DOI: 10.1111/cgf.12508
COMPUTER GRAPHICS *forum*
Volume 00 (2014), number 00 pp. 1–11

Seamless, Static Multi-Texturing of 3D Meshes

R. Pagés, D. Berjón, F. Morán and N. García

Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, Spain
{rps, dbd, fmb, ngs}@gti.ssr.upm.es

Abstract

In the context of 3D reconstruction, we present a static multi-texturing system yielding a seamless texture atlas calculated by combining the colour information from several photos from the same subject covering most of its surface. These pictures can be provided by shooting just one camera several times when reconstructing a static object, or a set of synchronized cameras, when dealing with a human or any other moving object. We suppress the colour seams due to image misalignments and irregular lighting conditions that multi-texturing approaches typically suffer from, while minimizing the blurring effect introduced by colour blending techniques. Our system is robust enough to compensate for the almost inevitable inaccuracies of 3D meshes obtained with visual hull–based techniques: errors in silhouette segmentation, inherently bad handling of concavities, etc.

Keywords: texture mapping, texture synthesis

ACM CCS: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Colour, shading, shadowing and texture

1. Introduction

Realistic 3D models are very useful and have been widely adopted for entertainment (computer games and films), but also for many other more serious fields such as health, education or security, where virtual/augmented/mixed reality applications play an increasingly important role. A key component which makes a 3D model look realistic is its texture. Indeed, in the process of capturing a 3D object from the real world, many problems appear, most of them related to the acquisition of its shape/geometry, typically a surface approximated by a triangular 3D mesh; but also related to the appearance attributes of that mesh, normally to be rendered with texture mapping. Although high-resolution scanning techniques are becoming more common nowadays, there are still many applications which generate a 3D model by processing a set of images taken from different viewpoints, which may be either sparsely distributed around the object or very densely, if a video stream is recorded by surrounding it. In both cases, a system is necessary to assign a texture and the corresponding texture coordinates to each vertex of the mesh, and to do so as automatically as possible.

We consider a very popular capturing scheme involving no complex structured light patterns or expensive projectors, but simply a group of regular photographs taken from different perspectives. These pictures can be provided by shooting just one camera at different time instants, if a static object is reconstructed, or a set of synchronized cameras in the case of a moving subject such a person (see Figure 1 to see our camera setup for 3D humanoid reconstruction). The calibration parameters of the camera(s) are extracted either using a calibration pattern or an autocalibration approach. A segmentation algorithm is first used to extract the silhouette of the subject in each of the pictures; then, thanks to the calibration parameters of the cameras, another algorithm extracts a visual hull (VH) of the subject's surface; finally, after voxelizing the VH, a plain (i.e. untextured) 3D mesh approximating that surface is extracted from it with the marching cubes algorithm. Our system was conceived for dressing that 3D mesh with a single texture atlas obtained by mixing the colour information provided by the different photographs.

In the case of avatar capture, the typical setup we are using processes around a dozen photographs taken from strategic locations so as to cover most of the face and body surfaces. The resolution of the input images will determine the quality of the resulting 3D models. Nevertheless, as VH-based techniques do not handle concavities correctly, these models undergo a facial refinement process which increases the polygonal resolution of the facial section of the mesh significantly.

Some high-quality view-dependent texture mapping techniques dress 3D models dynamically, i.e. at rendering time [DYB98].

Figure 1: *Typical camera setup. Top: a photo taken by one of the cameras of the setup, with other cameras highlighted in red. Bottom: the capture room may have very irregular lighting.*

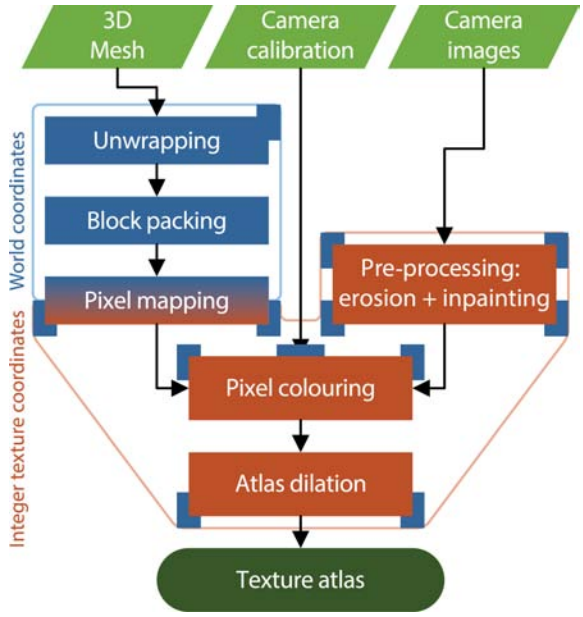

Figure 2: *Different stages of our system, which takes as input one 3D mesh plus several 2D camera images and the corresponding camera calibration parameters. Some processes operate in 3D, floating point, world coordinates (highlighted in blue) whereas others use 2D, integer, texture coordinates (in orange).*

However, the models they produce are less usable than those whose textures are computed statically, once and for all viewpoints, at modelling time. The reason for this is that the former needs a specific renderer able to handle the transitions between viewpoints, whereas any generic renderer can handle the latter.

Nevertheless, using static texture mapping does not preclude mixing the pixels from all input images, and we precisely propose a method which creates a single, static texture atlas using a combination of the colour information provided by different input images. The colour mixture performed by our method takes into account perspective distortion, which avoids discontinuities in the textured model, and creates a smooth seamless texture across the whole 3D model. An initial and less robust approach and its results were previously presented in another paper of ours [PAMB10]. In this, extended paper it is possible to find a deeper mathematical description of the processes, as well as a better analysis of the impact of the different stages of the algorithm. Besides, we have included a very important step to avoid the inclusion of unwanted background pixels in the blending stage. We also present a better study of the state of the art of multi-texturing approaches and an improved analysis of the results compared to some of these other techniques. All this makes it easier to reproduce, and also to decide whether it suits any particular multi-texturing scenario.

