Abstract—Three-dimensional colored models are of great interests to many fields. With the growing availability of inexpensive 3D sensing systems, it is easy to obtain triangular mesh and multiview textures. These range and vision data can be fused to provide such 3D colored models. However, low-cost sensing generates various noise components involving low-quality texture, errors in calibration and mesh modeling. Our primary objective is to establish high-quality 3D colored models on the basis of mesh and textures, while considering the noise types and characteristics. In this paper, we contribute in two ways. The first contribution is a point-based algorithm to color 3D models, where 3D surface points are used as primitives to process and store color information. The algorithm features three novel techniques: (a) accurate depth image estimation, (b) adaptive 3D surface point upsampling and (c) texture blending using those points. The algorithm provides colored models as dense colored point clouds, which can be rendered with various standard techniques for visualization. Our second contribution is an algorithm for textured model rendering, where blended textures are generated and mapped onto the mesh. The experimental results show that our algorithms efficiently provide high-quality colored models and enable visually appealing rendering, while being tolerant to errors from data acquisition. We also quantify the efficiency of our point upsampling algorithm with novel metrics assessing the influence of the 3D points.

Index Terms—data fusion, 3D colored model, texture blending, rendering, upsampling, multiview

I. INTRODUCTION

Colored 3D models of real-world environments are of high relevance to many applications including serious gaming, professional civil engineering and 3D printing. Recently, there is a growing availability of low-cost sensing and reconstruction platforms which output a triangular mesh together with multiview textures, to describe the geometry and appearance of a scene. The fusion of range and vision data to provide colored models is challenging due to various errors from inexpensive sensing devices. First of all, textures are usually of low quality, which is caused by the limited camera resolution, slight motion blur and lens distortion. Secondly, it is difficult to obtain perfect intrinsic and extrinsic calibration for camera viewpoints due to electronics tolerance and processing delays. Lastly, the triangular mesh contains a modeling inaccuracy because of the limitations in sensor quality and scanning procedures. With these errors in mind, we describe here a processing pipeline to generate visually appealing colored models based on a triangular mesh and multiview textures.

In computer graphics, texture mapping is one of the most well-known techniques to generate colored models [1]–[7]. This technique assigns a texture patch to every triangle or face by defining a one-to-one mapping from 2D image to 3D mesh. For meshes with large-sized triangles, such mapping efficiently creates textured models. However, texture mapping leads to an inflexible structure that usually requires remapping for changes of the mesh. Due to the separation between geometry and appearance, it is difficult to use the color information for other processing such as object segmentation and classification. To obtain a colored model with more flexibility, the appearance can be directly associated with the geometry by giving color to mesh vertices. However, mesh vertices are normally not dense enough to present all appearance information. Recently, a novel structure has been proposed called mesh colors [8], where color is not only given to vertices, but color samples are also interpolated for edges and faces such that their appearance can be better presented even with large triangles.

When using multiview textures to color a mesh, two issues need to be addressed. The first issue is to identify the visible triangles with respect to a viewpoint. This visibility problem can be solved either by ray casting [9] or Z-buffering [10], [11]. An efficient method that highly simplifies the visibility problem is to employ a depth image, which indicates the distance from the visible surface to the viewpoint. Recently, some capturing devices provide a depth image in supplement to an RGB image. When depth images are not available, they can be generated from the 3D model. The second issue is the color differences of objects captured from different viewpoints, due to lighting conditions and camera settings. Consequently, undesirable color changes are created in the model when the object is colored from different views. Texture blending is commonly used to smoothen these color changes. In literature, most blending algorithms are developed for texture mapping. In [1], [2], triangle-based blending algorithms are proposed to smoothen adjacent triangles that are assigned to different viewpoints. The blending performance is highly sensitive to the triangle size, where small triangles result in insufficient smoothing and large triangles lead to significant blurring. To avoid this problem, pixel-based blending is developed [3]–[6]. This method synthesizes textures, where pixels from different viewpoints that correspond to the same point in geometry are blended. The quality of the colored model is highly dependent on the resolution of the textures and their selected viewpoints.

To summarize, most existing algorithms for coloring 3D meshes employ texture mapping, which gives an inflexible structure and the related texture blending does not always guarantee decent coloring. In this paper, we explore an al-
ternative to create colored models. Using 3D surface points as processing primitive, we propose a point-based algorithm to generate colored models where the appearance is directly associated to the geometry. Three techniques are developed for our algorithm including depth image estimation, adaptive point upsampling, based on mesh colors and point-based texture blending. The output of our algorithm is a dense colored point cloud maintained in a mesh structure, which gives great flexibility at the rendering stage. Furthermore, we also develop an algorithm to generate textures from the colored point cloud for general texture mapping. The experimental results show that our algorithms yields visually appealing colored models, while being resilient to errors in low-quality textures, camera calibration and mesh modeling.

