Computers & Graphics 96 (2021) 61–70

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Efficient propagation of sparse edits on 360° panoramas☆

Yun Zhang[a,*], Fang-Lue Zhang[b], Yu-Kun Lai[c], Zhe Zhu[d]

[a] College of Media Engineering, Communication University of Zhejiang, China
[b] School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
[c] School of Computer Science and Informatics, Cardiff University, UK
[d] Department of Radiology, Duke University, USA

ARTICLE INFO

Article history:
Received 20 September 2020
Revised 28 February 2021
Accepted 30 March 2021
Available online 8 April 2021

Keywords:
360° panoramas
Sparse strokes
Edit propagation
Manifold-preserving
Adaptive KNN

ABSTRACT

We present an efficient method to propagate sparse user edits indicated by strokes on 360° panoramas. Our algorithm first projects each equirectangular pixel to its corresponding position on a 3D unit sphere, so each pixel can be characterized by a feature vector consisting of its 3D coordinates and RGB color values. We formulate edit propagation as an optimization problem that aims to satisfy the user edit constraints while preserving the manifold structure of the image at the same time. To solve the problem using a linear system efficiently, we first construct the K-D tree structure in the feature space to cluster pixels. Then we optimize the manifold structure where both the number of nearest neighbors and their corresponding weights are determined by the feature distributions. We further apply a multiresolution strategy to speedup the edit propagation. Our method is the first to perform interactive edit propagation on 360° panoramas. Experiments show that our method is able to generate seam-free and visually pleasing results, and users can receive instant feedback during interactive editing.

1. Introduction

360° panoramas allow recording a 360° view of the place being photographed, which has been used as an important source of Virtual Reality (VR) media. The most immersive way to view 360° panoramas is to use a dedicated head mounted display (HMD) like the Oculus Rift or HTC Vive. Due to its recent popularity, support for viewing panoramic contents has been added in traditional image viewers, where users can drag to view different parts of a panorama. While most recent works focus on panorama generation and compression [1], less attention has been paid to panorama editing. Directly applying planar image editing techniques to 360° panoramas is both inappropriate and inefficient for the following reasons: (1) 360° panorama is essentially defined on a spherical surface, which means directly applying distance metric used for 2D planar images is problematic, and can lead to visible seams and inconsistent propagations after editing (see e.g. Figs. 7, and 9); (2) panoramic images usually contain more content than planar images as they cover the entire 360° field-of-view, making the computational complexity and memory cost much higher.

Edit propagation is one of most important problems in image editing, which aims to propagate pixel-wise edits indicated by users' strokes to all the pixels in the image, with similar pixels nearby receiving most influence. The edits could be the amount of color adjustment, relighting, transparency value for defogging or matting, etc. Traditional image edit propagation techniques [2–6] map all the pixels to a 5D feature space (r, g, b, x, y) where r, g, b correspond to colors, and x, y are the pixel coordinates in 2D image plane. They favor the pixels with similar features to receive similar edits as the pixels covered by strokes and formulate this as an optimization problem to maximize that similarity while maintaining image structure. Different from the above methods, we take 360° panoramas using the equirectangular representation. Under this setting, the actual horizontal distance between the points represented by two neighboring pixels decreases along with increasing latitude. We visualize this distortion by the Tissot's indicatrices over a equirectangular 360° image in Fig. 1. This gives an intuition that common 2D distance metrics poorly describe the spatial relationship between pixels in an equirectangular image. In this paper, we lift pixels on the 2D image plane to their 3D spherical positions, where their actual distances can be approximated. More specifically, in an efficient KNN (K-nearest neighbor) search step, the distance between two pixels is calculated as their Euclidean distance in 3D coordinates. Then in the propagation step, the accurate big circle distance is used for effective propagation. The 3D spherical position along with RGB color channels form a

☆ This paper was recommended for publication by Joã o Luiz Dihl Comba.

* Corresponding author.
E-mail addresses: zhangyun_zju@zju.edu.cn (Y. Zhang), fanglue.zhang@vuw.ac.nz (F.-L. Zhang), Yukun.Lai@cs.cardiff.ac.uk (Y.-K. Lai), ajex1988@gmail.com (Z. Zhu).

https://doi.org/10.1016/j.cag.2021.03.005
0097-8493/

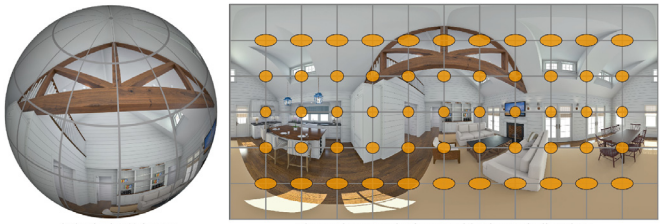

Fig. 1. Visualization of distortions in a 360° panoramas.

6D feature space (r, g, b, x, y, z), and we use the method of [7] to preserve the high-dimensional manifold structure, which helps to propagate users' edits in a spatially consistent manner. Since the optimization is performed over all the pixels, and the main computational cost lies in the optimization that involves solving a system of linear equations with the same scale as the number of pixels, we need to reduce the scale of the linear system since a typical panorama contains millions of pixels. To achieve this without degrading the visual quality, we propose to use two techniques for acceleration. We first cluster all pixels using a K-D tree structure in the feature space and just use the node corners of the K-D tree to approximate original pixels. huThis significantly reduces computation time and saves memory as the number of corners is orders of magnitude less than that of pixels. Then we just need to optimize the manifold structure of all the node corners by the adaptive KNN (K nearest neighbor) method where the number of nearest neighbors and the corresponding weights are determined by the feature distributions. For instant edit propagation in ultra high-definition panoramas, we further propose a multiresolution approach.

Inspired by previous edit propagation methods for planar images [2–6], our edit propagation on 360° panoramas is formulated as an optimization that aims to maximize the similarity of the edits between pixels with similar features with the following constraints: (1) edits on stroke pixels need to be maintained; (2) the amount of edits on other pixels should be determined by their distance to stroke pixels in the feature space. Since pixels of 360° images are essentially distributed on a sphere, their distance should be measured by the great circle distance,[1] which poses a challenge for the optimization. By approximating the great circle between two points by the line segment connecting them, we can approximate the manifold by a linear structure locally, and the problem can be formulated using a linear system and solved efficiently. This approximation maintains the ordering of distances compared with great circle distances, and is a more accurate approximation for close pixels which are more important for edit propagation.

To the best of our knowledge, we are the first to propose a stroke-based editing approach on 360° panoramas, and users only provide sparse strokes, thus avoiding the laborious and tedious regions of interest (ROIs) selection. We summarize our main contributions as follows:

- We propose the first stroke-based edit propagation method on 360° panoramas, which provides seam-free and visually pleasing editing results by introducing the spherical distance metric and constructing the manifold structure for each pixel in the spherical domain.
- To achieve instant feedback when propagating edits on ultra high-definition 360° panoramas, we further develop an efficient solution to propagate user-specified edits while preserving the spherical manifold structure and editing quality.

[1] A *great circle* is any circle that circumnavigates the sphere and passes through the center of the sphere. The great circle distance is the arc length between two points on the great circle that passes through the two points.

2. Related work

2.1. Edit propagation

The pioneering work in edit propagation, proposed by An and Pellacini [8], created a simple and intuitive interface for image editing, where users only provide sparse inputs (usually strokes) while the algorithm propagates the edits to the proper regions in the rest of the image, based on the pixel-level affinities. Due to the huge number of pixels, formulating the affinity-based edit propagation as an energy minimization problem that involves an all-pixel-pair propagation energy makes it infeasible to solve instantly. One way for acceleration is to use clusters [9,10] to represent pixels as their linear combinations, which significantly reduces the number of unknown variables. By reformulating edit propagation as a function interpolation problem in a high-dimensional feature space, Li et al. [6] efficiently solved the problem using radial basis functions. Another interpolation based method, proposed by Yatagawa and Yamaguchi [11], approximated the edit parameters with convex combinations of samples, which can achieve a better accuracy in terms of colors and edit parameters. Another acceleration approach worth mentioning is the hierarchical data structure based method [12] which achieved scalable edit propagation.

