Toward Image-dependent Gamut Mapping: Fast and Accurate Gamut Boundary Determination

Joachim Giesen[a], Eva Schuberth[a,b], Klaus Simon[b] and Peter Zolliker[b]

[a]Swiss Federal Institute of Technology (ETH), Zurich, Switzerland;
[b]Swiss Federal Laboratories for Materials Testing and Research (EMPA), St Gallen, Switzerland

ABSTRACT

We propose a new geometric method to compute color gamut boundaries fast and geometrically accurate. The method is designed for high-quality image-dependent gamut mapping in three dimensions. For such a mapping the gamut boundary must be constructed for every image individually and we cannot rely on precomputed lookup tables. This can only be practical if the gamut boundary can be computed very fast. The proposed method is fast compared to other geometric methods without sacrificing geometric accuracy of the computed boundary.

Keywords: Gamut mapping, gamut boundary description, discrete flow complex

1. INTRODUCTION

A color gamut is the entirety of colors that are physically realizable by a device or that are contained in an image. As such a gamut is a subset of a color space. Typically the color space is three dimensional and gamuts are finite subsets of it. Instead of working with the gamut directly it can be more convenient to work with a superset of the gamut which is better suited for further processing – most notably deriving good mappings between two given gamuts. Such a gamut mapping is a function that maps the color points of the source gamut to color points in the target gamut. The mapping should preserve as much of the color appearance as possible. Given a suited geometric approximation of both, the source and target gamut, it should be possible to algorithmically derive a good (in the sense of color appearance preservation) gamut mapping. It turned out that for this purpose polyhedrons that contain the gamuts can be suited approximations. The advantage of polyhedrons is that they allow for efficient implementations of basic geometric operations like computing the intersection with a given line. For the line intersection computation and other tasks it suffices to have a representation of the boundary of the polyhedron. Computations can even be sped up substantially if they are restricted to the boundary of a polyhedron. Thus here we want to compute the boundary of a polyhedron that contains the gamut and reflects its geometry as good as possible. We want to refer to these boundaries as color gamut boundaries or short gamut boundaries. The gamut boundaries are meant to be used as the input for an algorithm that computes a mapping of the corresponding gamuts. The quality of the computed mapping depends among other factors on how well the boundaries approximate the gamuts. Usually it takes more time to compute a better approximation or mapping. Thus one faces a quality vs. time trade off which is reflected in the two different approaches toward gamut mapping, see Chen and Kotera[1]:

- Device-to-device: The mapping is independent of the image. The source gamut description, the target gamut description and the mapping itself are determined once. Every image is transformed according to this mapping..

Further author information: (Send correspondence to E.S.)
E.S.: E-mail: eva.schuberth@inf.ethz.ch
J.G.: E-mail: giesen@inf.ethz.ch
K.S.: E-mail: klaus.simon@empa.ch
P.Z.: Email: peter.zolliker@empa.ch

- Image-to-device: The mapping is image-dependent. Only the target gamut description is determined once. The source gamut description and the mapping however have to be computed for every single image.

In the device-to-device approach time is preferred over quality in the trade off. The mapping between the gamuts of the source and target has to be derived only once *. When processing some image this pre-defined mapping only has to be evaluated which can be done fast, e.g., by using lookup tables. This approach is well developed and integrated in many color management systems, see the overview given by Morovic.[2,3] For smaller image gamuts less compression is necessary. The device-to-device approach does not take advantage of this and therefore compresses the image gamut more than necessary which results in sub-optimal color appearance preservation. The image-to-device approach takes the actual image gamut which is a subset of the device gamut into account. Thus it is able to better preserve the color appearance. But doing so is more time consuming since the the mapping not only has to be evaluated, but also has to be derived which of course takes more time. The derivation should be done in an automatic fashion since it is practically impossible to manually fine tune the mapping for each image - of course the lack of manual fine tuning could be a source of suboptimal color preservation. To derive a gamut mapping automatically an algorithm is needed that computes the mapping from some description of the source and target gamuts. Chen and Kotera[4] describe an algorithm that works on three dimensional gamut representations. As mentioned a suitable description is a polyhedral surface that encloses the gamut. Only the polyhedral surface has to be computed from the image gamut - we can assume that a good polyhedral surface corresponding to the device gamut has been pre-computed since the device is fixed. For the gamut mapping in the image-to-device approach in order to be fast it is necessary that the gamut boundary can be computed fast. Fast computation of gamut boundaries is the topic of this paper. But obviously speed can not be the only objective when computing gamut boundaries. In fact, we still face a quality vs. time trade off. It is reasonable that a rough polyhedral approximation can be computed much faster than a very accurate one. In this paper we propose the discrete flow complex as a good compromise between quality and time. The discrete flow complex is essentially a way to assign a polyhedral surface to color points on a grid. We demonstrate that the discrete flow complex can be computed efficiently on practical data sets and at the same time gives an accurate approximation of the input gamut.

