibiting high quality, e.g. in terms of angle and area preservation; and, near the surface, it is almost constant for small orthogonal displacements. The representation is applicable to a wide range of shapes and UV-maps, and unlocks several key advantages: it removes the need to duplicate vertices in the mesh to encode cuts in the map; it makes the UV-map representation independent from the meshing of the surface; the same texture, and even the same UV-map, can be shared by multiple geometrically similar models (e.g. all levels of a LoD pyramid); UV-maps can be applied to representations other than polygonal meshes, like point clouds or set of registered range-maps. Our schema is cheap on GPU computational and memory resources, requiring only a single, cache-coherent indirection to a small volumetric texture per fragment. We also provide an algorithm to construct a volume-encoded UV-map given a target surface.

Keywords: texture mapping, seamless parameterization
Concepts: •**Computing methodologies** → **Texturing;**

*e-mail:marco.tarini@isti.cnr.it

SIGGRAPH '16 Technical Paper, July 24 - 28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925898

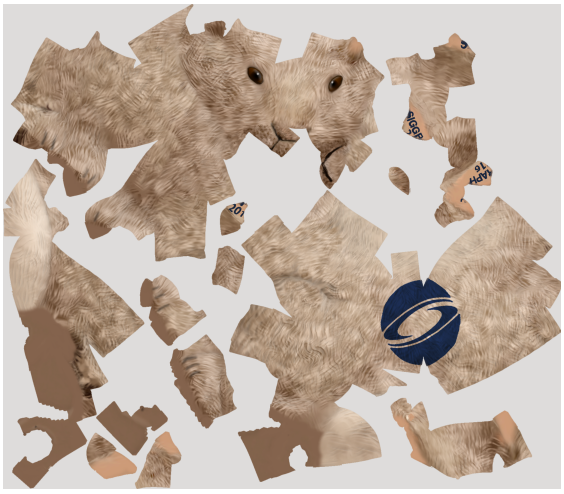

Figure 2: *Artist painted 2D texture (2048^2 pixels) used in Fig. 1.*

1 Introduction

Texture mapping is a fundamental mechanism in Computer Graphics which consists in enriching a surface S with a signal (e.g. colors or normals) from a $2D$ texture image T.

Texture mapping necessitates that every point of the surface S is mapped into a $2D$ position uv in texture space $\Omega = [0,1]^2$. This is termed the **UV-map** of S. A UV-map is a form of surface parameterization, but here we use the former term because we focus on texture mapping applications, which impose a specific set of requirements.

It is well understood that UV-maps require **cuts** (also called seams); this is both for topological reasons (when S is not a topological disk) and to reduce the unavoidable distortions introduced by the map (when S is not developable).

1.1 Overview

Traditionally, the UV-map of a polygonal mesh S is represented by explicitly assigning uv positions as per-vertex attributes. Standard linear interpolation inside faces provides a uv position for each point on S; cuts are encoded by means of duplicating some vertex and assigning different uv attributes to the two geometrically coinciding copies.

In contrast, we introduce **volume-encoded UV-maps**, where a 3D point p on S implicitly has a uv position given by

$$(u, v) = f(p) \qquad f : \mathbb{R}^3 \rightarrow \Omega$$

regardless of the belonging face. During fragment processing, f is evaluated at current point p, then the final texture is accessed at the resulting position (u, v), as usual (e.g. with bi-linear interpolation and MIP-mapping). Required cuts are, simply, discontinuities of f.

Function f is tailored, beforehand, for a given target surface S, seeking two objectives: the restricted map between S and $f(S)$ must be a good UV-map for S, e.g. it must present low distortions, injectivity, a good packing of texture space, etc; in the proximity of S, and for small displacements along directions orthogonal to S, f must be close to constant. We refer to the latter property as the **orthogonality** of f.

The challenge is to represent f so that it is extremely efficient to evaluate, yet expressive enough that the two above objectives can be met for a given target surface S. Our solution is based on defining f as the trilinear interpolation of (u, v) values defined in a coarse, regular volumetric grid, stored in a small 3D texture, and is detailed in Section 3. In order to demonstrate its usability, in Section 4, we show one heuristic to construct an f tailored for a given surface S.

One central aspect of our representation is the extremely small computation overhead imposed at rendering time (a single extra indirection); this works well for a large set of surfaces, but, as a limitation, it is not as general as per-vertex UV-map for shapes where tiny features are present and important (see Sec 7.1).

1.2 Motivations

Our volume-encoded UV-map schema offers multiple *qualitative* advantages over traditional, per-vertex ones:

Connectivity independence. Neither the UV-map, nor the 2D texture are tied to the way S is meshed. Since function f is computed per fragment, cuts of the UV-map need not be present in a mesh as sequences of edges, and mesh triangles can freely span across cuts without causing any artifact. Therefore, the same UV-map and the same 2D texture can be applied directly to any remeshing of the original mesh.

Geometry independence. Thanks to the *orthogonality* of f, the above point extends also to meshes having a similar but not identical geometric shape to S. For example, all the levels of detail models of a LoD pyramid can share both the same UV-map and texture (see Fig. 1 and many other examples in the demo).

No vertex duplications. Even considering a single polygonal mesh with a given connectivity, an advantage is that no *vertex duplication* is required, simplifying data structures[1] (e.g. augmenting procedurality, and bypassing the need of keeping the copies consistent during any processing).

[1] With per-vertex UV-maps, a workaround to avoid vertex replications is to assign uv positions not as per-vertex attributes but as per-wedge attributes. This just means replicating uv positions everywhere instead of replicating vertices at seams.

Representation independence. Because it does not rely on attribute interpolations inside faces, this schema is not limited to polygonal meshes. It can be directly applied on other surface representations, like point clouds (see Fig. 12), triangle soups with no consistent connectivity, collections of registered range scans (see Fig.10), or ray-traced implicit surfaces. For example, both the UV-map and the texture built for a set of range-scans or a point cloud can be directly employed over the two-manifold mesh resulting from their subsequent fusion.

Compactness. The UV-map is encoded independently from the mesh, and no uv position is stored (or sent) per vertex. The amount of memory used to encode f depends on the complexity of the overall coarse shape of the object, not on the tessellation density necessary to describe its fine-grained geometry, and can be (but is not always) much smaller.

Robustness of construction. The UV-map construction can safely ignore local mesh inconsistencies, like non two-manifold configurations, topological noise, holes etc.

2 State of the Art

Surface parameterization is an heavily studied subject (see [Hormann et al. 2007; Sheffer et al. 2006] for surveys), and its use as UV-maps for texturing is invariably included among the intended applications. Most of the concepts devised in this literature are still relevant to the case of our volume-encoded UV-maps, so we briefly summarize them here.

***Single-patch* parameterization techniques** target disk-like surfaces; typical sought objectives include: *area preservation*, *conformality* (angle preservation), *isometry* (implying both), and *injectivity* (implying absence of fold-overs). Lack of preservation of angles or areas, referred to as *distortion*, has to be minimized but is unavoidable for any non-zero Gaussian curvature surface. A generalization of seeking uniform areas preservation is to aim at *adaptivity* (also referred to as *signal specialization*), where more texture area is devoted to more important regions.

***Global* parameterization techniques** introduce seams to "cut open" a surface with any topology into one or more disks, thus reducing the problem to the previous case. Cuts also serve the purpose of trading Gaussian curvature inside patches for line curvature at their boundary, reducing distortions. A UV-map where cuts split S into separate disks, each mapped into a separate *chart*, is referred to as an *Atlas-based* UV-map (e.g. [Sander et al. 2003; Zhou et al. 2004; Pietroni et al. 2010]). In other approaches (e.g. [Gu et al. 2002; Pietroni et al. 2011]), S is mapped into a *single chart*.

Global techniques add more items to the list of sought objectives. Some of these are less crucial for UV-maps, compared to other applications of parameterizations like remeshing; these include *global smoothness* (implying that $u-v$ isolines are continuous across cuts, up to prescribed rotations), *alignment* to geometric features or to an input cross field defined over S. Vice versa, objectives like lack of global overlaps in uv space, good coverage of uv space (e.g. packing of patches in an Atlas), and an appropriate placement of cut lines, are crucial for UV-maps, but can be disregarded (and often are) when the parameterization is intended for remeshing.

Importantly, all these concepts are orthogonal to the representation of the UV-map. Just like per-vertex ones, our volume-encoded UV-maps can feature variable degrees of: *distortions*, *injectivity*, *adaptivity*; they, too, can be either *single-patch* or *global*, *Atlas-based* or not, *adaptive* or not, and so on.

2.1 Alternatives representations for UV-maps

In the literature several alternatives have been proposed to conventional per-vertex UV-maps defined per-vertex over triangle meshes. Each alternative achieves some subsets of the advantages of volume-encoded UV-maps listed in Sec. 1.2, through different routes, and with important differences.

Connectivity based representations. In *Ptex* [Burley and Lacewell 2008] and *Mesh-Color* [Yuksel et al. 2010] techniques, the high-frequency signal (e.g. color) is sampled at each mesh element independently, regardless of its shape. In Ptex, a small texture is automatically assigned at each polygonal element; in Mesh-Color, samples are arranged in a predefined pattern inside each triangle. Like in our case, no uv needs be stored (or sent to GPU) per vertex. However, this is achieved employing exactly the opposite strategy: the uv is fully determined by the connectivity of the mesh, instead of being fully independent from it. As a result, the signal for a given surface cannot be used if the surface is remeshed arbitrarily – for example, if it is coarsened (in Ptex, it can be used, however, if it is refined regularly, e.g. within the GPU). Moreover, the sample distribution on the surface is also dependent on the meshing.

