A Streaming Framework for Seamless Detailed Photo Blending on Massive Point Clouds

Ruggero Pintus[1], Enrico Gobbetti[1], and Marco Callieri[2]

[1]Visual Computing Group - CRS4, Italy
[2]Visual Computing Group - ISTI-CNR, Italy

Abstract

We present an efficient scalable streaming technique for mapping highly detailed color information on extremely dense point clouds. Our method does not require meshing or extensive processing of the input model, works on a coarsely spatially-reordered point stream and can adaptively refine point cloud geometry on the basis of image content. Seamless multi-band image blending is obtained by using GPU accelerated screen-space operators, which solve point set visibility, compute a per-pixel view-dependent weight and ensure a smooth weighting function over each input image. The proposed approach works independently on each image in a memory coherent manner, and can be easily extended to include further image quality estimators. The effectiveness of the method is demonstrated on a series of massive real-world point datasets.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Digitizing and scanning—; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—.

1. Introduction

Modern 3D scanners have greatly increased their resolution and speed, making it possible to produce massive datasets in a very short time. This abundance of sampled points finds a perfect field of application in Cultural Heritage (CH), where both dense and extensive sampling is required. While a typical representation for such data has been triangulated meshes, in recent years there has been a renowned interest towards the direct use of point clouds, which are easier to create, manage and render. However, since a dense geometrical sampling is not enough to cope with all the needs in CH study and preservation, 3D models are often enriched with color information. Even if there are hardware solutions able to sample geometry and color altogether, their color quality and resolution are often deemed insufficient for CH. For this reason, it is necessary to rely on additional photographic datasets. In this case, one needs to register the photos with the 3D model and then to transfer color to geometry.

All currently proposed solutions for large models use triangle meshes, either obtained directly from the geometric acquisition procedure (e.g., the intrinsic range map structure), or after a triangulation of raw unstructured point clouds. Among these works, texture-based methods work well when we have high-resolution images but small meshes (few million faces), while techniques that deal with huge datasets (tens or hundreds of million faces) employ out-of-core multi-resolution data structures (both for geometry and images) and per-vertex mapping. The latter methods are scalable in terms of color and geometry complexity, but require vast amounts of processing (and pre-processing) time.

In this paper, we present an efficient scalable out-of-core streaming technique for mapping highly detailed color information directly on extremely dense 3D point clouds using a constant small in-core memory footprint. We work directly on a coarsely spatially-reordered point stream, which can be adaptively refined by the method on the basis of image contents. Seamless multi-band image blending at each input point is obtained by GPU accelerated screen-space operators, which solve point set visibility, compute a local per-pixel view-dependent weight and also ensure a weighting function that smoothly varies over each input image. To our knowledge, this is the first method capable to perform seamless image blending on massive unstructured point clouds. Our results show that the resulting system is fast, scalable, and

high quality. In particular, we are able to process models with several hundred million points and billions of pixels in times ranging from minutes to a few hours (see Sec. 10), outperforming current state-of-the-art techniques in terms of time while achieving the same high quality.

2. Related work

Image blending for texturing 3D models is a wide and extensively studied field with a lot of successful applications, especially in CH [DCC*10, HSKM10]. In the following, we discuss only techniques closely related to ours.

Triangulated meshes are the dominant 3D data representation in CH applications. However, the use of point clouds has recently gained a lot of attention, either because it is a sensible choice to explore in real-time huge datasets (e.g., as the Domitilla Catacomb [SZW09]), or because it allows peculiar shading with interactive manipulation of lighting (e.g., as the Dancers Column [DDGM*04]). The possibility to efficiently render these datasets by using the GPU [BK03, GM04] is certainly the main cause of this shift between representations. Given this tendency, and the fact that producing high quality triangulations from sampled point clouds is a costly task, the techniques that work directly on unstructured point clouds are becoming more and more important.

Despite this interest towards point clouds, in literature there is a distinctive lack of techniques for the color mapping which directly operates on point sets (in particular, on *massive* ones). Color mapping on point clouds passes in most cases through global or local triangulation of the dataset [PV09]. In other cases, the colored point clouds just contain the color returned by the scanner [SZW09], which mixes data from multiple scans in a single point cloud and generally produces unpleasant confetti-style color variations, unsuited for CH usage. Tools for the manipulation and editing of point clouds, such as PointShop [ZPKG02], typically just implement basic operations for color transfer between photos and the point set, and are not applicable to very large datasets that do not fit in main memory.