2. Proposed Technique: Overview

As shown in Figure 2, our system is divided in several stages which can be classified as 3D, floating point, world coordinate (WC) processes versus 2D, *integer*, texture coordinates (ITC) processes. Note that our ITC are not the typical (u, v) *floating point* texture coordinates.

The first stage which operates with WC, explained in Section 4.1, is unfolding/unwrapping the 3D mesh using a zero-distortion approach. This creates a set of 2D patches which are later packed efficiently so the resulting image is as compact as possible (see Section 4.2). Section 4.3 elaborates on the last stage, which consists in a WC to ITC mapping, and where the final resolution of the resulting texture atlas is determined.

Before colouring the texture atlas resulting from the previous steps, it is necessary to pre-process the input images obtained from the camera setup. Section 4.4 is devoted to this stage, which removes the background in the images so it is not included in the textured model even if the volume of the 3D model is bigger than the subject, as is usually the case due to the nature of the VH technique.

Section 4.5 explains the most important stage, and the one where our main contribution resides: determining the colour of every pixel in the texture atlas. This process blends RGB data from different cameras to texture every triangle, but needs a previous rectifying stage, which uses the zero-distortion unwrapping system mentioned above. The blending itself is a customized interpolation which uses the most influential cameras for each vertex and triangle, thus avoiding discontinuities or visible seams in the textured model.

Lastly, to avoid any possible spurious inclusion of the texture atlas background in the textured 3D model, which could happen with renderers using bilinear (or bicubic) interpolation, it is advisable to apply an additional step. Once we have the texture atlas completely

coloured using our blending technique, we extend the shape of the unwrapped patches using a nearest neighbour approach detailed in Section 4.6.

3. Previous work

In the process of creating a realistic 3D model using several input images to be used in the texture mapping process, it is very convenient to create a single texture atlas where information from the different images is packed together. This helps compile (and, in some cases, also save) information in transmission or storage scenarios, where we would otherwise need to send or save all the images independently. Because of this, the creation of texture atlases is a topic which has been very well studied [WLK*09]. However, researchers and professionals do not agree on a common approach when they create 3D models from real objects.

Every approach first checks which is the best camera for texturing a determined region (group of triangles) of the mesh, and normally bases this decision on a perpendicularity criterion. If this was the only criterion taken into account, we could—and, most frequently, would—have big discontinuities across neighbouring regions assigned to neighbouring cameras. These discontinuities can be due to: (i) the perspective transformation present in every image, which may lead to misalignments in the textured model; (ii) irregular illumination conditions (see bottom row of Figure 1), as in general cameras are not (jointly) colour-calibrated and surfaces are non-Lambertian, which typically leads to big variations in colour and very noticeable seams in the global texture and (iii) poorly reconstructed 3D geometry which does not represent the model accurately. Proposed techniques which solve these problems can be divided into two big categories: image stitching versus image blending ones.

Image stitching techniques focus on solving the misalignment problems by finding correspondences in areas where there is a potential visible seam, and distorting iteratively the images to be stitched until all the correspondences have been aligned. This is the basic idea behind the approach proposed by Gal *et al.* [GWO*10], where a labelling system is used to perform these local geometric transformations. Their results are correct, but their iterative system can be very time-consuming. Moreover, when the texture applied has many high-frequency components and the geometry presents errors, the system is unable to form a seamless montage. A similar approach is the one proposed by Aganj *et al.* [AMK10], where the deformations applied to the input images come from estimating some displacement vectors by finding feature correspondences among the images. Another interesting approach is the one proposed by Lemptisky and Ivanov [LI07] to solve the problem using a Markov random field mosaicing system, and a seam levelling technique specially adapted to manifold 3D meshes. Although their results are also good, and most of the artefacts are successfully removed, some seams are still visible when small details are present in different images. These methods are initially conceived to handle misalignments, where they have a good performance. However, they need to add a global colour correction stage to compensate for different colour balance in different input images, which can lead to visual colour seams in case of extreme lighting conditions. Moreover, these techniques introduce some distortion in the images which can be too severe in some cases, mainly in images with many high-frequency components and especially if they are mapped onto inaccurate geometry.

On the other hand, image blending techniques try to solve the seam problems by merging the colour information provided by two cameras using different criteria. For instance, in the system proposed by Rocchini *et al.* [RCMS99], resampling is applied near texture patch borders. A classic approach is the one introduced by Baumberg [Bau02], where 2D image blending techniques are extended to 3D. Another technique is the one developed by Allène *et al.* [APK08], which first locates the seams very efficiently using graph-cuts, and then applies pixel-wise colour correction across neighbours with a multi-band image blending system. To obtain a good result with image blending techniques, it is important to consider the perspective distortion introduced in the images: for instance, Wang *et al.* [WKSS01] propose a system focused on perspective correction, which avoids aliasing thanks to a particular weighting method specially designed for blending. One of the most useful approaches is the one proposed by Callieri *et al.* [CCCS08], which blends the information from several masks for obtaining a reliable weighted colour contribution for every camera. These masks represent geometry, topology and colour in every pixel of the input images. This technique is focused on a colour per vertex texturing approach, which is valid for models with high polygonal resolution, but it can also produce a combined texture atlas if a parametrization of the 3D model is available. All blending techniques depend strongly on the calibration of the cameras, particularly in the acquisition of the extrinsic parameters, which are normally retrieved by feature matching algorithms. If these parameters are not very accurate, aliasing or ghosting effects are likely to be very apparent in the final textured model. To solve this problem, one can optimize the extraction of the extrinsic parameters using a contour-based approach, as in Marroquim *et al.*'s [MPMdCO11] work; or use the system proposed by Dellepiane *et al.* [DMC*12], which computes the optical flow between two overlapping images to correct the misalignment between them.

