Eurographics Italian Chapter Conference (2010)
E. Puppo, A. Brogni, and L. De Floriani (Editors)

Composition of texture atlases for 3D mesh multi-texturing

R. Pagés, S. Arnaldo, F. Morán and D. Berjón[†]

Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, Spain

Abstract

We introduce an automatic technique for mapping onto a 3D triangle mesh, approximating the shape of a real 3D object, a high resolution texture synthesized from several pictures taken simultaneously by real cameras surrounding the object. We create a texture atlas by first unwrapping the 3D mesh to form a set of 2D patches with no distortion (i.e., the angles and relative sizes of the 3D triangles are preserved in the atlas), and then mixing the color information from the input images, through another three steps: step no. 2 packs the 2D patches so that the bounding canvas of the set is as small as possible; step no. 3 assigns at most one triangle to each canvas pixel; finally, in step no. 4, the color of each pixel is calculated as a smoothly varying weighted average of the corresponding pixels from several input photographs. Our method is especially good for the creation of realistic 3D models without the need of having graphic artists retouch the texture.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

We face the problem of automatically and efficiently mapping a texture onto a closed, manifold triangle mesh approximating the surface of a real 3D object, of which several pictures have been taken simultaneously by calibrated cameras surrounding it. The process of obtaining the 3D mesh itself is not of our concern in this paper. In the wider system for which we have developed the technique presented here, we follow a typical space carving approach [Kut00]: we first build a cubic visual hull from the object silhouettes in the different pictures, and then process it with the well-known marching cubes algorithm.

Our aim is to add realism to the 3D mesh through classic texture mapping, i.e., choosing or creating a texture, and assigning a pair of texture coordinates to each mesh vertex. Whatever the origin (synthetic or natural) of this texture, it has to be stored/transmitted along with the naked mesh topology and geometry, and it is thus beneficial that it be as small as possible. In order to reduce the texture size, graphic artists typically aggregate all images to be mapped onto different mesh regions into a single *texture atlas* containing information for all the triangles in the mesh: see Figure 1.

Figure 1: *Typical human-generated texture atlas*

As we explain in Section 2, where we review the state of the art related to our technique, there are many ways of aggregating the images, most of which imply unwrapping the mesh by portions, cutting it along certain edges and flattening its triangles so that they lie on the same plane. As we also explain in Section 2, most mesh unwrapping methods introduce some distortion when the 3D triangles are trans-

[†] e-mail: {rps,sad,fmb,dbd}@gti.ssr.upm.es

DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2010/123-128

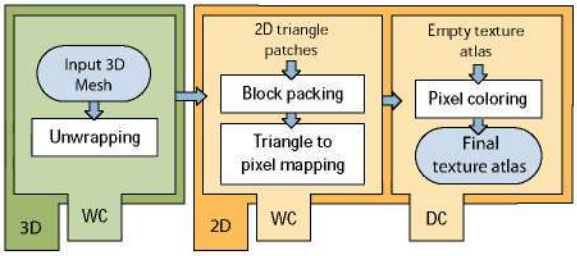

Figure 2: *Steps of our algorithm (WC: World Coordinates,* `float`*; DC: Device Coordinates,* `int`*)*

ferred to the 2D texture atlas, so that the number of cuts is minimized [SP05]. But this distortion can be avoided entirely to have the size of the 2D triangles in the texture atlas be proportional to that of the original 3D ones (preserving their angles). One of the features of the technique we propose is precisely that its 3D mesh unwrapping mechanism achieves zero distortion while being very efficient in doing so. The main contribution of our work, though, is the way in which we merge the color information from different pictures to calculate the value of each pixel in the texture atlas.

