# Video Recoloring via Spatial-Temporal Geometric Palettes

ZHENG-JUN DU, BNRist, Department of CS&T, Tsinghua University, China and Qinghai University, China
KAI-XIANG LEI, BNRist, Department of CS&T, Tsinghua University, China
KUN XU*, BNRist, Department of CS&T, Tsinghua University, China
JIANCHAO TAN, Kwai Inc., USA
YOTAM GINGOLD, George Mason University, USA

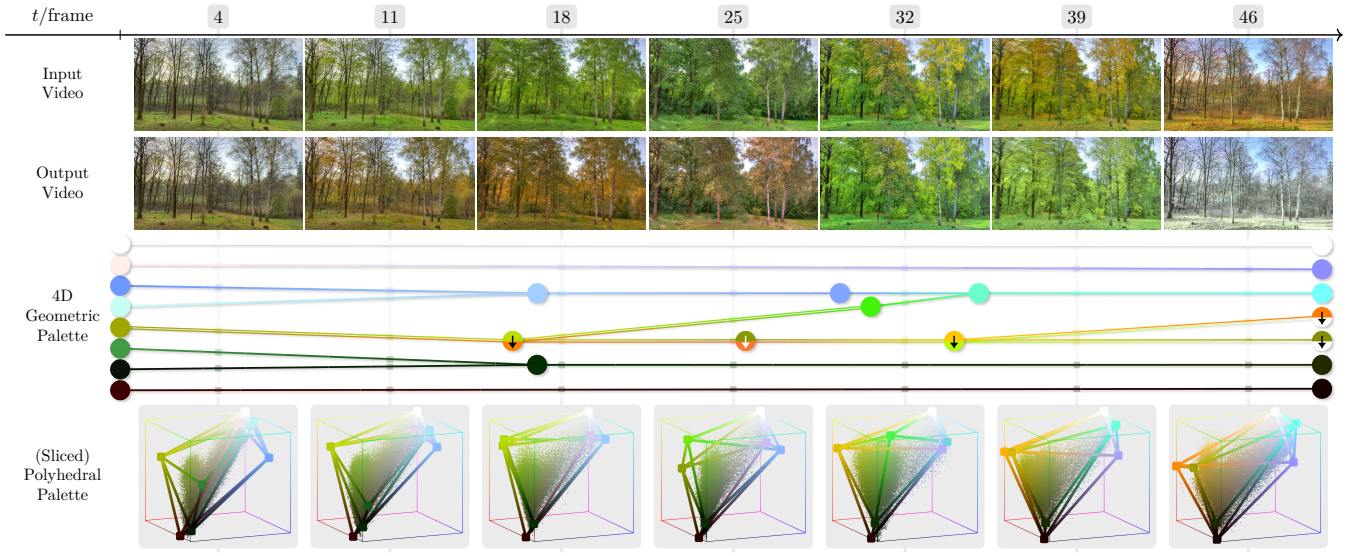



Fig. 1. Geometric palette extraction and video recoloring using our method. Top to bottom: frames of the input video, corresponding frames of the recolored video, a timeline view of the geometric palette, and RGB-space visualizations of the sliced polyhedral palettes and image colors at corresponding frames. The input video is a year-long time-lapse. Our extracted geometric palette captures object-level time-varying color changes rather well, e.g., the color of leaves changes from green to orange. We successfully change the video from spring-summer-autumn to spring-autumn-summer-winter order. In the geometric palette, each node denotes a palette color. The color modification of a node is displayed by a vertical arrow, where the upper/lower half-disk shows the color before/after the edit, respectively.©Eirik Solheim.

Color correction and color grading are important steps in film production. Recent palette-based approaches to image recoloring have shown that a small set of representative colors provide an intuitive set of handles for color adjustment. However, a single, static palette cannot represent the time-varying colors in a video. We introduce a spatial-temporal geometry-based approach to video recoloring. Specifically, its core is a 4D skew polytope with a few vertices that approximately encloses the video pixels in color and time, which implicitly defines time-varying palettes through slicing of the 4D skew polytope at specific time values. Our geometric palette is compact, descriptive, and provides a correspondence between colors throughout the video, including topological changes when colors merge or split. Experiments show that our method produces natural, artifact-free recoloring.

CCS Concepts: • **Computing methodologies** → **Image manipulation**; **Image processing**.

Additional Key Words and Phrases: Palette, Color, Painting, Image, Video, Spatial-temporal, Recoloring, Layer, RGB

**ACM Reference Format:**
Zheng-Jun Du, Kai-Xiang Lei, Kun Xu, Jianchao Tan, and Yotam Gingold. 2021. Video Recoloring via Spatial-Temporal Geometric Palettes. *ACM Trans. Graph.* 40, 4, Article 150 (August 2021), 16 pages. https://doi.org/10.1145/3450626.3459675

*Kun Xu is the corresponding author.

Authors' addresses: Zheng-Jun Du, BNRist, Department of CS&T, Tsinghua University, Beijing, China, Qinghai University, Xining, Qinghai, China, duzj19@mails.tsinghua.edu.cn; Kai-Xiang Lei, BNRist, Department of CS&T, Tsinghua University, Beijing, China, leikx18@mails.tsinghua.edu.cn; Kun Xu, BNRist, Department of CS&T, Tsinghua University, Beijing, China, xukun@tsinghua.edu.cn; Jianchao Tan, Kwai Inc., Bellevue, Washington, USA, tanjianchaoustc@gmail.com; Yotam Gingold, George Mason University, Fairfax, Virginia, USA, ygingold@gmu.edu.


0730-0301/2021/8-ART150 $15.00
https://doi.org/10.1145/3450626.3459675

## 1 INTRODUCTION

Palette-based image editing is a recent paradigm for adjusting the colorful appearance of an image [Chang et al. 2015; Mellado et al. 2017; Tan et al. 2015, 2018a; Wang et al. 2019; Zhang et al. 2017]. These approaches decompose an image into a palette and per-pixel mixing parameters. The palette is a small set of salient or representative colors. This provides users with an intuitive set of handles with which to edit the image. Users manipulate the palette colors by, for example, darkening a green color or changing a red color to orange. Each pixel's color is then updated based on the edited palette, producing a recolored image. The palette provides a more intuitive editing basis than the RGB channels of the input image. Palette-based image editing provides more control than approaches based on transferring the color from one image to another [He et al. 2019]; or global functions such as exposure adjustment, hue rotation, or a look-up table for the entire color space. A common formulation, and one we adopt as well, is to assign each pixel additive mixing weights. Then each pixel's color can be computed as $\sum_i w_i \mathrm{v}_i$, where $w_i$ is the weight for palette color $\mathrm{v}_i$. This is a kind of dimensionality or basis enlargement. Each pixel's RGB values are unmixed into more intuitive but higher dimensional mixing weights for the palette colors. In other fields, this is known as generalized barycentric coordinates (geometry processing), endmember extraction (hyperspectral imaging), or convex non-negative matrix factorization.

Color adjustment is also an important step in film production. Professional colorists divide the task into two stages, color correction and color grading. During color correction, colorists align the colors of different shots to provide a coherent color scheme and intensity throughout the film despite recording with different equipment or illumination or scenery. During color grading, colorists alter the colors to achieve a particular artistic look or feeling. Existing tools for video color editing are also based on transfer by example [Duchêne et al. 2017] or global operations [Adobe 2020].

The key **challenge** in applying palette-based editing to video color adjustment is that the colors change over time. For artistic control, color adjustments must also be able to change over time. A single, static palette cannot represent the time-varying colors in a video. Computing a separate palette per-frame would create an unusably large number of handles for the user to adjust.

*Contributions.* We extend the palette-based recoloring framework to video scenarios. We do this by finding a compact yet descriptive geometric palette for the video. This palette can be directly edited to recolor the video. Our geometric palette provides a correspondence between palette colors throughout the video, including when colors merge or split. Users can add new key frames to create complex, time-varying color changes. Our algorithm is based on the geometry of the video's colors in 4D (color and time). To obtain a simple, corresponded palette between key frames, we use a 4D skew polytope (i.e., a generalized polyhedron whose faces are allowed to be non-planar) in RGBT space to enclose the video colors over time.

## 2 RELATED WORK

### 2.1 Palette Extraction

A palette or color theme is a succinct representation of an image or video. Several methods investigated human perception of palettes. These works fit predictive models of human palette preference and generate perceptual palettes from input images [Cao et al. 2017; Feng et al. 2018; Lin and Hanrahan 2013; O'Donovan et al. 2011]. Several other approaches [Afifi 2019; Chang et al. 2015; Nguyen et al. 2017; Zhang et al. 2017] employ k-means clustering of pixel colors to extract palette colors. These palettes capture the dominant colors in an image.

Recently, several works extract palettes from images based on simplified (or approximated) convex hulls in RGB color space [Baptiste et al. 2019; Grogan and Smolic 2020; Kim and Choi 2020; Tan et al. 2018a, 2016; Wang et al. 2019]. The extracted convex hull has a simple geometric shape and its vertices intuitively represent palette colors. Pixel colors can be naturally represented as linear combinations of palette colors. Its linear nature makes the recoloring process intuitive, efficient, and results in better recoloring quality than clustering based approaches. However, convex-hull-based palettes may miss important colors that lie within the convex hull. Our method extends convex-hull palettes to videos.

In addition to the above families of approaches, several alternative approaches to palettes have been proposed. Duque-Arias et al. [2019] represent image colors using structured color lines. This is a more representative format, but not as compact as a palette and so less convenient for editing. DiVerdi et al. [2019] generated "playful palettes" that can be interpolated to obtain new colors according to a spatial arrangement of palette colors and their RGB values. Jeong et al. [2019] built a hierarchical palette representation for image. Shugrina et al. [2020] explored nonlinear interpolation of colors triads as a palette representation, which proved to be a simple yet powerful approximation. These approaches are not amenable to our geometric extension to videos.

### 2.2 Palette-Based Image Decomposition

After palette extraction, the image could be further decomposed into layers, where each layer is a spatially varying influence map of a palette color. Several methods proposed to use "over" compositing layers [Richardt et al. 2014; Tan et al. 2015, 2016]. The layers are order-dependent, translucent "coats of paint" of the palette colors. When blended one on top of the other, they reconstruct the input image. Koyama and Goto [2018] extended this to advanced blending modes. One particular blending mode, additive mixing weights, is notable for its simplicity and order-independence [Aksoy et al. 2017; Lin et al. 2017; Tan et al. 2018a; Wang et al. 2019; Zhang et al. 2021, 2017]. Our method uses additive mixing weights as well.

Several works based their decompositions on physically-inspired nonlinear color mixing models, namely Kubelka-Munk [Abed 2014; Aharoni-Mack et al. 2017; Tan et al. 2019, 2015]. Several methods consider illumination-aware palettes or layers [Cheng et al. 2020; Duchêne et al. 2017; Meka et al. 2019]. They also explored recoloring while changing the illumination of the input image for natural recoloring effects. In our setting, we use temporally coherent additive mixing weights.

### 2.3 Color Editing

Color editing, generally stated, is well-studied beyond palette-based approaches. Many approaches have explored user guidance by drawing colored scribbles or approximately-drawn spatial markings; the algorithms propagate these hints to other areas of the image or video [An and Pellacini 2008; Chen et al. 2012; Levin et al. 2004; Lischinski et al. 2006; Pellacini and Lawrence 2007; Xu et al. 2009]. Scribbles impose a higher cost on users than palette-based edits, since users must provide spatial and color guidance. Color transfer is another form of color editing, in which one image adopts the colors from another reference image. Color transfer has been explored by many classical methods [Bonneel et al. 2013; Greenfield and House 2003, 2005; Reinhard et al. 2001; Tan et al. 2018b; Welsh et al. 2002; Yang and Peng 2008] and recently with deep learning techniques [He et al. 2019; Huang et al. 2020; Luan et al. 2017]. These approaches consider color correspondence and semantic feature correspondence between the reference image and recolored image. While powerful, these methods provide very few editing controls to users beyond the choice of reference image.