As in Callieri's algorithm, our approach is based on an image blending technique which, instead of focusing only on the areas where a potential seam could appear, applies an image fusion technique across the whole mesh. This way, colour is balanced in the whole 3D model, and there are no misalignments across mesh regions. With this, we generate a combined texture atlas which avoids the need of transmitting or storing a large set of images.

4. Proposed Technique: Detailed Explanation

4.1. Mesh unwrapping

Many texturing algorithms choose to create a texture atlas with certain distortion due to the parametrization they use to map a 3D mesh onto a flat image. For instance, Floater [Flo97] uses a method based on graph theory to create his parametrization, and Lévy *et al.* [LPRM02] create theirs based on a least-squares approximation of the Cauchy–Riemann equations. Instead, we decided to perform a simple and efficient unwrapping technique which does not introduce any distortion in the resulting texture atlas: every triangle is translated to the texture atlas in proportion to its original size in the 3D mesh. This is crucial for our colour blending technique

described in Section 4.5, because it needs to rectify the projection of each triangle onto each input image so that colour information from different images is merged correctly, and our zero-distortion unwrapping guarantees that the rectified/unwarped triangles will have the exact same shape as the original 3D ones.

Although a zero-distortion approach could lead, if used carelessly, to more visible seams in the textured model, this potential issue is later solved with the image extension process at the last stage of our system. This zero-distortion mesh unwrapping system has been very useful in the design of an intertexture error measurement system also developed by us [PFM11].

Before starting to unwrap the 3D mesh, which we assume to be manifold, it is necessary to gather some topological information about the mesh, e.g. triangle adjacency. Once this is done, Algorithm 1 unwraps the 3D mesh onto a set of 2D patches.

Algorithm 1. 3D mesh unwrapping

```
Data: 3D mesh and triangle adjacency info
Result: a set of 2D patches representing the unwrapped
        mesh
patches[] ← set which will store the 2D patches;
while there are triangles left do
    currentPatch ← new 2D patch;
    seed ← a random seed triangle is added to
    currentPatch;
    perimeter ← list of edges in the patch perimeter;
    while there are available edges in perimeter do
        edge ← we pick an edge;
        tri ← we pick the triangle attached to edge
        outside the patch;
        if tri does not pass the efficiency growth test
        then
            mark edge as unavailable;
            continue;
        end
        for all the edges in the perimeter do
            currentEdge ← current edge;
            if edges of tri intersect with currentEdge
            then
                mark edge as unavailable;
                break;
            end
        end
        if edge is still available then
            add tri to the current patch;
            update perimeter with the new edges;
            mark tri as used;
        end
    end
    insert currentPatch in patches[];
end
```

This algorithm starts by choosing a random *seed* triangle in the mesh, whose edges are added to a variable called *perimeter* and marked as 'available' by default, which is the state the algorithm will check when the patch starts growing. As can be inferred from the pseudo-code description, there is an efficiency growth test which prevents the patch from having long octopus legs, and thus leaving big empty gaps in the texture atlas. This test is performed with every candidate triangle by controlling the relation between the area of the patch and its bounding box, and keeping it as close to one as possible. Thus, we keep the effective area of the texture atlas reasonably close to the total area of the image.

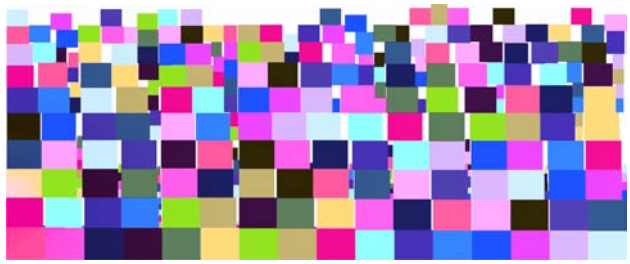

Figure 3: *Result of the block packing algorithm (patches are represented by their bounding boxes) [PAMB10].*

One of the key points in Algorithm 1 is checking for possible intersections among two or more edges in the same 2D patch when a new triangle is added. Since an edge is defined by two vertices, e.g. **a** and **b**, to see if two edges intersect, we equate the parametric expressions of the lines supporting each edge/vector:

$$(x, y) = \mathbf{a_i} + u_i(\mathbf{b_i} - \mathbf{a_i}) \ \ (i \in \{1, 2\}),$$

where u_i are the two unknowns of this simple equation system. If $u_i \in [0, 1]$, there is indeed an intersection, so the triangle is discarded for the current patch.

4.2. Block packing

Once the entire 3D mesh has been unwrapped onto 2D patches, it is important to find an efficient way of packing them so that they do not take up too much space. Although optimally packing the irregular shapes of the patches is possible, it is a very complex task, which would require high computational resources. Because of that, and because in the previous stage we have created compact patches, we have chosen to pack, rather than the patches themselves, their rectangular bounding boxes. This NP-hard problem has been studied extensively and is known as the 'pants packing' (or 'tetris packing') problem. Two of its best solutions are the ones proposed by Huang and Korf [HK09], and by Murata *et al.* [MFNK95]. An example of the specific use of block packing for texture mapping is described by Sander *et al.* [SSGH01]. In our case, we have chosen to implement a simplified version of their algorithm, where we fix one of the dimensions and optimize the other. This way, we obtain good results (as shown in Figure 3) while saving time.