This paper is organized as follows: In the next section we give an introduction to the field of geometric shapes from samples. Moreover we introduce discrete the discrete flow complex. In the third section we describe the imaging background of our work. That includes our choice of color space and a discussion on how to discretize color points to a grid without losing too much information. In the fourth section we describe a fast algorithm to compute the discrete flow complex. We conclude this article with a section about experimental results.

2. SHAPES

The problem to determine the boundary of a gamut is a special instance of the problem to associate a shape with a finite point set P in $\mathbb{R}^3$. The latter problem arises in many contexts. The choice of shape depends on the context and the objectives involved. Frequently encountered objectives are: the shape should capture the "geometry" of the point set as good as possible and the shape should be efficiently computable. These objectives are contradictory to a certain extent. For example an axis aligned bounding box of P can be computed very efficiently, but it does not capture any variations in the point set density and might contain large regions that do not contain any sample point at all. The convex hull of P, i.e., the common intersection of all convex subsets of $\mathbb{R}^3$ that contain P, is a subset of the bounding box. The convex hull adapts better to the actual distribution of the points, but is more costly to compute and at large has the same disadvantages as the bounding box. Thus a better volumetric description of the point set P should be a subset of the convex hull of P. One approach to compute such subsets is to decompose the convex hull of P into cells and to choose only some of these cells in order to describe the point set P. The Delaunay triangulation of P is an efficiently computable decomposition of the convex hull of P into tetrahedrons whose vertices are points from P. Alpha shapes are a family of shapes controlled by some parameter α, that choose certain simplices from the Delaunay triangulation. It was suggested by Cholewo and Love,[5] to use alpha shapes to represent gamut boundaries. Both Delaunay triangulations and alpha shapes can be derived from the Voronoi diagram of P, a structure that we introduce now.

*Not necessarily in an algorithmic fashion, i.e., it can be adjusted manually

Voronoi diagram. The Voronoi diagram of P is a cell decomposition of $\mathbb{R}^3$ into convex polyhedrons. Every Voronoi cell corresponds to exactly one sample point and contains all points of $\mathbb{R}^3$ that do not have a smaller distance to any other sample point, i.e., the Voronoi cell corresponding to $p \in P$ is given as the following set

$$\{x \in \mathbb{R}^3 \, : \, \forall q \in P \quad \|x - p\| \leq \|x - q\|\}.$$

Closed facets shared by two Voronoi cells are called Voronoi facets, closed edges shared by three Voronoi cells are called Voronoi edges and the points shared by four Voronoi cells are called Voronoi vertices. The term Voronoi object can denote either a Voronoi cell, facet, edge or vertex. The Voronoi diagram is the collection of all Voronoi objects. See Figure 1 for a two-dimensional example of a Voronoi diagram.

The Delaunay triangulation is dual to the Voronoi Diagram in a sense that we explain below. It decomposes the convex hull of the point set P into tetrahedrons.

Delaunay triangulation. The Delaunay diagram of P is a cell complex that decomposes the convex hull of the points in P. The convex hull of four or more points in P defines a Delaunay cell if the intersection of the corresponding Voronoi cells is not empty and there exists no superset of points in P with the same property. Analogously, the convex hull of three or two points defines a Delaunay face or Delaunay edge, respectively, if the intersection of their corresponding Voronoi cells is not empty. Every point in P is a Delaunay vertex. The term Delaunay object can denote either a Delaunay cell, face, edge or vertex.

The point set P is said to be in general position if there are no degeneracies of the following kind: no three points on a common line, no four points on a common circle or hyperplane and no five points on a common sphere. If P is in general position then the Delaunay diagram of P is a triangulation of the convex hull of P, i.e., all Delaunay cells are tetrahedrons. See Figure 1 for a two-dimensional example of a Delaunay triangulation.