We start with a group of images corresponding to a set of views of the 3D object, so it is relatively simple to determine which camera sees best each mesh triangle, and therefore to establish a relationship between triangles and pictures. Unfortunately, if only one camera/picture is assigned to each triangle, seams across false mesh regions are almost guaranteed to appear in the rendered result, due for instance to different illumination conditions for different cameras. To solve this, we have developed a three step algorithm to optimize the texture that (see Figure 2): *i)* packs the 2D patches yielded by our unwrapping method so that the *bounding canvas* of the set is as small as possible; *ii)* assigns at most one triangle to each canvas pixel; *iii)* calculates the value of each pixel as a smoothly varying weighted average of the color information provided by different cameras, thus avoiding discontinuities in the textured 3D model.

2. Previous work

As mentioned above, unwrapping a 3D mesh is one of the most common approaches for the elaboration of a texture atlas where the information of several images is compiled. Typically, the mesh needs to be split into different portions, as it is done in paper crafts, before they can be unwrapped with the least error possible in what are called 2D patches or charts. The decision of where to cut the mesh usually requires the input of an experienced artist, although there are some methods to do it automatically. Of the many ways to approximate the surface of a 3D mesh with simpler planar surfaces to avoid complex patch shapes, some search for the minimum error [STL06] or use generalized cylinders represented by sets of triangles (not always taken from the mesh) [MGE08]. But the most common approach to mesh unwrapping with a given distortion is to use some kind of parametrization: for example, Floater's shape preserving parametrization [Flo97] leads to visually smooth surface approximations, whereas others seek to optimize the patches with respect to metric distortion or shape quality [KLS03]. The main problems of these approaches are the presence of the unavoidable (although limited) distortion, and the complexity of the mathematics behind the algorithms, which makes their implementations much slower than it is desirable.

Once the different mesh patches are obtained, it is important to lay them out on the texture canvas as efficiently as possible, while still avoiding overlapping. This is known as the NP-hard pants packing (or "tetris packing") problem [Mil98], and many solutions to it have been reported. It is possible to try and pack the different charts considering the space left between the curves defined by their (complex) shapes [LPRM02], but most approaches are based on packing the bounding rectangles of the patches. Algorithms achieving very good performances in terms of lost area minimization, but not as good in terms of computational resources, are described in [MFNK95, HK09]. An example of the specific use of block packing for texture mapping is described in [SSGH01].

Our most important contribution is perhaps the merging of color information from all pictures of the 3D object to compose the texture atlas. Since the cameras surround the object, which may not even be contained entirely in every photograph (e.g., it may be a human body, and some cameras may focus on its torso or face), it is necessary to combine them into the texture atlas. Most of our effort consists in dividing the mesh into different regions related to one camera/picture each. This was also done in [LI07] where the texture is back-projected onto the 3D surface before undergoing a Markov random field energy optimization process. This approach produces seams which are later mostly, but not always completely removed using a leveling process. In order to avoid any potential seams, a subsequent approach [GWO*10] uses additional flexibility for choosing the texture projection parameters, and solves possible problems in camera registrations or surface reconstruction. However, it is an iterative and very time consuming algorithm. Because of that, we decided to design an algorithm that fuses the information from different cameras, even within the same texture triangle. If we unwarp the images, it is possible to merge the color information of the cameras we prefer, so illumination and occlusion problems are easilly solved, thanks to an algorithm which, in fact, does not take much time.

3. Proposed technique

3.1. 2D patch creation

As already mentioned, the first step of our algorithm unwraps the original 3D mesh into 2D patches with no distor-

tion, meaning that the 2D triangles in the patches are exactly the same size as in the 3D mesh (for the moment, both 3D and 2D coordinates have floating point precision: we are still far from talking about pixels and integers). We have chosen this approach because it is easy to optimize, and therefore very fast. However, it produces bigger texture atlases with more patches, and therefore many potential seams. This could be regarded as a drawback but, in fact, it simplifies the mixing of color information from different photographs, which will be done later.

This algorithm is based on triangle adjacency information of the mesh, so it is necessary to calculate this first. Each new patch starts with a random *seed* triangle which is placed into the plane. After that, its neighbors are calculated and also placed in their position. This process is repeated with every triangle (marking each one as "used" so it is not placed twice), but it is necessary to establish several conditions to avoid intersections or irregular growth which could generate "octopus-shaped" patches.