Palette-based recoloring methods provide a trade-off between user control and recoloring. Our method follows the palette-based color editing framework, decomposing videos into 4D geometric palettes and efficient additive mixing weights.

### 2.4 Color Quantization

In a quantized image, i.e., a GIF-formatted image, instead of storing a full RGB color, each pixel only stores the index to a color lookup table. The color lookup table, which is also referred to as a palette, ususally contains 64–256 distinct colors. Color quantization is the process of converting an image to a quantized representation, which reduces the number of distinct colors while preserving visual fidelity. Various color quantization methods [Balasubramanian and Allebach 1991; Celebi 2011; Gervautz and Purgathofer 1988; Orchard et al. 1991; Wen and Celebi 2011; Wu 1992] have been proposed to obtain the color palette (color lookup table) from an image. Some other works extend color quantization to videos [Cheung and Chan 2003; Kusswurm 1998; Roytman and Gotsman 1995], or simultaneously achieve color quantization and dithering [Huang et al. 2016]. See Ozturk et al. [2014] for a review of color quantization. The palette used in color quantization is different from the palette used in our method. First, the purpose of quantization is to reduce image storage rather than recoloring. Second, quantization palettes are usually much larger.

## 3 BACKGROUND AND PRELIMINARIES

Our approach builds on convex-hull-based methods for palette extraction and additive mixing weight computation for images [Tan et al. 2018a, 2016; Wang et al. 2019].

*Additive Mixing Weights.* Given an image $\mathrm{I}$ and a palette $\mathrm{V}$ of colors, the color of each pixel $\mathrm{p}$ can be expressed as a weighted sum of palette colors:

$$\mathrm{p} = \sum_{i}^{|\mathrm{V}|} w_{i,\mathrm{p}} \mathrm{v}_i, \forall \mathrm{p} \in \mathrm{I}, \tag{1}$$

$\mathrm{v}_i \in \mathrm{V}$ denotes a palette color, and $w_{i,\mathrm{p}}$ denotes the *additive mixing weight* of pixel $\mathrm{p}$ corresponding to the palette color $\mathrm{v}_i$. If the convex hull of the palette colors encloses the pixel color in color-space, then mixing weights can be convex: $\sum_i w_i = 1, 0 \le w_i \le 1$. We use RGB space throughout this work.

*Convex-Hull-Based Palettes.* Tan et al. [2018a, 2016] proposed to compute the 3D convex hull of all pixel colors in RGB space, and then to iteratively simplify the convex hull to obtain a color palette $\mathrm{V}$. The iterative simplification stops when further simplification would exceed a maximum allowable reconstruction error. Reconstruction errors occur whenever pixel colors lie outside the simplified hull, whose vertices are constrained to lie within the RGB color gamut $[0, 1]^3$. There are three desirable properties for a convex-hull-based palette: a small number of vertices (colors), high reconstruction accuracy, and a *compact* palette. Too many vertices are tedious for the user to manipulate. A compact palette is more representative of image colors (i.e. closer). This increases maximum weight values, which increases the directness of control offered by the palette. Wang et al. [2019] improved the compactness of the simplified convex hull via a vertex refinement step. Note that the polyhedrons generated by Tan et al. [2018a, 2016] and Wang et al. [2019] are simplified and approximate convex hulls, not accurate ones. For simplicity of description, we will continue to use the term *convex hull* to denote these approximations.

We measure the reconstruction error or loss as the average distance of all pixels to the convex hull of the palette:

$$R(\mathrm{V}, \mathrm{I}) = \frac{1}{|\mathrm{I}|} \sum_{\mathrm{p} \in \mathrm{I}} \|\mathrm{p} - \mathrm{proj}(\mathrm{p})\|, \tag{2}$$

where $\mathrm{proj}(\mathrm{p})$ is the projection of $\mathrm{p}$ into the convex hull of $V$. ($\mathrm{proj}(\mathrm{p}) = \mathrm{p}$ when $\mathrm{p}$ is already inside.) To avoid enumerating all pixels, one can compute a faster, approximate reconstruction error by randomly sampling a small subset of pixels.

We use Wang et al. [2019]'s definition of compactness loss:

$$C(\mathrm{V}, \mathrm{I}) = \frac{1}{|\mathrm{V}|} \sum_{\mathrm{v} \in \mathrm{V}} \|\mathrm{v} - \mathrm{v}_N\|, \tag{3}$$

where $\mathrm{v}_N$ is the average of the k-nearest image pixels to $\mathrm{v}$ in color-space. We use $k = 50$. Reconstruction error and compactness are in tension with each other. We define the overall loss $L$ of a single frame as their weighted sum:

$$L(\mathrm{V}, \mathrm{I}) = \lambda R(\mathrm{V}, \mathrm{I}) + C(\mathrm{V}, \mathrm{I}), \tag{4}$$

where $\lambda$ control the relative contribution of the two terms. We set $\lambda = 50$ for all our examples.

## 4 4D GEOMETRIC PALETTE

This section defines our 4D time-varying geometric palettes, which extend convex-hull-based image palettes. Section 5 explains how we extract the geometric palette from an input video.

There are several desiderata when extending palette-based editing from images to videos. (1) The user should be able to edit time-varying effects. For example, a time lapse video may depict foliage changing from summer-like to autumn-like—or the user may wish to add this effect. (2) The palette and video should remain temporally coherent, so that edits don't introduce undesired abrupt

changes. (3) The palette should remain easy to use, which means it should have a small number of representative colors that accurately reconstruct the video.

In defining our 4D geometric palettes, we extend convex-hull-based image editing approaches [Tan et al. 2018a, 2016; Wang et al. 2019]. One naive approach would be to treat the entire video as a single, volumetric image and extract a single, global convex-hull palette shared by the entire video [Tan et al. 2018b]. The time-varying nature of the input video must be encoded entirely as time-varying per-pixel weights. In this naive approach, it is impossible to edit time-varying color changes in the input, since the extracted palette is static by construction. Moreover, the compactness of a shared, global palette will be low, since it must enclose all colors across all frames simultaneously.

Another naive extension would be to compute an independent convex-hull-based decomposition for each frame of the video. However, this approach creates far too many vertices for users to edit and has no temporal coherence. Finally, one could compute the convex hull of pixels in 4D RGBT space (3D color plus time). Similarly, it also prone to suffer from poor compactness. We compare to these and other techniques in Section 6.2.

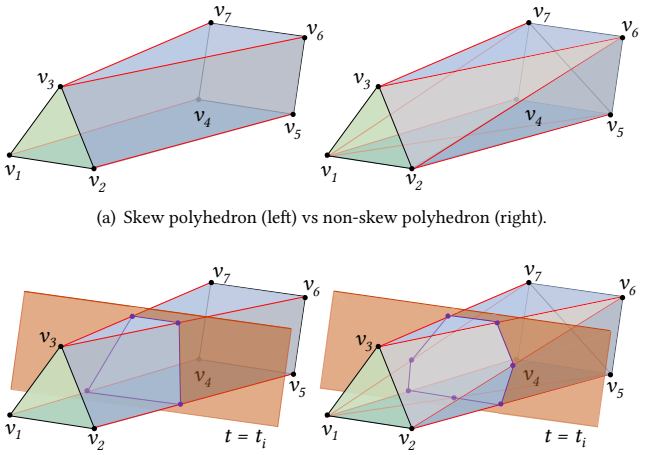
Our goal is to generate a geometric palette for a video, i.e., a 3D polyhedron per video frame that approximately encloses the frame's colors. We call this per-frame polyhedron the frame's *polyhedral palette*. Polyhedral palettes have all the same desiderate as a simplified convex hull (Section 3). In addition, polyhedral palettes in different frames should be related according to three additional desirable properties. First, polyhedral palettes should be temporally coherent between adjacent frames. Sudden changes should be avoided. Second, topological changes should be allowed, e.g., one frame can have a polyhedral palette with 4 vertices and the next frame can have one with 5 vertices. This corresponds to colors appearing or disappearing in time. Third, the overall geometric palette should expose to the user as few parameters as possible. Exposing all parameters (i.e., vertices and edges) of all frames' polyhedral palettes would be tedious and difficult to control. Our interface should provide a way to conveniently control all or multiple polyhedral palettes simultaneously.

To achieve these goals, we base our 4D geometric palettes on a geometric structure called a *4D skew polytope*. We project all video pixels into 4D RGBT space (3D RGB color plus the time dimension). The 4D skew polytope is constructed to enclose all (or most) video pixels in 4D RGBT space. The 3D polyhedral palette for a specific video frame is *implicitly defined* as the slice of the 4D skew polytope at a specific time value. As a result, our representation has few parameters. It only needs to store the vertices and edges of the 4D skew polytope. Video recoloring is achieved by editing the RGB values of each vertex of the 4D skew polytope or by introducing new colors along its inter-frame edges.

## 4.1 4D Skew Polytope

We define our 4D skew polytope, its slicing operator, and how to compute mixing weights in detail.

*4.1.1 Definition.* A 4D polytope (skew or non-skew) in RGBT space can be defined as $P = (V, E, F, \Gamma)$, where $V$, $E$, $F$, and $\Gamma$ are the set of vertices (palette colors), edges, faces, and 3-cells of the 4D polytope, respectively. Each vertex $v = (r, g, b, t) \in V$ is a 4D point in RGBT space. For simplicity, we normalize time values within $[0, 1]$, e.g., the first video frame and last video frame correspond to $t = 0$ and $t = 1$, respectively. (RGB values already lie within $[0, 1]$.) There are two types of edges: *intra-frame edges* and *inter-frame edges*. An intra-frame edge connects two vertices in the same video frame, i.e., having the same $t$ value. An inter-frame edge connects two vertices in different video frames.




(a) Skew polyhedron (left) vs non-skew polyhedron (right).

(b) Sliced skew polytope (left) versus sliced non-skew polytope (right).

Fig. 2. A lower-dimensional illustration of our 4D geometric palette. Here, the geometric palette is a 3D skew polytope for a video using a 2D color space. The time axis runs orthogonally to face $v_1v_2v_3$. (a) A skew polytope (left) can have non-planar faces, unlike non-skew polytope (right). (b) Slicing a skew polytope results in a simpler polygon (left) than slicing a non-skew polytope (right).

*4.1.2 Slicing.* A polytope of dimension $(n-1)$ can be obtained by intersecting or *slicing* a polytope of dimension $n$ with a hyper-plane. For example, one obtains a 2D polygon by slicing a 3D tetrahedron. In this way, we naturally obtain the 3D polyhedral palette for the $i$−th video frame by slicing the 4D polytope $P$ with a hyper-plane $t = t_i$:

$$\mathrm{V}_i = S(P, t_i), \tag{5}$$

where $S(\cdot)$ indicates the slicing operator, $t_i$ denotes the time of the $i$−th video frame, and $\mathrm{V}_i$ denotes the resulting polyhedral palette.

The 4D polytope together with the slicing operator allows us to obtain time varying 3D polyhedral palettes. The generated polyhedral palettes vary continuously along the time dimension. Topological changes may occur. An illustration of slicing is given in Figure 2 (b).

*4.1.3 Skew vs non-skew.* In mathematics, a skew polyhedron is a generalized polyhedron whose faces are allowed to be non-planar [Wikipedia 2020]. In contrast, the faces of a non-skew polyhedron must be planar. Since we have 4D data, we use the higher dimensional generalization, skew polytopes. We use non-planar faces for two reasons. First, a skew polytope has fewer edges and hence its representation is simpler. Second, slicing a non-skew 4D polytope

produces a 3D polyhedron with unnecessarily more vertices. This is illustrated in 3D in Figure 2, where slicing a non-skew polyhedron produces a polygon with 7 vertices, whereas slicing a skew polyhedron produces a polygon with 4. Fewer vertices are easier to visualize and control.

