EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

Volume 31 (*2012*), *Number 2*

Mesh Colorization

George Leifman and Ayellet Tal

Technion – Israel Institute of Technology

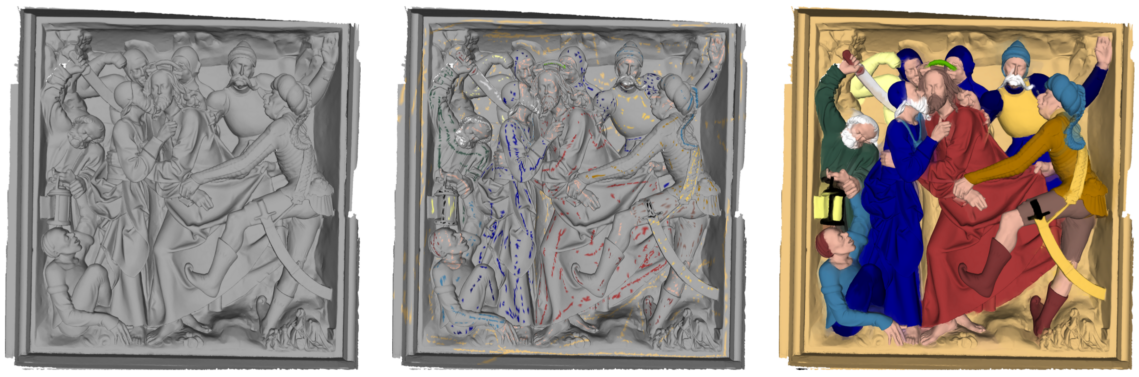

(a) Input model (b) User's scribbles (c) Colorized model

Figure 1: *Given a 3D model (a), the user scribbles on it using the desired colors (b). Our algorithm completes the colorization and generates the model shown in (c).*

Abstract

This paper proposes a novel algorithm for colorization of meshes. This is important for applications in which the model needs to be colored by just a handful of colors or when no relevant image exists for texturing the model. For instance, archaeologists argue that the great Roman or Greek statues were full of color in the days of their creation, and traces of the original colors can be found. In this case, our system lets the user scribble some desired colors in various regions of the mesh. Colorization is then formulated as a constrained quadratic optimization problem, which can be readily solved. Special care is taken to avoid color bleeding between regions, through the definition of a new direction field on meshes.

1. Introduction

Colorization was introduced by Markle in 1970 to describe the computer-assisted process for adding color to black-and-white movies or TV programs [Bur]. The term is now used generically to describe any technique for adding color to monochrome stills and footage and was extensively investigated in computer graphics and computer vision [LLW04, LWCO*07, QWH06]. For 3D models, colorization has hardly been explored. Instead, models are usually textured by images, which manage to "cover" the model by a rich set of textures.

There are, however, applications that do not need rich textures, but rather require colorization by just a handful of colors. For such applications, texture mapping is not only an overkill, but it might also produce incorrect output. More importantly, it requires an image that is similar to the model and contains the "right" texture—an image that does not necessarily exist. In such cases, if the user could easily and quickly colorize the mesh with a few brush strokes, traditional texturing would be avoided. This paper proposes such a colorization algorithm; see Figure 1.

An interesting application of colorization algorithms is in the field of archaeology. This domain has been re-

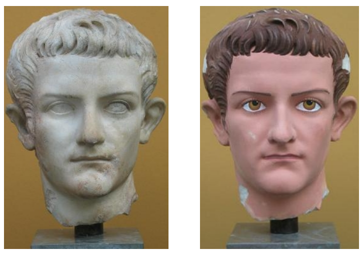


Figure 2: *Head of Caligula. Left: Original, Roman, A.D.39-41, marble. Right: Manual color reconstruction (Photo courtesy of Ny Carlsberg Glyptotek, Copenhagen).*

cently considered quite extensively in computer graphics [Rus05, KTN*06, KST08, TFBW*10]. The common perception of the great statues and buildings of ancient Greece and Rome is that they were all pure unpainted stone or green tarnished bronze. However, lately researchers have been arguing that this may not be what these classic monuments really looked like back in the era of their creation. In fact, it is believed that these statues were quite alive, vibrant, and full of color [BM08], as illustrated in Figure 2. The colors were created using mineral and other organic materials. Unfortunately, after centuries of deterioration any trace of pigment leftover when discovered, would have been taken off during the cleaning processes. Researchers argue that the number of colors and hues used by the artists was limited. In addition, chemical analysis can often estimate the original color. In this case, colorization algorithms will be able to restore the look of the scanned statues.

We propose a novel mesh colorization algorithm. It does not require mesh segmentation, which often fails to correctly identify complex region boundaries. Our algorithm is inspired by the image colorization algorithm of [LLW04]. There, the user can scribble some desired colors in the interiors of various regions of the image. Colorization is then formulated as a constrained quadratic optimization problem, where the basic assumption is that adjacent pixels having similar intensities should have similar colors.

The extension to meshes is not straightforward, due to three issues. First, a fundamental assumption in images is that the work is performed in the YUV color space, and that the intensity Y is given. Y is the monochromatic luminance channel (intensity), whereas U and V are the chrominance channels that encode the color. To determine whether two neighboring pixels should be colorized using the same color, their intensities are compared. In the case of meshes, the "intensity channel" does not exist. Therefore, a different technique is needed for determining whether neighboring points should be colorized similarly. Second, a limitation of [LLW04] is that colors may bleed into each other. This is fixed in subsequent papers [HTC*05], by applying edge detection that bounds the regions. On meshes, however, existing edge detection algorithms often generate broken curves, through which colors can bleed. Finally, in images, the algorithms estimate the two color channels U and V, whereas Y is maintained. This makes the result look natural, as colors have nuances. In meshes, there is no such input intensity channel that can be retained.

In this paper we propose an algorithm that handles the first two challenges. We present a vertex similarity measure that can be used to determine whether two vertices should get the same color. Based on this similarity, we formulate an optimization problem that can be solved efficiently. Moreover, we introduce a new direction field on meshes. We show how the optimization problem can be modified using our direction field, so as to prevent bleeding despite the fact that surface edges are broken. The third issue is solved naturally. Originally, the different hues in a region of an image were created due to lighting effects. In three dimensions, the lighting of the scene produces natural results.