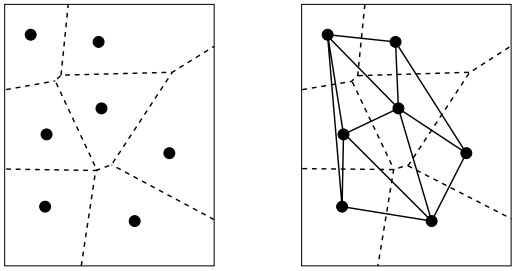

Figure 1. On the left the Voronoi diagram of a finite point set in the plane and on the right both, the Voronoi diagram and the Delaunay triangulation of the same point set.

Remember that our goal is to filter certain simplices from the Delaunay triangulation whose union gives a good description of the "geometry" of the point set P. Alpha shapes are an efficient way to do so. Alpha shapes are a family of shapes parameterized by some real value α. Every alpha shape in this family is a Delaunay triangulation restricted to a set of balls of radius α centered at the points in P. A Delaunay simplex belongs to the restricted Delaunay triangulation if the Voronoi cells of its vertices have a common non-empty intersection with the set of balls. We summarize this definition below and give some examples in Figure 2.

Alpha shapes. For a given value of $\alpha \in [0, \infty)$ the corresponding alpha shape of P is the Delaunay triangulation of P restricted to the set of all balls with radius α centered at the points in P. Note that for $\alpha = 0$ the alpha shape consists just of the set P and for very large values of α the alpha shape is the Delaunay triangulation of P. See Figure 2 for two-dimensional examples of alpha shapes.

Alpha shapes are intimately linked to another family of shapes called flow shapes. Flow shapes have been developed by Giesen and John.[6] They are derived from a dynamical system associated with a distance function induced by the point set P.

Induced distance function. The distance function

$$h : \mathbb{R}^3 \to \mathbb{R}, x \mapsto \min_{p \in P} \|x - p\|$$

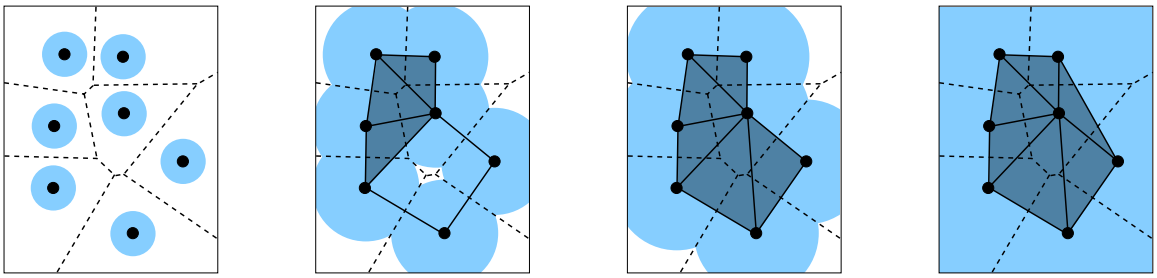

Figure 2. From the left to the right are shown alpha shapes for growing values of α.

induced by P is closely related to the Voronoi diagram of P in the sense that the value of h at x is determined by the point in P in whose Voronoi cell x is contained.

It was observed by Edelsbrunner[7] and later proven by Giesen and John[6] that the critical points of the distance function, i.e., its local extrema and the saddle points, can be characterized in terms of Delaunay simplices and Voronoi objects. The characterization immediately leads to an efficient algorithm to compute all critical points. In shape modeling we are not interested in the critical points themselves, but in the so called stable manifolds of the critical points. The stable manifold of a critical point is a subset of $\mathbb{R}^3$ whose definition we give below.

Induced flow and stable manifolds. Even though the distance function h is not smooth there is a unique direction of steepest ascent of h at every non-critical point of h. Assigning to the critical points of h the zero vector and to every other point in $\mathbb{R}^3$ the unique unit vector of steepest ascent, defines a vector fiel on $\mathbb{R}^3$. This vector field is not smooth but nevertheless gives rise to a flow on $\mathbb{R}^3$, i.e., a mapping

$$\phi : [0, \infty) \times \mathbb{R}^3 \to \mathbb{R}^3,$$

such that at every point $(t, x) \in [0, \infty) \times \mathbb{R}^3$ the right derivative

$$\lim_{t \leftarrow t'} \frac{\phi(t, x) - \phi(t', x)}{t - t'}$$

exists and is equal to the unique unit vector of steepest ascent at x. The flow basically tells how a point would move if it always followed the steepest ascent of the distance function h. The curve that a point x follows is given by $\phi_x : \mathbb{R} \to \mathbb{R}^3, t \mapsto \phi(t, x)$ and is called the orbit of x. See Figure 3 for some example orbits in two dimensions.