Besides the efficiency improvements, much research attention has been paid to improving the visual quality of edit propagation. To avoid visual artifacts after editing, Ma and Xu [13] proposed an algorithm to mitigate the aliasing artifacts. While the method is simple to implement, it achieves excellent anti-aliasing results. Chen et al. [7] first proposed manifold preserving edit propagation. By representing each pixel as a linear combination of its neighbors in the feature space, their method is more robust to color blending in the input data compared with previous methods. Manifold preserving edit propagation can be accelerated by K-D trees [5] or quad-trees [2]. These approaches however require adequate user inputs to ensure editing quality. To address this, some frameworks [14,15] target reducing the burden at the user end by only requiring a small amount of user guidance. Besides edit propagation on images, Yatagawa and Yamaguchi [16] proposed a temporally coherent video editing method on a frame-by-frame basis.

Recently, machine learning based methods, especially deep learning based approaches, have been successfully applied to edit propagation. Oh et al. [17] formulated the edit propagation as a classification problem which can be efficiently solved using a support vector machine (SVM) to support high-resolution image inputs. Chen et al. [4] proposed to utilize sparse dictionary learning to improve the memory efficiency while maintaining a high visual fidelity. Zhang et al. [18] proposed a color decomposition method for flexibly recoloring images while preserving the inherent color characteristics. Endo et al. [3] trained a deep neural network on users' strokes and used the trained model to determine in which regions to propagate the edits.

While all the above methods achieve promising edit propagation results on 2D planar images/videos, directly applying them to 360° panoramas without considering 3D spatial relations between pixels leads to unsatisfactory results.

2.2. 360° panorama editing

360° media, consisting of 360° videos and images, is a great way in VR applications to provide users with an immersive experience. Due to the limitations in devices, directly capturing 360° panoramas using an ultra-wide camera suffers from image quality loss, especially in the regions far from the image center. Therefore, the most popular way in 360° panorama creation is stitching together multiple images captured by a multi-camera rig [19,20]. Then proper blending techniques [21] can be used to

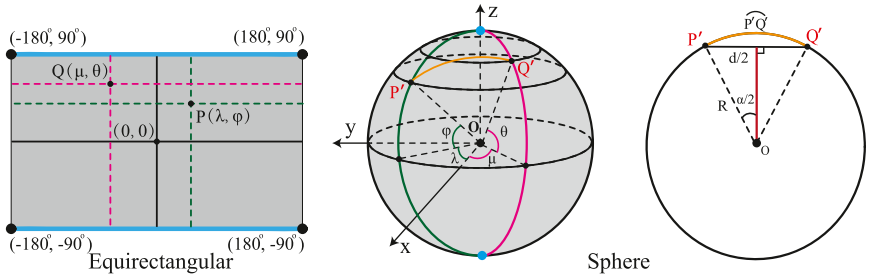

Fig. 2. Transformation of equirectangular coordinates into spherical coordinates.

ensure a smooth transition between overlapping images. However, existing multi-camera rigs usually cannot capture the entire 360° field of view, and completion [22] techniques are usually applied to hole regions which often appear at the top or bottom of the panorama. A detailed survey about panoramic content creation can be found in [23]. The quality of the constructed panoramas can be evaluated using visual quality assessment [1]. Typical editing operations, such as copy-and-paste [24], can be adapted to 360° panoramas by considering the sphere geometry constraint embedded in panoramic images. Such constraints also need to be taken into account in panoramic image classification [25,26] and object detection [27] in panoramas.

3. Our approach

3.1. Overview

As shown in Fig. 7 (left), the input of our method is an equirectangular 360° panorama and user-specified strokes indicating the desired edits (with different colors indicating different types of appearance editing, e.g., a red stroke means recoloring to red, while a black stroke means keeping unchanged). In our framework, each pixel of the input 360° panorama is characterized by a 6-dimensional feature vector $\mathbf{f}_i = (\mathbf{c}_i/\sigma_c, \mathbf{p}_i/\sigma_p)$, where $\mathbf{c}_i$ refers to its RGB color and $\mathbf{p}_i$ refers to its 3D position (x, y, z) on a unit sphere. σ_c and σ_p are used to balance the importance of the two components.

To project the 2D coordinate of a pixel $P(x_p, y_p)$ (See Fig. 2) in an equirectangular image (width $= W$, height $= H$) to its 3D spherical coordinate (x, y, z), we first calculate its longitude and latitude coordinates (λ, ϕ) on the sphere by

$$\begin{cases} \lambda = ((x_p + 0.5)/W - 0.5) * 2 * \pi \\ \phi = (0.5 - (y_p + 0.5)/H) * \pi. \end{cases} \tag{1}$$

Then, we calculate its 3D coordinates on a unit sphere, where we set the 3D point $(1, 0, 0)$ as the original point, and set its longitude and latitude to 0. Finally the 3D coordinates can be calculated as

$$\begin{cases} x = \cos\phi \cos\lambda \\ y = \cos\phi \sin\lambda \\ z = \sin\phi. \end{cases} \tag{2}$$

Because 360° panoramas are essentially defined on sphere, the actual distance between pixels cannot be measured correctly by directly applying the Euclidean distance metric on (x, y) image coordinates. Instead, using (x, y, z) spherical coordinates can reflect the actual positions of pixels and guarantee correct edit propagation on the spherical domain. It not only conforms to the spherical nature of 360° images, but also simplifies and accelerates the KNN search on sphere, thus can well preserve the spherical manifold structure in edit propagation.

We formulate the 360° panorama edit propagation as an optimization problem, where we encourage the solution to preserve the edits on pixels within the strokes while propagating the edits to nearby pixels in the feature space, making the editing results more natural. Inspired by Chen et al. [2,5,7], our optimization for edit propagation is formulated in a manifold-preserving framework.

In the following subsections, we first describe how to construct the manifold on 360° panoramas. Then we present the mathematical formulation of the edit propagation on panoramas, which is an energy minimization defined on pixel edits. We also provide several techniques to speed up the computation, including K-D tree clustering, adaptive KNN and multiresolution speedup.

3.2. Manifold construction on 360° panoramas

Inspired by Roweis and Saul [28], we use Locally Linear Embedding (LLE) to map a high dimensional space to a low dimensional manifold, with the intuition that each feature point can be approximately described by a linear combination of its neighbors. The weight of each neighbor is calculated by minimizing the following objective:

$$\sum_{i=1}^{N} \|\mathbf{f}_i - \sum_{j=1}^{K} w_{ij}\mathbf{f}_{n_{ij}}\|^2, \tag{3}$$

where N refers to the number of pixels, K is the number of neighbors used for a pixel. w_{ij} is the weight for the jth neighbor of the ith pixel, satisfying $\sum_{j=1}^{K} w_{ij} = 1$. n_{ij} is the pixel index of the jth neighbor of the ith pixel. $\mathbf{f}_i$ is the feature vector of the ith pixel and $\mathbf{f}_{n_{ij}}$ is the feature vector of its jth neighbor.

To minimize the above defined energy function, we first need to find out the neighbors for each pixel. Existing K-nearest neighbor (KNN) search techniques are designed based on the L^2-Norm between the feature vectors, which means that all dimensions will be treated equally when calculating the distance. However, on the spherical surface, the spatial distance between two points should be the great-circle distance, i.e., the length of the arc connecting them, see Fig. 2. To be compatible with existing KNN search techniques, we instead use the Euclidean distance($\overline{P'Q'}$) between the two points to approximate their great-circle distance ($\overset{\frown}{P'Q'}$), since the Euclidean distance can be easily obtained by calculating the L^2-Norm between their coordinates in 3D space. This is also a reasonable approximation since only spatially close pixels are of concern in this process, and we can utilize efficient KNN search technologies (such as a K-D tree) to calculate the top K nearest neighbors of each pixel. The weight matrix $\mathbf{W}$, which is an $N \times K$ matrix and whose element w_{ij} is defined in Eq. 3, can be obtained by solving a sparse linear system [28].

3.3. Manifold preserving edit propagation

We denote users' edit for each pixel as g_i and its propagated edit as e_i. Note that g_i is known and e_i is the target to optimize.

