Masked Photo Blending: mapping dense photographic dataset on high-resolution sampled 3D models

M. Callieri, [*] P. Cignoni, [*] M. Corsini, [*] R. Scopigno [*]

Abstract

The technological advance of sensors is producing an exponential size growth of the data coming from 3D scanning and digital photography. The production of digital 3D models consisting of tens or even hundreds of millions of triangles is quite easy nowadays; at the same time, using high-resolution digital cameras it is also straightforward to produce a set of pictures of the same real object totalling more than 50M Pixel.

The problem is how to manage all this data to produce 3D models that could fit the interactive rendering constraints. A common approach is to go for mesh parametrization and texture synthesis, but finding a parametrization for such large meshes and managing such large textures can be prohibitive. Moreover, digital photo sampling produces highly redundant data; this redundancy should be eliminated while mapping to the 3D model but, at the same time, should also be efficiently used to improve the sampled data coherence and the appearance representation accuracy.

In this paper we present an approach where a multivariate blending function weights all the available pixel data with respect to geometric, topological and colorimetric criteria. The blending approach proposed is efficient, since it mostly works independently on each image, and can be easily extended to include other image quality estimators. The resulting weighted pixels are then selectively mapped on the geometry, preferably by adopting a multiresolution per-vertex encoding to make profitable use of all the data available and to avoid the texture size bottleneck. Some practical examples on complex datasets are presented.

Key words: 3D scanned models, image inverse projection, texture mapping, blending function, multiresolution encoding, interactive rendering, out-of-core processing.

1. Introduction

The HW/SW improvement of digital photography and 3D scanning makes it possible to acquire very dense sampling of both geometric and optical surface properties of real objects. Modern 3D scanning devices can sample a surface with a sampling rate as small as 0.25 millimeter. Producing very detailed 3D models (composed of millions of triangles) is now possible in a very short time with the new generation of software tools, which reduces significantly the range map registration effort (see an example in Pingi et al.(1)). A model such as the one shown in Figure 7 can be produced with 3 hours of scanning (200 range maps) and 1 day of post-processing (with a final model size of 12 million triangles). On more complex/large artifacts, 3D models in the order of hundreds million faces can be obtained.

The resolution of digital cameras' CCD improved also in an impressive manner; middle-quality digital cameras can provide resolutions of 8M pixels or more. A single object can thus be sampled with a few shots producing easily raw datasets of more than 50M pixel data. We use the term *raw* since a significant overlap exists in those pixel dataset, and the sampling quality varies a lot as well.

[*] Istituto di Scienza e Tecnologie dell'Informazione (ISTI) CNR, Pisa, Italy
[callieri|cignoni|corsini|scopigno]@isti.cnr.it

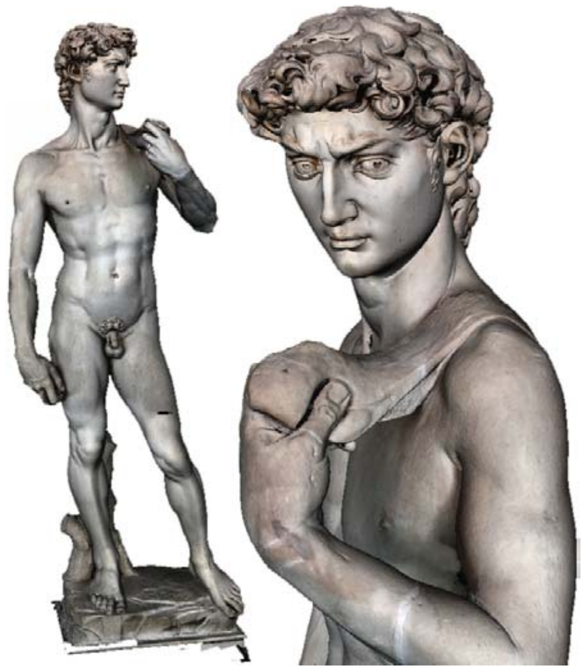

Fig. 1. A 56 million faces 3D model with color coming from a 520 Mpixel dataset (64 photos)

Different approaches exist for recomputing the inverse projection needed to solve the mapping from an uncalibrated photo to the 3D space (2; 3; 4; 5). This paper focuses on the subsequent phase in the scanning pipeline, i.e., how to process redundant pixel data and how to efficiently map such information on a high resolution 3D model.

One of the most common approaches is to build up a parametrization of the 3D mesh that fits well the pool of images available, and producing a new texture map, either by joining subregions of the input images or by resampling (6; 7; 8; 9; 3; 10; 11). Unfortunately, the management of very dense geometric and photographic sampling is very complicated. The texture-based approach is ideal when we have both low-to-moderate resolution meshes (50K-1M faces), usually produced by simplification or subsampling, and moderate pixel datasets (1M-5M). Moreover, multiresolution encoding is usually a must for huge meshes, and the adoption of multiresolution approach for texture-based representation of color (11) implies the need of a multi-resolution texture atlas, with the associated redundancy and increased space occupancy.

In this paper we present an approach where a multivariate blending function is proposed to weight all the available pixel data w.r.t. geometric, topological and colorimetric criteria (Section 3). This blending function (which operates in the image space) allows to mix weighted pixels from the various images. In this way we aim to maximize the use of the information contained in the input images, to correct incoherence between different photos while at the same time trying to reduce the blurr caused by simple blending approaches. Section 4 presents our weighting function defined as a combination of various metrics, each of them addressing a specific aspect of the images: view angle, distance from sensor, sharpness and so on. We discuss in Section 6 how to detect some defects in the source image dataset and how to correct them. The approach presented is efficient in time (since most of the computations are local to the single image) and highly extensible to other image-based evaluation heuristics.

We show in Section 5 how this function can be used to map the color information to the geometry, by choosing either a *texture-based approach* (which requires mesh parametrization) or by adopting a multiresolution *per-vertex encoding*. We describe in Section 7 how this process can be implemented out-of-core, to deal with huge geometric and image datasets. Finally, we present in Section 8 some results of our color mapping approach, relying on the per-vertex encoding. The latter has been preferred to texture mapping since it can be applied more easily to modern multiresolution approaches; moreover, it is more robust w.r.t. the need to manage incomplete, topologically dirty and complex meshes which are, unfortunately, the standard results of 3D scanning.

