DOI: 10.1111/cgf.14735

COMPUTER GRAPHICS *forum*
Volume 42 (2023), number 6, e14735

Texture Inpainting for Photogrammetric Models

A. Maggiordomo,[1] P. Cignoni[2] and M. Tarini[1]

[1]University of Milan, Milan, Italy
mggndr89@gmail.com, marco.tarini@unimi.it
[2]ISTI – CNR, Pisa, Italy
paolo.cignoni@isti.cnr.it

Abstract
We devise a technique designed to remove the texturing artefacts that are typical of 3D models representing real-world objects, acquired by photogrammetric techniques. Our technique leverages the recent advancements in inpainting of natural colour images, adapting them to the specific context. A neural network, modified and trained for our purposes, replaces the texture areas containing the defects, substituting them with new plausible patches of texels, reconstructed from the surrounding surface texture. We train and apply the network model on locally reparametrized texture patches, so to provide an input that simplifies the learning process, because it avoids any texture seams, unused texture areas, background, depth jumps and so on. We automatically extract appropriate training data from real-world datasets. We show two applications of the resulting method: one, as a fully automatic tool, addressing all problems that can be detected by analysing the UV-map of the input model; and another, as an interactive semi-automatic tool, presented to the user as a 3D 'fixing' brush that has the effect of removing artefacts from any zone the users paints on. We demonstrate our method on a variety of real-world inputs and provide a reference usable implementation.

Keywords: rendering, texture mapping, modelling, surface parameterization, texture synthesis

CCS Concepts: • Computing methodologies → Computer graphics; Texturing

1. Introduction

Photogrammetry is now a primary tool for producing high-quality, realistic 3D digital models, in the form of textured meshes. This technology has been enabled by a plethora of open-source [GGC*21, MMPM16] and commercial [rs22, Agi22] 3D photo-reconstruction software tools that are able to produce 3D models of high-visual quality from a combination of high-resolution geometry and extremely high-resolution texture data encoding material and colour information at a micro-scale level. This finds applications in a broad variety of fields such as movies asset production, videogames, cultural heritage, virtual reality and others.

Although models reconstructed from photographic data are constantly improving in quality, they also present recurring modelling artefacts [MPCT20]. A particularly frequent case is local texturing defects, *i.e*. problems in the texture image associated with the 3D mesh, which originate from a variety of issues and challenges in the acquisition process (see Section 3.2).

In this paper, we present a technique to correct these texture inconsistencies *a posteriori*, *i.e*. after the model acquisition has been completed. By working on the reconstructed models, our method is oblivious to the acquisition process and can be used to counter a wide range of potential problems with texture images, regardless of their origin. The basic idea is that the defective sub-areas of the texture image are discarded and *inpainted* anew by seamlessly filling the gap, guided by the surrounding context. While the inpainted regions may drift from the real-world object represented by the model (which we do not want to assume to be still available), we want it to look plausible and realistic. Inpainting methods are attractive because the vast majority of the texture data is valid and contains reliable information that can be used to interpolate the texture content and cover localized artefacts.

We are inspired in this by the recent advances of modern image inpainting frameworks, where a similar operation is successfully performed on natural images (photographs) by leveraging Machine Learning tools. However, it is not trivial to extend these results to the case of diffuse colour textures of 3D models. At a first glance, this can be attempted using either of two strategies.

Inpainting in texture space. A simple way is to feed the texture image directly to the same inpainting procedure used for

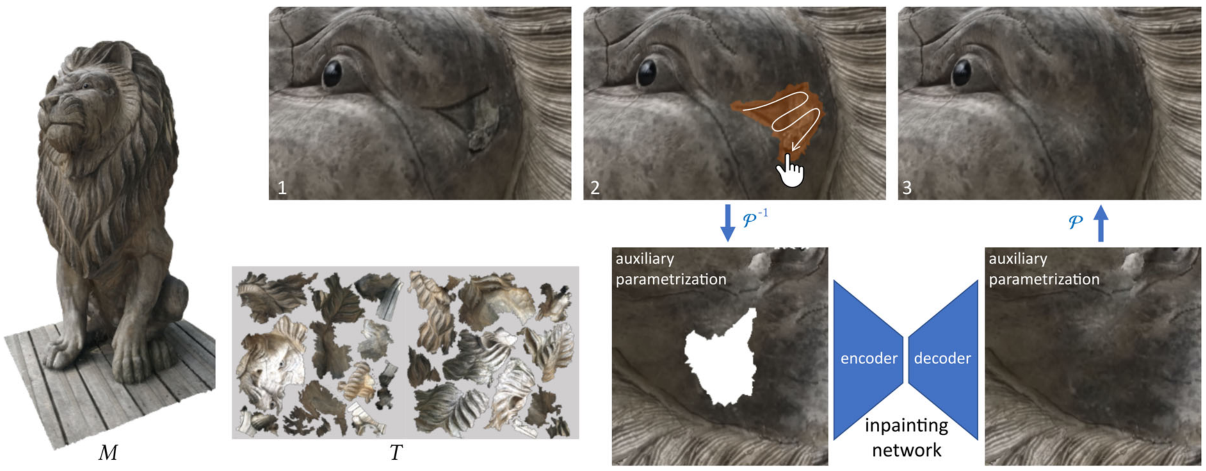

Figure 1: *Example of our inpainting operation performed in screen space, over a 3D rendering, introducing artefacts. See also attached video (timestamp: 6 min 38 s).*

photographs. However, this does not work reliably because of the semantics of texture image, which, in the context of photogrammetric models, invariably comes in the form of an atlas of texture islands (or charts), separated by gaps of unused space. Being unaware of the atlas discontinuities and texture distortions, any inpainting method applied to the global texture would produce visible seam artefacts. This is a critical issue, because photogrammetric models present heavily fragmented UV-maps [MPCT20], with intricate and dense texture seams.

Inpainting in image space. Another alternative that comes to mind is to apply image inpainting to the renderings, covering the artefacts that originate from the defective texture. The inpainted regions can then be re-baked into the texture by re-projection. This avoids the problem with texture seams and textures gaps, and is attractive because the texture inpainting procedure is fed synthetic images that are similar to the natural images that they are intended (and trained) for. Unfortunately, the inpainting method is unaware of the depth-jumps, and of the distinction between foreground and background, which needlessly complicates the semantics of the signal to be reconstructed. For example, the surrounding regions that happen to be close in image space can and will bleed into the inpainting area (see Figure 1). Moreover, screen-space inpainting is not robust to perspective distortion (see Figure 2). These problems are particularly severe in areas with high surface curvature or non-trivial topologies, where view projections are bound to come with occlusions.

1.1. Our solution

Our solution is to apply inpainting to an *ad hoc* parametric domain that is defined on-the-fly for each inpainting operation. Given a region of the mesh M that presents a texturing defect, we create a local 'auxiliary' parametrization of the affected faces and a region around them. We then synthesize a disposable texture for this region, inpaint over the defected region and transfer inpainted texels back into the original texture (possibly, across seams).

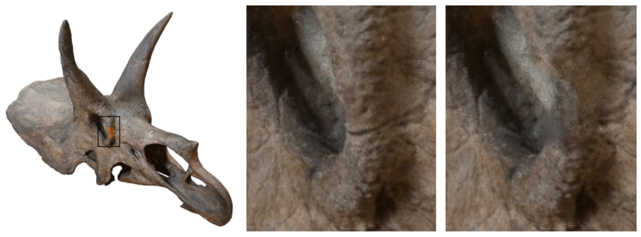

Figure 2: *Comparison of screen space and our local texture-space inpainting using multiple views. Changing the observer's position reveals stretching artefacts near the object silhouette induced by the narrow viewing angle and the resulting perspective distortion affecting the projection of the inpainted screen-space pixels over the object surface.*

This approach avoids all pitfalls from either the strategies above (depth-jumps, background, gaps in texture space, perspective distortions, texture seams); we can reach much lower texture distortions thanks to the localized nature of the parametrization. In short, this strategy presents to the inpainting algorithm a texture image with almost no distortion (either of the UV-map or view projection): every part of the texture to be inpainted, and its context that is used to guide the inpainting, appears as if seen orthogonally to the surface everywhere, which improves the learnability of the inpainting process.

1.2. Contributions

We introduce a robust specialized method to apply texture inpainting operations to 3D models.

Frameworks. We identify, implement and test two distinct frameworks for using our new inpainting in practice (in Section 4) that can be used in isolation or in conjunction. One, which is fully automatic, targets texturing defects stemming from shortcomings inherited by original UV-map, after a new better UV-map is available; the other framework is interactive, and allows a user to target any texturing defects by painting over them with a ‘fixing’ brush.

Both frameworks are modular with respect to the specific inpainting method employed, and can seamlessly integrate and benefit from future advancements in the field. The idea is to improve the performance and applicability of any existing image- or texture-completion methods by running (and possibly training) them on a domain more friendly to inpainting, given by a novel specialized auxiliary parametrization, and by automatizing their application.

Specialized local parametrization. Given an area of the mesh to be inpainted, we need to produce a parametrization over the plane for it and its surrounding region, minimizing the mapping distortion; to reduce the distortion, we allow for carefully placed cuts to be introduced. This task is only partly similar to the extremely well-studied problem of producing an almost-isometric (global) parametrization for a given mesh. The differences in objectives are discussed in Section 5.1, and our construction procedure is described in Section 5.2.

Training data. The above process is also the basis for producing a dataset of images of a new category, which we use to train a network specialized for our task. As observed, this dataset is not the same as either natural images or textures, in spite of originating entirely from natural images. It can be described as pictures of the surface of natural objects transformed so that the image is (almost) orthogonal to the surface in every position. We describe a procedure to produce the training data in Section 6.2, and make the resulting data available for future investigations.

Automatic identification of UV-mapping induced defects. We present an algorithm in Section 7, to identify the areas requiring inpainting, considerably reducing the workload required by users to benefit from texture inpainting techniques.

To facilitate reproducibility and adoption of our work, we provide a reference implementation, completed with the inpainting network, the training data, the interactive texture inpainting system and the automatic command-line inpainting tool.