To obtain the optimal e_i for every pixel in a manifold preserving manner, we define the following energy function [7]:

$$E = \sum_{i=1}^{N} u_i (e_i - g_i)^2 + \sum_{i=1}^{N} \left(e_i - \sum_{j=1}^{K} w_{ij} e_{n_{ij}} \right)^2, \tag{4}$$

where $u_i \in \{0, 1\}$ is the edit indicator, used to indicate whether there is user edit on pixel i (0 - no user edits, 1 - otherwise). The first term of the energy function encourages the final result to follow the users' edits, while the second term preserves the manifold structure of the panorama. We do not give weights to them since we think those two terms are of equal importance.

3.4. Optimization

Since the energy function defined in Eq. (4) is quadratic, it can be minimized by solving a large sparse linear system. However, directly solving it is costly since the complexity of the linear system depends on the pixel number N (in the order of millions for panoramas) and neighborhood size K of each pixel. For instant feedback, we prefer smaller N and K, and propose the following acceleration strategies. We use a K-D tree to cluster all the pixels in the feature space to reduce the number of unknowns in the above objective, see Section 3.4.1. We also propose adaptive KNN to adaptively determine the neighborhood size K and weights of neighbors, see Section 3.4.2. For further acceleration, we propose a multiresolution speedup strategy, see Section 3.4.3. By combining those strategies, we achieved significant acceleration without loss of the quality of editing results.

3.4.1. K-D tree construction

As in [5,9], we apply the K-D tree structure to cluster the pixels of the input 360° panorama in the feature space. With this hierarchical data structure, the objective can be rewritten as a function of the K-D tree node corners instead of pixels, which largely reduces the number of unknowns and order of magnitude. With M ($M << N$) indicating the number of node corners of the K-D tree, the energy function in Eq. (4) can be rewritten as:

$$E = \sum_{i=1}^{M} m_i^2 \left(\widetilde{u}_i (\widetilde{e}_i - \widetilde{g}_i)^2 + \sum_{i=1}^{M} \left(\widetilde{e}_i - \sum_{j=1}^{k} w_{ij} \widetilde{e}_j \right)^2 \right), \tag{5}$$

where i enumerates all node corners of the K-D tree; $\widetilde{u}_i \in [0, 1]$ and $\widetilde{g}_i$ refer to the edit strength and the value at node corner i, which are defined as a weighted sum of all pixels in the neighboring tree nodes. m_i is used to define the multiplicity of pixels contributing to the node corner (see [5] for more details). w_{ij} represents the weights of neighboring node corners of node corner i, which are used to reconstruct the manifold structure over all node corners. With the edited result $\widetilde{e}_i$ on each node corner, we calculate the edit on each pixel through a multiple linear interpolation using its enclosing node corners.

3.4.2. Adaptive KNN

After the K-D tree construction, the manifold structure is reconstructed using the node corners, and now KNN refers to the K nearest corners. In this section, we propose to optimize the manifold structure using the *adaptive KNN*, which adaptively determines the number of nearest neighbors and their corresponding weights according to the feature distributions. See Fig. 3, the fixed K strategy is not optimal, and it always requires larger K to ensure satisfying editing results. Inspired by Ma and Xu [5], we propose to use an adaptive K to achieve a good balance between the efficiency and visual effects. Observing that different regions in the spherical domain require different numbers of neighbors to preserve their local manifold structure (e.g., a K-D tree node corner with many close-by neighbors in the feature space or regions in high latitude require fewer neighbors), we define the local density of a node corner as the averaged L^2-Norm to its K_d neighbors.

$$d_i = \frac{1}{K_d} \sum_{j=1}^{K_d} \| \mathbf{f}_i - \mathbf{f}_{n_{ij}} \|_2, \tag{6}$$

where K_d is a constant and refers to the number of neighbors used to calculate density, and we set $K_d = 8$ in this paper; $\mathbf{f}_i$, $\mathbf{f}_{n_{ij}}$ are the feature vectors of node corners i and its j^{th} neighbor corner, respectively. Then the adaptive number of neighborhood K_i for each node corner i is defined as:

$$K_i = \cos(a \cdot \theta) \cdot \frac{d_i - d_{\min}}{d_{\max} - d_{\min}} (K_{\max} - K_{\min}) + K_{\min}, \tag{7}$$

where $d_{\min}$, $d_{\max}$ refer to the minimum and maximum distances between all neighbors respectively, and $K_{\min}$, $K_{\max}$ define the dynamic range of the number of neighbors. Since the 360° panorama is severely stretched near the polar regions in its equirectangular representation, we use $\cos(a \cdot \theta)$ as a weight to give less impact to node corners closer to the poles of the sphere. θ is the latitude, and a is used to constrain the weight change. In our experiments, we set $K_{\min} = 2$, $K_{\max} = 8$ and $a = 0.6$.

In general, minimizing Eq. (5) preserves the manifold structure of each node corner. To speedup the minimization, we propose to optimize the manifold structure by further magnifying the weights of similar node corners, while reducing the weights of nodes with large differences, which is defined as:

$$\widetilde{w}_{ij} = \frac{w_{ij} \cdot (1 + \beta \zeta_{ij})}{\sum_{j=1}^{K_i} w_{ij} \cdot (1 + \beta \zeta_{ij})}, \tag{8}$$

where

$$\zeta_{ij} = 1 - S\left(\sum_{k=0}^{2} (\mathbf{f}_i^k - \mathbf{f}_{n_{ij}}^k)^2 + f_{arc}\left(\sum_{k=3}^{5} (\mathbf{f}_i^k - \mathbf{f}_{n_{ij}}^k)^2 \right) \right).$$

$S(\cdot)$ is the **Sigmoid** function defined as $S(x) = 1/(e^{-x} + 1)$, which can smoothly map variable values from 0 to 1, and we use it to adjust the weights of neighbors of a point, where similar node corners get larger weights. ζ refers to the sum of color distance and position distance on the sphere, and we set $\beta = 0.5$ in our implementation. $\mathbf{f}_i^k$ denotes the kth component of the feature vector of the ith node corner, where $k = 0, 1, 2$ refers to the r, g, b values, and $k = 3, 4, 5$ refers to the x, y, z coordinates on the unit sphere. As shown in the right of Fig. 2, $f_{arc}(\cdot)$ maps the Euclidean distance ($\overline{P'Q'}$) to the great circle distance ($\overset{\frown}{P'Q'}$), which is defined as:

$$f_{arc}(d) = (2 \arcsin(\sqrt{d}/2))^2. \tag{9}$$

This mapping makes the adjustment of weights better correlates to the actual distance on the sphere.

3.4.3. Multiresolution speedup with downsampling and guided filtering based refinement

360° panoramas are born to be high-resolution, and typically contain tens of millions of pixels, which makes the KNN search, K-D tree construction and interpolation very time-consuming. For efficient propagation, we design a novel workflow as shown in Fig. 4. Taking an ultra high-resolution 360° panorama image as input, we first downsample it to a lower resolution, and then use the K-D tree structure to hierarchically cluster all pixels, and minimize the energy function on clustered node corners to obtain the editing map. After obtaining the editing map under low resolution, we upsample [29] it to the original size of the input panorama, and refine it using the guided filter [30], which takes the original input panorama as reference. Finally, with the refined editing map

