Contents lists available at ScienceDirect

Computers & Graphics

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

Interactive image manipulation using morphological trees and spline-based skeletons

Jieying Wang[a], Dennis J. Silva[a,b,*], Jiří Kosinka[a], Alexandru Telea[c], Ronaldo F. Hashimoto[b], Jos B.T.M. Roerdink[a]

[a] *Bernoulli Institute, University of Groningen, 9747 AG Groningen, the Netherlands*
[b] *Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, São Paulo-SP, 05508-090, Brazil*
[c] *Department of Information and Computing Sciences, Utrecht University, 3584 CC Utrecht, the Netherlands*

ARTICLE INFO

Article history:
Accepted September 6 2022

ABSTRACT

The ability to edit an image using intuitive commands and primitives is a desired feature for any image editing software. In this paper, we combine recent results in medial axes with the well-established morphological tree representations to develop an interactive image editing tool that provides global and local image manipulation using high-level primitives. We propose a new way to render interactive morphological trees using icicle plots and introduce different ways of manipulating spline-based medial axis transforms for grayscale and colored image editing. Different applications of the tool, such as watermark removal, image deformation, dataset augmentation for machine learning, artistic illumination manipulation, image rearrangement, and clothing design, are described and showcased on examples.

1. Introduction

Image manipulation plays a key role in image processing and computer graphics. Many image modification methods exist [1, 2, 3, 4, 5], most of which are based on raster techniques. Images represented in vector form [6] have been shown to be easier and more natural for humans to edit, mainly because vector images are represented using higher-level primitives, often controlled by arrangements of control points with an intuitive and predictable influence on the image.

Recently, Wang *et al.* [7] generated vector representations of the medial axis transform (MAT) from raster representations of input shapes and explored its potential use for binary image manipulation. While generating very interesting deformation results, it is limited to binary images and only to the most basic operations. Next, they extended the spline-based MAT [7] to the spline-based dense medial descriptors (SDMD) [8] to realize image compression of grayscale and color images. Equipped with this vector image representation, we now explore its suitability for image manipulation. To this end, we develop an experimental tool for users to interactively manipulate grayscale and color images. It exploits SDMD to reach its full potential by providing both local and global control to the user over the elements of the method. At the same time, we leverage the icicle representation of morphological trees of an image, and combine it with SDMD. The contributions of our work are as follows:

- *Novelty*: Our method is, to our knowledge, the first approach to combine morphological trees and vector representations for image manipulation;
- *Generality*: Our tool can directly handle any raster image of any resolution;
- *Interactivity*: Except for the initial encoding process, which can be pre-computed and is calculated only once, all subsequent manipulations are in real-time, which brings users instant interactivity;
- *Applications*: We demonstrate the good performance of

*Corresponding author:
e-mail: dennis@ime.usp.br (Dennis J. Silva)

our tool in a variety of applications, including watermark removal, image deformation, data augmentation for machine learning tasks, artistic effect generation, image rearrangement, and clothing design.

We start by reviewing related work (Sec. 2), which is followed by a detailed description of our image manipulation tool (Sec. 3). Then we show concrete applications of our tool (Sec. 4) and discuss its merits and limitations (Sec. 5), before concluding this paper (Sec. 6).

2. Related work

We structure related work into two groups: image manipulation methods (Sec. 2.1) and morphological tree representations (Sec. 2.2).

2.1. Image manipulation methods

Image manipulation has attracted a lot of research over the years due to its popularity and commercial importance. One such application that attracts a lot of attention is image or shape deformation, which can be roughly classified as follows.

Free-form deformation (FFD) is a popular approach for image (and shape) manipulation [1, 9, 10]. This method explicitly divides the (image) space into many domains, *e.g.*, lattices [1] and cages [11, 12], and manipulates each domain by moving control points defined in it, as illustrated in Fig. 1 (a). While allowing precise and flexible control [4, 13], setting FFD domains is tedious, requiring the user to laboriously manipulate many control vertices [3]. In addition, FFD methods do not take into account the natural way in which objects move in the real world [14, 15].

Skeleton-based approaches are also widely used for shape deformation, which using a pre-defined *skeleton* to manipulate the input shape [4, 16]. Note that this skeleton is not exactly the medial axis used in [7, 8]. Rather, it is similar to the *bones* of a character, see Fig. 1 (b). The typical workflow of skeleton-based methods is to bind the components of the character to be edited to a pre-defined skeleton such that each component follows the motions of its associated bones. Skeleton-driven approaches are also commonly used in the deformation of 3D shapes [17, 18, 19]. While offering intuitive control of 2D or 3D shapes, binding a shape to a skeleton, either manually or automatically, is not a trivial task [17], especially for shapes lacking an obvious bone-and-joint structure, *e.g.*, jellies [3], to mention just one salient example.

Physics-based methods [3, 14, 20, 21] can be regarded as variants of detail-preserving differential mesh deformation techniques [22], which deform shapes by modeling their *rigidity*. These methods allow the user to directly manipulate a shape through a click-and-drag interface, as shown in Fig. 1 (c), and generate physically natural results by minimizing local shape distortion. However, such methods are computationally expensive, resulting in slow convergence, and require careful tuning of several parameters [14].

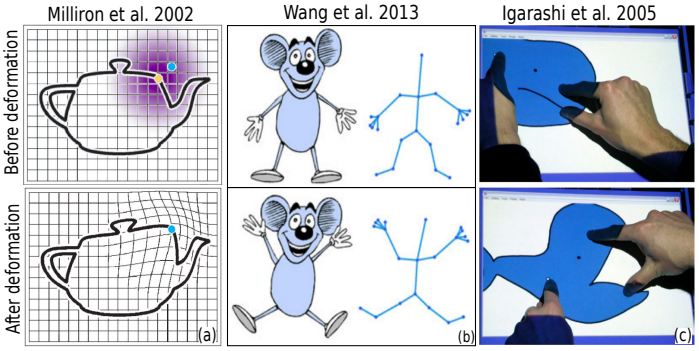

Fig. 1. Illustrative images of three shape deformation techniques. (a) A free-form deformation example taken from [1]. (b) Skeleton-based image manipulation in [4]. (c) A physically-based approach example taken from [3].

Image deformation techniques, as described above, are most suitable for images with sharply delineated and simple shapes. For more complex images, additional image manipulation applications have been investigated. Pérez *et al.* proposed Poisson Image Editing [2], a gradient-based image manipulation method, which is a simple and efficient way for many operations, such as seamless cloning, contrast enhancement, texture flattening, and local illumination/color changes. Since then, numerous applications have exploited the benefits of working in the gradient domain. Raskar [23] presented a class of image fusion techniques to automatically combine images of a scene captured under different illuminations. Levin [24] proposed a technique for image stitching which combines several individual images that have some overlapped regions. Sun [25] formulated the problem of natural image matting as one of solving Poisson equations with the matte gradient field. Finally, Aris [26] proposed a general variational framework for non-local image inpainting. More related work can be found in [27].

In recent years, deep learning-based methods have significantly boosted the performance of image manipulation due to the availability of large amounts of data that one can train on [5, 28, 29, 30]. These methods mainly focus on a task called *image-to-image translation*, which aims to convert a specific aspect of a given image into another, ranging from changing the facial expression [5] or hair color [29] of a person to modifying the seasons of scenery images [30]. While yielding amazing image manipulation results, these methods require a significant number of labeled image pairs. To avoid this, Vinker [31] introduced a novel method for training deep conditional generative models from a *single* image. After training, this method is able to perform challenging image manipulation tasks by modifying the primitive representation. However, this approach requires training a separate network for every image, which can be expensive on large datasets. Furthermore, deep learning-based methods generally do not have a convenient interface for user-interactive operation.

In this paper, we propose an interactive image manipulation method that differs from all the previously described techniques. We integrate two novel works: the icicle representation for morphological trees (described next) and the SDMD [8] used for image representation.

2.2. Morphological tree representation

As is well known, medial descriptors, or skeletons can only be computed for binary shapes. Thus, in order to be able to represent a grayscale image I with skeletons, we decompose I in n (256 for 8-bit images) binary images (called level sets) by upper thresholding $T_i^{\uparrow} = \{\mathbf{x} \in I \,|\, I(\mathbf{x}) \geq i\}, 0 \leq i < n$ or lower thresholding $T_{\downarrow}^{i} = \{\mathbf{x} \in I \,|\, I(\mathbf{x}) \leq i\}, 0 \leq i < n$. This works efficiently for image compression tasks [7, 8]. Yet, when it comes to image manipulation, finer-grained spatial control of each level set is required. Figure 2 shows an example. The synthetic image (a) considered in the figure contains nested triangles and nested discs; (b) then shows its four upper-level sets. When one wants to remove, rotate, scale, or move those triangles, it is inconvenient to manipulate them *individually*. To conquer this, we propose to use the morphological tree representation [32, 33, 34], which represents hierarchically all connected components of an image. Thus, a morphological tree is a *complete* representation of an image.