Given a critical point x of h, the set of all points whose orbit ends in x, i.e., the set of all points that flow into x, is called the stable manifold of x. The collection of all stable manifolds forms a cell complex which is called flow complex. See Figure 3 for examples of stable manifolds in two dimensions.

Alpha shapes turned out to be a convenient way to filter simplices from the Delaunay triangulation. Flow shapes serve the same purpose in the context of the flow complex, i.e., for any given value of α the corresponding flow shape filters certain stable manifolds from the flow complex.

Flow shapes. Given the flow ϕ induced by P and $\alpha \geq 0$, the flow shape corresponding to α is the collection of all stable manifolds of critical points x with $h(x) \leq \alpha$. It can be shown that for every fixed value of α the corresponding flow shape and alpha shape are homotopy equivalent.[8]

We intent to use the boundary of the flow complex as a representation of the gamut boundary. Note that the corresponding alpha shape is just the Delaunay triangulation of P whose boundary is the convex hull of P. The reason why we do not have to turn to a flow shape for some suited value of α in order to get a geometrical reasonable model of the point set is, that the flow complex already adapts well to the "geometry" of the point set. That is, one advantage of our approach over alpha shapes is that we do not have to find an appropriate

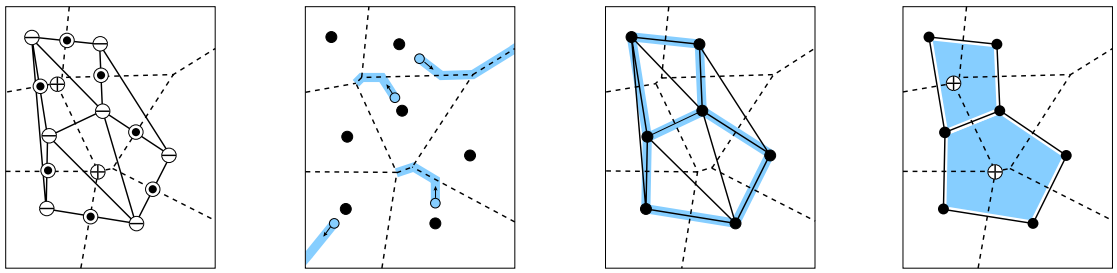

Figure 3. From the left: 1) The local minima $\ominus$, saddle points $\odot$ and local maxima $\oplus$ of the distance function induced by the sample points (local minima). 2) Some orbits of the flow induced by the sample points. 3) The stable manifolds of the saddle points. 4) The stable manifolds of the local maxima.

value of α. However, the major drawback of this direct approach - a drawback it shares with taking the convex hull or an alpha shape - is that it is far too costly to compute these structures for the point sets we are dealing with in gamut mapping. In order to compute gamut boundaries very fast we propose to quantize the input point set to a grid.

Grid and voxel. A grid is subset of $\mathbb{Z}^3$ of the form $\{l_i, l_i+1, \ldots, u_i\} \times \{l_j, l_j+1, \ldots, u_j\} \times \{l_k, l_k+1, \ldots, u_k\}$. An often used substructure of a grid is a voxel. A voxel is a subset of the grid that contains eight points of the form $(i,j,k), (i,j,k+1), (i,j+1,k), (i,j+1,k+1), (i+1,j,k), (i+1,j,k+1), (i+1,j+1,k)$ and $(i+1,j+1,k+1)$.

Of course we can also compute the flow complex of the point set P after we have quantized it to a grid, but this would not be any faster. To get an approximation of the flow complex of the quantized point set P the following observation helps: the flow complex contains all points in $\mathbb{R}^3$ whose orbits do not continue to infinity. We use this observation to define the discrete flow complex.

Discrete flow complex. The discrete flow complex is the subset of the grid that contains all grid points whose orbit does not continue to infinity. The discrete flow complex is a superset of all points from P which became grid points after quantization.

Still it is not obvious that the discrete flow complex can be computed much faster than the flow complex of the same set of grid points. As we will see later it turns out that we can restrict most of our computations when computing the discrete flow complex to the points on the grid which results in a fast algorithm to compute the discrete flow complex. In this algorithm we use a discrete versions of the distance function induced by the point set P.

Discrete distance function and Voronoi diagram. The discrete distance function is just the specialization of the continuous distance function from $\mathbb{R}^3$ to the grid to which the point set P has been quantized. The discrete Voronoi diagram assigns to every point of the grid the coordinates of its closest point in the quantized point set P. The remarkable fact is that both, the discrete distance function and the discrete Voronoi diagram can be computed in time, linear in the size of the grid.