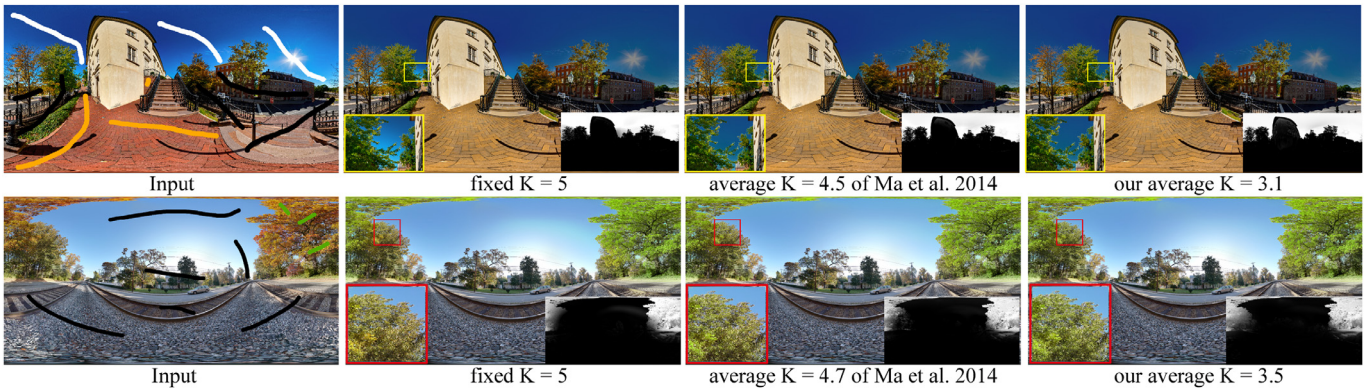

Fig. 3. Comparisons between different K-value strategies. Black strokes indicate regions to remain unchanged, white strokes indicate regions to relight, and strokes of other colors indicate the regions to edit to the same target colors. Column 1: input image with strokes. Column 2: results using fixed K. Column 3: results using adaptive K strategy of Ma and Xu [5]. Column 4: results using our adaptive K strategy. See the zoom-in views and propagation maps for detailed comparisons.

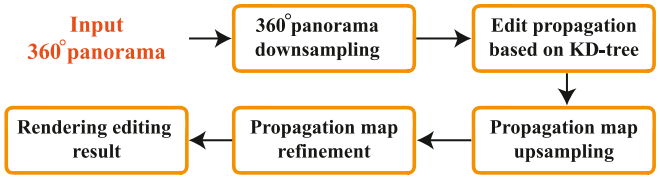

Fig. 4. Flowchart for edit propagation of HD images.

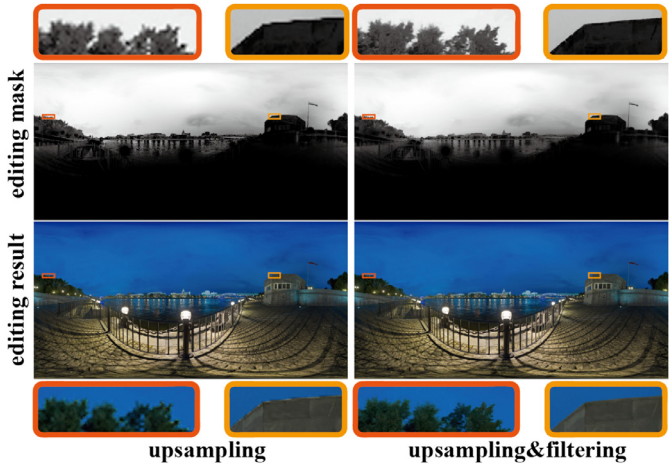

Fig. 5. Comparison of results obtained with direct upsampling and our approach using propagation map upsampling and guided filtering.

and user-specified edits, we render the final editing result. An example of the upsampled propagation map and the corresponding editing result is shown in Fig. 5. Compared with the result of direct upsampling approach, the result of our method (upsampling & filtering) is more visually pleasing with clearer boundaries between affected and unchanged regions.

4. Results

Our experiments are performed on a PC with an Intel i7-8700 3.2 GHz CPU and 32 GB RAM. We implement our method in C++, and set $\sigma_c = 0.2$, $\sigma_p = 1.0$, $K_{\min} = 2$, $K_{\max} = 8$ for all the cases. We demonstrate the effectiveness of our approach on a variety of editing tasks, including panorama relighting, panorama recoloring and panorama background replacement. We also apply these tasks to panoramic videos. Compared with previous edit propagation methods, our approach is more flexible since it supports multiple-color editing and simultaneous recoloring and relighting. In all examples of this paper, we use black strokes to indicate regions to remain unchanged, white strokes to indicate regions to relight (we will set new light values for editing), and strokes of other colors to indicate regions to recolor.

4.1. Evaluation

To evaluate our approach, we demonstrate the effectiveness of the adaptive KNN strategy, progressive propagation and compare our approach with other state-of-the-art approaches. We also evaluate the efficiency of our approach.

Adaptive K. Using Adaptive K in our approach is critical for visually consistent editing. We compare the results of our approach using fixed K, adaptive K in [5] and our adaptive K strategy in Fig. 3. We aim to darken the sky and recolor the ground in the first example, and turn the leaves green in the second example. The zoom-in views show that the fixed $K = 5$ cannot well propagate the relighting edits thoroughly and fail to propagate the green color in the left side, whereas previous work [5] and our adaptive strategy can produce better darkening and recoloring effects. Compared with [5], our strategy can generate slightly better darkening and recoloring results, and the smaller average K values help to reduce the time and memory cost.

Progressive propagation. To better demonstrate how the edit propagation works if users draw multiple strokes, we perform progressive propagation, in which we propagate edits for strokes of the same color in each step. See Fig. 6, in the first step, the ground region is painted yellow by the yellow strokes, while the sky and grass regions are incorrectly colored because there are no other constraints to keep their original color; then the top region turns to orange after drawing orange strokes; finally, the sky and grass regions get their original colors back after placing a few black strokes (which indicate maintaining their original colors). The three steps above vividly show the role of each set of strokes and their affected areas. Propagating edits progressively can guide users to place more strokes to achieve appropriate results. We also provide edit propagation maps for the orange and yellow strokes in each step.

Comparisons with state-of-the-art methods. We first compare our method with the latest state-of-the-art edit propagation method (Manifold quad-tree) [2] in Fig. 7, and our method is superior in generating seam-free results; see the close-up images. We also compare our method with other representative methods, including interpolation based method (RBF) [6], manifold preserving method (Manifold K-D tree) [5] and deep learning based method

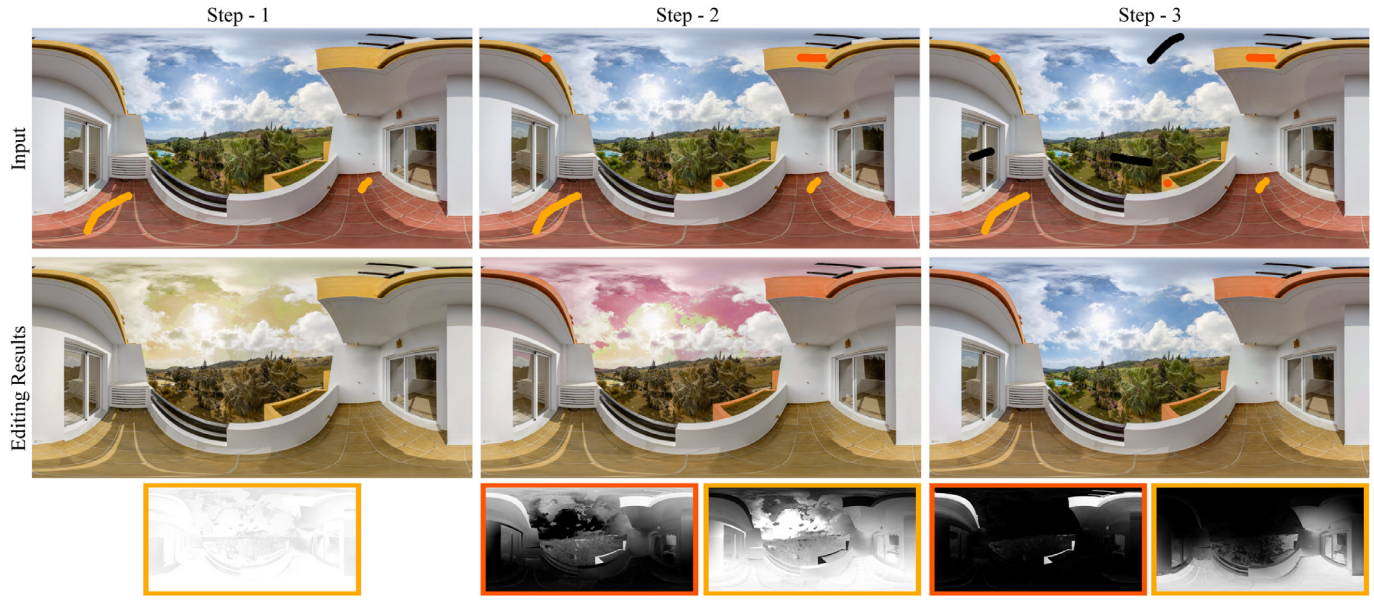

Fig. 6. Edit propagation results by drawing strokes progressively. In the first step, yellow strokes are placed to color the ground region; then, a few orange strokes are added to color the top and middle region; in the final step, black strokes are drawn to indicate pixels that need to be kept unchanged. The second row shows the edit propagation results after propagating the edits indicated by the strokes in each step. The last row gives propagation maps w.r.t. orange and yellow strokes respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

