Journal of the Optical Society of America A
OPTICS, IMAGE SCIENCE, AND VISION

Median cut color quantization algorithm: retrospective

M. Emre Celebi[1,*] and María-Luisa Pérez-Delgado[2]

[1]Department of Computer Science and Engineering, University of Central Arkansas, Conway, Arkansas 72035, USA
[2]Department of Computer Science and Automatics, Universidad de Salamanca, Escuela Politécnica Superior de Zamora, Av. Requejo, 33, Zamora 49022, Spain
*ecelebi@uca.edu

Received 21 August 2025; revised 4 January 2026; accepted 12 January 2026; posted 12 January 2026; published 29 January 2026

Color quantization, reducing the number of distinct colors in a given image with minimal distortion, is a common image processing operation with many applications in visual computing. Heckbert's *median cut* algorithm, which dates back to the early 1980s, is generally considered the first true color quantization algorithm. Heckbert's seminal work generated numerous subsequent studies extending his algorithm in various ways. In this retrospective, we present a detailed analysis of the median cut algorithm and demonstrate how it influenced later color quantization, vector quantization, and data clustering algorithms.

https://doi.org/10.1364/JOSAA.577058

1. INTRODUCTION: THE BIRTH OF COLOR QUANTIZATION

High-resolution true-color images have become ubiquitous over the past quarter century [1]. Such images often have thousands of distinct colors, complicating nearly all operations that can be performed on them. Color quantization (CQ) is an image processing operation that reduces the number of distinct colors in a given image with minimal distortion.

Figure 1 shows the coloring pencils image (Ref. [2], CC BY-SA 3.0 license, 768 × 512 pixels) quantized to 4, 16, 64, and 256 colors using the median cut algorithm. It can be seen that the reproduction with 256 colors is nearly indistinguishable from its original.

Let $\boldsymbol{I} = [\boldsymbol{i}_{r,c}]_{H\times W}$ be a $W \times H$ true-color (i.e., 24-bit) red–green–blue (RGB) input image, with $\boldsymbol{i}_{r,c}$ denoting the pixel at the intersection of row r ($\in \{1, \dots, H\}$) and column c ($\in \{1, \dots, W\}$). The 8-bit red ($i_{r,c,1}$), green ($i_{r,c,2}$), and blue ($i_{r,c,3}$) components of pixel $\boldsymbol{i}_{r,c}$ attain values in $\{0, \dots, 255\}$. Further, let K denote the desired number of colors in the output image; typically, we have $K \in \{2, \dots, 256\}$. CQ comprises two phases: color palette (CPAL) design and pixel mapping. In the former phase, a palette $\mathcal{C}$ of size K representing the colors in $\boldsymbol{I}$ is generated, while, in the latter phase, each pixel in $\boldsymbol{I}$ is assigned to the nearest color in $\mathcal{C}$. The output of CQ is a reduced-color RGB image $\tilde{\boldsymbol{I}} = [\tilde{\boldsymbol{\imath}}_{r,c}]_{H\times W}$ containing *only* the K palette colors. The objective is to minimize the distortion between the input and output images, that is,

$$\min_{\mathcal{C}=\{\boldsymbol{c}_1,\dots,\boldsymbol{c}_K\}\subset \boldsymbol{I}} D_{\mathcal{C}}(\boldsymbol{I}, \tilde{\boldsymbol{I}}), \tag{1}$$

such that the pixel at location (r, c) of $\tilde{\boldsymbol{I}}$ is given by

$$\tilde{\boldsymbol{\imath}}_{r,c} = \arg\min_{\boldsymbol{c}_k\in\mathcal{C}} d(\boldsymbol{i}_{r,c}, \boldsymbol{c}_k), \tag{2}$$

where $\mathcal{C} = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$ is the CPAL, and $d(\cdot, \cdot)$ and $D(\cdot, \cdot)$ are measures of distortion between a pair of colors and color images, respectively (with the latter being a function of the former).

Due to its analytical tractability, it is customary to take $d = \ell_2^2$ (squared Euclidean distortion) and D as the sum-of-squared-distortion (SSD), that is,

$$D_{\mathcal{C}}(\boldsymbol{I}, \tilde{\boldsymbol{I}}) = \sum_{r=1}^{H}\sum_{c=1}^{W} \left\|\boldsymbol{i}_{r,c} - \tilde{\boldsymbol{\imath}}_{r,c}\right\|_2^2, \tag{3}$$

where $\|\cdot\|_2$ is the *Euclidean norm* (*aka* the ℓ_2 *norm*) given by

$$\left\|\boldsymbol{i}_{r,c} - \tilde{\boldsymbol{\imath}}_{r,c}\right\|_2 = \sqrt{\left(i_{r,c,1} - \tilde{\imath}_{r,c,1}\right)^2 + \left(i_{r,c,2} - \tilde{\imath}_{r,c,2}\right)^2 + \left(i_{r,c,3} - \tilde{\imath}_{r,c,3}\right)^2}. \tag{4}$$

It is important to note that the ℓ_2 norm treats the three color components equally, which is problematic from a perceptual point of view, as the RGB color space is perceptually nonuniform (i.e., equal ℓ_2 distances in the space do *not* necessarily correspond to equal perceived color differences); see Section 3. However, true-color images are almost universally gamma-corrected [3], which alleviates this perceptual nonuniformity issue. Note also that there are many clustering objectives besides