Volume based representations. If the high-frequency signal was sampled in the 3D volume embedding S, any problem linked to the mapping would be avoided. A naive implementation of this idea has a prohibitive cubic cost of memory occupancy, but countermeasures have been proposed. *Spatial hashing* techniques [Lefebvre and Hoppe 2006; García et al. 2011] can be used to store only the non-empty cells in a virtual 3D texture. However, there is no *geometry independence* (for example a simplified model would likely pass in places where no signal is defined). If bilinear interpolation is to be supported, the total memory occupancy of [Lefebvre and Hoppe 2006] for the same number of samples is about one order of magnitude larger than in our schema (see Sec. 6 for a quantitative comparison). The signal samples, being regularly spaced in a 3D grid, are poorly distributed on the 2D surface, and their density cannot be made *adaptive*. The same considerations (except adaptivity) apply to the techniques striving to compress a volumetric texture which encodes a signal only in a small volume around a given surface, by means of *octrees* [Benson and Davis 2002; Lefebvre et al. 2005] or similar structures, like *brickmaps* [Christensen and Batali 2004]. The per-fragment workload and total memory cost are even higher than with hashing (as shown in [Lefebvre and Hoppe 2006]).

Other connectivity unaware representations. As mentioned, in standard schemes cuts constrain mesh connectivity and impose the presence of replicated vertices. There are, however, a few exceptions. These are: the GPU hard-wired cube-map texture mechanism (when used to texture map a sphere-like surface), the special cylindrical and toroidal UV-maps [Tarini 2012] (only for surfaces with the respective topology), and, for general topologies, Polycube-maps [Tarini et al. 2004] and TileTrees [Lefebvre and Dachsbacher 2007; Dachsbacher and Lefebvre 2008]. The common mechanism behind all these solutions, and ours, consists in a per-fragment processing capable of re-directing a final 2D texture access to the proper location, in a way that is aware of cuts. In *Polycube-maps*, S is mapped over the surface of a textured polycube, via an per-vertex uv assignment (which requires three parametric coordinates rather than two). The per-fragment workload is considerably heavier than in our case. Also, the automatic robust construction of polycube-based UV-maps for a given S is a difficult problem, still unsolved despite intensive research (e.g. [He et al. 2009; Xia et al. 2011]), more so than with traditional maps. In *tiletrees*, squared texture patches are kept in the leaf nodes of an octree structure, allowing for a robust and easy construction. This share with our work the idea that the final uv positions for a point is defined as a function of its position (and, there, its normal), avoiding per-vertex storage. In tiletrees, exploitation of normals allows to deal better with tiny mesh features than in our case, due to the likely differences in normals. However, limited geometry independence is offered; e.g. a low res and high-res models cannot share the same UV-map or texture, because the former cannot reproduce the normals of the latter. Moreover, the per-fragment workload is orders of magnitude more demanding (requiring dozens of texture accesses to traverse a hierarchy), and the overall storage is also considerably heavier, as we show later. The quality of the map is also drastically worse, both in terms of cuts and distortion (see Fig. 11).

With respect to all alternatives, an important additional benefit offered by our representation is the ability to express maps which are comparable, in customizability and quality, to traditional per-vertex ones (see Fig. 11). For example, we fully preserve the fruitful analogy between textures and 2D images, thus making it possible to reuse 2D painting tools over T, or to apply 2D image filters, to define 3D tangent directions (needed for tangent-space normal maps), and, in general, to embed our new representation in existing production and usage pipelines (see Sec. 7).

2.2 Special UV-map construction techniques

Although we are mainly interested in proposing a novel representation (Sec. 3), our construction approach (Sec. 4) can be compared to existing unconventional parametrization techniques.

Construction not over manifold meshes. Techniques have been presented which are able, as ours, to produce UV-maps over surfaces initially represented by means other than two-manifold triangle meshes, like rangescan collections [Pietroni et al. 2011], point clouds [Zhang et al. 2010; Jakob et al. 2015]. Even if the targeted application is remeshing rather than texturing (for example, global overlaps in parametric space are not addressed), the results could probably be adapted. The produced map is still defined by the per-sample assignment of uv positions (and is therefore tied to the surface used as input). Still, a form of *representation independence* is reached: one difference is that in our case not only the construction process but also the final result is independent from the original representation of the surface.

Simplification aware UV-maps. Parameterizations which are defined over *simple* domains, e.g. featuring straighter and fewer cuts, impose fewer constraints in the connectivity of meshes using them as UV-maps [Tarini et al. 2011; Gu et al. 2002], and sometimes [Sander et al. 2001; Purnomo et al. 2004] this an explicit motivation for doing so. These approaches make the map more *connectivity independent*, but some degree of dependence remains (e.g. adaptation of simplification algorithms is still required). Also note that this does not avoid the need of vertex duplications in data structures.

Parameterizations constructed as restrictions. Our construction method can be considered akin to a small class of alternative approaches to geometry processing, consisting in working on volumetric basis functions (hat functions, in our case) and restricting the resulting 3D function to the surface e.g. [Chuang et al. 2009]; this approach has been explored for example to successfully speed-up parametrization of densely tessellated meshes [Panetta et al. 2012], intended for remeshing (e.g., taking care of continuity of parametric lines across cuts). A conceptual difference is that we are not interested solely in the restriction of f on S, but also on its behavior in the *proximity* of S. For example, these solutions achieve *representation* and *connectivity independence*, like ours, but limited *geometry independence*, because *orthogonality* is not explicitly sought. Note also that our specific representations, targeting real-time texture mapping, is designed around evaluation efficiency and GPU friendliness.

3 Volume-encoded UV-maps

First, we focus on the simpler case of a map defined over disk-like surface, so that f can be continuous everywhere (Sec. 3.1), then we show how to represent cuts (Sec. 3.2).

3.1 Disk-like surfaces

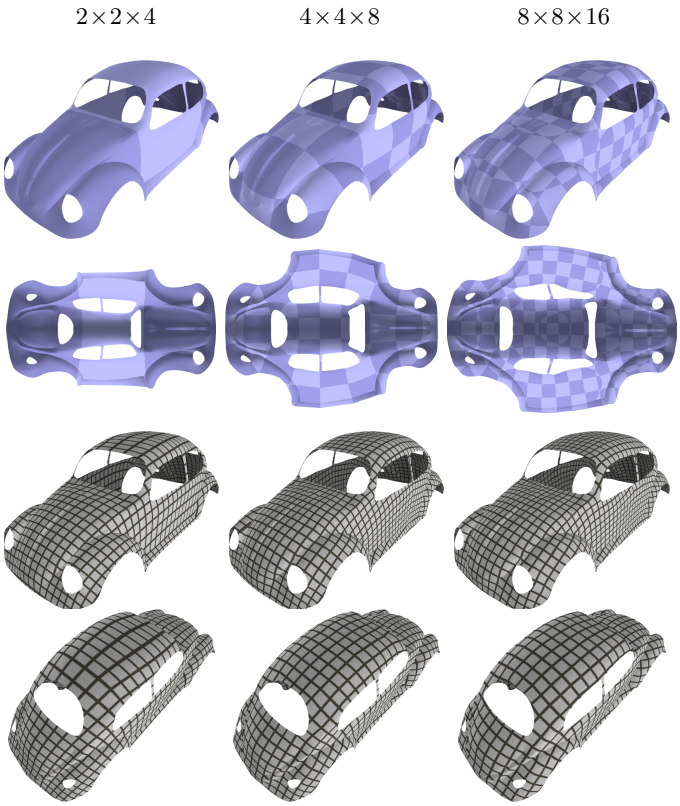

Figure 3: *Effect of resolution on volume-encoded UV-map quality. A single patch volume-encoded UV-map computed at three different resolutions (reported on top). Top row: the surface S is color coded according to belonging* voxel. *(e.g. on left, there are 1×1×3 voxels). Second row: uv space. Two bottom rows: uv isolines on the 3D model. As revealed visually and by measurements (table 1), higher resolutions allow for more isometric maps.*

The domain of f is a solid 3D axis-aligned box B containing surface S. First, we trivially map B into a '**logical space**' $L \subset \mathbb{R}^3$:

$$L = [0, s_x - 1] \times [0, s_y - 1] \times [0, s_z - 1]$$

though a conformal-scale-and-translate operation $\sigma : B \to L$. The three natural numbers $s_{[x,y,z]}$ are the **logical resolution** of the volume-encoded UV-map (which is unrelated to the resolution of the 2D texture T).

We store a uv position (in $\Omega = [0, 1] \times [0, 1]$) at each integer position of L. A unit-sized cube in L, with its eight corners at integer locations, is termed a **voxel**.

Our mapping function f is given by

$$f(p) = \tau(\sigma(p)) \tag{1}$$

where function $\tau : P \to \Omega$ is piecewise defined, inside each voxel of logical space, as the trilinear interpolation of the values stored at the corners of that voxel:

$$\tau(q) = \sum_{a,b,c \in \{0,1\}^3} \hat{q}_x^a \, \hat{q}_y^b \, \hat{q}_y^c \begin{pmatrix} u_{abc} \\ v_{abc} \end{pmatrix} \tag{2}$$

where (u_{abc}, v_{abc}) is the uv position stored in P at the (a, b, c) corner of the voxel containing q, scalars $\hat{q}^1_{[x,y,z]}$ are the fractional part of the x, y or z coordinate of q, and $\hat{q}^0_{[x,y,z]} = 1 - \hat{q}^1_{[x,y,z]}$.

The choice of values $s_{x,y,z}$ reflects different balances between conciseness and expressive power: smaller values reduce memory occupancy, larger values result in more freedom at expressing UV-maps with, potentially, higher quality (see Fig. 3), and are necessary for complex overall shapes. Typical values range between 4 and 64.