II. SYSTEM OVERVIEW

In this work, we assume the input consists of a triangular mesh and a set of multiview textures with calibration parameters. Figure 1 depicts an example of an input dataset. Our processing pipeline contains two stages: colored model generation and photo-realistic rendering as shown in Figure 2.

For colored model generation, we propose a point-based algorithm, where 3D surface points are used as primitive to process and associate color information. These points come from vertices of the input mesh as well as from upsampled surface points based on the mesh. Each point has a \((x, y, z)\) coordinate and is assigned an \((R, G, B)\) color value after processing. Consequently, the output colored model is represented as a colored point cloud maintained in a mesh structure. Three successive steps are performed to color the model: (1) a depth image is estimated for each texture to identify visible triangles; (2) points are adaptively upsampled using the structure of mesh colors, so that the appearance is properly represented; (3) multiview textures are analyzed and blended to find an appropriate color for every point. A detailed description of our point-based algorithm is presented in Section III.

To render the obtained colored model, four standard techniques in the current graphics pipeline can be used for visualization. The first option is mesh-based rendering with Phong shading [12], which requires a refined mesh reconstructed for the upsampled point cloud. Alternatively, point-based rendering [13], [14] can be applied to avoid extra triangulation. The third option is mesh colors rendering as described in [8], since our point upsampling is based on mesh colors. Finally, general-textured-model rendering can be used. For this purpose, blended textures first need to be generated from the colored point cloud and then mapped to the mesh. This rendering technique enables efficient visualization for a mesh with large triangles. The related discussion for textured model rendering is presented in Section IV.

III. POINT-BASED COLORED MODEL GENERATION

In this section, we present our point-based algorithm to generate 3D colored models. We store color information by surface points and provide appropriate coloring via blending. Our algorithm is robust to errors from texture deficiencies, camera calibration and mesh modeling. Let us now describe the three processing steps in more detail.

A. Visibility Test

In order to efficiently determine the visible triangles with respect to a viewpoint, we create a depth image based on the input mesh for each viewpoint with the same resolution as the texture image. The value of each depth pixel represents the distance of the corresponding surface point to the camera. For pixels that do not correspond to any surface points, their depth are infinite.

A naive depth estimation can be performed by projecting mesh vertices to the image and store the shortest distance. This solution only produces a sparse depth image, since usually vertices are not dense enough. The result is also error-prone, because it fails to detect vertices blocked by a face whenever their projections lie inside the projection of the face.
To obtain an accurate depth image, we project triangles instead of single points. This allows us to use interpolation to fill the depth value for any pixels inside the triangle projection. As a bonus, we avoid any points behind the visible surface to be overlooked and thus detect such points when they appear. First, we initialize all pixels with an infinite depth value. Then, for each triangle in the mesh, the vertices are projected to the depth image. A potentially visible triangle is identified when the projections of all vertices are inside the image. For such a triangle, the depth of any pixel inside the projected triangle is calculated by barycentric interpolation. The pixel value is updated with a Z-buffering technique such that the smallest depth value is kept. Figure 3 shows the depth image generated by our algorithm in comparison to the depth image produced by projecting points. For visualization, the depth values are scaled to \([0, 255]\), where invalid pixels with infinite depth are colored black.

### B. Adaptive Point Upsampling

In our colored model generation, we associate appearance to 3D points. However, the amount of vertices in a mesh is normally not sufficient to store all necessary color information. This can cause significant loss of details, where any interpolation using vertex colors fails to restore full appearance. To reduce this problem, we upsample points before coloring.

Naturally, the projected area of a triangle directly influences the amount of information represented by the triangle. The larger the projected area, the more pictorial details are contained within that triangle. It is therefore reasonable to upsample points proportional to the size of the projected area. Usually, the projected area varies with respect to the triangle size, the camera position and orientation. With these observations in mind, an adaptive point upsampling is needed and the mesh colors structure is suitable for our purpose.

The mesh colors structure is an extension of vertex colors, where colors are not only assigned to vertices, but color samples are also added to edges and faces. It resembles a texture patch, except that the colors are directly associated to the geometry. For the triangular mesh, a resolution factor \(R\) is specified for each triangle such that \(R - 1\) points are added to each edge and \((R - 1)(R - 2)/2\) points are added to the face. Figure 4 depicts some examples of mesh colors structures with different resolution factors. All points belonging to a triangle are evenly spaced, so that they are easily computed using barycentric coordinates with respect to the three vertices. Let point \(p_{ij}\) belong to a triangle with \(0 \leq i \leq R\) and \(0 \leq j \leq i\), its barycentric coordinates are given by \(\lambda_{ij} = (i/R, j/R, (1 - i - j)/R)\). All points on the triangle can then be calculated by a coordinate matrix \(\Lambda\), with

\[
\Lambda = \begin{bmatrix}
\Lambda_{00} & \Lambda_{01} & \cdots & \Lambda_{0(R-1)} & \Lambda_{0R} \\
\Lambda_{10} & \Lambda_{11} & \cdots & \Lambda_{1(R-1)} & \cdot \\
& & \ddots & \ddots & \ddots \\
& & & \ddots & \ddots \\
\Lambda_{(R-1)0} & \Lambda_{(R-1)1} & \cdots & \cdot & \cdot \\
\Lambda_{R0} & \cdot & \cdots & \cdot & \cdot \\
\end{bmatrix}.
\]