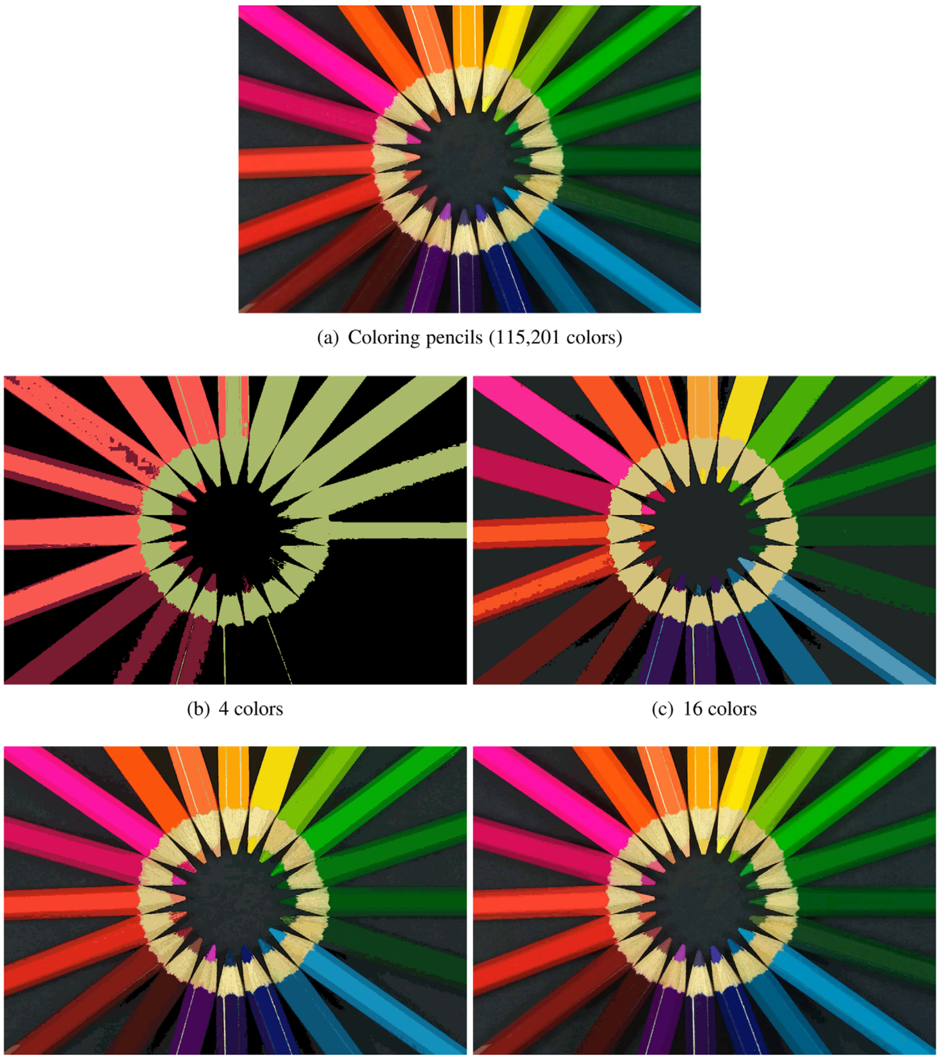


Fig. 1. Coloring pencils and its various quantized versions.

SSD, including those proposed in the fuzzy [4], information-theoretic [5], kernel [6], probabilistic [7], and spectral clustering [8] literatures.

Since natural images typically contain a broad range of colors, reproducing such images accurately with a small CPAL can be difficult. In fact, considering the number of possible colors in the RGB space ($2^{24} = 16,777,216$), it is easy to see that CQ is a large-scale combinatorial optimization problem.

Jain and Pratt [9] coined the term *color quantization* in 1972 as an application of signal quantization to color images. The authors acknowledged that the RGB components should ideally be quantized jointly as a vector (i.e., *vector quantization*) rather than independently (i.e., *scalar quantization*). However, for practical reasons (e.g., computational limitations), they investigated uniform scalar quantization applied to each color component independently. Unfortunately, such a componentwise approach disregards the spectral correlations between the components. Specifically, an independent uniform scalar quantizer tends to assign palette colors to regions where few input colors reside, as colors in natural images are hardly, if ever, distributed uniformly in the RGB space.

The *median cut* (MCUT) algorithm, developed by Heckbert first in his 1980 undergraduate thesis [10] and then published in a 1982 journal paper [11], is generally considered the first true CQ algorithm. MCUT has been so influential that it is still the most popular and widely implemented CQ algorithm four decades after its introduction, as evidenced by its implementation in numerous programming languages, including (i) C/C++ [12–16], (ii) Go [17], (iii) Java [18–20], (iv) JavaScript [21], (v) MATLAB [22], (vi) Perl [23], (vii) PHP [24], (viii) Python [25], (ix) R [26], (x) Ruby [27], (xi) Rust [28], (xii) Swift [29], and (xiii) Tcl [30]. Recent applications of MCUT include (i) video style transfer [31], (ii) graphical user interface evaluation [32], (iii) embedded simultaneous localization and mapping [33], (iv) photorealistic rendering of neural radiance fields [34], (v) photo sequence synthesis [35], (vi) scientific document visualization [36], (vii) virtual reality scientific data visualization [37], (viii) scene image text detection [38], (ix) sports video analysis [39], and (x) entertainment video summarization [40].

Heckbert motivated MCUT from a signal quantization perspective. However, it is more convenient to describe his algorithm as a divisive hierarchical clustering algorithm. Given an input dataset and an integer $K \geq 2$ denoting the desired number of clusters, a divisive clustering algorithm divides the data space into K disjoint regions using $(K-1)$ successive binary divisions. At each iteration, the algorithm selects a region (corresponding to a cluster) and divides it into two using a hyperplane. Upon termination, the algorithm returns the centroids of the resulting K regions (i.e., the arithmetic means of the data points that fall into these regions) as the cluster centers.

MCUT is simply an instance of the divisive clustering algorithm above applied to an RGB image whose colors reside in a three-dimensional color space. Specifically, MCUT selects the cluster with the greatest range on any color axis (i.e., red, green, or blue) at each iteration and divides it into two using a plane orthogonal to the same axis, passing through the *median* point (hence the name of the algorithm). This is the *iterative* MCUT algorithm Heckbert proposed in his thesis [10]. In a subsequent journal paper [11], Heckbert described a *recursive* MCUT algorithm that divides *every* cluster at each iteration. The recursive algorithm generates a CPAL in which each color represents a roughly equal number of input colors, while the iterative one does *not* impose such a restriction. It is easy to see that the iterative algorithm is more adaptive to the input color distribution than the recursive one. This is because the former algorithm divides the most elongated cluster at each iteration (to reduce its distortion), whereas the latter one divides every cluster *regardless* of its color distribution. Therefore, we will focus on the iterative algorithm in the rest of this discussion. Figure 2 illustrates this algorithm on a two-dimensional artificial dataset containing four compact and well-separated clusters [41]. It can be seen that, despite its simplicity, the algorithm recovers the underlying clusters reasonably well, except for a few misclassified peripheral data points.

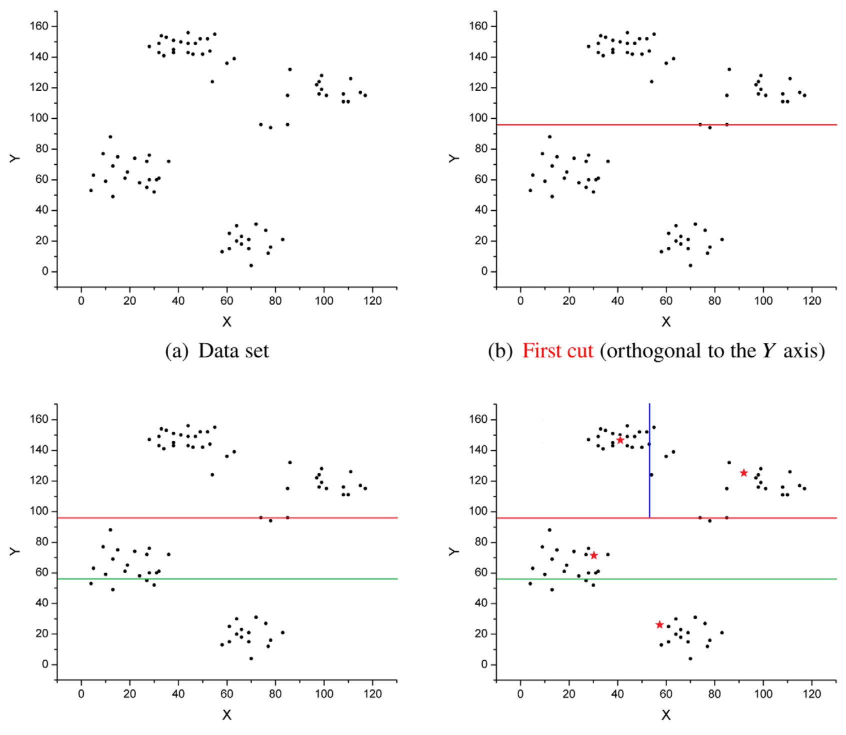
2. NOVELTIES OF MCUT

The novelties of Heckbert's work include the following.