GPU evaluation of f. At rendering time, uv values of P are stored as texels of a two channeled 3D volumetric texture of the same resolution. Evaluation of $f(p)$, in the fragment processor, consists simply in applying scale-translation $q = \sigma(p)$ (a single Multiply-And-Add operation, which can be lifted to vertex computations) and a single trilinearly interpolated texture access at q, which returns $\tau(q)$. This operation substitutes the linear interpolation of per-vertex uv in traditional schemas. Note that we exploit the native hardware-supported trilinear interpolation of texels values, which is highly optimized in term of speed and GPU/texture-RAM bandwidth. Moreover, the access is extremely cache-coherent. The returned value is used directly as the address for the final texture T.

Rationale. Even if f needs be evaluated only in proximity of $\sigma(S)$, we store values in a simple, padded regular grid, foregoing any hierarchical structure. This choice is critical to achieve a undemanding constant time evaluation of f at rendering time. The strengths (and the limitations) of our approach stem from this choice. The key observation is that uv values can be made to vary smoothly and regularly over S, contrary to the signal stored in the final 2D texture T; only low resolutions are typically needed for P, also thanks to the expressive power of rectilinear interpolations, making its memory occupancy contained despite the volumetric nature.

3.2 Encoding cuts

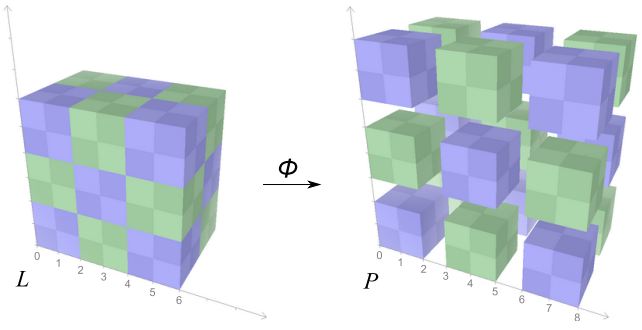

Figure 4: *Function ϕ, with zone side $k = 2$, mapping a logical space L of res 7×7×5 (composed of 6×6×4 voxels), into physical space P; a unit sized gap is added by ϕ to separate each zone. Here and in other images, different zones are alternatingly colored green and blue, and different voxels within a zone are colored with alternating darker and lighter shades.*

So far, f is C^0 continuous everywhere by construction. In the general case we need to introduce discontinuities in f, which corresponds to cuts in the UV-map of S. We partition logical space L into blocks of $k \times k \times k$ voxels called **zones**, for a given small integer

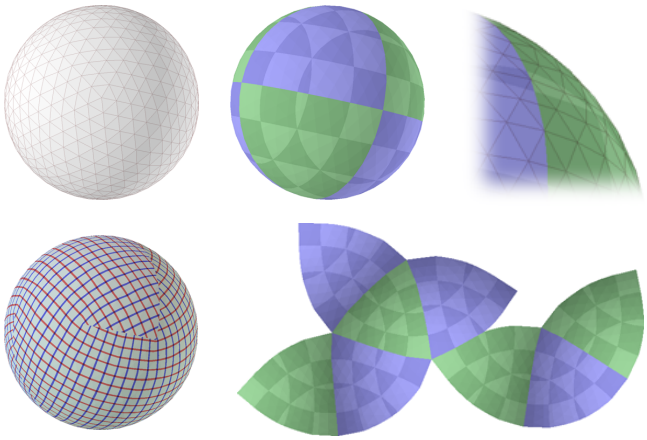

Figure 5: *A simple example of volume-encoded UV-map applied over a sphere model. Top: S colored according to belonging voxel and block (see color coding in Fig. 4). Note the triangles spanning across the cut, in the wireframe (close-up, top-right). Bottom: uv lines over S, and uv space.*

k. The idea is to define the map within each zone independently, so that f is allowed to be discontinuous across zones.

To implement this, we now differentiate between the *logical space* L, where the surface is mapped to by σ, and a *physical space* P, where uv values are stored at integer locations. P accommodates a single layer of **gap voxels** between *zones* of L (Fig. 4).

The function ϕ which maps a position in L into the corresponding position in P is:

$$\phi(q) = q + \lfloor q/k \rfloor \tag{3}$$

(where $\lfloor\ \rfloor$ is the component-wise floor operator). In place of (1), we re-define f as:

$$f(p) = \tau(\ \phi(\ \sigma(\ p\ )\ )\ ) \tag{4}$$

Intuitively, the role of ϕ is skip over *gap voxels*, so that function τ never gets computed inside them; any uv value stored in P belongs to exactly one zone, and interpolation τ never occurs between uv values belonging to different zones.

Consider two neighboring voxels of L belonging to different zones, and their shared quadrangular face. At each of the four corners of the face, uv values can be chosen independently for the two voxels (beacuse they correspond to distinct integer positions of P). If every uv pair are chosen to be matching, f will be continuous across the two voxels; otherwise, a discontinuity will occur across the face, which then we term **cut-face**. Note that, of the $k \times k$ *potential cut-faces* separating two adjacent zones, we can elect all, none, or only some to be (actual) *cut-faces*.

The final cut-lines on S are the intersections between $\sigma(S)$ and the *cut-faces*.

GPU evaluation of f. The additional computation costs to add discontinuities is negligible, amounting to a per-fragment evaluation of (3). Evaluation of sub-expression qk^{-1} can be lifted to the vertex processor. This leaves only a flooring and an addition operation per fragment.

Rationale. In our schema, the choice of value k determines a grid of *potential cut lines*, any subset of which can then be elected as actual cut lines (see Fig. 4). This limits the freedom of picking the cut lines for a map, but the benefit, once again, is an immediate evaluation of f, dealing with cuts without adding another level of indirection.

Memory costs. For smaller values of k, the grid of potential cut lines become denser, but, for the same a logical space L resolution, the occupancy of P is increased by a factor $(k+1)^3/k^3$, which is:

k	1	2	3	4	5	6	7	8	9
factor	8.00	3.38	2.37	1.95	1.73	1.59	1.49	1.42	1.37

4 Construction

We now present a possible procedure to construct a volume-encoded UV-map for an input target surface S.

Objectives. In the restriction of f on S, we seek the set of characteristics of a parameterization making it useful for texture mapping applications: we aim to maximize isometry (preservation of both angles and areas) [Hormann et al. 2007]; we go for injective maps, i.e. we want to avoid *local* and *global overlaps* in uv space; we strive to keep the amount of cutting (discontinuities of f) low; we want most of the texture area to be covered by $f(S)$, to avoid wasting texture RAM; optionally, we can target a prescribed local ratio r between surface area and texture area, with r a scalar varying over S, so that parts of S which are more important are mapped into larger texture regions.

Aside from these aims, which are shared by per-vertex UV-maps schemes, we also target a new objective over f, *orthogonality* (see Sec. 1.1), needed for the *geometry independence*.

Preliminary: sampling S. First, S is sampled, and the set of geometry **surface samples** $\{s_0, .., s_n\}$ will serve as the sole input of our UV-map construction. Each surface sample s_i represents a small portion of S and comes with a 3D position p_i, a normal $\vec{n}_i$, and an area a_i.

Surface-sample sets of this kind can be produced easily and robustly for many initial representations of S. If S is a *polygonal mesh* or an *unstructured triangle soup*, then a surface sample is produced for each polygon; beforehand, any triangle spanning multiple voxels is subdivided into as many subpolygons, so to obtain a good coverage with no assumption on triangle sizes. If S is a set of reciprocally aligned *range scans* (each range scan consisting in a grid of range points), then a unit area surface sample is produced for each range-point. If S comes in the form of an *unstructured point cloud*, we produce a surface sample for each point; normals, if unavailable by other means, can be estimated e.g. with [Mitra et al. 2004].

Optionally, a desired local area ratio r_i can be associated to each surface sample s_i (otherwise assumed to be 1, meaning uniform area preservation over all S); values of r_i can be set to mirror the relative importance attributed to the portion of S around s_i, thus allowing for *signal-specialized* maps (e.g. see Fig. 6).

Preliminary: determine resolution. The resolution of the volume-encoded UV-map is a parameter corresponding, as mentioned, to a trade-off between achievable map quality and memory usage. In our implementation, we use only powers of two, to maximize supportability by graphic cards, usually ranging between 2^2 and 2^6 (see Table 1). If the resolution is insufficient, it can prove impossible to avoid local loss of injectivity: unless that can be tolerated, the resolution must be increased and the construction repeated. The aspect ratios of the logical space is always selected to match as closely as possible the bounding box of S.

4.1 Construction of continuous UV-maps

We first cover the construction of volume-encoded UV-maps needing no cuts (Sec. 3.1). The objective is to assign a uv position to each 3D integer position of P being the corner of any **non-empty** voxels, i.e. the ones containing at least one surface sample. The set of these uv values constitutes the variables of our system.

Inspired by the success of this approach on standard UV-map construction [Lévy et al. 2002; Desbrun et al. 2002], we optimize for desired properties on the gradients of the solution. Given an assignment of variables, the gradients of f in u and v directions evaluated at position p, $\nabla u(p)$ and $\nabla v(p)$, expressed in *logical space*, are given by the gradients of τ (eq. 2) computed in $q = \sigma(p)$:

$$\nabla u(p) = \sum_{a,b\in\{0,1\}} \begin{pmatrix} \hat{q}_y^a\ \hat{q}_z^b\ (u_{1ab} - u_{0ab}) \\ \hat{q}_x^a\ \hat{q}_z^b\ (u_{a1b} - u_{a0b}) \\ \hat{q}_x^a\ \hat{q}_y^b\ (u_{ab1} - u_{ab0}) \end{pmatrix} \quad (5)$$

$$\nabla v(p) = \sum_{a,b\in\{0,1\}} \begin{pmatrix} \hat{q}_y^a\ \hat{q}_z^b\ (v_{1ab} - v_{0ab}) \\ \hat{q}_x^a\ \hat{q}_z^b\ (v_{a1b} - v_{a0b}) \\ \hat{q}_x^a\ \hat{q}_y^b\ (v_{ab1} - v_{ab0}) \end{pmatrix} \quad (6)$$

where symbols have the same meaning as in (2). Note that $\nabla u(p)$ and $\nabla v(p)$ are linear with the variables.