2. Related work

Our technique has predecessors in automatic or semi-automatic systems designed for alleviating texture defects of models, often explicitly targeting photogrammetric models. Some, like Huang *et al.* [HDGN17], work on the original pictures used for the reconstruction, which, in contrast, we do not need to assume to be available; a few methods [TSPD16] apply inpainting to in *image space*, while others [FDGM18, SKCA20] (including ones integrated in commercial products [Ado22]) apply inpainting in *texture space*, incurring in the respective limitations we outlined in Section 1.

Several steps of our technique require facing challenges that have been studied in isolation.

2.1. Parametrizing photogrammetric models

Techniques for reconstructing a 3D model from a set of digital photos, based on Photogrammetry or Structure-From-Motion approaches [HZ04, SCD*06, RSN*14] are commonly used to create 3D textured assets for many different areas such as cultural heritage (for documentation, analysis and preservation), the entertainment industry, digital art, personal leisure and many others. These models have often very complex UV-map layouts and many issues as documented, for example in Maggiordomo *et al.* [MPCT20]; while some recent approaches try improving these assets working on the parametrization [MCT21], there remains the problem that the content of the photo-reconstructed object often presents small artefacts that need to be corrected.

2.2. Image Inpainting

Image inpainting [BSCB00] refers to the process of reconstructing specific portions of an image while maintaining consistency. Inpainting is a low-level image processing task with many different applications, such as photo editing, object removal and restoration of corrupted parts. There is a vast literature on this subject. Traditional approaches use either diffusion-based image-synthesis approaches [BSCB00, BBC*01] or reuse portion/patches of the current image [EF01, BSFG09]. Some methods work on the structure of the scene to get some kind of guidance by manually specified points of interest [DCOY03], or by perspective distortion [PSK06]. Other attempts to automatically recover these structures by exploiting tensor voting techniques [JT03], searching in tile space [KKDK12] or by recovering perspective planar surfaces [HKAK14]. Older patch-based approaches shared the common limitation that the synthesized portion can come only from the existing input image. A first method that exploited a very large collection of general images was proposed in Hays and Efros [HE07] and others followed the same concept on more specialized problems like faces [MPK09, DDZ11].

In the last 10 years, inpainting increasingly relies on Convolutional Neural Networks (CNNs) [XXC12, KSSH14, RXYS15]. The first attempts suffered from limitations on image resolution and size of missing parts; fully convolutional models such as Context Encoders [PKD*16] and Generative Adversarial Networks [ISSI17] eased these limitations.

Recent approaches based on Deep Learning [GYH21, HZW*21] have been proven to work well with images drawn from specific classes (e.g. faces, cars) for which approximating the data distribution is somewhat easier, but the inpainting of generic images remains difficult. A recent review on this subject can provide further information [EAAMA20].

2.3. Geometric deep learning and textured meshes

Geometric deep learning is a rapidly evolving field, and recently novel network architectures have been proposed to operate on 3D models enriched with texture data. TextureNet [HZY*19] introduces a 3D convolution operator based on local surface patches [BMRB16, MBBV15] that leverages a 4-RoSy direction field [JTPSH15, HZN*18] to parametrize a local neighbourhood around each vertex and extract consistent feature descriptors on 3D surfaces of arbitrary topology. PFCNN [YLP*20] introduces a similar convolution operator that operates on aligned tangent spaces to match the behaviour of 2D convolution over images. These methods apply convolutions to feature maps resampled to local grids, but still output vertex descriptors in the end; conversely, we need to *synthesize* high-resolution signals, which remains an interesting future direction and generalization of such patch-based approaches.

TM-Net [GWY*21] is a generative model that has been recently proposed to synthesize 3D textured objects. The model produces objects assembled from 'parts' modeled as deformed squares tessellated at low resolution, with the texture data synthesized by a separate network and to a fixed UV layout. Contrary to this, we need to be able to handle arbitrary geometry and UV atlas layouts.

2.4. Local parametrization

Constructing a global parametrization for a given surface is a longstanding open problem [FH05], in spite of recent progress such as, among others, [LKK*18, ZLG*18, PTH*17, LYNF18]. In a sense, the problem is insurmountable because an ideal parametrization that is almost isometric and free from seams does not exist for a generic surface, and automatic construction techniques can only choose between different trade-offs, primarily between cuts (loss of continuity) and isometric distortions (loss of isometry). Where appropriate, this awareness has led to the development of solutions that, instead of seeking a static parametrization, construct, on the fly, a new local parametrization over a small surface area surrounding the region of interest. The limited size of the surface lowers the total amount of curvature that has to be flattened, reducing the need for cuts or distortions. This approach has been followed successfully for tasks such as surface analysis [PCCS11], 3D model inspection [PTW13], texturing implicit surfaces [SGW06] and many others. Our work also falls in this category, and we employ this approach for inpainting over texturing defects. To this end, we adapt our parametrization construction algorithm to the requirements of this particular scenario, which are detailed in Section 5.1.

3. Overview

Our method works on a photo-reconstructed 3D triangular high-resolution irregular mesh M, enriched with a texture image T connected to M via a UV-map U (*i.e.* a per-vertex assignment of texture coordinates). The input texture T is obtained by photographs shot at the real object as part of the acquisition process, and presents localized defects of various nature.

3.1. Texture inpainting operations

We correct the texture with a succession of individual *texture inpainting operations*, each targeting individual defective areas.

An inpainting operation acts on a subset of the mesh, *i.e.* it takes as an additional input a tagging of the mesh faces whose texture content is to be discarded and replaced by new content automatically generated according to the surrounding context. We perform this operation by first parameterizing the affected area and its neighbourhood into a square region S (Section 5) that serves as the domain of the inpainting operation (Section 6); specifically, we resample the texture T over S (Section 6.1), perform the inpainting operation on the resulting image and resample the newly created texels over T.

3.2. Sources of texture defects

We need to distinguish between two classes of sources of texture defects.

Defects stemming from the parametrization. A class of texturing defects originates from issues in the UV-map U, resulting in the inadequate distribution of texels of T over M. Such issues include excessive angle or area distortions, and overlaps, either caused by flipped triangles, or (self-)overlaps of texture charts; all these cases are common in photogrammetric models [MPCT20], reporting them in 83% of the datasets. This class also includes the case of minor colour discontinuities appearing at texture seams, *e.g.* due to photograph misalignments. Another instance is resolution jumps in the texture domain, which are frequent because the resolution of a textured area is often linked to the pixel density of the source photograph, which depends on factors such as shooting distance or camera resolution.

Defects stored as incorrect colours in T. This class includes defects originating from all other reconstruction errors, including uneven lighting between photos, reflections and other view-dependent lighting effects which end up stored in the diffuse map (for example, by failures of de-shading algorithms), wrong photo re-projection due to camera registration errors, unaccounted photographic radial distortions, inaccuracies in shape reconstructions *etc.* Re-projection errors are particularly disruptive in areas where the texture source switches from one photograph to another (not necessarily in correspondence with a UV seam), and appear as either discontinuity artefacts or ghosting (when photographs are blended). Pixels from source photos can also be re-projected over parts M which are very far from the correct one, due to misaligned depth jumps.

Our texture inpainting operations can be used on both classes of defects, but within different frameworks (see next section).

4. Frameworks of use

We propose two different frameworks to issue the sequence of texture inpainting operations (Section 4): a fully automatic one, targeting defects stemming from the parametrization, and an interactive one in which defects are corrected on demand, using an interactive 3D brush tool. The two frameworks can be used individually or in cascade. We discuss other potential frameworks of use in the conclusions.

4.1. Fully automatic pipeline

Problems stemming from the UV-map (Section 3.2) cannot be fixed by working on T alone because any reconstructed signal still cannot be represented using the same UV-map U. It is necessary to first produce a better UV-map U_{new} for M, either from scratch, discarding and substituting the original U [LZX*08], or incrementally, by modifying U locally [MCT21] (step 1, in Figure 3). The creation

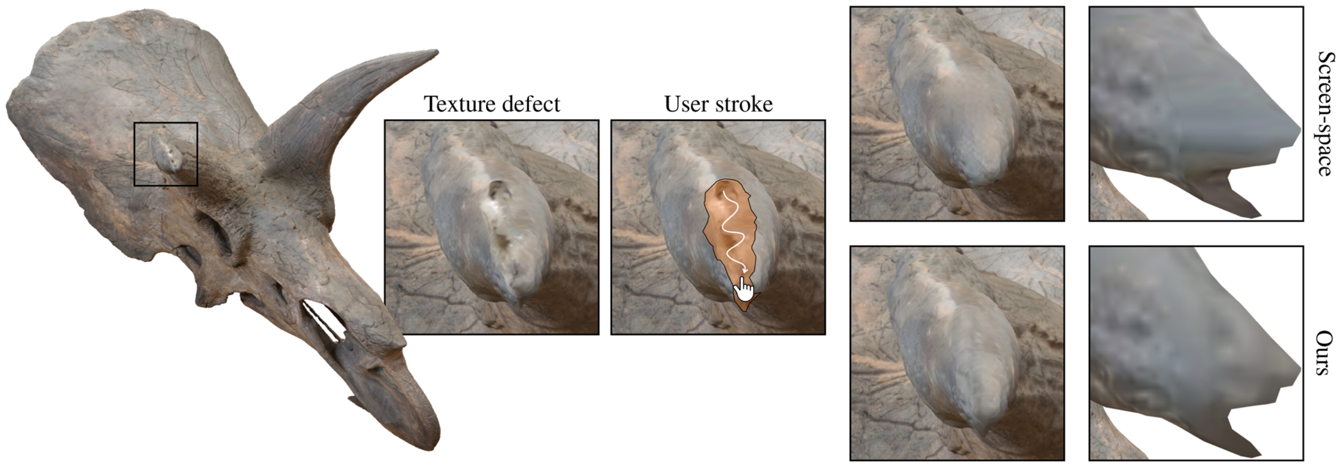

Figure 3: *The fully automatic pipeline, showing the context of use of our texture inpainting technique (step 4). See text.*