The most common morphological tree representations contain trees of shapes [35] and component trees [36]. The latter are usually represented by compact and non-redundant data structures called max-trees $(\mathcal{U}(I), \subseteq)$ and min-trees $(\mathcal{L}(I), \subseteq)$. The sets $\mathcal{U}(I)$ and $\mathcal{L}(I)$ are composed of the connected components (CCs) of $T_i^{\uparrow}$ and $T_{\downarrow}^{i}$, respectively, *i.e.*,

$$\mathcal{U}(I) = \{C \in CC(T_i^{\uparrow}(I)) : i \in [0, n)\}$$

and

$$\mathcal{L}(I) = \{C \in CC(T_{\downarrow}^{i}(I)) : i \in [0, n)\},$$

where $CC(T_i)$ denotes the sets of either 4- or 8-connected components of the threshold sets T_i. The max-tree representation of Fig. 2 (a) is shown in (c). From that, one can either process each triangle shape individually, or, alternatively, all triangles collectively by selecting all descendant nodes of node E. Component trees can be computed and processed efficiently [37, 38, 39], which is widely used in object recognition [39], 3D segmentation [40], and remote sensing [41].

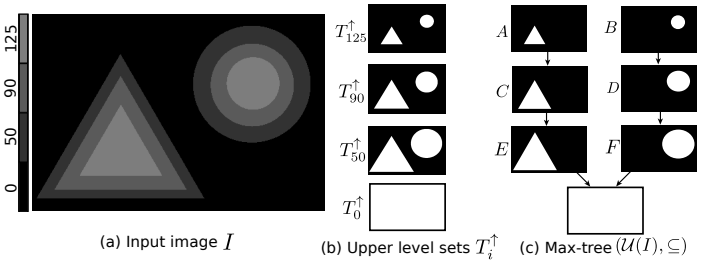

Fig. 2. A synthetic image (a), its 4 upper-level sets (b), and the corresponding max-tree with 7 nodes, *i.e.*, connected components (c).

To interactively manipulate a grayscale image with component trees, many visualization tools have been proposed [42, 43]. In such tools, the user either sets parameters for the manipulation task or selects regions in the input image. Next, the tool shows interactively the filtering or segmentation results. However, since max-trees of natural images have tens of thousands of nodes (see the example in Fig. 3), the user only interacts with the image and parameters, and not directly with the max-tree.

Fig. 3. Lena image (512×512 pixels) and its complete max-tree, which contains approximately 41000 nodes. Image taken from [44].

To simplify the structure of component trees, Tavares [44] proposed a simplification procedure based on two attributes: extinction value [45] and the area of nodes. They further improved the simplification by applying an area difference filter, yielding a more meaningful graphical representation of component trees [46], as shown in Fig. 4. However, the simplified tree is no longer a complete representation of the original image. To alleviate this, we next propose to apply a new representation of the component trees: icicle plots [47] in Sec. 3.1. Icicle plots not only contain all the information of the original image, but they are also more compact and more organized.

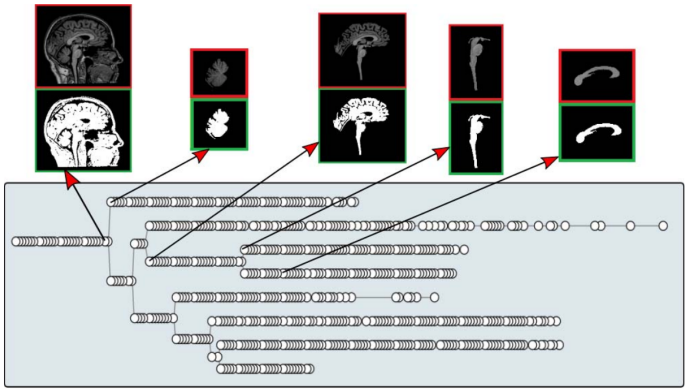

Fig. 4. Interactive max-tree of a brain scan image and some node samples with their respective connected component. Image taken from [46].

3. Proposed method

As stated in Sec. 2.1, our proposed method combines two novel works: an icicle representation for component trees and an interactive spline manipulation. Figure 5 demonstrates the pipeline of our proposed method. Given a grayscale image, we first compute its max-tree or min-tree, which is next represented in an icicle plot (Sec. 3.1). All the nodes in the icicle plot are associated with their corresponding spline control points (step 1). Next, we allow users to select single or multiple nodes for subsequent manipulation (step 2). Section 3.2 describes several methods for node selection. The associated connected components and control points of the selected nodes are displayed for

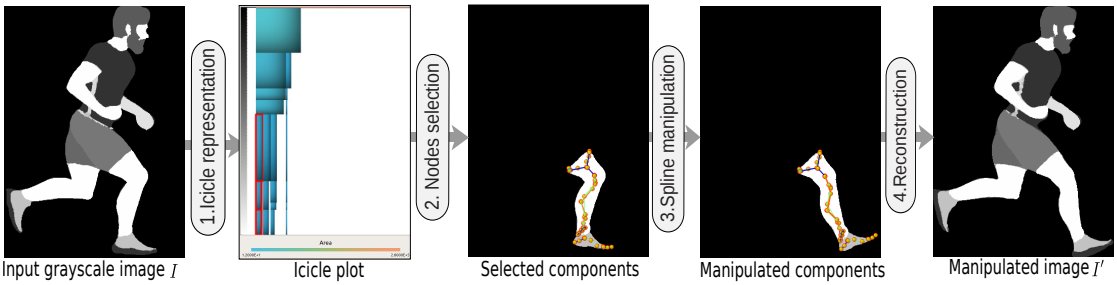

Fig. 5. Interactive image manipulation pipeline.

interactive spline deformation (step 3), which is described in detail in Sec. 3.3. Finally, the manipulated image is reconstructed (step 4).

3.1. Icicle plot representation

Icicle plots [47, 48, 49], also called icicle trees, represent hierarchical data in the form of stacked rectangles, usually ordered from top to bottom, following the order of nodes in a tree from its root to its leaves. Compared with other representations, such as node-link visualizations, icicle plots allow an easier reading of the nesting relationships, the areas of the nodes, and various attributes of the nodes, such as, in our case, the grayscale of the encoded objects, their perimeter, circularity, complexity, or the number of skeleton points or spline control points. As such, we choose the icicle plot metaphor to represent the hierarchical component trees of grayscale images.

Figure 6 shows the icicle plot of the max-tree for the synthetic image in Fig. 2. The selected image is on purpose simple, for illustration purposes. Each icicle, or node, corresponds to a connected component (CC) in the upper-level sets of the input image. As visible, the brightest disc (filled with red) in the original image corresponds to the node marked by the red box in the lower right corner of the icicle tree. The slender orange rectangle at the top is the root node, which takes up the entire width. Each child node is placed under its parent with the width proportional to the *area* of the component. The grayscale that each node reaches down to is exactly the gray value of the selected CC. The height spanned by each node on the grayscale bar is the gray level difference between the level of the selected CC and the previous level. The fill color of each node can be coded by various attributes, *e.g.*, the number of skeleton points, as shown in Fig. 6. Other attributes, including area, perimeter, circularity, and complexity, are also implemented in the tool and available via its user interface.

To decrease the number of pixels needed to render a node of the morphological tree and to keep the visual separation of the nodes, we draw the rectangles of the plot with no border and apply a shading scheme. In this approach, each rectangle is a *quad* primitive with a HSV base color associated which is tessellated by the graphics hardware. During the tessellation process, we compute a luminance profile which replaces the *value* (V from HSV) channel of the HSV color. To obtain a good visual separation of the node, we apply an asymmetrical cushion-like [50, 51] luminance profile. This profile is obtained by computing two 1D cubic Bézier curves: one that is sampled

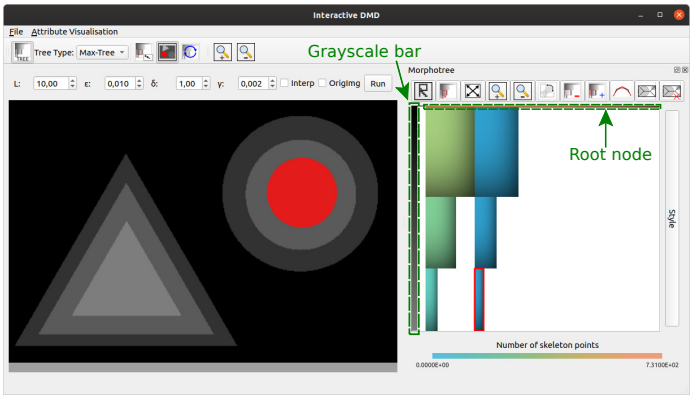

Fig. 6. The synthetic image in Fig. 2 and its max-tree in icicle representation. One component/node has been selected (red).

vertically and another horizontally in the quad tessellation process. Then the final luminance of the pixel is obtained by multiplying the samples of the Bézier curves at the parametric (u, v) coordinates produced by the tessellation shader. Fig. 7 graphically shows the process of obtaining the luminance profile from the vertical and horizontal curves (or profiles).