Optimization-based formulation: As mentioned in Section 1, Heckbert formulated CQ as an optimization problem: select a CPAL that minimizes the distortion between the input and output images. For computational reasons, he adopted the SSD objective given by Eq. (3). He pointed out that, in one dimension (i.e., the case of scalar quantization), SSD can be minimized in polynomial time-using dynamic programming. Such is the case when the input is an 8-bit grayscale image [42]. He also conjectured that the SSD minimization problem is computationally intractable in higher dimensions (i.e., the case of vector quantization). His conjecture was proved nearly three decades later by Mahajan *et al.* [43], who proved that the problem is NP-hard *even* in two dimensions for $K \geq 2$.

Uniform scalar quantization: Heckbert applied uniform scalar quantization to each color component as a preprocessing step. He accomplished such a *pre-quantization* by cutting the least-significant 3 bits from each 8-bit color component. Bit cutting reduces the amount of color data to be clustered, which in turn reduces the time and memory requirements of the CPAL design phase. In addition, bit cutting can make the peaks of the input color distribution more prominent, making it easier to detect them. Bit cutting was also used in many later CQ algorithms [1].

- *Hashing*: The colors in a $W \times H$ RGB image can be stored in a two-dimensional array of size $N \times 3$, where $N = WH$ is the number of pixels in the image. However, such an array is clearly wasteful, as it stores each repeated color as many times as it occurs in the image. Assuming 3 bits per component bit cutting, an alternative data structure would have been a three-dimensional array of size $2^5 \times 2^5 \times 2^5$, in which each element corresponds to the frequency of a particular color. Although such an array allows efficient access to each of its elements, it is still wasteful, as many images do *not* contain all possible 2^{15} colors. In his thesis, Heckbert proposed a more space-efficient storage scheme based on a hash table, a generalization of the array data structure that uses a hash function to map keys (i.e., RGB colors) to indices in an underlying array [44] (while each key maps to a distinct index in an ordinary array, the mapping is many-to-one in a hash table; hence, the savings in memory space.) Heckbert used a hash function that concatenates the least-significant 3 bits of the 5-bit color components. For the 640×480 test images he used, the two- and three-dimensional arrays above would have contained 900 K and 32 K elements, respectively, while the hash table contained *only* 0.5 K elements. Although hashing was already being used in multispectral image processing in the 1970s [45], Heckbert can still be considered an early adopter of this versatile data structure in color image processing. For an overview of data structures used in CQ, refer to Celebi [1].
- *Adaptive and nonuniform* CQ: Due to computational limitations, early color quantizers (e.g., the one proposed by Jain and Pratt [9]) typically employed non-adaptive (*aka* image-independent) and uniform quantization. Here, the qualifiers *non-adaptive* and *uniform* refer to the generation of a *universal* CPAL based on a given color space (e.g., RGB) and the placement of the palette colors uniformly throughout the color space, respectively. On the other hand, Heckbert proposed an adaptive (*aka* image-dependent) and nonuniform color quantizer, which generates a *custom* CPAL based on the color distribution of a given input image, with the palette colors placed nonuniformly throughout the color space. Such an adaptive and nonuniform color quantizer nearly always produces superior results to a non-adaptive and uniform one, and the more nonuniform the input color distribution, the greater the quality difference between the two quantizers.
- *Connection to divisive hierarchical clustering, decision tree induction, tree-structured vector quantizer design, and space partitioning*: As mentioned in Section 1, MCUT can be viewed as a divisive hierarchical clustering algorithm [46] applied to an RGB image. Heckbert [11] mentions in passing that his recursive algorithm is "nearly identical" to Bentley's algorithm for constructing a *k-d tree* [47], a multidimensional search tree that generalizes the binary search tree; the most prominent difference between the two algorithms is that the former divides the data points along the longest color axis, whereas the latter cycles through the axes, that is, if there are D axes numbered 1 through D, axis $((L \bmod D) + 1)$ is used as the dividing axis at recursion level $L \in \{0, 1, \ldots\}$. It is also *not* difficult to see the resemblance between MCUT and the algorithms used for binary decision tree



(a) Data set (b) First cut (orthogonal to the Y axis)

(c) Second cut (orthogonal to the Y axis) (d) Third cut (orthogonal to the X axis)

Fig. 2. Illustration of MCUT on a two-dimensional artificial dataset (red stars in subfigure (d) represent the cluster centroids).

induction [48], binary tree-structured vector quantizer design [49], and binary space partitioning [50].

- *Lloyd's k-means algorithm*: Like most of its successors, MCUT is a heuristic algorithm with *no* guarantee on the quality of its solution. For this reason, Heckbert [11] suggested using Lloyd's k-means algorithm [51,52] to improve the output of MCUT, thereby ensuring *at least* local optimality of the resulting CPAL. His choice was *not* coincidental, as the SSD objective given by Eq. (3) is precisely the objective minimized by k-means. *Without* presenting any numerical results, he stated that such a postprocessing step reduced MCUT's distortion *only* slightly. Later, Wan *et al.* [53] reached a similar conclusion about their MCUT variant, likely based on limited experimentation with *only* three images and two palette sizes ($K \in \{8,64\}$). Furthermore, many early CQ researchers (e.g., [53–56]) considered k-means to be prohibitively slow due to its iterative nature; in general, the number of iterations required to converge to a locally optimal solution *cannot* be predicted in advance and depends on the number, dimensionality, and distribution of the data points, the number of clusters sought, and the initial centers [1]. Celebi [57,58] showed that an accelerated k-means implementation can not only be fast but also significantly improve the output of MCUT and many of its variants, both qualitatively and quantitatively. Recent studies [59–61] further improved the efficiency and effectiveness of k-means, making it a highly practical CQ algorithm.
- *Dithering*: Due to its lossy nature, CQ inevitably leads to loss of color and fine details. For example, representing a large, almost uniformly colored region with a small or poorly designed CPAL can lead to the appearance of bands of uniform colors with visually disturbing contours in between. Heckbert suggested eliminating such *false contours* using *dithering* [62], a classical image processing operation that exploits the spatial integration property of the human visual system to create the illusion of more colors. Figure 3 compares the coloring pencils image from Fig. 1 with or without Floyd–Steinberg [62] dithering applied after MCUT quantization. Observe that the output images with 4, 16, 64, and 256 colors from Fig. 1 are duplicated in the left column of Fig. 3 for convenient comparison. It can be seen that for the smallest four-color CPAL, dithering made a substantial difference by recovering many details (e.g., the bodies of entire pencils in the lower-right corner), whereas for the larger 64- and 256-color CPAL, it barely made any difference. Finally, in the case of the 16-color CPAL, dithering mitigated or even eliminated some of the false contours (e.g., the pencil at nearly the 9 o'clock position and the pencil below it). Unfortunately, these visual improvements came at the expense of noticeable noise in the background and on the bodies and tips of most pencils.
- *Accelerated pixel mapping*: Once the CPAL is designed, the output image is generated by pixel mapping. For each pixel in the input image, the trivial pixel-mapping algorithm performs an exhaustive search for its nearest palette color given by Eq. (2). This mapping is clearly inefficient, as, for any given input pixel, most palette colors will likely be too far to be its nearest color.