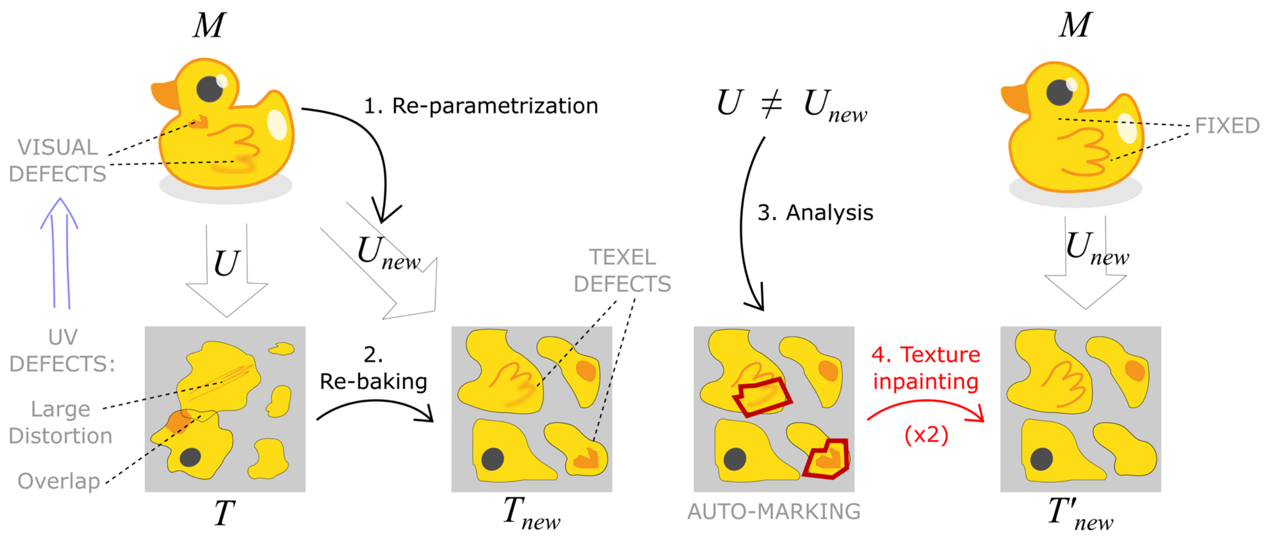

Figure 4: *An example of our inpainting operation issued by an interactive stroke of a 'fixing' brush performed over a photogrammetric 3D model M of a lion (2), originally featuring a texture defect (1). The system constructs a local parametrization $\mathcal{P}$ for a region surrounding the stroke, fills its domain with texture data from the model texture T, performs an inpainting operation on the region affected by the brush (big arrow) and transfers the result back into the texture image, fixing the texture defect (3).*

of U_{new} is completely orthogonal to our method, and any existing technique can be adopted. In our experiments, we use Jiang *et al.* [JSP17].

Given the artefact-free U_{new}, we re-sample the original texture image T into a new texture T_{new} (step 2 in Figure 3). In doing so, the original shortcomings of U are transferred into the texels of T_{new}, where they can be fixed.

By analysing and comparing U and U_{new}, we automatically mark all faces of M that presented sampling problems in U, but are no longer an issue in U_{new} (step 3 in Figure 3). We detail our algorithm to do so in Section 7. The marked faces are then divided into connected components, and inpainting operations are issued on each component separately, in cascade (step 4 in Figure 3).

4.2. Interactive mode

In the interactive framework, we apply the inpainting method on demand, allowing a user to simply paint the areas of M perceived as defective with a 'fixing' brush, similar to the ones used commonly in 3D painting or sculpting interfaces (see Figure 4). From the user's perspective, each stroke of the brush triggers an inpainting operation over the covered triangles, and the results are shown back immediately, at the end of the stroke, by updating the texture.

The interactive mode sidesteps the need to automatically identify defects, and the difficulty of even defining what constitutes a defect in T (which is not trivial, in absence of a ground truth). On the other hand, it requires inpainting operations to be performed in interactive times, a feature that our method provides.

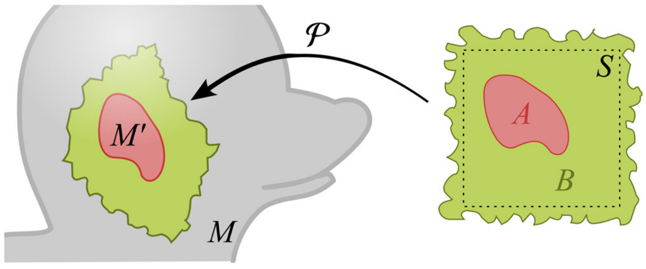

Figure 5: *Our auxiliary parametrization $\mathcal{P}$ mapping a planar region B, which encapsulates a square S, onto the surface M (left). A central sub-region $A \subset B$ is mapped into $\mathcal{P}(A) = M'$, corresponding to the region to be inpainted (red), while the rest of $B \setminus A$ is mapped into an area of the mesh that will provide the context for the inpainting operation (green).*

5. Auxiliary parametrization

Given a cluster of connected faces $M' \subset M$ that needs to be inpainted, our first task is to flatten it and its surrounding region into a squared 2D area S, which will serve as the input domain for the inpaint operation. We do so by building a local 'auxiliary' parametrization function $\mathcal{P} : B \subset \mathbb{R}^2 \to \mathbb{R}^3$ that maps a region of the plane B into some subset of M encapsulating M' (see Figure 5). Area $A \subset B$ must be mapped into $\mathcal{P}(A) = M'$, and it will be filled by the inpainting procedure (red in the Figure 5); the rest of $B \setminus A$ must be mapped into a surrounding mesh area M', and provides the context for the inpainting procedure (green in Figure 5).

Note that in our notation, the parametrization function goes from the manifold to the plane, rather than the inverse (which is more customary), because in our case, $\mathcal{P}$ can be non-injective and therefore non-invertible.

The problem of constructing this parametrization (that is, determining A, B, the function $\mathcal{P}$, and its image on the mesh $\mathcal{P}(B)$) is related to the extensively studied field of surface parametrization, but there are differences in our objectives, as follows.

5.1. Objectives for the auxiliary parametrization

Low distortions. Like in most other texture-mapping related contexts, we want to minimize isometric distortions, that is, we want $\mathcal{P}$ to be as length preserving as possible. Our tolerance for distortions is particularly low in the A domain, where they would have the effect to deform the inpainted image while transferring it back to the final texture. Distortions in the $B \setminus A$ part of the domain are more tolerable, but must also be kept low, as they may hinder the learning process during both training and the application of the inpainting by disrupting the context. One motivation for using a localized parametrization (in place of the provided parametrization U_{new}) is that a smaller mesh can be parametrized with lower distortions.

Cuts scarcity. We require the region A to be free from texture seams, that is, to be mapped with continuity over $\mathcal{P}(A) = M'$ (the mesh region to inpaint). On the contrary, on the rest of the domain $B \setminus A$, we can introduce texture cuts, whenever they are needed to lower the distortion.

Undefined boundaries. Differently from the typical parametrization scenario, we do not know beforehand which part of the mesh $\mathcal{P}(B) \subset M$ must be parametrized. We require it to encapsulate M', that is, $\mathcal{P}(B) \supset M' = \mathcal{P}(A)$; we also need the parametric domain B to fully encapsulate a large *squared* region S around A; finally, we want A to be far from the boundaries of S, so that sufficient context is given to the inpainting procedure. Within these requirements, we are free to define $\mathcal{P}(B)$ and B in the best way to fulfil the other objectives.

Non-strict injectivity. In contrast to typical scenarios, we do not need P to be fully injective. Instead, we tolerate that different regions inside B (but not inside A) map into the same part of M. This will cause some pictorial detail over M to be repeated inside the context region of S. Allowing for repetitions is necessary toward the goal of filling the entire S in the parametric domain, in cases where M does not provide enough surface around $P(A)$ due to extreme intrinsic curvature or open boundaries. From an inpainting perspective, pattern repetitions in the context area are preferable to either leaving empty gaps inside S, or distorting the details more (see Figure 6).

5.2. Construction of auxiliary parametrization

To fulfil our objectives, we adopt the following steps.

1. We construct A and parametrization $\mathcal{P}_A : A \to M'$.
2. We initialize B and $\mathcal{P} : B \to M$, to A and $\mathcal{P}_A$, and iteratively extend B, one triangle at a time, until a sufficiently large squared region S around A is fully covered.

In the first phase, any existing single-patch disk parametrization method can be employed. We opted for the local–global As Rigid As Possible (ARAP) parametrization [LZX*08], because of its robustness and performance in terms of isometric distortions. The mapping is not strictly guaranteed to be free from local overlaps (which would infringe the required injectivity), but we never encountered this issue. If need be, existing countermeasures can be applied during the global phase of ARAP [BCE*13].

The second phase follows the spirit of the incremental approach of Myles and Zorin [MZ12]. We initialize B as A, and at every step (see Figure 7), we select one boundary edge over B, according to a prioritization strategy (see below), and attempt an *expansion* step of B over that edge, adding to B the parametric domain t for one new triangle of M, and a new one vertex v to the boundary δB; the 2D position of v is determined as the minimizer of the total isometric energy [LZX*08] defined over t (keeping its two other vertices constant). After every expansion, we evaluate up to two potential *merge* steps of v (an operation called 'edge retract' in Hoppe [Hop96]) with either the boundary vertices v' and v'' located two edges away from v on the boundary, clockwise and counterclockwise; a merge operation consists in fusing together two consecutive boundary edges, and a potential operation exists if and only if these two edges are mapped by P into the same mesh edge (see Figure 7, right).