The contribution of this paper is twofold. First, we propose a novel interactive mesh colorization method (Sections 3–4). It lets the user colorize 3D models by simply providing a few scribbles on the mesh. The system is based on our similarity measure and our novel vector field. Second, we apply our method to paint reconstruction of ancient sculptures (Section 5).

2. Related work

This section briefly discusses four related issues: image colorization, texture mapping, sketch-based mesh segmentation and mesh colorization.

Image colorization: Early systems, such as [MH87], required manual outlining of the different regions. More recently, [WAM02] proposed a semi-automatic technique for colorizing a gray-scale images by transferring color from a reference color image. Later, [LLW04] proposed a simple, yet effective, user-guided colorization method. The user can scribble the desired colors in the interiors of various regions and the system spreads the colors to the rest of the image. Other algorithms that are based on color scribbles have subsequently been presented by [Sap05, YS06], where impressive results were produced from a small number of scribbles. The focus of [HTC*05] was on preventing color bleeding over object boundaries, by using adaptive edge detection. The methods mentioned above require intensive user intervention when the image contains complex textures. To address this issue, [QWH06] proposed a colorization technique for Manga images that preserves mainly pattern continuity, rather than intensity continuity. [LWCO*07] employed texture continuity to colorize pattern-intensive natural images.

Mesh texture mapping: Texture mapping is used to add color information to meshes. The two common approaches for texture mapping are constrained parameterization [Lév01, ZWT*05, HLS07, TBTS08] and the photogrammetric approach [WD97, TT09]. In the first approach, the model–image correspondence is calculated by unwrapping the manifold onto the image, constraining the parameterization by user-defined features. The alternative approach takes into account the 3D geometry of the photographed image. The missing camera parameters are estimated and the recovered camera is used to re-project the model onto the source image, implicitly defining the model–image mapping. Rather than maintaining a map from texture to geometric space, in [YKH10] it is proposed to associate colors with mesh geometry.

Differently, we assume that no image exists, which can be mapped to the mesh. Instead, the user knows which colors should be used in the different regions and thus, can easily specify them with a few brush strokes.

Sketch-based mesh segmentation: Several works on sketch-based mesh segmentation have been recently proposed [JLCW06, LHMR08, LHMR08, ZWC*10, ZT10]. An excellent comparative survey is presented in [MFL11]. The goal is to segment the mesh into semantic parts through a series of user interactions. The user simply draws freehand sketches on the mesh surface to specify the foreground and the background, and the algorithm updates the segmentation using this information. By iteratively providing more interactions, the user can refine the segmentation.

Similarly to images, colorization and segmentation relate to each other, yet differ. In particular, as observed in [LLW04], segmentation algorithms often fail to correctly identify fuzzy or complex region boundaries, such as the boundary between a subject's hair and her face.

Mesh colorization: We are not aware of any related work that proposes an effective colorization method for 3D meshes. There are, however, some commercial tools, such as Adobe Illustrator, CorelDRAW, 3ds Max, and 3D-Brush, which allow the user to paint vertices/faces of 3D meshes. All these tools require high-level expertise, and even for experts the colorization is still a time-consuming task.

3. The basic algorithm

To colorize a model, the user scribbles the desired colors on the mesh (Figure 1(b)). For each face the scribble passes through, the closest vertex is colorized with the color of the scribble. These colored vertices are considered the user-defined constraints. The algorithm then automatically propagates the colors to the remaining vertices of the mesh.

Our underlying assumption is that nearby vertices, whose geometry is similar, should have the same color. This assumption leads to an optimization problem that can be solved efficiently using standard techniques. Intuitively, the colors are propagated via mesh edges, where the likelihood of propagation via a specific edge is proportional to the similarity of the vertices adjacent to the edge.

Our basic algorithm proceeds in two steps. First, a similarity measure between neighboring vertices is computed and assigned to the corresponding edges. Then, given the scribbles and the above similarities, the colors are propagated to the whole mesh. We further elaborate on each of these steps below. The output is a mesh in which every vertex has a designated color. While this basic algorithm produces a colorized mesh, some color bleeding might occur. This topic is addressed in Section 4.

3.1. Vertex similarity

To measure the similarity between given vertices, we first discuss a descriptor that characterizes the geometry of a vertex and then present an effective similarity measure for these descriptors.

Vertex descriptor: We seek a descriptor that robustly characterizes the local geometry of the vertex. We propose to use a variation of spin images [JH99], described hereafter. Spin images are 2D histograms, which are constructed by spinning small windows, termed the *support regions*, around each vertex. The support region at vertex v is a cylinder centered at v, where the axis is aligned with the surface normal at v. The support region is divided linearly into j segments radially and k segments vertically, forming a set of $j \times k$ rings. The spin image of v is computed by counting the vertices that fall within each ring and assigning this sum to its associated *bin*, forming a 2D histogram.

Counting the vertices directly, as done in the original definition of the spin image descriptor, might be sensitive to sampling, i.e., remeshing changes the descriptor. We propose a modification to the spin image descriptor that not only reduces this sensitivity, but also improves the descriptor's ability to detect subtle changes in the mesh geometry. The new descriptor of vertex v is computed by summing the differences between the mean curvatures of v and that of each of the vertices that fall in each ring. This sum is stored in the bin associated with the ring.

In our implementation, the geometric width of the bin is set to the median of the edge lengths. Since we are interested in a description of the local geometry of a single vertex (rather than a patch), we use a relatively small support region, by setting $j = k = 8$.

We considered alternative local descriptors. We will show in Section 5 that they are less suitable for our purposes.

Similarity measure: We look for a similarity measure between vertex descriptors, which is robust to small changes in the mesh, such as noise or different triangulations. We use the *diffusion distance* for this purpose [LO06]. This approach models the difference between two histograms as a temperature field and considers the diffusion process on the field. Then, the integration of a norm on the diffusion field over time is used as a dissimilarity measure between the histograms. For computational efficiency, a Gaussian pyramid is used to discretize the continuous diffusion process.

