Let There Be Color! Large-Scale Texturing of 3D Reconstructions

Michael Waechter, Nils Moehrle, and Michael Goesele

TU Darmstadt, Germany

Abstract. 3D reconstruction pipelines using structure-from-motion and multi-view stereo techniques are today able to reconstruct impressive, large-scale geometry models from images but do not yield textured results. Current texture creation methods are unable to handle the complexity and scale of these models. We therefore present the first comprehensive texturing framework for large-scale, real-world 3D reconstructions. Our method addresses most challenges occurring in such reconstructions: the large number of input images, their drastically varying properties such as image scale, (out-of-focus) blur, exposure variation, and occluders (*e.g.*, moving plants or pedestrians). Using the proposed technique, we are able to texture datasets that are several orders of magnitude larger and far more challenging than shown in related work.

1 Introduction

In the last decade, 3D reconstruction from images has made tremendous progress. Camera calibration is now possible even on Internet photo collections [20] and for city scale datasets [1]. There is a wealth of dense multi-view stereo reconstruction algorithms, some also scaling to city level [7,8]. Realism is strongly increasing: Most recently Shan *et al.* [18] presented large reconstructions which are hard to distinguish from the input images if rendered at low resolution. Looking at the output of state of the art reconstruction algorithms one notices, however, that color information is still encoded as per-vertex color and therefore coupled to mesh resolution. An important building block to make the reconstructed models a convincing experience for end users while keeping their size manageable is still missing: texture. Although textured models are common in the computer graphics context, texturing 3D reconstructions from images is very challenging due to illumination and exposure changes, non-rigid scene parts, unreconstructed occluding objects and image scales that may vary by several orders of magnitudes between close-up views and distant overview images.

So far, texture acquisition has not attracted nearly as much attention as geometry acquisition: Current benchmarks such as the Middlebury multi-view stereo benchmark [17] focus only on geometry and ignore appearance aspects. Furukawa *et al.* [8] produce and render point clouds with very limited resolution, which is especially apparent in close-ups. To texture the reconstructed geometry Frahm *et al.* [7] use the mean of all images that observe it which yields insufficient visual fidelity. Shan *et al.* [18] perform impressive work on estimating

D. Fleet et al. (Eds.): ECCV 2014, Part V, LNCS 8693, pp. 836–850, 2014.

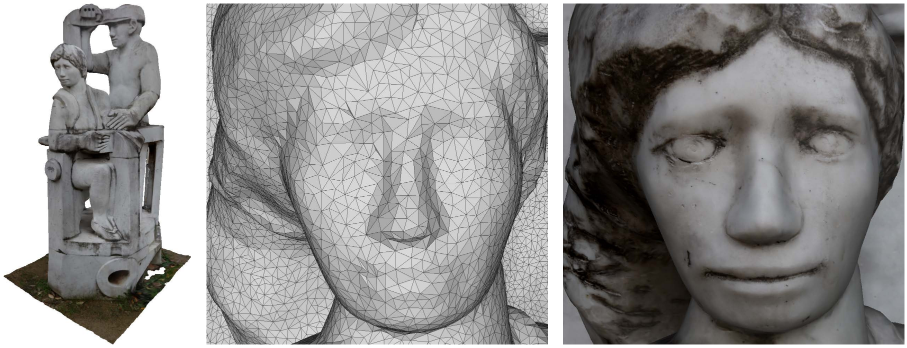

Fig. 1. *Left to right:* Automatically textured model reconstructed from a set of images, mesh close-up, and the same mesh rendered with texture

lighting parameters per input image and per-vertex reflectance parameters. Still, they use per-vertex colors and are therefore limited to the mesh resolution. Our texturing abilities seem to be lagging behind those of geometry reconstruction.

While there exists a significant body of work on texturing (Section 2 gives a detailed review) most authors focus on small, controlled datasets where the above challenges do not need to be taken into account. Prominent exceptions handle only specialized cases such as architectural scenes: Garcia-Dorado *et al.* [10] reconstruct and texture entire cities, but their method is specialized to the city setting as it uses a 2.5D scene representation (building outlines plus estimated elevation maps) and a sparse image set where each mesh face is visible in very few views. Also, they are restricted to regular block city structures with planar surfaces and treat buildings, ground, and building-ground transitions differently during texturing. Sinha *et al.* [19] texture large 3D models with planar surfaces (*i.e.*, buildings) that have been created interactively using cues from structure-from-motion on the input images. Since they only consider this planar case, they can optimize each surface independently. In addition, they rely on user interaction to mark occluding objects (*e.g.*, trees) in order to ignore them during texturing. Similarly, Tan *et al.* [22] propose an interactive texture mapping approach for building façades. Stamos and Allen [21] operate on geometry data acquired with time-of-flight scanners and therefore need to solve a different set of problems including the integration of range and image data.

We argue that texture reconstruction is vitally important for creating realistic models without increasing their geometric complexity. It should ideally be fully automatic even for large-scale, real-world datasets. This is challenging due to the properties of the input images as well as unavoidable imperfections in the reconstructed geometry. Finally, a practical method should be efficient enough to handle even large models in a reasonable time frame. In this paper we therefore present the first unified texturing approach that handles large, realistic datasets reconstructed from images with a structure-from-motion plus multi-view stereo pipeline. Our method fully automatically accounts for typical challenges inherent in this setting and is efficient enough to texture real-world models with hundreds of input images and tens of millions of triangles within less than two hours.