Most works on color mapping are targeting triangulated meshes, and deal in particular with the problems related to the integration of data coming from different photos. If a surface element (vertex or face) is viewed by more than one camera, there is a set of candidate pixels which contribute to its final color. Since a simple averaging of images produces color discontinuities, all recent papers compute a per-pixel weight to drive the per-vertex (or per-face) texture blending. They use different approaches such as surface orientation [BMR01, Bau02], texture discrepancy between adjacent triangles [XLL*10], viewing distance [CCCS08], number of texels per triangle [APK08], image-space distance to depth contours [CCCS08, Bau02], user defined stencil masks [CCCS08], photometric estimators [BMR01], etc.. Most of them are strictly dependent on a face-based representation, while some can also be applied to point clouds. Our weighting scheme is inspired by the masks of Callieri et al. [CCCS08], but uses a single screen-space operator, applied to the depth frame buffer, which incorporates the effect of many previously mentioned weights. Further, it ensures a smooth weighting function even in the presence of user-defined stencil masks, leading to continuous blends.

Various methods exist to manage the computed weights to blend images. Some techniques employ a *best fragment* approach using a *Markov Random Field* to assign a single image to each surface patch [GWO*10, XLL*10]. The resulting texture is optimized using gradient domain blending [PGB03] or other image-stitching methods. Some papers use global or local color correction applied to adjacent patches [XLL*10, BFA07, AF03]. Other methods perform a per-vertex blending with weighted averaging [CCCS08, BMR01]. All of these approaches could be used in a multi-band framework blending [BA83, APK08], where an image is decomposed in frequency bands and for each of them a different weighting function is applied. As in Baumberg [Bau02], we use a two frequency band blending, avoiding blurring or ghosting due to small misregistration of the cameras with the 3D model.

Most of the discussed techniques assume that both geometry and images fit in memory, which limits the scalability of the methods. Some methods assume, instead, that the 3D model is very small and fits in memory, while the image and mask dataset are stored in out-of-core structure (e.g., [XLL*10, CCCS08]). Callieri et al. [CCCS08] deal with massive models by using two out-of-core structures, one for the model and one for images and masks. However, random access traversals increase processing time. Our streaming framework efficiently blends images on massive point clouds in a low-memory setting and through memory-coherent operations, providing unprecedented performance. None of the mentioned techniques support, as we do, adaptive geometry refinement as a function of image content.

3. Technique overview

The proposed technique is outlined in Fig. 1. We take as input a point stream and a set of images with the corresponding camera parameters for inverse projection from image plane to 3D space. We start by coarsely reorganizing the stream in a spatially coherent order by making it follow a space-filling curve. For each image, we then perform the following operations. We render the point cloud from the camera point of view, using a rendering method that performs a screen-space surface reconstruction. We then estimate a per-pixel weight with a screen-space operator applied to the depth buffer and with the contributions of other possible masks (e.g., user defined stencil masks). Then, we extract an edge map from weights, compute a distance transform on it and multiply the weight by a function that has zero values on the edges and grows smoothly accordingly with the distance map. In this way, we ensure that the weighting function varies smoothly

over each image, avoiding color discontinuities for a seamless blending. We use these weights to project color from image pixels to points. For each image we perform two streaming passes, which exploit the spatial ordering for visibility culling. In the first pass the model is rendered in the depth buffer, while in the second pass we accumulate colors and update weights. A multi-band approach is adopted and different weights are computed for each frequency band. After all images are blended, we perform one last streaming pass to produce the final color for each point the final color by merging band contributions. For adaptive point cloud refinement, before applying the color blending algorithm, we traverse the entire pipeline once to update the geometry, by replacing the accumulation pass with the point population step (see Sec. 9).

4. Out-of-core Streaming Implementation

Projecting high resolution images on detailed models is a challenging problem. This is mainly due to a combination of ever increasing model complexity with the current hardware design trend that leads to a widening gap between slow data access speed and fast CPU and GPU data processing speed. This is particularly true when images and models are so large that they cannot be fully loaded in memory for processing. Developing efficient data access and management techniques is key in solving our problem efficiently.

In this work, we took the decision of sequentially processing images one after the other. I/O times are thus reduced, and main memory usage remains constant. Point clouds are, instead, processed using a streaming approach for all required operations, which are projecting the model to the depth buffer, accumulating colors and weights, and selectively refining the model on the basis of image contents.

The fundamental idea behind this approach is to always process data sequentially by reading from disk only a limited buffer at any time. This allows processing massive data sets very efficiently due to coherent memory-access and disk look-ahead pre-fetch policies. Moreover, the system is fully scalable, since at any given time only a fixed small fraction of the entire data set needs to reside in main memory. All our operations are applied only to points visible from the current camera. Thus, in order to reduce work, we reorganize the stream so that visibility culling can be efficiently performed while remaining within a streaming framework. This is done by coarsely reordering it along a space filling curve, and by rapidly skipping unnecessary sections while streaming.