This distance is shown to be robust to deformation and to noise in histogram-based local descriptors. It is a cross-bin distance, which allows comparison between bins at different locations. Moreover, it is quick to compute since it runs in linear time.

Specifically, given 2-dimensional histograms h_i and h_j (spin images), the diffusion distance $D(h_i,h_j)$ is defined as:

$$D(h_i,h_j) = \sum_{l=0}^{L} k(|d_l^{ij}|), \tag{1}$$

where $$d_0^{ij} = h_i - h_j$$

$$d_l^{ij} = [d_{l-1}^{ij} \cdot \phi(\sigma)] \downarrow_2, l = 1,\ldots,L$$

are different layers of the pyramid. The notation $\downarrow_2$ denotes half size down-sampling. L is the number of pyramid layers and σ is a constant standard deviation for the Gaussian filter ϕ (we use $L = 3$ and $\sigma = 0.5$). In our implementation $k(.)$ is the L_1 norm, making the diffusion distance a true metric.

There are other well-known distances used in the literature. We experimented with L_2, χ^2, Jeffrey distance [Jef46], and the Earth Mover's Distance (EMD). We found that the first three are more prone to noise than the diffusion distance, since they are not cross-bin distances, whereas EMD, which is cross-bin, is computationally more expensive.

Edge assignment: Finally, we are given two vertices v_i and v_j and their corresponding descriptors h_i and h_j. We use the diffusion distance D between the descriptors to calculate the similarity between the vertices s_{ij} and assign it to the edge between v_i and v_j. Specifically,

$$s_{ij} = \frac{MaxD - D(h_i,h_j)}{MaxD}, \tag{2}$$

where $MaxD$ is the maximal diffusion distance between all the pairs of neighboring vertices of the mesh. Intuitively, s_{ij} can be viewed either as the color conductivity of the edge or as the likelihood of color propagation through the edge. This is so since it indicates the "amount of color" that can flow between v_i and v_j.

3.2. Color propagation:

We wish to impose the constraint that two neighboring vertices should have similar colors if their geometry is similar. Therefore, for all the vertices of mesh M, we attempt to minimize the difference between the color at vertex v_i and the weighted average of the colors at neighboring vertices $v_j \in N(v_i)$. To achieve this, for each color channel C in the YUV color space, we minimize the following cost function:

$$\Omega(C) = \sum_{v_i \in M} \left(C(v_i) - \sum_{v_j \in N(v_i)} k_{ij} C(v_j) \right)^2. \tag{3}$$

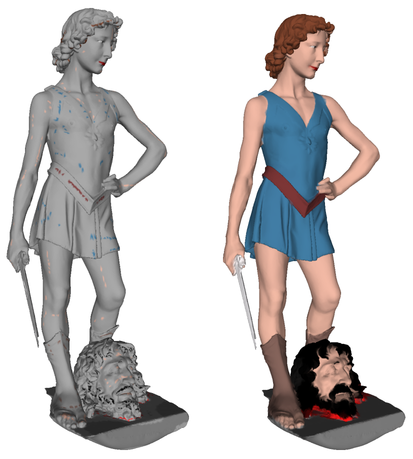

(a) User's scribbles (b) Colorized mesh

Figure 3: *David and Goliath. A good colorization is obtained by the basic algorithm described in Section 3.*

In this equation, k_{ij} is a weight function, which is large when the descriptor of v_i is similar to that of v_j and small otherwise, and for which $\sum_{v_j \in N(v_i)} k_{ij} = 1$. We define it as

$$k_{ij} \propto e^{s_{ij}^2 / 2\sigma_i^2}, \tag{4}$$

where s_{ij} is the similarity of v_i and v_j, as defined in Equation (2) and σ_i is the standard deviation of the similarity values between each of the vertices in $N(v_i)$ and v_i.

In many cases, the above scheme produces satisfactory results, as illustrated in Figure 3. However, sometimes the resulting colorization might include color bleeding near the boundaries between the regions. This is evident when looking at the hand holding a book in Figure 4(b), which is a magnification of Figure 9. In the next section we show how to handle color bleeding by adjusting the similarities calculated in Equation (2).

4. Feature line field

To handle color bleeding, we wish to decrease the similarity between vertices that belong to different regions. An indication that vertices reside in different regions is that they are located on opposite sides of a feature line, such as a valley. A straightforward solution is to use the feature lines directly, by setting the distance between neighboring vertices in accordance with their relative position to the feature lines. This solution will fail, since algorithms that compute feature lines usually produce broken lines. As a result, colors can bleed through the gaps in the lines. Figure 4(c) illustrates the valleys detected by [OBS04], where the gaps are noticeable.

We propose to cope with this situation by using a new vector field on meshes, which is defined in accordance with the feature lines. We term this vector field as the *feature line*

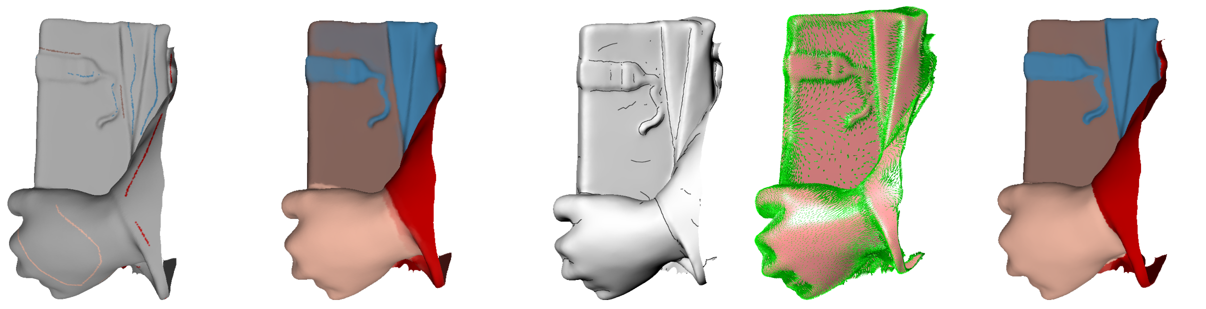


Figure 4: *Handling color bleeding (zooming into Figure 9). This example shows how the bleeding effects disappear when the similarity measure of Equation (2) is modified according to the penalties defined using our vector field (Equation (8)).*

field. The key idea is to direct the vectors in such a way that vertices residing on the two different sides of a feature line (i.e., in different regions) will get opposite orientations. These directions can then be used to penalize the similarity measure of neighboring vertices, when need be.