2 Related Work

Texturing a 3D model from multiple registered images is typically performed in a two step approach: First, one needs to select which view(s) should be used to texture each face yielding a preliminary texture. In the second step, this texture is optimized for consistency to avoid seams between adjacent texture patches.

View Selection. The literature can be divided into two main classes: Several approaches select and blend multiple views per face to achieve a consistent texture across patch borders [5,13]. In contrast, many others texture each face with exactly one view [9,10,15,23]. Sinha *et al.* [19] also select one view, but per texel instead of per face. Some authors [2,6] propose hybrid approaches that generally select a single view per face but blend close to texture patch borders.

Blending images causes problems in a multi-view stereo setting: First, if camera parameters or the reconstructed geometry are slightly inaccurate, texture patches may be misaligned at their borders, produce ghosting, and result in strongly visible seams. This occurs also if the geometric model has a relatively low resolution and does not perfectly represent the true object geometry. Second, in realistic multi-view stereo datasets we often observe a strong difference in image scale: The same face may cover less than one pixel in one view and several thousand in another. If these views are blended, distant views blur out details from close-ups. This can be alleviated by weighting the images to be blended [5] or by blending in frequency space [2,6], but either way blending entails a quality loss because the images are resampled into a common coordinate frame.

Callieri *et al.* [5] compute weights for blending as a product of masks indicating the suitability of input image pixels for texturing with respect to angle, proximity to the model, and proximity to depth discontinuities. They do however not compute real textures but suggest the use of vertex colors in combination with mesh subdivision. This contradicts the purpose of textures (high resolution at low data cost) and is not feasible for large datasets or high-resolution images. Similarly, Grammatikopoulos *et al.* [13] blend pixels based on angle and proximity to the model. A view-dependent texturing approach that also blends views is Buehler *et al.*'s Lumigraph [4]. In contrast to the Lumigraph we construct a global textured model and abstain from blending.

Lempitsky and Ivanov [15] select a single view per face based on a pairwise Markov random field. Their data term judges the quality of views for texturing while their smoothness term models the severity of seams between texture patches. Based on this Allène *et al.* [2] and Gal *et al.* [9] proposed data terms that incorporate additional effects compared to the basic data term. Since these methods form the base for our technique, we describe them in Section 3.

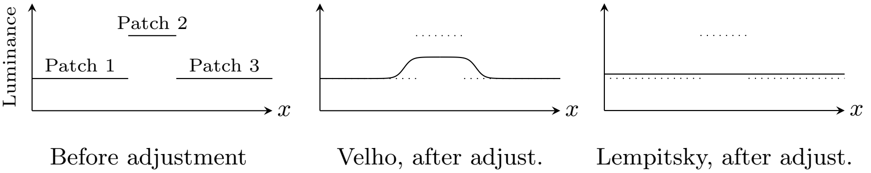

Fig. 2. Color adjustment (here in 1D). *Left:* Patch 2 is lighter than Patch 1 and 3, *e.g.* due to different exposure. *Center:* Velho and Sossai [23] let the luminance transition smoothly towards the seam's mean. *Right:* Lempitsky and Ivanov [15] adjust globally.

Color Adjustment. After view selection the resulting texture patches may have strong color discontinuities due to exposure and illumination differences or even different camera response curves. Thus, adjacent texture patches need to be photometrically adjusted so that their seams become less noticeable.

This can be done either locally or globally. Velho and Sossai [23] (Figure 2 (center)) adjust locally by setting the color at a seam to the mean of the left and right patch. They then use heat diffusion to achieve a smooth color transition towards this mean, which noticeably lightens Patches 1 and 3 at their borders. In contrast, Lempitsky and Ivanov [15] compute globally optimal luminance correction terms that are added to the vertex luminances subject to two intuitive constraints: After adjustment luminance differences at seams should be small and the derivative of adjustments within a texture patch should be small. This allows for a correction where Patch 2 is adjusted to the same level as Patch 1 and 3 (Figure 2 (right)) without visible meso- or large-scale luminance changes.

3 Assumptions and Base Method

Our method takes as input a set of (typically several hundred) images of a scene that were registered using structure-from-motion [1,20]. Based on this the scene geometry is reconstructed using any current multi-view stereo technique (*e.g.*, [7,8]) and further post-processed yielding a good (but not necessarily perfect) quality triangular mesh. This setting ensures that the images are registered against the 3D reconstruction but also yields some inherent challenges: The structure-from-motion camera parameters may not be perfectly accurate and the reconstructed geometry may not represent the underlying scene perfectly. Furthermore, the input images may exhibit strong illumination, exposure, and scale differences and contain unreconstructed occluders such as pedestrians.

We now give an overview over how Lempitsky and Ivanov [15] and some related algorithms work since our approach is based on their work. Section 4 describes the key changes made in our approach to handle the above challenges.

The initial step in the pipeline is to determine the visibility of faces in the input images. Lempitsky and Ivanov then compute a labeling l that assigns a view l_i to be used as texture for each mesh face F_i using a pairwise Markov random field energy formulation (we use a simpler notation here):

$$E(l) = \sum_{F_i \in \text{Faces}} E_{\text{data}}(F_i, l_i) + \sum_{(F_i, F_j) \in \text{Edges}} E_{\text{smooth}}(F_i, F_j, l_i, l_j) \tag{1}$$

The data term E_{data} prefers "good" views for texturing a face. The smoothness term E_{smooth} minimizes seam (*i.e.*, edges between faces textured with different images) visibility. $E(l)$ is minimized with graph cuts and alpha expansion [3].