Before a triangle is added to the patch, two conditions are checked. The first is that its edges do not intersect any edges of the triangles already in the patch. To detect an intersection, we check if the two new sides added to the patch corresponding to the new triangle intersect with any of the sides of the current patch perimeter. Any conflicting triangle is discarded and left for another patch. The second check is performed to prevent an inefficient patch growth. To do so, we consider a bounding box for each patch, defined by its minimum and maximum values for both the x and y coordinates. We also keep track of the total area covered by triangles inside this bounding box, so that at any point we are able to obtain the ratio between the area occupied by triangles and that of the box. The higher this ratio, the better, so, if adding a triangle makes it go under some threshold (which can depend on the number of triangles already added to the patch), the triangle is discarded to avoid patch boxes with big empty zones. If a candidate triangle does not fulfill both conditions simultaneously, it is discarded for the current patch. After all candidates of the patch boundary have been discarded, it is time to start a new patch with a new random *seed*.

3.2. Block packing

After examining all the solutions to the problem explained in Section 2, we decided to elaborate a simplified version of the rectangle packing algorithm where one dimension (e.g., width) is fixed, and the other is minimized. This approach may be less efficient space-wise, but it is much simpler and, therefore, faster. Figure 3 shows an example of a possible block packing result with our simplified algorithm.

3.3. Assigning triangles to pixels

The texture atlas is created by placing a (square) pixel grid onto the general bounding box and coloring each pixel. The size of each pixel cell in the grid is determined by an argument of the algorithm, therefore, it is possible to change the final resolution of the texture atlas depending on its future uses or the conditions of the transmission channel. In this section we explain how we assign a triangle T to each texture pixel centered at P. We need T to unambiguously back-project P onto a 3D point P_{3D} (lying on T), to then merge the colors captured by the different cameras at P_{3D} and calculate the final color assigned to the pixel. In the next section we explain the color averaging process.

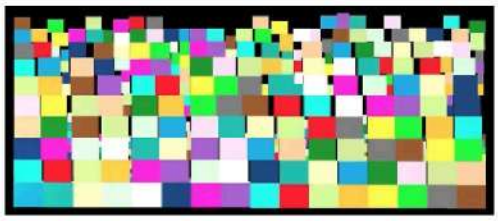

Figure 3: *Example of packing algorithm results*

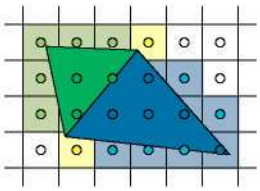

Figure 4: *Pixel to triangle mapping*

In general, we assign T to the pixel centered at P if P lies inside T, even if the pixel is only partially covered by T: see Figure 4. However, we sometimes find a partially covered pixel whose center does not lie inside any triangle, but whose edges intersect one or more edges of one or more triangles; we assign such a pixel the triangle which covers the largest area inside its square. And of course, there are completely uncovered pixels (the fewer, the better) that are not assigned any triangle. When a pixel has been assigned a triangle, it is convenient to express the position of P in barycentric coordinates, i.e., as a weighted average of the positions of the three vertices, A, B and C, of T: $P = aA + bB + cC$ (note that $a+b+c = 1$ is always true, but that $0 \leq \{a,b,c\} \leq 1$ holds true if and only if P lies inside T). Of course, the same equation, with the same barycentric coefficients a, b and c, is equally valid to express P_{3D} in terms of A_{3D}, B_{3D} and C_{3D}, the three vertices of the 3D triangle on which P_{3D} lies.