3. IMAGING BACKGROUND

We assume that the colors contained in an image are given as a subset of of a RGB cube. That is, every color point is given by three integer coordinates between 0 and 255. These points are mapped into a device independent color space which we assume to be CIE Lab though our method does not depend on this choice and should also work for other device independent color spaces, e.g., CIECAM, mLAB or CIELUV. The CIE Lab space is parameterized using three orthogonal coordinate axes called L, a and b. The L-value of a point in CIE Lab represents lightness, the a-value represents redness-greenness and the b-value represents yellowness-blueness. The ranges of the L, a and b values are: $0 \leq L \leq 100$ and $-128 \leq a, b \leq 127$. In general the color points do not have integer coordinates anymore after being mapped into the CIE Lab space. Our goal is to compute a polyhedral surface in the CIE Lab space that encloses all the color points contained in an image, i.e., its gamut. Among all such surfaces we want to choose one that is efficiently computable and fits the gamut well. In order

to satisfy our first objective, i.e., efficient computation, we quantize the color points to a grid. The quantization is done as follows: the L-axis of the CIE Lab space is discretized to 101 equal distant values and the a and b axes are both discretized to 256 equal distant values each. Thus the interval length in each axis is $\Delta E = 1$. In doing so we obtain a grid of size $101 \times 256 \times 256$ which is our basic data structure. The maximal error we make by quantizing the continuous CIE Lab values is half of the length of the diagonal of a voxel in the grid, i.e., the error is smaller than $\frac{\sqrt{3}}{2}$. This error is negligible, as even under ideal conditions most humans can perceive only distances larger than one. The quantized gamut consist of all color points mapped to their closest grid point and our goal becomes to compute a well suited polyhedral surface that encloses the quantized gamut.

4. THE ALGORITHM

In this section we describe an algorithm that computes such a polyhedral surface containing the quantized color points. The algorithm actually computes the discrete flow complex from which a polyhedral surface can be derived easily. The algorithm at first computes the distance function and the flow that are both induced by the color points. The algorithm decides for every grid point depending on its orbit whether it belongs to the discrete flow complex or not. The algorithm consists of the following steps:

1. Initial grid labeling.
2. Computation of the discrete distance function.
3. Determination of saddle points and steepest ascent.
4. Determination of the flow and relabeling.

We will now explain all steps in detail. In doing so we will account for both, algorithmic and implementation issues.

Initial grid labeling. The basic data structure on which the algorithm builds is a grid where every grid point is labeled with the number of occurences of the corresponding colors in the source image. In particular a grid point is labeled with zero if its corresponding colors do not appear in the source image. Note that because of the quantization a grid point may have more than one corresponding color. The labeled grid representation is used and modified throughout the algorithm. After the termination of the algorithm the resulting discrete flow complex is described by this data structure, too: Every point that belongs to the shape has a label different from zero while points labeled with zero lie outside the shape. Just for computing the gamut boundary a simple zero-one-labeling would be sufficient, i.e., a grid point receives label one if at least one of its corresponding colors is present in the source image and zero otherwise. By counting the number of occurrences of a color we store additional information. This information can be used as a density measure for color points when developing a gamut mapping algorithm.

For efficiency reasons we reduce the grid after labeling it. Every grid plane at the boundary of the grid can be removed from the grid if it only contains grid points labeled zero. We recusively remove such grid planes as long as it is possible.

Computation of the discrete distance function. After labeling and reducing the grid we compute for every grid point the quadratic Euclidean distance to its closest color point, i.e., the closest grid point with a label different from zero. We work with the quadratic value in order to avoid square root computations. This allows us to work with integers only. Saito and Toriwaki[9] developed a time and memory efficient algorithm for computing the quadratic Euclidean distance function and the discrete Voronoi diagram on a grid. Their algorithm is iterative and scans the grid six times, twice in every dimension. The basic idea behind their algorithm is that the distance value at a grid point can be computed from the distance values at the grid points in the same row or column, respectively.