Heckbert proposed the first accelerated pixel-mapping algorithm in the CQ literature, *locally sorted search* (LLS), which starts by dividing the RGB cube into $L \times L \times L$ subcubes, where $L \geq 2$ is a user-defined number (i.e., $L = 8$ for $K = 256$ colors

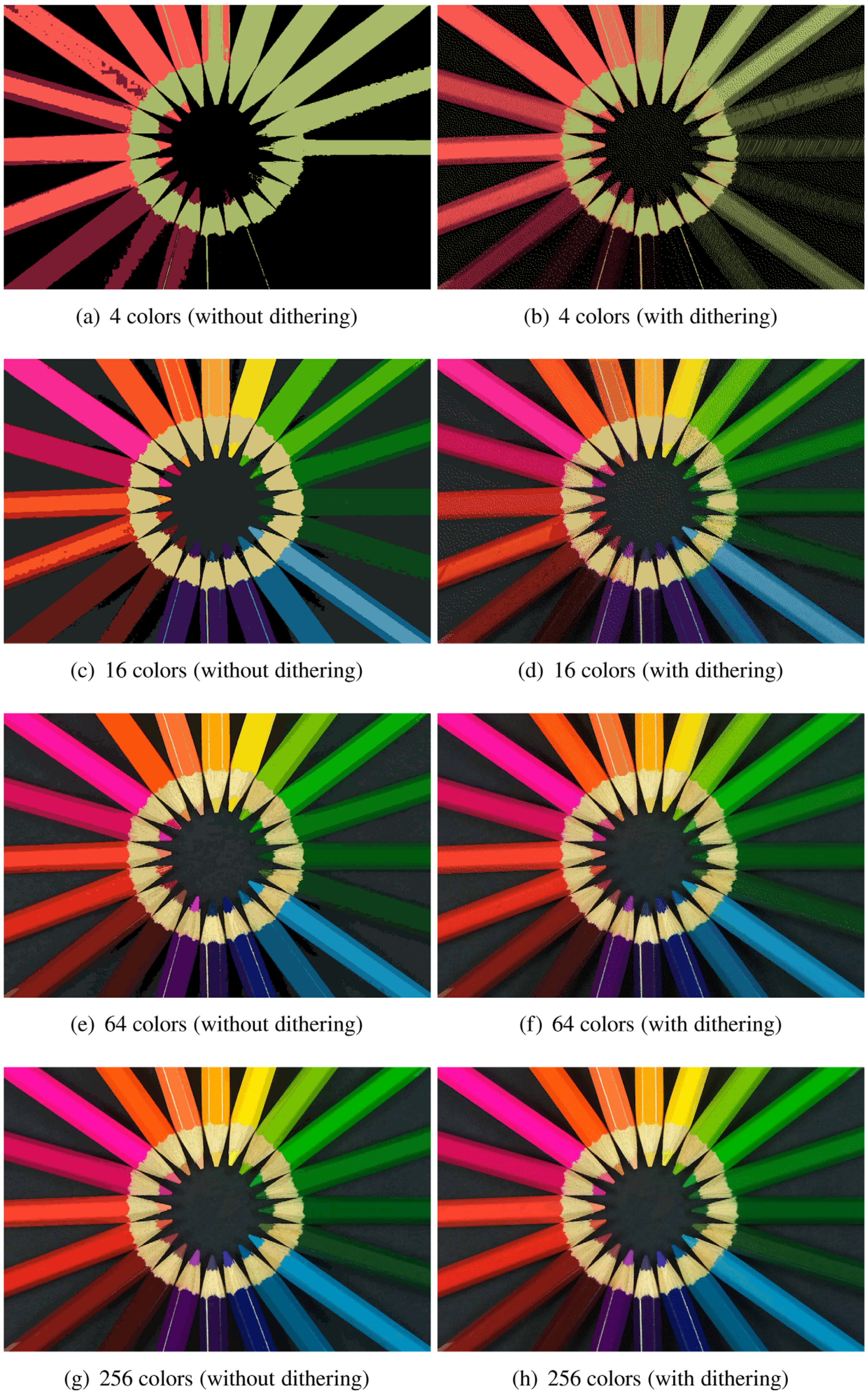

(a) 4 colors (without dithering)
(b) 4 colors (with dithering)
(c) 16 colors (without dithering)
(d) 16 colors (with dithering)
(e) 64 colors (without dithering)
(f) 64 colors (with dithering)
(g) 256 colors (without dithering)
(h) 256 colors (with dithering)

Fig. 3. Comparison of coloring pencils with (right column) or without (left column) dithering applied after MCUT quantization.

[10,11]). For each subcube, the algorithm maintains a sorted list of palette colors that are nearest to some color in that subcube. Given an input pixel, LLS first determines the subcube containing its color and then performs an exhaustive search in that subcube's list. There are three problems with LLS: (i) it accelerates the trivial algorithm *only* slightly (Heckbert reported a mere three-fold acceleration on one of his test images for $K = 256$); (ii) it divides the RGB cube uniformly regardless of the input color distribution; and (iii) it is nontrivial to estimate the optimal L value for a given image. Many accelerated pixel-mapping algorithms have been proposed since Heckbert; refer to Celebi [1] for an overview.

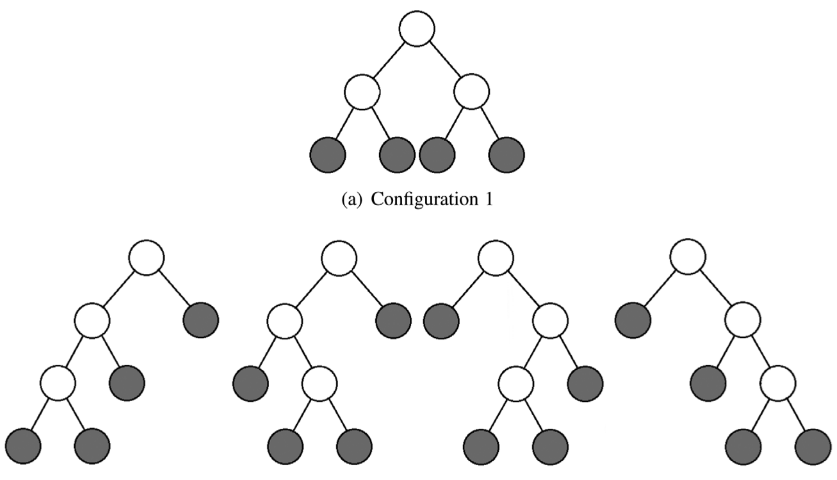


Fig. 4. All possible ways to obtain four clusters using three successive binary divisions (shaded nodes represent the clusters).