Using this approach, by default, we set the control points to $[0.9, 0.9, 0.9, 0.45]$ for both curves, which results in a high value of luminance at the top left, gradually reducing towards the bottom right. This shading produces a dark bottom-right region that meets the bright top-left regions of its neighboring rectangles, leading to a visual separation between the nodes. An example of tree rendering using these parameters can be found in Fig. 7. Since we modify the color luminance for shading, the rendering uses an iso-luminant color map [52, 53] (see Fig. 7, bottom right). Although icicle plots are compact and can display hundreds of nodes, depending on the area of the nodes and window size, the nodes might be displayed too small. To address this issue, we developed *zoom in*, *zoom out*, and *zoom restore* functionalities. For complex images, the user can explore the connected components of the image by clicking around the nodes of the tree, in the pixels of the image, or moving the selection to parent (by pressing the key *P*) and zoom in to explore related connected components of small details of the image. When the tree is zoomed in on, the user can hold the *Alt* key for panning around the tree. When the small details editing is finished, the user can bring back the full tree visualisation by using the zoom-restore functionality. An example of this

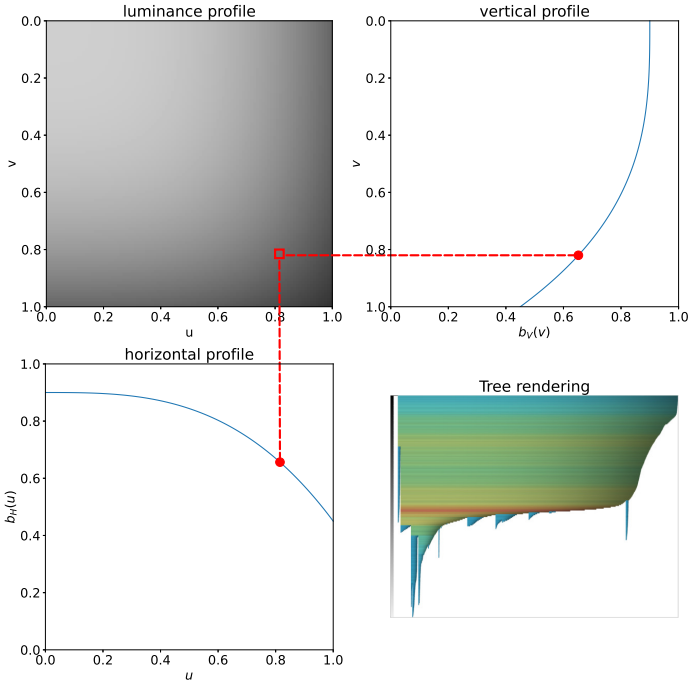

Fig. 7. Our luminance profile scheme. Each node of the tree is rendered as a quad primitive which is tessellated by the graphics hardware. The luminance of a sampled pixel at parameters (u, v) is computed by multiplying $b_V(u)$ and $b_H(v)$, where b_V and b_H are $1D$ Bézier curves with control points $V = [0.9, 0.9, 0.9, 0.45]$ and $H = [0.9, 0.9, 0.9, 0.45]$, respectively. A tree with hundreds of nodes rendered using this method is displayed at the bottom right of the figure. In this rendering, the base color is assigned following an iso-luminant color map of the node perimeters.

process is shown in a video recording in the supplementary material [54] and a zoomed in tree is depicted in Figure 8.

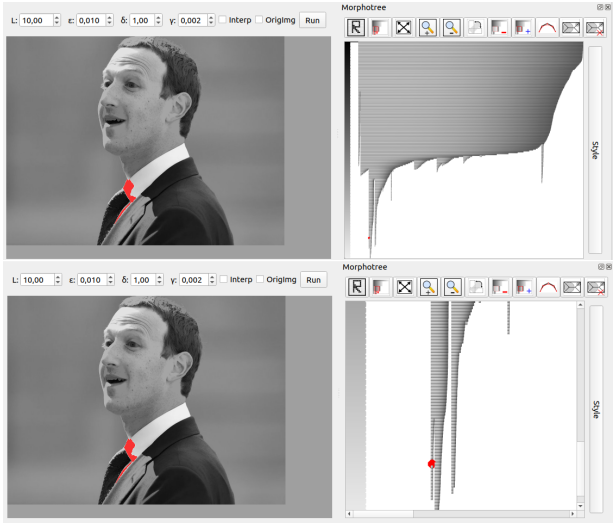

Fig. 8. A grayscale image and its morphological tree. (top) a node selected in the full tree, (bottom) zoom-in at the selected node.

To sum up, icicle representations clearly show the nesting relationship, the size, the gray level, and other custom attributes of connected components of an input image. In addition, they depict tree nodes using only a few pixels and yet keep a visual separation of neighboring nodes using a shading approach. Having this compact and well-organized representation, we are now ready to perform image manipulation. The proposed interactive image editing tool is the combination of a high-level global manipulation and a more detailed deformation of local components.

3.2. Global manipulation

Global manipulation, also considered to be *inter-node manipulation*, generally includes removing or restoring single or multiple nodes or CCs, which is useful for applications such as image segmentation, local luminance changes, and watermark removal (Sec. 4.1). Node selection can be implemented manually or algorithmically, as described next.

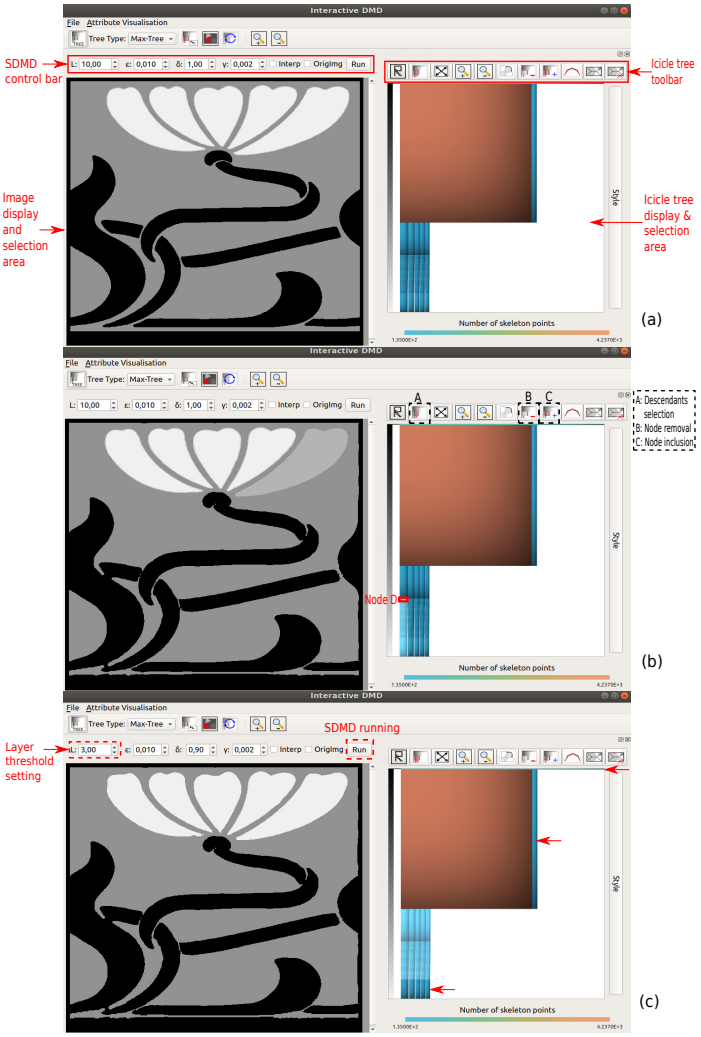

Fig. 9. Global manipulation of an art deco image. (a) The original image and its icicle tree. (b) The result of removing node D and all its descendants (corresponding to the rightmost petal). (c) The result of retaining the three most representative level sets.