Below, we first define our vector field for any type of feature lines. In our implementation, however, we focus on valleys, since for most examples they are likely to separate the different regions, as demonstrated in Figure 4(c). We then explain how to compute the field on a triangular mesh. Finally, we discuss its use for adjusting the similarity measure between vertices, which was defined in Equation (2).

4.1. Feature line field definition

A vector field F for a manifold surface S is a smooth vector-valued function that associates every point $p \in S$ with a vector $F(p)$. Vector fields on surfaces have been used in various graphics applications, such as texture synthesis [Tur01, XCOJ*09], non photo-realistic rendering [ZMT06], and shape deformation [vFTS06]. There exist many papers on editing, generating, manipulating, and filtering vector fields [ZMT06, FSDH07, XCOJ*09].

We seek a vector field that satisfies the following two requirements for every point: (1) For neighboring points that belong to different regions, the directions of the associated vectors should be opposite. (2) For neighboring points that belong to the same color region, the directions of the associated vectors should have similar orientations. Note that though we require that the directions conform with the regions, we never compute segmentation to regions.

We construct the field in two steps. First, to satisfy Requirement (1), we define the vectors for points in the neighborhood of the feature lines to be directed perpendicularly to the feature line, pointing outwards. Then, to satisfy Requirement (2), we extend the above definition to the whole surface, by searching for a smoothest direction field that respects the values in the neighborhood of the feature lines. Utilizing the Laplacian as the smoothness measure, we define the field as the solution to Poisson's equation. The values of the field near the feature lines serve as boundary conditions for the equation.

4.2. Feature line field computation

In practice, our surface is given as a triangular mesh. We need to compute a vector $F(v)$ for every vertex v of the mesh. The entire set of vectors constitutes the field. We describe the computation for the case of valleys; similar schemes can be developed for other feature lines, such as ridges [OBS04], demarcating curves [KST08], etc.

Intuitively, valleys are similar to their geographical counterparts, as they indicate sharp changes in the surface orientation. Mathematically, valleys are the loci of the minimum principal curvature along the principal direction.

We start by computing the field values near the valleys. This is done as follows. First, the principal curvatures are computed for every vertex of the mesh. These values are used to find the faces through which the valleys pass [OBS04]. These faces are the gray triangles in Figure 5. Next, we mark all the vertices that belong to the valley faces, shown in red, black, and blue in Figure 5. The faces adjacent to exactly two marked vertices are called the *surrounding faces* (the green triangles).

If a marked vertex belongs to a single surrounding face (the blue vertices), its feature line field direction lies on the line perpendicular to the edge adjacent to it (and to its neighboring marked vertex) on the plane the triangle resides on. The direction is oriented towards the unmarked vertex of this surrounding triangle. If a marked vertex belongs to multiple surrounding faces (the red vertices), we first compute its direction for each face individually, as explained above. Then, we set its feature line field direction to the average of all the field vectors computed for each face independently. In Figure 5, the feature line field directions are the arrows.

To compute the field for the other vertices, we use a linear

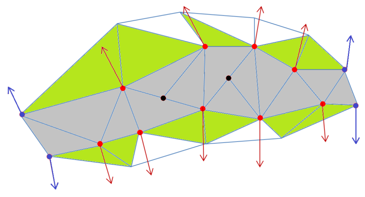


Figure 5: *Computation of the field near a feature line. The feature line passes through the gray triangles and the green triangles are their surrounding faces. The arrows are the direction vectors, pointing away from the feature line.*

approximation of the Laplacian. Let v be a vertex of the mesh and $N(v)$ be its set of neighbors. Let $w_i = \cot(\alpha_i) + \cot(\beta_i)$ be the sum of cotangents of the angles opposite the edge (v, v_i) in the triangles sharing this edge. The Laplacian of a scalar function f on the mesh is calculated, similarly to [BPR*06], as:

$$\Delta f(v) = \sum_{i \in N(v)} w_i(f(v) - f(v_i)). \tag{5}$$

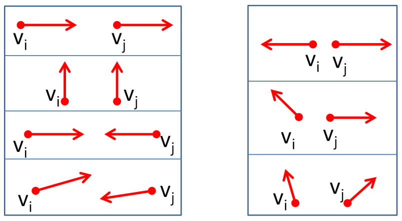
Finally, we solve the Laplace's equation $\Delta f(v) = 0$ for each of the field's components (scalars x, y, z). This system of linear equations, whose unknowns are the components of the field, is solved efficiently using a standard sparse linear equation solver (we use sparse Cholesky factorization).

Since our field is a direction field, we normalize the resulting vectors. Note that when approximating the field components independently and then normalizing them, the Laplacian is not guaranteed to remain minimal. In practice, however, the difference between the resulting approximation and the sought-after solution is very small.

Our vector field differs from other vector fields both in definition and in computation. While the vectors of [ZMT06, FSDH07, XCOJ*09] follow the direction of the features, we aim at directing them perpendicularly to the features (Requirement (1)). Computationally, while the method of [ZMT06] takes special care to handle singularities, which is essential for texture mapping, we tolerate singularities, as will be evident next. Unlike [ZMT06, FSDH07], our method is not interactive. Similarly to other methods, such as [XCOJ*09], we use a discrete Laplace operator for smoothing. However, the vectors generated by our algorithm need not necessarily lie on the surface. As explained in Section 4.3, this additional degree of freedom is beneficial, since the angle between the directions of neighboring vertices is indicative of their similarity. Moreover, the computation of the field becomes simpler.