3. EXTENSIONS OF MCUT

Future work suggested by Heckbert includes the following.

- *Alternative color spaces and distortion measures*: For computational reasons, Heckbert adopted the RGB space and the ℓ_2^2 distortion measure, a combination still prevalent in color image processing. However, he suggested that performing CQ in a perceptually more uniform color space using an appropriate distortion measure (e.g., CIELAB and ℓ_2) may lead to better results. To this date, very little work has been done to investigate the effects of color space and distortion measure on CQ [1], *at least* in the algorithmic literature (refer to Nieves *et al.* [63] for a colorimetric approach based on the CIELAB color space).
- *Alternative divisive clustering strategies*: In a divisive clustering algorithm, a cluster is selected at each iteration and divided into two using a hyperplane. Although the selected cluster can be divided into $B > 2$ subclusters, most divisive algorithms adopt the binary strategy for simplicity, which is what we assume in the remainder of this discussion. Observe that the extreme case of a K-ary divisive strategy corresponds to a partitional clustering algorithm rather than a hierarchical one.

Consider a dataset $\mathcal{X}$ with N data points in $\mathbb{R}^D$. It is easy to see that the binary divisive strategy represents $\mathcal{X}$ as a full binary tree with the following properties: (i) the root node represents $\mathcal{X}$, while every other node represents a proper subset of $\mathcal{X}$; (ii) a non-leaf node's children divide their parent's elements into two subsets; and (iii) the leaf nodes form a *partition* of $\mathcal{X}$ (i.e., the corresponding subsets of $\mathcal{X}$ are exhaustive, mutually exclusive, and nonempty).

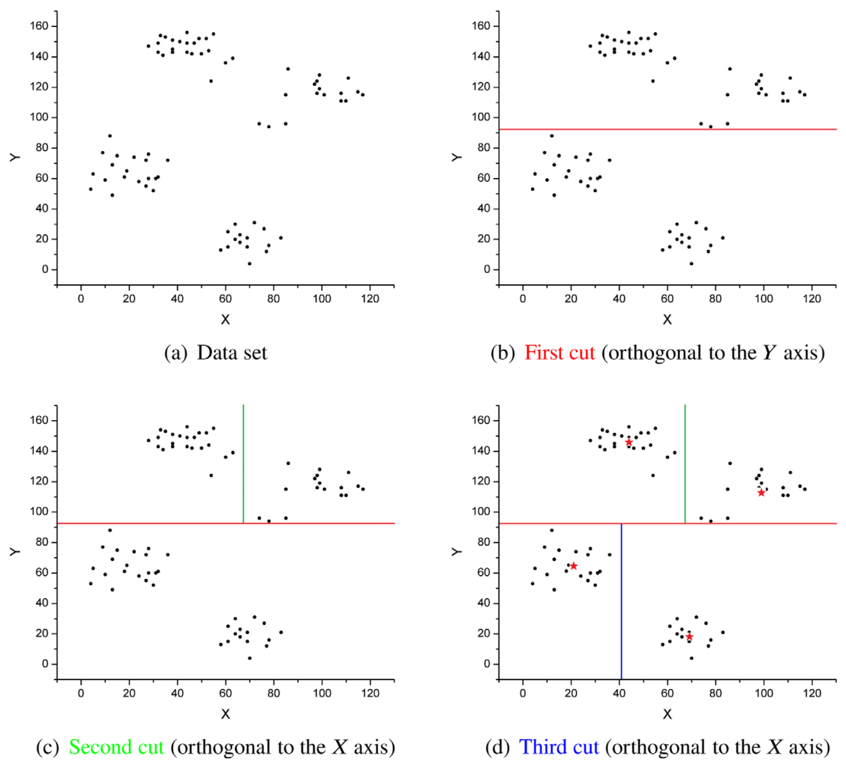
The number of $(K-1)$ successive binary divisions of $\mathcal{X}$ then equals the number of full binary trees with K leaves given by $(2K-2)!/(K!(K-1)!)$, which can be approximated by $4^{K-1}/\sqrt{\pi(K-1)^3}$ as $K \to \infty$. As an example, Fig. 4 illustrates the five possible ways to obtain $K = 4$ clusters using three successive binary divisions. Observe that a recursive algorithm (i.e., one that divides every cluster at each iteration) can generate *only* the first configuration, whereas an iterative one can generate all five configurations.

Now, consider the case of $K = 2$, where we have a single cluster corresponding to the entire dataset $\mathcal{X}$. It can be shown that the number of ways to divide $\mathcal{X}$ using a hyperplane is on the order of N^D [64]. In fact, not surprisingly, the SSD minimization problem has been proven to be NP-hard *even* for two clusters for $D \geq 2$ [65,66]. Hence, both determining an optimal order of binary divisions for $\mathcal{X}$ and optimally dividing $\mathcal{X}$ (or any of its subsets) into two are computationally intractable problems.

The computational hardness of optimal binary divisive clustering prompted researchers to propose numerous efficient heuristics since the early 1960s. These heuristics can be characterized by the following decisions they make at each iteration: (i) the cluster to divide, (ii) the orientation of the dividing hyperplane, and (iii) the position of the dividing hyperplane (*aka* the dividing point).

In MCUT, the cluster with the greatest range on any color axis is selected at each iteration. This cluster is then divided using a plane orthogonal to the axis above, passing through the median point on the same axis. Essentially, MCUT divides the *longest* cluster at each iteration, where the length of a cluster is measured along the color axes. This is a computationally efficient approach, but the most elongated cluster is *not* necessarily the one with the greatest distortion. From a distortion minimization perspective, then, a more sensible choice would have been the cluster with the greatest SSD. Such a divisive CQ heuristic was first proposed by Wan *et al.* [53]. Heckbert also suggested an alternative, where the cluster with the greatest variance on any color axis is selected at each iteration, which is then divided using an orthogonal plane passing through the data point minimizing the sum of variances of the resulting two subclusters. Milvang [67] and Wu [68] developed the earliest CQ algorithms based on Heckbert's variance minimization heuristic.