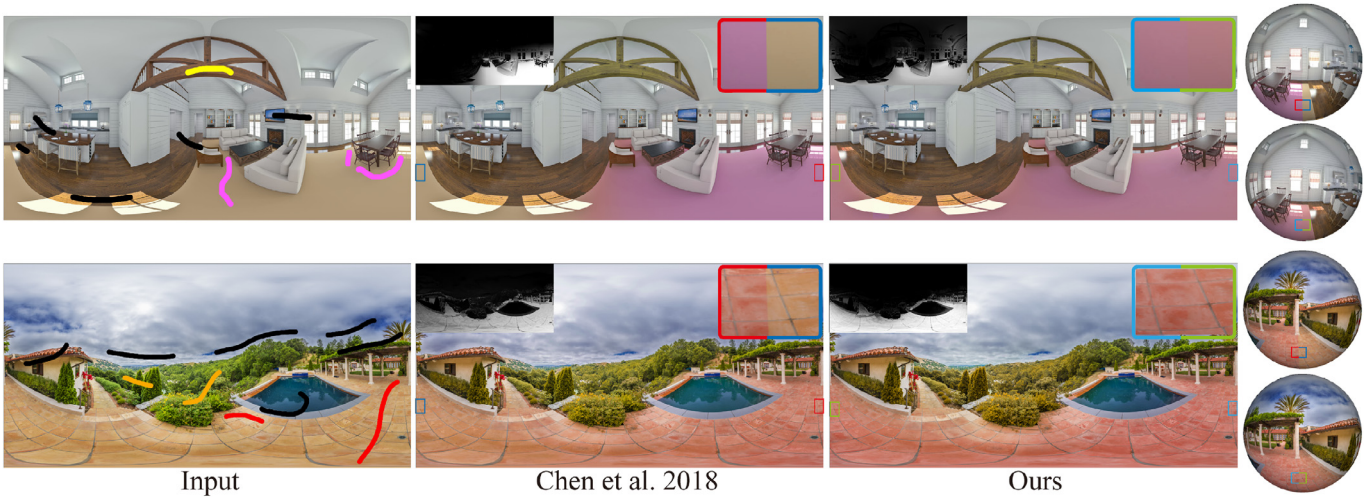

Fig. 7. Comparisons with the method in Chen et al. [2]. The strokes in the input are defined in the same way as in Fig. 3. The zoom-in windows magnify patches crossing equirectangular vertical boundaries, and the origins of the close-ups are shown in both equirectangular and spherical views using different colored rectangular windows. We also provide their propagation maps for better comparisons (see the top-left corner). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(DeepProp) [3]. Results are shown in Fig. 8. Since previous methods define neighborhoods in 2D plane, they all suffer from the "seam" problem due to the discontinuity between the left and right boundaries, and the propagation always fails to expand near the pole region; see the zoom-in views and propagation maps. As for the result of (DeepProp) [3] in the top-right, although it avoids the "seam" problem, there are many artifacts indicated by yellow arrows due to the incorrect propagation. In contrast, our method produces smoother and more natural-looking propagation results without noticeable artifacts.

For fair comparison, we also compare our results with the editing results of altered 2D edit propagation methods that consider cyclic images, see the first two rows of Fig. 9. The spherical views show that the continuity problem on the rectangular representation could be fixed by using the cyclic 2D distance metric. However, measuring the pixel distance in the 2D domain inevitably results in inaccurate distance between pixels, and the error increases from the equator to the top and bottom ends in the 2D domain. In order to correctly measure the distance between neighboring pixels, we instead consider their distance in the spherical domain. The 3D representation we use is a way that can approximate the distance on the spherical surface, and experiments show the advantages of our method over previous ones; see the marked green boxes and edit propagation maps in Fig. 9.

To investigate how the lengths of strokes affect the final editing results, we also provide additional examples using more and longer strokes which can mitigate the problem of insufficient strokes near top/bottom regions, as shown in the last two rows of Fig. 9. Obviously providing more and longer strokes improves the results of all the methods, the results of our method are still more consistent and smoother than the results of other methods; see the marked green boxes and propagation maps in Fig. 9. In addition,

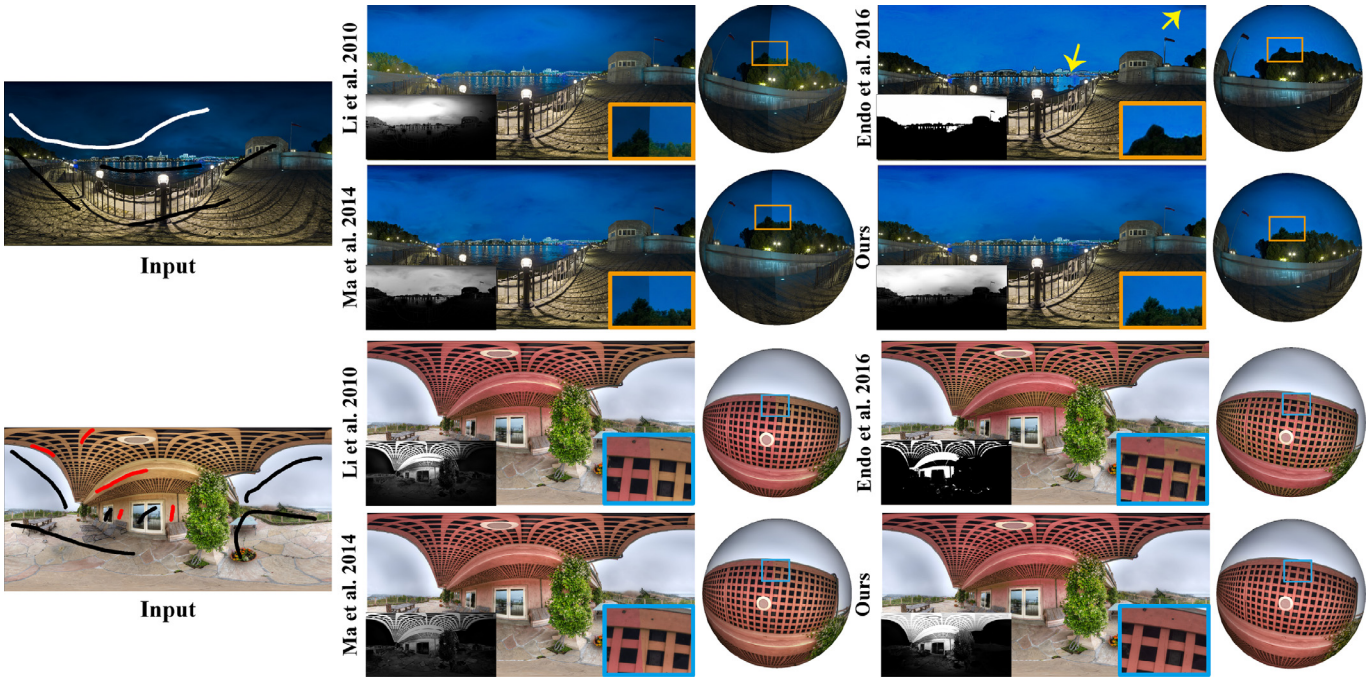

Fig. 8. Comparisons with the methods in Li et al. [6], Ma and Xu [5] and Endo et al. [3]. The strokes in the input are defined in the same way as in Fig. 3. Methods in [3,5,6] all generate pixels with inconsistent colors at the left and right border of the equirectangular image. In comparison, our method could generate smooth transition on the border region if users watch that region in the VR mode. More particularly, although the method in [3] is able to generate consistent color at the border sometimes (top-right), it cannot preserve the texture details well, and produces more artifacts (see the yellow arrows). For better comparisons, we also provide the propagation map of each result (see the bottom-left corner). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