4.3. Pixel mapping

At this point, we have already unwrapped and packed a set of 2D patches, but they are still expressed in WC. The transition between WC and ITC is what we call pixel mapping, and it is the process

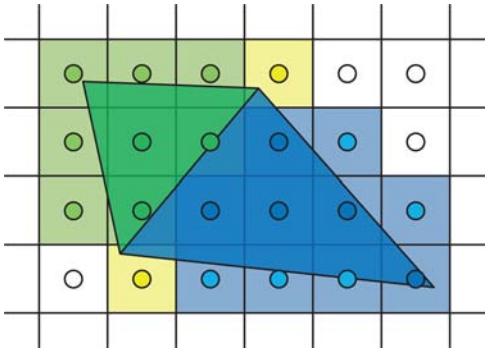

Figure 4: *Pixel to triangle mapping [PAMB10].*

which will determine the size of the final texture atlas. The user is able to create a texture atlas which suits the specific requirements of the application where it will be used later, or the transmission conditions.

To go from WC to ITC, we create a square grid in WC, which will represent 2D pixels, as well as a 2D array with the dimensions of the final image, which will store important information for the subsequent blending process: on which triangle t each pixel $\mathbf{p}$ lies. This is used for calculating α, β and γ, the barycentric coordinates of that pixel $\mathbf{p}$ with respect to that triangle t. Barycentric coordinates are very useful, as they represent any pixel $\mathbf{p}$ inside a triangle t as a convex sum of its three vertices $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$:

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \;\; (0 \leq \alpha, \beta, \gamma \leq 1; \;\; \alpha + \beta + \gamma = 1).$$

Besides, these same barycentric coordinates are also valid to express $\mathbf{P}$, the backprojected 3D point version of $\mathbf{p}$, in terms of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, the three vertices of the 3D triangle on which $\mathbf{P}$ lies. The triangle t on which $\mathbf{p}$ lies is determined by checking if the centre of $\mathbf{p}$ lies inside t. Partially covered pixels are also assigned to the corresponding triangle, whether their centre lies inside t or not. This differs from classic computer graphics approaches such as Bresenham's line drawing algorithm [Bre65], which discards less than 50% covered pixels when determining which ones belong to a line segment. In our case, however, since we do not want the background to be part of the texture, all pixels covered only partially by an edge must be completely assigned to the corresponding triangle.

Figure 4 illustrates how this algorithm works:

- pixels with a dark green/blue centre are assigned to the green/blue triangle because their centre lies on it;
- light green/blue pixels are also assigned to the green/blue triangle because they are partially covered by it;
- yellow pixels could be assigned to either the green or the blue triangle (it is just a matter of triangle processing order) because they are partially covered by both;
- white pixels do not intersect any triangle, so they are assigned to none.

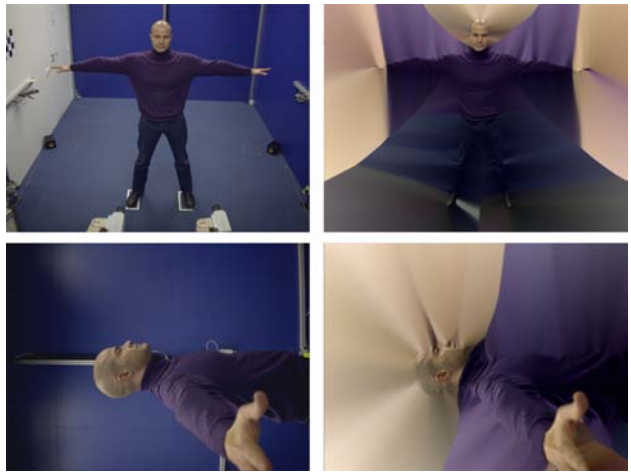

Figure 5: *Image pre-processing stage: input (left) and output (right) images.*

4.4. Image pre-processing

Due to the nature of the VH technique, the real 3D volume is a subset of the reconstructed one, so the resulting model is always slightly bigger than the original subject. This is even more noticeable in setups with a reduced number of cameras, and can lead to obvious texturing errors, since the background of the images can be wrongly included in the effectively used pixels of the texture atlas. Because of this, before starting to mix the colour information from the input images, we pre-process them. This makes our system more robust against occlusions and mesh inaccuracies, yielding perceptually correct texturing results in potentially problematic mesh regions.

We first erode the input foreground mask of each image to remove any remaining background pixels from the silhouette contour. Then, we apply the inpainting algorithm proposed by Telea [Tel04] over the area defined by the eroded mask, but to modify only the pixels labelled as background in the original (uneroded) mask (other inpainting techniques could be used for this purpose). Figure 5 shows the result of this pre-processing stage for two input images.

4.5. Pixel colouring

The most important step in our entire system is the one devoted to colour blending. As mentioned above, our system mixes information provided by different cameras, so it is very important to correct first the perspective distortion of each triangle, as seen by each camera, to avoid artefacts during blending.

We have used the scanline algorithm proposed by Wolberg [Wol90] which uses quadratic interpolation to avoid the non-linearity due to perspective. This way, we make sure the same 3D point is used when colour information from its projection by different cameras (i.e. from the corresponding pixels on different images) are combined, regardless of perspective distortion.