2. Previous research

Various approaches have been proposed for selecting the most correct color which has to be applied to each part of the 3D model. As stated in the introduction, the standard approach is to compute a texture by assembling subparts of the original input images. Additionally, some corrections can be applied to deal with incoherence in the borders between different images. An example of this approach is in Callieri et al. (10), where the mesh is covered using the most orthogonal image for each mesh portion, redundancy is used to correct color discontinuities in the boundary between images and then the correction is propagated to the whole texture space. Camera orthogonality is used also in Lensch et al. (3) to choose which part of the 3D model is to be mapped in which photo, the images are then fully blended, using the entire redundant area.

Conversely, other approaches generate color mapping without reassembling the original images; an example is in Yu et al. (12), where the texture is filled directly texel by texel with values coming from an inverse rendering process based on the original images. In the same fashion, we will use the input photos as a source to calculate the correct colors that will be used to fill texture map texels or to perform per-vertex coloring.

The idea of using a per-pixel blending function is not completely new; some other approaches that use a per-pixel weight have been proposed. Both Bernardini et al.(13) and Baumberg(14) use a weighted blending function to compute the texture color, but without exploiting all the potentiality of this method. Per-pixel weighted blending has also been used in image processing, as shown in Rankov et al.(15). However, we are focusing on the definition of an extensible and flexible framework for image blending. We can identify two main problems regarding color mapping from photographic data: difficulties related to large dataset management and difficulties related to image discordancy. We propose a way to weight and blend multiple images that is able to work with an arbitrary amount of input data. We examine benefits of the per-vertex color encoding, in terms of quality and compactness, while not excluding the algorithm applicability on standard texture mapping. Moreover, we show that with a complete and clean definition of various different weighting schemes, a good data arrangement and application rules, it is possible to obtain very good results in color mapping, overcoming most of the image-to-image incoherence.

3. The Masked Blending Function

The main idea is to create a blending function that operates in the image space, capable to mix data from the various images by weighting them w.r.t. the quality of each contribution. The objective of this function is to be able to work with photos taken in arbitrary conditions since, especially when working in the cultural heritage field, having a calibrated de-shaded and artifact-free photo dataset is not a viable option. To acquire models of cultural heritage artifacts it is often necessary to work on the field in museums or even outdoor, making it impossible to setup controlled lighting conditions.

The first problem is to detect those pixels in each photo which sample the object surface (since some pixels can depict the background), and for each of those which are the coordinates of the surface point which has been sampled. The projective mapping principle is well known. Since the photos follow the laws of perspective, if we know the camera parameters than we can determine if (and where) a point on the surface is mapped inside the image boundaries. An efficient manner to compute those pixel-to-surface correspondences is by rendering the 3D model with the same projection parameters of the given photo. In this way the resulting depth map can be used to discriminate, similarly to shadow mapping, if a point on the surface that projects inside the image is effectively visible from that point of view, or if it is occluded by other geometry. In this way it is possible to assign safely a pixel color, taken from that photo, to a point onto the surface.

But the real problem rises when the same surface point can take the color from many source images and thus from many different pixels. Since there is more than one candidate, it is not advisable to simply choose the color from a single image, ignoring the other values. But also doing a simple blend could not be a good choice, because not all sampled colors yields the same degree of quality. Since each pixel in the source images has a specific quality (that can be evaluated adopting various metrics), it is necessary to take into account this quality while blending. Our approach is therefore to define an extensible masked blending approach, where multiple local image evaluation heuristics can be defined to weight each single pixel. Once we have defined a suitable set of weighting masks, we are able to assign the most correct color to each point of the surface as a weighted mean of all possible sources. The problem is how to choose those local quality evaluation heuristics for our weighting method, which should be local and efficient both in time and space.

4. The weighting mask generation

The core of our blending function is the *weighting mask* that is generated for each image. The weight mask states the quality of each pixel and, consequently, how much it will contribute to the final color of the 3D points it maps on.

Various metrics can be applied to evaluate the quality of image pixels. Since we have not only the images but also a faithful geometric representation of the object, we can effectively use this information

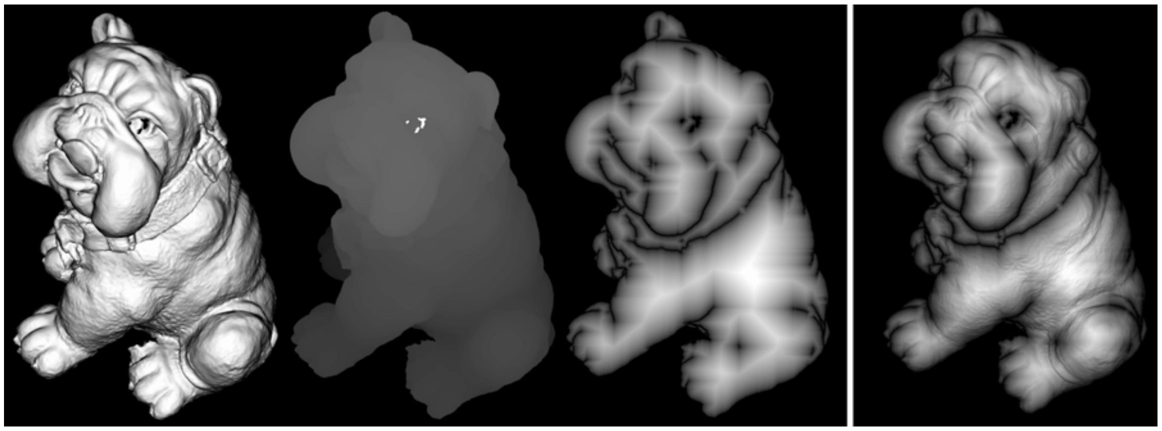

Fig. 2. An example of the core weighting masks. From left to right: Angle Mask, Depth Mask, Border Mask. Rightmost, all the masks combined in the final mask. Caveat: the contrast of the depth and border masks has been increased for enhanced readability.

to perform a higher quality evaluation. Each metric can take into account a different characteristic of the source image. For example, pixel quality can be measured based on camera orthogonality to the corresponding surface point, because orthogonal views appear less "stretched". But a pixel that is "good" for this particular metric can at the same time be considered "bad" by another metric that considers the distance from the sensor (the farther the surface, the less dense is the sampling available in the image). Following this approach, we define multiple metrics, and mix them all in a single measure. It is clear that, following this strategy, we can evaluate and mix an arbitrary number of metrics, making the system flexible and extensible (as we will show in Section 4.1).

Please note that, since we need a per-pixel blending, all the masks we compute have exactly the same resolution of the corresponding photo image. Once the camera parameters are known, it is possible to calculate each metric with a series of controlled renderings and video buffer processing. As introduced before, the depth buffer corresponding to each image is required to perform a correct color projection (visibility check); this buffer is calculated just once at the begin of the process and used as a base for the computation of the other masks.