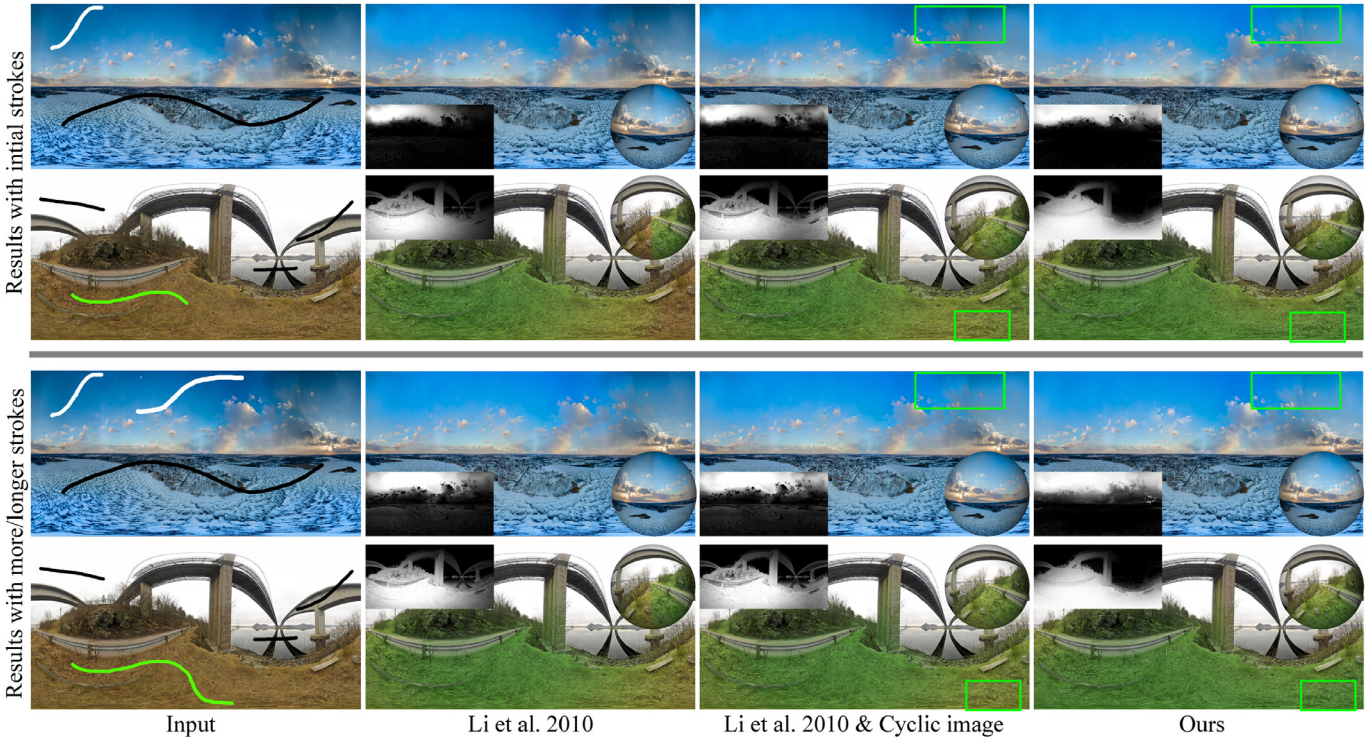

Fig. 9. Comparison with 2D edit propagation on cyclic images. The first two rows give the results using initial strokes, and the next two rows show the results using more and longer strokes. The input strokes are defined in the same way as in Fig. 3. The second column shows the edit propagation results of Li et al. [6]; the third column improves the result of the second column by considering the cyclic images; the last column is our results. For better demonstration, we also provide the propagation maps and spherical views in left and right corners. The green boxes indicate the differences between the results of performing [6] on cyclic images and our method. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

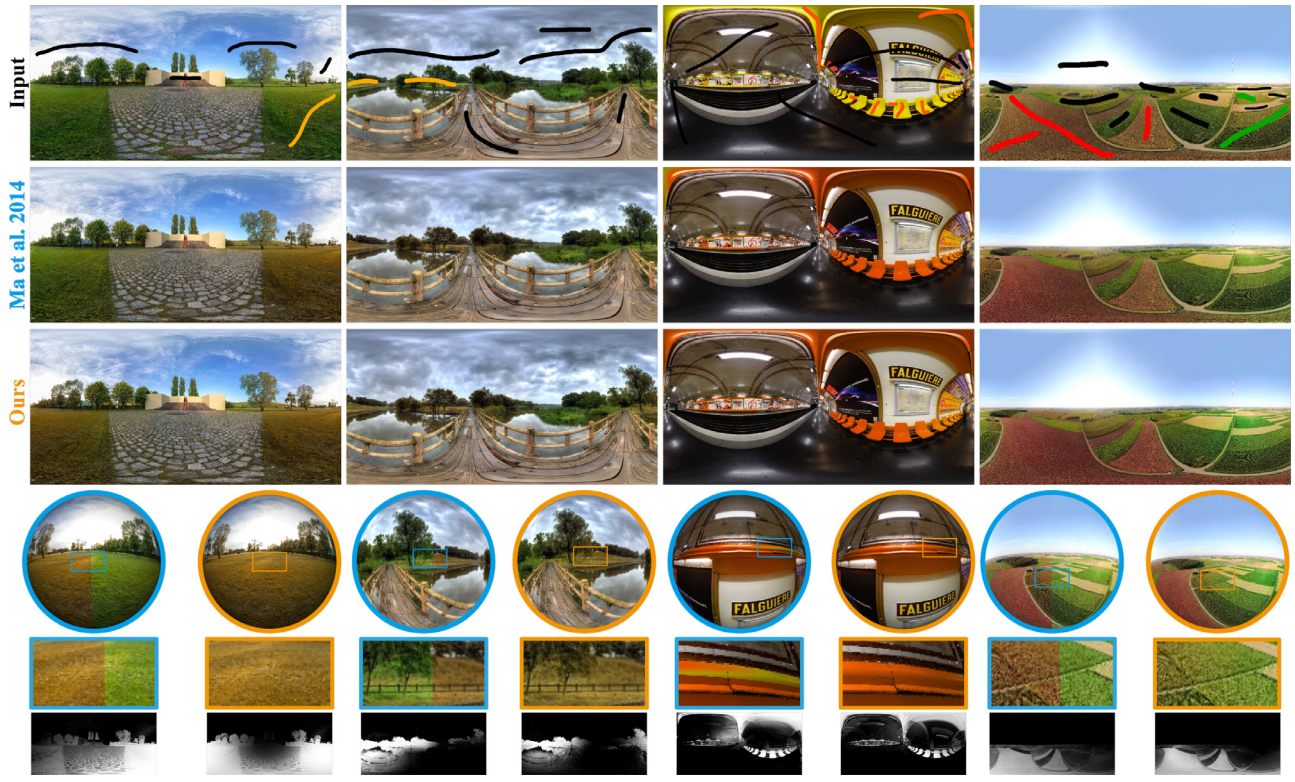

Fig. 10. More comparisons with the method of Ma and Xu [5]. Compared with [5], our method can smoothly propagate edits in the spherical domain, and can better preserve the continuity of the editing between the left and right boundaries. For better comparisons, we provide propagation maps, spherical and zoom-in views. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Performance comparisons.