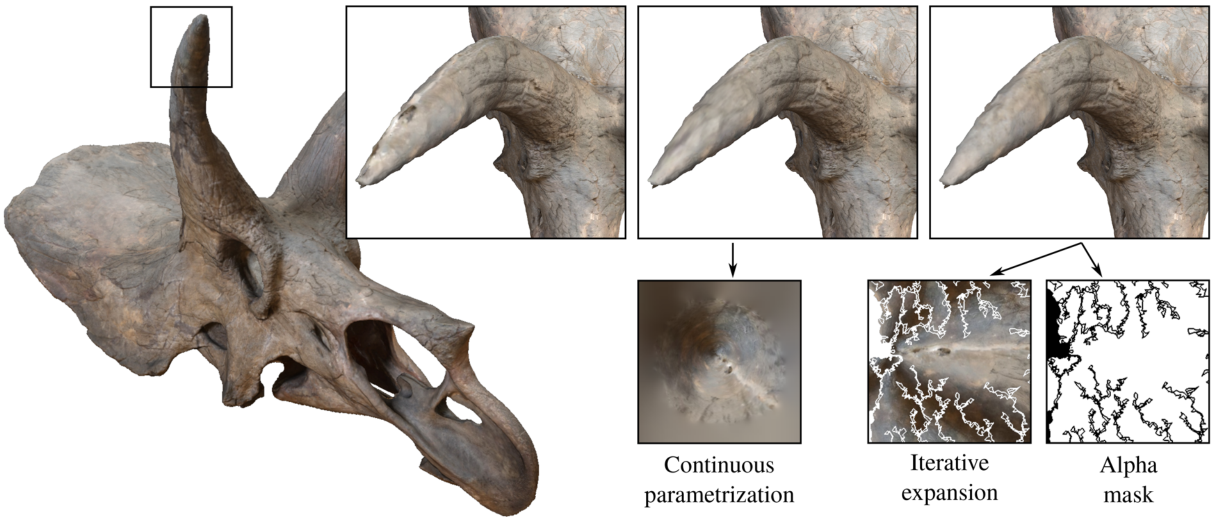

Figure 6: *An inpainting operation covering a highly curved region. Using a* continuous *auxiliary parametrization, without cuts or repetitions, introduces severe distortions, which negatively impact the generation and transfer of inpainted texels, and result in stretching artefacts (middle inset). By allowing cuts, repetitions and overlaps, we produce a better context for the inpainting operation (rightmost inset). Overlapping texels are marked in the alpha mask, are shown in white in the RGB image and as black in the alpha masks.*

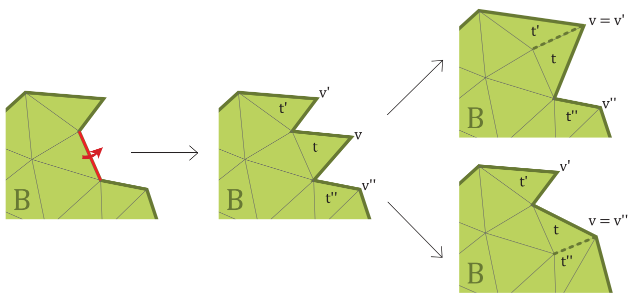

Figure 7: *An expansion step, performed over one edge of δB (in red, left), expanding B and δB (middle), potentially followed by either one or the other (or both, in succession), fusion step (right). See text.*

The details of this step are as follows.

Creating cuts. After a merge operation, the 2D position of the merged vertex is redefined as the minimizer of the combined, area-weighted, ARAP energy of all affected triangles in B. If the minimal ARAP energy value is larger than a tolerance value (we used 0.1), the merge operation is rejected. In this case, a seam is created in the final parametrization. Differently from similar approaches, we do not 'freeze' these edges, and both can be elected in subsequent expansion steps, thus creating texture repetitions (see below).

Crack-less overlaps avoidance. Different from similar approaches, we do not prevent the expansion or merge steps that would create an overlap of the patch. Instead, simply not elect for expansion any edge which is already *fully* in the interior of B. In other words, expansion of B is stopped after the overlap occurred. In this way, thus creating a narrow (up to one triangle wide) overlapping area, instead of a narrow empty area, at the seams of B. These overlaps will be dealt with in the subsequent patch creation phase.

Repetitions. We also allow expansions over triangles that are already represented inside B, resulting in potential repetitions, in S, of the textures of the triangles of M visited multiple times; while not ideal, repetitions are preferable to empty regions. For example, when unfolding the side area of a thin cylinder, B can wrap around M multiple times. As another example, when a merge operation is rejected (due to excessive distortions), both copies of the same mesh edge in parameter space can still be expanded in subsequent steps, repeating the triangle on the other side of the edge anew.

Definition and update of S. After the initialization phase, we set S as the square having four times the area of that Axis Aligned Bounding Box (AABB) of A, or its largest dimension (whichever is larger). After every step, we recentre S into the centre of the AABB of B.

Edge prioritization strategy. At every step, we pick for expansion the edge in δB that is currently the farthest from the boundary of the rectangle S, so to encourage a uniform expansion of B over all S; more precisely, we score each vertex along δB with its distance from the nearest of the four sides of S, identify the highest scoring vertex v_{max}, and pick the edge on either side of v_{max} (we choose the side connecting v_{max} with the highest-scoring vertex).

Stopping criteria. We never pick one edge for expansion that is fully outside S, or already fully in the interior of B. When no edge

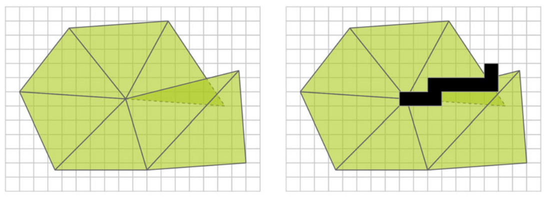

Figure 8: *The alpha channel of the texture patch is zeroed (black pixels, right) to annotate the discontinuities near the border edges of overlapping triangles.*

in δB is eligible, we stop the loop. At that point, the entire surface of S will be covered at least once.

6. Texture inpainting network

The next step is to inpaint the auxiliary texture, that is, to automatically generate new texel values to fill in the texels in the area marked as 'defective', according to the context around that area.

We employ a close adaptation of a recently proposed Deep Convolutional inpainting network [LRS*18], although any other method can be plugged into our framework. This model uses an encoder–decoder architecture with skip-connections analogously to UNet [RPB15], and partial convolutions [LSW*18] to improve performance with masked image inputs. The network takes as input an image and a binary mask denoting valid and invalid pixels, and outputs a reconstructed RGB image.

We leave the network architecture and reconstruction loss mostly unaltered from previous art, but perform training on a dataset created specifically for our task. We implemented the model in PyTorch.

6.1. Texture re-resampling

Once we produce the auxiliary parametrization, we resample the original texture into S, and generate the binary mask by marking all pixels inside A.

We perform this task with a GPU-assisted rendering over S into a texture patch. The original texture is accessed with a standard bilinearly interpolated access to the original texture. The mask is drawn by rendering all the marked triangles in the patch.

As discussed, small areas of S can feature overlapping triangles (in correspondence with parametrization cuts), meaning that this rendering will present pixel overdraws (we render the triangles in inverse order of expansion, and disable depth-testing to ensure the last drawn textured triangle overwrites the others), which will potentially generate artificial colour discontinuities. We generate an extra binary alpha channel to annotate such discontinuities, by drawing the visible edges of the overlapping triangles (Figure 8). The additional 1-bit channel is then fed to the network, together with the RGB channels; this is intended as a way to 'inform' it of the artefact nature of the colour discontinuity.

6.2. Training

One of the biggest drawbacks of data-driven approaches, and deep learning in particular, is the requirement of large amounts of data to drive the training process. Prior knowledge about the data itself can significantly affect the performance of neural networks, both in terms of network architecture, and what kind of data the network is exposed to during training.

Therefore, we choose to train the inpainting network using a task-specific dataset, that is, texture patches created from scanned objects, instead of using general-purpose datasets of natural images that are already available. This is motivated by the peculiarity of our inputs which, as discussed, differ significantly from generic photographs. Since texture patches lack features such as perspective distortion and depth jumps, the training process can be expected to require less data, and to converge more quickly, for the same quality. For the training, we generate a collection of around 50,000 patches sampled from a public benchmark of textured 3D real-world models [MPCT20] (100 patches from each model). We create training and validation sets with a random 85–15% split.

Patch extraction. Given a textured model, we extract a number of patches by simulating strokes over the mesh and applying the auxiliary parametrization construction (Section 5.2), followed by the synthesis of the mask and the RGBA image (Section 6.1)

Training. The network is trained with batches of 24 samples at 256×256 pixels. We use 340 epochs of 40k images each, and fine-tune for 60 more epochs. Because texture patches contain not only RGB data, but also an additional channel of binary alpha values marking colour discontinuities induced by the overdraw, we alter the first layer of the inpainting network to optionally accept four-channel inputs. The output is still an RGB image and the reconstruction loss used to train the model simply ignores the extra input channel. The training was carried out on an NVIDIA RTX 3060 GPU with 12 GBs of VRAM; loss values and validation performance are reported in Figure 16, bottom.

7. Automatic UV-map defect identification

In our automatic framework (Section 4.1), triangles in M that require an inpainting process are identified by comparing the original UV-map U, where the original texture image is defined, with the new UV-map U_{new}, which is provided to the system and used to resample the final texture.

The rationale is that U_{new} is assumed to be constructed to minimize isometric distortions with the 3D mesh M, but balances this objective with a number of other requirements (such as sparsity of cuts, or different texture resolution for different parts of the mesh, etc). While U_{new} defines the *requested distribution* of texels on M, U defines *the actual availability* of texels.

7.1. Per triangle labelling

First, we analyse each triangle t of M and determine if that triangle requires inpainting, that is, if its image in T fails to provide an adequate source of texels for t. This is the case if any of the three problems below is detected.