Reordering occurs once during pre-processing, and requires only two additional streaming passes over the input point cloud. In the first pass, we coarsely estimate the local density of the cloud by using a small in-core regular grid of cubic cells that subdivides the bounding box of the input dataset and counting the number of points per cell. The size of a single cubic cell is heuristically estimated in advance from the maximum expected number of points per cell by $l = \sqrt{A\frac{M}{N}}$, where A is the side area of the model bounding cube, M is the approximate desired point count per cell, and N is the total number of samples. From this regular grid, we then estimate the output point stream layout, by traversing the grid in 3D Morton order, and sequentially allocating blocks of points in the output point cloud. In a second and final pass on the input point cloud, we fill the output point cloud, maintained as a memory-mapped array, by reading in chunks of points, lo-

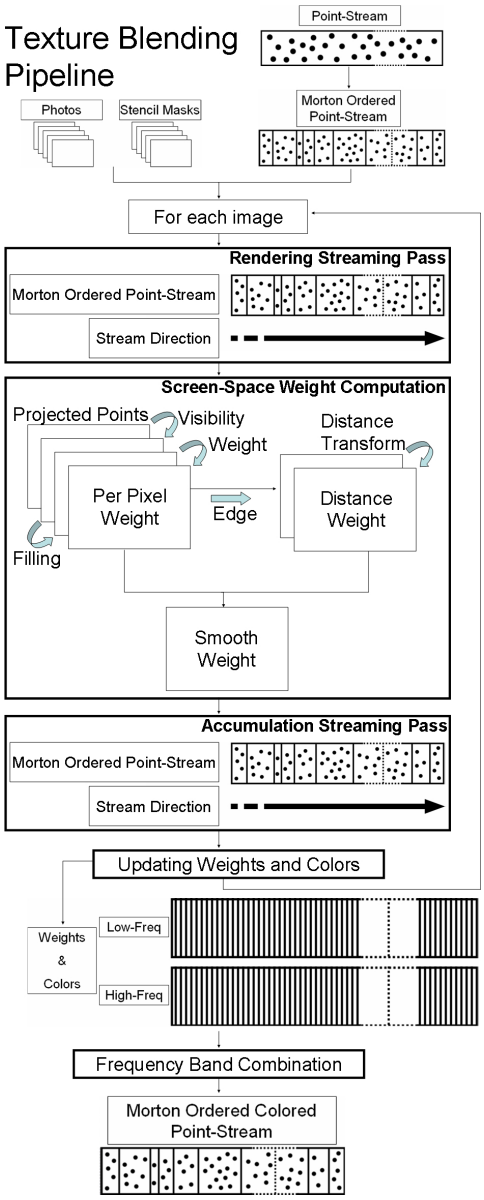

Figure 1: ***Algorithm Pipeline.*** *The inputs are a point stream and an image set. After a fast pre-processing, for each image we render and fill in the point cloud, compute a per-pixel weight, and accumulate color, updating temporary out-of-core arrays for each frequency band. In the last streaming pass, we combine band contributions.*

cally sorting them in Morton order [Mor66] to increase memory coherence, and writing them out at the correct output location.

The end result of this pre-processing step is a coarsely reordered point cloud, paired with an array of cell descriptors containing for each cell the cell bounding box, for visibility culling, the position of the first contained sample in the output array, and the number of contained samples. The 3D set of points is thus transformed in a one-dimensional spatially coherent set that increases the likelihood that neighboring cells produce the same visibility check output during streaming passes. This results in increased performance, since it removes unnecessary I/O and processing operations, while decreasing the number of jumps over the point stream, optimizing the disk look-ahead pre-fetch behavior.

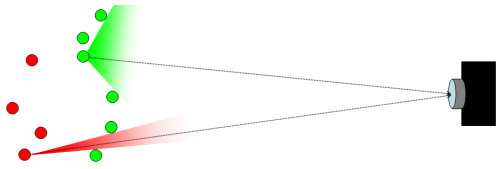

Figure 2: ***Visibility insight.*** *Visibility is related to the accessibility of a point with respect to the camera. Red dots are occluded points, while green dots are visible points.*