### 4.2 Computing Mixing Weights

At any time $t$, we can obtain the sliced 3D polyhedral palette $\mathrm{V}_t$ (Equation 5). We can compute the mixing weights $W_\mathrm{p}$ for a color $\mathrm{p} = (r, g, b)$ with respect to $\mathrm{V}_t$ using any generalized barycentric coordinate scheme. In our experiments, we use Mean Value Coordinates (MVC) [Floater et al. 2005; Ju et al. 2005; Wang et al. 2019]. This allows us to express $\mathrm{p} = W_\mathrm{p}\mathrm{V}_t$, where $W_\mathrm{p}$ is a row-vector and $\mathrm{V}_t$ is the matrix whose rows are the palette colors.

In turn, the vertices $\mathrm{V}_t$ of the polyhedral palette lie along edges of the 4D geometric palette. They can be expressed as a linear combination of the edge's endpoints:

$$\mathrm{V}_t = W_t\mathrm{V}, \tag{6}$$

where $\mathrm{V}$ is the matrix whose rows are the 4D geometric palette's vertices.

Since both steps are linear, they can be combined through matrix multiplication. In this way, any RGBT point $\mathrm{p} = (r, g, b, t)$ can be directly expressed with additive mixing weights in terms of the 4D geometric palette:

$$\mathrm{p} = W_{RGBT}\mathrm{V}, \tag{7}$$

where $W_{RGBT} = W_\mathrm{p}W_t$.

## 5 4D GEOMETRIC PALETTE EXTRACTION

In this section, we explain how to extract a 4D geometric palette from a given video.

### 5.1 Overview

A straightforward approach to generate a 4D skew polytope from a video is to use the (simplified) 4D convex hull enclosing all video pixels in RGBT space. However, the resulting 4D convex hull often encloses a lot of empty space along the time dimension and not satisfy our compactness property. A comparison to this straightforward approach is given in Section 6.2.

Instead, we propose a progressive approach to extract the 4D skew polytope. Our approach mainly consists of 4 steps:

- **Initialization.** We generate an initial 4D skew polytope by gluing together the approximated 3D RGB convex hulls of adjacent video frames. This initial 4D skew polytope is overly complex (i.e., has a large number of vertices and edges). It is a starting point for later topological refinement and mesh simplification. See Section 5.2.
- **Block Merging.** In the initialization step, each video frame generated its polyhedral palette independently. As a result, it is likely that the polyhedral palettes of adjacent frames have different topology, e.g., one has 5 vertices but the other has 6 vertices. Topological changes between frames are often unnecessary and will lead to difficulty during later mesh simplification. Hence, we introduce a *block merging* algorithm that greatly reduces the number of topological changes between frames. See Section 5.3.
- **Vertex Removal.** To reduce the complexity of the 4D skew polytope, we simplify the 4D skew polytope by iteratively removing vertices until the overall loss reaches a predefined threshold. See Section 5.4.
- **Vertex Refinement**. After the above simplification step, we perform vertex refinement to further reduce the overall loss and increase temporal coherence. See Section 5.5.

The full pipeline is illustrated in Figure 3.

*5.1.1 Loss function.* We use the following loss throughout our approach, described in detail below. Given a video sequence $\mathrm{I}$ with $N$ frames, we denote the extracted 4D skew polytope as $P$. Following previous work on image palettes [Wang et al. 2019], we use Equations 2, 3, and 4 to define the reconstruction loss, compactness loss, and overall loss (i.e., combining reconstruction and compactness) of a video frame. The overall loss for the entire video is defined as the average loss of all frames:

$$L_v(P, \mathrm{I}) = \frac{1}{N}\sum\nolimits_{i=1}^{N} \theta_i L(S(P, t_i), \mathrm{I}_i), \tag{8}$$

where $i$ enumerates over all frames, $S(P, t_i)$ (Equation 5) denotes the sliced 3D polyhedral palette of $P$ at time $t_i$ (or frame $i$), $\mathrm{I}_i$ and $\theta_i$ denote the $i$-th video frame and its corresponding weight, respectively.

### 5.2 Generating the Initial 4D Skew Polytope

*5.2.1 Generating a polyhedral palette for each frame.* For each video frame, we use the method in Tan et al. [2016] to extract a simplified 3D RGB convex hull. We refine the vertex positions to reduce the overall loss (Equation 4) using the method in Wang et al. [2019]. This results in an approximated RGB convex hull (i.e., RGB polyhedron) which is usually a bit smaller than the original convex hull with lower compactness loss. This is the frame's polyhedral palette.

*5.2.2 Gluing adjacent frames.* For each pair of adjacent frames, we need to connect the vertices of the two polyhedral palettes to form a two-frame 4D skew polytope. Formally, denote the vertex sets of the two frames' polyhedral palettes as $\mathrm{V} = \{v_i\}$ and $\mathrm{U} = \{u_j\}$. Our goal is to generate a set of inter-frame edges $\mathrm{E} = e_k$, where each $e_k$ connects a vertex in $\mathrm{V}$ and another vertex in $\mathrm{U}$.

We introduce 4 conditions to ensure that the resulting two-frame 4D skew polytope is not only valid (a closed polytope), but also has a clear vertex correspondence. These conditions are illustrated in Figure 4. The first two conditions ensure that each vertex has a correspondence, and that the correspondence does not create a transient color that appears and disappears between two frames.

- **Inter-degree Rule 1.** We define the *inter-degree* of a vertex as the number of inter-frame edges it has. The degree of every vertex in $\mathrm{V}$ and in $\mathrm{U}$ should $\geq 1$. Zero inter-degree is not allowed.
- **Inter-degree Rule 2.** For each inter-frame edge, i.e., connecting a vertex $v$ and a vertex $u$, At most one vertex in them is allow to have inter-degree $\geq 2$.

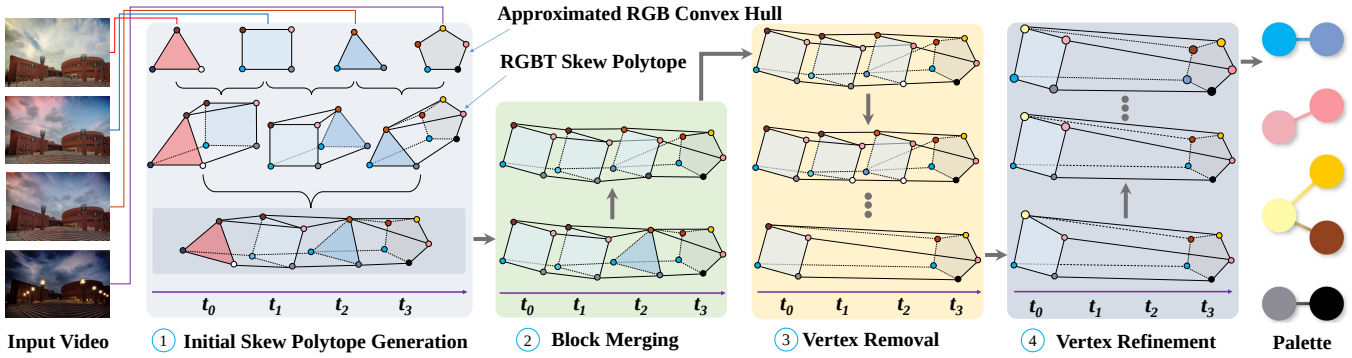



Fig. 3. The pipeline of our geometric palette extraction approach. Our method takes a video as input and automatically extracts the corresponding 4D time-varying geometric palette. The pipeline mainly contains four steps. 1) Initial skew polytope generation. We first extract the approximated RGB convex hull for each video frame, and then build the RGBT skew polytope on adjacent video frame pairs, finally, we glue all two-frame skew polytopes together. 2) Block merging. We divide the video frames into different blocks based on the topology of their polyhedral palettes, and perform block merging that enforce the similar adjacent blocks share the same topology. 3) Vertex removal. We iteratively remove vertices in order to simplify the 4D skew polytope. 4) Vertex refinement. We locally refine vertex positions to further reduce the overall loss and increase temporal coherence. The resulted 4D skew polytope corresponds to the generated geometric palette. Note that our 4D skew polytope is illustrated in a lower dimension (3D) for better understanding.

The second two conditions ensure that an intra-frame edge in one frame maps to another intra-frame edge (or else merges to a vertex) in the other frame:

- **Topological Rule 1.** For each vertex $v$ that has multiple inter-frame edges, i.e., connects to two different vertices $u_1$ and $u_2$, then $u_1$ and $u_2$ must be already connected through an intra-frame edge. (same for $u$ connecting to $v_1$ and $v_2$).
- **Topological Rule 2.** For each intra-frame edge with two end vertices $v_1$ and $v_2$, suppose $v_1$ connects to $u_1$ and $v_2$ connects to $u_2$, then, we must satisfy that $u_1 = u_2$ (they are the same vertex), or $u_1$ and $u_2$ must be already connected through an intra-frame edge.

The above requirements help make topological correspondences or changes between polyhedral palettes more meaningful. There will be three allowable kinds of vertex correspondence between polyhedral palettes: 1) One-to-One: a vertex $v$ in V connects to only one vertex $u$ in U (and vice versa). 2) Many-to-One: multiple vertices in V connect to a single vertex $u$ in U; we refer to $u$ as a *merging vertex*. 3) One-to-Many: a vertex $v$ in V connects to multiple vertices in U; we refer to $v$ as a *splitting vertex*. Note that many-to-many vertex correspondences are not allowed.

In addition to those requirements, we also desire that the 4D skew polytope is smooth in the time dimension. For example, if frames 1 and 2 both have a red and green vertex, connecting red to red and green to green is obviously better than connecting red to green and green to red. Formally, we seek a set of inter-frame edges E that minimizes the following equation subject to the above conditions: $\sum_{(u,v)\in E} \|u_{RGB} - v_{RGB}\|$. We compute a two-frame 4D RGBT-space convex hull using all vertices in $V \bigcup U$ and restrict the search space to the set of inter-frame edges in the 4D convex hull. The optimal set of inter-frame edges is found through an exhaustive search.

*5.2.3 Glue all frames.* Finally, we glue all two-frame 4D skew polytopes created from all frame pairs together. This forms the initial all-frame 4D skew polytope. Note that the initial 4D skew polytope is overly complex. It contains too many vertices and edges and has complicated topology. To address this, we perform topological refinement and mesh simplification, described below.

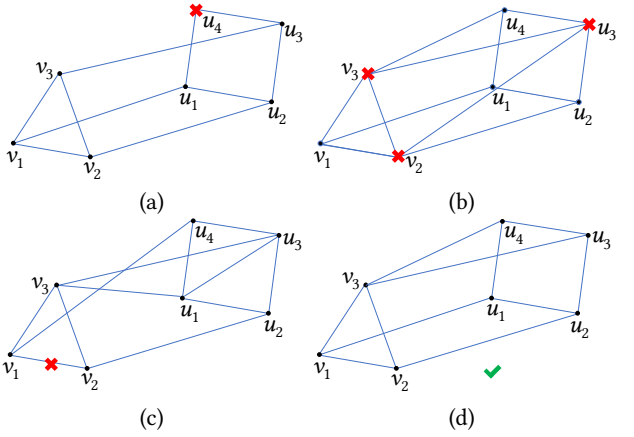



Fig. 4. The inter-degree rules and topological rules. (a) $u_4$ is an isolated vertex with degree = 0, which violates inter-degree rule 1; (b) both endpoints of inter-frame edges $(v_2, u_3)$ and $(v_3, u_3)$ with inter-degree $\geq 2$, which violates inter-degree rule 2; (c) the intra-frame edge $(v_1, v_2)$ has no corresponding vertex and edge in other frames, which violates topological rule 2; (d) All inter-frame edge rules and topology rules are met, and $v_3$ is a splitting vertex.

## 5.3 Block Merging

The initial 4D skew polytope is overly complex, containing hundreds of vertices and edges. It needs to be further simplified to be used as a 4D geometric palette. As a mesh, it may seem straightforward to simplify it via a "vertex removing" algorithm that iteratively removes vertices. However, directly simplifying the initial 4D

skew polytope leads to frequent topological changes in the sliced polyhedral palettes.