As data term the base method uses the angle between viewing direction and face normal. This is, however, insufficient for our datasets as it chooses images regardless of their proximity to the object, their resolution or their out-of-focus blur. Allène *et al.* [2] project a face into a view and use the projection's size as data term. This accounts for view proximity, angle and image resolution. Similar to this are the Lumigraph's [4] view blending weights, which account for the very same effects. However, neither Allène nor the Lumigraph account for out-of-focus blur: In a close-up the faces closest to the camera have a large projection area and are preferred by Allène's data term or the Lumigraph weights but they may not be in focus and lead to a blurry texture. Thus, Gal *et al.* [9] use the gradient magnitude of the image integrated over the face's projection. This term is large if the projection area is large (close, orthogonal images with a high resolution) or the gradient magnitude is large (in-focus images).

Gal *et al.* also introduce two additional degrees of freedom into the data term: They allow images to be translated by up to 64 pixels in x- or y-direction to minimize seam visibility. While this may improve the alignment of neighboring patches, we abstain from this because it only considers seam visibility and does not explain the input data. In a rendering of such a model a texture patch would have an offset compared to its source image. Also, these additional degrees of freedom may increase the computational complexity such that the optimization becomes infeasible for realistic dataset sizes.

Lempitsky and Ivanov's smoothness term is the difference between the texture to a seam's left and right side integrated over the seam. This should prefer seams in regions where cameras are accurately registered or where misalignments are unnoticeable because the texture is smooth. We found, that computation of the seam error integrals is a computational bottleneck and cannot be precomputed due to the prohibitively large number of combinations. Furthermore, it favors distant or low-resolution views since a blurry texture produces smaller seam errors, an issue that does not occur in their datasets.

After obtaining a labeling from minimizing Equation 1, the patch colors are adjusted as follows: First, it must be ensured that each mesh vertex belongs to exactly one texture patch. Therefore each vertex on a seam is duplicated into two vertices: Vertex v_{left} belonging to the patch to the left and v_{right} belonging to the patch to the right of the seam.[1] Now each vertex v has a unique color f_v before adjustment. Then, an additive correction g_v is computed for each vertex,

[1] In the following, we only consider the case where seam vertices belong to $n = 2$ patches. For $n > 2$ we create n copies of the vertex and optimize all pairs of those copies jointly, yielding a correction factor per vertex and patch.

by minimizing the following expression (we use a simpler notation for clarity):

$$\underset{\mathbf{g}}{\operatorname{argmin}} \sum_{\substack{v \text{ (split into} \\ v_{\text{left}} \text{ and } v_{\text{right}}) \\ \text{lies on a seam}}} \left(f_{v_{\text{left}}} + g_{v_{\text{left}}} - (f_{v_{\text{right}}} + g_{v_{\text{right}}})\right)^2 + \frac{1}{\lambda} \sum_{\substack{v_i, v_j \text{ are ad-} \\ \text{jacent and in} \\ \text{the same patch}}} \left(g_{v_i} - g_{v_j}\right)^2 \quad (2)$$

The first term ensures that the adjusted color to a seam's left $(f_{v_{\text{left}}} + g_{v_{\text{left}}})$ and its right $(f_{v_{\text{right}}} + g_{v_{\text{right}}})$ are as similar as possible. The second term minimizes adjustment differences between adjacent vertices within the same texture patch. This favors adjustments that are as gradual as possible within a texture patch. After finding optimal g_v for all vertices the corrections for each texel are interpolated from the g_v of its surrounding vertices using barycentric coordinates. Finally, the corrections are added to the input images, the texture patches are packed into texture atlases, and texture coordinates are attached to the vertices.

4 Large-Scale Texturing Approach

Following the base method we now explain our approach, focusing on the key novel aspects introduced to handle the challenges of realistic 3D reconstructions.

4.1 Preprocessing

We determine face visibility for all combinations of views and faces by first performing back face and view frustum culling, before actually checking for occlusions. For the latter we employ a standard library [11] to compute intersections between the input model and viewing rays from the camera center to the triangle under question. This is more accurate than using rendering as, *e.g.*, done by Callieri *et al.* [5], and has no relevant negative impact on performance. We then precompute the data terms for Equation 1 for all remaining face-view combinations since they are used multiple times during optimization, remain constant, and fit into memory (the table has $\mathcal{O}(\#\text{faces} \cdot \#\text{views})$ entries and is very sparse).

4.2 View Selection

Our view selection follows the structure of the base algorithm, *i.e.*, we obtain a labeling such as the one in Figure 3 (left) by optimizing Equation 1 with graph cuts and alpha expansion [3]. We, however, replace the base algorithm's data and smoothness terms and augment the data term with a photo-consistency check.

Data Term. For the reasons described in Section 3 we choose Gal *et al.*'s [9] data term $E_{\text{data}} = -\int_{\phi(F_i, l_i)} \|\nabla(I_{l_i}(p))\|_2 \, \mathrm{d}p$. We compute the gradient magnitude $\|\nabla(I_{l_i})\|_2$ of the image into which face F_i is projected with a Sobel operator and sum over all pixels of the gradient magnitude image within F_i's projection $\phi(F_i, l_i)$. If the projection contains less than one pixel we sample the gradient magnitude at the projection's centroid and multiply it with the projection area.