In accordance to Figure 4, we denote the coordinates of vertices, edge points and face points with blue, red and green, respectively. The matrix \(\Lambda\) is an upper triangular matrix, where \(\Omega = (0, 0, 0)\) are coordinates of non-existing points.

For adaptive upsampling, we determine the resolution factor \(R\) for each triangle according to its maximum projected area \(\alpha_m\) among all visible textures. The number of pixels inside the projected triangle serves as an estimation for the projected area. The resolution \(R\) is then specified by \(R = \lceil \sqrt{\alpha_m} \rceil\) with \(R_{\min} = 1\). This policy ensures that any three closest color samples that form an equilateral triangle, encompass at most one pixel. Consequently, we approximate the least amount of points required to store the necessary appearance. In our system, upsampled points and original mesh vertices are stored as one single point cloud and referenced to triangles by indices. They can also be arranged in the format required by rendering of the mesh colors structure.

### C. Color Blending and Assigning

As discussed in Section I, texture blending is necessary to reduce undesirable color changes when using multiview textures. The unappealing coloring is illustrated in Figures 5(a) and 5(b), where color values are selected by the closest view distance and smallest view angle, respectively. In addition to abrupt color changes, the errors from imperfect camera calibration as well as the missing objects structure from inaccurate mesh modeling lead to wrongly projected color values.

With these observations, we develop the point-based blending to achieve better coloring. First, color samples are collected for every point from visible viewpoints. When projecting a point to a texture image, bilinear interpolation is used to calculate the color value. Second, color outliers are removed.
by statistical analysis. For this, we compute the baseline color \( \mathcal{C}_M = (R_M, G_M, B_M) \) as the median of all color samples for each individual color channel. The color distance between a color sample \((R_i, G_i, B_i)\) and the baseline color is defined by

\[
d_C = \max\{ |R_i - R_M|, |G_i - G_M|, |B_i - B_M| \}.
\]

Any color sample with \( d_C \) larger than threshold \( d_T \) is considered to be an outlier. By removing outliers, we reject most of wrong color values resulting from calibration and modeling errors. In our experiment, \( d_T = 50 \) yields good results. Last, the blended color is computed as a weighted average of the remaining valid color samples. We have developed a new empirical weight for the valid color sample, which is inserted in a blending function. Given \( N \) valid color samples, the blending function is defined by

\[
\mathcal{C} = \frac{\sum_{i=1}^{N} (\cos \theta_i |/d_i) \times C_i}{\sum_{i=1}^{N} (\cos \theta_i |/d_i)},
\]

where variable \( i \) denotes the \( i \)th color sample, \( \theta_i \) is the angle between the point normal and camera viewpoint, \( d_i \) is the distance to the viewpoint and vector \( C \) represents the \((R, G, B)\) color value. The point normal is estimated with local least-squares plane fitting. With this blending function, we assume a color sample is more reliable when it is closer to the camera and the local surface is more orthogonal to the viewpoint.

Figure 5(c) shows the point cloud colored with our algorithm. In comparison to Figure 5(a) and 5(b), a clean and smooth coloring is obtained without artificial color changes. Figure 5(d) depicts a dense colored point cloud with adaptive point upsampling prior to coloring. Obviously, more appearance information is available with upsampling, yielding a more visually appealing colored model.

IV. TEXTURED MODEL RENDERING

Various standard techniques from commonly used graphics processing pipeline can be used to render our colored model, as described in Section II. In this section, we explore the textured model rendering. For this purpose, we generate blended textures from the colored point cloud and map them to the mesh.

Our texture generation is based on depth images, which are generated in the previous stage to identify visible triangles. For each depth image \( D \) a corresponding RGB texture \( T \) is generated. For depth pixel \( d(u, v) \in D \) with infinite depth, the texture pixel \( t(u, v) \in T \) is set to black. For depth pixel \( d(u, v) \in D \) with a finite depth \( d \), a 3D point \( p(x, y, z) \) is computed by

\[
x = (u - c_x) \times z/f_x \\
y = (v - c_y) \times z/f_y, \\
z = d
\]

where \( c_x, c_y, f_x, f_y \) are the center and focal length of camera provided by calibration. The nearest point to \( p \) is searched in the colored point cloud and its color is assigned to \( t(u, v) \in T \). Figure 6 shows the generated texture in comparison to the original texture. It can be seen that our inexpensive color assignment yields good texture with well-preserved appearance. However, due to blending of low-quality textures with imperfect calibration and inaccurate mesh modeling, the generated textures are less sharp with slight shape distortion, especially for close objects. To obtain smoother textures, our algorithm can be easily extended to use \( k \)-nearest neighbors.