Excessive distortions. Let u_0 and u_1 be 2D triangles that are the images of t in the two UV-maps U, U_{new}. The final texel re-sampling will consist in sampling a regular texel grid over u_1 and, for each sample, copy the (interpolated) colour value from the position in u_0 at the same barycentric coordinates. We employ the notions, well understood in the context of surface parametrization, to quantify the *distortions* of this mapping [FH05] (one difference being that our mapping is between 2D triangles). This analysis determines if the re-sampling will result in oversampling or undersampling, in any direction. Specifically, let F denote the linear map from u_1 to u_0 and J_F its Jacobian, that is

$$J_F = [a_0|b_0][a_1|b_1]^{-1} \tag{1}$$

with a_0, b_0 being 2D column vectors denoting two arbitrarily chosen 2D edge-vectors in u_0, and a_1, b_1 being the corresponding ones in u_1; let σ_0 and σ_1 the singular-values extracted from the Singular Value Decomposition of J_F ($\sigma_0 \leq \sigma_1$) If σ_0 is larger than a given threshold, we mark the triangle as defective. As a default value for the threshold, we use 1.2, which corresponds to the choice to mark all and only the triangles region that will undergo an over-sampling of 20% in any one direction. If $\sigma_{0,1} < 1$, this denotes a situation where the original texture is undersampled to produce the final texture, and requires no intervention (because we assume that the sampling rate determined by U_{new} is the desired one).

Overlaps. We mark the triangles of M whenever u_0 overlaps with the image of any other triangle in U. This situation occurs when U is non-injective, unavoidably leading to texture artefacts when the same area of the original texture is copied multiple times over different areas of the final texture. Any detected overlap invalidates both overlapping triangles. To speed up detection, we use a regular grid paired with hashing as a spatial indexing structure.

Seam discontinuity. Another test we perform is to check whether or not the colour signal stored in the original texture T happens to be mismatching on the two sides of any texture seam of U (in other contexts, such as CAD models or manually modelled game assets, this occurrence can be assumed to be by design, as texture seams often separate semantically different areas; in photogrammetric models, which are our target, it evidences a likely reconstruction defect, as discussed in Section 3.2). To detect the mismatch, we linearly sample every seam edge on both sides, collecting bi-linearly interpolated samples at the two corresponding locations of T, and measure their Euclidean (r, g, b) distance. If the difference, averaged over the edge, is larger than a threshold (we used 0.1 in a colour intensity scale from 0 to 1), we mark any triangle sharing one vertex with that edge.

7.2. Clustering defective faces

Once the triangles have been individually marked, we group them into connected components, each to be addressed by one inpainting operation (Figure 9). This may lead to an excessive number of patches, with jagged and discontinuous borders. To address this issue, we apply a patch border regularization using morphological operators, in a dilate-erode procedure. We apply two steps of dilation (marking any triangle sharing one vertex with an already marked one) followed by two steps of erosion (un-marking any triangle sharing one unmarked vertex).

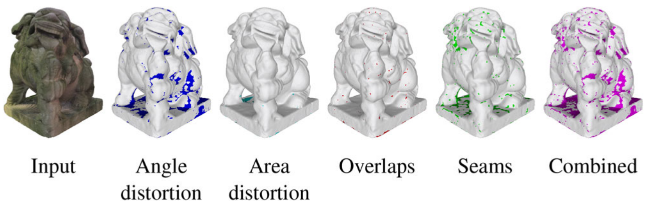

Figure 9: *Segmentation and clustering of defective faces.*

8. Results and experiments

We empirically validate both our interactive and fully automatic frameworks, by performing a range of experiments on real-world data aimed at producing a qualitative (Sections 8.2, 8.3 and 8.6) and quantitative (Section 8.4) assessment, including direct comparisons with competing strategies.

8.1. Examples of results

Interactive inpainting sessions. Figures 10, 11, and 18 and the attached video show examples of results of various interactive inpainting sessions. More experiments of this kind can be conducted using the attached implementation. The network correctly interpolates textures and recovers details that are consistent with the patch surroundings. In Figure 11, the isotropic content in the rock texture is correctly replicated, while in Figure 10, the inpainted areas correctly capture the anisotropy of the wood grain and fibres; in both cases, realistic looking missing detail is produced that seamlessly blend with the surroundings.

Automatic defect correction. Figure 12 shows the results of inpainting the automatic framework (Section 4.1), where texture defects are automatically identified and corrected. The procedure is reasonably quick, with the iterative inpainting accounting for most of the processing time. In the reported example (400k faces), the entire process takes around 10 min. More examples can be produced using the command-line tool in the provided implementation. While simple, our heuristic detects several visible artefacts, which are effectively corrected by the inpainting procedure. On occasion, the heuristic produces patches that are too large for the neural network to inpaint effectively, producing visible artefacts (see Figure 19).

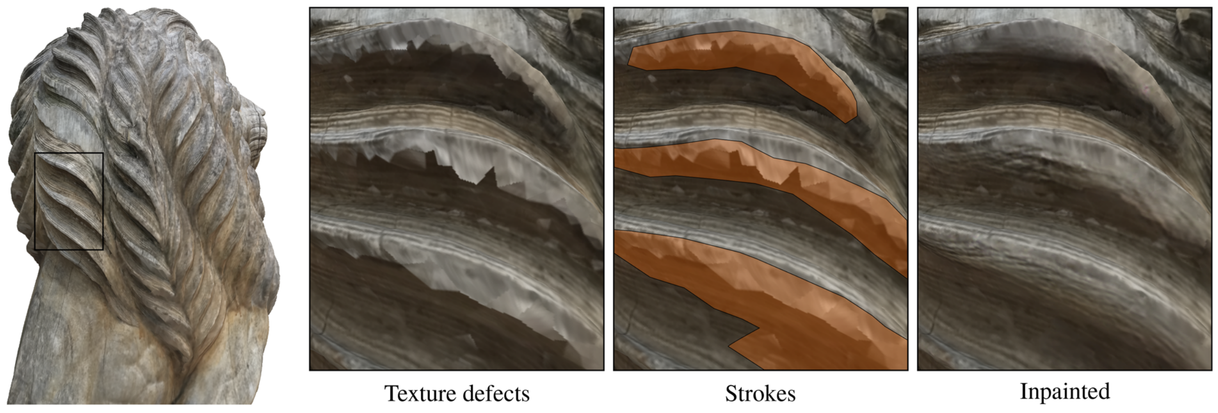

Figure 10: *Interactive inpainting of a wooden statue. The interpolation over the artefacts correctly reproduces the wood grain.*

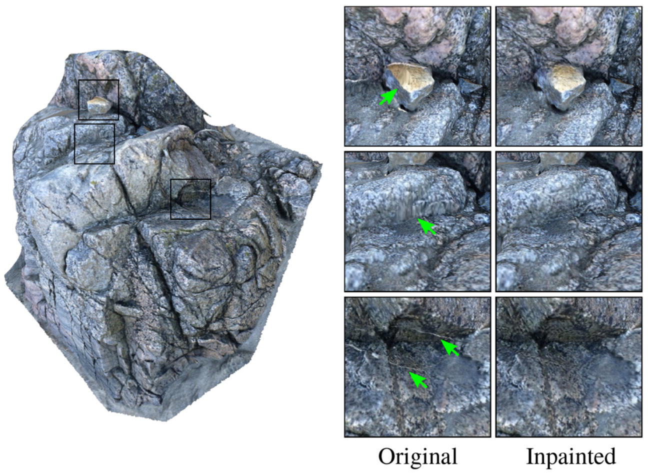

Figure 11: *Interactive inpainting of a textured rock using the texture-space network. The network correctly reproduces the surface texture and remedies artefacts. Sharp discontinuities in the surface material (first row) are more difficult to treat and the network tends to produce overly smooth transitions.*

8.2. Comparative evaluation

We compare our results against two alternative strategies: inpainting performed in screen-space, and inpainting performed in texture-space using texture synthesis.

Comparison with screen-space inpainting. For this comparison, we trained the same network from scratch on generic photographs taken from the Places365-Standard dataset [ZLK*18]. We used batches of six samples at a resolution of 512×512 pixels, as suggested in Liu *et al.* [LRS*18], and a learning rate of $2 \cdot 10^{-4}$ for 140 epochs (35k images each), followed by a fine-tuning with a learning rate of $5 \cdot 10^{-5}$ for 30 more epochs. Overall, the process took about 10 days.

Visual results can be seen in Figure 13. Experiments evidence a significant quality improvement in moving from screen-space inpainting to texture-space inpainting, (third to fourth column). The screen-space inpainting systematically produces visible artefacts and is unable to correctly capture the silhouette of the 3D geometry and its impact in the overall image lighting. As a result, the inpainted area always looks flat and disconnected from its surroundings.

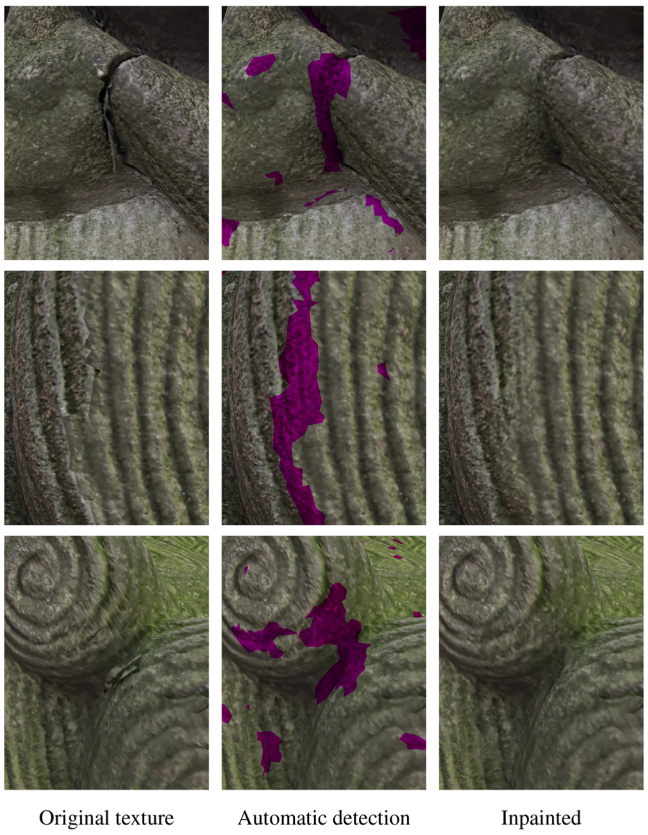

Figure 12: *Inpainting of automatically detected defects. Top: UV-overlaps and texture seams. Middle: visible texture seam due to the colour discrepancy induced by adjacent texture charts with different sampling resolutions. Bottom: UV-map distortion.*