5. Point cloud rendering with screen-space surface reconstruction

In order to implement our image composition method using small and efficient screen-space operators, we need, as input, a screen-space representation of the surface defined by the point cloud. Given the position, orientation and camera parameters, we employ a point-based technique implemented in GPU that solves point visibility and fills the holes in the depth buffer between projected visible points. Visibility computation is based on the concept of "accessibility" of a point with respect to the camera. In Fig. 2 we show an example in 2D; the green dots represent the points with a "big" solid angle (visible points), while the red ones have a "small" solid angle (invisible points). For each frame pixel (background or projected) we check a fixed image-dependent neighborhood in the depth buffer to estimate if neighbor points occlude it or not. After the visibility pass, a sparse depth buffer has to be filled to obtain a smooth and usable representation. We employed a non-linear iterative method that is implemented in a multi-pass off-screen diffusion process and exploits a bilateral-filtering approach with a 3x3 kernel. We perform two different operations: screen filling uses the information from the visibility pass to compute the depth value of background points (holes between projected points), while filtering meaningfully blends the original and the reconstructed data (to eventually remove the noise coming from the previous filling). The main idea is to update depth values at each iteration with a weighted average of the neighbors using the following weights: $w_i = \left(1 - \frac{r_i}{2}\right)\left[1 - min\left(1, \frac{|z_i - z_0|}{\delta}\right)\right]$. The first is a radial term, while the other term is 1 if the central pixel and the pixel i have the same depth, while tends to zero if the difference in depth grows. δ is the tolerance used for considering two z values comparable, and therefore participating to blending.

Since background points do not have a valid initial depth, which is necessary for the filtering, we initialize them with the median of the neighbors. The choice of median instead of average comes from an edge preserving filling strategy. At the end, the output is a consistent filled depth texture.

6. Weighting function

The weighting function depends on two estimations; the former is a local per-pixel weight computation, while the latter (detailed in Sec. 7) is a distance map driven by edges extracted from the local weight. Both are implemented with fragment shaders. Fig. 3(left) shows the rendered point cloud we use to explain each step of our technique. It is a part of a church acquired with a time-of-flight laser scanner. In the second column there are the images we want to map and blend, while the images in the third one are the corresponding user defined stencil masks.

The local weight depends on the depth signal and requires a single GPU pass. For each projected point i, let us call P_i and P_i^S its eye and screen coordinates. The position of the camera P_C in eye coordinate is the origin. We look in a screen-space region of dimension K around point i and, for each pixel k in this neighborhood, we consider its P_k and P_k^S. Then we compute $w_i = \frac{\sum_{k=1}^{K} w_k^S w_k}{\sum_{k=1}^{K} w_k^S}$, where $w_k = 1 - |\vec{n}_c \cdot \vec{n}_k|$, with $\vec{n}_c = \frac{P_c - P_i}{\|P_c - P_i\|}$ and $\vec{n}_k = \frac{P_k - P_i}{\|P_k - P_i\|}$, and $w_k^S = \left[1 - \frac{\|P_k^S - P_i^S\|^2}{R^2}\right]^4$. The weight w_k^S is a screen-space radial modulation and the value R ensures that both all pixels inside the neighborhood have a non-zero contribution and the nearest point outside it has a zero weight. The term w_k incorporates in a single value the effects of the most used weight estimators. *Surface Orientation*: if the surface is orthogonal with the viewing direction all dot products are zero (we have the maximum value), while the sum of absolute values of the dot product ensures that the more the surface is tilted, the less is the weight value. *Silhouettes and Contours*: depth discontinuities result in very high dot products, so the per-pixel weight has low values over object silhouettes and depth buffer borders. Fig. 3(fourth column) shows the obtained weights. We could integrate them with other masks too (e.g., distance, focus, stencil, image border masks etc.). In our case, we apply user defined stencil masks (as in [CCCS08]); they are very useful to remove from photos objects unrelated to CH models (e.g., people visiting the site, scaffoldings etc.). The combination of weights and stencil masks is depicted in Fig. 3(right). If we use this weight to blend images, we obtain the colored model in Fig. 4 (top-left), which has color seams due to discontinuities in weight. These discontinuities comes from the

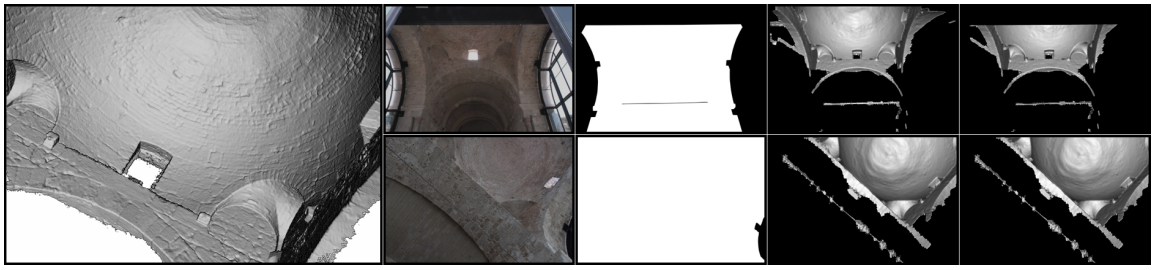

Figure 3: ***Weighting Function Computation.*** *From left to right: rendered point cloud; photos being blended; user defined stencil masks; per-pixel weights; masked per-pixel weights.*

stencil mask, occlusions, image boundaries, and non-smooth weight transitions over the geometry.