We call a pair of two adjacent frames (a *frame pair*) topologically *consistent* if (1) their polyhedral palettes have the same topology, i.e., the same number of vertices and the same intra-frame connectivity; and (2) their inter-frame edges (Section 5.2.2) are all one-to-one. Otherwise, the frame pair is called topologically *inconsistent*. Many initial frame pairs are inconsistent, because the polyhedral palette for each frame is generated independently and temporal coherence is not considered in the process. Even frame pairs with small color or content changes may be inconsistent. To address this problem, we propose an algorithm to create blocks of contiguous, topologically consistent frames.

*5.3.1 Blocks.* We define a block as a sequence of contiguous frames where each pair of adjacent frames is consistent, meaning all frames have identical topology and all inter-frame edges are one-to-one. A block can contain a single frame (when it is inconsistent with both of its adjacent frames) or multiple frames. The entire video (or the 4D skew polytope) can be viewed as a sequence of blocks $B = \{b_i\}$, which we call a *block configuration*.

Two adjacent blocks can be merged. The *block merge* operation merges a block $b_2$ into an adjacent block $b_1$, producing a merged block $b$ which contains all frames of $b_1$ and $b_2$. The frames in $b_2$ discard their polyhedral palettes and adopt new ones consistent with those in $b_1$. Without loss of generality, assume $b_1$ comes before $b_2$. Denote the last frame of block $b_1$ as $s$ and its polyhedral palette $\mathrm{V}_s$. Block $b_2$ contains frames $s+1, \ldots, s+n$, for which we need to compute new polyhedral palettes $\mathrm{V}_{s+1}, \ldots, \mathrm{V}_{s+n}$. As shown in Figure 5, the new polyhedral palettes are defined iteratively as the vertex refinement of the previous frame's:

$$\mathrm{V}_k = \mathrm{VertexRefine}(\mathrm{V}_{k-1}, \mathrm{I}_k), \quad k = s+1, \ldots, s+n. \tag{9}$$

VertexRefine iteratively optimizes [Wang et al. 2019] the vertices of $\mathrm{V}_{k-1}$ to better match frame $\mathrm{I}_k$ according to the overall loss (Equation 4). The topology of $b_2$'s new polyhedral palettes is taken directly from $b_1$. The inter-frame edges inside the merged block $b$ are regenerated to simply be one-to-one. Note that block merging is not commutative. Merging $b_2$ into $b_1$ is different from merging $b_1$ into $b_2$.

*5.3.2 Evaluating a block configuration.* We wish to reduce the number of blocks of a block configuration while maintaining the reconstruction accuracy and compactness of the 4D skew polyhedron. Hence, we use the overall loss in Equation 8 to measure the quality of a block configuration, where the frame weight $\theta_i$ is defined as:

$$\theta_i = \left(1 + \alpha \frac{n_j}{N}\right)\left(1 + \beta m_j\right), \tag{10}$$

where block $b_j$ contains frame $i$, $n_j$ denotes the number of frames in $b_j$, $N$ denotes the total number of frames, and $m_j$ denotes the vertex number of the polyhedral palette in each frame of $b_j$. The term $\left(1 + \alpha n_j/N\right)$ penalizes large blocks and hence encourages small blocks to be merged first. The term $(1 + \beta m_j)$ penalizes larger polyhedral palettes (i.e., with large numbers of vertices) and encourages the retention of blocks with smaller polyhedral palettes. $\alpha$ and $\beta$ are parameters controlling the strength of the two terms, which we empirically set as $\alpha = 10$ and $\beta = 1$.

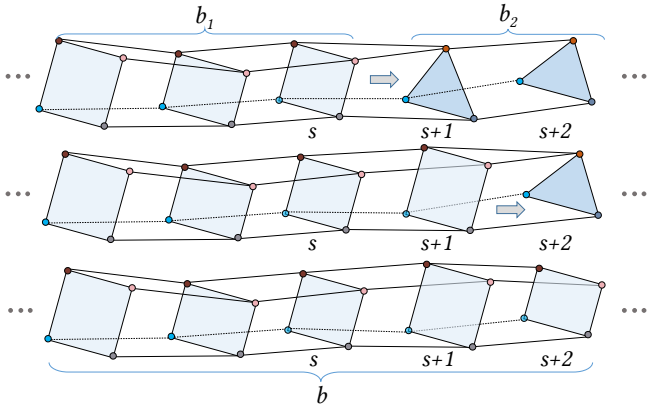



Fig. 5. Block Merging. Two adjacent blocks $b_1$ and $b_2$ (top row) are merged to a single block $b$ (bottom row), where the polyhedral palettes in block $b_2$ discard their own topology and use the topology in block $b_1$ instead.

*5.3.3 Block merging algorithm.* Our algorithm for topological refinement of the 4D Skew Polytope iteratively merges blocks until the block configuration loss reaches a threshold. The details are as follows:

(1) We start with the initial block configuration $B$ and compute its loss as $L_b^0$ according to Equation 8, 10.
(2) For every pair of adjacent blocks $b_1$ and $b_2$ in $B$, we compute the quality (block configuration loss) of both possible block merge operations, $b_1$ into $b_2$ and $b_2$ into $b_1$.
(3) We greedily select the merge with the minimal block configuration loss among all possible merges computed in Step (2). We update the block configuration $B$ accordingly.
(4) Repeat Steps (2) and (3) until the current block configuration loss $L_b$ is larger than a threshold multiplied by the initial loss, i.e., $L_b > \eta_1 L_b^0$ or only one block is left. $\eta_1$ is a predefined threshold, which we typically set to $\eta_1 = 3.5$.

After block merging, the number of blocks is greatly reduced, resulting in a 4D skew polytope with much simpler topology. Any remaining topological changes—vertices with inter-frame edge degree greater than two (i.e., splitting and merging vertices)—correspond to colors appearing and disappearing (resp. splitting and merging) in the video.

## 5.4 Vertex Removal

After block merging, the 4D skew polytope has simpler topology but still contains unnecessary vertices. Unnecessary vertices manifest as redundant keyframes for a palette color, where the color does not change in the keyframe or else it changes exactly as would be expected by linearly interpolation. In this step, we progressively simplify the 4D skew polytope through iterative vertex removal without changing its topology.

*5.4.1 Removing one vertex.* We remove a vertex by replacing it with a *ghost vertex*. The ghost vertex is *virtually* defined and its position is enforced to lie on the inter-frame edge between non-ghost edge endpoints. As shown in Figure 6, after we remove vertex $u$, vertex $v$ and vertex $w$ are then connected to form a new inter-frame

edge $vw$. The ghost vertex $u'$ is enforced to lie on $vw$. Specifically, it is the intersection point of the hyperplane $t = t_i$ and the edge $vw$.

Our vertex removal scheme has two restrictions. First, splitting and merging vertices cannot be removed, since removing such vertices would change the topology generated by block merging. In Figure 6, vertex $r$ is a splitting vertex and cannot be removed. Second, vertices in the first and last frames cannot be removed.

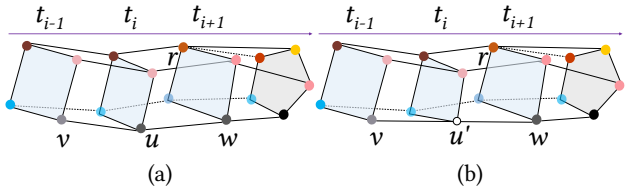



Fig. 6. Removing a vertex in the skew polytope. (a) Before removing vertex $u$; (b) vertex $u$ is removed and replaced by a ghost vertex $u'$.

*5.4.2 Iterative removal.* Similar to the block merging process, we also use the overall loss in Equation 8 to measure the quality of a simplified 4D skew polytope, this time with the frame weight $\theta_i = 1$ for all frames. We iteratively remove vertices until the overall loss reaches a threshold. The details are as follows:

(1) We start with the 4D skew polytope generated from block merging and compute its initial loss $L_v^0$ according to Equation 8.
(2) For each possible vertex, we compute the overall loss with that vertex removed. To reduce the search time, we only consider the $k$ vertices closest to their potential ghost position in RGBT-space as candidates for removal. In our implementation, we set $k = 10$.
(3) We greedily select and remove the vertex whose removal results in the smallest overall loss.
(4) Repeat Steps (2) and (3) until the current overall loss becomes a factor greater than the initial overall loss, i.e., $L_v > \eta_2 L_v^0$, or no additional vertices can be removed. We always set $\eta_2 = 3$.

## 5.5 Vertex Refinement

After simplifying the 4D skew polytope, we perform additional vertex refinement to further reduce the overall video loss. We also expect the 4D skew polytope to be temporally coherent so that polyhedral palettes change smoothly along the timeline. To measure the smoothness of a 4D skew polytope $P$, we define an additional smoothness term as:

$$K(P) = \frac{1}{|E|} \sum_{(u,v)\in E} \left| \frac{u_{RGB} - v_{RGB}}{u_t - v_t} \right|, \tag{11}$$

where we enumerate each edge $(u, v)$ in the inter-frame edge set $E$ of the 4D skew polytope $P$ and sum the slope of color variation with respect to time. The refinement loss is defined as the overall video loss (Equation 8) with the addition of this smoothness term:

$$L'_v(P, I) = L_v(P, \mathrm{I}) + \gamma K(P), \tag{12}$$

where the weight $\gamma$ is set to 0.01 and the frame weight $\theta_i$ in the overall video loss is set to 1.

We locally adjust the positions of all vertices to minimize the refinement loss in Equation 12. We employ an iterative scheme. Specifically, at each iteration, we select one vertex for local adjustment and keep all other vertices fixed. We use the NLopt library [Johnson [n.d.]] to locally adjust the position of a vertex $v_{RGBT}$ in 4D RGBT space constrained to the 4D box $[v_R - 0.1, v_R + 0.1] \times [v_G - 0.1, v_G + 0.1] \times [v_B - 0.1, v_B + 0.1] \times [v_b^s, v_b^e]$. Where $v_b^s$ and $v_b^e$ are the start and end times of the block where $v$ is located. To maintain the block structure, vertices in the first/last frames and splitting/merging vertices may not change their $t$ value. We cycle through all vertices several times for the above iterative refinement process until convergence (typically 3 cycles).

# 6 EXPERIMENTS

We ran our geometric palette extraction method on a PC with a 3.6 GHz CPU and 16 GB RAM. The initial 3D polyhedral palette for each video frame is generated using the Python code from Tan et al. [2018a] and C++ code provided by Wang et al. [2019].

We implemented an interactive tool on a MacBook Pro with a 3.1 GHz CPU and 8 GB RAM. Our GUI presents the 4D geometric palette in a timeline view familiar from video editing packages and a color picker for adjusting the selected palette color. The currently selected frame is shown before and after recoloring, along an RGB-space visualization of the frame's original and edited colors and sliced polyhedral palette.

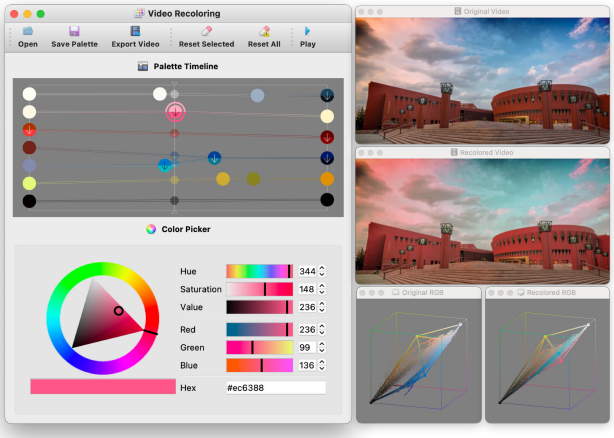



Fig. 7. Our video recoloring GUI presents the 4D geometric palette in a familiar timeline view (upper left). The currently selected frame before and after recoloring is shown in the upper and middle right, respectively, along with an RGB-space visualization of the frame's colors and sliced polyhedral palette.