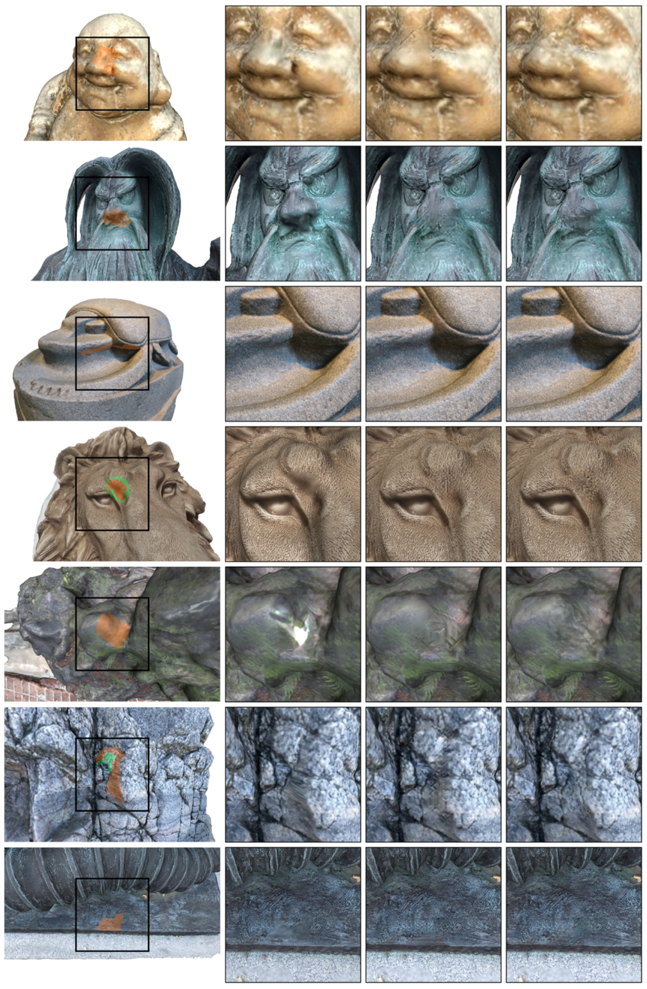

Figure 13: *Visual comparison of inpainting results in different spaces. The framing is the same one used for screen-space inpainting.*

Conversely, out inputs constructed from the auxiliary parametrizations visibly improve the results both in texture richness and smoothness of blending the inpainted patch with the surroundings. Other insights on the differences between texture-space and screen-space inpainting are given by the assessment of the respective training process, in Section 8.5.

An additional, inherent advantage of texture-space inpainting is that the results are independent of the observer's position. This is shown in Figure 2, where a texture defect is interactively selected for inpainting. With screen-space inpainting, switching to a different view reveals projection artefacts induced by perspective distortion; our method, instead, is only affected by the distortion of the underlying parametrization, which is usually drastically lower.

Comparison with texture synthesis. For this comparison, we use a publicly available implementation of a State of the Art texture synthesis technique [Opa19, Emb19, EL99, WL00], and feed it the same input as that of our neural network (the input texture patch, and a binary mask marking the area to be inpainted). We have integrated this tool in our implementation; the GUI exposes it as a different inpainting mode, easing the comparison with the neural model on the exact same inputs.

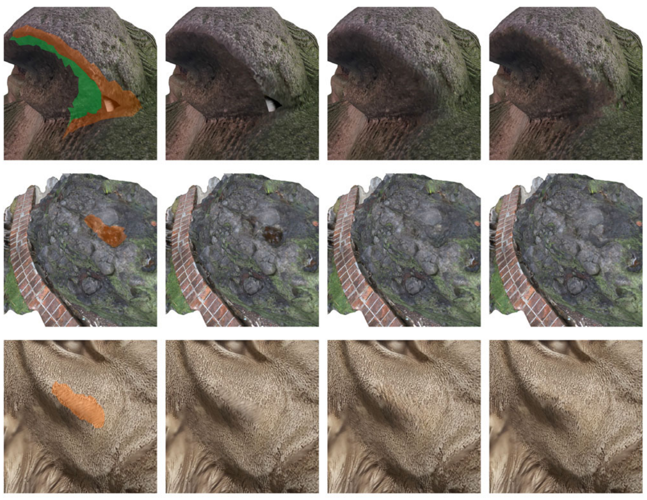

Figure 14: *Inpainting with the texture-space network and texture synthesis produces results of similar quality. Texels interpolated with the neural network blend smoothly with the surrounding context, while texture synthesis occasionally produces sharp material transitions (first row).*

Examples of visual results can be seen in Figure 14, and more can be made using the attached implementation. The results of the neural network are qualitatively similar to using texture synthesis, but sometimes the texture synthesis produces sharp transitions between surface textures (Figure 14, first row, where the texture synthesis hallucinates a sudden transition from moss to rock); when used to fix a blurred input, texture synthesis sometimes produces lower frequencies than desired (third row).

8.3. Comparison with inpainting in global texture space

Figure 15 compares our local strategy that leverages the auxiliary parametrization against a more trivial approach of inpainting directly in global texture space. For global texture inpainting, we use a neural network trained on texture patches without overlaps and alpha masking (as described in Section 8.4). We show an example where an inpainting operation is attempted near a seam of the global texture space, resulting in disconnected regions of the texture to require inpainting. Our approach bypasses the problem, resulting in a significant quality improvement in the produced result. Another drawback of the direct approach is that the entire texture must be fed to the inpainting model, drastically increasing the time and memory requirements of the operation. For reference, inpainting a 4k texture with our CNN model allocates more than 16 GBs of memory and is an order of magnitude slower compared to inpainting our 512×512 patches.

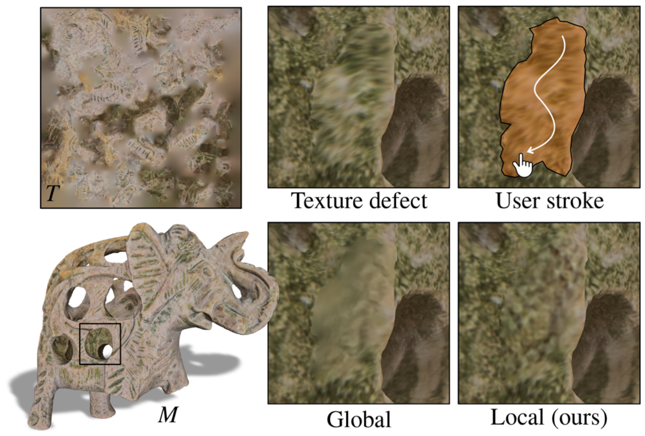

Figure 15: *Comparison of inpainting in global texture space and our approach of building a local patch. Global inpainting causes the unused background texels to bleed into the texture charts.* Model: Broken Stone Elephant by Andrea Spognetta (CC BY-NC 4.0).

Table 1: *Validation of our inpainting approach using the SSIM score on texture patches extracted from unseen 3D models.* Natural images *is the screen-space model (trained on the Places2 dataset),* Textures *is the texture-space model without UV-overlaps and alpha masking,* Textures with Alpha mask *is our proposed model, trained on input domains with UV-overlaps and alpha-masking. SSIM scores are computed using [Ima21].*

	Network trained on		
	Natural images	Textures	Textures with Alpha mask
Average SSIM	0.526	0.546	**0.551**
Winning rate	12.6%	39.1%	**48.3%**

Bold values indicate the best score for each row.

8.4. Quantitative evaluation and ablation study

We also compare results obtained by allowing or disallowing triangles to overlap the auxiliary parametrization, and report quantitative measures that validate our choice to do so. Without overlaps, empty ‘cracks’ regions appear in the image patch and are left for the network to fill.

We evaluate three different variations of inpainting in *texture-space*: feeding texture patches *without* overlaps to the screen-space network; feeding patches *without* overlaps to a network retrained on examples of the same kind; and our proposal of feeding patches with alpha-masked overlaps to the network trained on examples of the same kind.

For this quantitative analysis, we extract 721 texture patches (simulating user strokes from interactive inpainting sessions) from a fresh set of eight models *not* used in the training dataset, and measure the texture reconstruction quality using the original texture as ground truth reference. Table 1 reports the average Structural Similarity (SSIM) [WBSS04], restricted to the inpainted area, and the number of times each network produced the best score. The screen-space network is out-performed, further corroborating the argument

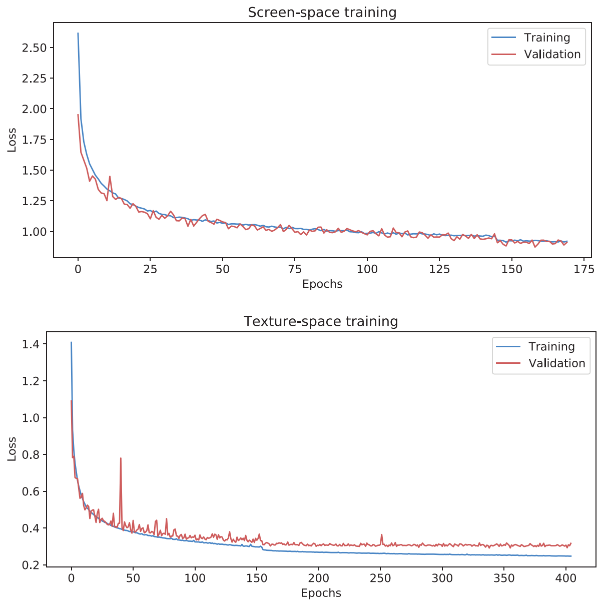

Figure 16: *Training runs for the screen-space network (top) and the texture-space network trained on our patch-dataset (bottom). The screen-space model converges with a larger loss due to being trained and validated with samples at higher resolution* (512×512 *pixels instead of* 256×256*).*

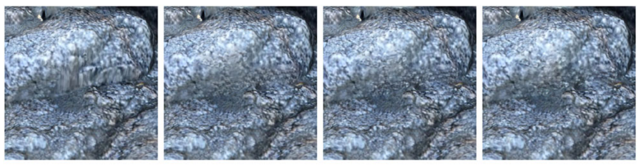

Figure 17: *Texture-space network improvement during training for stretching artefacts removal. The network progressively learns to produce patches both with richer texture detail and that blend better with the surrounding texels.*

that tailored training on texture-space patches is beneficial to our task; the inclusion of UV-overlaps and alpha masking further improves overall inpainting performance.