The data term's preference for large gradient magnitudes entails an important problem that Gal *et al.* do not account for because it does not occur in their controlled datasets: If a view contains an occluder such as a pedestrian that has not been reconstructed and can thus not be detected by the visibility check, this view should not be chosen for texturing the faces behind that occluder. Unfortunately this happens frequently with the gradient magnitude term (*e.g.* in Figure 9) because occluders such as pedestrians or leaves often feature a larger gradient magnitude than their background, *e.g.*, a relatively uniform wall. We therefore introduce an additional step to ensure photo-consistency of the texture.

Photo-Consistency Check. We assume that for a specific face the majority of views see the correct color. A minority may see wrong colors (*i.e.*, an occluder) and those are much less correlated. Based on this assumption Sinha *et al.* [19] and Grammatikopoulos *et al.* [13] use mean or median colors to reject inconsistent views. This is not sufficient, as we show in Section 5. Instead we use a slightly modified mean-shift algorithm consisting of the following steps:

1. Compute the face projection's mean color c_i for each view i in which the face is visible.
2. Declare all views seeing the face as inliers.
3. Compute mean μ and covariance matrix Σ of all inliers' mean color c_i.
4. Evaluate a multi-variate Gaussian function $\exp\left(-\frac{1}{2}(c_i - \mu)^\mathsf{T} \Sigma^{-1}(c_i - \mu)\right)$ for each view in which the face is visible.
5. Clear the inlier list and insert all views whose function value is above a threshold (we use $6 \cdot 10^{-3}$).
6. Repeat 3.–5. for 10 iterations or until all entries of Σ drop below 10^{-5}, the inversion of Σ becomes unstable, or the number of inliers drops below 4.

We obtain a list of photo-consistent views for each face and multiply a penalty on all other views' data terms to prevent their selection.

Note, that using the median instead of the mean does not work on very small query sets because for 3D vectors the marginal median is usually not a member of the query set so that too many views are purged. Not shifting the mean does not work in practice because the initial mean is often quite far away from the inliers' mean (see Section 5 for an example). Sinha *et al.* [19] therefore additionally allow the user to interactively mark regions that should not be used for texturing, a step which we explicitly want to avoid.

Smoothness Term. As discussed above, Lempitsky and Ivanov's smoothness term is a major performance bottleneck and counteracts our data term's preference for close-up views. We propose a smoothness term based on the Potts model: $E_{\text{smooth}} = [l_i \neq l_j]$ ($[\cdot]$ is the Iverson bracket). This also prefers compact patches without favoring distant views and is extremely fast to compute.

4.3 Color Adjustment

Models obtained from the view selection phase (*e.g.*, Figure 3 (right)) contain many color discontinuities between patches. These need to be adjusted to minimize seam visibility. We use an improved version of the base method's global adjustment, followed by a local adjustment with Poisson editing [16].

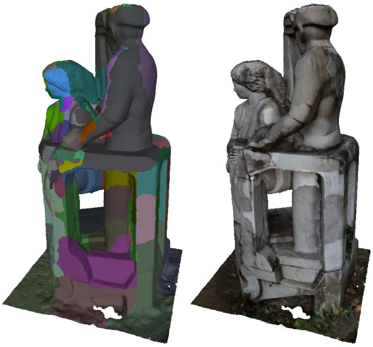

Fig. 3. *Left:* A mesh's labeling. Each color represents a different label, *i.e.* input image. *Right:* The textured result with visible luminance differences between patches.

Global Adjustment. A serious problem with Lempitsky and Ivanov's color adjustment is that $f_{v_{\text{left}}}$ and $f_{v_{\text{right}}}$ in Equation 2 are only evaluated at a single location: the vertex v's projection into the two images adjacent to the seam. If there are even small registration errors (which there always are), both projections do not correspond to exactly the same spot on the real object. Also, if both images have a different scale the looked up pixels span a different footprint in 3D. This may be irrelevant in controlled lab datasets, but in realistic multi-view stereo datasets the lookups from effectively different points or footprints mislead the global adjustment and produce artifacts.

Color Lookup Support Region. We alleviate this problem by not only looking up a vertex' color value at the vertex projection but along all adjacent seam edges, as illustrated by Figure 4: Vertex v_1 is on the seam between the red and the blue patch. We evaluate its color in the red patch, $f_{v_{1,\text{red}}}$, by averaging color samples from the red image along the two edges $\overline{v_0 v_1}$ and $\overline{v_1 v_2}$. On each edge we draw twice as many samples as the edge length in pixels. When averaging the samples we weight them according to Figure 4 (right): The sample weight is 1 on v_1 and decreases linearly with a sample's distance to v_1. (The reasoning behind this is that after optimization of Equation 2 the computed correction $g_{v_{1,\text{red}}}$ is applied to the texels using barycentric coordinates. Along the seam the barycentric coordinates form the transition from 1 to 0.) We obtain average colors for

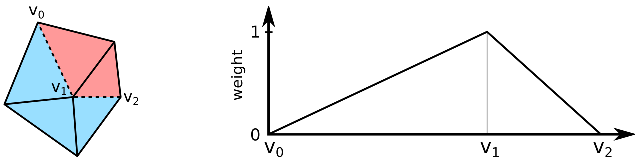

Fig. 4. *Left:* A mesh. Vertex v_1 is adjacent to both texture patches (red and blue). Its color is looked up as a weighted average over samples on the edges $\overline{v_0 v_1}$ and $\overline{v_1 v_2}$. *Right:* Sample weights transition from 1 to 0 as distance to v_1 grows.

the edges $\overline{v_0 v_1}$ and $\overline{v_1 v_2}$, which we average weighted with the edge lengths to obtain $f_{v_{1,\mathrm{red}}}$. Similarly we obtain $f_{v_{1,\mathrm{blue}}}$ and insert both into Equation 2.