## 6.1 Evaluation

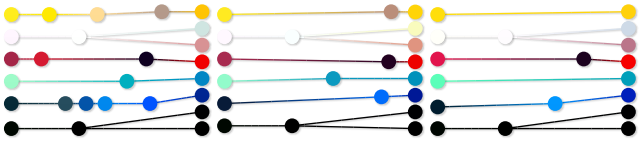
In this subsection, we evaluate the effects of parameters used in our method: three parameters ($\alpha$, $\eta_1$, and $\beta$) used in block merging (Section 5.3) and one parameter ($\eta_2$) used in vertex removal (Section 5.4). We also evaluate the necessity of the vertex refinement step (Section 5.5) and validate our choice of mixing weights.

Table 1. The number of blocks left after block merging with different values of $\alpha$ and $\eta_1$. The second column (Init.) displays the initial number of blocks before block merging. We found setting $\alpha = 10$ and $\eta_1 = 3.5$ leads to a good balance.

| Example | Init. | $\eta_1$ ($\alpha = 5$) | | | $\eta_1$ ($\alpha = 10$) | | | $\eta_1$ ($\alpha = 15$) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1.5 | 3.5 | 6.0 | 1.5 | 3.5 | 6.0 | 1.5 | 3.5 | 6.0 |
| Forest (Fig. 13(a)) | 61 | 3 | 1 | 1 | 3 | 2 | 1 | 5 | 2 | 1 |
| Street (Fig. 13(b)) | 75 | 8 | 3 | 1 | 12 | 3 | 2 | 13 | 4 | 2 |
| Cloud (Fig. 14(a)) | 77 | 1 | 1 | 1 | 2 | 1 | 1 | 5 | 1 | 1 |
| Night (Fig. 14(c)) | 74 | 1 | 1 | 1 | 6 | 3 | 1 | 6 | 2 | 1 |
| Water (Fig. 14(d)) | 98 | 5 | 2 | 1 | 9 | 3 | 2 | 12 | 5 | 2 |

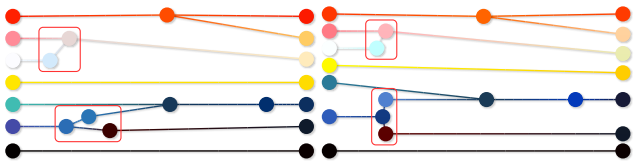
Table 2. The averaged vertex count of all polyhedral palettes after block merging with different values of $\beta$. We found setting $\beta = 1.0$ achieves a good balance.

| Example | $\beta = 0$ | $\beta = 1.0$ | $\beta = 2.0$ | $\beta = 5.0$ |
|---|---|---|---|---|
| Forest (Fig. 13(a)) | 8.0 | 7.4 | 7.4 | 7.4 |
| Street (Fig. 13(b)) | 8.0 | 8.0 | 8.0 | 8.0 |
| Cloud (Fig. 14(a)) | 8.0 | 7.0 | 7.0 | 7.0 |
| Night (Fig. 14(c)) | 7.0 | 7.0 | 6.6 | 6.6 |
| Water (Fig. 14(d)) | 7.4 | 7.0 | 7.0 | 7.0 |



(a) $\eta_2 = 1.5$, 26 vert (b) $\eta_2 = 3.0$, 20 vert (c) $\eta_2 = 6.0$, 18 vert

Fig. 8. Geometric palettes generated with different values of $\eta_2$. The palettes are generated from video example Playground (see Fig. 13 (c)).



(a) with vertex refinement (b) without vertex refinement

Fig. 9. Geometric palettes generated with and without vertex refinement. Without vertex refinement, color stops appear temporally closer to each other, leading to abrupt color changes.The palettes are generated from video example Night (see Fig. 14 (c)).

*6.1.1 Parameters $\alpha$ and $\eta_1$.* In block merging, the parameter $\alpha$ (Equation 10) controls how much the loss depends on block size, and $\eta_1$ controls when the iteration stops (i.e., when the current block configuration loss $L_b > \eta_1 L_b^0$). They together control how many blocks are left after block merging. Typically, larger $\alpha$ and smaller $\eta_1$ lead to more blocks and hence more complex topology of the 4D skew polytope, while smaller $\alpha$ and larger $\eta_1$ lead to fewer blocks and simpler topology.

We tested our block merging algorithm with different value combinations of $\alpha$ and $\eta_1$ on various examples. For each case, we recorded the number of blocks left after block merging. The results are provided in Table 1. Overall, we found that setting $\alpha = 10$ and $\eta_1 = 3.5$ produces a moderate number of blocks for all examples, leading to a good balance between topology complexity and ease of control.

*6.1.2 Parameter $\beta$.* In block merging, larger values of parameter $\beta$ (Equation 10) encourage using polyhedral palettes with smaller number of vertices. With $\beta = 0$, when merging two blocks containing polyhedral palettes with different vertex numbers, the algorithm will tend to keep the polyhedral palette with more vertices and discard the one with fewer vertices. Since polyhedral palettes with more vertices have more degrees of freedom to optimize, they typically produce smaller overall frame loss.

We tested our block merging algorithm with different values of $\beta$. We recorded the averaged vertex count of polyhedral palettes of all frames after block merging. The results are shown in Table 2. From the results, we see that setting $\beta > 0$ generally decreases the average vertex count compared to setting $\beta = 0$. We found setting $\beta = 1.0$ produces a moderate averaged vertex number and use it for all our results.

*6.1.3 Parameter $\eta_2$.* During vertex removal, the threshold $\eta_2$ controls when the iterative vertex removing process stops (i.e., $L_v > \eta_2 L_v^0$) and hence determines the complexity (i.e., vertex count) of the 4D skew polytope. Fewer vertices allow more convenient interaction but tend to result in higher overall loss. So a trade-off needs to be made. Figure 8 (a)–(c) show several geometric palettes generated with different values of $\eta_2$. It can be seen clearly that larger values of $\eta_2$ result in fewer vertices. Meanwhile, fewer vertices lead to larger overall video loss. We set $\eta_2 = 3.0$ to achieve a balanced trade-off.

*6.1.4 Vertex refinement.* As the final step in our pipeline, vertex refinement is performed to further reduce the overall video loss and improve temporal coherence. Figure 9 compares the geometric palette generated with and without vertex refinement. Vertex refinement decreases the slope of sharp inter-frame edges, and hence improves temporal coherence.

*6.1.5 Mixing weights.* In our experiments, we use Mean Value Coordinates (MVC) [Floater et al. 2005; Ju et al. 2005; Wang et al. 2019] to compute mixing weights. The choice of mixing weights is arbitrary. We experimented with an alternative RGBXY-space closed-form approach to compute smooth mixing weights [Tan et al. 2018a]. While both methods achieve satisfactory (and similar) recoloring results (Figure 10), the RGBXY method requires a costly additional preprocessing step in order to construct a 5D RGBXY convex hull for each video frame, taking minutes and gigabytes of memory per frame. In contrast, MVC is much faster, allowing real-time weight computation without preprocessing.

## 6.2 Comparisons

Recoloring approaches such as color transfer or edit propagation provide either less user control (color transfer) or require more user interaction (edit propagation). These are totally different kinds of user interaction from our method. Hence, we only compare our method with existing palette-based editing methods, including RGB convex hull methods [Tan et al. 2018a, 2016; Wang et al. 2019] and

(a) input video (b) MVC (c) RGBXY

Fig. 10. The same frame of an input video (a) recolored using MVC (b) and RGBXY (c) mixing weights. Both results are similarly satisfactory, yet MVC weights are much faster to compute and need no preprocessing. The results are generated from frame #99 of video example Forest (see Fig. 13 (a)).

a clustering-based method [Chang et al. 2015]. Since those existing palette-based methods are all designed for image recoloring, we extend them to handle videos.

For the RGB convex hull method, to extract a palette from a video, we gather all pixel colors in all frames together and compute a simplified RGB convex hull [Tan et al. 2018a, 2016] followed by local vertex refinement [Wang et al. 2019]. For recoloring, we extended these methods with two options for editing using the extracted palette. One option uses a constant palette across all frames. The other option treats the first and last frames as two keyframes. Users can adjust the palettes of the two keyframes separately. The palettes of in-between frames are computed by linear interpolation.

We extended the clustering based method [Chang et al. 2015] to handle video coloring in a similar way. We extract a palette from all pixel colors of all frames together. We similarly treat the first and last frames as two keyframes, linearly interpolating the editing palettes at the keyframes to achieve time-varying editing effects.

We also experimented with RGBT convex hulls, a simpler extension of the RGB convex hull method [Tan et al. 2016] to 4D. We first construct a 4D convex hull in RGBT space which encloses all video pixels over time. Each pixel's 4D values are its RGB colors and time $t$. We iteratively simplify the 4D RGBT convex hull as in Tan et al. [2016] until the vertex count of the convex hull reaches a predefined threshold. The 4D convex hull is a 4D polytope by definition, so we also slice it to obtain a polyhedral palette at a specific frame.

Figure 11 shows a comparison of our method with the RGB convex hull method, and the RGBT convex hull method. The input video Seasons is a time-lapse video of trees in four seasons. The recoloring intent is to make the earth in spring darker, leaves in summer greener, leaves in autumn more reddish, and fallen leaves in winter whiter. Notice that the geometric palette generated by our method captures object-level time-varying color changes rather well, e.g., the leaf color changes from green to yellow to orange. Our method successfully achieves the recoloring intent by adjusting the colors of a few vertices. In contrast, existing methods fail to adjust the leaf color in the middle of the video as expected, i.e., in summer and autumn. In addition, the polyhedral palettes generated by the RGBT convex hull method (Figure 11 bottom row) are extremely complex and less interpretable compared to the polyhedral palettes generated by our method.

Figure 12 shows an additional comparison on a second video Building. The recoloring intent is to make the clouds redder and sky greener during sunset in the middle of the video, but keep the sky color unchanged apart from sunset (during the day and night).

Table 3. Statistics. For each video example, we provide the number of frames, the total number of pixels, the vertex count of the extracted 4D polytope, the preprocessing time (pre) for extracting initial polyhedral palettes for all frames using [Tan et al. 2016; Wang et al. 2019], and the time taken by our (4D) geometric palette extraction method (excluding the time taken for extracting initial polyhedral palettes).

| Example | frame no. | pixel no. | vert no. | pre time | 4D time |
|---|---|---|---|---|---|
| Seasons (Fig. 1, Fig. 11) | 50 | 13M | 23 | 18 min | 12 min |
| Building (Fig. 12) | 120 | 96M | 20 | 36 min | 40 min |
| Forest (Fig. 13 (a)) | 100 | 46M | 14 | 48 min | 33 min |
| Street (Fig. 13 (b)) | 100 | 86M | 23 | 42 min | 48 min |
| Playground (Fig. 13 (c)) | 80 | 64M | 20 | 33 min | 35 min |
| Sunrise (Fig. 13 (d)) | 150 | 52M | 17 | 35 min | 52 min |
| City (Fig. 13 (e)) | 120 | 58M | 26 | 22 min | 55 min |
| Fountain (Fig. 13 (f)) | 60 | 17M | 24 | 26 min | 12 min |
| Cloud (Fig. 14 (a)) | 120 | 117M | 22 | 30 min | 61 min |
| Morning (Fig. 14 (b)) | 100 | 35M | 18 | 38 min | 40 min |
| Night (Fig. 14 (c)) | 100 | 92M | 22 | 26 min | 56 min |
| Water (Fig. 14 (d)) | 100 | 92M | 23 | 26 min | 45 min |
| Mountain (Fig. 15) | 120 | 55M | 20 | 31 min | 45 min |

In addition, we desire to make the building at the beginning of the video brighter, and the sky at the end of the video darker. Our method successfully achieve this recoloring intent. The other methods fail due to the lack of precise control in time. For example, the RGB convex hull method inevitably and undesirably changes the color of the sky in the first frame when editing the sky color in the middle frames.