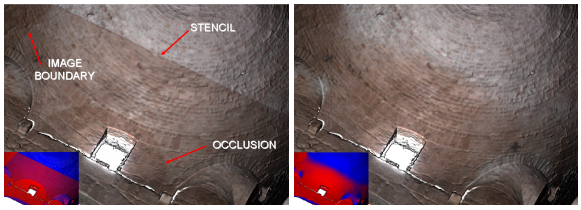

Figure 4: ***Photo Blending Strategy.*** *Photo blending of two widely different images acquired with different color settings and lighting conditions. On the left, masked per-pixel weights are used. On the right weights are smoothed with our method. Inset figures show the contributions of the two images, indicated in red and blue.*

7. Seamless blending through weight smoothing

To produce a seamless blending, the main idea is to find edges in the computed weighting function and to modulate weights over the image with a function proportional with the distance from those discontinuities. We first apply an edge extractor to find edges in the weight and we assign a zero weight to these pixels and a value equal to 1 to the others Fig. 5(top row). This is implemented in a single GPU pass. In order to take into account image boundaries, we set boundary weights to zero. For non-edge pixels we compute a quadratic function of the normalized distance from the nearest edge, using a multi-pass screen-space operator. We use a jump-based approach similar to Rong and Tan [RT06]. They perform a jump pass and some diffusion passes of step length 1 to decrease the residual error. We instead perform only two jump passes and in the second pass we set the distance as the less between the new and the old value. In this way the error will converge to zero and we ensure a smooth distance field Fig. 5(middle row). Then, we multiply normalized distance mask with the previous weight function. Fig. 5(bottom row) shows final weights. As showed in Fig. 4(right) this approach guarantees a weight sufficiently smooth to produce a meaningful and seamless color signal. To better explain the behavior of the proposed method, small sub-figures in Fig. 4 show the results of the corresponding blending strategy in the extreme case of two completely different images (red and blue).

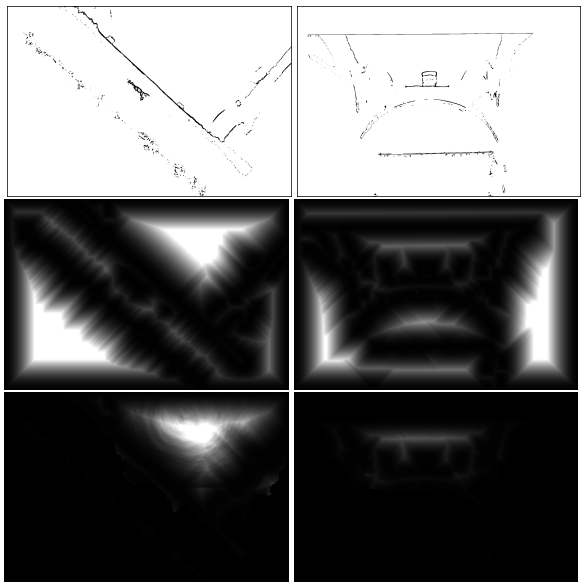

Figure 5: ***Smooth weight generation.*** *Rows from top to bottom: edge map; distance transform; per-pixel weight modulated with distance transform term. Figures are modified in brightness and contrast in order to enhance readability.*

8. Color accumulation and multi-band blending

We perform a streaming pass over the point set to project image color on the 3D model. For each sample we project its position in the depth buffer and, using a tolerance factor, we decide whether it is visible or not. If the point passes the visibility check, we accumulate the color information taken from the current image using the computed weight. We maintain, during the loop on the image set, two temporary values for each point: the weighted color and the sum of the previous computed weights. Since we adopt a multi-band blending, the accumulation is performed for each band separately. For this reason, we have to split the input image set in N_B sets, where N_B is the number of frequency bands, and, for each point, we need to maintain a vector of N_B tuples of color

and sum of weights. For each band we need to decide the weighting strategy too. Typically, linear filters are used for low-frequency bands, while non-linear or even *best-weight* approaches are used for high-frequency signals. At the end of M image projections we need an additional streaming pass to merge all the bands together. As in Baumberg ([Bau02]), we decide to apply a two band approach. We apply a blurring operator to obtain the low frequency image set, while each high frequency image is computed as the difference between the original and the low-frequency version. The computation of low and high frequency bands is fast performed in GPU. We use the weighting function w explained in Sec. 6 for the first set, while we use w^4 for the high-frequency band. As shown in Fig. 6, two band method avoids blurring or ghosting due to non-perfect alignment between images and point cloud.