For f to be *orthogonal*, $\nabla u(p)$ and $\nabla v(p)$ must be on the tangent plane of S, i.e. they must be orthogonal to its local normal $\vec{n}$; for the map to be *conformal*, $\nabla u(p)$ and $\nabla v(p)$ must be equal in length and reciprocally orthogonal; for the map to also *preserve areas*, we need $||\nabla u|| \cdot ||\nabla v||$ to be equal to one (or, more generally, to the prescribed local area ratio r); to rule out local *fold-overs* in the map, we need to ensure the "handedness" of the frame $\nabla u(p), \nabla v(p), \vec{n}$ to be consistent, e.g. to be always right-handed.

All these conditions are satisfied exactly, at a surface sample s_i (with pos. p_i, normal $\vec{n}_i$ and targeted area ratio r_i), if and only if:

$$\vec{n}_i \times \nabla u(p_i) = \nabla v(p_i) \quad (7)$$

$$\nabla v(p_i) \times \vec{n}_i = \nabla u(p_i) \quad (8)$$

$$\nabla u(p_i) \times \nabla v(p_i) = r_i \cdot \vec{n}_i \quad (9)$$

($\times$ being the cross product). Therefore, we define our energy E_{TOT} as a measure of the fulfillment of the above vectorial equalities, area weighted and summed over all surface samples:

$$E_{TOT} = \sum_i a_i \cdot (E_i^{(7)} + E_i^{(8)} + E_i^{(9)}) \quad (10)$$

where $E_i^{(7)}$, $E_i^{(8)}$, $E_i^{(9)}$ are the squared residual of (7), (8), (9):

$$E_i^{(7)} = \| \vec{n}_i \times \nabla u(p_i) - \nabla v(p_i) \|^2$$

$$E_i^{(8)} = \| \nabla v(p_i) \times \vec{n}_i - \nabla u(p_i) \|^2$$

$$E_i^{(9)} = \| \nabla u(p_i) \times \nabla v(p_i) - r_i \cdot \vec{n}_i \|^2$$

The energy terms $a_i \cdot (E_i^{(7)} + E_i^{(8)})$ are quadratic, and can be seen as an equivalent, for the volumetric case, of the Least Squares Conformal Maps energy designed for common per-vertex UV-maps [Lévy et al. 2002]. Analogously to [Lévy et al. 2002], minimizing these terms conveniently amounts to solve a sparse Least Squares problem, which accounts for *conformality*, and tends to avoid *fold-overs*. In our formulation, this also optimizes for *orthogonality*, a characteristic which has no counterpart in per-vertex UV-maps (because, for them, U and V gradients are tangent to the surface by construction). In other words, our novel objective of *orthogonality* turns out to be as easily pursued as *conformality*.

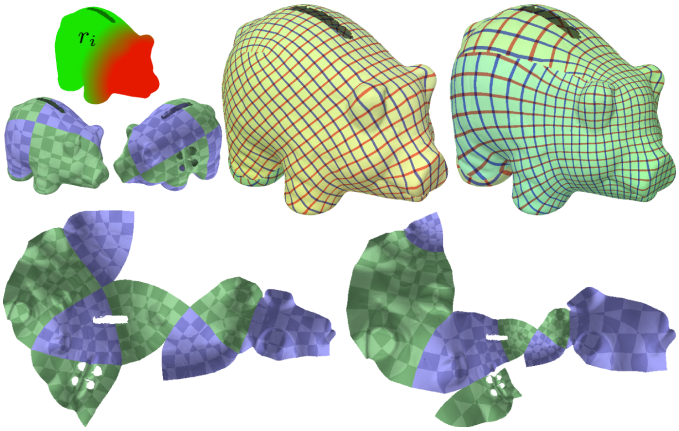

Figure 6: *Example of an adaptive volume-encoded UV-map. User-defined prescribed area ratio r_i is color-coded from red ($r_i = 1$) to green ($r_i = 9$). Left: a UV-map with constant area ratio r. Right: an adaptive UV-map (devoting more texture areas to the red regions). Above: uv lines. Below: uv space.*

Unfortunately, to seek *area preservation*, and thus *isometry*, we also need the terms $a_i \cdot E_i^{(9)}$, which are quartic, and specifically bi-quadratic.

System solution. Directly solving a large bi-quadratic multivariate systems is impractical, so we adopt a simple local-global heuristic, which consists in solving a succession of global sparse linear systems, until convergence. In the first system we minimize (10) disregarding term $E_i^{(9)}$, which as noted results in a quadratic Least Squares problem. The minimizer is determined up to a global 2D scaling, rotation, and translation of the variables, which is specified by imposing constant values to two uv positions.

In each subsequent system, we first compute a pair of local constant vectors $\vec{u}_i$ and $\vec{v}_i$ for each sample i, as the vectors $\nabla u(p_i)$ and $\nabla v(p_i)$, computed from the previous solution with (5) and (6), then projected on the plane orthogonal to $\vec{n}_i$, made reciprocally orthogonal (with polar decomposition), and re-scaled so to have length $\sqrt{r_i}$.

Then, in (10), we substitute $E_i^{(9)}$ with the average of two approximations of it, $E_i^{(9A)}$ and $E_i^{(9B)}$:

$$E_i^{(9A)} = \| \nabla u(p_i) \times \vec{v}_i - r_i \cdot \vec{n}_i \|^2$$

$$E_i^{(9B)} = \| \vec{u}_i \times \nabla v(p_i) - r_i \cdot \vec{n}_i \|^2$$

resulting in a new quadratic Least Squares system for the next iteration (with the same set of variables).

As a speedup, instead of solving each system in full, we perform, in all iterations except the last one, just a few steps of a simple gradient descent, starting from the previous solution. Another optimization consists in merging together samples sharing the same voxels, aggregating the associated values, to reduce the system size.

Finalizing the map. In the final phase, in order to further increase the *geometry independence* of the map, the uv values produced by the system as corners of non-empty voxels are propagated to all other integer locations of P, abiding to the *orthogonality* of f: first we assign an average normal to all non-empty voxels of L (as the area-weighted averaged normal of all samples inside that voxel); then we expand this normal over empty voxels, with a trivial diffusion algorithm (we zero the normals of empty voxels then we

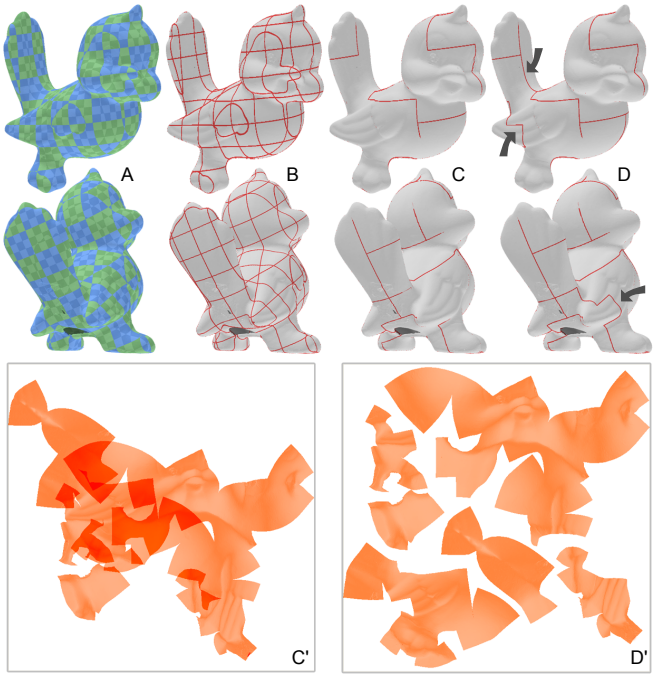

Figure 7: *Construction of an Atlas-based UV-map, for the 'birdie' model.* A: *voxels and zones (color coding as Fig. 4);* B: *consequent grid of potential cuts;* C: *the flood-filling algorithms sweeps the mesh, leaving only a few actual cuts, and producing a single self overlapping chart (*C'*);* D: *the final phase splits this chart into non-overlapping charts, reintroducing a few cuts (arrows), then packs them in the final atlas (*D'*).*

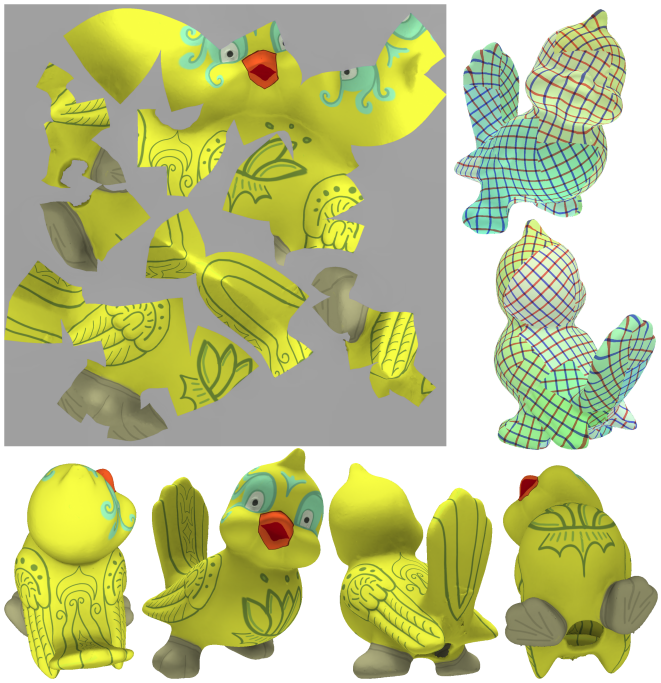

Figure 8: *Volume-encoded UV-map (birdie dataset), shown with an example artist-painted texture (top-left). Top-right: uv lines.*

iteratively assign to each empty voxel the averaged normal of all neighbors); finally we project the uv values stored at integer locations of P to the unassigned locations following this volumetric normal field. This might generate foldovers far away from S, where it is inconsequential.