The quantitative losses measured for videos Seasons (Figure 11) and Building (Figure 12) also verify that our generated palette is more compact. Our palette produces much lower compactness loss (i.e., $0.036$ and $0.1255$) than the RGB convex hull method (i.e., $0.0649$ and $0.3201$) and the RGBT convex hull method (i.e., $0.2635$ and $0.3515$), with only slightly higher reconstruction loss.

## 6.3 Results

We have used our method to recolor a diverse set of videos. Most of our example videos are time-lapses in order to exaggerate the color variation over time. In general, they contain complex and relatively rapid color changes and are rather challenging. Results are shown throughout the paper and in galleries in Figures 13 and 14. For example, in Figure 13(c), we adjust the color of the sky from orange to purple. In Figure 13(e), after recoloring, the light color gradually changes from green to yellow, and the morning looks warmer than the original video. In Figure 13(f), we make the tower's color change consistent with the fountain color variation. In Figure 14(a), we perform more color variations to the cloud and the mosque to make them more colorful.

Statistics for all examples are provided in Table 3, including the frame count, video resolution, size of the extracted 4D geometric palette (i.e., vertex count), the preprocessing time (i.e., the time for extracting initial polyhedral palettes for all frames using [Tan et al. 2016; Wang et al. 2019]) and the running time of our geometric palette extraction method. Once the geometric palette is extracted,

we provide real-time recoloring feedback, since only a single frame is needed for previewing. Our method produces natural, artifact-free recoloring. The time-varying palette reflects color changes in the input video, providing handles necessary for time-varying recoloring intents such as changing light and sky colors and their reflections.

Our tool also supports adding new keyframes (i.e., color stops) to the geometric palette. This allows more complex temporal color changes with more sophisticated user interactions. Such an example is shown in Figure 15. With the help of newly added color stops, we are able to introduce non-linear temporal color changes not present in the input video. We change the sky, cloud, and mountain from lavender to blue and then back to lavender.

### 6.4 User Study

We performed a user study to evaluate our palette extraction and recoloring algorithm as well as its GUI interface. We invited 16 participants, 7 male and 9 female, aged between 21 and 26. Nine of the participants were art school students or had extensive experience using state-of-the-art commercial video colorization tools. We asked the participants to perform four video recoloring tasks. Two tasks were guided with specific recoloring goals: make the video Building look old, and make the seasons more distinct in the video Seasons. The other two tasks were open-ended. We asked participants to freely edit two given videos however they like. Prior to the recoloring tasks, we provided a tutorial and a trial stage for participants to learn to use our tool. After participants completed all tasks, they responded to a questionnaire surveying our palette extraction accuracy, recoloring quality, and user experience.

Feedback was generally very positive. For example, 100% of participants were satisfied or very satisfied with the UI, 87.5% agree that the extracted geometric palettes reflected the video color distribution, 87.5% reported that they can easily achieve their recoloring intent using our tool, and 93.5% reported that the video recoloring result matched their expectations. Many of the participants were familiar with colorization in professional video editing software like DaVinci Resolve yet found our approach more flexible and reported that they would prefer our tool in the future. To simplify our interface for the user study, we disabled the ability to add new color stops. Expert users were already familiar with this functionality, however, and reported wanting such functionality in the post-study survey. This suggests that our approach can meet the needs of professional colorists. Statistics from our user study are provided in the supplementary materials.

## 7 LIMITATIONS AND CONCLUSION

### 7.1 Limitations

Our method has several limitations. While our method is able to support a range of useful color manipulations, linear-mixing-weight-based approaches cannot handle non-linear color manipulations such as tone mapping. Another limitation is that palette-based editing methods perform global recoloring, so some spatially-varying or per-object changes require spatial masks or additional semantic information. Finally, our method will probably be less successful for multi-shot videos with discontinuous temporal color changes, since we assume colors change smoothly over time. This problem could be bypassed by decomposing the multi-shot video into shots, and then extracting a 4D palette for each shot separately.

### 7.2 Conclusion and Future Works

We have shown that palette-based image recoloring can be extended naturally to videos by making the color palette a first-class object in a timeline view. By framing the problem in terms of a 4D RGBT skew polytope, our approach is able to capture topological changes to the palette. Our skew polytope leads to a vastly simpler and more editable polytope than a naive approach. Importantly, our user study participants, including those with extensive experience using state-of-the-art commercial colorization tools, preferred our approach.

There are many opportunities to improve the performance of our time-varying palette extraction. Currently, the analysis of a new video clip can be somewhat lengthy. We plan to explore frame skipping, GPU acceleration and additional pruning to further speed up our geometric palette extraction process. For example, removing a vertex may only affect nearby frames, so the loss function of distant frame need not be recomputed. Merging two large blocks typically leads to a big increase in the loss function, so we can prune rather than 'wasting time' trying to merge them.

We also wish to explore the constraints that come with nonlinear video editing workflows, where videos may be spliced and joined and otherwise edited. We also wish to explore palette extraction for streaming videos.

Another promising direction for future work is to generalize the piecewise linear nature of our RGBT geometry to allow curved color stops using splines as edges or even curved faces (as in Shugrina et al. [2020]).

## ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Project Numbers: 61822204, 61521002, 61863031). Yotam Gingold was supported in part by the United States National Science Foundation (IIS-1453018) and a gift from Adobe Systems Inc.. We would like to thank the reviewers for their constructive comments and suggestions.

## REFERENCES

Farhad Moghareh Abed. 2014. *Pigment identification of paintings based on Kubelka-Munk theory and spectral images.* Rochester Institute of Technology.

Adobe. 2020. Premiere Pro CC. https://www.adobe.com/products/premiere.html

M Afifi. 2019. Dynamic length colour palettes. *Electronics Letters* 55, 9 (2019), 531–533.

Elad Aharoni-Mack, Yakov Shambik, and Dani Lischinski. 2017. Pigment-Based Recoloring of Watercolor Paintings. In *NPAR* (Los Angeles, USA).

Yağiz Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. 2017. Unmixing-based soft color segmentation for image manipulation. *ACM Transactions on Graphics (TOG)* 36, 2 (2017), 19.

Xiaobo An and Fabio Pellacini. 2008. AppProp: All-pairs Appearance-space Edit Propagation. *ACM Trans. Graph.* 27, 3, Article 40 (Aug. 2008), 9 pages.

Raja Balasubramanian and Jan P Allebach. 1991. New approach to palette selection for color images. In *Human Vision, Visual Processing, and Digital Display II*, Vol. 1453. International Society for Optics and Photonics, 58–69.

Delos Baptiste, Mellado Nicolas, Vanderhaeghe David, and Cozot Remi. 2019. RGB Point Cloud Manipulation with Triangular Structures for Artistic Image Recoloring. *arXiv preprint arXiv:1912.04583* (2019).

Nicolas Bonneel, Kalyan Sunkavalli, Sylvain Paris, and Hanspeter Pfister. 2013. Example-based video color grading. *ACM Trans. Graph.* 32, 4 (2013), 39–1.

Ying Cao, Antoni B Chan, and Rynson WH Lau. 2017. Mining probabilistic color palettes for summarizing color use in artwork collections. In *SIGGRAPH Asia 2017 Symposium on Visualization*. 1–8.

M Emre Celebi. 2011. Improving the performance of k-means for color quantization. *Image and Vision Computing* 29, 4 (2011), 260–271.

Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. 2015. Palette-based Photo Recoloring. *ACM Trans. Graph.* 34, 4 (July 2015).

Xiaowu Chen, Dongqing Zou, Qinping Zhao, and Ping Tan. 2012. Manifold preserving edit propagation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–7.

Wengang Cheng, Pengli Dou, and Dengwen Zhou. 2020. An Illumination Insensitive and Structure-Aware Image Color Layer Decomposition Method. In *International Conference on Multimedia Modeling*. Springer, 163–175.

Wan-Fung Cheung and Yuk-Hee Chan. 2003. Color quantization of compressed video sequences. *IEEE transactions on circuits and systems for video technology* 13, 3 (2003), 270–276.

Stephen DiVerdi, Jingwan Lu, Jose Echevarria, and Maria Shugrina. 2019. Generating playful palettes from images. In *Proceedings of the 8th ACM/Eurographics Expressive Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*. Eurographics Association, 69–78.

Sylvain Duchêne, Carlos Aliaga, Tania Pouli, and Patrick Pérez. 2017. Mixed Illumination Analysis in Single Image for Interactive Color Grading. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Los Angeles, California) *(NPAR '17)*. ACM, New York, NY, USA, Article 10, 10 pages.

D Duque-Arias, Santiago Velasco-Forero, Francois Goulette, Jean-Emmanuel Deschaud, and Beatriz Marcotegui. 2019. A graph-based color lines model for image analysis. In *International Conference on Image Analysis and Processing*. Springer, 181–191.

Zunlei Feng, Wolong Yuan, Chunli Fu, Jie Lei, and Mingli Song. 2018. Finding intrinsic color themes in images with human visual perception. *Neurocomputing* 273 (2018), 395–402.

Michael S. Floater, Geza Kos, and Martin Reimers. 2005. Mean value coordinates in 3D. *Computer Aided Geometric Design* (2005).

Michael Gervautz and Werner Purgathofer. 1988. A simple method for color quantization: Octree quantization. In *New trends in computer graphics*. Springer, 219–231.

Gary R Greenfield and Donald H House. 2003. Image recoloring induced by palette color associations. (2003).

Gary R Greenfield and Donald H House. 2005. A palette-driven approach to image color transfer. In *Proceedings of the First Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*. 91–99.

Mairéad Grogan and Aljosa Smolic. 2020. Image Decomposition using Geometric Region Colour Unmixing. In *European Conference on Visual Media Production*. 1–10.

Mingming He, Jing Liao, Dongdong Chen, Lu Yuan, and Pedro V Sander. 2019. Progressive color transfer with dense semantic correspondences. *ACM Transactions on Graphics (TOG)* 38, 2 (2019), 1–18.

Hao-Zhi Huang, Kun Xu, Ralph R. Martin, Fei-Yue Huang, and Shi-Min Hu. 2016. Efficient, Edge-Aware, Combined Color Quantization and Dithering. *IEEE Transactions on Image Processing* 25, 3 (2016), 1152–1162.

Yifei Huang, Sheng Qiu, Changbo Wang, and Chenhui Li. 2020. Learning Representations for High-Dynamic-Range Image Color Transfer in a Self-Supervised Way. *IEEE Transactions on Multimedia* (2020).

Taehong Jeong, Myunghyun Yang, and Hyun Joon Shin. 2019. Succinct Palette and Color Model Generation and Manipulation Using Hierarchical Representation. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 1–10.

Steven G. Johnson. [n.d.]. *The NLopt nonlinear-optimization package*. http://github.com/stevengj/nlopt

T Ju, S Schaefer, and J Warren. 2005. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics* 24, 3 (2005), 561–566.

Suzi Kim and Sunghee Choi. 2020. Automatic Color Scheme Extraction from Movies. In *Proceedings of the 2020 International Conference on Multimedia Retrieval*. 154–163.

Yuki Koyama and Masataka Goto. 2018. Decomposing images into layers with advanced color blending. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 397–407.

Daniel C Kusswurm. 1998. Color quantizer for video sequences. In *Very High Resolution and Quality Imaging III*, Vol. 3308. International Society for Optics and Photonics, 105–113.

Anat Levin, Dani Lischinski, and Yair Weiss. 2004. Colorization Using Optimization. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 689–694.

Sharon Lin, Matthew Fisher, Angela Dai, and Pat Hanrahan. 2017. LayerBuilder: Layer Decomposition for Interactive Image and Video Color Editing. *CoRR* abs/1701.03754 (2017). arXiv:1701.03754 http://arxiv.org/abs/1701.03754

Sharon Lin and Pat Hanrahan. 2013. Modeling how people extract color themes from images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3101–3110.

Dani Lischinski, Zeev Farbman, Matt Uyttendaele, and Richard Szeliski. 2006. Interactive Local Adjustment of Tonal Values. *ACM Trans. Graph.* 25, 3 (July 2006), 646–653.

Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. 2017. Deep photo style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4990–4998.

Abhimitra Meka, Mohammad Shafiei, Michael Zollhoefer, Christian Richardt, and Christian Theobalt. 2019. Live Illumination Decomposition of Videos. *arXiv preprint arXiv:1908.01961* (2019).

Nicolas Mellado, David Vanderhaeghe, Charlotte Hoarau, Sidonie Christophe, Mathieu Brédif, and Loic Barthe. 2017. Constrained palette-space exploration. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 60.

RMH Nguyen, B Price, S Cohen, and MS Brown. 2017. Group-Theme Recoloring for Multi-Image Color Consistency. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 83–92.

Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2011. Color Compatibility from Large Datasets. *ACM Trans. Graph.* 30, 4, Article 63 (2011).

Michael T Orchard, Charles A Bouman, et al. 1991. Color quantization of images. *IEEE transactions on signal processing* 39, 12 (1991), 2677–2690.

Celal Ozturk, Emrah Hancer, and Dervis Karaboga. 2014. Color image quantization: a short review and an application with artificial bee colony algorithm. *Informatica* 25, 3 (2014), 485–503.

Fabio Pellacini and Jason Lawrence. 2007. AppWand: Editing Measured Materials Using Appearance-driven Optimization. *ACM Trans. Graph.* 26, 3, Article 54 (July 2007).

E. Reinhard, M. Adhikhmin, B. Gooch, and P. Shirley. 2001. Color transfer between images. *IEEE Computer Graphics and Applications* 21, 5 (July 2001), 34–41.

Christian Richardt, Jorge Lopez-Moreno, Adrien Bousseau, Maneesh Agrawala, and George Drettakis. 2014. Vectorising bitmaps into semi-transparent gradient layers. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 11–19.

Evgeny Roytman and Craig Gotsman. 1995. Dynamic color quantization of video sequences. *IEEE Transactions on Visualization and Computer Graphics* 1, 3 (1995), 274–286.

Maria Shugrina, Amlan Kar, Sanja Fidler, and Karan Singh. 2020. Nonlinear color triads for approximation, learning and direct manipulation of color distributions. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 97–1.

Jianchao Tan, Stephen DiVerdi, Jingwan Lu, and Yotam Gingold. 2019. Pigmento: Pigment-Based Image Analysis and Editing. *Transactions on Visualization and Computer Graphics (TVCG)* 25, 9 (2019). https://doi.org/10.1109/TVCG.2018.2858238

Jianchao Tan, Marek Dvorožňák, Daniel Sýkora, and Yotam Gingold. 2015. Decomposing Time-Lapse Paintings into Layers. *ACM Transactions on Graphics (TOG)* 34, 4 (July 2015), 61:1–61:10.

Jianchao Tan, Jose Echevarria, and Yotam Gingold. 2018a. Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–10.

Jianchao Tan, Jose Echevarria, and Yotam Gingold. 2018b. Palette-based image decomposition, harmonization, and color transfer. *arXiv preprint arXiv:1804.01225* (2018).

Jianchao Tan, Jyh Ming Lien, and Yotam Gingold. 2016. Decomposing Images into Layers via RGB-Space Geometry. *Acm Transactions on Graphics* 36, 1 (2016), 1–14.

Yili Wang, Yifan Liu, and Kun Xu. 2019. An Improved Geometric Approach for Palette-based Image Decomposition and Recoloring. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 11–22.

Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. 2002. Transferring Color to Greyscale Images. *ACM Trans. Graph.* 21, 3 (July 2002), 277–280.

Quan Wen and M Emre Celebi. 2011. Hard versus fuzzy c-means clustering for color quantization. *EURASIP Journal on Advances in Signal Processing* 2011, 1 (2011), 1–12.

Wikipedia. 2020. Regular skew polyhedron. https://en.wikipedia.org/wiki/Regular_skew_polyhedron

Xiaolin Wu. 1992. Color quantization by dynamic programming and principal analysis. *ACM Transactions on Graphics (TOG)* 11, 4 (1992), 348–372.

Kun Xu, Yong Li, Tao Ju, Shi-Min Hu, and Tian-Qiang Liu. 2009. Efficient Affinity-based Edit Propagation using K-D Tree. *ACM Transactions on Graphics* 28, 5, Article 118 (2009), 118:1–118:6 pages.

C. Yang and L. Peng. 2008. Automatic Mood-Transferring between Color Images. *IEEE Computer Graphics and Applications* 28, 2 (2008), 52–61. https://doi.org/10.1109/MCG.2008.24

Q. Zhang, Y. Nie, L. Zhu, C. Xiao, and W.-S. Zheng. 2021. A Blind Color Separation Model for Faithful Palette-based Image Recoloring. *IEEE Transactions on Multimedia* (2021), 1–1. https://doi.org/10.1109/TMM.2021.3067463 Conference Name: IEEE Transactions on Multimedia.

Qing Zhang, Chunxia Xiao, Hanqiu Sun, and Feng Tang. 2017. Palette-Based Image Recoloring Using Color Decomposition Optimization. *IEEE Transactions on Image Processing* 26, 4 (2017), 1952–1964.

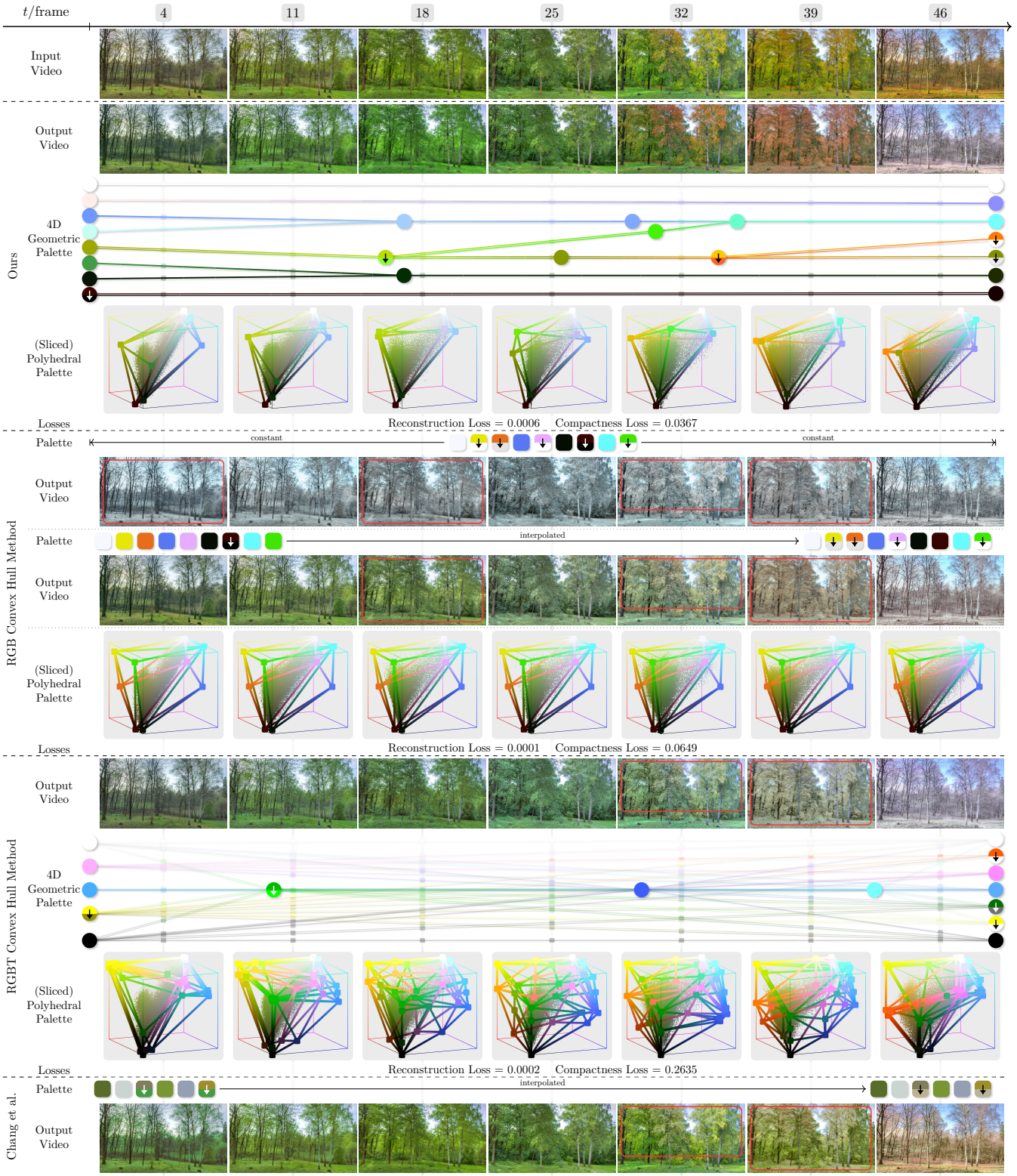



Fig. 11. Comparison of our method with RGB convex hull based methods [Tan et al. 2018a, 2016; Wang et al. 2019], a clustering base method [Chang et al. 2015], and the RGBT convex hull method. The recoloring intent is to emphasize the change in seasons by making the earth darker in spring, the leaves greener in summer and redder in autumn, and the fallen leaves whiter in winter. Our palette captures object color changes in the video, such as the change in leaf color from green to yellow, and finally orange. This enables direct and flexible user edits. Note the differences in the highlighted regions. ©Eirik Solheim.