Figure 6: ***Multi-band blending.*** *Comparison of single-band (lower triangle) and multi-band (upper triangle) blending. Multi-band blending avoids blurring due to small 2D-3D misalignment.*

9. Adaptive point cloud refinement

A major problem arises if the number of pixels in the image set is much greater than the geometrical sampling. In this case a lot of details in the image will be lost in the colored 3D model, since our approach produces a color per point and for each image we assign to a sample a weight and a color corresponding with one pixel only. If we look at a pixel neighborhood we could apply techniques similar to feature-preserving down-sampling to avoid missing high frequency components but, if the number of points is too low compared to the pixel sampling, the issue remains critical. Since during the off-screen rendering we build a non-sparse view-dependent surface representation of the model, we can re-project back the new filled points from the depth buffer to the object to increase point cloud local density. A brute-force approach is to re-project back all the points corresponding with valid pixels, such that the final model will have more points than the sum of all valid pixels; roughly speaking, no pixel color information is lost. This increases the number of points even if we don't need it, e.g., in the parts with a single, flat color. Our idea is to adaptively add points only for the color features. Exploiting multi-band computation, we add a point to the geometry if the corresponding high frequency pixel value is above a fixed threshold. Since we blend one image at a time we cannot re-project points and accumulate colors together in a single streaming pass, because each added point could be visible from an already mapped photo; this will produce a non-coherent image blending. If we want to add high frequency points we must perform the entire pipeline twice. The first time we substitute the accumulation pass in Fig. 1 with the re-projection and for each image we update the geometry in the out-of-core structure. After all images are considered, we re-launch the original algorithm to blend the textures on the new adaptively over-sampled point cloud. Fig. 7 shows how very fine details are better preserved if this adaptive point cloud refinement is adopted. A part of the model in Fig. 7(left) is critical because while in the high resolution images it was acquired a very narrow and long crack of the real model (highlighted in the red rectangle), the point cloud has a low local density. Fig. 7 (middle) shows both the row and the rendered point cloud after texturing in the case no refinement is applied; the crack is not continuous and in some parts it is barely visible. Our refinement technique adaptively adds points only across the crack, without loss of any segments and making geometrical features clearly readable Fig. 7 (right). The rendering of the point cloud only (in upper triangles) shows how we are able to reach high quality results in terms of fine detail preservation just adding a very low number of points.

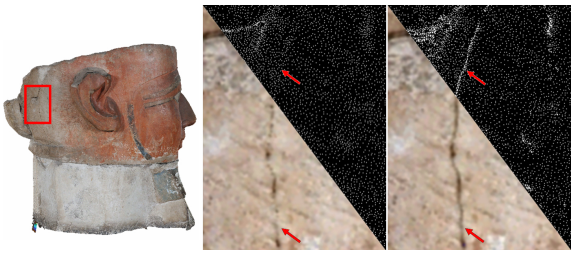

Figure 7: ***Adaptive point cloud refinement.*** *Left: complete model. Middle: image blending on original point cloud. Right: image blending on adaptively refined point cloud.*

10. Results

The proposed technique was implemented on Linux using C++ with OpenGL and GLSL. Our benchmarks were executed on a PC with an Intel 2.4GHz CPU, 4GB RAM, a 500GB 7200RPM Hard Disk and a NVidia GeForce GTX 460. To evaluate the effectiveness of our approach, in terms of both computational time and quality, we present results on massive 3D datasets mapped with a high number of images (see Table 9). *David* is a 3D model acquired with a triangulation laser scanner, while *Church* and *Archaeological Site* are models from time-of-flight laser scanner. All tests were executed with disk caches flushed.

Model	Resolution (# Points)	Images (width x height)	M pixels	No Morton	Morton	Tmp Disk Space
David	28M	67 (2336x3504)	548	5m21s	5m12s	960MB
Church	72M	162 (3872x2592)	1625	4h30m	47m30s	2.2GB
Arch. Site	133M	19 (3872x2592)	19	2h55m	1h43m	4GB
David	470M	67 (2336x3504)	548	21h6m	4h40m	14GB

Table 1: ***Dataset specifications and algorithm performances.*** .