	Fig. 7(1)	Fig. 7(2)	Fig. 8(1)	Fig. 8(2)	Fig. 10(1)	Fig. 13(2)	Fig. 13(3)
Type	image	image	image	image	image	video	video
Resolution	5120 × 2560	6144 × 3072	12362 × 6181	6144 × 3072	6000 × 3000	1280 × 640	1920 × 960
#. Frames	–	–	–	–	–	272	225
w/o Multiresolution	9.5s	17.7s	31.4s	15.5s	17.6s	–	–
Multiresolution							
Adaptive K [5]	2.5s	3.4s	6.9s	3.2s	3.3s	39.5s	50.9s
Our Adaptive K	1.6s	2.1s	5.8s	1.9s	1.7s	25.5s	39.6s
Li et al. [6]	1.8s	2.3s	4.9s	1.6s	1.4s	20.8s	35.7s
Ma et al. [5]	2.7s	2.5s	6.1s	2.2s	2.4s	31.7s	57.6s

our method is more efficient because fewer strokes help to reduce the complexity of optimization.

We give more comparison results between our method and previous manifold preserving edit propagation method [5] in Fig. 10. Results marked in blue and orange are produced by Ma and Xu [5] and our method, respectively. Our method performs better in these cases, where the continuity between the left and right boundaries is well preserved and the sparse edits are smoothly propagated in the spherical domain; see the spherical, zoom-in views and propagation maps of each example.

Performance. In Table 1, we give the performance of different methods on the images/videos appeared in this paper. Those include our method using adaptive K strategy in [5] and our adaptive K and weighting strategy. Given that most input 360° panoramas contain millions of pixels, we apply the K-D tree acceleration for all the cases to save memory and reduce computational cost. Take the first case of Fig. 7 as an example, our method takes a total of 1.6s, which includes the K-D tree construction(0.08s), KNN search(0.53s), energy minimization(0.39s), K-D tree interpolation(0.03s), color adjustment(0.17s) and multiresolution(0.4s). We also compare the performance of our method with and without the multiresolution strategy. Without the multiresolution strategy, a 360° panorama containing 12362 × 6181 (more than 76 Million) pixels costs 31.4s using our adaptive K and weighting strategy. With the multiresolution strategy, the adaptive K [5] costs 6.9s and our adaptive KNN strategy costs only 5.8s, which is a significant improvement over previous methods. Finally, we report the performance of previous 2D edit propagation methods [5,6]. For a fair comparison, we also apply the multiresolution strategy on the previous methods. Results show that our method is faster than [5], and achieves comparable performance with [6].

4.2. Applications

Our method enables many interesting image editing applications, and can produce visually pleasing editing results. Fig. 11 shows an application of background replacement. We let users draw *black* and *white* strokes on the input panoramic image to roughly specify the background and foreground content. Then our edit propagation method is used to calculate the possibilities of pixels belonging to background to generate a soft background mask. Such masks are then used as a guide to blend two 360°

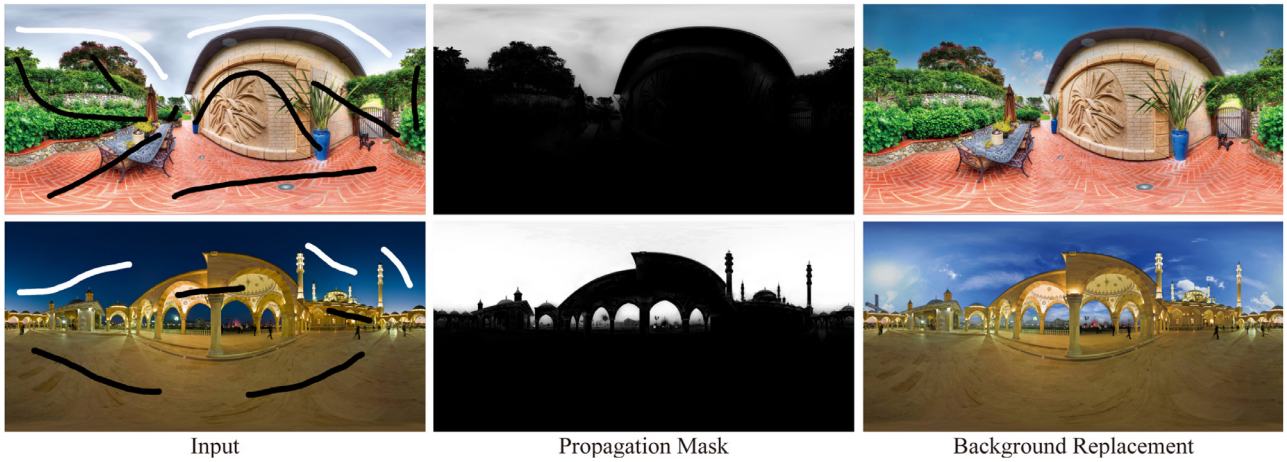

Fig. 11. Background replacement of 360° panoramas.

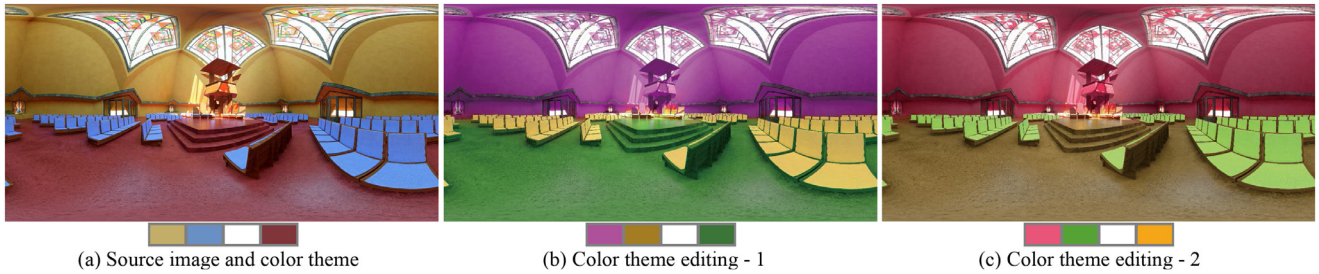

Fig. 12. Color theme adjustment for 360° panoramas.

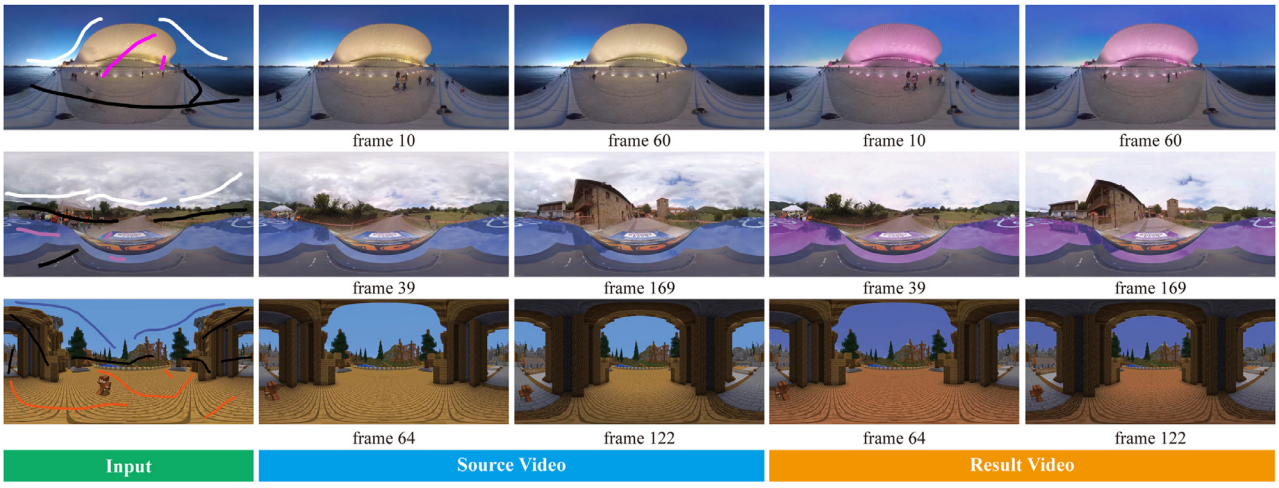

Fig. 13. Edit propagation in 360° videos.

panoramas to produce background replacement results. Fig. 12 presents another application which adjusts the color theme of 360° panoramas.