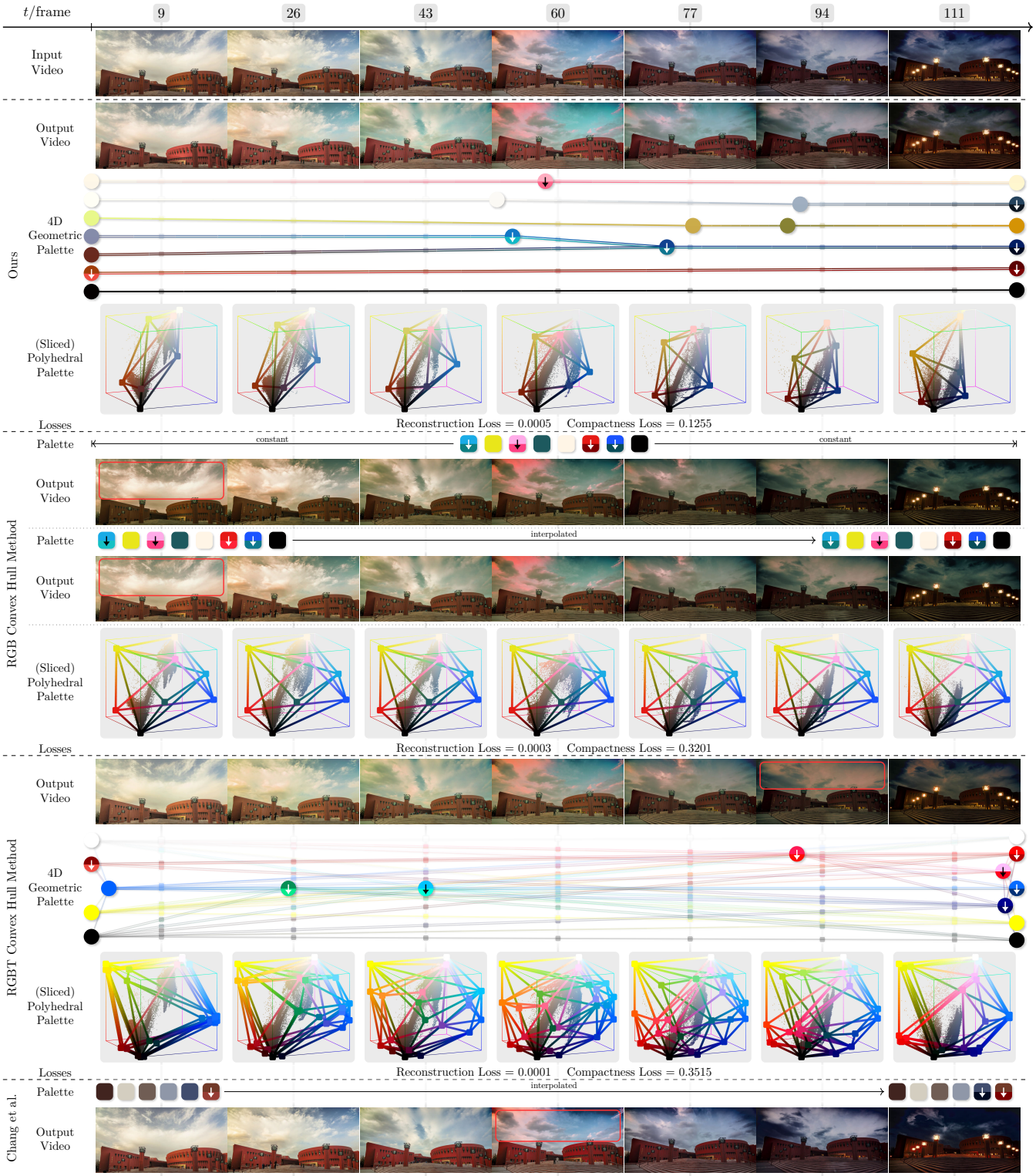



Fig. 12. Another comparison of our method with RGB convex hull based methods [Tan et al. 2018a, 2016; Wang et al. 2019], a clustering base method [Chang et al. 2015], and the RGBT convex hull method we describe. The recoloring intent is to make the clouds glow red and the sky tint green during sunset, but keep the original sky color unchanged at other times (day and night). In addition, the intent is to make the building at the beginning of the video brighter and the sky at the end of the video darker. The RGBT convex hull method extracts the wrong time for sunset, confusing the user on how to achieve their goal and undesirably modifying the sky color after sunset. Note the differences in the highlighted regions. ©Haitong Yu.

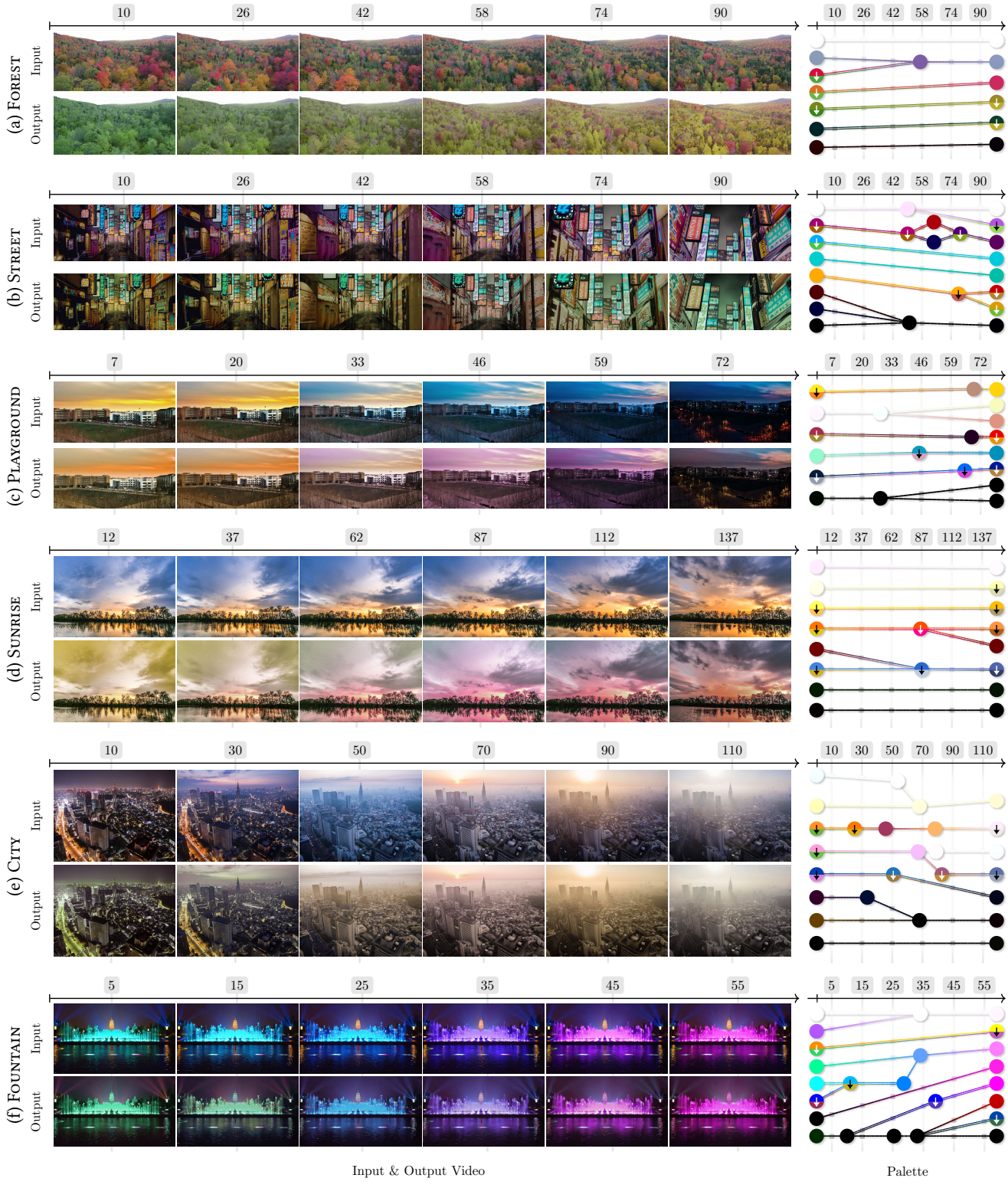



Fig. 13. Diverse results of video recoloring using our method. For each example, the timeline view of the extracted geometric palette is shown on the right. Each circular node represents a palette color at a point in time. Color adjustments are displayed by a vertical arrow, upper half-disk: before editing, lower half-disk: after editing. ©Home Adrift (a), Carl_watermark (b), Haitong Yu (c), Felix Mittermeier (d), Travel Hungry For Life (e), Subway-Sun (f).

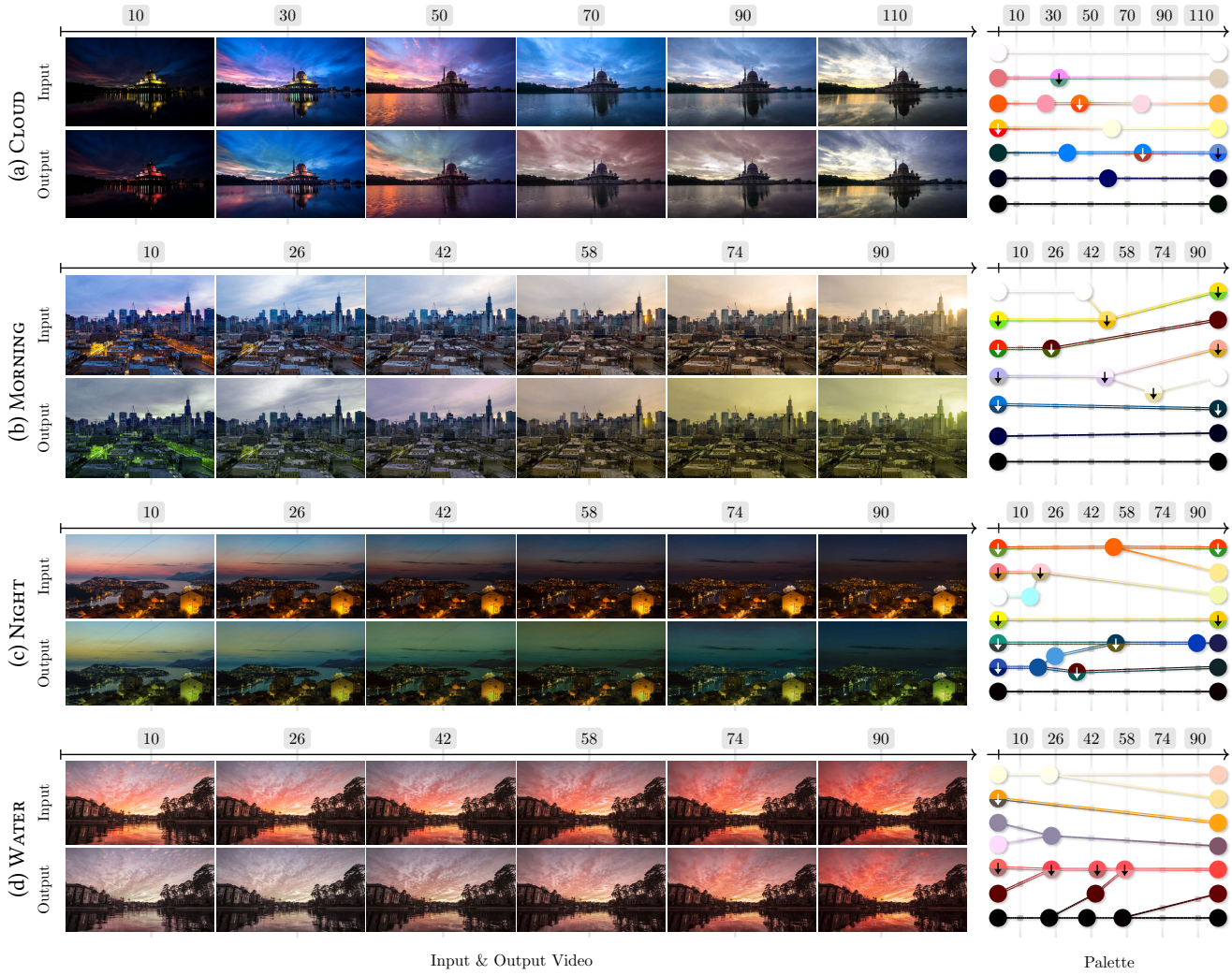



Fig. 14. More recolored videos using our method. ©Islamic Stock Library (a), Bhargava Marripati (b), Lukas Bieri (c), NewLayer (d).

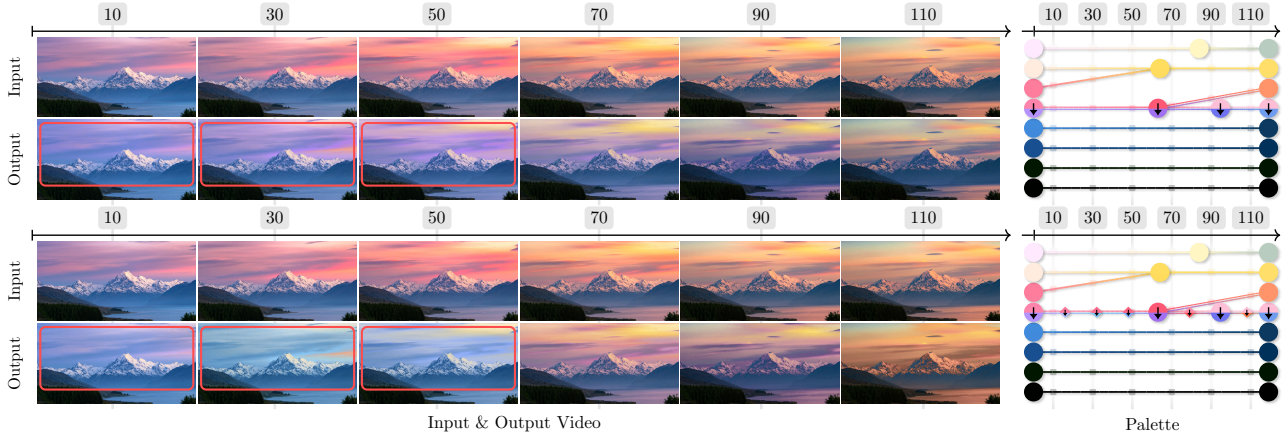



Fig. 15. Recolored results without (top) and with (bottom) the addition of new keyframes. ©Guguheguagua.