The basic weights used in our system are:

Angle mask: this is the simplest quality evaluation for each image pixel, proportional to the angle between the viewing ray and the surface normal. Similar to Lambertian illumination, the quality is higher when the view direction is orthogonal to the surface and lower when glazing. The computation of this mask is quite straightforward: we simply render the 3D model under the image view, using a white lambertian material with a light source placed in the viewpoint; the rendered image is exactly the weight mask we need.

Depth mask: the weight of an image pixel is higher as the surface is closer to the camera. This is an approximation of the ratio pixel/surface (informally, how dense is the sampling encoded on this image part). Obviously this ratio depends also on view orientation, but this has already been taken in account when calculating the Angle mask. Calculation is, again, quite easy, using the same depth map that has been calculated at the begin of the process. Using directly the depth map would imply a linear quality decrease due to distance, that is incorrect. We produce better results using a squared distance, that represents the fact that pixel/surface ratio decreases following a quadratic law.

Border mask: this mask measures how far a pixel in the image is from both image borders and discontinuities in the depth map (silhouette borders). The farther we are from these borders, the higher the quality of the photo sampling. To compute this mask, the first step is to mark as point of zero value both the image borders and the depth discontinuities. The latter can be calculated by simply using a Sobel filter on the non-normalized depth map (every pixel contains the actual distance from the sensor, not the normalized [0-1] value): using such wider range makes the Sobel filter much more precise. For the rest of the pixels, the weight is calculated as the image-space (pixel-to-pixel) distance from the borders on the image space.

The composed weighting mask will be generated by assembling various metrics through multiplication. In this way we maintain for each mask its continuity and minima. Preservation of minima is very important to remove outliers. If a given pixel is considered "bad" according to a given metric, we should be sure that it has the least probability of being used. An example of the core masks described above and the final assembled weight mask is showed in Figure 2.

While the meaning of the first two weights is not difficult to understand, the third one is a bit more tricky. A discontinuity on the depth map indicates a silhouette point for that point of view; on the corresponding image those pixels probably blend data from two non-contiguous regions (or from the background), with problems of coherence in color, focusing and depth. Image borders are also problematic; it is well known that (due to lens imperfections) the quality of a photograph is minimal on the borders. Those two pixel regions should ideally not contribute to color determination; the quality improves the farther we go from those problematic areas. We use the depth map as a starting point because some discontinuities on the photo are more easily detectable when looking at the 3D models. Moreover, the image-space distance calculation ensures continuity of the weights, which is also a nice property.

Similar weighting metrics are also presented in a work by Pulli et al. (16); in particular the angle mask, identical in scope and calculation, and a feathering weight that is used by the authors to provide smooth transition between different images, but without taking in account internal silhouettes and quality degradation towards image borders.

4.1. *Additional masking weights*

The most interesting feature of this approach is its flexibility; the system works by evaluating multiple metrics and then mixing them in a single quality measure. In order to improve our system, we can add more metrics to address for specific dataset problems.

The masks described in the previous section are the most important ones, i.e., the *core masks* of the system. These masks are always applied to all datasets, because they evaluate measures that are related to photography in general and not to a specific dataset. In our experience, we found that there are other two recurring problems in the dataset we processed. For this reason, we implemented two additional masks: the *stencil* and *focus* mask.

Stencil mask: in some cases there is the need to exclude certain parts of the source image (e.g., people in front of a building, photographic artifacts). In those cases a simple stencil masking can be applied. Images we need to correct are (manually) marked with a specific color; the system will then assign zero weight to those pixels. In alternative, for each photo that requires a stencil mask, we provide another grey-scale image that basically contains the 0-1 mask that has to be assigned.

Another situation that often occurs when processing photos is to detect focusing problems. When doing a photographic coverage of an objects there is the need of being close to the object to obtain enough detail. This closeness can results in problems with the autofocus system; out of focus areas should not be used for mapping, otherwise we will obtain a blurred result. There can be two main problems:

– an entire image can result not perfectly in focus, i.e., when the camera cannot focus correctly because it is too close to the surface or because the object of interest is very small;
– just a portion of the image is out-of-focus, because the depth extent covered by the image is large, and some parts of the object falls beyond the focusing range.

These two problems can be resolved by adding a focus mask:

Focus Mask: to build this mask the focusing of each pixel of the image is evaluated in a way that is almost identical to the procedure used by digital cameras while performing the autofocus selection. Typically, the autofocus procedure set the focus distance by maximizing the sharpness of some reference points in the framed image. Therefore, we evaluate the per-pixel "focusness" of the source image using a sharpness operator that is the energy of the image Laplacian (17), applied on a window of 20×20 pixels centered on each evaluated pixel.

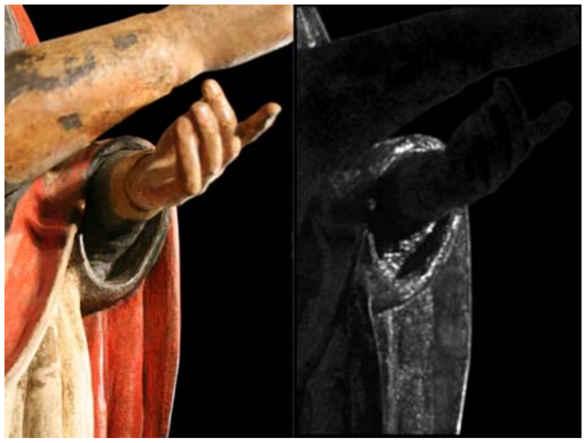

Fig. 3. Focus masking example. An image of the wooden statue with large depth extent. As detected by the focus mask,the hands are out of focus due to depth of field, while the red part of the cloth is perfectly focused.

Figure 3 shows an example of focus masking; in this case the focus problem was due to excessive depth of field. With this masking the out of focus areas are assigned a very low weight. In this way they do not introduce blurring when blended with other images, but they could still be available as color source if no other data can be mapped on the same surface region.

4.2. *Weights computation*

It is important to note that the computation of the various metrics is not computationally expensive. Most of the computation is done in the image space with simple image filters and does not require the access to complex data structure such as the topology of the 3D model.

The weight mask computation complexity is linear in the size of the input photos; more precisely, linear in the number of pixels in the input images which cover a portion of the 3D model (pixels covering the background are not processed).