The image of f must fit into texture space $\Omega = [0,1] \times [0,1]$. A global rescale and translation is therefore applied to all uv values stored in P , remapping the 2D bounding square encompassing all uv values into the unity square.

4.2 Construction of global UV-maps

For maps which embed cuts, a zone size k (Sec. 3.2) is selected as an extra parameter, usually ranging from 2 to 8. Just like resolution selection, this parameter represents trade-off between quality and occupancy; again, the construction can even fail (and needs be repeated) for inadequate choices. These cases are detected already in the first iterations.

Depending on the selection on k, we get a grid of *potential cut faces* (Fig. 7-B). Many of them are to be "deactivated", by enforcing four **equality constraints** between pairs of uv values stored in P at their corners (see Sec. 3.2); the rest are (actual) *cut faces*.

To pick a proper set of *equality constraints*, we adopt a flood-fill based heuristic; its objective is to leave enough cuts to "unfold" the mesh into a *single chart* (Fig. 7-C). Next, the global system is resolved, as in Sec. 4.1, with sets of constrained uv values represented by unique variables in the system. The resulting single chart often presents *global overlaps* (Fig. 7, C'). If so, in a following phase, it is divided in non-overlapping sub-charts (reintroducing previously deactivated cuts) and packed in Ω, thus obtaining an *Atlas based* volume-encoded UV-map (Fig. 7, D+D'). The rest of this section details every stage of this process.

Flood filling. We seed the procedure from an arbitrary non-empty voxel of L and process that voxel. To process a voxel v in L means to solve a small sub-system (as in Sec. 4.1), which has, as variables, only the uv values associated to the eight corners of P associated to v; this requires accounting for only the energy terms associated to the samples falling in voxels of L which are affected by any of these corners (up to 27 voxels around and including v, and sometimes fewer on the borders of zones). Any uv value which was already assigned in a previous step is considered a constant.

At every step, we pick a non-processed, non-empty voxel v_i adjacent to an already processed voxel v_j. If v_i and v_j belong to different *zones*, then the *potential cut-face* separating them is "deactivated", by prescribing four new equality constraints between all four pairs of corners in P corresponding to the traversed face. Then v_i is processed. The system is repeated until all non-empty voxels are reached (if S is composed of several disconnected components, this can require to re-seed the system multiple times).

After processing a voxel, each newly computed uv value at a position $q' \in P$ on the border of a *zone* is compared to any uv value already found at a location $q'' \in P$ such that q' and q'' correspond to the same location in L (there are from 0 to 7 such locations). If the difference is smaller than a given threshold (we used value 0.2), a new equality constraint is added between the uv values at q' and q''. At the end of the process, we discard superfluous *equality constraints*, i.e. ones not contributing to the "deactivation" of at least one *potential cut face*. To achieve better results, every 10 steps we perform a global optimization for all uv values determined so far.

Cuts placement. The ordering of traversal of the flood-filling procedure affects the choice of cuts. The traversed voxel-faces are always "deactivated", so we prioritize traversal of voxel-faces where we prefer cuts not to appear.

Any face which is not a *potential cut face* is always prioritized over ones which are, because we are unable to create a cut there. Other than that, we prefer smooth areas of S to be free from cuts. Also, we want the cut-position to remain close to constant, in uv space, for small displacements from S. Consequently, f must be close to constant for displacements over a *cut face* γ_L which are orthogonal to the cut-line $\gamma_L \cap S$. Since we also want f to be *orthogonal* to S, this means that we favor cut faces which are orthogonal to S in L.

We pick the face with the smallest "*cut-value*": a voxel face a, with normal $\vec{n}_a$, and separating two voxels v_0 and v_1 having average normal $\vec{n}_0$ and $\vec{n}_1$, (with $\vec{n}_{(0+1)}$ being the average of these two normals), has a *cut-value* of:

$$|(\vec{n}_{(0+1)} \cdot \vec{n}_a)| - (\vec{n}_0 \cdot \vec{n}_1) \tag{11}$$

The first term favors cuts in parts of S which are orthogonal, in logical space, to the cut-face. The second term discourages cuts over smooth parts of S. Other user defined requisites for cut placement, e.g. concealment, could be easily embedded in (11).

Overlaps removal and packing. Possible global overlaps in Ω are dealt with by partitioning the single chart produced so far into a set of non-intersecting **charts** $c_1 \dots c_n$. Charts are separated by means of re-activation of a few deactivated *potential cuts face*, as follows.

First, non empty voxels of L are grouped into **chunks** each being a set of adjacent non-empty voxels belonging to a single *zone* of L (two voxels are adjacent if they share a corner; multiple chunks can coexist inside a *zone* if they share no corner). A graph is constructed where each node is one *chunk*. An arch is added between any two neighboring chunks separated by at least one deactivated potential cut-face, and its strength value is assigned as the summed *cut-values* (11) of all such cut-faces. Each connected component of this graph represents a *chart*. Initially the graph is connected and there is only one chart.

If any single chunk self-overlaps in Ω, nothing can be done to prevent self-intersections, short of changing parameter k or resolution of the volume-encoded UV-map. Otherwise, we iteratively detect a pair of overlapping chunks belonging to the same chart. We perform a graph-cut to separate them, by removing a set of arches with the maximal cumulative strength (any method can be employed to address this sub-problem, we used [Ford and Fulkerson 1962]). The process ends when no more intra-chart overlaps are detected.

In the last phase, the separated charts are packed into Ω (see Fig. 7). It is an analogous procedure to the one carried out with conventional *Atlas-based* per-vertex UV-maps, e.g. [Zhou et al. 2004]. We are free to employ any packing heuristic, because each chart c_i can consistently undergo arbitrary roto-translations β_i (or, indeed, any $2D$ affine transformation): it is sufficient to apply β_i to all uv values associated to a chart i. This is sound, because function τ (equation (2)) is linear with the uv values stored in P.

5 Using volume-encoded UV-map

5.1 Filling the final texture

A wealth of techniques and authoring tools exist to support the creation of a 2D texture T for a standard, per-vertex UV-mapped model. For example, T can be produced by manual painting (either directly over T, or over S in 3D, storing paint strokes in T), by resampling registered pictures (if S models a real world artifact), by sampling details from an higher resolution model, by baking (e.g. global) lighting, by resampling an existing texture, by texture synthesis, etc (see Figures 14, 13, and others).

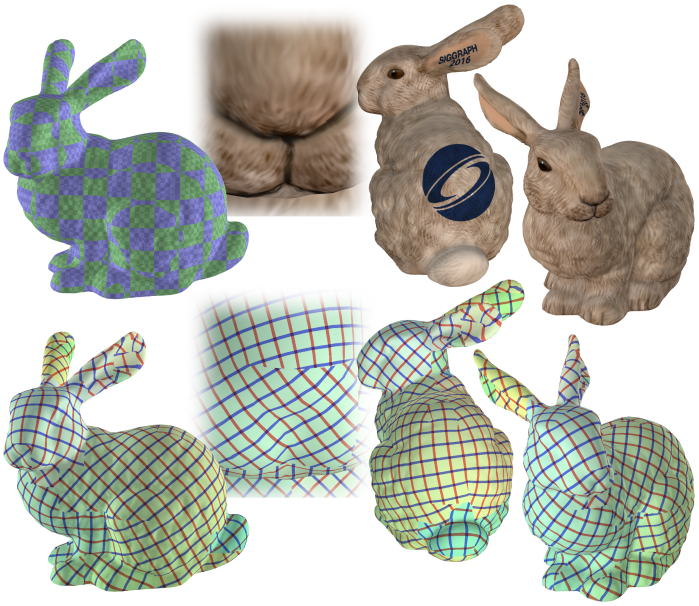

Figure 9: *Volume-encoded UV-mapped Bunny model. Top-left: divisions of S in voxels/blocks. Bottom: uv iso-lines, shown on a closeup around a texture seam (in the nose region), and on two additional views. Top: textured results (T shown in Fig. 2).*

To allow direct use of any of these tools, a volume-encoded UV-map can be temporarily converted into a per-vertex UV-map, only for the purpose of building the texture, as follows. First, each triangle spanning more than one logical voxel is split into one sub-polygon per voxel. Then, the uv attribute $f(\dot{p}_v)$ is assigned to each vertex with position $\dot{p}_v$ (taking care to resolve f correctly at seams). Even if they match at all vertex positions, the two maps are still not strictly equal, as the uv values of the volume-encoded one are tri-linearly interpolated in 3D space, whereas the values in the per-vertex one are linearly interpolated in the triangle plane. The difference can be made arbitrarily small, by refining the mesh before per vertex uv assignment. The T built for the per-vertex UV-mapped model can be used by the volume-encoded UV-mapped model.

5.2 Accessing the final texture

The many widely used mechanisms of standard UV-maps are directly applicable with our new representation.

Bilinear interpolation. Importantly, a volume-encoded UV-map allows for native bilinear interpolation in the final access to the texture T, just as much as a standard UV-map schema featuring cuts. As common, texels replication around cuts is needed in T to avoid *color bleeding* artifacts: texels on one side of a cut must be replicated next to the other side.

MIP-mapping. isotropic MIP-mapping is also trivially supported. This requires, as usual, replication of texels values at cuts in all MIP-map levels. The only difference with standard UV-maps is that the MIP-map level for a fragment at world position p must be computed with the screen space derivatives of its logical position $\sigma(p) \in L$, instead of its uv position $f(p)$. Instead, anisotropic MIP-mapping requires tangent directions (see next page) in order to compute the screen space rate of change of u and v.