Heckbert's rationale for dividing at the median point is that the resulting two subclusters will be nearly equal in size. This makes sense when building a multidimensional search tree such as a k-d tree, but for divisive clustering, there is *no* justification to require an almost even division regardless of the data distribution [53]. In fact, it can be shown that an SSD-optimal division of a multivariate normal cluster [66] requires the hyperplane



(a) Data set
(b) First cut (orthogonal to the Y axis)
(c) Second cut (orthogonal to the X axis)
(d) Third cut (orthogonal to the X axis)

Fig. 5. Illustration of VARCUT on the dataset shown in Fig. 2 (red stars in subfigure (d) represent the cluster centroids).

to be orthogonal to the dominant eigenvector of the cluster covariance matrix (rather than being orthogonal to the longest axis) and pass through the mean (rather than the median) point on the same axis. It can also be shown that the mean is closer to the optimal dividing point than the median for *any* non-symmetric continuous univariate distribution [54]. Orchard and Bouman [55] proposed the first CQ algorithm based on the aforementioned SSD-optimal strategy. Later, Wu [56] relaxed the multivariate normality assumption by sweeping a plane orthogonal to the dominant eigenvector, passing through the data point that maximizes the reduction in the SSD. Note that when the dividing plane is orthogonal to a color axis at each iteration (as in MCUT), the RGB cube is divided into rectangular boxes using axis-parallel cuts. By contrast, when the dividing plane is orthogonal to the dominant eigenvector, which is a linear combination of the color axes, at each iteration, the RGB cube is divided into convex polyhedra using oblique cuts.

A particularly simple and efficient heuristic divisive CQ algorithm, dubbed VARCUT, divides the cluster with the greatest SSD using a plane orthogonal to the color axis with the greatest variance, passing through the mean point on the same axis [69]. Figure 5 illustrates this heuristic on the dataset shown in Fig. 2. It can be seen that VARCUT overcomes MCUT's limitations and successfully recovers the underlying clusters.

Various divisive clustering algorithms can be designed by modifying the aforementioned decisions; refer to Celebi [1] for a detailed survey.

- *Integrated* CQ *and dithering*: Heckbert suggested eliminating false contours in the output image using the Floyd–Steinberg dithering algorithm [62]. However, while CQ aims to minimize the distortion between the input and output images, dithering tends to increase this distortion. For this reason, Heckbert stated that treating these two operations independently may *not* be ideal from a distortion minimization perspective. For algorithms that integrate CQ and dithering, refer to [70–72].
- *Interactive* CQ: In some cases, the region of interest (ROI), e.g., the eyes, occupies a small area in the input image. Most CQ algorithms tend to neglect such regions in their zeal to minimize distortion. Heckbert suggested an interactive CQ algorithm that allows the user to manually specify one or more ROIS. The colors in these ROIS are then weighted more in the CPAL design phase, increasing their representation in the resulting palette. However, the potential of such an interactive CQ algorithm, e.g., in non-photorealistic rendering, remains unexplored.

4. MORE RECENT DEVELOPMENTS IN CQ

More recent developments in the CQ literature that were *not* anticipated by Heckbert include the following.

- *Dynamic* CQ: Popular CQ algorithms assume a user-defined, *static* palette size. Various *dynamic* CQ algorithms have been proposed over the past two decades that can automatically determine the palette size at run time [1]; refer to Tirandaz *et al.* [73] for a recent study.
- *Metaheuristic-based* CQ: Some of the most recent CQ algorithms are based on nature-inspired metaheuristics [74], including simulated annealing, variable neighborhood search, genetic algorithms, evolution strategies, particle swarm optimization, and ant colony optimization. On standard objectives (e.g., SSD), these powerful algorithms produce significantly

better results than conventional algorithms such as MCUT [75]. They can also optimize more complex (e.g., perceptually based) objectives or even multiple objectives simultaneously. However, they are often considerably slower and more difficult to implement and use (due to their randomized nature and user-defined parameters). Note that some of these algorithms generate their initial solution using a conventional algorithm, typically an MCUT variant [76,77]. For references to metaheuristic-based CQ algorithms, refer to a survey by Celebi [1] and a comparative study of 10 such algorithms by Pérez–Delgado and Günen [75].

- *Task-oriented* CQ: The vast majority of CQ algorithms are based on unsupervised clustering [78]. These algorithms aim to minimize the visual distortion or, equivalently, maximize the visual fidelity between the input and output images [79]; thus, they can be termed *perceptually oriented* CQ algorithms. Recently, various *task-oriented* CQ algorithms based on convolutional neural networks have been proposed [80–88]. These algorithms differ from their perceptually oriented counterparts in two important ways. First, rather than visual fidelity, they aim to maximize accuracy on a predefined computer vision task, such as image classification or object detection. Second, they are supervised; that is, before they can be used for CQ, their internal parameters must be optimized by training. Due to their supervised formulation, these algorithms can outperform conventional CQ algorithms (e.g., MCUT) on the task for which they are trained. However, their superiority appears to be confined to the *extreme color quantization* [89] setting, in which the CPAL to be designed is very small (e.g., $K \in \{2, 4, 8\}$). Therefore, in the absence of sufficient training data or outside the aforementioned setting, perceptually oriented CQ algorithms such as MCUT and its variants are still to be preferred.

5. CONCLUSION

This retrospective examined the celebrated median cut algorithm proposed by Heckbert over four decades ago. In addition to detailing the first true color quantization algorithm, Heckbert's seminal work introduced much of the terminology used in the color quantization literature to this day, described the first divisive color quantization algorithm, proposed bit cutting as a preprocessing step and k-means clustering as a postprocessing step, recommended a hash table to compactly represent the input color data, developed the first accelerated pixel-mapping algorithm, and suggested dithering to eliminate false contours in the output image. Despite its limitations, Heckbert's work not only established color quantization as a subfield of color image processing but also inspired the development of numerous color quantization, vector quantization, and data clustering algorithms.

Funding. National Science Foundation (OIA-1946391).

Acknowledgment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors gratefully acknowledge the median cut color quantization and Floyd–Steinberg dithering implementations provided in GIMP [12].

Disclosures. The authors declare no conflicts of interest.

Data availability. Data underlying the results presented in this paper are not publicly available at this time but may be obtained from the authors upon reasonable request.