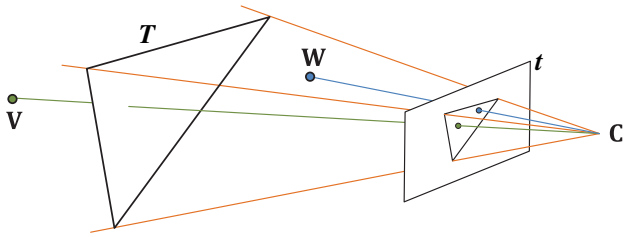

Figure 6: *Occlusion test is performed for a particular vertex (e.g.* **V** *or* **W***), given a particular triangle (T) and a particular camera position (***C***).*

Once unwarping is done, for every triangle T in the 3D mesh, we assign a rating to each camera C. We could initially think that the best criterion to rank the cameras is perpendicularity of their principal axes to T. However, since we have unwarped the triangles, we are not too concerned about perspective distortion, so a better rating is the area (number of pixels) covered by T in the unwarped image related to C. Moreover, before assigning the final rating, we perform two important tests: the first checks if T is facing away from C, and the second if any of the vertices of T is occluded by another section of the mesh. In any of these two situations, a null rating is assigned to C for T.

In our occlusion-checking algorithm, a (front-facing) triangle is occluded if any of its vertices is occluded by another triangle; and a vertex **V** is occluded by a triangle T in the image taken by C if the projection of **V** lies inside t (the projection of T by C), and the plane containing T lies between **V** and **C** (the position of C). Figure 6 illustrates the occlusion of a vertex by a triangle for a given camera: **V** is occluded from C by T, but **W** is not.

Note that our system allows the user to manually assign a higher rating to a specific camera to have it be more influential than the others in some particular scenarios. For instance, when using the multi-texturing technique described here within our humanoid reconstruction system, we usually increase the rating of the frontal camera capturing the face of the subject, but only for those triangles located in the facial region of the 3D mesh. This process is automatic, since we use a face detection system based on the approach proposed by Viola and Jones [VJ01] to determine which is the frontal camera.

If we were to calculate the final colour by mixing the RBG data provided by the best N cameras (according to the previously described rating), we would have a texture full of colour seams. Because of this, we need smooth transitions of the camera ratings along the mesh, which we obtain by calculating vertex-camera ratings once we have the triangle-camera ones. The vertex-camera ratings are the ones assigned to each vertex for each camera, and are calculated by averaging the triangle-camera ratings of all triangles sharing that particular vertex, much like vertex normals are typically calculated by averaging triangle normals.

To get the desired smooth transition of the camera ratings along the mesh, we perform linear interpolation of vertex-camera ratings inside each triangle, as shown in Figure 7 (note the similarity with the popular shading technique by Gouraud). This yields a rating r_{ij}

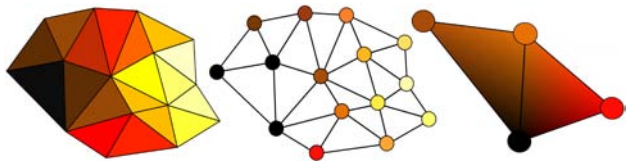

Figure 7: *Triangle- versus vertex-camera ratings [PBM13]. Left: ratings calculated for each triangle independently (note the abrupt transitions). Middle: ratings calculated for each vertex by averaging the ratings of surrounding triangles. Right: detail of the ratings calculated across two triangles with smooth transition.*

for each pixel p_i and camera j. Finally, we calculate the final colour Clr_i of pixel p_i as a weighted average of the colours provided by each camera for $\mathbf{P_i}$ (the 3D point corresponding to p_i), Clr_{ij}:

$$Clr_i = \left(\sum_j r_{ij} \cdot Clr_{ij}\right) \Big/ \left(\sum_j r_{ij}\right).$$

This last subprocess is of course the one which needs the most time and computer resources. However, since it is repeated for each pixel of the final texture atlas, we have also implemented a GPU version of it, which significantly helps to improve the performance of the algorithm.

4.6. Atlas dilation

Since most 3D renderers use bilinear or bicubic interpolation for texture mapping, they normally use a square pixel window which could wrongly include background colour information in their calculations when the window is travelling along patch frontiers. This would again result in visible seams, which is why, similarly to what we do in the input image pre-processing stage described in Section 4.4 (see also Figure 5), we include a final post-processing step to dilate the 2D patches obtained from the unwrapping system.

We have used a morphological extension process which uses a square structuring element to dilate the patch in all dimensions. The colour assigned to each new pixel added is calculated using nearest neighbour interpolation.

5. Results and Discussion

Although the technique presented in this paper was originally conceived for texturing 3D meshes representing humanoids, within an automatic avatar capture system, any reconstructed 3D mesh could be textured using a set of images of the model taken from different viewing perspectives.

5.1. Visual quality

The visual quality of the resulting textured models can be observed in Figures 8–14. For instance, Figure 8 shows the reconstructed model of a woman wearing a blouse with many creases in the back. Our system is robust enough to handle these creases, and yields smooth texture transitions between different mesh regions—in fact, there are no 'mesh regions' any longer. Another two critical cases are

Figure 8: *Seamless multi-texturing results for a woman's body. Note how the creases of her blouse are handled satisfactorily.*

Figure 9: *Seamless multi-texturing results for two men's bodies: complete 3D models with (left) and without (right) textures applied.*

Figure 10: *Details of the two textured 3D models from Figure 9. Top row: striped T-shirt from the first model, seen from the back (left) and front (right). Bottom two rows: facial section of the second model, seen from several viewpoints.*

illustrated Figures 9 and 10, which show the reconstructed models of two men: the first was wearing a striped T-shirt, which was correctly processed, avoiding aliasing (actually, in this case, the combination of stripes and creases is very well handled); as for the second, we have focused on the facial section of the 3D mesh, which is the most important perceptually (note how, again, there are no visible texturing seams, although the rating assigned to the frontal camera was much higher than those assigned to the other cameras).