Since the generated textures are based on a blended colored point cloud, texture mapping is highly simplified. A triangle can be assigned to any visible texture patches without producing artificial color changes. However, to provide more pictorial details for rendering, we assign a weight to the visible texture patches by \( \omega_i = \alpha_i \times | \cos \theta_i | \), where \( \alpha_i \) and \( \theta_i \) are the area and angle between point normal and viewpoint of the \( i \)th patch, respectively. The patch with highest weight is assigned to the triangle.
V. Experiments and Results

In this section, we present a series of experiments to evaluate our algorithms. We have implemented our algorithms on Linux Ubuntu 12.04, and all measurements are performed on a CPU i5 core 750 of 2.67 GHz with 3.8 GB memory. The input datasets are acquired from an extended RGB-D reconstruction system with a Kinect sensor [15], [16], which provides a polygon mesh and a set of 640 × 480 textures with calibration parameters. Six datasets are used in evaluation, encompassing a wide variation in the scale of the mesh, the amount of points and the number of textures.

In the first experiment, we examine the general performance of our algorithms for colored model generation and rendering. In Figure 5, we already show the comparison between different coloring methods and point upsampling. This improvement in finding appropriate color values is especially beneficial for datasets with errors in calibration and mesh modeling. In addition to Figure 5, four colored point clouds produced by our algorithm are shown in Figure 8(a). These results illustrate that our colored models give natural appearance representations. Figure 8(b) shows the rendering of textured models produced by our algorithm which provide visually appealing visualization.

In the second experiment, we evaluate the efficiency of adaptive point upsampling with respect to enhancing the capability of appearance representation. For a direct illustration, we compare the colored point cloud and the corresponding generated textures with and without point upsampling in Figure 7. The result clearly shows that the upsampled point cloud contains more texture information and presents appearance in a more complete and detailed way. This improvement in color association is also reflected in the generated texture, which becomes much sharper with more pictorial details. For a further assessment of point upsampling efficiency, we define the Pixel Using Rate (PUR) and the Pixel Repetition Rate (PRR). The PUR is the percentage of visible pixels mapped to points, which measures the amount of visible texture information associated to the point cloud. The PRR is the average frequency of visible pixels being mapped to points, which reflects the redundancy in points. In ideal situation, both PUR and PRR equal unity. The higher PUR, the more complete appearance is stored. The lower PRR, the fewer redundant points are added. The average PUR and PRR of the six datasets are shown in Table I. Obviously, the PUR is much higher with point upsampling, while the PRR is slightly increased. This result shows that the adaptive point upsampling provides a good basis to store necessary appearance information without adding too many unnecessary points.

In the third experiment, we examine the computational efficiency of the point-based algorithm for colored model generation. The average execution time for processing one texture is given in Table II. We compare the execution time of coloring point clouds with and without adaptive point upsampling. It can be observed that the algorithm is efficient in processing large datasets and the upsampling procedure does not increase execution time significantly.

VI. Conclusions

In this paper, we have studied the problem of 3D colored model generation and textured model rendering, based on multiple textures and a triangular mesh. While focusing on—but not limited to— inexpensive reconstruction platforms, we have developed algorithms that are tolerant to noise from (a) low-resolution images with motion blur, (b) imperfect camera calibration and (c) inaccurate mesh modeling. We
contribute in two ways. First, we have presented a point-based algorithm to generate colored models, using 3D surface points as processing primitives. Three techniques have been explored for this algorithm, including an accurate depth image estimation, an adaptive surface point upsampling and texture blending using these points. The final colored model is described as a dense colored point cloud maintained in a mesh structure, where the appearance information from multiview textures is correctly extracted and associated to the geometry.

We have quantified the efficiency of our point upsampling algorithm with novel metrics, indicating the influence of the 3D points. These metrics are the Pixel Using Rate and the Pixel Repetition Rate, measuring the percentage of texture information being extracted and the frequency of pixels being mapped to points. The generated colored models offer great flexibility in applying various standard rendering techniques for visualization. Our second contribution is the textured model generation for rendering, where we have developed an algorithm to generate blended textures from colored point clouds and map the textures onto the mesh. The experimental results show that our proposed algorithms efficiently produce visually appealing colored models and enable high-quality rendering. For future research, we are focusing on improving the quality of input data to achieve a better colored model.

Possible techniques to be explored include image deblurring and image-based camera calibration.

REFERENCES