4.3. Similarity measure adjustment

Recall that the main purpose of our vector field is to adjust the similarities of Equation (2), so as to prevent color bleeding. Moreover, we defined our vector field so that the vectors of neighboring vertices that lie on different sides of a feature line are directed away from each other. Therefore, the similarity value of such vertices should be penalized (i.e., decreased). Conversely, there is no need to penalize pairs of neighboring vertices whose field vectors are either similar or facing each other. The former may be similar either due to our direction assignment or due to the smoothness requirement. They may be directed towards each other in the internal parts of a region, as a result of applying the Laplacian. Figure 6 illustrates the two scenarios.



(a) within a region (b) in different regions

Figure 6: *Possible field directions of neighboring vertices. In (a), the directions are either similar or directed towards each other (the latter may happen in the center of a region). In these cases $\delta_{ij} \leq 0$ and therefore $P_{ij} = 0$ and the similarity values of Equation (2) are unchanged. In (b), the neighboring vertices are on opposite sides of a valley. These cases should be penalized and indeed, $\delta_{ij} > 0$, $0 < P_{ij} \leq 1$, and Equation (2) is modified.*

To determine the cases in which the similarity values should be penalized, we proceed as follows. Given a pair of neighboring vertices v_i and v_j, we translate each vertex along its field vector by ε (ε is set to half of the shortest edge of the mesh). If the distance between these translated vertices is larger than the distance between the original vertices, it means that the vectors are directed away from each other. Therefore, the similarity values between these vertices should be penalized. On the other hand, if the distance decreases or does not change, the vectors are oriented towards each other or to the same direction.

Equation (6) performs this computationally and distinguishes between the cases. Given two vertices v_i and v_j, it computes δ_{ij}, where a positive δ_{ij} indicates the first case and a non-positive δ_{ij} indicates the second. δ_{ij} is defined as:

$$\delta_{ij} = \|((\mathbf{v_i} + \varepsilon F(v_i)) - (\mathbf{v_j} + \varepsilon F(v_j)))\| - \|\mathbf{v_i} - \mathbf{v_j}\|, \tag{6}$$

where $\mathbf{v}$ denotes the coordinates of vertex v and $F(v)$ is the field vector associated with v. The translation by a small ε is essential for handling the case where the vertices are close to each other and directed towards each other.

We can now proceed and define the penalty P_{ij} of the similarity between vertices v_i and v_j, in accordance with δ_{ij}.

Obviously, if $\delta_{ij} \leq 0$, $P_{ij} = 0$. Otherwise, let θ_{ij} be the angle between vectors $F(v_i)$ and $F(v_j)$. The penalty should depend on the angle between the vectors. If θ_{ij} is close to 180°, the vectors are almost in opposite directions and the penalty should be large, whereas if θ_{ij} is close to 0°, the penalty should be smaller. It is defined as:

$$P_{ij} = \begin{cases} 0, & \delta_{ij} \leq 0 \\ \sin(\frac{\theta_{ij}}{2}), & \delta_{ij} > 0 \end{cases} \tag{7}$$

Finally, we adjust the similarities s_{ij} of Equation (2). They are defined as follows and substituted into Equation (4):

$$\hat{s}_{ij} = s_{ij} \cdot (1 - P_{ij}). \tag{8}$$

Figure 4 illustrates how color bleeding is prevented using our modified similarities. In this example, the basic colorization algorithm, described in Section 3, results in some undesirable color bleeding effects between the cloth and the hand, the book and the hand, and the cyan book cover and the book. Adjusting the similarities in accordance with our field directions solves the problem, as shown in Figure 4(e).

5. System and results

In a typical colorization session, the user starts by scribbling a few color strokes on the mesh and then fine-tunes the colorization by adding more scribbles. Usually, most of the scribbles are loosely placed. Only during the final fine-tuning stage, a couple of scribbles may need to be added closer to the boundaries. Figure 7 demonstrates a possible session, which took a couple of minutes altogether.

The number of scribbles and the interaction time depend on the complexity of the model. For instance, in Figures 3,7 very few scribbles ($\sim$10) were used and the colorization was very fast (see the accompanying movie). Figure 1 was the most complex, requiring around 50 strokes and 10 minutes.

Results: Figure 8 shows additional examples, where convincing results are generated by our algorithm, given a small number of color scribbles. Note how our algorithm manages to distinguish between the individual fingers and the cloth or the other hand-held objects. Moreover, despite the multiple folds of the cloth, it is easily colored using only a few scribbles that cross it.

Figure 9 demonstrates another capability of the system—colorizing a region in different colors despite the fact that geometrically it is a single region. In this example, the eyes are colorized in blue, white (the sclera), and black (the iris). In such cases, all the user needs to do is to add the desired feature lines manually, by drawing them on the mesh. Then, these manual feature lines are added to the set of computed feature lines and treated similarly. Since our algorithm is designed to work with broken lines, the user does not have to carefully draw closed lines, which simplifies the drawing process considerably. In this example, six lines (bounding each eye, sclera, and iris) were added, which took the user about three minutes.

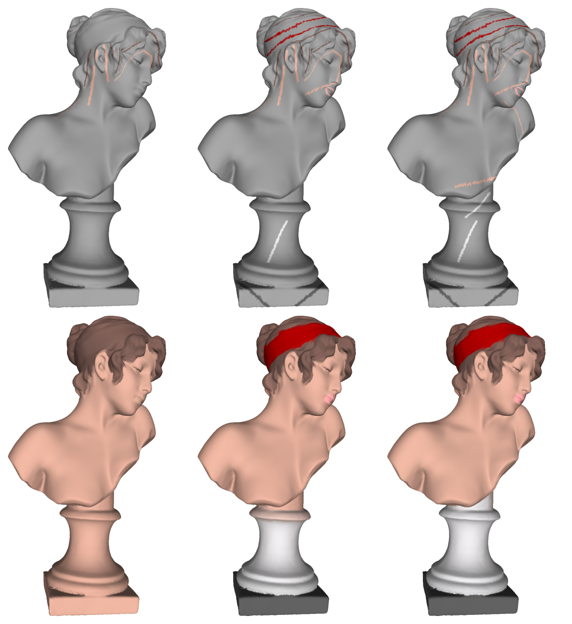


Figure 7: *A typical colorization session. The user starts by scribbling two colors (left). Then, the user adds red to the head band, light gray to the column, dark gray to the base and pink to the lips (middle). Finally, a slight color bleeding to the column is easily fixed by additional two scribbles close to the boundaries (right).*