For optimization, Equation 2 can now be written in matrix form as

$$\|\mathbf{A}\mathbf{g} - \mathbf{f}\|_2^2 + \|\mathbf{\Gamma}\mathbf{g}\|_2^2 = \mathbf{g}^\mathsf{T}(\mathbf{A}^\mathsf{T}\mathbf{A} + \mathbf{\Gamma}^\mathsf{T}\mathbf{\Gamma})\mathbf{g} - 2\mathbf{f}^\mathsf{T}\mathbf{A}\mathbf{g} + \mathbf{f}^\mathsf{T}\mathbf{f} \qquad (3)$$

$\mathbf{f}$ is a vector with the stacked $f_{v_{\mathrm{left}}} - f_{v_{\mathrm{right}}}$ from Equation 2. $\mathbf{A}$ and $\mathbf{\Gamma}$ are sparse matrices containing ±1 entries to pick the correct $g_{v_{\mathrm{left}}}$, $g_{v_{\mathrm{right}}}$, g_{v_i} and g_{v_j} from $\mathbf{g}$. Equation 3 is a quadratic form in $\mathbf{g}$ and $\mathbf{A}^\mathsf{T}\mathbf{A} + \mathbf{\Gamma}^\mathsf{T}\mathbf{\Gamma}$ is very sparse, symmetric, and positive semidefinite (because $\forall \mathbf{z}$: $\mathbf{z}^\mathsf{T}(\mathbf{A}^\mathsf{T}\mathbf{A} + \mathbf{\Gamma}^\mathsf{T}\mathbf{\Gamma})\mathbf{z} = \|\mathbf{A}\mathbf{z}\|_2^2 + \|\mathbf{\Gamma}\mathbf{z}\|_2^2 \geq 0$). We minimize it with respect to $\mathbf{g}$ with Eigen's [14] conjugate gradient (CG) implementation and stop CG when $\|\mathbf{r}\|_2 / \|\mathbf{A}^\mathsf{T}\mathbf{f}\|_2 < 10^{-5}$ ($\mathbf{r}$ is the residual), which typically requires < 200 iterations even for large datasets. Due to automatic white balancing, different camera response curves and different light colors (noon vs. sunset vs. artificial light) between images it is not sufficient to only optimize the luminance channel. We thus optimize all three channels in parallel.

Poisson Editing. Even with the above support regions Lempitsky and Ivanov's global adjustment does not eliminate all visible seams, see an example in Figure 11 (bottom row, center). Thus, subsequent to global adjustment we additionally perform local Poisson image editing [16]. Gal *et al.* [9] do this as well, but in a way that makes the computation prohibitively expensive: They Poisson edit complete texture patches, which results in huge linear systems (with $>10^7$ variables for the largest patches in our datasets).

We thus restrict the Poisson editing of a patch to a 20 pixel wide border strip (shown in light blue in Figure 5). We use this strip's outer rim (Fig. 5, dark blue) and inner rim (Fig. 5, red) as Poisson equation boundary conditions: We fix each outer rim pixel's value to the mean of the pixel's color in the image assigned to the patch and the image assigned to the neighboring patch. Each inner rim pixel's value is fixed to its current color. If the patch is too small, we omit the inner rim. The Poisson equation's guidance field is the strip's Laplacian.

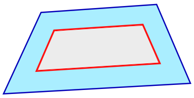

Fig. 5. A texture patch has a border strip (light blue) with an outer (dark blue) and inner rim (red)

For all patches we solve the resulting linear systems in parallel with Eigen's [14] SparseLU factorization. For each patch we only compute the factorization once and reuse it for all color channels because the system's matrix stays the same. Adjusting only strips is considerably more time and memory efficient than adjusting whole patches. Note, that this local adjustment is a much weaker form of the case shown in Figure 2 (center) because patch colors have been adjusted globally beforehand. Also note, that we do not mix two images' Laplacians and therefore still avoid blending.

dataset	# views	# mesh faces	image resolution
Statue	334	4.9 million	5616×3744
City Wall	561	8.2 million	2000×1500
Castle Ruin	287	20.3 million	5616×3744

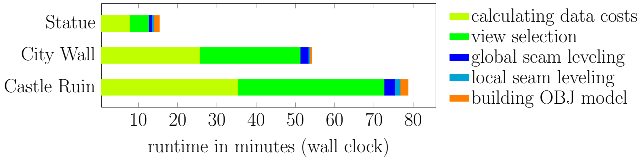

Fig. 6. Summary of the three datasets used in the evaluation and runtime of the individual parts of the algorithm

5 Evaluation

We now evaluate the proposed approach using three datasets (Statue, City Wall, Castle Ruin) of varying complexity, see Figure 6 for a summary of their properties. Even our largest dataset (Castle Ruin, Figure 7) can be textured within less than 80 min on a modern machine with two 8 core Xeon E5-2650v2 CPUs and 128 GB of memory. Main computational bottlenecks are the data cost computation whose complexity is linear in the number of views and the number of pixels per view, and the graph cut-based view selection which is linear in the number of mesh faces and the number of views. When altering the above datasets properties, *e.g.*, by simplifying the mesh, we found that the theoretical complexities fit closely in practice. The data term computation is already fully parallelized but the graph cut cannot be parallelized easily due to the graph's irregular structure. Still, this is in stark contrast to other methods which may yield theoretically optimal results while requiring a tremendous amount of computation. *E.g.*, Goldluecke *et al.* compute for several hours (partially on the GPU) to texture small models with 36 views within a super-resolution framework [12].