Manual selection refers to the user directly selecting the node or component that one wants to cut out or restore in the interface. Since the image components in the left window and the nodes in the right window are associated, one can select the part one wants to manipulate by either directly clicking the node in

the icicle tree or the component on the image. In both ways, one can select and deselect multiple nodes by holding down the *shift* key. When a set of nodes are selected, their connected components are painted in red on the image panel, so the user can quickly see the region they are about to edit. The clicking-and-selection operation is straightforward and convenient. Yet, this operation can be cumbersome when there are plenty of nodes to be manipulated. To address this, we added a function to select all descendant nodes of the currently selected node by clicking icon A. Figure 9 (b) illustrates this by manipulating a simple art deco image (a). Deleted nodes are set to translucent. As visible, by discarding all descendant nodes of node D, their corresponding components (the rightmost petals in multiple levels) on the image are also eliminated, which indirectly achieves the effect of local brightness changes. The operation of restoring a node is similar to deleting one. By selecting a deleted (translucent) node and then clicking the node inclusion icon (C), one can restore the node. However, one cannot add nodes that did not exist in the original tree.

Algorithm-based selection aims to set the number of layers L in the parameters setting area, and then run the SDMD method, so that the method will select and retain the most representative L layers using the *cumulative histogram* layer selection scheme. The program constantly checks whether $\mathcal{A}(T_i^{\uparrow}) - \mathcal{A}(T_{j=i+m}^{\uparrow}) > \lambda$, where $\mathcal{A}(T_i^{\uparrow})$ means the *area* or the number of pixels of $T_i^{\uparrow}$, and m is the minimally-perceivable luminance difference to the human eye (set empirically to 5 [55] on a luminance range of $[0, 255]$). If the difference between $\mathcal{A}(T_i^{\uparrow})$ and $\mathcal{A}(T_{j=i+m}^{\uparrow})$ is smaller than λ, we increase j until the inequality is satisfied. At that point, we select layer $T_j^{\uparrow}$ and repeat the process until we reach the last layer. To set a suitable λ, we do a L-to-λ conversion by the binary search method; see details in [56]. Figure 9 (c) shows the selection result when L is set to 3. The original image contains tens of level sets or layers. Although not easily visible, there are various grayscale values at the edges of the petals. By setting L to 3, the method selects the three most informative layers, as indicated by the red arrows. As can be seen from the results in the left window, almost no important information is removed. The algorithm-based selection can only preserve or remove *all* nodes of a certain layer. Still, in combination with the manual operations described above, more refined global manipulation can be achieved.

Global manipulation is useful for image segmentation [57, 58, 59]. Figure 10 illustrates a skull-stripping segmentation [60, 61] by showing several components (A, A1, A2, B, C) of a magnetic resonance (MR) image (Fig. 10 (a)) reconstructed from several nodes and their corresponding descendants. As visible, the whole brain (A), including the brain stem (A1) and cerebellum (A2), as well as the parotid tissue (B) and nasal tissue (C), are successfully segmented. Moreover, our proposed method is fast and does not require any preprocessing such as intensity normalization or denoising.

3.3. Local manipulation

Local manipulation is also seen as *per-node deformation* and mainly refers to deforming a single connected component by manipulating its spline control points (CPs). Literature [7] has

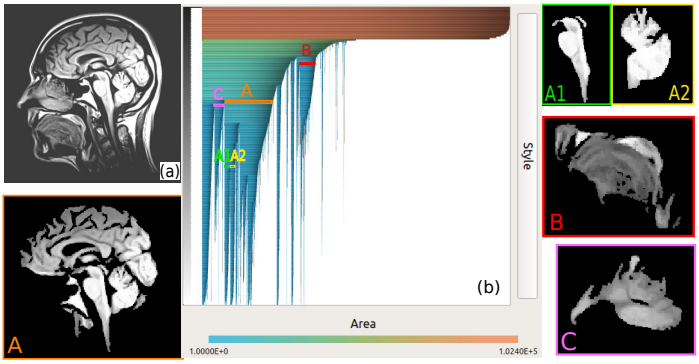

Fig. 10. Skull-stripping segmentation. An MR image (a), its icicle tree (b), and several components (A, A1, A2, B, C) of the image.

illustrated several preliminary operations, including moving, adding, and removing CPs and increasing or decreasing the *degree* of the spline representing the medial axis transform. In this section, we further expand this idea by presenting more functions. We use the node D in Fig. 9 (a) as an example to introduce our user interface. By selecting node D and then clicking the icon to the right of icon C, we open the user manipulation interface, as shown in Fig. 11. We next introduce one by one all the tools we propose for this task.

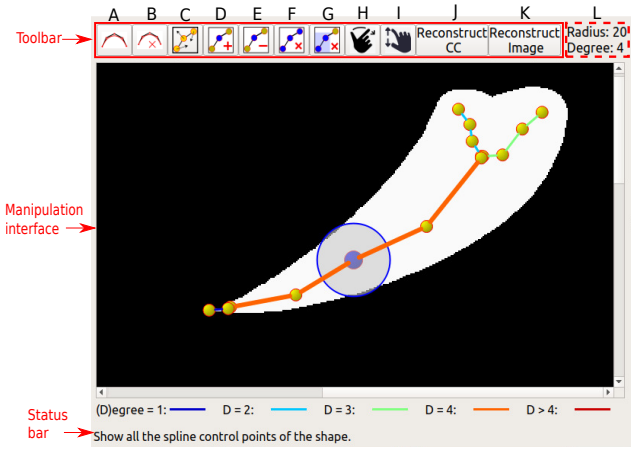

Fig. 11. User interface for detailed spline manipulation.

Displaying all CPs: By clicking icon A, all CPs of the current component are shown in the manipulation interface. Each component has one or several skeleton branches, thus resulting in one or more splines. Control points (CPs) on the same spline are connected by lines of the same color, which indicates the degree of the spline, as shown at the bottom of the interface. In contrast to icon A, the function of icon B is to make all CPs invisible.

Changing the radius/degree: When the mouse hovers over a CP, its radius size is displayed, and the radius value is also updated in the L area. When a point is clicked, it is highlighted in blue. Then, when holding down the *shift* key and scrolling the mouse wheel, the radius (both the graphical representation and the actual value) changes accordingly. The operation of modifying the degree is similar, except that the *shift* key has to

be replaced with the *D* key.

Adding a CP to the spline: Icon D allows users to add a CP to the spline. Note that the clicked point needs to fall in the (invisible) rectangle formed by any two consecutive CPs of the spline. Otherwise, a new spline (with two CPs) will be created upon such a click.

Removing CPs in a spline: The user is allowed to remove one or more CPs in a spline by pressing icons E or G. One can also delete the entire spline via icon F.

Rotating/scaling CPs: Icon H is used for rotating all selected CPs. After clicking this icon, one first needs to select a rotation center, then select the CPs to be processed by dragging the displayed rubber band marker with the mouse. Next, one can hold down the *R* key and scroll the mouse wheel to specify the desired rotation angle. The scaling function is similar, except that icon H and the *R* key have to be replaced with icon I and the *S* key.

Copying/cutting CPs: These two functions are similar. First, the user can select one or more CPs, then press the *C/X* key, then click somewhere for the CP(s) to be pasted and press the *V* key to effectuate the actual CP pasting.

Reconstruction: Icons J and K are used to reconstruct the manipulated component and the whole image, respectively. The changed splines are first rasterized on the desired pixel grid to generate the manipulated skeletons. Then, we reconstruct the component with the medial discs envelope method [8].

Figure 12 shows the manipulation of a shadow puppet character, in which most of the above operations are covered, including deleting CPs ($-c$), decreasing radius values ($-r$), and moving (M), rotating (R), scaling (S), and copying (C) control points. We start by making the figure head smaller by pressing icon I, next selecting all the CPs that represent the head, and then scrolling the mouse wheel down to adjust them to the appropriate size. Then we *move* all CPs down slightly to make the result more realistic. For the left arm, we intend to separate the hand from the body. For this, we first delete the selected spline A (with 5 CPs) in Fig. 12 (a), then rotate the arm clockwise by about 30 degrees, and next copy the right hand into the left side and rotate it by the suitable angle. We also decrease the radius of both CPs of the spline at the elbow by 8 pixels. In addition, the left leg and right arm of the character are also rotated by about 20 degrees clockwise and counterclockwise, respectively.

4. Applications

In the previous section, we introduced our proposed interactive image editing tool. Section 3.2 introduced several schemes for selecting multiple icicle nodes. Section 3.3 demonstrated the internal manipulation of a single node. Combining the two, *i.e.* to move, scale, rotate, remove, and paste multiple nodes at once, we enable more interesting and powerful applications of our method, as detailed next.