Replicated textures. Even if our construction targets injective maps, our representation can freely express UV-maps where one part of T is mapped over several parts of the model (a practice common in games to save memory e.g. exploiting symmetries).

Tangent-Space Normal-Maps. Our representation is compatible with Normal-Maps stored in Tangent space, because tangent directions are defined everywhere in the volume (and therefore on the surface) and can be recovered quickly. In the traditional case, (approximated) tangent and "bi-tangent" directions are precomputed and stored at vertices, and then interpolated inside faces; in our case, we precomputed them at 3D texels and store them in volumetric texture sized like P, then trilinearly interpolate them at rendering time. This is analogous to the case of uv positions, leading to a similar, potentially convenient, spatial tradeoff (see Table. 1, second-last column). In our setup, approximated 3D tangent directions pre-computation is particularly straightforward: each coordinate is found by finite difference of u values (v for bi-tangent) of the two 2 neighboring elements in the respective direction. See Fig. 15. Technical details are found in the Appendix (additional materials).

Animated meshes. Although the uv values are accessed using the 3D object positions, our schema can be directly applied to common animated models, e.g. Blend-Shapes or skinned models with Skeletal Animations (see Fig. 16), simply feeding to f the rest position (which is needed anyway by the shader).

6 Results

We tested volume-encoded UV-maps on several datasets, shown in the figures of this papers, plus more in the demo. To allow a visual assessment the quality of the maps, we show uv isolines over every model, in the images through the paper. All the shown mappings are fully injective, with the noted exception.

Table 1 summarizes numeric measurements of the maps. We report achieved distortions with two measures: the L^2 Stretch energy [Sander et al. 2001] of the restriced map, which reflect area and angle distortion, and our own energy (10), which, as discussed, additionally reflects orthogonality. To compute the former, we first converted the maps to a per-vertex representation, after adequately refining the triangle meshing (see 5.1). Stretch energy, as well as visual uv isolines inspection, confirms that the quality of the maps is fully adequate for texturing applications.

Our construction heuristic is reasonably fast, usually taking within a minute of computation time on commodity hardware.

To foster further research and direct applications, we release a demo (in the additional materials), which allows inspection of the volume-encoded UV-mapped models shown in this paper and others. It illustrates the case of an end-application. Full shader programs are revealed by the touch of a button.

Space efficiency comparisons. in addition to the other qualitative (and more significant) considerations, our representation compares favorably as for space occupancy with Perfect Spatial Hashing [Lefebvre and Hoppe 2006] and TileTrees [Lefebvre and Dachsbacher 2007] (which in turn are more compact than [Benson and Davis 2002; Lefebvre et al. 2005]). To store 562K distinct rgb samples for the Armadillo dataset, [Lefebvre and Hoppe 2006] stores 83^3 blocks of 3^3 rgb values, plus a 91 KB offset table, for a total of 45.3 MBytes. For the same example and the same number of distinct samples, [Lefebvre and Dachsbacher 2007] reports a total occupancy of 11.3 MBytes. With a $128\times128\times64$ P (two 16 bits channels, 4 MBytes) and a 1024^2 T (one rgb per texel, 3 MBytes) with 60.4% packing ratio, we are able to store more, and much better distributed, distinct rgb samples for a significantly smaller total memory cost: 7 MBytes. Ours scales better too: increasing the number of samples affects only T. Note that, for many other datasets, the cost for our representation can be drastically smaller than for this case (see table 1).

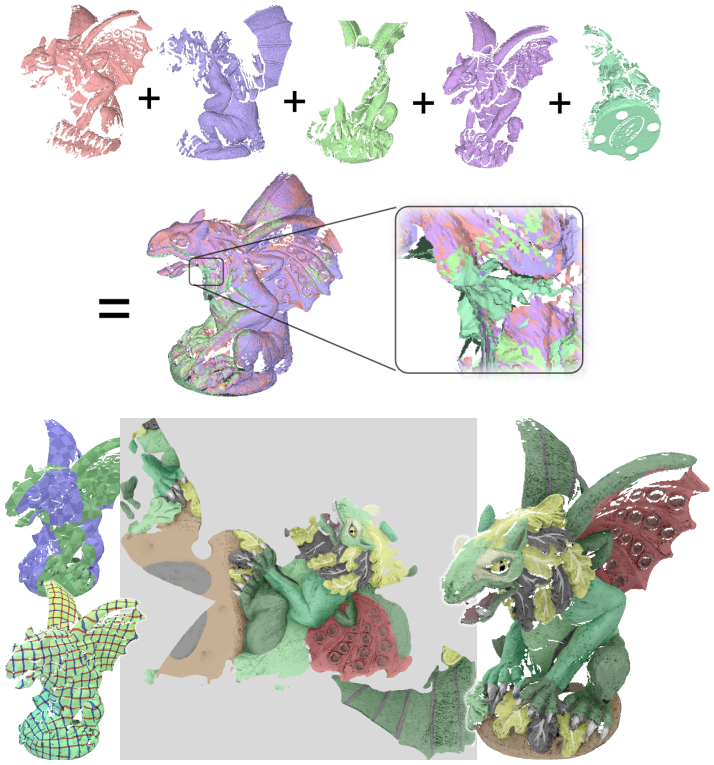

Figure 10: *A volume-encoded UV-mapped set of 5 registered range-images (top). Despite the typical inconsistencies of this kind of data-sets, like severe incompleteness, noise, overlaps, and lack of a consistent global connectivity (see inset, middle), a volume-encoded UV-map could be both easily constructed and directly used to apply an artist painted texture (bottom). Bottom-left: spatial partition in voxels and zones, and uv isolines.*

7 Discussion

Compared to the standard, per-vertex alternative, volume-encoded UV-maps can impact existing asset creation and usage pipelines in several ways.

End-user perspective. The most important advantage for the final application is the gained ability to reuse both the UV-map and the texture among different representations of the same object (e.g. LoD levels of a game), in stark contrast with the common situation where *neither* can be shared. Also, 3D meshes are freed from the need of vertex-duplications at seams. Lastly, the overall memory consumption can be (but is not always) strongly reduced, depending on the primitive count and the shape complexity of the model. The price, in terms of computation overhead, is very small, amounting to one single, cache-coherent extra indirection per fragment (in stark contrast with alternatives previously presented in literature). For the rest, an volume-encoded UV-map can be used exactly a standard one (see Sec. 5.2).

UV-mapper perspective. In the industry, the task of producing good UV-maps, e.g. by artists, is often computer-assisted rather than fully automatic. While single-chart "unfolding" (distortion minimization) is delegated to automatic algorithms, two other subtasks, cut-line identification and final packing of charts, usually require some user intervention, despite the research in these areas; partly, this is because the artists require control to exploit potential symmetries, place cuts according to semantics, and for other specific goals. This situation is substantially replicated with our new

Figure 11: *Left: the armadillo dataset (textured model, UV-lines, and 2D texture). Right: a visual comparison of 2D texture layouts with two similarly purposed representations: PolycubeMaps [Tarini et al. 2004] and TileTrees [Lefebvre and Dachsbacher 2007] (images courtesy of the respective authors). In spite of being much cheaper to decode, our representation is capable of expressing quality maps whose overall characteristics (in terms of cuts and distortions) resembles the per-vertex UV-maps commonly used in the industry.*

Table 1: *Data on the example of volume-encoded UV-maps shown through this paper. For each dataset, we report: the used 3D resolution of the UV-map and block size k (see Sec. 3.2); the quality of the map, as our energy, eq. (10), (which accounts for both* isometry *and* orthogonality*), and, where applicable, the L^2 Stretch energy [Sander et al. 2001] of the restricted mapping (which accounts for* isometry *only); the percentage of unique final texel samples used on the total in 2D texture; whether or not the result is free from overlaps; and the total size required to store the map in GPU memory. To help put the latter in context, in the last two columns it is presented as: the percentage of the memory occupancy that would be required by any conventional per-vertex representation of the UV-map for the same model (disregarding the vertex duplications which would be required); and, the overhead with respect to a standard hi-res 2D texture T of* 2048×2048.

Dataset			**Volume-encoded UV-map**								
			Parameters		*Quality measures*				*Memory occupancy*		
Name	*Size*	*see Figure*	physical res	zone size	our energy (0 = best)	Stretch eng. (1 = best)	used 2D texels	fully inj.?	RAM (bytes)	as % of per-vertex	as % of 2D texture
beetle	34K △	3,left	$2\times2\times4$	-	0.784	1.536	70.1%	yes	64	0.09%	$< 0.01\%$
		3,center	$4\times4\times8$	-	0.186	1.115	61.0%	yes	512	0.71%	$< 0.01\%$
		3,right	$8\times8\times16$	-	0.108	1.068	52.1%	yes	4K	5.72%	$< 0.1\%$
sphere	1280 △	5, 13	$8\times8\times8$	4	0.055	1.032	59.5%	yes	2K	79.8%	$< 0.1\%$
gargoyle	258K pts	10	$16\times16\times16$	7	0.346	—	38.9%	yes	16K	1.5%	$< 0.1\%$
piglet	74K △	6,left	$16\times16\times16$	7	0.134	1.110	41.9%	yes	16K	17.3%	$< 0.1\%$
		6,right	$16\times16\times16$	7	0.157	2.206†	45.4%	yes	16K	17.3%	$< 0.1\%$
birdie	97.7K △	7, 8, 13	$32\times32\times32$	3	0.107	1.134	50.8%	yes	128K	67.0%	$< 1\%$
bimba	100K △	(demo)	$32\times32\times32$	3	0.146	1.280	61.9 %	yes	128K	32.8%	$< 1\%$
warrior	500K pts	12	$32\times64\times32$	4	0.193	—	50.9%	yes	256K	13.1%	2%
bunny	69.4K △	1, 9, 15	$64\times64\times64$	5	0.061	1.090	53.1%	yes	1M	753%	6%
fertility	100K △	14	$128\times64\times32$	2	0.050	1.033	56.9%	yes	256K	65.5%	6%
orc	30K △	16	$64\times64\times16$	6	0.050	1.028	52.4%	yes	256K	65.0%	2%
armadillo	330K △	11	$128\times128\times64$	3	0.119	1.106	63.0%	yes	4M	638%	25%
dragon	871K △	17	$64\times64\times32$	2	0.239	1.614‡	62.0%	no	512K	30%	3%