Determination of critical points and steepest ascent. In this step we determine for every grid point if it is critical or not. If not we store the direction of steepest ascent at the point. We need this direction later to determine the flow. There are three types of critical points of the distance function: local maxima, local minima

and saddle points. Local minima are directly given as the color points themselves and thus have a positive label. This is why we have to consider only grid points with a label equal to zero in order to find the critical points. For every such grid point we compute the ascent of the distance function in the directions pointing to all those neighbors on the grid that have a larger distance function value. Among these ascents we search for the maximum. Here we chose the 26 neighborhood on the grid. The following can happen:

- None of the neighbors has a higher distance function value than the grid point: Then we say the grid point is a discretized local maximum. As local maxima belong to the resulting flow complex we set its label to one.
- There is exactly one neighbor in whose direction the ascent is maximal. Then the grid point is not critical. We store the direction to this neighbor as the direction of steepest ascent at the grid point.
- There are at least two neighbors in whose direction the ascent is maximal:
  - If those neighbors lie in opposite directions we say the point is a saddle point [†]. Since saddle points belong to the resulting flow complex we set its label to one.
  - If those neighbors do not lie in opposite directions the grid point is not critical. Thus we have to store a direction of steepest ascent at the point. In order to determine which direction to store we use symbolic perturbation, i.e., we store the direction to the neighbor that has the smallest index according to the lexicographical order on the coordinates.

As described above we only approximate the critical points. In general the critical points need not lie on the grid. With this approximation the algorithm becomes more efficient without sacrificing too much of geometric accuracy. We also only approximate the direction of steepest ascent at non-critical points. This also does not change the resulting shape much.

Determination of the flow and relabeling. In this step we determine for every grid point that is not critical if it flows to infinity or not. A grid point is said to flow to infinity if it flows into the boundary of the grid. Beginning at a grid point p we follow the direction of steepest ascent until we reach a critical point or the boundary of the grid. If we reach a critical point we label p with one. If we reach the boundary of the grid then p flows to infinity and thus its label stays unchanged. In order to speed up the computation we exploit the fact that every grid point that we visit while following the flow of p is treated in the same way as p, i.e., all grid points on the orbit of p flow to the same critical point as p or to infinity if p does so. If we reach a grid point that is not labeled zero we can already stop following the direction of steepest ascent. In this case p and all the grid points on orbit of p that we have visited so far get the label one. If we reach a grid point for which we have already determined that it flows to infinity we can also stop. In the latter case p and all the grid points on the orbit of p have flow to infinity, too.

Once we have the determined the flow and the relabeled grid we obtain the discrete flow complex as the set of all grid points whose label is different from zero. The grid points that are labeled zero, consequently, belong to the complement of the discrete flow complex.

5. IMPLEMENTATION AND RESULTS

The algorithm was implemented using the Java2 SDK, Standard Edition, v.1.4.2.

Visualization. The output of the algorithm is just a labeling of the quantized CIE Lab space, i.e., the grid on which the algorithm works. Note that in the output of the algorithm all grid points have non-negative labels. All grid points that have positive labels belong to the discrete flow complex and all grid points that are labeled zero belong to its complement. To visualize the discrete flow complex we implemented a simple algorithm that meshes the interface between the discrete flow complex and its complement with squares that are all of the same size. A voxel in the grid is on the interface between the discrete flow complex and its complement if it contains vertices, i.e., grid points, that have positive labels as well as vertices that have label zero. The mesh consists of

[†]We say that neighbors lie in opposite directions if they do not belong to to the same voxel in the grid

Image	Image size	Number of colors	Grid size	Running time
RGBcolors	393216	220851	101 x 174 x 206	18.0s
Woman	196608	21025	94 x 76 x 109	4.4s
Bouquet	196608	45219	94 x 92 x 133	5.9s
Teapot	172800	2702	101 x 67 x 59	2.4s

Table 1. Additional data for the test images

all squares in the boundary of the voxels on the interface that all have positive labels. The rough appearance of the gamut boundaries in Figure 4 is due to our coarse meshing strategy. Note that the mesh is computed only for visualization purposes and is not meant to be used in the gamut mapping algorithms that we plan to develop in the future. Actually, in order to derive a gamut mapping from the discrete flow complex it might not be necessary to have an explicit mesh that represents the gamut boundary. We can imagine to do the meshing implicitly when evaluating geometric primitives like line-gamut intersections. To render the meshes we used the open source program Geomview which is available for Unix platforms at `http://www.geomview.org`.

Results. In order to test the algorithm we used our implementation on a PC with a Pentium III (1600MHz) processor. For the tests we used a suite of test images, both natural and artificial. In Figure 4 we show results for four test images: RGBcolors, Woman, Bouquet and Teapot. The latter three are test images from the International Organization for Standardization (ISO) and are available at `http://www.iso.org`. RGBcolors is a generated image that contains the colors of a quantized RGB-cube. Each axis of the cube is quantized to 64 values. The figure shows the four test images and their image gamuts. Note that the test images are color images that are only printed in gray scale here.