In the following sections, we evaluate the individual components of our approach and compare them to related work.

Data Term and Photo-Consistency Check. As argued previously, the data term must take the content of the candidate images and not only their geometric configuration into account. Allène *et al.*'s data term [2] computes a face's projection area and thus selects the image in the center of Figure 8 for texturing the statue's arm, even though this image is not focused on the arm but on its background. Gal *et al.*'s data term [9] favors instead large gradient magnitudes and large projection areas. It thus uses a different image for the arm, yielding a much crisper result as shown in Figure 8 (right).

A failure case of Gal's data term are datasets with little surface texture detail. For example the Middlebury Dino [17] is based on a uniform white plaster statue, all views have the same distance to the scene center and out-of-focus blur is not an issue. Thus, Gal's data term accounts for effects that do not occur in this

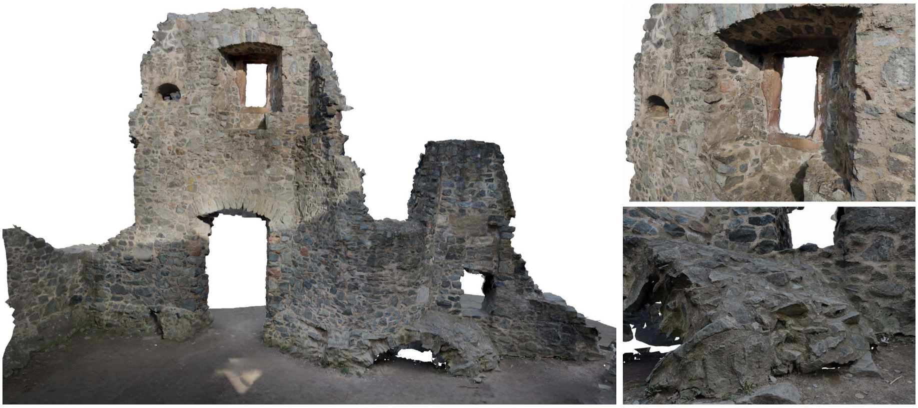

Fig. 7. Some texturing results of the Castle Ruin dataset

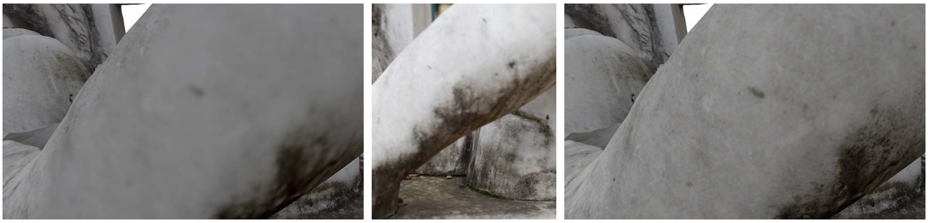

Fig. 8. *Left:* Statue's arm textured with Allène's [2] data term. *Center:* Detail from the image used for texturing the arm in the left image, exhibiting out-of-focus blur. *Right:* The arm textured with Gal's [9] data term. (Best viewed on screen.)

dataset. We found that the data term instead overfits to artifacts such as shadows and the color adjustment is incapable of fixing the resulting artifacts.

Another drawback of Gal's data term is, that it frequently uses occluders to texture the background (as shown in Figure 9 (top right)) if the occluder contains more high-frequency image content than the correct texture.

In Figure 9 (bottom left) we show the photo-consistency check at work for one exemplary face. This face has many outliers (red), *i.e.* views seeing an occluder instead of the correct face content. The gray ellipse marks all views that our check classifies as inliers. Only one view is misclassified. Note the black path taken by the shifted mean: The starting point on the path's bottom left is the algorithm's initial mean. Sinha *et al.* [19] and Grammatikopoulos *et al.* [13] use it without iterating in mean-shift fashion. It is, however, far off from the true inlier mean and if we used a fixed window around it many views would be misclassified. If we used all views' covariance matrix instead of a fixed window all views except for the bottommost data point would be classified as inliers. Our approach only misclassifies the one outlier closest to the inlier set and is able to remove the pedestrian (see Figure 9 (bottom right)).

Our proposed photo-consistency check may fail, *e.g.* if a face is seen by very few views. In this case there is little image evidence for what is an inlier or an