8.5. Effectiveness of the learning process

Figure 16 reports training and validation losses for both screen-space and texture-space networks; loss values are averaged and integrated over the image area, accounting for the resolution difference. The data indicate that both networks are successfully learning the respective domain, but a comparison suggests that the inpainting task is indeed made ‘easier’ to learn in texture-space, using the auxiliary parametrization, as we conjectured, empirically validating the premise at the basis of our work.

Figure 17 qualitatively shows the progress of the texture-space network as the training is performed. At the beginning, the

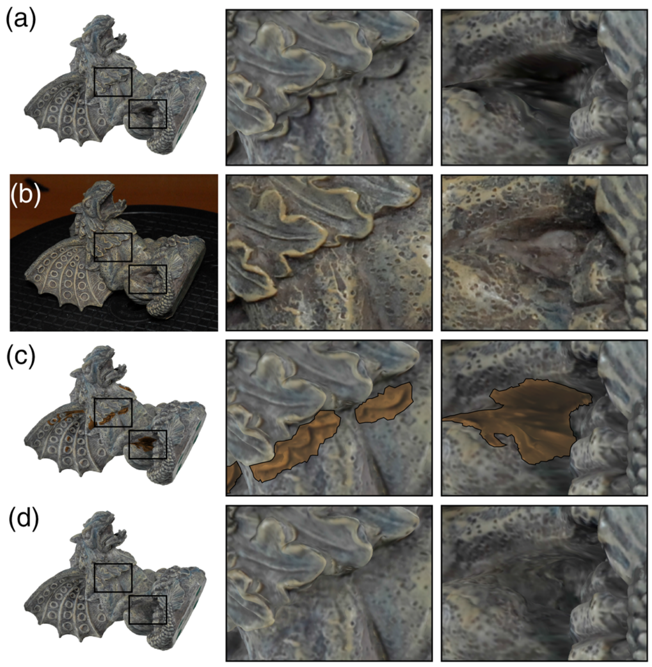

Figure 18: *A qualitative assessment of the ability of our framework to restore the look-and-feel of a defective texture. From top to bottom: (a) a rendering of a 3D reconstructed model, featuring texture defects caused by missing photographic data; (b) a real photograph of the same object (*not *used in the texture reconstruction), showing the real aspect of the defected areas; (c) the manually selected region that marks the defected region; (d) the resulting inpainted texture. On the right: two close-ups. Comparing (b) and (d) suggests that in this instance, the overall aspect of the object is successfully recovered.*

network learns to produce rich patches with varied detail; as the training advances, most of the improvement involves a better blending of the inpainted patch with the surroundings, by adjusting contrast and tone.

8.6. Comparison with real photographic data

Finally, we show a qualitative comparison of the ability of our inpainting framework to restore the original appearance of a textured object, using an actual photograph of the real-world object as ground truth.

In this experiment, we *exclude* a photograph from the texture reconstruction process, and compare it with the reconstructed texture before and after inpainting, as seen from the held-out view; the results are shown in Figure 18. In the first row, we show the texture obtained by excluding the view from the texturing process, which is affected by noticeable ghosting artefacts (first close-up) and missing colour data (second close-up) compared to the ground truth photograph in the second row. The third and fourth rows show the regions selected for inpainting and the inpainted texture, with ghosting artefacts removed and plausible surface texture generated by the network in place of the missing data.

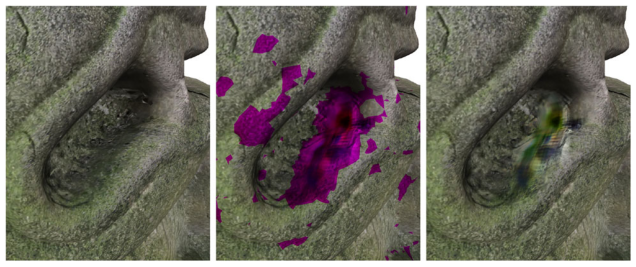

Figure 19: *Artefacts produced by the neural network when attempting to inpaint large defective regions.*

9. Conclusions

We presented a methodology to employ inpainting techniques to textures of models coming from photogrammetric acquisitions. Our contribution is orthogonal to present and future advancements in inpainting techniques themselves and provides a setup where most such techniques can be first trained and then applied. This allows for the direct inclusion of future works in a fast-advancing field such as inpainting, even if it is designed for natural images (and not textures). The new inpainting models are likely to still benefit from the present work, that is, from the idea of operating in a tailored re-parametrized texture domain, resorting to marked repetitions to limit both gaps and distortions.

Limitations. An issue of current Neural-Network-based inpainting, inherited by our system, is the limited size of the missing region in pixels. When the missing region is too large, the network can produce unexpected results, as shown in Figure 19. Using Neural Networks to inpaint high-resolution images and large missing areas is a currently developing research area, and recent works show promising results by combining and integrating patch-based strategies in the neural architecture to hallucinate high-frequencies.

Our work also inherits the inability of neural networks and texture synthesis approaches to always deal correctly with repetitive structures and patterns (*e.g.* grids, checkerboards).

Ethics and misuse. Altering the appearance of specific classes on captured models, *e.g.* cultural heritage objects or real people, raises ethical concerns; while the original texture defect does not faithfully document the reality, neither does our 'fixed' texture, which has the additional peril of deceptively looking realistic. To counteract this problem, it may be helpful to visually annotate the inpainted regions.

Future improvement. Our texture inpainting operator can be potentially employed in other frameworks for texture repairing, in addition to the ones we present in Section 4. First, it may be possible to identify areas needing inpainting by means of a separated CNN, trained for the purpose, for the cases where full automatism is a requirement; this is not trivial because there is no natural domain where to perform the detection; possibilities include the global texture or synthetic renderings. Second, the inpainting process might be embedded in reconstruction tools, letting the latter identify regions requiring inpainting (*e.g.* by providing a confidence value for each texel).

Acknowledgements

This work was partially funded by the "Social and hUman ceNtered XR" (SUN) Horizon Europe project (Grant agreement n. 101092612), and a gift from NVIDIA.

Open Access Funding provided by Consiglio Nazionale delle Ricerche within the CRUI-CARE Agreement.

References

[Ado22] Adobe: Substance 3D designer. (2022). https://www.adobe.com/products/substance3d-designer.html [Accessed 22nd December 2022]

[Agi22] Agisoft: Agisoft Metashape: A 3D reconstruction software. (2022). https://www.agisoft.com/

[BBC*01] Ballester C., Bertalmio M., Caselles V., Sapiro G., Verdera J.: Filling-in by joint interpolation of vector fields and gray levels. *IEEE Transactions on Image Processing 10*, 8 (2001), 1200–1211. https://doi.org/10.1109/83.935036

[BCE*13] Bommes D., Campen M., Ebke H.-C., Alliez P., Kobbelt L.: Integer-grid maps for reliable quad meshing. *ACM Transactions Graphics 32*, 4 (July 2013). https://doi.org/10.1145/2461912.2462014

[BMRB16] Boscaini D., Masci J., Rodolà E., Bronstein M.: Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain, 2016), Curran Associates Inc., pp. 3197–3205.

[BSCB00] Bertalmio M., Sapiro G., Caselles V., Ballester C.: Image inpainting. In *SIGGRAPH'00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New Orleans, Louisiana, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 417–424. https://doi.org/10.1145/344779.344972

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D. B.: PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions Graphics 28*, 3 (2009), 24.

[DCOY03] Drori I., Cohen-Or D., Yeshurun H.: Fragment-based image completion. *ACM Transactions Graphics 22*, 3 (July 2003), 303–312. https://doi.org/10.1145/882262.882267

[DDZ11] Deng Y., Dai Q., Zhang Z.: Graph Laplace for occluded face completion and recognition. *IEEE Transactions on Image Processing 20*, 8 (2011), 2329–2338. https://doi.org/10.1109/TIP.2011.2109729

[EAAMA20] Elharrouss O., Almaadeed N., Al-Maadeed S., Akbari Y.: Image inpainting: A review. *Neural Processing Letters 51*, 2 (2020), 2007–2028.

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH'01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, California* (New York, NY, USA, 2001), Association for Computing Machinery, pp. 341–346. https://doi.org/10.1145/383259.383296

[EL99] Efros A. A., Leung T. K.: Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV)* (Kerkyra, Greece, 1999), vol. 2, pp. 1033–1038. https://doi.org/10.1109/ICCV.1999.790383

[Emb19] Embark Studios: Texture synthesis. (2019). https://github.com/EmbarkStudios/texture-synthesis

[FDGM18] Fayer J., Durix B., Gasparini S., Morin G.: Texturing and inpainting a complete tubular 3D object reconstructed from partial views. *Computers and Graphics 74* (Aug. 2018), 126–136. https://doi.org/10.1016/j.cag.2018.05.012

[FH05] Floater M. S., Hormann K.: Surface parameterization: A tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*. N. A. Dodgson, M. S. Floater and M. A. Sabin (Eds.). Springer Verlag, Berlin (2005), pp. 157–186.

[GGC*21] Griwodz C., Gasparini S., Calvet L., Gurdjos P., Castan F., Maujean B., De Lillo G., Lanthony Y.: AliceVision Meshroom: An open-source 3D reconstruction pipeline. In *MMSys'21: Proceedings of the 12th ACM Multimedia Systems Conference, Istanbul, Turkey* (New York, NY, USA, 2021), Association for Computing Machinery, pp. 241–247. https://doi.org/10.1145/3458305.3478443

[GWY*21] Gao L., Wu T., Yuan Y.-J., Lin M.-X., Lai Y.-K., Zhang H.: TM-NET: Deep generative networks for textured meshes. *ACM Transactions Graphics 40*, 6 (Dec. 2021). https://doi.org/10.1145/3478513.3480503

[GYH21] Guo X., Yang H., Huang D.: Image inpainting via conditional texture and structure dual generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Virtual* (2021), pp. 14134–14143.