REFERENCES

1. M. E. Celebi, "Forty years of color quantization: a modern, algorithmic survey," Artif. Intell. Rev. **56**, 13953–14034 (2023).
2. M. E. Celebi and M. L. Pérez-Delgado, "CQ100: a high-quality image dataset for color quantization research," J. Electron. Imaging **32**, 033019 (2023).
3. C. A. Poynton, ""Gamma" and its disguises: the nonlinear mappings of intensity in perception, CRTs, film, and video," SMPTE J. **102**, 1099–1108 (1993).
4. J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms* (Springer, 1981).
5. J. C. Principe, *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives* (Springer, 2010).
6. M. Filippone, F. Camastra, F. Masulli, *et al.*, "A survey of Kernel and spectral methods for clustering," Pattern Recognit. **41**, 176–190 (2008).
7. G. J. McLachlan and K. E. Basford, *Mixture Models: Inference and Applications to Clustering* (Marcel Dekker, 1988).
8. L. Ding, C. Li, D. Jin, *et al.*, "Survey of spectral clustering based on graph theory," Pattern Recognit. **151**, 110366 (2024).
9. A. K. Jain and W. K. Pratt, "Color image quantization," in *Proceedings of 1972 National Telecommunications Conference* (1972), pp. 4–8.
10. P. S. Heckbert, "Color image quantization for frame buffer display," Bachelor's thesis (Massachusetts Institute of Technology, 1980).
11. P. Heckbert, "Color image quantization for frame buffer display," ACM SIGGRAPH Comput. Graph. **16**, 297–307 (1982).
12. "GIMP: GNU Image Manipulation Program" [accessed 4 January 2026], https://www.gimp.org/.
13. "Leptonica Library" [accessed 4 January 2026], http://www.leptonica.org/.
14. "libjpeg" [accessed 4 January 2026], https://libjpeg.sourceforge.net/.
15. "LibTIFF: TIFF Library and Utilities" [accessed 4 January 2026], https://gitlab.com/libtiff/libtiff.
16. "Netpbm" [accessed 4 January 2026], https://netpbm.sourceforge.net/.
17. S. Keys and A. Pervukhin, "Quant," GitHub [accessed 4 January 2026] (2018), https://github.com/soniakeys/quant.
18. "ImageJ: Image Processing and Analysis in Java" [accessed 4 January 2026], https://imagej.net/ij/.
19. N. Lagomarsini, A. Aime, D. Romagnoli, *et al.*, "Java Advanced Imaging Extension," GitHub [accessed 4 January 2026] (2025), https://github.com/geosolutions-it/jai-ext.
20. "Java Imaging Utilities" [accessed 4 January 2026], https://sourceforge.net/projects/jiu/.
21. M. Nedrich, "Palette Maker," GitHub [accessed 4 January 2026] (2017), https://github.com/mattnedrich/palette-maker.
22. H. Biswas, "CVIP Toolbox for MATLAB," GitHub [accessed 4 January 2026] (2020), https://github.com/SIUE-CVIP-Lab/Computer-Vision-and-Image-Processing-Toolbox-for-MATLAB.
23. "Image::Pngslimmer" [accessed 4 January 2026], https://metacpan.org/pod/Image::Pngslimmer.
24. K. Subileau, C. Rishøj, D. Schneidhoffer, *et al.*, "Color Thief PHP," GitHub [accessed 4 January 2026] (2025), https://github.com/ksubileau/color-thief-php.
25. "Pillow: Python Imaging Library (Fork)" [accessed 4 January 2026], https://pypi.org/project/pillow/.
26. "RImagePalette: Extract the Colors from Images" [accessed 4 January 2026], https://cran.r-project.org/package=RImagePalette.
27. N. Hussain and F. Schuindt, "Camalian," GitHub [accessed 4 January 2026] (2021), https://github.com/nazarhussain/camalian.
28. "pngquant: Lossy PNG Compressor" [accessed 4 January 2026], https://pngquant.org/.
29. S. Xu, "ImageUtils," GitHub [accessed 4 January 2026] (2017), https://github.com/showxu/ImageUtils.