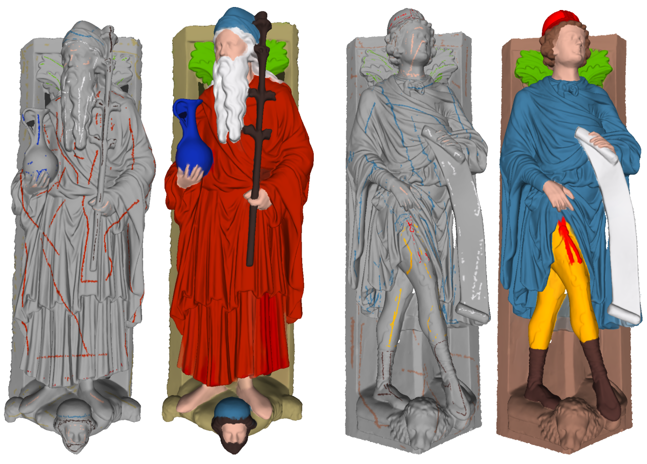


Figure 8: *Only a few scribbles are needed to colorize the cloth, despite its folded structure. Note how the fingers are separated from the folded cloth or the hand-held objects.*

Figure 10(a) demonstrates the robustness of our method to noise. Synthetic noise of zero mean and variance of 0.1% of the diagonal of the model's bounding box was added to the vertices. The result obtained using the same set of scribbles

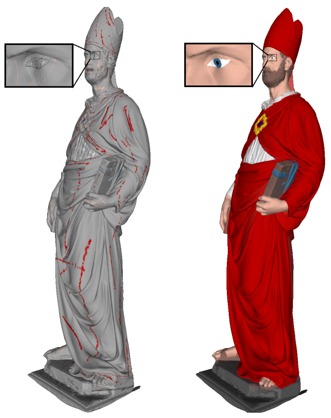
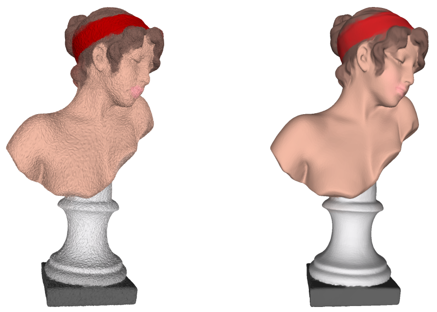


Figure 9: *Colorization of a mesh when different colors are used within the same region (the eye). The user draws the desired feature lines.*



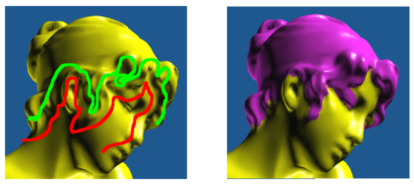
(a) Robustness to noise (b) Robustness to density

Figure 10: *Robustness to noise and mesh density. (a) Synthetic noise was added to the vertices. The result obtained using the same set of scribbles is similar to that of Figure 7. (b) Simplification of the mesh from 150K to 25K vertices does not change the colorization results.*

is good and similar to clean model in Figure 7. Figure 10(b) demonstrates the robustness of our method to mesh density.

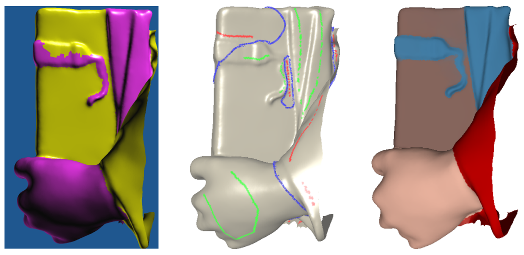
Comparison to sketch-based segmentation: We are not aware of any colorization algorithm we could compare to. Therefore, we compare our method to sketch-based foreground/background segmentation, though the problems are fundamentally different. Unlike segmentation, colorization need not identify the precise region boundaries.

In particular, we compare our algorithm to [JLCW06], which was found in [MFL11] to outperform other algorithms according to most of the comparison criteria. Figure 11 shows a segmentation result, where a larger and more accurate set of strokes than those used in Figure 7, are drawn. It can be seen that similarly to the observation made in [LLW04] regarding image colorization vs. segmentation, "segmentation is a very difficult problem and even state-of-the-art methods may fail to automatically delineate all the correct boundaries, such as the intricate boundary between the hair and the forehead."



(a) User's scribbles (b) Segmented mesh

Figure 11: *Sketch-based segmentation. Even with a larger and a more accurate set of strokes than those used in Figure 7, the fuzzy boundary between the hair and the face is not accurately detected by [JLCW06].*



(a) [JLCW06] (b) [ZWC*10] (c) Our result

Figure 12: *Comparison to two segmentation algorithms using the same scribbles. In [ZWC*10], the blue curves indicate the boundaries between the segments.*

Figure 12 compares our result both to [JLCW06] and to [ZWC*10], using the same scribbles as those used to produce Figure 4. It is noticeable that the segmentation results are inferior to that produced by our colorization algorithm.

An additional difference is that the released foreground/background interfaces, often used in interactive segmentation, do not support segmentation with branching points, i.e., points where three or more boundaries meet. Such a case is illustrated in Figure 12, where the book, the sleeve, and the hand meet at a branching point.

Finally, objects such as the one in Figure 1, which consist of many shapes whose geometries overlap, are very difficult to segment. We failed to segment this model using the segmentation systems discussed above, possibly due to multiple branching points. Moreover, in other regions of the mesh, these systems did not segment correctly the hair and the beards. Conversely, our algorithm manages to colorize this model thanks to its ability to utilize the imperfect valleys between regions.