† Note that L2 Stretch energy ignores the user-specified target area ratios. ‡ Computed over non folded faces only.

structure: in Sec. 4.1 we provided an automatic single-chart distortions minimization procedure, fit to replace its counterparts for standard maps, and in Sec. 4.2 we showed that heuristics addressing the two mentioned sub-tasks have their close equivalent in our schema (the one we propose represent one possible solution, but manually designed volume-encoded UV-maps would probably feature superior quality). The only additional task that we require is the selection of resolution and patch size selection, which is easy due to the small space of choices. In addition, the adoption of our scheme unlocks new possibilities. The map can be constructed over surface representations other than meshes, thanks to its volumetric nature – for example, in a range-scanning pipeline, UV-mapping can precede, rather than follow, mesh fusion; even for a standard mesh, the construction is more robust, not requiring a clean, two-manifold input, and is intrinsically easier, because fewer variables are typically involved; the result is directly reusable when the mesh tessellation changes – this can be crucial in the movie and game industry, where, traditionally, hi-res and low-poly versions of a model have to be UV-mapped separately.

Texture artist perceptive. Any existing technique currently used to author a texture for a given model can be directly adopted, thanks to the ability to temporarily convert our maps (see Sec. 5.1). As a matter of fact, the textures shown in this work and in the attached demo are authored by five texture artists who directly employed existing texture authoring tools, covering each one of standard approaches listed in Sec 5.1. One key factor here is that our representation can express maps which are qualitatively similar to the traditional ones, in terms of cut density and distortions (much more so than alternatives, as illustrated in Fig. 11).

Conclusions. In all cases where it is applicable, adoption of the proposed representation unlocks crucial, unprecedented advantages over the per-vertex alternative which is currently ubiquitous, e.g., in games, so we consider it to have a big potential impact.

7.1 Limitations

Generality. The biggest limitation concerns generality. When S presents tiny geometric features, then a fully injective volume-encoded UV-map can require too expensive a volumetric resolution; otherwise, construction can fail, for either one of two reasons: too complex details falling in one voxel, or insufficient *potential cuts* to cut open S. An example of a failure case is shown in Fig 17. In other words, our approach is directly usable over a class of models only. The class is wide, as indicated by our experiments on real world data (see Table 1), and includes numerous non-trivial cases of practical utility. One reason is that trilinear interpolation of grid values turns out to be surprisingly expressive: consider for example how the entire model in Fig. 3, can be accommodated in as few as $1 \times 1 \times 3$ voxels, still featuring acceptable distortions and zero overlaps, notwithstanding that the model features elements facing opposite directions inside a voxel. Yet, there are no guarantees.

Inherited issues. Our new representation completely removes a few of the long standing hindrances of the conventional, and ubiquitously used, per-vertex maps, but leaves others unaffected, which are, instead, specifically targeted by alternative routes. The main ones are: first, that the map construction is still, overall, a laborious task, involving choices which are difficult to be completely automated (this is bypassed, for example, by approaches like [Christensen and Batali 2004; Lefebvre and Dachsbacher 2007; Yuksel et al. 2010; Burley and Lacewell 2008]); moreover, the parametrization is global so the task is to be repeated if the model undergoes substantial changes (in contrast to the quick local refreshes possible, for example, in [García et al. 2011]). Second, that cuts may be visible on close-ups, due to inconsistent filtering on the two sides

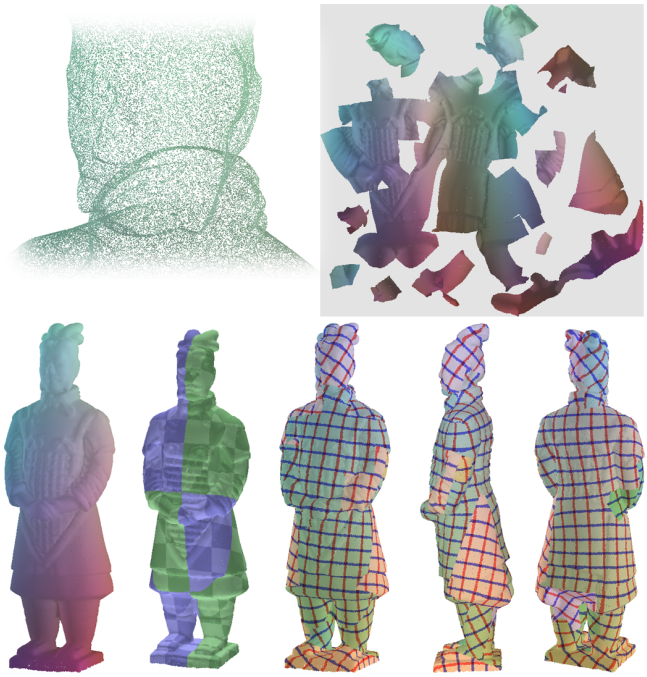

Figure 12: *A volume-encoded UV-mapped dense point cloud. Top right: a close-up of the head, revealing the nature of the dataset. Renderings are done by means of simple point splatting.*

(if barely: e.g. see Fig. 9); this issue is addressed, or bypassed, in approaches like [Ray et al. 2010; Purnomo et al. 2004; Tarini et al. 2004; Burley and Lacewell 2008; Tarini 2012]; note that the common countermeasure consisting in placing the cuts in less visible parts of S can, in principle, be applied to our representation too. Third, the atlas packing necessarily leaves some unused texture space (unlike, e.g., [Lefebvre and Hoppe 2006; Yuksel et al. 2010; Burley and Lacewell 2008; Tarini 2012; Tarini et al. 2004]).

7.2 Future work

About map construction. Alternative algorithms (and interactive tools) to construct volume-encoded UV-maps can be designed, also seeking new objectives. For example our representation is potentially capable of expressing *grid-preserving* UV-maps, which, as shown in [Ray et al. 2010], can be useful to conceal cuts, especially with low resolution 2D textures; in order to construct a grid-preserving volume-encoded UV-map, additional constraints need be imposed between mismatching uv positions stored at *cut-faces*. Another approach is the "conversion" of a existing per-vertex UV-map into a volume-encoded representation: the input map could be used to guide the construction of a new volume-encoded one.

About map representation. A natural candidate to extend the generality is to resort to some hierarchical volumetric structure, but this would defy the central aspect of our solution, crucial for its practical usability: the negligible rendering-time overhead, costing a single indirection. Instead, a hybrid approach could be studied in the future, where the 3D "logical" position $p' \in L$ is stored as a vertex attribute (chosen independently from its object-space coordinates p) and, per-fragment, the interpolated value of p' is fed to function f instead of p. Values of p' would have to be designed, in preprocessing, so to morph S and "inflate" thin features. The resulting schema would have characteristics in between per-vertex and volume-encoded UV-maps.

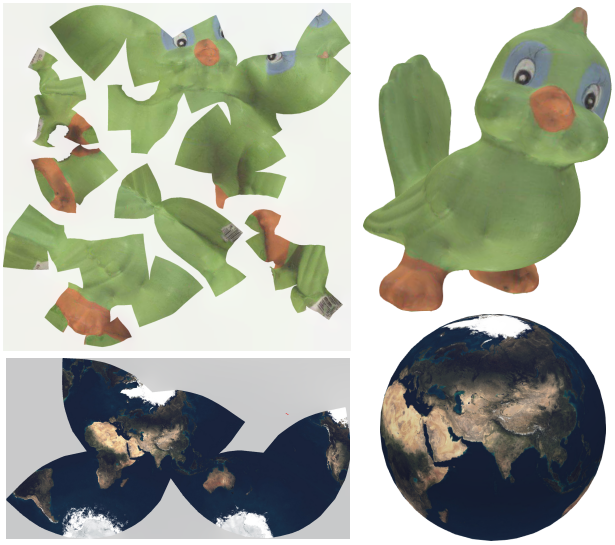

Figure 13: *Some color textures filled for volume-encoded UV-mapped models, using standard texture filling tools: resampling from projected calibrated pictures (birdie model, top), and resampling from an existing texture (sphere model, below).*

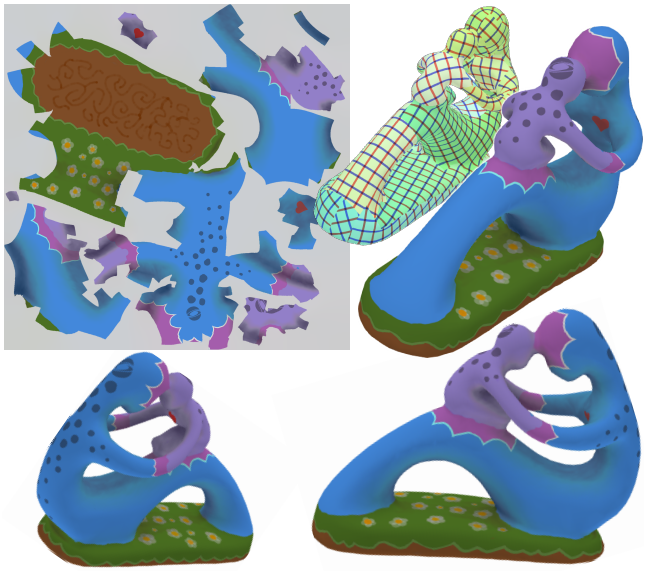

Figure 14: *Volume-encoded UV-mapped Fertility model, with an artist painted texture (top-left).*

Acknowledgments

We are grateful to artists who authored many of the 2D textures used as examples: Marco Callieri (Birdie), Gaia Pavoni (Dragon, all bumpmaps), Romain Rouffet (Armadillo, Logo Bunny), Urban Schrott (Orc). We thank the anonymous reviewers.