3.4. Pixel coloring

Coloring the final pixels of the texture atlas is the most important step in the whole process. Since not all cameras see necessarily P_{3D} equally well (in fact, some of them may not

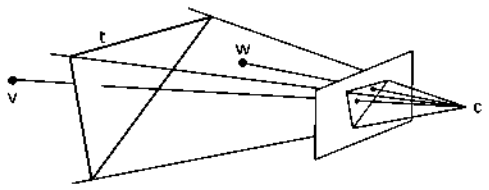

Figure 5: *Occlusion test*

see it at all), we need a way to rank them according to how appropriate they are to contribute to the color of the pixel centered at P. Below we explain how we establish the camera ratings, and then how we calculate the final pixel color through weighted averaging.

Prior to executing the camera rating algorithm, a vertex rating one must be run to fill in a matrix `vtx_ratings[nCam][nVtx]` (`nCam` being the number of cameras and `nVtx` that of vertices in the 3D mesh): `vtx_rating[c][v]` tells us how well camera `c` is suited to give us information about vertex `v`. To get there, we first calculate another matrix `tri_ratings[nCam][nTri]` (`nTri` being the number of triangles in the 3D mesh) giving the rating of every camera for every triangle. Using the calibration parameters of each camera `c`, each triangle `t` is projected onto the corresponding image. We measure the projection area in pixels, that is, how much information this camera "knows" about `t`. This area is calculated using the cross product of the vectors that represent two of the edges of `t`. The area is half of the magnitude of this vector, while its sense gives us information on whether the triangle is facing forwards or backwards. As in closed manifold meshes back-facing triangles are always occluded, we can assign a null rating right away to all triangles whose projected area is negative. However, front-facing triangles may also be occluded: our algorithm considers `t` (a front-facing triangle) to be occluded if at least one of its vertices is occluded by another triangle. In the same manner, a vertex `v` is occluded by a triangle `t` if the projection of `v` lies inside the projection of `t` and the plane containing `t` lies between `v` and the position of the camera `c` (see Figure 5).

For each triangle, we run this test for the vertices that lie inside of it in the projection. If the result of the test is that `v` is occluded (`t` is closer to the camera than `v`), then all its adjacent triangles are given a null rating (i.e., this camera cannot reliably provide information about them because they cannot be seen entirely). If the result is that `t` is occluded by `v`, then it is `t` that is given a null rating. To all the triangles that have not yet been given a null rating, we give a positive rating proportional to their area. Once the camera-triangle ratings have been obtained, the camera-vertex ones can be calculated: `vtx_ratings[c][v]` is the (unweighted) average of the ratings, for camera `c`, of the triangles sharing `v` (unless any of these ratings in null, in which case, `vtx_ratings[c][v] = 0`).

After the ratings are calculated, we loop over the texture pixels and, thanks to that matrix and to the barycentric coordinates of P, we obtain in a straightforward manner the rating of each camera for the pixel. Next, we select the n best cameras (n is a parameter of our algorithm and is typically chosen depending on the quality of the input images), $c_i \mid i \in [1;\ldots;n]$, with respective ratings r_i, and we calculate the final pixel color Clr as a weighted average of the colors provided by each camera for P_{3D}, Clr_i:

$$Clr = \frac{\sum_i r_i Clr_i}{\sum_i r_i}$$

To obtain Clr_i, we must take into account the calibration parameters of camera c_i to project P_{3D} onto the image captured by c_i, and then calculate Clr_i by interpolating bi-linearly or bi-cubically the colors of the nearby pixels.

3.5. Dilation of the 2D patches

If we check the texture after the previous steps, we will be able to notice a big amount of seams in the 3D model which are due to imperfections in the mesh extraction process and in the precision of texture coordinates. Because of that, in patch frontiers, the image background color is sometimes extracted instead of the the texture atlas color. To avoid this situation, a image dilation process as the one described in [Dou92] is used. A structuring element (in our case, a 3x3 pixel box) travels along the edge of the patch adding new pixels to the current patch. The color of each new pixel will be an average of the colors obtained from the nearest pixels in the patch. This process may have to be repeated several times until seams disappear (in our experience, five iterations have always been enough).