We need to render the 3D model (using the view specs of each image) in two phases: for the computation of the depth map and for the computation of the angle map. Even in these cases, the time required is not large, since rendering a 3D model and reading back the buffer is a matter of a fraction of a second. This is true even when working with huge 3D models, assuming that those are rendered using modern out-of-core multiresolution algorithms (18).

Another interesting characteristic is that the computation of the metrics can be done independently for each image; we will later exploit this property to organize the out-of-core processing scheme (see Section 7).

4.3. *Weight normalization*

Mixing the various intermediate masks to produce the final one using a simple product is fine to preserve properties of the different weights without numerical problems.

However, numerical cancellation in the weighed mean can show up when applying the function (Section 5). If weights are quite discordant (very large values vs. very small values) multiplying color values (normally in the range 0-1) for the weight, accumulating and then dividing by the sum of all weight can produce artifacts even when using double precision values.

As the number of masks used increases, it is necessary to be cautious to select the range of the weights; having all mask between 0 and 1 would be the best thing. This condition is already fulfilled by the angle mask, that is already in the desired range. For border mask, it is possible to encode the distance from border (in pixels) normalized with respect to the largest image size (in pixels) in the dataset.

For the depth mask this can be obtained by normalizing depth with respect to the range between the minimum and maximum distance found among all images. this, however, requires two pass: a first pass to compute the depths, obtaining minimum and maximum depths and a second pass for normalization. A faster but still valid strategy is to find an over-estimation of minimum and maximum depth using camera positions and the object bounding box and using the values to do calculate normalized depth in a single step.

5. Application of the weighting function

Given a point on the 3D surface, it is easy (as described in Section 3) to determine the set of source images that effectively “see” that surface point and the corresponding pixel coordinates. These source colors will be multiplied by the weights contained in the image masks and accumulated, assigning the weighted mean as final color to the 3D point.

A very nice characteristic of this function is its generality: given a point on the surface, a color is

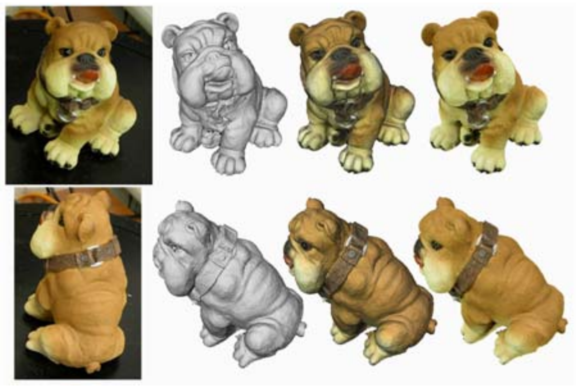

Fig. 4. An example of color mapping. The 3D model is composed by 1 million triangles, with a mapped photographic dataset of 8 images 2560×1920. From left to right: original photos, geometry, color + lambertian shading, color only.

returned. It is then possible to use it to fill a texture map or, with a simpler approach, to compute per-vertex colors on a 3D model. Moreover, the masks are *independent* from the resolution of the mapping target; after the weighting masks have been calculated once, it is possible to use them to apply the color on different level of resolution of the 3D object.

If a parametrization of the 3D model is available, it is possible to use the weighting function to compute the color of each texel in the texture map. Since there is a full correspondence between texels and points on the 3D surface, it is necessary to calculate the function for each texel. The generation of a suitable parametrization for an arbitrary model is not in the scope of this paper and, especially when dealing with very large or topologically unclean models, it is often a very hard task. For this reason we believe the simpler approach of computing per-vertex color is a much better solution in all those cases where both a high resolution geometry and photographic data are available. With per-vertex color we mean saving a color value for each vertex of the mesh, using simple interpolation of the three vertex colors inside each triangle. Most 3D file formats can support this data, and most of the available viewer and rendering engine can make profitable use of this kind of encoding. The same is not true for texture mapping where, for example, programs may have a limit in number, size and aspect ratio of texture images and limitations on the kind of UV mapping storage (per vertex, per wedge, atlas). A major advantage of per-vertex encoding is space efficiency, since we have just a color per vertex (rather than one texture coordinate per vertex and a texture atlas to be transferred to the GPU).

The images shown in this paper are all obtained using per-vertex color. When dealing with densely scanned objects, it is common to have a sub-millimetric geometric representation. This detail can be used even in realtime, thanks to multiresolution and out-of-core rendering techniques (18), coupled with modern video cards. It is therefore worth to try using this geometric density to store the color information. Using a multiresolution rendering engine, which renders each part of the scene at a different resolution level, each triangle usually projects to a few display pixels. A per-vertex encoding is then more than enough for a realistic visualization. An example can be the capital shown in Figure 6, where the per-vertex mapping is able to convey a good color detail even on a small, 1 million triangles model; only a very close inspection shows a small quality degradation. The limit of this encoding is not the sheer size of the model: if not presenting high frequency color details, a per-vertex encoding would be good also for very simple models (under 100.000 triangles). What matters most is the ratio between the pictorial and geometric detail: the per-vertex encoding begin to suffer when the pictorial detail is 3-4 times higher than the geometric detail and becomes useless as the ratio reach one order of magnitude (1 mm brush strokes on a wall reconstructed at 1 cm resolution).

Using all the redundancy contained in the image dataset, we are able to obtain a high quality color mapping. Even if we are blending multiple images, high frequency details are not lost thanks to the weighting function. Image areas with more detail will have a higher weight, that will dominate over other lower, more blurred, images. At the same time, discrepancy between different photos is no more detectable, since blending the overlap area between different images will mask the discontinuities.

Since the color blending function relies on redundancy to determine the most correct color to apply, a good level of overlapping is essential to overcome strong color discrepancies due to illumination changes or color biasing between different images. Having all points of the surface covered by at least three images can be enough to correct most of this problems. This may appear as a strong requirement, but such a dense sampling of the surface is common in 3D scanning. When not sufficient overlap is available, or some images present a poor color coherence with the others, some artifacts can be perceived in the resulted mapping. The next section presents a partial solution to this sub-problem.

6. Image Color Correction

As stated before, the weights express the quality of the pixels on a local basis. After mapping the color to the mesh, we can also evaluate for each image its global usefulness in terms of average, min and max quality. If an image results much lower in quality with respect to the others, a warning message can be sent to the user: this allows the user to exclude the image from the process or to further decrease the weight values for the entire image. The second option is quite helpful if that critical image is the only one that covers a portion of the 3D model; eliminating it completely would produce a non-mapped section, while reducing the weight (e.g., to 1/2 or 1/4 of the original values) greatly reduces its influence on the overlapping region, still conserving the data on the unshared surface section.