Figure 8: ***Colored David 0.25mm dataset.*** *A 470 million points 3D model with color coming from a 548 Mpixel dataset (67 photos).*

The parameters guiding the method depend on the sampling rates of the point cloud and the image set. First of all, the size of visibility kernel and the number of filling iterations grow as the maximum ratio between pixel sampling rate and 3D sampling rate. In turn, since the filling operation, for each point, ensures smoothness inside a region that depends on the number of iterations, weight should be computed within a region of similar size. For the datasets presented here, we used very conservative settings corresponding to a 21x21 visibility kernel (and thus 21 filling iterations). Finally, the role of the edge extractor is to limit the maximum acceptable value of the normalized weight screen-space gradient, assigning a zero weight if the slope is higher than a threshold, which we set at the 10% of the highest possible difference between two weight values (i.e., 0.1).

Results obtained with *David* at 28M points (56M triangles) can be compared with those obtained with a similar hardware configuration and using the current state-of-the-art work dealing with massive models [CCCS08], which also maintains both the 3D model and the images out-of-core. They build a multi-resolution data that requires about 50 minutes of pre-processing and they randomly access this data during the blending; for this reason their approach takes about 15.5h (930 minutes) for the entire computation (pre-processing included). Their temporary files have a disk space occupancy of 6.2 GB. Our results clearly shows that our method outperforms previous solutions. The proposed streaming approach, by using space partitioning, results in a speed up of two orders of magnitude. It should be noted that models of this size, since our data structures are very compact, can be effectively cached by the I/O subsystem in memory, which is not possible for the more costly multi-resolution triangulations employed by Callieri et al.. The increase is, however, also evident for models which are much larger. E.g., our *David* 470M can be processed in less than five hours, while Callieri et al. [CCCS08] require over fifteen hours for the *David* 28M dataset.

Our visibility culling method, based on Morton ordering, is particularly valid for large datasets that do not fit in main memory, as in the cases of *Church*, *Archaeological Site* and *David* 470M. With these datasets we have a large fraction of time needed to traverse the point stream, so the precomputed Morton-ordering helps to increase efficiency in both rendering the geometry and accumulating colors. The *Church* dataset (Fig. 4) is a point cloud with 72M points and 162 photos (more than 1.5G pixels) and it takes 2 minutes and 14 seconds for pre-processing and 45 minutes 12 seconds for texture blending with visibility check; this is about the 17% of the 4 hours and 30 minutes needed for the computation without visibility check (no Morton-ordering). The computation requires temporary disk occupancy of 2.2GB. The *Archaeological Site* model (Fig. 6) has 133M points and we project 19 images of 10M pixels in 1 hour and 43 minutes (about 13 mins. pre-processing and 1.5h blending); the size of temporary files is 4GB. The blending without Morton-ordering and visibility culling takes about 3 hours. The biggest model we have tested is the *David* with 470M points and 67 images (548M pixels). Its size on disk is about 18GB for the input and output point cloud, 212MB for the images and about 14GB for temporary files. The pre-processing time needed to order the point set is about 17 minutes while the blending time is about 264 minutes; the computation without ordering and visibility checks requires 21 hours, about five times more.

Representative examples of the quality of the photo blending are presented in Fig. 8, which depicts some details of the *David* 470M dataset. Finally, Fig. 9 illustrates another example of high quality blending using adaptive point cloud

refinement. In this figure, we show another critical part of the same 3D model in Fig. 7. Here the low resolution point cloud results in bad feature preservation while, in the refined textured model, we add some points in the color contours (high frequency pixels) to achieve a meaningful representation of silhouettes (e.g., non-noisy iris boundaries). The original point cloud in Fig. 7 and Fig. 9 has 800Kpoints and the image set contains more than 60Mpixels; a brute force refinement (re-project all valid pixels in the image set) would add 13.3Mpoints, while the result of our adaptive method has only 700Kpoints more than the original. Thus, we get a highly detailed colored model saving 94% of the points.