4. Results

The first obvious result of this process is a reduction in the amount of information needed to store or transmit the texture data. Besides, 3D renderers work much more efficiently when using a single texture image instead of many. Another advantage is the scalability of the output image: it can be adapted to different storage/transmission requirements. As for processing times, the one required for the unwrapping step is more or less linear with the number of vertices/triangles. This can be seen in Table 1, which presents the results for two sample meshes (see Figure 6). The test machine specs. are as follows: CPU: Core 2 Quad @ 2.66 GHz; RAM: 4 GiB DDR2 @ 667 MHz; Graphics card: nVidia GTX280 with 1 GiB of RAM. Our current implementation always runs the unwrapping step on the CPU, since it is not easy to parallelize it. Conversely, the "atlas painting" step does lend itself very well to GPU programming, so we have obtained very different results for the time spent in this step on CPU and

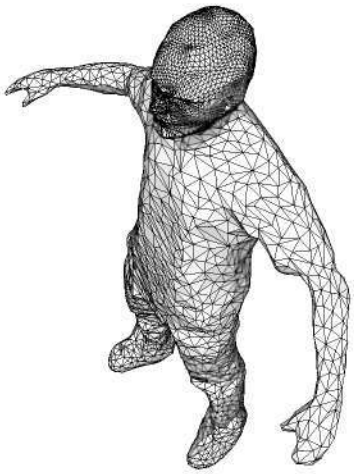

Figure 6: *Test mesh 2 (test mesh 1 is similar but has less triangles in the head area)*

GPU: when run on CPU, the atlas painting time depends linearly on the atlas size (number of pixels); when run on the GPU, there is an overhead cost for the transfer of information from the CPU to the GPU and back, and there is not a linear dependency. Despite this constant cost, GPU processing times are strikingly lower than those of the CPU. If we compare these time results with the ones in [GWO*10], it is possible to see a huge difference. While our algorithm takes less than one minute for a mesh with near 11000 triangles (see Table 1), their method needs seven. When the resolution is increased to 12000 triangles, our time remains nearly the same, while theirs is doubled.

Contrary to the method presented in [LI07], we combine the information of several images, which avoids alignment problems among texture sections. Thanks to that, visually, the results are stunningly precise, as the transitions between triangles are not noticeable and the color fading inside each triangle is also smooth. Figures 7 shows an example texture atlas resulting from our algorithm, and Figures 8 and 9 show different 3D meshes once the corresponding atlases have been mapped onto them. Notice that, although the face is split across several patches in the atlas of Figure 7, in the top mesh of Figure 9 the texture appears perfectly seamless.

5. Conclusions

Given the increasingly high demand of more realistic 3D models using textures coming from several actual photographs, we have developed the innovative, high definition multi-texturing system for 3D meshes presented in this paper. We combine and merge the color information provided by all images taken by a set of different calibrated cameras into a single texture atlas, so only one image has to be stored/transmitted to be mapped later onto the 3D triangle mesh approximating the surface of the object. The amount of time required to process all the images in order to compose the atlas depends of course on the spatial texture resolution (number of pixels) needed, but this delay is incurred once, and always off-line. Besides, this step, and especially the interpolation of the colors from the original images, are highly parallelizable, and therefore it is easy to increase their speed by using specialized processors such as GPUs. Moreover, this algorithm works perfectly with meshes with very high triangle resolution with the only consequence of an increase of the number of 2D patches.

Figure 7: *Automatically generated texture atlas*

Acknowledgements

Our work has been partially funded by the Spanish Administration agency CDTI under project CENIT-VISION 2007-1007. The authors also want to thank Mikel Fernández Oreja, Julien Quelen and their colleagues at Telefónica I+D in Barcelona for their invaluable cooperation.

References

[Dou92] DOUGHERTY E.: *An introduction to morphological image processing*. Spie, 1992.