The usefulness of this feature should not be underestimated: when dealing with a very large dataset (e.g., the one used for the David 3D model is more than 60 images), detecting imperfections and problems among all available images by visual analysis can be a difficult task.

A better error evaluation can be done regarding the color coherence. Every time we evaluate the blending function for a 3D point, we sample the contributing images to retrieve the source colors (with associated weights) and then we compute the correct color as the weighted mean. The difference between the sampled color and the final color is a measure of how much the source image was coherent with the other neighbors. We can determine, for each image, how severe this incoherence is by keeping track of all those mapping errors. If we detect that the error value is above a threshold, it is possible to mark that image as discordant.

In this case it is also possible to provide a correction for the erroneous image. For each pixel sampled on this image, we know which is the color that, after blending, is mapped onto the 3D surface. Each one of those color couples represents a color transformation; we are interested in a global correction able to make the image more coherent with its neighbors. This global color transformation, described in Agathos and Fisher (19), can be expressed as a 4×4 *color correction matrix* M, that can be evaluated by solving a linear system showed in Equation 1:

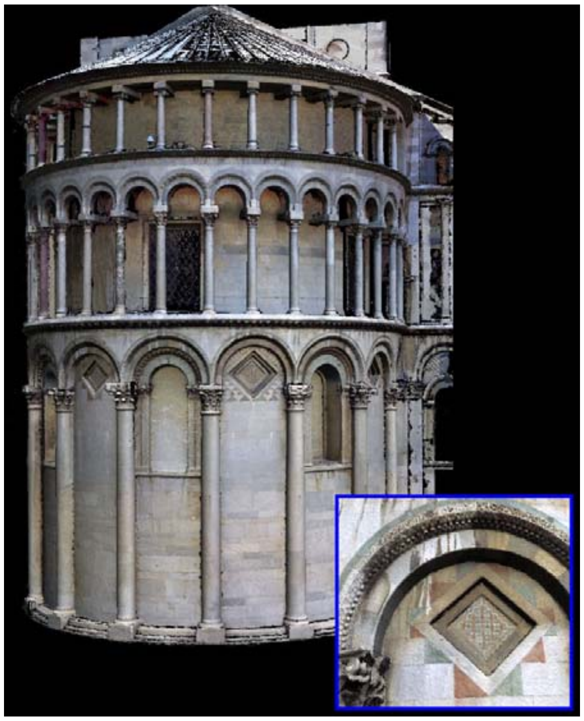
Fig. 5. Apse of the Pisa Cathedral. A multiresolution model (46 million triangles) with color applied from 36 4000×4000 images.

$$\begin{bmatrix} R_1\,G_1\,B_1\,1 \\ \vdots \\ R_n\,G_n\,B_n\,1 \end{bmatrix}_{samp} \cdot \begin{bmatrix} \\ M \\ \\ \end{bmatrix} = \begin{bmatrix} R_1G_1B_11 \\ \vdots \\ R_nG_nB_n1 \end{bmatrix}_{final} \quad (1)$$

where the vector of sampled colors is multiplied by the correction matrix M (the unknown) to obtain the vector containing the final colors.

The correction matrix can therefore be applied to the image and the result can be presented to the user; if the user believes the new image is better than the older one, the image dataset is updated and the mapping process is repeated. This color correction is done, again, taking into account the weight associated to the pixel and computing the matrix in the CIE XYZ color space (20), to guarantee a more stable correction.

It is notable that, while this kind of correction is technically sound, in practice it could fail. This is because the color couples used to build the matrix are not well distributed in the color space, since they are only associated to the colors that are present in the common areas of source images. In this case, the linear system used to compute the color correction matrix could become ill-conditioned, introducing color-shifting problems. A similar approach for color correction is used in Bannai et al.(21), but also in the dataset presented in their paper a similar color-shifting artifact is present. This is the reason why we chose to have the system prompt the correction to the user for confirmation, instead of doing the correction automatically.

To avoid this kind of color problems, it is much advisable to perform a color calibration *on the field*, during image acquisition. Including in each shot a Macbeth color chart greatly helps in having a wide sampling color space. Using the sampled color values from the chart image, compared to the color chart reference, it is possible to obtain a stable color correction matrix. We have a simple program that automatically sample the image and build the correction matrix, the user just have to click on two corners of the color chart on the image. Some image acquisition programs do this step semi automatically and there are Photoshop scripts and Matlab code snippets with the same purpose. If the lighting setup is particularly stable, it is not necessary to calibrate each photo with a color chart, but a global color correction matrix can be derived from just a few shots. In our acquisition campaigns we usually apply this strategy, but we had to implement the more generic correction approach described previously to deal with image datasets coming from other sources.

7. Large Dataset Management

Executing the proposed pipeline for color reconstruction from photos could require excessive memory resources on very large datasets.

The problem of dealing with large dataset depends on the size of both the 3D geometry and the photographic data. When "all in memory" approach is adopted, we need sufficient RAM for storing the 3D geometry, each input photo with the associated depth map and weight mask (both float vector, same size of the image). The memory capacity can be easily exceeded. It is also impractical to set up a system which adopts loading and discarding the images and the maps upon request, since there is not much locality and the size of these "chunks" is quite large. It is much more efficient to compute and organize the data in a way that is easy to access on disk and to adopt Out-Of-Core (OOC) strategies.

Designing the system, we distinguished two different cases of OOC management: the most common

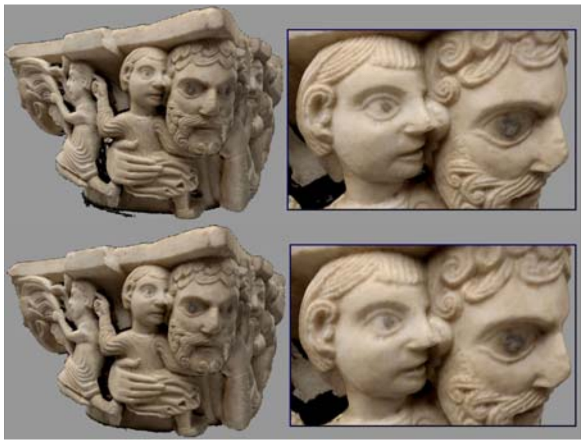

Fig. 6. Marble capital. Top row: 6 million triangles, bottom row: 1 million triangles. The photographic dataset used was 8 images 1800×1200, per-vertex mapping encoding. Even reducing the complexity of the 3D model, the mapping quality remains high; quality degradation due to the per-vertex approach starts to appear on the smaller model as we get very close to the object (the eye of the small figure is 6mm wide).

case is when the 3D model fit in memory, but the image and mask dataset have to be managed out-of core. In the second case, also the 3D model is too large to be kept in memory, requiring the interaction of two OOC systems.