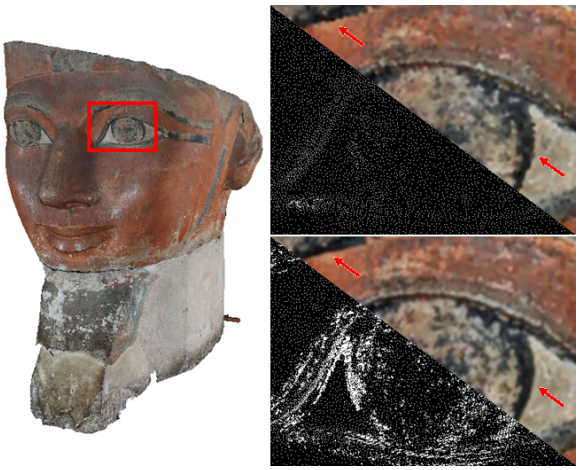

Figure 9: ***Adaptive point cloud refinement.*** *Left: complete model. Middle: image blending on original point cloud. Right: image blending on adaptively refined point cloud.*

11. Conclusions and future work

We have presented an efficient scalable streaming technique for mapping highly detailed color information on extremely dense point clouds. Our seamless multi-band image blending method works directly on a coarsely spatially-reordered point stream and can adaptively refine point cloud geometry on the basis of image content. It works independently on each image in a memory coherent manner, and can be easily extended to include further image quality estimators. Currently, photo blending is applied as an off-line processing task. Since blending of single images is relatively fast, we are extending the system by integrating it within the same interactive application used for alignment. This would allow fast visual checks of the quality of the result, and could help with image registration. In particular, by analyzing the correlation of images in areas of multiple overlap, it should be possible to guide the system in the alignment phase.

Acknowledgments. This research is partially supported by EU FP7 grants 231809 (3DCOFORM) and 242341 (INDIGO). Geometric data courtesy of Digital Michelangelo project.

References

[AF03] AGATHOS A., FISHER R. B.: Colour texture fusion of multiple range images. In *3DIM* (2003), pp. 139 – 146.

[APK08] ALLENE C., PONS J.-P., KERIVEN R.: Seamless image-based texture atlases using multi-band blending. In *ICPR* (dec 2008), pp. 1 –4.

[BA83] BURT P. J., ADELSON E. H.: A multiresolution spline with application to image mosaics. *ACM ToG 2*, 4 (1983), 217–236.

[Bau02] BAUMBERG A.: Blending images for texturing 3d models. In *British Machine Vision Association Conf.* (2002), pp. 404–413.

[BFA07] BANNAI N., FISHER R. B., AGATHOS A.: Multiple color texture map fusion for 3d models. *Pattern Recogn. Lett. 28* (April 2007), 748–758.

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern gpus. In *Pacific Graphics* (Oct. 2003), pp. 335–343.

[BMR01] BERNARDINI F., MARTIN I. M., RUSHMEIER H.: High-quality texture reconstruction from multiple scans. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (Oct./Nov. 2001), 318–332.

[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 (Aug. 2008), 464–473.

[DCC*10] DELLEPIANE M., CALLIERI M., CORSINI M., CIGNONI P., SCOPIGNO R.: Improved color acquisition and mapping on 3d models via flash-based photography. *J. Comput. Cult. Herit. 2* (2010), 1–20.

[DDGM*04] DUGUET F., DRETTAKIS G., GIRARDEAU-MONTAUT D., MARTINEZ J.-L., SCHMITT F.: A point-based approach for capture, display and illustration of very complex archeological artefacts. In *VAST 2004* (Dec. 2004), pp. 105–114.

[GM04] GOBBETTI E., MARTON F.: Layered point clouds. In *Eurographics Symposium on Point Based Graphics* (2004), pp. 113–120.

[GWO*10] GAL R., WEXLER Y., OFEK E., HOPPE H., COHEN-OR D.: Seamless montage for texturing models. *Computer Graphics Forum 29*, 2 (2010).

[HSKM10] HANKE K., STOELLNER T., KOVACS K., MOSER M.: Combination of different surveying methods for archaeological documentation: the case study of the bronze age wooden chest from mitterberg. In *CAA*. (2010).

[Mor66] MORTON G. M.: *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*. Tech. rep., 1966.

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Trans. Graph. 22*, 3 (2003), 313–318.

[PV09] PU S., VOSSELMAN G.: Building facade reconstruction by fusing terrestrial laser points and images. *Sensors 9*, 6 (2009), 4525–4542.

[RT06] RONG G., TAN T.-S.: Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proc. I3D* (2006), pp. 109–116.

[SZW09] SCHEIBLAUER C., ZIMMERMANN N., WIMMER M.: Interactive domitilla catacomb exploration. In *VAST09* (Sept. 2009), Eurographics, pp. 65–72.

[XLL*10] XU L., LI E., LI J., CHEN Y., ZHANG Y.: A general texture mapping framework for image-based 3d modeling. In *ICIP 2010.* (2010), pp. 2713 –2716.

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: an interactive system for point-based surface editing. In *SIGGRAPH '02* (2002), ACM, pp. 322–329.