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

	Mesh1: 2502 vertices, 5000 triangles						**Mesh2**: 5626 vertices, 11086 triangles					
Output size (Mpx)	3.16		7.03		12.47		2.23		4.96		8.87	
Mesh unwrap (s)	0.24						0.59					
Block packing (s)	0.02						0.06					
Pixel-triangle mapping (s)	0.52		1.11		1.94		0.38		0.80		1.38	
Drawing (s, on CPU/GPU)	16.14	0.30	35.56	0.43	62.51	0.57	11.58	0.29	25.24	0.37	44.16	0.49
Total (s, on CPU/GPU)	16.92	1.08	36.93	1.80	64.71	2.77	12.58	1.29	26.66	1.79	46.16	2.50
Output size (KiB, JPEG)	612		1043		1522		520		888		1299	

Table 1: *Results for test mesh 1 (2502 vertices, 5000 triangles) and mesh 2 (5626 vertices, 11086 triangles)*

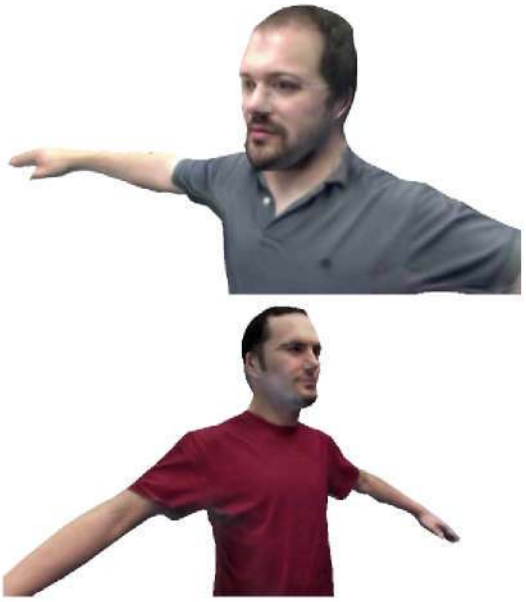

Figure 8: *Final textured meshes*

Figure 9: *Final textured meshes. Note the smoothness of the texture in the face or in the jacket folds.*

[GWO*10] GAL R., WEXLER Y., OFEK E., HOPPE H., COHEN-OR D.: Seamless montage for texturing models. In *Computer Graphics Forum* (2010), vol. 29, John Wiley & Sons, pp. 479–486.

[HK09] HUANG E., KORF R.: New improvements in optimal rectangle packing. In *Proceedings of the 21st international joint conference on Artifical intelligence* (2009), Morgan Kaufmann Publishers Inc., pp. 511–516.

[KLS03] KHODAKOVSKY A., LITKE N., SCHRODER P.: Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics 22*, 3 (2003), 350–357.

[Kut00] KUTULAKOS K.N. AND SEITZ S.: A theory of shape by space carving. *International Journal of Computer Vision 38*, 3 (2000), 199–218.

[LI07] LEMPITSKY V., IVANOV D.: Seamless mosaicing of image-based texture maps. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR'07* (2007), 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* (1995), Published by the IEEE Computer Society, p. 0472.

[MGE08] MASSARWI F., GOTSMAN C., ELBER G.: Paper-craft from 3D polygonal models using generalized cylinders. *Computer Aided Geometric Design 25*, 8 (2008), 576–591.

[Mil98] MILENKOVIC V.: Rotational polygon containment and minimum enclosure. In *Proceedings of the fourteenth annual symposium on Computational geometry* (1998), ACM, p. 8.

[SP05] STRAUB R., PRAUTZSCH H.: Creating Optimized Cut-Out Sheets for Paper Models from Meshes. In *Nineth SIAM Conference on Geometric Design and Computing, Phoenix, AZ, USA, October* (2005), vol. 30.

[SSGH01] SANDER P., SNYDER J., GORTLER S., HOPPE H.: Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, p. 416.

[STL06] SHATZ I., TAL A., LEIFMAN G.: Paper craft models from meshes. *The Visual Computer 22*, 9 (2006), 825–834.