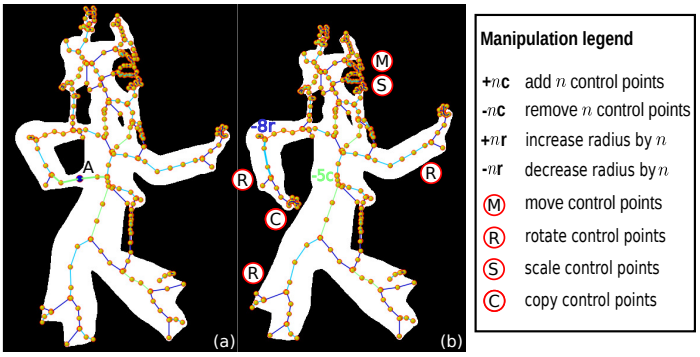

Fig. 12. Deformation of a shadow puppet character (a) by manipulating the spline control points (b). Manipulations are colored just as the spline they affect.

4.1. Visible watermark removal

Visible watermarks are widely used in images and videos to protect copyright ownership. Analyzing watermark removal helps to strengthen the anti-attack techniques in an adversarial way, which attracted increasing attention and became a hot research topic [62, 63, 64, 65]. Due to the uncertainty of the size, shape, color, transparency, and location of watermarks, developing an *automatic* visible watermark removal method remains a difficult task. Some techniques even require user-guidance [62, 63] or assume that test images have the same watermark region [66]. Our image manipulation tool provides a way to remove watermarks, but in an *interactive* way, rather than automatically like the methods mentioned above. Our proposed method is very simple. We first select watermark-related nodes through the several schemes introduced in Sec. 3.2. Then we enter local manipulation (Sec. 3.3), and press icon E in Fig. 11 to delete all control points associated to the watermark. Two manipulation demonstrations are available in the supplementary material [54].

Figure 13 shows the results of our method on six watermarked images. As can be seen from the three grayscale images, our method works well not only for images where the embedded watermark is brighter than the surrounding area (Fig. 13 (b1, f1)), which can be easily manipulated with the selection-and-deletion scheme described in Sec. 3.2, but also for images where the embedded watermark has a similar or lower intensity to the surrounding area (Fig. 13 (d1)). Our proposed tool also yields good results for color images (a2, c2, e2) by manipulating their three components, *e.g.* YUV, independently. However, since the manipulated image is reconstructed from the skeletons, this watermark removal method also has the common drawbacks of skeleton-based image representation methods [8, 56, 67], *i.e.* it cannot deal well with images with many thin and small-scale details, such as plants on the mountains (b2, e2) and animal fur (c2, d2). Yet, the skeleton-based method is good at processing images with relatively large shapes but thin watermark shapes, such as shown in Fig. 13 (a1). For such images, the SDMD method with suitable parameters (by setting the SDMD control bar in Fig. 9) inherently removes those thin watermark patterns even without using global or local manipulation. This process takes only a few seconds (see the demon-

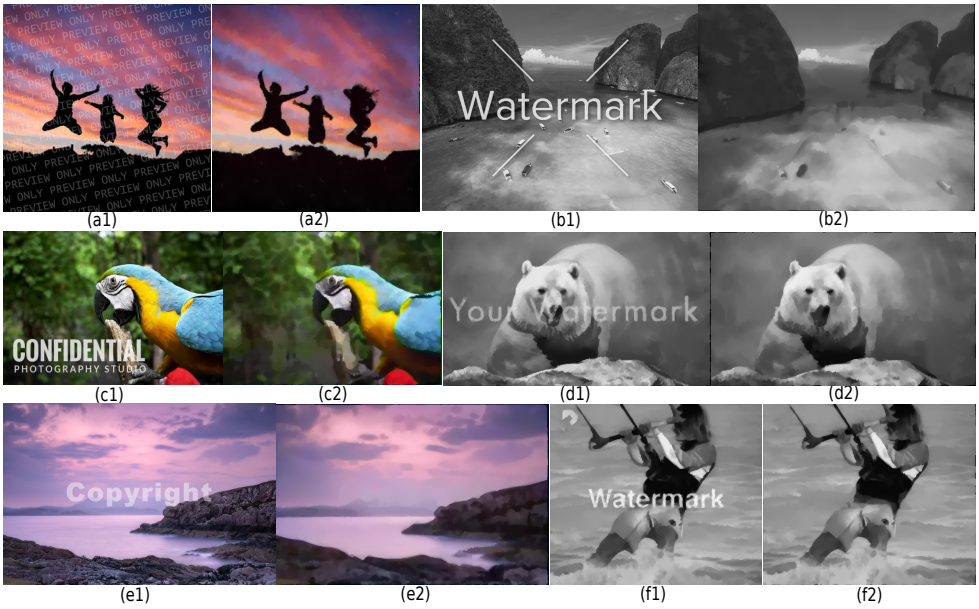

Fig. 13. Example results on watermark removal. (a1)–(f1) Watermarked images. (a2)–(f2) The results of removing the watermarks with our method.

stration in the supplementary material), rather than several minutes as with the GIMP tool (see details in Sec. 5.1).

4.2. Image deformation

The previous section has shown how to achieve good watermark removal performance by combining the multiple node selection in the global manipulation (Sec. 3.2) with the CPs deletion in the local manipulation (Sec. 3.3). In this section, we combine the node selection with deletion and addition of CPs in local manipulation to implement *image deformation*.

Figure 14 illustrates an example by showing several steps to remove glasses from a cartoon avatar. We first remove the glasses' lenses (step 1) by deleting nodes A, B, and C in Fig. 14 (b2) by pressing icon B in Fig. 9 (a). Then we eliminate the glasses' frame (step 2) by selecting node D and all its descendants in (b2), entering the local manipulation interface (c2), and deleting all CPs related to the frame shape (region E in c2). Step 2 produces a very light eye contour (see (c1)), so in the next step, we aim to darken the eye outline (step 3). We select only node D in (b2) and enter the user interface (d2). Then we use the CP adding function (icon C in Fig. 9 (a)) to put in several splines and manipulate their CPs to form the eye contour, as shown in region F in (d2). Now we successfully remove the glasses from the original image (a1) and generate a reasonable result (d1).

Following the same idea, we further generate four other facial changes, as shown in Fig. 15 (a1–a4). To generate (a1), we first remove all CPs related to the glasses and the eyes. Then we add four splines to represent the smiling eyes. The remaining ones use similar operations. We first delete control points that represent the smiling mouth, then we add the new mouth (a2), mustache (a3), and beard (a4) in turn by adding new splines.

Figure 15 (b1–b4) shows the manipulations of a more complex running horse image. To generate (b2), we first rotate all the CPs representing the horse counterclockwise by about 30 degrees, then rotate the two hind legs of the horse clockwise by about 35 degrees. Next, we rotate the fore cannon bones clockwise by certain angles to achieve the bending of the forelegs. The demonstration is available in the supplementary material [54]. We manipulate the remaining two examples (b3, b4) similarly, mainly by rotating and moving the control points representing the front legs.

4.3. Dataset augmentation

Many machine learning setups require a high number of samples to avoid overfitting and to increase their performance. However, acquiring annotated samples for a dataset is usually hard and costly [68]. Thus, producing new samples by applying transformations to existing samples in a dataset using *data augmentation* techniques has been used to address these issues [68, 69, 70]. We show an example of using our tool to automatically augment a handwritten digit dataset by randomly applying small changes to the samples. The general idea is simple: we take an annotated sample (in this case we already know its class), and then produce different versions of this sample by randomly generating small changes on the control points of different threshold levels.