We also extend our algorithm to 360° videos. For 360° videos, we add another time dimension t, and the feature vector becomes a 7D vector $\mathbf{f}_i = (\mathbf{c}_i/\sigma_c, \mathbf{p}_i/\sigma_p, \mathbf{t}_i/\sigma_t)$, where σ_t is a parameter used to set the importance of the time dimension. We first specify edits by sparse strokes in keyframes, and feature vectors in all frames are organized together, and the video propagation problem can be solved in the image propagation framework. For better propagation across frames, we set $\sigma_t = 10.0$ for each feature vector. Fig. 13 shows propagation results in 360° videos[2]. For simple scenes we only need to specify edits in a few frames, and the edits can be smoothly propagated across all frames.

5. Conclusions

In this paper, we propose an approach to efficiently propagating sparse edits on 360° panoramas. We formulate this problem as an energy optimization, and our solution preserves the edits on strokes and the manifold structure of each pixel. To produce

[2] See the video demo of our results at https://youtu.be/sF9X-jae2dA.

seam-free and visually pleasing results, we map the 2D equirectangular pixels to the 3D coordinates, and approximate the spherical distance with the Euclidean distance in 3D space, which enables fast KNN searches in the feature space. To provide instant feedback during the interactive editing, we propose an effective solution which includes the following strategies: K-D tree construction to cluster pixels in feature space, which largely reduces unknowns in the linear equations to solve; adaptive KNN, which further simplifies the linear system of the optimization while improving the visual quality of the editing results at the same time; a multiresolution approach with a downsampling-upsampling strategy for further acceleration. Comparative results show that our proposed method is advantageous over previous methods, and can produce satisfying edit propagation results efficiently.

Our method has the following limitations: (1) since only the color and position information is used to represent the feature vector, our method is not aware of the semantic information, which may lead to incorrect color propagation in challenging cases; 2) users still need to carefully specify strokes as incorrect strokes may introduce artifacts.

In the future, we will consider adding more features to represent pixels, such as textures and semantic information, to enable high-level edit propagation in 360° panoramas. For more efficient user interaction, we will also consider more intuitive and robust ways of user guidance.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Yun Zhang: Conceptualization, Methodology, Writing - original draft, Software, Data curation, Funding acquisition. **Fang-Lue Zhang:** Conceptualization, Validation, Methodology, Writing - review & editing. **Yu-Kun Lai:** Writing - review & editing, Formal analysis, Methodology. **Zhe Zhu:** Investigation, Visualization, Writing - review & editing.

Acknowledgments

The authors would like to thank all anonymous reviewers. This work was supported by Zhejiang Province Public Welfare Technology Application Research (No. LGG19F020001), the National Natural Science Foundation of China (No. 61602402), Victoria Early Career Research Excellence Award (No. 224525) and the Royal Society (No. IES/R1/180126).

References

[1] Xu M, Li C, Zhang S, Callet PL. State-of-the-art in 360 video/image processing: perception, assessment and compression. J Sel Top Signal Process 2020;14(1):5–26.
[2] Chen Y, Zong G, Cao G, Dong J. Efficient manifold-preserving edit propagation using quad-tree data structures. Multimed Tools Appl 2018;77(6):6699–712.
[3] Endo Y, Iizuka S, Kanamori Y, Mitani J. Deepprop: extracting deep features from a single image for edit propagation. Comput Graph Forum 2016;35(2):189–201.
[4] Chen X, Li J, Zou D, Zhao Q. Learn sparse dictionaries for edit propagation. IEEE Trans Image Process 2016;25(4):1688–98.
[5] Ma L-Q, Xu K. Efficient manifold preserving edit propagation with adaptive neighborhood size. Comput Graph 2014;38:167–73.
[6] Li Y, Ju T, Hu S-M. Instant propagation of sparse edits on images and videos. Comput Graph Forum 2010;29(7):2049–54.
[7] Chen X, Zou D, Zhao Q, Tan P. Manifold preserving edit propagation. ACM Trans Graph 2012;31(6):132:1–132:7.
[8] An X, Pellacini F. Appprop: all-pairs appearance-space edit propagation. ACM Trans Graph 2008;27(3):40.
[9] Xu K, Li Y, Ju T, Hu S-M, Liu T-Q. Efficient affinity-based edit propagation using K-D tree. ACM Trans Graph 2009;28(5):118.
[10] Bie X, Huang H, Wang W. Real time edit propagation by efficient sampling. Comput Graph Forum 2011;30(7):2041–8.
[11] Yatagawa T, Yamaguchi Y. Sparse pixel sampling for appearance edit propagation. Vis Comput 2015;31(6-8):1101–11.
[12] Xiao C, Nie Y, Tang F. Efficient edit propagation using hierarchical data structure. IEEE Trans Vis Comput Graph 2011;17(8):1135–47.
[13] Ma L-Q, Xu K. Efficient antialiased edit propagation for images and videos. Comput Graph 2012;36(8):1005–12.
[14] Xu L, Yan Q, Jia J. A sparse control model for image and video editing. ACM Trans Graph 2013;32(6):197:1–197:10.
[15] Wang W, Xu P, Song Y, Hua M, Zhang M, Bie X. Distinguishing local and global edits for their simultaneous propagation in a uniform framework. IEEE Trans Image Process 2015;24(8):2478–87.
[16] Yatagawa T, Yamaguchi Y. Temporally coherent video editing using an edit propagation matrix. Comput Graph 2014;43:1–10.
[17] Oh C, Ryu S, Kim Y, Kim J, Park T, Sohn K. Sparse edit propagation for high resolution image using support vector machines. In: Proceedings of the IEEE international conference on image processing, ICIP 2015, Quebec City, QC, Canada, September 27-30, 2015. IEEE; 2015. p. 4042–6.
[18] Zhang Q, Xiao C, Sun H, Tang F. Palette-based image recoloring using color decomposition optimization. IEEE Trans Image Process 2017;26(4):1952–64.
[19] Anderson R, Gallup D, Barron JT, Kontkanen J, Snavely N, Hernández C, Agarwal S, Seitz SM. Jump: virtual reality video. ACM Trans Graph 2016;35(6).
[20] Matzen K, Cohen MF, Evans B, Kopf J, Szeliski R. Low-cost 360 stereo photography and video capture. Trans Graph (Proc SIGGRAPH) 2017;36(4). article no. 148
[21] Zhu Z, Lu J, Wang M, Zhang S, Martin RR, Liu H, Hu S. A comparative study of algorithms for realtime panoramic video blending. IEEE Trans Image Process 2018;27(6):2952–65.
[22] Zhu Z, Martin RR, Hu S-M. Panorama completion for street views. Comput Vis Media 2015;1(1):49–57.
[23] Wang M, Lyu X-Q, Li Y-J, Zhang F-L. VR content creation and exploration with deep learning: a survey. Comput Vis Media 2020;6(1):3–28.
[24] Zhao Q., Wan L., Feng W., Zhang J., Wong T.-T.. 360 panorama cloning on sphere. 2017. 1709.01638.
[25] Frossard P, Khasanova R. Graph-based classification of omnidirectional images. In: Proceedings of the IEEE international conference on computer vision workshops, ICCV workshops 2017, Venice, Italy, October 22-29, 2017. IEEE Computer Society; 2017. p. 860–9.
[26] Coors B, Condurache AP, Geiger A. Spherenet: learning spherical representations for detection and classification in omnidirectional images. In: Proceedings of the computer vision - ECCV 2018 - 15th European conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IX. In: Lecture Notes in Computer Science, vol. 11213. Springer; 2018. p. 525–41.
[27] Yang W, Qian Y, Kamarainen J-K, Cricri F, Fan L. Object detection in equirectangular panorama. In: Proceedings of the 24th international conference on pattern recognition, ICPR 2018, Beijing, China, August 20-24, 2018. IEEE Computer Society; 2018. p. 2190–5.
[28] Roweis ST, Saul LK. Nonlinear dimensionality reduction by locally linear embedding. Science 2000;290(5500):2323–6.
[29] Kopf J, Cohen MF, Lischinski D, Uyttendaele M. Joint bilateral upsampling. ACM Trans Graph 2007;26(3):96.
[30] He K, Sun J, Tang X. Guided image filtering. IEEE Trans Pattern Anal Mach Intell 2013;35(6):1397–409.