Figure 11 shows the improvements obtained by applying the pre-processing stage described in Section 4.4, where the silhouette is extended (original errors are marked in red).

As any other multi-texturing system, ours needs to find a compromise between seam correction and blurring: the maximum resolution will be available when only one camera is used for texturing a certain area of the mesh; however, in that case, seams will probably appear all over the model. This problem is illustrated in Figures 12 and 13. The first one shows the difference between using just one camera for each triangle, which leads to abrupt colour transitions,

Figure 11: *Difference between applying and not applying the pre-processing stage (original errors are marked in red).*

Figure 12: *Maximum number of cameras to be used in the image blending process: one (top left), two (top right), four (bottom left) and nine (bottom right).*

and (at most) two, four and nine. As can be seen, there is almost no difference between the two latter cases. That is because the number of used cameras is just an upper bound and our system automatically discards cameras with a null or very low rating. In the second figure, we can see the effect of blurring when high-frequency components

Figure 13: *High-frequency elements: difference between using a maximum of one (top left), two (top right), four (bottom left) and nine (bottom right) cameras.*

Figure 14: *Three versions of the same model with different triangle counts: textured (top) and plain (bottom) 3D meshes.*

are present in the input images. Again, the system is able to detect the best number of cameras to minimize the blurring effect produced by the bilinear interpolation, which acts as a low-pass filter, while preserving correct blending between different areas of the mesh.

Another key asset of our system is its robustness with respect to polygonal resolution. Figure 14 shows the result of applying our multi-texturing process to a 3D mesh whose triangle count has been reduced using a decimation approach based on quadric error metrics. The texturing results themselves are as good for the very low quality 3D mesh as for the higher quality ones.

To test the performance of our algorithm, we compare its results with the results obtained from two different techniques proposed by Callieri, a multi-camera blending approach we already talked about in Section 3 [CCCS08], and another technique which textures each section of the mesh using just one single image [CCS02]. For this, we have used a very high resolution facial 3D model acquired using Beeler *et al.*'s [BBB*10] technique. This model has been rendered from four different view points to obtain four images which have been fed to the three texturing algorithms. Moreover, we have reduced the number of polygons that the original model had (3.7M) to 300k, 30k and 7k to be able to see the difference in performance in every scenario. Figure 16 shows the visual results when every model is rendered from a fifth different viewing perspective.

Figure 15: *3.4 million triangle 3D mesh acquired using Beeler's approach [BBB*10] and used for testing the performance of our algorithm.*

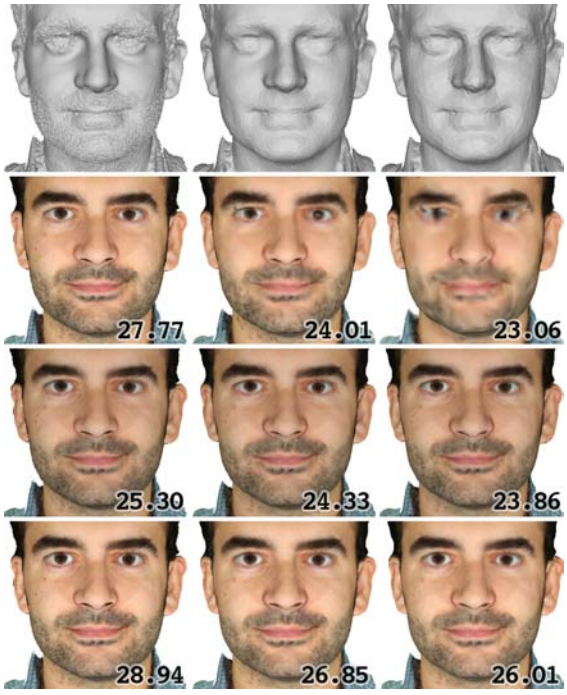

Figure 16: *Algorithm performance test. The first row shows the plain input mesh with 300k, 30k and 7k triangles (from left to right). Second row shows the three models textured using Callieri's Masked Photo Blending approach [CCCS08]. In the third row, models are textured using another technique by Callieri et al. [CCS02]. The last row shows our results. The value of the calculated PSNR in dB between the original model (in Figure 15) and each corresponding model is superimposed on each image.*

In the first row of Figure 16, we see the plain input mesh in each case: 300k, 30k and 7k triangles. The second row shows the result of Callieri's photo blending approach with a colour per vertex colouring scheme. The third row shows the results of texture mapping using just one image per section of the mesh. Finally, the fourth row shows the result applying our multi-texturing approach.

Figure 17: *Detail of the difference between the 7k triangle models textured with [CCS02] (left) and our technique (right).*

As most of the results are very similar, we have calculated the PSNR value, in the 2D image domain (pixel-wise), between the original mesh (shown in Figure 15, right) and the image obtained by each texturing approach (in Figure 16), also rendered from the same perspective. These values are superimposed on each case. As it is possible to see, our approach obtains the best PSNR values in every situation. When the polygonal resolution of the 3D model is decreased, a colour per vertex scheme is not a viable solution, so both texture mapping approaches get better results. All the models in the third row exhibit lower brightness and contrast due to the colour correction process applied by Callieri *et al.* [CCS02], which sacrifices colour accuracy to improve continuity across different regions. On the other hand, our per-pixel colour blending model corrects colour transitions without altering the general brightness of the model.

When observed from a close distance, the models which are textured with our system look less blurred. This is illustrated in Figure 17, where the eye detail is shown in one of Callieri's approaches and ours.

5.2. Computational cost

The first result of the multi-texturing technique we propose is an obvious reduction of information needed for texturing the 3D mesh, since instead of having one image per camera we use just one texture atlas for the whole mesh. This compact representation not only eases storage and transmission but also rendering because it does not require writing custom fragment shaders to process input from an *a priori* unknown number of original source cameras.