As a proof of concept, we apply this method to augment a subset of the MNIST dataset [71] using our tool's functionality to generate images by applying random changes; this can be accessed by clicking on Icon C (Fig. 11). In this experiment, we created a dataset called 50-MNIST by randomly choosing 50 samples for each digit of the MNIST dataset and training an SVM on it. After that, we augmented the dataset. We scaled up the images 10 times to allow our tool to successfully produce relevant skeletons that can be manipulated. For each image, we ran the SDMD pipeline, keeping only the 10 most relevant threshold values, selected all nodes other than the root node (to have all details of the digit selected), and then for each control point of each selected node we applied a sequence of transfor-

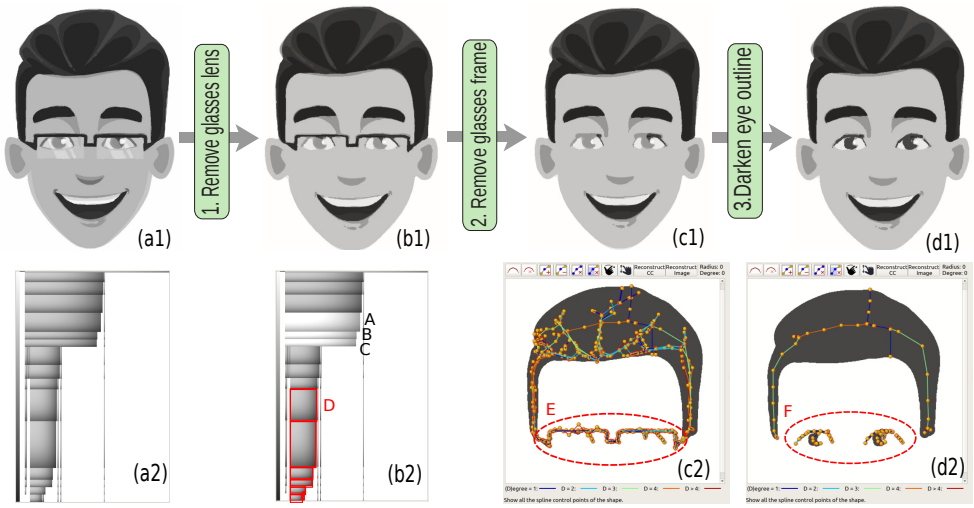

Fig. 14. Key steps for removing glasses from a cartoon avatar (a1). Images (a2) and (b2) show icicle trees of (a1) and (b1), respectively. Image (c2) shows the corresponding components and control points of node D and its descendants, while (d2) shows those of node D only.

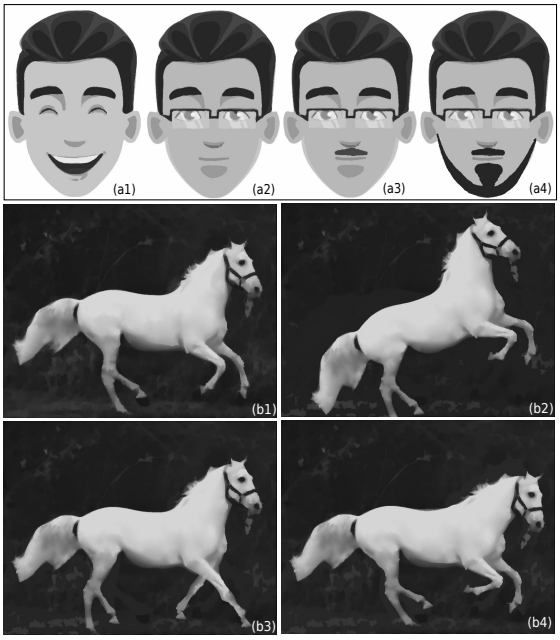

Fig. 15. Image deformation examples. (a1–a4) Four additional facial variations resulting from the manipulation of the original image in Fig. 14 (a1). (b2–b4) Deformations of a running horse image (b1).

mations: displacement, radius scale, rotation, and scale. For each transformation, we define a range of values as follows:

1. Displacement: $\texttt{min}_{\texttt{dis}} = -15$ and $\texttt{max}_{\texttt{dis}} = 15$.
2. Radius scale: $\texttt{min}_{\texttt{rad}} = 0.75$ and $\texttt{max}_{\texttt{rad}} = 1.50$.
3. Rotation: $\texttt{min}_{\texttt{rot}} = -15$ and $\texttt{max}_{\texttt{rot}} = 15$.
4. Scale: $\texttt{min}_{\texttt{sca}} = 0.75$ and $\texttt{max}_{\texttt{sca}} = 1.50$.

Using these values, we computed the mean $\mu_{\texttt{T}} = \frac{\texttt{min}_{\texttt{T}}+\texttt{max}_{\texttt{T}}}{2}$ and the standard deviation $\sigma_{\texttt{T}} = \texttt{max}_{\texttt{T}} - \texttt{min}_{\texttt{T}}$, where $\texttt{T} \in \{\texttt{dis}, \texttt{rad}, \texttt{rot}, \texttt{sca}\}$, for a normal distribution of the parameter of each transformation. Then, for each control point and each transformation, we drew a parameter from its corresponding normal distribution and applied the transformation. Then, using a script, we generated 15 new images by applying these transformations with the parameter randomly drew for each sample of 50-MNIST. This procedure produced images that are slightly perturbed versions of the original sample. Examples of the generated images are shown in Fig. 16.

Fig. 16. Examples of generated samples. The left-most column shows samples of 50-MNIST. The other columns present their 'augmented' samples.

Then, we combined the 50-MNIST dataset and the generated samples to train an SVM, using the same hyper-parameters that were used to train the SVM on 50-MNIST. We computed performance scores using the MNIST test dataset for both SVMs: trained on 50-MNIST and augmented 50-MNIST. The computed scores are shown in Table 1. In general, our data augmentation strategy increased the accuracy of the model from 0.83 to 0.88. It is worth noting that our approach produces small shape perturbation as observed in Fig. 16, like the shape of the holes in digits zero and two; the top and the tail lines of digit two. As far as we know, these shape changes are not easily randomly obtained by other techniques. In addition, we use a similar approach to 15-EMNIST, randomly picked 15 samples for each class of EMNIST [72], and increased the accuracy of the SVM model from 0.47 to 0.54 (the classifier full table is available in the supplementary material [54]). Thus, the results show the potential of our tool for data augmentation tasks.

4.4. Other applications

In this section, we further combine the node selection in the global manipulation (Sec. 3.2) with more features in local manipulation (Sec. 3.3), including scaling, moving, rotating,

Table 1. Testing scores of the SVM model trained on 50-MNIST and augmented 50-MNIST (aug. 50-MNIST).

	Precision		Recall		F1-score		
Digit	50-MNIST	Aug. 50-MNIST	50-MNIST	Aug. 50-MNIST	50-MNIST	Aug. 50-MNIST	Support
0	0.92	0.93	0.91	0.95	0.92	0.94	980
1	0.93	0.90	0.93	0.98	0.93	0.94	1132
2	0.89	0.93	0.78	0.84	0.84	0.88	1032
3	0.87	0.87	0.76	0.86	0.81	0.86	1010
4	0.90	0.87	0.82	0.86	0.86	0.87	982
5	0.77	0.80	0.77	0.87	0.77	0.83	892
6	0.90	0.91	0.84	0.94	0.87	0.92	958
7	0.63	0.88	0.87	0.90	0.73	0.89	1028
8	0.76	0.86	0.79	0.76	0.78	0.81	974
9	0.80	0.84	0.78	0.83	0.79	0.83	1009
Accuracy					0.83	0.88	10000

cutting and copying CPs, to implement additional applications.

Artistic illumination effects can be achieved with our method. Fig. 17 shows two examples to illustrate the potential of our tool to produce these artistic effects. The enlargement (a2), diminution (b2), movement (a3, b3), and removal (a4, b4) of the white light spots can be implemented by scaling up, scaling down, moving or rotating, and removing, respectively, all control points in all nodes representing these highlights on the three components (YUV) of the tomato (a1) or the copper ball image (b1). The manipulation demonstration is available in the supplementary material [54]. While the proposed image editing tool handles these simple objects with ease, we do not claim that our tool does already have a well-established relighting mechanism for dealing with very complex objects.

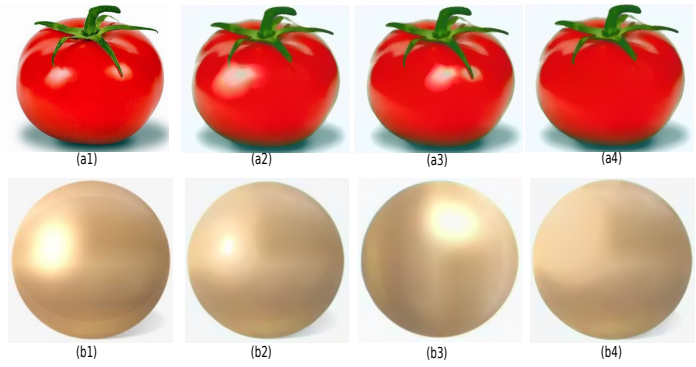

Fig. 17. Examples of producing artistic effects. Original images (a1, b1). Simulations of light spots enlarging (a2), shrinking (b2), offsetting (a3, b3), and removing (a4, b4).