Alternative vertex descriptors: The choice of a vertex descriptor (Section 3.1) is critical to the success of the algorithm. We considered alternative local descriptors, includ-

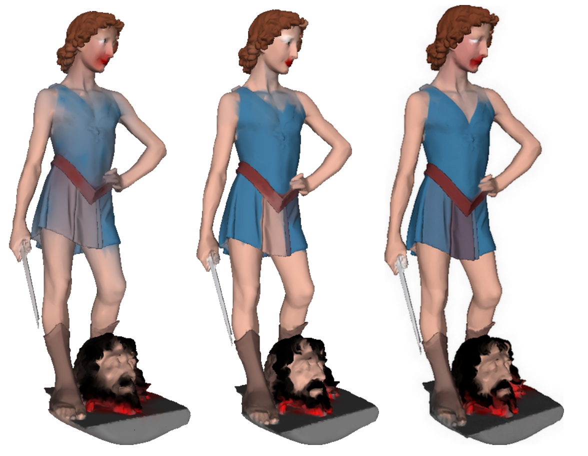


Figure 13: *The colorization results obtained when using other vertex descriptors are inferior to ours (Figure 3).*

Mesh size	Preprocessing step	Online session
50K faces	43 sec	0.7 sec
100K faces	71 sec	1.5 sec
200K faces	198 sec	3.3 sec
400K faces	480 sec	7.1 sec

Table 1: *Running times for meshes of different sizes. The complexity is linear and the online stage is fast.*

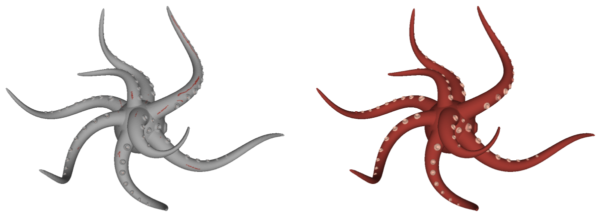


Figure 14: *Limitation. To colorize the suction cups, we should draw a stroke on each of them.*

ing the heat kernel signature (HKS) [SOG09], shape index [KvD92], Willmore energy [BS05], and combinations of vertex normals and curvatures (e.g., the improved Isophotic metric [JLCW06]). They were all found less suitable for distinguishing between adjacent vertices residing in different regions. Consequently, the colorizations obtained were inferior, as illustrated in Figure 13 for three state-of-the-art descriptors.

Implementation and running times: The system is implemented in Matlab, where some parts (i.e., the similarity computation and the valley generation) are implemented in C++. The user scribbles on the mesh using the Z-painting tool of Meshlab [CCR08].

We tested the algorithm on an Intel Core i5 laptop with 4GB of memory, Windows 7 operating system. The preprocessing step computes the similarity between each pair of adjacent vertices. This step is computed once, regardless of the number of iterations in the colorization session. The online stage solves the minimization problem of the color propagation. It is computed whenever the user adds new scribbles and can thus be performed multiple times. Table 1 summarizes the running times for meshes of different sizes. It can be seen that both stages are approximately linear in the size of the mesh and that the online stage is performed in just a few seconds even for large meshes.

Limitations: In general, our algorithm requires a relatively dense input mesh, which is almost always the case in our application. When the input mesh is highly irregular or sparse, or when the vertices of two valleys are one vertex apart, re-meshing may be needed. For example, in Figure 10(b), simplification to 25K vertices did not require re-meshing, but if the mesh were simplified to 10K vertices, re-meshing would have to be performed.

Second, adding manual feature lines, as done in Figure 9, requires re-computation of the field, which slows down the running time.

Finally, our algorithm does not handle 3D patterns or symmetries automatically. Figure 14 shows such a case, where in order to colorize the suction cups of the octopus, we need to draw a color stroke on each suction cup.

6. Conclusions

This paper introduced a novel colorization algorithm for meshes. The user can draw several strokes quickly and easily and our system propagates the colors.

We addressed two challenges when designing our color propagation algorithm. The first relates to the choice of points that need to get the same color and the second concerns the avoidance of color bleeding. For the first problem, we discussed a vertex descriptor and a similarity measure that can jointly distinguish between local geometries in the neighborhood of a vertex. For the second, we proposed a new feature line field that can be utilized in the similarity measure, so as to prevent color bleeding. We demonstrated the capabilities of our algorithm by colorizing a variety of statues. In all these examples, we obtained good results in only a few minutes of interactive user work.

In the future, we would like to address 3D patterns and symmetries. Moreover, it will be interesting to examine the use of our algorithm in NPR applications. Finally, remeshing could be utilized in regions of color transitions, in order to improve the results.

Acknowledgements: We thank Artec3D for providing the models. This research was supported in part by the Israel Science Foundation 628/08 and by Ollendorff Foundation.

References

[BM08] Bonn-Muller E.: Carved in living color. *Archaeology 61*, 1 (2008). 2

[BPR*06] Botsch M., Pauly M., Rossl C., Bischoff S., Kobbelt L.: Geometric modeling based on triangle meshes. In *SIGGRAPH Courses* (2006). 6

[BS05] Bobenko A., Schröder P.: Discrete willmore flow. In *SGP* (2005), pp. 101–110. 8

[Bur] Burns G.: Colorization. *Museum of Broadcast Communications: Encyclopedia of Television*. http://www.museum.tv/eotvsection.php?entrycode=colorization. 1

[CCR08] Cignoni P., Corsini M., Ranzuglia G.: Meshlab: an open-source 3D mesh processing system. *ERCIM News*, 73 (2008), 45–46. 9

[FSDH07] Fisher M., Schroder P., Desbrun M., Hoppe H.: Design of tangent vector fields. *ACM Transactions on Graphics 26*, 3 (2007), 56:1–9. 5, 6

[HLS07] Hormann K., Lévy B., Sheffer A.: Mesh parameterization: theory and practice. In *SIGGRAPH courses* (2007). 2