We summarize additional data on the tests and the used images in Table 1. There the image size is the number of pixels of a test image, the number of colors is the number of different color points on the grid, the grid size is the number its extension after it has been reduced in the first step of the algorithm and the running time is measured in seconds.

Discussion. Essentially the running time of the algorithm is determined by the size of the grid after reduction. This is due to the fact that all steps besides the initial labeling of the quantized CIE Lab space and its reduction require only a linear scan of the reduced grid. In other words, not the image size or the total amount of colors contained in an image is crucial for the running time, but the color distribution within the color space. Table 1 shows that the discrete flow complex of an image gamut can be computed fast. Still these running times are not sufficient in an online scenario. But note that our implementation is only a prototype implementation where no optimization was done. A more sophisticated implementation in another programming language, e.g., C++, could be much faster and would enable online applications.

As mentioned before the rough appearance of the gamut boundaries in Figure 4 is due to our coarse meshing. The gamut boundary of the Teapot however has not only a rough surface but also an inherent cliffy shape. This can be explained by the fact that the image contains only few colors that are not very well distributed. In general one can make up the following relationship: the more colors are contained in an image and the better these colors are distributed, the smoother the image gamut is. For example the gamut of the image RGBcolors, which is very similar to the gamut of an RGB device, contains many well distributed colors. Existing gamut mapping algorithms require a somehow nice surface. Algorithms based on operations like the intersection of a line with the gamut boundary fail, if the surface is too cliffy. This is why in general we cannot directly apply existing gamut mapping algorithms to the discretized flow complex of the image gamut. In order to arrive at a gamut mapping algorithm based on the discrete flow complex there are two possibilities: on the one hand the flow complex could be smoothed. Then existing algorithms could be applied. On the other hand one could develop completely new algorithms that can also deal with cliffy surfaces. Only the second approach permits

to tap the full potential of image gamuts. One should not forget that the overall goal when designing a gamut mapping algorithm still is to perform well in psychophysical tests. To achieve this, new requirements like having a good local mapping behavior, see Zolliker,[10] also have to be taken into account. This also indicates that it is beneficial to develop new algorithms.

6. CONCLUSIONS

We have presented a new method for computing gamut boundaries fast and geometrically accurate. Our approach to do so is based on the discrete flow complex. In our experiments it turned out that the discrete flow complex is well suited for this task. In the future we want to use the computed gamut boundaries for fast, high-quality, image-dependent gamut mapping in three dimensions. At the moment it looks like that existing gamut mapping algorithms cannot be adapted easily in order to work with the gamut boundaries that we compute here. These boundaries can have a quite intricate geometry. That is, new gamut mapping algorithms have to be developed.

REFERENCES

1. H. Chen and H. Kotera, "Gamma-compression gamut mapping method based on the concept of image-to-device," *Journal of Imaging Science and Technology* **45**, pp. 141 – 151, 2001.
2. J. Morovic and M. R. Luo, "The fundamentals of gamut mapping: A survey," *Journal of Imaging Science and Technology* **45 (3)**, pp. 283–290, 2001.
3. J. Morovic in *Digital Color Imaging*, G. Sharma, ed., *The Electrical Engineering and Applied Signal Processing Series*, handbook 10th chapter, Gamut Mapping, pp. 639 – 685, CRC Press LLC, 2003.
4. H. Chen and H. Kotera, "Three-dimensional gamut mapping method based on the concept of image dependence," *Journal of Imaging Science and Technology* **46 (1)**, pp. 44 – 62, 2002.
5. T. J. Cholewo and S. Love, "Gamut boundary determination using alpha-shapes," in *Color Science, Systems and Applicatio*, *Proceedings of IS&T and SIDs Seventh Color Imaging Conference*, pp. 200 – 204, 1999.
6. J. Giesen and M. John, "The flow complex: A data structure for geometric modeling," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 285 – 294, 2003.
7. H. Edelsbrunner in *Discrete & Computational Geometry: The Goodman-Pollack Festschrift*, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., ch. Surface reconstruction by wrapping finite point sets in space, pp. 379 – 404, 2002.
8. T. K. Dey, J. Giesen, and M. John, "Alpha-shapes and flow shapes are homotopy equivalent," in *Proceedings of the 35rd Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 493 – 502, 2003.
9. T. Saito and J. Toriwaki, "New algorithms for euclidean distance transformation on an n-dimensional digitized picture with applications," *Pattern Recognition* **27**, pp. 1551 – 1565, 1994.
10. P. Zolliker, M. Daetwyler, and K. Simon, "On the continuity of gamut mapping algorithms," in *Color Imaging X: Processing, Hardcopy and Applications*, SPIE/ IS&T, 2005. To be published.