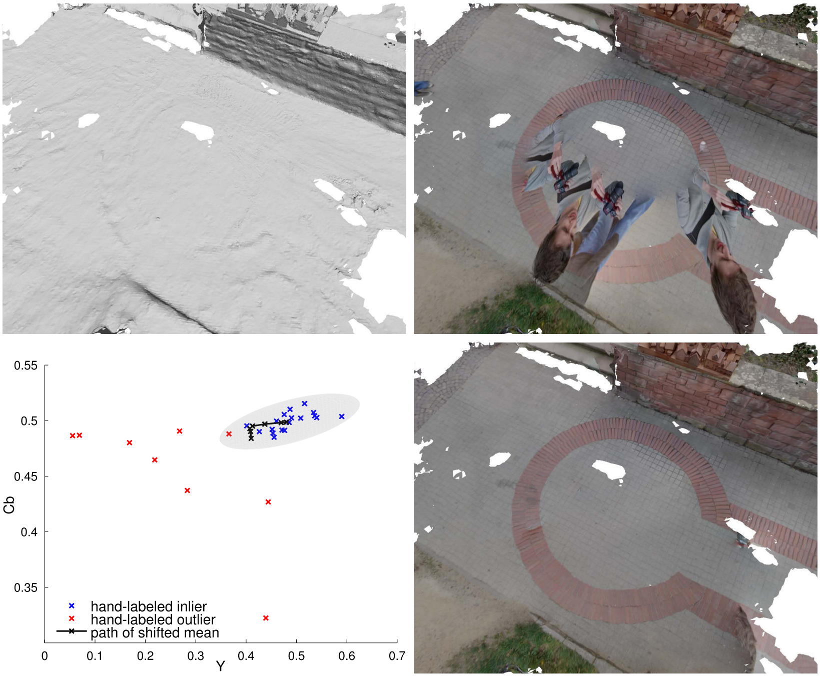

Fig. 9. *Top left:* Input mesh. *Top right:* Pedestrian used as texture for the ground. *Bottom left:* Photo-consistency check visualization for a single face. Red and blue crosses are the face's mean color (here we show only the Y and C_b component) in different images. *Bottom right:* Mesh textured with photo-consistency check.

outlier and the classification becomes inaccurate. However, in our experience this is often fixed by the view selection's smoothness term: If view X is erroneously classified as outlier for face A, but not for face B, it may still be optimal to choose it for texturing both A and B if the data penalty for face A is smaller than the avoided smoothness penalty between A and B. This can also fail, especially in border cases where there are not enough neighbor faces that impose a smoothness penalty, such as in the bottom right corner of the bottom right image in Figure 9.

Smoothness Term. Lempitsky and Ivanov use the error on a seam integrated along the seam as smoothness term for the view selection. The result is a model where blurry texture patches selected from distant views occur frequently since they make seams less noticeable (see Figure 10 (left)). Using the Potts model instead yields a considerably sharper result (see Figure 10 (right)).

Color Adjustment. The global color adjustment used in the base method [15] operates on a per-vertex base. It fails if a seam pixel does not project onto the same 3D position in the images on both sides of its seam. This can occur for various reasons: Camera misalignment, 3D reconstruction inaccuracies or a different 3D footprint size of the looked up pixels if the projected resolution differs

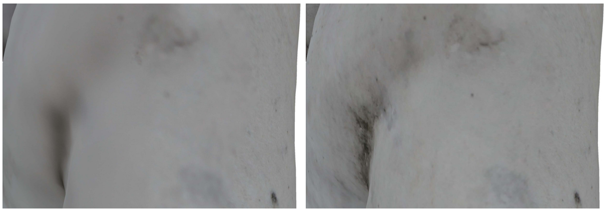

Fig. 10. *Left:* Statue's shoulder with Lempitsky and Ivanov's smoothness term. *Right:* Resulting model with the Potts model smoothness term. (Best viewed on screen.)

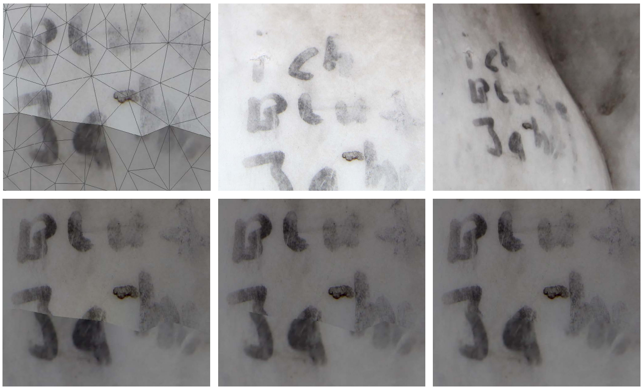

Fig. 11. *Top row:* Unadjusted mesh with wireframe, detail of input image used for upper patch, and detail of input image used for lower patch. *Bottom row:* Artifact from global color adjustment caused by texture misalignment, result of global adjustment with support region, and result after Poisson editing.

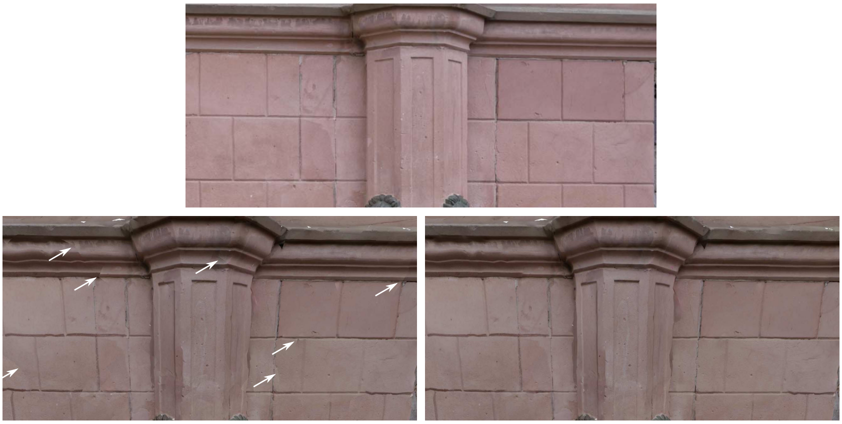

Fig. 12. *Top row:* One of the input images. *Bottom row:* Reconstructed City Wall color adjusted without support region and without Poisson editing, and the same reconstruction adjusted with support region and Poisson editing. (Best viewed on screen.)

in both images. In Figure 11 (top left) the vertex on the right side of the letter 'J' projects into the letter itself to the seam's top and into the lighter background to the seam's bottom. The result after global adjustment (bottom left) exhibits a strong color artifact. We alleviate this using support regions during sampling (bottom center). The remaining visible seams are fixed with Poisson editing as shown in Figure 11 (bottom right). Poisson editing does obviously not correct texture alignment problems but disguises most seams well so that they are virtually invisible unless the model is inspected from a very close distance.