We have measured the performance of our system (the computer we ran our tests on was equipped with an Intel Core i7-2600K CPU with 16 GiB RAM, and an NVIDIA GTX 580 graphics card with 1.5 GiB RAM) to evaluate the impact on processing time of both the geometric complexity (i.e. the number of triangles) of the input 3D mesh and the size (i.e. number of pixels) of the output texture atlas. Figure 18 shows processing times for different geometric resolutions of the same model, and a constant output atlas size of 3 Mpx. As can readily be seen, the processing time for the unwrapping phase grows quadratically with the number of input triangles, while the processing time for the pixel colouring phase remains almost unaffected, showing that the complexity of the model is not so influential in the latter.

Conversely, Figure 19 illustrates the computational cost of the pixel colouring phase for different sizes of the output atlas, using a single input model. In this case, the processing time grows

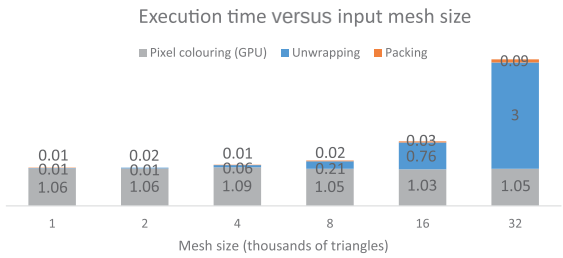

Figure 18: *Execution time (in seconds) versus input 3D mesh size (triangle resolution is successively doubled) for a constant texture atlas output size (3 Mpx).*

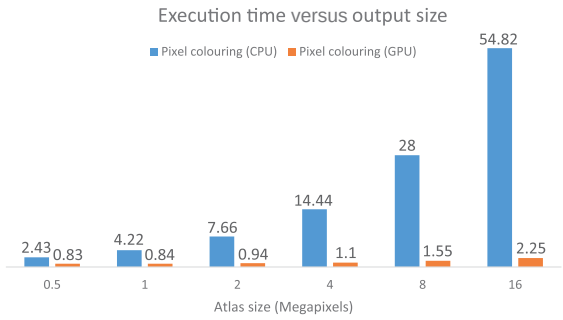

Figure 19: *Execution time (in seconds) of both the CPU and GPU implementations of our drawing algorithm versus texture atlas output size.*

linearly with the number of pixels of the output atlas. Besides, as we explained in Section 4.5, the pixel colouring step is the one which consumes the most processing resources and, as the colours of different pixels can be determined independently, it is a perfect match for a GPU implementation. Figure 19 shows a comparison of the CPU and GPU implementations of our system with different output atlas sizes, and leaves little doubts with regards to the benefits obtained thanks to GPU programming, since the time cost is significantly reduced.

6. Conclusions

We have developed an innovative, static multi-texturing system which can help save data in storage and/or transmission scenarios and, more importantly, yields good quality texture atlases. Our system overcomes all the classical problems that multi-texturing approaches face: it removes all noticeable seams due to texture misalignments, irregular illumination, and presents smooth colour transitions all over the whole texture atlas (and, therefore, all over the textured 3D mesh), while minimizing the blurring effect.

Our system deals with many 3D reconstruction problems inherent to VH-based techniques: the shape of the model is rarely reconstructed accurately, since concavities are normally missing and total volume is exaggerated. These issues make multi-texturing even harder than it already is. However, our system solves these problems and yields realistic multi-textured models. The robustness of our approach in non-optimal scenarios, where not very accurate 3D meshes are used to model the shape of the subject, is one of its most important features. Besides, when used for avatar reconstruction, our system also provides handles to give more texture resolution to the most important section of a humanoid: its face.

Our technique produces smooth colour transitions among different textured areas thanks to the nature of our colour blending approach, which takes into account the weighted contribution of several cameras in every triangle of the mesh. We can assume that there will always be one dominant camera in the most important regions of the mesh (in terms of number of triangles) and, in those cases, the contribution of the other cameras will be minimized, while the transition areas among these regions will have a more even camera contribution. This will help to preserve the original colours present in the original input images with a very smooth transition between them, also minimizing the blurring effect. However, in cases where the colour balance in these input images is very different, the colour difference between areas will be noticeable. This limitation of our system can be seen in Figure 12 (bottom row): although there are no colour seams between areas, we can see a different tone in the T-shirt, under the arm. This could be corrected by adding another pre-processing stage to the pipeline to equalize the colour balance of all input images, but this is something we wanted to avoid. Another possible solution could be assigning a lower rating to a certain camera when there is a very noticeable colour difference among images which should be similar.

Our system is specially good when the polygonal resolution of the input 3D model is not very high. On the other hand, in cases where the triangle count is very high (as the model presented in Figure 15) it would be more memory efficient to use a colour per vertex approach, which would be also interesting for a multi-resolution solution.

The techniques presented above have been used in a more complete 3D human model reconstruction system [PBM13]. Besides, we have used them as well to show how the results of two different 3D facial model reconstruction techniques (based on both passive [PAM11] and active [PM12] approaches) look after multi-texturing.

Acknowledgements

We would like to thank the people at Telefónica I+D Barcelona for their invaluable help in this project. This work was supported in part by the Ministerio de Economía y Competitividad of the Spanish Government under grants TEC2010-20412 and TEC2013-48453 (projects Enhanced 3DTV and MR-UHDTV), by the Spanish Administration Agency CDTI under grant 2007-1007 (CENIT-VISION) and by the European Commission under grant 610691 (BRIDGET).