[HDGN17] Huang J., Dai A., Guibas L., Niessner M.: 3Dlite: Towards commodity 3D scanning for content creation. *ACM Transactions Graphics 36*, 6 (Nov. 2017). https://doi.org/10.1145/3130800.3130824

[HE07] Hays J., Efros A. A.: Scene completion using millions of photographs. *ACM Transactions Graphics 26*, 3 (July 2007), 4. https://doi.org/10.1145/1276377.1276382

[HKAK14] Huang J.-B., Kang S. B., Ahuja N., Kopf J.: Image completion using planar structure guidance. *ACM Transactions Graphics 33*, 4 (July 2014). https://doi.org/10.1145/2601097.2601205

[Hop96] Hoppe H.: Progressive meshes. In *SIGGRAPH'96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New Orleans, Louisiana* (New York, NY, USA, 1996), Association for Computing Machinery, pp. 99–108. https://doi.org/10.1145/237170.237216

[HZ04] Hartley R., Zisserman A.: *Multiple View Geometry in Computer Vision* (2nd edition). Cambridge University Press, Cambridge, UK, 2004.

[HZN*18] Huang J., Zhou Y., Niessner M., Shewchuk J. R., Guibas L. J.: Quadriflow: A scalable and robust method for quadrangulation. *Computer Graphics Forum 37*, 5 (2018), 147–160. https://doi.org/10.1111/cgf.13498

[HZW*21] Hui S., Zhou S., Wan X., Wang J., Deng Y., Wu Y., Gong Z., Wang J.: Auxiliary loss adaption for image inpainting. *arXiv preprint arXiv:2111.07279* (2021).

[HZY*19] Huang J., Zhang H., Yi L., Funkhouser T., Niessner M., Guibas L. J.: Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, California, June 2019).

[Ima21] ImageMagick. (2021). https://imagemagick.org

[ISSI17] Iizuka S., Simo-Serra E., Ishikawa H.: Globally and locally consistent image completion. *ACM Transactions Graphics 36*, 4 (July 2017). https://doi.org/10.1145/3072959.3073659

[JSP17] Jiang Z., Schaefer S., Panozzo D.: Simplicial complex augmentation framework for bijective maps. *ACM Transactions Graphics 36*, 6 (Nov. 2017). https://doi.org/10.1145/3130800.3130895

[JT03] Jia J., Tang C.-K.: Image repairing: Robust image synthesis by adaptive nd tensor voting. In *CVPR'03: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Madison, Wisconsin* (USA, 2003), IEEE Computer Society, pp. 643–650.

[JTPSH15] Jakob W., Tarini M., Panozzo D., Sorkine-Hornung O.: Instant field-aligned meshes. *ACM Transactions Graphics 34*, 6 (Oct. 2015). https://doi.org/10.1145/2816795.2818078

[KKDK12] Kopf J., Kienzle W., Drucker S., Kang S. B.: Quality prediction for image completion. *ACM Transactions Graphics 31*, 6 (Nov. 2012). https://doi.org/10.1145/2366145.2366150

[KSSH14] Köhler R., Schuler C., Schölkopf B., Harmeling S.: Mask-specific inpainting with deep neural networks. In *German Conference on Pattern Recognition* (2014), Springer, pp. 523–534.

[LKK*18] Li M., Kaufman D. M., Kim V. G., Solomon J., Sheffer A.: OptCuts: Joint optimization of surface cuts and parameterization. *ACM Transactions Graphics 37*, 6 (2018), 1–13.

[LRS*18] Liu G., Reda F. A., Shih K. J., Wang T.-C., Tao A., Catanzaro B.: Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)* (Munich, Germany, 2018).

[LSW*18] Liu G., Shih K. J., Wang T.-C., Reda F. A., Sapra K., Yu Z., Tao A., Catanzaro B.: Partial convolution based padding. In *arXiv preprint arXiv:1811.11718* (2018).

[LYNF18] Liu L., Ye C., Ni R., Fu X.-M.: Progressive parameterizations. *ACM Transactions Graphics 37*, 4 (July 2018). https://doi.org/10.1145/3197517.3201331

[LZX*08] Liu L., Zhang L., Xu Y., Gotsman C., Gortler S. J.: A local/global approach to mesh parameterization. In *SGP'08: Proceedings of the Symposium on Geometry Processing, Copenhagen, Denmark* (Goslar, DEU, 2008), Eurographics Association, pp. 1495–1504.

[MBBV15] Masci J., Boscaini D., Bronstein M. M., Vandergheynst P.: Geodesic convolutional neural networks on Riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops* (Santiago, Chile, December 2015).

[MCT21] Maggiordomo A., Cignoni P., Tarini M.: Texture defragmentation for photo-reconstructed 3D models. *Computer Graphics Forum, 40* (2021), 65–78.

[MMPM16] Moulon P., Monasse P., Perrot R., Marlet R.: OpenMVG: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition, 2016* (Cancun, Mexico, 2016), Springer, pp. 60–74.

[MPCT20] Maggiordomo A., Ponchio F., Cignoni P., Tarini M.: Real-world textured things: A repository of textured models generated with modern photo-reconstruction tools. *Computer Aided Geometric Design 83* (2020), 101943. https://doi.org/10.1016/j.cagd.2020.101943

[MPK09] Mohammed U., Prince S. J. D., Kautz J.: Visiolization: Generating novel facial images. *ACM Transactions Graphics 28*, 3 (July 2009). https://doi.org/10.1145/1531326.1531363

[MZ12] Myles A., Zorin D.: Global parametrization by incremental flattening. *ACM Transactions Graphics 31*, 4 (2012), 1–11.

[Opa19] Opara A.: More like this, please! texture synthesis & remixing from a single example. In *Conference on Procedural Content Generation for Games (Everything Procedural)* (Breda, Netherlands, 2019). Talk.

[PCCS11] Pietroni N., Corsini M., Cignoni P., Scopigno R.: An interactive local flattening operator to support digital investigations on artwork surfaces. *IEEE Transaction on Visualization and Computer Graphics (Proceedings of Visualization 2011, Providence, Rhode Island) 17, 12* (Dec. 2011), 1989–1996.

[PKD*16] PATHAK D., KRAHENBUHL P., DONAHUE J., DARRELL T., EFROS A. A.: Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Las Vegas, Nevada, 2016), pp. 2536–2544.

[PSK06] PAVIĆ D., SCHÖNEFELD V., KOBBELT L.: Interactive image completion with perspective correction. *The Visual Computer 22*, 9 (2006), 671–681.

[PTH*17] PORANNE R., TARINI M., HUBER S., PANOZZO D., SORKINE-HORNUNG O.: Autocuts: Simultaneous distortion and cut optimization for UV mapping. *ACM Transactions Graphics 36*, 6 (Nov. 2017). https://doi.org/10.1145/3130800.3130845

[PTW13] PAL K., TERRAS M., WEYRICH T.: Interactive exploration and flattening of deformed historical documents. *Computer Graphics Forum 32*, 2pt3 (2013), 327–334. https://doi.org/10.1111/cgf.12052

[RPB15] RONNEBERGER O., FISCHER P., BROX T.: U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015, Munich, Germany)*. LNCS. (vol. *9351*). Springer (2015), pp. 234–241. (available on arXiv:1505.04597 [cs.CV]).

[rs22] Capturing Reality. RealityCapture: A 3D reconstruction software. (2022). https://www.capturingreality.com/realitycapture

[RSN*14] REMONDINO F., SPERA M. G., NOCERINO E., MENNA F., NEX F.: State of the art in high density image matching. *The Photogrammetric Record 29*, 146 (2014), 144–166.

[RXYS15] REN J. S., XU L., YAN Q., SUN W.: Shepard convolutional neural networks. *Advances in Neural Information Processing Systems 28* (2015), 901–909.

[SCD*06] SEITZ S. M., CURLESS B., DIEBEL J., SCHARSTEIN D., SZELISKI R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (New York, New York, 2006), IEEE Computer Society, pp. 519–528.

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. In *SIGGRAPH'06: Proceedings of the ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), Association for Computing Machinery, pp. 605–613. https://doi.org/10.1145/1179352.1141930

[SKCA20] SAINT A., KACEM A., CHERENKOVA K., AOUADA D.: 3DBooSTeR: 3D body shape and texture recovery. In *Computer Vision – ECCV 2020 Workshops* (Cham, 2020), A. Bartoli and A. Fusiello (Eds.), Springer International Publishing, pp. 726–740.

[TSPD16] THONAT T., SHECHTMAN E., PARIS S., DRETTAKIS G.: Multi-view inpainting for image-based scene editing and rendering. In *Proceedings of the 2016 Fourth International Conference on 3D Vision (3DV)* (2016), pp. 351–359. https://doi.org/10.1109/3DV.2016.44

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612. https://doi.org/10.1109/TIP.2003.819861

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH'00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New Orleans, Louisiana, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 479–488. https://doi.org/10.1145/344779.345009

[XXC12] XIE J., XU L., CHEN E.: Image denoising and inpainting with deep neural networks. *Advances in Neural Information Processing Systems 25* (2012), 341–349.

[YLP*20] YANG Y., LIU S., PAN H., LIU Y., TONG X.: PFCNN: Convolutional neural networks on 3D surfaces using parallel frames. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Virtual* (June 2020).

[ZLG*18] ZHAO H., LI X., GE H., LEI N., ZHANG M., WANG X., GU X.: Conformal mesh parameterization using discrete Calabi flow. *Computer Aided Geometric Design 63* (2018), 96–108. https://doi.org/10.1016/j.cagd.2018.03.001

[ZLK*18] ZHOU B., LAPEDRIZA A., KHOSLA A., OLIVA A., TORRALBA A.: Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 40*, 6 (2018), 1452–1464. https://doi.org/10.1109/TPAMI.2017.2723009

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information

Video S1