30. "Reduce Colour Depth: Median Cut" [accessed 4 January 2026], https://wiki.tcl-lang.org/page/Reduce+Colour+Depth+-+Median+Cut.
31. Y. Cao, X. Xiao, R. Sun, *et al.*, "Stylefool: fooling video classification systems via style transfer," in *Proceedings of. 2023 IEEE Symposium on Security and Privacy* (2023), pp. 1631–1648.
32. M. Zen, N. Burny, and J. Vanderdonckt, "A quality model-based approach for measuring user interface aesthetics with grace," Proc. ACM Hum.-Comput. Interact. **7**, 1–47 (2023).
33. Q. Picard, S. Chevobbe, M. Darouich, *et al.*, "Image quantization towards data reduction: robustness analysis for SLAM methods on embedded platforms," in *Proceedings of 2022 IEEE International Conference on Image Processing* (2022), pp. 4158–4162.
34. A. Yu, R. Li, M. Tancik, *et al.*, "Plenoctrees for real-time rendering of neural radiance fields," in *Proceedings of IEEE/CVF International Conference on Computer Vision* (2021), pp. 5752–5761.
35. C. Mousas, C. Krogmeier, and Z. Wang, "Photo sequences of varying emotion: optimization with a valence-arousal annotated dataset," ACM Trans. Interact. Intell. Syst. **11**, 16 (2021).
36. W. Zeng, A. Dong, X. Chen, *et al.*, "VIStory: interactive storyboard for exploring visual information in scientific publications," Journal of Visualization **24**, 69–84 (2021).
37. S. Johnson, F. Samsel, G. Abram, *et al.*, "Artifact-based rendering: harnessing natural and traditional visual media for more expressive and engaging 3D visualizations," IEEE Trans. Vis. Comput. Graph. **26**, 492–502 (2019).
38. W. Zhu, J. Lou, L. Chen, *et al.*, "Scene text detection via extremal region based double threshold convolutional network classification," PLoS ONE **12**, e0182227 (2017).
39. R. A. Sharma, V. Gandhi, V. Chari, *et al.*, "Automatic analysis of broadcast football videos using contextual," Signal Image Video Process. **11**, 171–178 (2017).
40. F. Hohman, S. Soni, I. Stewart, *et al.*, "A viz of ice and fire: exploring entertainment video using color and dialogue," in *Proceedings of 2nd Workshop on Visualization for the Digital Humanities* (2017).
41. E. H. Ruspini, "Numerical methods for fuzzy clustering," Inf. Sci. **2**, 319–350 (1970).
42. X. Wu, "Optimal quantization by matrix searching," J. Algorithms **12**, 663–673 (1991).
43. M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar *k*-means problem is NP-Hard," in *Proceedings of 3rd International Workshop on Algorithms and Computation* (2009), pp. 274–285.
44. T. H. Cormen, C. E. Leiserson, R. L. Rivest, *et al.*, *Introduction to Algorithms*, 4th ed. (The MIT Press, 2022).
45. S. Shlien and A. Smith, "A rapid method to generate spectral theme classification of LANDSAT imagery," Remote Sens. Environ. **4**, 67–77 (1975).
46. P. Macnaughton-Smith, W. T. Williams, M. B. Dale, *et al.*, "Dissimilarity analysis: a new technique of hierarchical sub-division," Nature **202**, 1034–1035 (1964).
47. J. L. Bentley, "Multidimensional binary search trees used for associative searching," Commun. ACM **18**, 509–517 (1975).
48. J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," J. Am. Stat. Assoc. **58**, 415–434 (1963).
49. A. Buzo, R. M. Gray, A. H. Gray, Jr., *et al.*, "Speech coding based upon vector quantization," IEEE Trans. Acoust. Speech Signal Process. **28**, 562–574 (1980).
50. H. Fuchs, Z. M. Kedem, and B. F. Naylor, "On Visible surface generation by a priori tree structures," in *Proceedings of 7th Annual Conference on Computer Graphics and Interactive Techniques* (1980), pp. 124–133.
51. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," IEEE Trans. Commun. **28**, 84–95 (1980).
52. S. Lloyd, "Least squares quantization in PCM," IEEE Trans. Inf. Theory **28**, 129–137 (1982).
53. S. J. Wan, S. K. M. Wong, and P. Prusinkiewicz, "An algorithm for multidimensional data clustering," ACM Trans. Math. Softw. **14**, 153–162 (1988).
54. X. Wu and I. H. Witten, "A fast *k*-means type clustering algorithm," Tech. Rep. 85/197/10 (University of Calgary, 1985).
55. M. Orchard and C. Bouman, "Color quantization of images," IEEE Trans. Signal Process. **39**, 2677–2690 (1991).
56. X. Wu, "Statistical colour quantization for minimum distortion," in *Computer Graphics and Mathematics*, B. Falcidieno, I. Herman, and C. Pienovi, eds. (Springer, 1992), pp. 189–202.
57. M. E. Celebi, "Fast color quantization using weighted sort-means clustering," J. Opt. Society Am A **26**, 2434–2443 (2009).
58. M. E. Celebi, "Improving the performance of *k*-means for color quantization," Image Vis. Comput. **29**, 260–271 (2011).
59. S. Thompson, M. E. Celebi, and K. H. Buck, "Fast color quantization using MacQueen's *k*-means algorithm," J. Real-Time Image Process. **17**, 1609–1624 (2020).
60. A. D. Abernathy and M. E. Celebi, "The incremental online *k*-means clustering algorithm and its application to color quantization," Expert Syst. Appl. **207**, 117927 (2022).
61. H. Bounds, M. E. Celebi, and J. Maxwell, "Color quantization using an accelerated Jancey *k*-means clustering algorithm," J. Electron. Imaging **33**, 053052 (2024).
62. R. W. Floyd and L. Steinberg, "An adaptive algorithm for spatial greyscale," Proc. Soc. Inf. Disp. **17**, 75–77 (1976).
63. J. L. Nieves, L. Gomez-Robledo, Y. J. Chen, *et al.*, "Computing the relevant colors that describe the color palette of paintings," Appl. Opt. **59**, 1732–1740 (2020).
64. E. F. Harding, "The number of partitions of a set of *n* points in *k* dimensions induced by hyperplanes," Proc. Edinburgh Math. Soc. **15**, 285–289 (1967).
65. D. Aloise, A. Deshpande, P. Hansen, *et al.*, "NP-hardness of Euclidean sum-of-squares clustering," Mach. Learn. **75**, 245–248 (2009).
66. S. Dasgupta and Y. Freund, "Random projection trees for vector quantization," IEEE Trans. Inf. Theory **55**, 3229–3242 (2009).
67. O. Milvang, "An adaptive algorithm for color image quantization," in *Proceedings of 5th Scandinavian Conference on Image Analysis* (1987), Vol. **1**, pp. 43–47.
68. X. Wu, "Efficient statistical computations for optimal color quantization," in *Graphics Gems II*, J. Arvo, ed. (Academic Press, 1991), pp. 126–133.
69. M. E. Celebi, Q. Wen, and S. Hwang, "An effective real-time color quantization method based on divisive hierarchical clustering," J. Real-Time Image Process. **10**, 329–344 (2015).
70. J. Puzicha, M. Held, J. Ketterer, *et al.*, "On spatial quantization of color images," IEEE Trans. Image Process. **9**, 666–682 (2000).
71. D. Özdemir and L. Akarun, "Fuzzy algorithms for combined quantization and dithering," IEEE Trans. Image Process. **10**, 923–931 (2001).
72. H. Z. Huang, K. Xu, R. R. Martin, *et al.*, "Efficient, edge-aware, combined color quantization and dithering," IEEE Trans. Image Process. **25**, 1152–1162 (2016).
73. Z. Tirandaz, D. H. Foster, J. Romero, *et al.*, "Efficient quantization of painting images by relevant colors," Sci. Rep. **13**, 3034 (2023).
74. C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: overview and conceptual comparison," ACM Comput. Surv. **35**, 268–308 (2003).
75. M. L. Pérez-Delgado and M. A. Günen, "A comparative study of evolutionary computation and swarm-based methods applied to color quantization," Expert Syst. Appl. **231**, 120666 (2023).
76. M. L. Pérez-Delgado and J. A. R. Gallego, "A hybrid color quantization algorithm that combines the greedy orthogonal bi-partitioning method with artificial ants," IEEE Access **7**, 128714–128734 (2019).
77. M. L. Pérez-Delgado and J. A. R. Gallego, "A two-stage method to improve the quality of quantized images," J. Real-Time Image Process. **17**, 581–605 (2020).
78. M. E. Celebi and K. Aydin, eds., *Unsupervised Learning Algorithms* (Springer, 2016).
79. M. L. Pérez-Delgado and M. E. Celebi, "A comparative study of color quantization methods using various image quality assessment indices," Multimedia Syst. **30**, 40 (2024).
80. I. Yoo, X. Luo, Y. Wang, *et al.*, "GIFnets: differentiable GIF encoding framework," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (2020), pp. 14473–14482.
81. Y. Hou, L. Zheng, and S. Gould, "Learning to structure an image with few colors," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (2020), pp. 10116–10125.
82. S. Su, L. Gu, Y. Yang, *et al.*, "Name your colour for the task: artificially discover colour naming via colour quantisation transformer,"

in *Proceedings of IEEE/CVF International Conference on Computer Vision* (2023), pp. 12021–12031.

83. J. H. Park, S. H. Kim, J. C. Lee, *et al.*, "Scalable color quantization for task-centric image compression," ACM Trans. Multimedia Comput. Commun. Appl. **19**, 82 (2023).
84. B. Yuan, Z. Wang, M. Baktashmotlagh, *et al.*, "Color-oriented redundancy reduction in dataset distillation," in *Proceedings of 38th Conference on Neural Information Processing Systems* (2024), pp. 53237–53260.
85. J. Li, L. Fu, S. Yang, *et al.*, "HiEI: A universal framework for generating high-quality emerging images from natural images," in *Proceedings of 18th European Conference on Computer Vision* (2024), pp. 129–145.
86. Y. Hou, S. Gould, and L. Zheng, "Scalable deep color quantization: a cluster imitation approach," IEEE Trans. Image Process. **33**, 5273–5283 (2024).
87. J. Li, L. Fu, S. Yang, *et al.*, "MI-CAPTCHA: enhance the security of CAPTCHA using mooney images," in *Proceedings of 39th AAAI Conference on Artificial Intelligence* (2025), Vol. **39**, pp. 1383–1391.
88. Y. Wang, G. Liu, Z. Zhu, *et al.*, "VivID: A visually improved GIF encoding network design," IEEE Trans. Circuits Syst. Video Technol. **35**, 6101–6113 (2025).
89. S. Redfield and J. G. Harris, "The role of extreme color quantization in object recognition," in *Proceedings of 1st International Conference on Color in Graphics and Image Processing* (2000), pp. 225–230.