References

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Trans. Graph. 21*, 3, 785–790.

BURLEY, B., AND LACEWELL, D. 2008. Ptex: Per-face texture mapping for production rendering. In *Eurographics Symp. on Rendering*, EGSR'08, 1155–1164.

CHRISTENSEN, P. H., AND BATALI, D. 2004. An irradiance atlas for global illumination in complex production scenes. In *Proc. of EG Conf. on Rendering Techniques*, EGSR'04, 133–141.

CHUANG, M., LUO, L., BROWN, B. J., RUSINKIEWICZ, S., AND KAZHDAN, M. 2009. Estimating the Laplace-Beltrami operator by restricting 3D functions. *Symp. on Geom. Proc.*, 1475–1484.

CIGNONI, P., RANZUGLIA, G., CALLIERI, M., CORSINI, M., GANOVELLI, F., PIETRONI, N., AND TARINI, M., 2011. Meshlab. http://www.meshlab.org/.

DACHSBACHER, C., AND LEFEBVRE, S. 2008. *Efficient and Practical TileTrees (in Shader X6)*. Shader X6. Charles River Media.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Comput. Graph. Forum 21*, 3, 209–218.

FORD, L., AND FULKERSON, D. 1962. *Flows in networks*. Princeton U. Press.

GARCÍA, I., LEFEBVRE, S., HORNUS, S., AND LASRAM, A. 2011. Coherent parallel hashing. *ACM Trans. Graph. 30*, 6.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21 (3)* (21/07/2002), 355–361.

HE, Y., WANG, H., FU, C., AND QIN, H. 2009. A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics 33*, 3, 369–380.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. *SIGGRAPH Course Notes*.

JAKOB, W., TARINI, M., PANOZZO, D., AND SORKINE-HORNUNG, O. 2015. Instant field-aligned meshes. *ACM Trans. Graph. 34*, 6.

LEFEBVRE, S., AND DACHSBACHER, C. 2007. Tiletrees. In *Proc. of the Symp. on Interact. 3D Graph. and Games*, ACM, 25–31.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. In *ACM Trans. Graph.*, vol. 25, ACM, 579–588.

LEFEBVRE, S., HORNUS, S., NEYRET, F., ET AL. 2005. Octree textures on the gpu. *GPU gems 2*, 595–613.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3, 362–371.

MITRA, N. J., NGUYEN, A., AND GUIBAS, L. 2004. Estimating surface normals in noisy point cloud data. In *spec. issue of Inter. Jour. of Comp. Geom. and Appl.*, vol. 14, 261–276.

PANETTA, J., KAZHDAN, M., AND ZORIN, D. 2012. Volumetric basis reduction for global seamless parameterization of meshes. Tech. rep., New York University.

PIETRONI, N., TARINI, M., AND CIGNONI, P. 2010. Almost isometric mesh parameterization through abstract domains. *IEEE Trans. on Vis. and Comp. Graph. 16*, 4, 621–635.

PIETRONI, N., TARINI, M., SORKINE, O., AND ZORIN, D. 2011. Global parametrization of range image sets. *ACM Trans. Graph.*, 149:1–149:10.

PURNOMO, B., COHEN, J. D., AND KUMAR, S. 2004. Seamless texture atlases. In *Proc. of Symp. on Geom. Proc.*, ACM, 65–74.

RAY, N., NIVOLIERS, V., LEFEBVRE, S., AND LEVY, B. 2010. Invisible Seams. *Comput. Graph. Forum*.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proc. of SIGGRAPH '01*, ACM, New York, NY, USA, 409–416.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *Proc. of Symp. on Geom. Proc.*, SGP '03, 146–155.

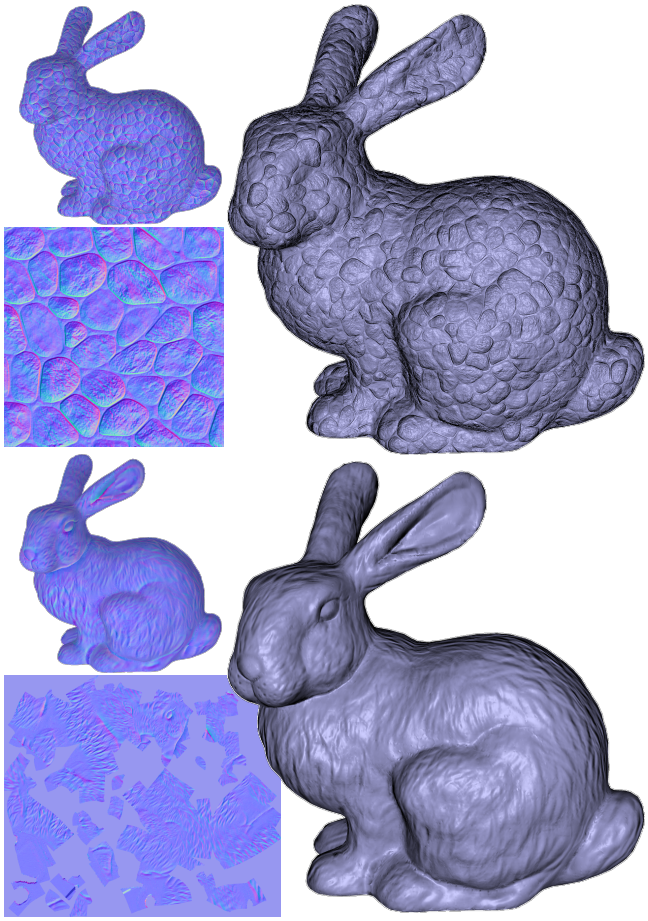

Figure 15: *Because we are able to quickly recover 3D tangent directions at rendering time (see Additional Materials), volume-encoded UV-mapped models can be used with Tangent Space normal-maps, which are the commonest kind in e.g. games. Left: the normal-maps. Right: the dynamically relighted results. Top: a tileable example, repeated* 8×8 *times (the pattern breaks at cuts). Bottom: a non-tiled example, sculpted over the model by an artist.*

SHEFFER, A., PRAUN, E., AND ROSE, K. 2006. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision 2*, 2, 105–171.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. *ACM Trans. Graph. 23*, 853–860.

TARINI, M., PUPPO, E., PANOZZO, D., PIETRONI, N., AND CIGNONI, P. 2011. Simple quad domains for field aligned mesh parametrization. *ACM Trans. Graph. 30*, 6.

TARINI, M. 2012. Cylindrical and toroidal parameterizations without vertex seams. *Journal of Graphics Tools 16*, 3, 144–150.

XIA, J., GARCIA, I., HE, Y., XIN, S., AND PATOW, G. 2011. Editable polycube map for gpu-based subdivision surfaces. In *Symp. on interactive 3D graphics and games*, ACM, 151–158.

YUKSEL, C., KEYSER, J., AND HOUSE, D. H. 2010. Mesh colors. *ACM Trans. Grap. 29*, 2, 15:1–15:11.

ZHANG, L., LIU, L., GOTSMAN, C., AND HUANG, H. 2010. Mesh reconstruction by meshless denoising and parameterization. *Computers & Graphics 34*, 3, 198–208.

ZHOU, K., SYNDER, J., GUO, B., AND SHUM, H.-Y. 2004. Isocharts: stretch-driven mesh parameterization using spectral analysis. In *Proc. of Symp. on Geom. Proc.*, ACM, SGP'04, 45–54.

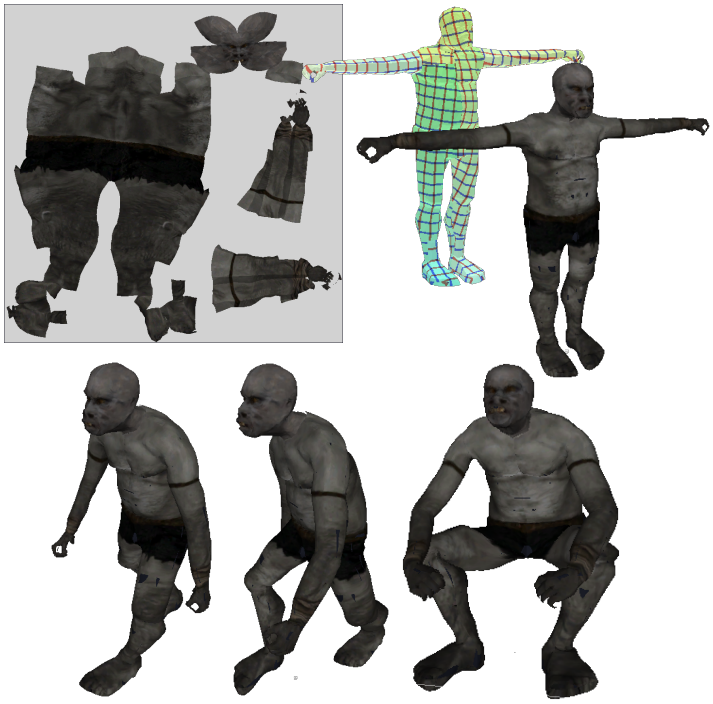

Figure 16: *A volume-encoded UV-mapped animated model with linear blend skinning. Top-left: its artist-painted texture. The volumetric function* $(u, v) = f(p)$ *is evaluated on the rest-pose position, before the skinning transformation is computed in the GPU programs.*

Figure 17: *A failure case. This "dragon" model is not a good candidate for our UV-map representation: the thin parts, e.g. in the mouth, cannot be parametrized without loss of injectivity. The resulting UV-map has four cases of partially self-overlapping charts, producing local artifacts when the artist-painted texture is mapped on the mesh (e.g. see inset, lower right—and the attached demo).*