References

[AMK10] Aganj E., Monasse P., Keriven R.: Multi-view texturing of imprecise mesh. In *Proceedings of Computer Vision–ACCV 2009* (Xi'an, China, 2010), Springer, pp. 468–476.

[APK08] Allène C., Pons J.-P., Keriven R.: Seamless image-based texture atlases using multi-band blending. In *ICPR 2008: Proceedings of 19th International Conference on Pattern Recognition* (Tampa, FL, USA, 2008), IEEE, pp. 1–4.

[Bau02] Baumberg A.: Blending images for texturing 3D models. In *Proceedings of BMVC* (Cardiff, UK, 2002), vol. 3, p. 5.

[BBB*10] Beeler T., Bickel B., Beardsley P., Sumner B., Gross M.: High-quality single-shot capture of facial geometry. *ACM Transactions on Graphics 29*, 4 (2010), 40:1–40:9.

[Bre65] Bresenham J. E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965), 25–30.

[CCCS08] Callieri M., Cignoni P., Corsini M., Scopigno R.: Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3D models. *Computers & Graphics 32*, 4 (2008), 464–473.

[CCS02] Callieri M., Cignoni P., Scopigno R.: Reconstructing textured meshes from multiple range RGB maps. In *Proceedings of VMV* (Erlangen, Germany, 2002), Citeseer, pp. 419–426.

[DMC*12] Dellepiane M., Marroquim R., Callieri M., Cignoni P., Scopigno R.: Flow-based local optimization for image-to-geometry projection. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2012), 463–474.

[DYB98] Debevec P., Yu Y., Borshukov G.: *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*. Springer, Vienna, 1998.

[Flo97] Floater M. S.: Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design 14*, 3 (1997), 231–250.

[GWO*10] Gal R., Wexler Y., Ofek E., Hoppe H., Cohen-Or D., : Seamless montage for texturing models. *Computer Graphics Forum 29* (2010), 479–486.

[HK09] Huang E., Korf R. E.: New improvements in optimal rectangle packing. In *Proceedings of IJCAI* (Pasadena, CA, USA, 2009), pp. 511–516.

[LI07] Lempitsky V., Ivanov D.: Seamless mosaicing of image-based texture maps. In *CVPR'07: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (Minneapolis, MN, USA, 2007), IEEE, pp. 1–6.

[LPRM02] Lévy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics 21*, 3 (2002), 362–371.

[MFNK95] Murata H., Fujiyoshi K., Nakatake S., Kajitani Y.: Rectangle-packing-based module placement. In *ICCAD-95: Proceedings of 1995 IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers* (San Jose, CA, USA, 1995), IEEE, pp. 472–479.

[MPMdCO11] Marroquim R., Pfeiffer G., Moura de Carvalho F., Oliveira A. A.: Texturing 3D models with low geometric features. In *Proceedings of 24th SIBGRAPI Conference on Graphics, Patterns and Images* (Maceio, Brasil, 2011), IEEE, pp. 1–8.

[PAM11] Pagés R., Arnaldo S., Morán F.: Face lift surgery for reconstructed virtual humans. In *Proceedings of 2011 International Conference on Cyberworlds* (Banff, ON, Canada, 2011), IEEE, pp. 249–253.

[PAMB10] Pagés R., Arnaldo S., Morán F., Berjón D.: Composition of texture atlases for 3D mesh multi-texturing. In *Proceedings of 2010 Eurographics Italian Chapter Conference* (Genoa, Italy, 2010), pp. 123–128.

[PBM13] Pagés R., Berjón D., Morán F.: Automatic system for virtual human reconstruction with 3D mesh multi-texturing and facial enhancement. *Signal Processing: Image Communication 28*, 9 (2013), 1089–1099.

[PFM11] Pagés R., Fuentes D., Morán F.: ITEM: Inter-texture error measurement for 3D meshes. In *Web3D '11: Proceedings of the 16th International Conference on 3D Web Technology* (Paris, France, 2011), ACM, pp. 31–37.

[PM12] Pagés R., Morán F.: 3D facial merging for virtual human reconstruction. In *Proceedings of 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video* (Zurich, Switzerland, 2012), IEEE, pp. 1–4.

[RCMS99] Rocchini C., Cignoni P., Montani C., Scopigno R.: Multiple textures stitching and blending on 3D objects. In *Proceedings of the 10th Eurographics Conference on Rendering* (Granada, Spain, 1999), Eurographics Association, pp. 119–130.

[SSGH01] Sander P. V., Snyder J., Gortler S. J., Hoppe H.: Texture mapping progressive meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (Los Angeles, CA, USA, 2001), ACM, pp. 409–416.

[Tel04] Telea A.: An image inpainting technique based on the fast marching method. *Journal of Graphics Tools 9*, 1 (2004), 23–34.

[VJ01] Viola P., Jones M.: Rapid object detection using a boosted cascade of simple features. In *CVPR 2001: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Kauai, HI, USA, 2001), vol. *1*, IEEE, pp. I–511.

[WKSS01] Wang L., Kang S. B., Szeliski R., Shum H.-Y.: Optimal texture map reconstruction from multiple views. In *CVPR 2001: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Kauai, HI, USA, 2001), vol. *1*, IEEE, pp. I-347–I-354.

[WLK*09] Wei L.-Y., Lefebvre S., Kwatra V., Turk G.: State of the art in example-based texture synthesis. In *Proceedings of Eurographics 2009, State of the Art Report, EG-STAR* (Munich, Germany, 2009), pp. 93–117.

[Wol90] Wolberg G.: *Digital Image Warping*. Wiley-IEEE Computer Society Press, Los Alamitos, CA, USA, 1990.