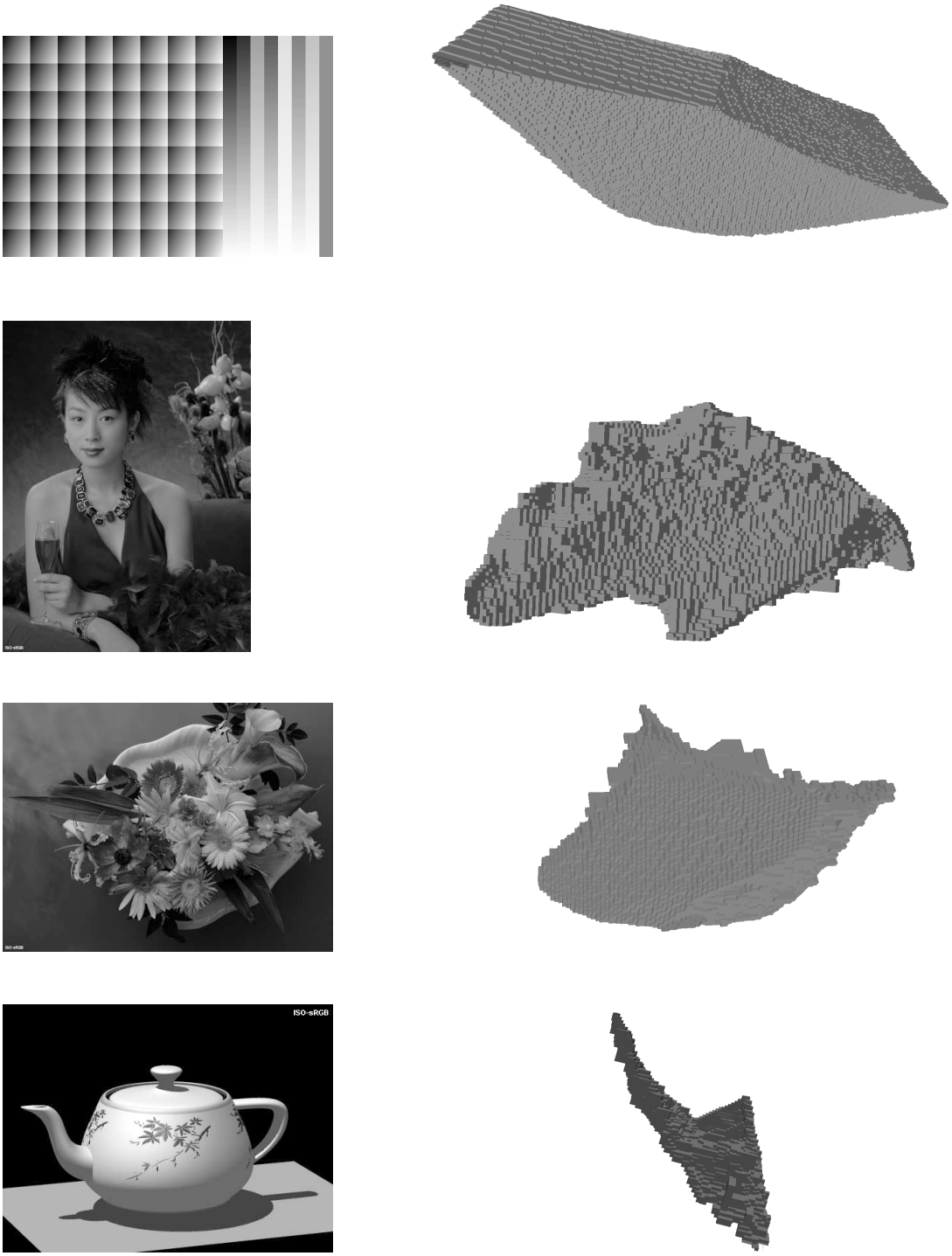

Figure 4. The test images and the discrete flow complex of their gamuts. The images from top to bottom: RGBcolors, Woman, Bouquet, Teapot