The mask computation can be done independently for each image, since it is a local processing; this greatly helps the OOC management, since it is possible to calculate one mask at a time, keeping the minimum data in main memory. Computing the mask is easy if the 3D model fits in memory, but it is also viable if the 3D model is kept OOC. Since we only use controlled rendering, it is just necessary that the OOC structure we are using for the 3D model supports rendering. In this way it is possible to generate all the data necessary to apply the blending function. Moreover, the masking data for each photo can be stored in a single image-buffer file with a tight packing: for each pixel we store the color values (from the image), the depth value (for the shadow-mapping projection) and the mask value (for blending). Thanks to these two properties (independent calculation and compact storage), building the OOC scheme becomes easier.

Organizing the mapping phase is straightforward when the model is in main memory. We can load from disk one image-buffer at a time, for each mesh vertex which maps on this photo we compute the associated color contribution, the corresponding weight value according to the image mask and finally accumulate this color and weight into the vertex data structure. After all images have been processed, we obtain the required mean weighted color iterating again on each vertex and computing the mean of all the weighted contributions. This strategy, with a dataset of N images requires N+1 visit to all mesh vertices (one for each image, plus one for mean color calculation), but minimize memory-disk swapping. The same strategy of processing one image at a time can be applied in the case of texture-mapped color encoding to synthesize the new texture.

Conversely, when the 3D model is too large to fit in RAM and has to be managed using OOC structures, the mapping phase is a bit more tricky. Using a multiresolution data structure like the one presented in (18) makes it quite difficult to accumulate color and weights by processing one image at a time, since for each image we need to traverse the mesh vertices in a random manner; traversing the multiresolution data many times increases the processing time and the lack of supporting structures for accumulation is another issue. For this reason, we implemented a color mapping strategy where the multiresolution data structure is traversed just once. For each vertex in the multiresolution mesh, we can efficiently detect which images "sees" this vertex (by just using the inverse camera transformation); for each of these images, the color contribution and weight corresponding to the vertex are retrieved by directly accessing the various image-buffer files using memory mapping system calls. The overall reconstruction time is therefore increased wrt. the all-in-memory case, but in a sustainable manner (see the David and Apse results in Table 1).

8. Results

To evaluate the effectiveness of our approach we tested it on various datasets. In all cases, the input images have been previously registered to the geometry with the tool presented in (5). All 3D models have been produced with 3D scanning technology, therefore thee meshes are not watertight nor topologically clean. The images included in this paper show some of the models obtained with per-vertex color. Table 1 lists some processed objects with details on their size, source images used and time required for performing color mapping. The examples range from a small object with a relatively low geometric and photographic complexity like the dog of Figure Figure 4 to a complex, high resolution model

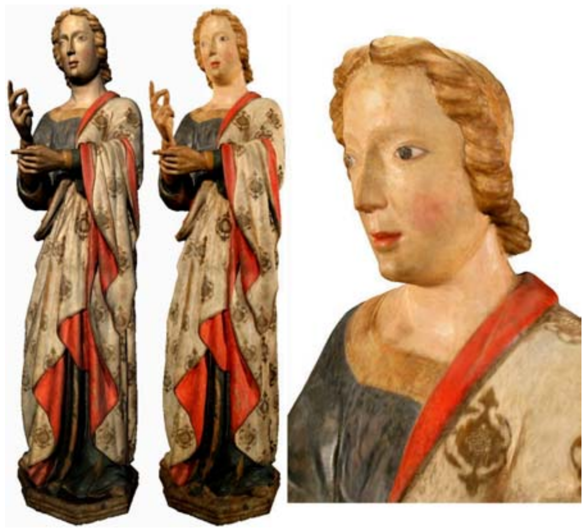
Fig. 7. Life-size painted wooden statue (5 million triangles, photographic dataset 33 images 1694×2496). Left: shaded+color, center: color only, right: a more detailed view.

of a building with a very dense photographic as the Apse of Figure 5.

In all dataset, the images have been aligned using our image registration tool (5). The tool is able to compute intrinsic and extrinsic camera parameters by following the standard approach of using a series of corresponding points selected by the user on the photo and on the 3D geometry. The corresponding points are processed iteratively using two different algorithms to calculate the parameters (2; 4) until convergence. The tool is able also to work using image-to-image correspondences (to use also feature that are only visible in the images, like paint details) and is able to work directly on very large 3d models like the one used in this tests, since it is possible to use multiresolution models to render and pick the correspondences. The tool proved flexible and accurate; it has to be considered that the images in the test datasets comes from at least 4 different cameras, with very different intrinsic parameters and also the models are of quite diverse nature.

All datasets have been processed using the three core mask, the focus mask has been used on the painted wooden statue (as some photos had out-of-focus areas). For the Apse it was also used the stencil mask, to cull out part of the scaffolding and other extraneous elements (like pigeons).

Additionally, to demonstrate the possibility to manage a very large dataset, we selected a test case of an extraordinary size. We mapped on the 3D model of Michelangelo's David two complete

Table 1
Specifications of the color mapped objects (note that for multiresolution models, the number of triangles indicated is the size of the model extracted at full resolution and not the number of triangles in the entire multiresolution data structure, which usually is around six times the former).

Object	Resolution (# triangles)	Images	M pixel	Time
Dog statuette	1 M	8 (2560×1920)	39	3 min
Marble capital	6 M	8 (1800×1200)	17	8 min
Painted statue	5 M	33 (1694×2496)	138	15 min
David, single resolution (pre-restoration)	15 M	60 (1920×2560)	294	3.0 hr
David, single resolution (post-restoration)	15 M	64 (2336×3504)	523	3.5 hr
David, multiresolution (pre-restoration)	56 M	60 (1920×2560)	294	13.0 hr
David, multiresolution (post-restoration)	56 M	64 (2336×3504)	523	15.5 hr
Apse, multiresolution	46 M	36 (4000×4000)	576	10.5 hr

photographic campaigns depicting the status of the statue before and after the restoration. The two photo dataset were taken by a professional photographer at the begin and at the end of the restoration of the statue (2002-2004), the mapping of the first dataset is shown in Figure 1. The photo dataset are composed respectively by 61 images 1920×2560 and 68 images 2336×3504 (around 300M and 500M pixels). The 3D model comes from the scanning campaign of the Digital Michelangelo Project performed by Stanford University (22). We used two different geometric datasets for mapping: the first one composed by 15 million and the second 56 million triangles. While in the first case it was possible to keep the geometry in main memory (and perform OOC mapping), in the second case both photographic *and* geometric data were accessed using OOC structures.