[HTC*05] Huang Y.-C., Tung Y.-S., Chen J.-C., Wang S.-W., Wu J.-L.: An adaptive edge detection based colorization algorithm and its applications. In *ACM Multimedia* (2005), pp. 351–354. 2

[Jef46] Jefrey H.: An invariant form for the prior probability in estimating problems. *Mathematical and Physical Sciences, the Royal Society London 186* (1946), 453–461. 4

[JH99] Johnson A., Hebert M.: Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Trans. Pattern Anal. Mach. Intell. 21*, 5 (1999), 433–449. 3

[JLCW06] Ji Z., Liu L., Chen Z., Wang G.: Easy mesh cutting. *Computer Graphics Forum 25*, 3 (2006), 283–291. 3, 8

[KST08] Kolomenkin M., Shimshoni I., Tal A.: Demarcating curves for shape illustration. *ACM Transactions on Graphics 27*, 5 (2008), 157:1–9. 1, 5

[KTN*06] Koller D., Trimble J., Najbjerg T., Gelfand N., Levoy M.: Fragments of the city: Stanford's digital forma urbis romae project. *J. Roman Arch. 61* (2006), 237–252. 1

[KvD92] Koenderink J., van Doorn A.: Surface shape and curvature scales. *Image and Vision Computing 10* (1992), 557–565. 8

[Lév01] Lévy B.: Constrained texture mapping for polygonal meshes. In *SIGGRAPH* (2001), pp. 417–424. 2

[LHMR08] Lai Y., Hu S., Martin R., Rosin P.: Fast mesh segmentation using random walks. In *SPM* (2008), pp. 183–191. 3

[LLW04] Levin A., Lischinski D., Weiss Y.: Colorization using optimization. *ACM Transactions on Graphics 23*, 3 (2004), 689–694. 1, 2, 3, 8

[LO06] Ling H., Okada K.: Diffusion distance for histogram comparison. In *CVPR* (2006), vol. 1, pp. 246–253. 3

[LWCO*07] Luan Q., Wen F., Cohen-Or D., Liang L., Xu Y.-Q., Shum H.-Y.: Natural image colorization. In *Eurographics Symposium on Rendering* (2007). 1, 2

[MFL11] Meng M., Fan L., Liu L.: A comparative evaluation of foreground/background sketch-based mesh segmentation algorithms. *Computers & Graphics 35*, 3 (2011), 650–660. 3, 8

[MH87] Markle W., Hunt B.: Coloring black and white signal using motion detection. *Canadian Patent Nr. 1291260* (1987). 2

[OBS04] Ohtake Y., Belyaev A., Seidel H.: Ridge-valley lines on meshes via implicit surface fitting. *ACM Transactions on Graphics 23*, 3 (2004), 609–612. 4, 5

[QWH06] Qu Y., Wong T.-T., Heng P.-A.: Manga colorization. *ACM Transactions on Graphics 25*, 3 (2006), 1214–1220. 1, 2

[Rus05] Rushmeier R.: Eternal Egypt: experiences and research directions. In *Modeling and Visualization of Cultural Heritage* (2005), pp. 22–27. 1

[Sap05] Sapiro G.: Inpainting the colors. In *ICIP* (2005), pp. 698–701. 2

[SOG09] Sun J., Ovsjanikov M., Guibas L.: A concise and provably informative multi-scale signature based on heat diffusion. In *SGP* (2009), pp. 1383–1392. 8

[TBTS08] Tai Y.-W., Brown M., Tang C.-K., Shum H.-Y.: Texture amendment: reducing texture distortion in constrained parameterization. *ACM Transactions on Graphics 27*, 5 (2008), 136:1–6. 2

[TFBW*10] Toler-Franklin C., Brown B., Weyrich T., Funkhouser T., Rusinkiewicz S.: Multi-feature matching of fresco fragments. *ACM Transactions on Graphics 29*, 6 (2010). 1

[TT09] Tzur Y., Tal A.: Flexistickers: photogrammetric texture mapping using casual images. *ACM Transactions on Graphics 28*, 3 (2009), 45:1–10. 2

[Tur01] Turk G.: Texture synthesis on surfaces. In *SIGGRAPH* (2001), pp. 347–354. 5

[vFTS06] von Funck W., Theisel H., Seidel H.: Vector field based shape deformations. *ACM Transactions on Graphics 25*, 3 (2006), 1118–1125. 5

[WAM02] Welsh T., Ashikhmin M., Mueller K.: Transferring color to greyscale images. *ACM Transactions on Graphics 21*, 3 (2002), 277–280. 2

[WD97] Weinhaus F. M., D. V.: Texture mapping 3D models of real-world scenes. *ACM Computing Surveys 29*, 4 (1997), 325–365. 2

[XCOJ*09] Xu K., Cohen-Or D., Ju T., Liu L., Zhang H., Zhou S., Xiong Y.: Feature-aligned shape texturing. *ACM Transactions on Graphics 28*, 5 (2009), 108:1–7. 5, 6

[YKH10] Yuksel C., Keyser J., House D.: Mesh colors. *ACM Transactions on Graphics 29*, 2 (2010), 15:1–11. 3

[YS06] Yatziv L., Sapiro G.: Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing 15* (2006), 1120–1129. 2

[ZMT06] Zhang E., Mischaikow K., Turk G.: Vector field design on surfaces. *ACM Transactions on Graphics 25*, 4 (2006), 1294–1326. 5, 6

[ZT10] Zheng Y., Tai C.: Mesh decomposition with cross-boundary brushes. *Computer Graphics Forum 29*, 2 (2010), 527–535. 3

[ZWC*10] Zhang J., Wu C., Cai J., Zheng J., Tai X.: Mesh snapping: Robust interactive mesh cutting using fast geodesic curvature flow. *Computer Graphics Forum 29*, 2 (2010), 517–526. 3, 8

[ZWT*05] Zhou K., Wang X., Tong Y., Desbrun M., Guo B., Shum H.: TextureMontage: seamless texturing of arbitrary surfaces from multiple images. *ACM Transactions on Graphics 24*, 3 (2005), 1148–1155. 2