Image rearrangement is also easily implemented using our method. Figure 18 shows an example by rearranging the birds' positions. We aim to exchange the position of two birds on the far right (D, E) and the two ones in the middle (B, C), and rotate bird D. We start by cutting the CPs (by pressing the *X* key) that encode the two birds on the far right (D, E) and pasting them (by pressing the *V* key) into an empty space in the spline manipulation interface; see the demonstration in the supplementary material [54]. Then we select the CPs representing birds B and C by dragging the displayed rubber band marker with the mouse and move them to the far right. Next, we select and move the CPs that represent birds D and E to the second and third positions. We end up using the rotate function (icon G in Fig. 11) to rotate bird D by about 30 degrees clockwise.

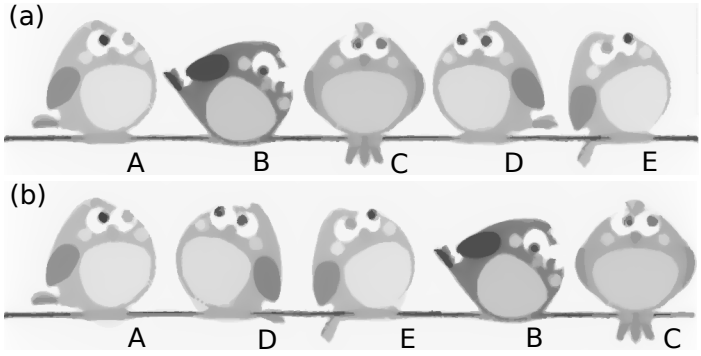

Fig. 18. Image rearrangement example. (a) The original image and (b) the manipulated result. We swap the position of two birds on the far right (D, E) and the two birds in the middle (B, C) and rotate bird D.

Clothing design can also be executed with our tool. Figure 19 gives two examples. For Fig. 19 (a), we intend to change the long-tube shoe to a short-tube one, and the thick sole to be thinner. For this, we first eliminate CPs that encode the long tube, and then reduce the radius of all CPs of the splines representing the sole of the shoe by about 1/3. For Fig. 19 (b), we simply add two stripes to the T-shirt by adding several splines encoding the stripes. Note that the added stripes should be put in the same position for the three components, *i.e.*, YUV, of the color image, otherwise false colors are produced; see Sec. 5. As visible, our approach generates images of good quality and fidelity.

5. Discussion

In this section, we discuss several aspects of our interactive image manipulation tool.

5.1. Comparison with GIMP

GNU Image Manipulation Program (GIMP) is a well-developed and widely used tool. An advanced interface and rich features allow GIMP to handle simple image painting as

Table 2. Comparison of GIMP and our tool for seven different applications in terms of manipulation convenience (top) and operation time (bottom; in minutes). Manipulation convenience is judged on a Likert scale that ranges from 'very easy' to 'very hard' (++, +, +/–, –, – –).

Application	Segmentation	Watermark removal	Deformation	Dataset augmentation	Artistic effects	Rearrangement	Clothing design
GIMP	+/– ~ 4	+/– 3–10	– ~ 6	+ ~ 2	++ <1	+ ~ 3	++ <1
Our tool	++ ~ 1	+/– 1–3	+ ~ 3	++ <1	++ <1	+ ~ 1	++ <1

Fig. 19. Clothing design examples. (a) Making a long tube, thick-soled, shoe into a short tube thin-soled shoe. (b) Adding two stripes to a T-shirt.

well as complex image manipulation. Table 2 compares the performance of GIMP and our method on seven applications introduced in Sec. 3.2 and Sec. 4. The operation time listed in the table only counts the manipulation time for each task, *i.e.*, for GIMP, excluding the time to open the image, and for our tool, excluding the initial encoding process (see Sec. 5.2). All experiments were performed on a Linux PC with an Nvidia RTX 2060 GPU. The participant is familiar with both our tool and GIMP, and has normal vision without color blindness issues. All the manipulation demonstrations are available in the supplementary material [54].

Image segmentation: For the skull-stripping segmentation (Fig. 10), GIMP takes around 4 minutes. The segmentation of each tissue requires careful marching with the mouse along the edges of the object with the *free* or *fuzzy* selection tool. The marching ants can be tuned by *adding to* and *subtracting from* the current selection modes. With our tool, however, it is only a matter of finding the nodes representing each tissue, which takes about 1 minute.

Watermark removal: To remove the watermarks, GIMP can use the *clone* or *healing* tool to cover the watermark with the pattern near the watermark, which takes around 3 minutes to clean an image with regular watermarks (such as (f1) in Fig. 13). However, when dealing with images with lots of watermark patterns (like (a1) in Fig. 13), this process lasts more than 10 minutes. In contrast, our tool only needs to remove the control points representing the watermarks, which takes only 1 to 3 minutes.

Image deformation: In GIMP, *cage transform* is used to deform an image. However, this feature is not handy when a certain part of the image needs to be rotated. For that task, the *rotate* tool plus the *clone* tool can help. To achieve the deformation in Fig. 15 (b2), GIMP takes about 6 minutes, whereas our method implements this only with the rotation function (icon H in Fig. 11), which takes around 3 minutes.

Dataset augmentation: For a single sample, our tool generates random changes by simply moving the corresponding CPs, which only needs dozens of seconds. While GIMP can achieve this using *cage* and *healing* tools, which takes about 2 minutes. Most importantly, our tool can *automatically* produce a high number of sample variants by randomly adjusting the positions of the sample CPs, which is almost impossible for GIMP.

Artistic effect: Both GIMP and our method can achieve artistic results easily and in very little time. To shrink the light spot in Fig. 17 (a1), GIMP can use the *shrink area* mode in the *warp transform* while our tool can simply apply the scaling feature (icon I in Fig. 11).

Image rearrangement: This application can be implemented with cut and paste, which is achieved in both GIMP and our tool. The difference is that GIMP pastes the content to a different layer, whereas our tool can operate on the same image layer, which takes less time.

Clothing design: GIMP can achieve this application by directly drawing on the image using *pencil* or *ink* tool, or by erasing certain content with the *eraser*. Our method can add new elements by adding new splines (icon D in Fig. 11) or remove content by delete the corresponding CPs (icon G in Fig. 11).

To sum up, compared with GIMP, our tool implements the above seven applications with the same or more convenient manipulation in similar or less time. Furthermore, our tool can achieve one function that GIMP cannot, *i.e.*, *automatically* producing a high number of sample variants for certain machine learning tasks. However, we acknowledge that the seven applications evaluated in the table are the ones that our tool excels at. There are also some applications that are easy to be achieved with GIMP but not for our tool, such as drawing content with a brush, copying content from another image, adding text, and more. In terms of quality, we admit that our skeleton-based method does not handle images with fine details well, which is not a problem for GIMP.

5.2. Running time

The longest (slowest) step in our end-to-end pipeline is the encoding process at the beginning. This is so since all the image information needs to be encoded, including all components on all layers. Take the original image in Fig. 10 as an example, whose intensities span from 0 to 255. Although its resolution is only 320×320 pixels, encoding all the information, *i.e.*, computing skeletons, and running spline fitting for all the

components, takes 118 seconds on a commodity PC. In addition, encoding runtime depends on the morphological tree size (number nodes). The size of the morphological tree depends on the content of the image; larger regions with tiny details like foliage and fur produce many nodes. Thus, encoding a larger image could be faster than a smaller image if its morphological tree contains sufficiently fewer nodes. We ran runtime experiments for a dataset of images with different content and sizes varying from 256×256 to 640×480, and plot a graph comparing the runtime of the encoding process according to the number of max-tree nodes (Fig. 20). More detail on this experiment as well as other runtime plots relating to image size and grayscale resolution are available in the supplementary material [54]. Fortunately, the encoding operation only needs to be

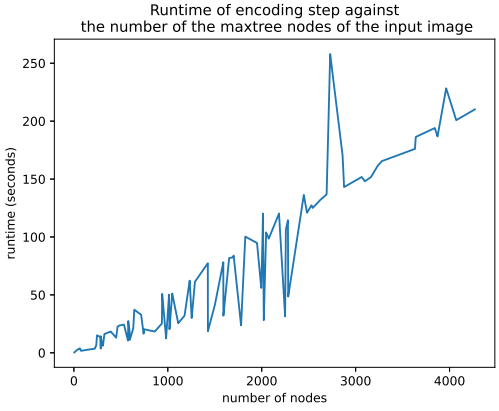

Fig. 20. Runtime experiment plot.

executed one time. Once the encoding process is complete, the subsequent series of operations, whether deleting, adding icicle nodes, all the manipulation operations on CPs, or reconstructing the components or images, occur in real-time. In practical applications, for inexperienced users, it takes approximately 2 to 4 minutes to successfully remove the watermark in an image or achieve a reasonable image deformation result. As for image rearrangement, artistic effect generation, and clothing design, it takes around 1 to 2 minutes to perform.