These color adjustment problems in Lempitsky and Ivanov's method occur frequently when mesh resolution is much lower than image resolution (which is the key application of texturing). Vertex projections' colors are in this case much less representative for the vertices' surroundings. Figure 12 shows such an example: The result without support region and Poisson editing (bottom left) exhibits several strong color artifacts (white arrows) when compared to a photograph of the area (top). The result with support region and Poisson editing (bottom right) significantly improves quality.

6 Conclusions and Future Work

Applying textures to reconstructed models is one of the keys to realism. Surprisingly, this topic has been neglected in recent years and the state of the art is still to reconstruct models with per-vertex colors. We therefore presented the first comprehensive texturing framework for large 3D reconstructions with registered images. Based on existing work, we make several key changes that have a large impact: Large-scale, real-world geometry reconstructions can now be enriched with high-quality textures which will significantly increase the realism of reconstructed models. Typical effects occurring frequently in these datasets, namely inaccurate camera parameters and geometry, lighting and exposure variation, image scale variation, out-of-focus blur and unreconstructed occluders are handled automatically and efficiently. In fact, we can texture meshes with more than 20 million faces using close to 300 images in less than 80 minutes.

Avenues for future work include improving the efficiency of our approach and accelerating the computational bottlenecks. In addition, we plan to parallelize on a coarser scale to texture huge scenes on a compute cluster in the spirit of Agarwal *et al.*'s [1] structure-from-motion system.

Acknowledgements. We are grateful for financial support by the Intel Visual Computing Institute through the project *RealityScan*. Also, we thank Victor Lempitsky and Denis Ivanov for providing the code for their paper.

Source Code. This project's source code can be downloaded from the project web page www.gris.informatik.tu-darmstadt.de/projects/mvs-texturing.

References

1. Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building Rome in a day. In: ICCV (2009)

2. Allène, C., Pons, J.P., Keriven, R.: Seamless image-based texture atlases using multi-band blending. In: ICPR (2008)
3. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. PAMI 23 (2001)
4. Buehler, C., Bosse, M., McMillan, L., Gortler, S., Cohen, M.: Unstructured lumigraph rendering. In: SIGGRAPH (2001)
5. Callieri, M., Cignoni, P., Corsini, M., Scopigno, R.: Masked photo blending: Mapping dense photographic dataset on high-resolution sampled 3D models. Computers & Graphics 32 (2008)
6. Chen, Z., Zhou, J., Chen, Y., Wang, G.: 3D texture mapping in multi-view reconstruction. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Fowlkes, C., Wang, S., Choi, M.-H., Mantler, S., Schulze, J., Acevedo, D., Mueller, K., Papka, M. (eds.) ISVC 2012, Part I. LNCS, vol. 7431, pp. 359–371. Springer, Heidelberg (2012)
7. Frahm, J.-M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., Pollefeys, M.: Building rome on a cloudless day. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part IV. LNCS, vol. 6314, pp. 368–381. Springer, Heidelberg (2010)
8. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Towards Internet-scale multi-view stereo. In: CVPR (2010)
9. Gal, R., Wexler, Y., Ofek, E., Hoppe, H., Cohen-Or, D.: Seamless montage for texturing models. Computer Graphics Forum 29 (2010)
10. Garcia-Dorado, I., Demir, I., Aliaga, D.G.: Automatic urban modeling using volumetric reconstruction with surface graph cuts. Computers & Graphics 37 (2013)
11. Geva, A.: ColDet 3D collision detection, http://sourceforge.net/projects/coldet
12. Goldluecke, B., Aubry, M., Kolev, K., Cremers, D.: A super-resolution framework for high-accuracy multiview reconstruction. IJCV (2013)
13. Grammatikopoulos, L., Kalisperakis, I., Karras, G., Petsa, E.: Automatic multi-view texture mapping of 3D surface projections. In: 3D-ARCH (2007)
14. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010), http://eigen.tuxfamily.org
15. Lempitsky, V., Ivanov, D.: Seamless mosaicing of image-based texture maps. In: CVPR (2007)
16. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Transactions on Graphics 22(3) (2003)
17. Seitz, S.M., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: CVPR (2006)
18. Shan, Q., Adams, R., Curless, B., Furukawa, Y., Seitz, S.M.: The visual Turing test for scene reconstruction. In: 3DV (2013)
19. Sinha, S.N., Steedly, D., Szeliski, R., Agrawala, M., Pollefeys, M.: Interactive 3D architectural modeling from unordered photo collections. In: SIGGRAPH Asia (2008)
20. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: Exploring photo collections in 3D. In: SIGGRAPH (2006)
21. Stamos, I., Allen, P.K.: Geometry and texture recovery of scenes of large scale. Computer Vision and Image Understanding 88(2) (2002)
22. Tan, Y.K.A., Kwoh, L.K., Ong, S.H.: Large scale texture mapping of building facades. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37 (2008)
23. Velho, L., Sossai Jr., J.: Projective texture atlas construction for 3D photography. The Visual Computer 23 (2007)