As introduced previously, blending masks can be reused to map an object at different resolution. We generated the masks using the larger (56M triangles) model and then used the data to apply color on both (56M and 15M triangles). The time needed to generate the OOC data necessary for color blending was 30 minutes for the first dataset and 50 minutes for the second one. The size on disk of the temporary data was, respectively, 3.6 GB and 6.2 GB for pre- and post-restoration data.

The second phase, color blending and mapping for the low resolution model, took 2.5 hours for both datasets (in this phase, time is almost only dependent on the number of function evaluation performed). The time increased while mapping on the larger dataset, since also the geometry was stored OOC; mapping time was 12.5 hours for the first dataset and 14 for the second. Moreover, the increase in mapping time also follows the fact that the mapped model is a multiresolution data structure (18) that contains a continuous LOD representation of the geometry (hence, many more vertices to map).

The results are visible in Figure 8. There are still parts without color, but this is caused by a lack of photographic coverage and does not depend on defects/bugs of the mapping process. These missing areas have been caused by the topology of the statue, its size and the situation of the scaffolding, that changed frequently over the course of restoration, making it difficult both to plan the shoots and to reproduce them correctly on site.

9. Open issues and future enhancements

One of the main problems in using calibrated photos is that the quality of mapped color greatly depends on the quality of camera calibration. If the cameras are poorly calibrated the result will be a blurred color information or, when sharp features are present, ghosting effect (multiple copies of the same feature). A simple strategy to detect local misalignment when mapping the color can be to consider the consistency of the various color candidates for the same point: if one sample is very different with respect to the mean of the other candidates, an image misalignment can be present. However, to be effective, this consideration would require many samples for each point; moreover, in our case, this detection would be only detectable on a per-point basis, making it very unreliable. More complex strategies, like the one presented in our old texture mapping paper by Rocchini et al.(23), will require the availability of some form of topology for

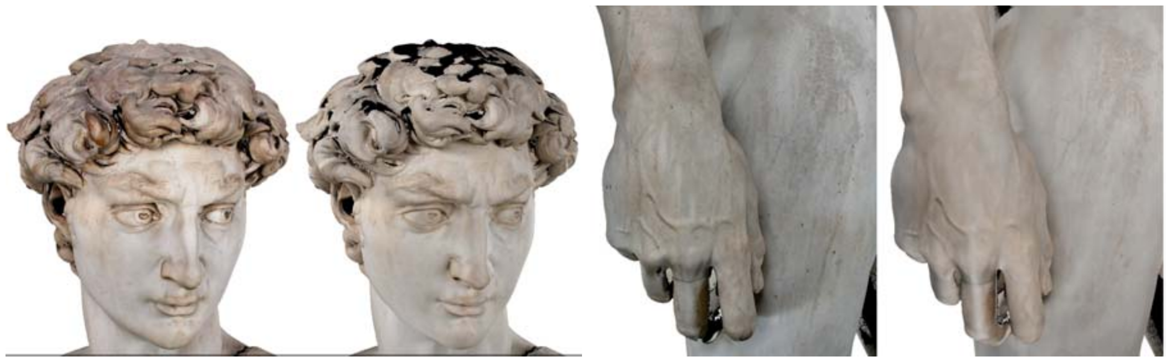

Fig. 8. Details of the David with the two mapped datasets. Leftside: before restoration, Rightside: after restoration. (upper part of head in the post restoration phase was not covered by the photographic campaign)

the 3D model, in order to correlate neighborhood point misalignment and to apply an image-warping process to correct the alignment. Unfortunately, generating and storing topology for such large models is almost impossible. At the moment we have no solution for this problem; if a misalignment artifact is detected (blurriness and/or ghosting), the image alignment is manually refined using our registration tool and the mapping process is repeated. We plan to make this iteration (color mapping - adjustment - color mapping) more automatic, by sharing data between the image alignment tool and the color mapping tool.

A major problem when mapping photos onto a 3D geometry is the presence of *shadows* and *highlights*. For this reason an interesting goal could be to detect parts of the image that need to be excluded because under shadow. Shadows have been widely analyzed in literature for image and video processing applications and several techniques for shadow detection and removal have been developed in recent years. Most of these techniques rely on color analysis. Suitable color models are exploited to classify in-shadow pixels (24; 25). In general, geometric information of shadows are less used due to the fact that the geometry depicted in the image is usually unknown. Recently, Sato et al. (26) have addressed the problem of estimating the illumination distribution from images with known geometry and unknown reflectance properties (which corresponds to our cases). This method could be easily adapted to our purposes: after the illumination distribution is recovered, for each image, the pixels in shadow or in an highlight can be identified with a good degree of precision and a corresponding *shadow/highlight mask* can be built. Following this idea the next versions of our photo blending system will account also for this kind of artifacts.

Another open issue is how to deal with *surface sections not sampled by the photos*, like the top of the head of the David (post-restoration dataset). Texture InPainting Approaches (27; 28) have been recently proposed and can be adopted to synthesize color information for unsampled regions.

A final issue is how to manage dataset where the *photo detail is sampled more densely than geometry*. Even when working with huge 3D models like the 56 Million triangles David, the number of vertices can be still lower than the amount of available pixels. A simple, brute-force refinement strategy (subdivide all triangles to generate new vertices, thus accommodating more data) can solve the problem but will increase the complexity of the geometric dataset. A more interesting solution would be to refine selectively the existing geometry only where there are significant color changes. An even better solution would be to store this refinement data (i.e., the colors associated to the refined vertices) without refining explicitly the original geometry, but implementing the refinement at the rendering stage, hopefully finding proper support in the new features of future programmable GPUs.

10. Conclusion

We have presented a framework for image blending and mapping onto 3D models. The system relies on local and geometry-based per-pixel quality evaluation of the source images and a robust mapping function. We showed that by using simple weighting functions it is possible to take into account all the typical problems related to the integration and mapping of photographic datasets over scanned 3D models. With this weighting mask we can efficiently use all the redundancy present in the source images to reduce illumination artifacts and incoherence between different images. We also show how it is possible to extend the basic system to overcome specific dataset problems, like un-focused areas.

To demonstrate the capabilities of the proposed approach, we tested it on various dataset proving its robustness. To assess the possibility to manipulate huge datasets, we selected some very large dataset and applied the OOC version of the algorithm, obtaining high quality results in a reasonable time.

Aknowledgements. We would like to acknowledge the contributions by Franca Falletti, the Director of the Galleria dell'Accademia Museum and responsible for the David restoration. The David 3D model is courtesy of the Digital Michelangelo Project, Stanford University. The color images of the David model have been produced by Rabatti and Domingie Photographers and Galleria dell'Accademia, Florence, Italy. We gratefully acknowledge the financial support of: "Friends of Florence" association (Florence), EU NoE IST-2002-507382 "EPOCH" and the CNR Project "Fruizione - Tecniche di supporto e modalità innovative".

References

[1] P. Pingi, A. Fasano, P. Cignoni, C. Montani, R. Scopigno, Exploiting the scanning sequence for automatic registration of large sets of range maps, Computer Graphics Forum 24 (3) (2005) 517–526.

[2] R. Tsai, A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, IEEE Journal of Robotics and Automation RA-3 (4).

[3] H. Lensch, W. Heidrich, H. Seidel, Automated texture registration and stitching for real world models, in: Proc. 8th Pacific Graphics 2000 Conf. on Computer Graphics and Application, IEEE, Los Alamitos, CA, 2000, pp. 317–327.

[4] F. Dornaika, C. Garcia, Robust camera calibration using 2d to 3d feature correspondences, in: Proceedings of the International Symposium SPIE –Optical Science Engineering and Instrumentation, Videometrics V, Volume 3174, 1997, pp. 123–133.

[5] T. Franken, M. Dellepiane, F. Ganovelli, P. Cignoni, C. Montani, R. Scopigno, Minimizing user intervention in registering 2d images to 3d models, The Visual Computer 21 (8-10) (2005) 619–628, special Issues for Pacific Graphics 2005.

[6] E. Beauchesne, R. Roy, Automatic relighting of overlapping textures of a 3d model, in: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03), Vol. 2, 2003, p. 166.

[7] A. Sheffer, E. Praun, K. Rose, Mesh parameterization methods and their applications, Foundations and Trends in Computer Graphics and Vision 2 (2) (2006) 105–171.

[8] A. Bornik, K. Karner, J. Bauer, F. Leberl, H. Mayer, High-quality texture reconstruction from multiple views (2002).

[9] P. Neugebauer, K. Klein, Texturing 3d models of real worls objects from multiple unregistered photographic views, Computer Graphics Forum (Eurographics'99 Proc.) 18 (3) (1999) 245–255.

[10] M. Callieri, P. Cignoni, R. Scopigno, Reconstructing textured meshes from multiple range rgb maps, in: 7th Int.l Fall Workshop on Vision, Modeling, and Visualization 2002, IOS Press, Erlangen (D), 2002, pp. 419–426.

[11] L. Borgeat, G. Godin, F. Blais, P. Massicotte, C. Lahanier, Gold: interactive display of huge colored and textured models, ACM Trans. Graph. 24 (3) (2005) 869–877.

[12] Y. Yu, P. Debevec, J. Malik, T. Hawkins, Inverse global illumination: recovering reflectance models of real scenes from photographs, in: A. Rockwood (Ed.), SIGGRAPH 99 Conf. Proc., Annual Conf. Series, ACM SIGGRAPH, Addison Wesley, 1999, pp. 215–224.

[13] F. Bernardini, I. Martin, H. Rushmeier, High-quality texture reconstruction from multiple scans, IEEE Transactions on Visualization and Computer Graphics 7 (4) (2001) 318–332.

[14] A. Baumberg, Blending images for texturing 3d models, in: BMVC 2002, Canon Research Center Europe, 2002.

[15] V. Rankov, R. Locke, R. Edens, P. Barber, B. Vojnovic, An algorithm for image stitching and blending, in: Proceedings of SPIE. Three-Dimensional and Multidimensional Microscopy: Image Acquisition and Processing XII, Vol. 5701, 2005, pp. 190–199.

[16] K. Pulli, H. Abi-Rached, T. Duchamp, L. Shapiro, W. Stuetzle, Acquisition and visualization of colored 3d objects, in: Proceedings

of ICPR 98, 1998, pp. 11,15.

[17] M. Subbarao, T. Chio, A. Nikzad, Focusing techniques, Optical Engineering 32 (11) (1993) 2824–2836.

[18] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno, Batched multi triangulation, in: Proceedings IEEE Visualization, IEEE Computer Society Press, Conference held in Minneapolis, MI, USA, 2005.

[19] A. Agathos, R. B. Fisher, Colour texture fusion of multiple range images., in: 3DIM, IEEE Computer Society, 2003, pp. 139–146.

[20] CIE, Commission Internationale de l'Eclairage Proceedings, Cambridge University Press, 1931.

[21] N. Bannai, A. Agathos, R. Fisher, Fusing multiple color images for texturing models, in: 3DPVT04, 2004, pp. 558–565.

[22] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, D. Fulk, The digital michelangelo project: 3D scanning of large statues, in: K. Akeley (Ed.), Siggraph 2000, Computer Graphics Proceedings, Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000, pp. 131–144.
URL `http://visinfo.zib.de/EVlib/Show?EVL-2000-49`

[23] C. Rocchini, P. Cignoni, C. Montani, R. Scopigno, Multiple textures stitching and blending on 3d objects, in: D. Lischinsky, G. Ward (Eds.), Rendering Techniques '99, Springer-Verlag Wien, 1999, pp. 119–130.

[24] T. Gevers, Reflectance-based classification of color edges, in: 9th International Conference on Computer Vision (ICCV'03), Vol. 2, 2003, p. 856.

[25] E. Salvador, A. Cavallaro, T. Ebrahimi, Cast shadow segmentation using invariant color features, Computer Vision and Image Understanding 95 (2) (2004) 238–259.

[26] I. Sato, Y. Sato, K. Ikeuchi, Illumination from shadows, IEEE Transaction on Pattern Analysis and Machine Intelligence 25 (3) (2003) 290–300.

[27] K. Zhou, X. Wang, Y. Tong, M. Desbrun, B. Guo, H.-Y. Shum, Texturemontage: Seamless texturing of arbitrary surfaces from multiple images, ACM Trans. Graph. 24 (3) (2005) 1148–1155.

[28] H. Yamauchi, J. Haber, H.-P. Seidel, Image restoration using multiresolution texture synthesis and image inpainting, in: Computer Graphics International (CGI 2003), IEEE, Tokyo, Japan, 2003, pp. 120–125.