5.3. Limitations

Our interactive tool is not yet able to handle the three components of color images *simultaneously*, but only facilitates their manipulation separately. This leads to a problem where false colors, ghosting, and artifacts can occur when changes to the three components do not coincide. Figure 21 shows two examples. For (a), when the added stripes for the Y component and U component of the right image in Fig. 19 are in different positions, ghosting is introduced, as indicated by the arrows. Similarly, in (b), when the added eye shapes in the three components of the image in Fig. 15 (a1) do not coincide, artifacts and false colors are produced; see where the arrow points. However, we argue that these ghosting and false colors have perceptually little impact, and they can be avoided with careful manipulation.

We admit that for inexperienced users, some per-node manipulations may not be so intuitive and fully straightforward,

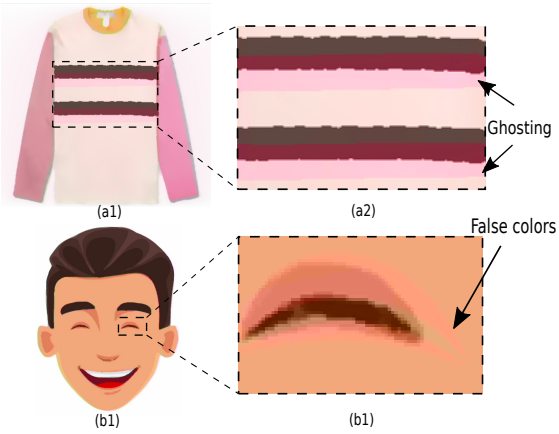

Fig. 21. Ghosting, false colors, and artifacts are introduced when changes to the three color components do not coincide.

e.g., adding splines to form the eye shape in Fig. 14 (d2), or performing some complex deformation operations on the legs of the binary horse shape (see the supplementary material [54]). To achieve these, some practice and experience with our tool is required. Yet, apart from that, we believe that removing, moving, rotating, scaling, cutting, and copying some CPs in whole are all easy to understand and conduct for inexperienced users. Besides, the global manipulation (Sec. 3.2) is straightforward to learn to use. There are only two windows in the interface: The left one is the image display interface while the right one shows the icicle tree; see Fig. 6 and Fig. 9, respectively. The two windows are *linked*, *e.g.*, when one clicks an icicle node on the right, its corresponding component is highlighted on the left, as shown in Fig. 6, and vice versa. In the supplementary material [54], we provide our full source code and all demonstration videos for replication purposes.

6. Conclusion

In this paper, we have presented a novel interactive image manipulation tool, which combines the spline-based medial axis for shape manipulation [7] with an icicle representation of component trees. Dealing with component trees instead of threshold sets allows finer-grained spatial control of each level set. We have demonstrated how to operate our tool in detail in Sec. 3. To verify the effectiveness of our tool, we have illustrated it by several applications of editing real-world images. Only manipulating icicle nodes globally (Sec. 3.2), such as removing multiple nodes in the icicle tree, achieves simple watermark removal tasks (Fig. 13 (a, e)). When adding local spline manipulations, more interesting applications are achieved. When the global manipulation is combined with the control points deletion function, more complex watermark removal tasks are achieved (Fig. 13 (b, c, d, f)). When combined with removing and adding CPs, interesting image deformation is achieved (Sec. 4.2). Also, we have shown an application of these deformations in data augmentation by generating new samples for a handwritten digit dataset (Sec. 4.3). We have also combined the global manipulation with more features in local manipulation, including scaling, moving, rotating, cutting and

copying CPs, to implement artistic illumination effects, image rearrangement, and clothing design (Sec. 4.4).

Several future work directions are possible. First, more functions can be added to our tool. One possibility is to allow to open two images simultaneously and stitch their content, such as seamlessly stitching objects from one image into the background of another image. Some additional functions, such as allowing users to directly move the node up or down in the icicle plot interface to achieve the intensity change of that node, can also be considered. Secondly, improving our tool to process color images more conveniently is also important to study. Finally, we aim to explore the potential of our tool for more applications in the future, such as image smoothing and image abstraction.

References

[1] Milliron, T, Jensen, RJ, Barzel, R, Finkelstein, A. A framework for geometric warps and deformations. ACM Trans Graph 2002;21(1):2051.

[2] Pérez, P, Gangnet, M, Blake, A. Poisson image editing. ACM Trans Graph 2003;22(3):313318.

[3] Igarashi, T, Moscovich, T, Hughes, JF. As-rigid-as-possible shape manipulation. ACM Trans Graph 2005;24(3):11341141.

[4] Wang, X, Yang, W, Peng, H, Wang, G. Shape-aware skeletal deformation for 2D characters. The Visual Computer 2013;29:545553.

[5] Choi, Y, Choi, M, Kim, M, Ha, JW, Kim, S, Choo, J. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018, p. 8789–8797.

[6] Barla, P, Bousseau, A. Gradient art: Creation and vectorization. In: Image and Video-Based Artistic Stylisation. Springer; 2013, p. 149–166.

[7] Wang, J, Kosinka, J, Telea, A. Spline-based medial axis transform representation of binary images. Computers & Graphics 2021;98:165–176.

[8] Wang, J, Kosinka, J, Telea, A. Spline-based dense medial descriptors for lossy image compression. Journal of Imaging 2021;7(8).

[9] Sederberg, TW, Parry, SR. Free-form deformation of solid geometric models. SIGGRAPH Comput Graph 1986;:151160.

[10] MacCracken, R, Joy, KI. Free-form deformations with lattices of arbitrary topology. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. 1996, p. 181188.

[11] Lipman, Y, Levin, D, Cohen-Or, D. Green coordinates. ACM Trans Graph 2008;27(3):110.

[12] Gain, J, Bechmann, D. A survey of spatial deformation from a user-centered perspective. ACM Trans Graph 2008;27(3):121.

[13] Reis, JPD, Kosinka, J. Injective hierarchical free-form deformations using THB-splines. Computer-Aided Design 2018;100:30–38.

[14] Weng, Y, Xu, W, Wu, Y, Zhou, K, Guo, B. 2D shape deformation using nonlinear least squares optimization. The Visual Computer 2006;22:653–660.

[15] Mota, T, Esperana, C, Oliveira, A. 2D shape deformation based on positional constraints and layer manipulation. In: 2011 Brazilian Symposium on Games and Digital Entertainment. 2011, p. 1–10.

[16] Tagliasacchi, A, Delame, T, Spagnuolo, M, Amenta, N, Telea, A. 3D skeletons: A state-of-the-art report. Computer Graphics Forum 2016;35(2):573–597.

[17] Lewis, JP, Cordner, M, Fong, N. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. 2000, p. 165172.

[18] Yan, HB, Hu, S, Martin, RR, Yang, YL. Shape deformation using a skeleton to drive simplex transformations. IEEE Transactions on Visualization and Computer Graphics 2008;14(3):693–706.

[19] Jacobson, A, Sorkine, O. Stretchable and twistable bones for skeletal shape deformation. ACM Trans Graph 2011;30(6):18.

[20] Sýkora, D, Dingliana, J, Collins, S. As-rigid-as-possible image registration for hand-drawn cartoon animations. In: Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering. 2009, p. 2533.

[21] Yang, W, Feng, J, Wang, X. Structure Preserving Manipulation and Interpolation for Multi-element 2D Shapes. Computer Graphics Forum 2012;:22492258.

[22] Yu, Y, Zhou, K, Xu, D, Shi, X, Bao, H, Guo, B, et al. Mesh editing with poisson-based gradient field manipulation. ACM Trans Graph 2004;23(3):644651.

[23] Raskar, R, Ilie, A, Yu, J. Image fusion for context enhancement and video surrealism. In: Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering. 2004, p. 85152.

[24] Levin, A, Zomet, A, Peleg, S, Weiss, Y. Seamless image stitching in the gradient domain. In: European Conference on Computer Vision; vol. 4. 2004, p. 377389.

[25] Sun, J, Jia, J, Tang, CK, Shum, HY. Poisson matting. ACM Trans Graph 2004;23:315–321.

[26] Arias, P, Facciolo, G, Caselles, V, Sapiro, G. A variational framework for exemplar-based image inpainting. International Journal of Computer Vision 2011;93:319–347.

[27] Di Martino, JM, Facciolo, G, Meinhardt-Llopis, E. Poisson Image Editing. Image Processing On Line 2016;6:300–325.

[28] Isola, P, Zhu, JY, Zhou, T, Efros, AA. Image-to-image translation with conditional adversarial networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017, p. 5967–5976.

[29] Kim, T, Cha, M, Kim, H, Lee, JK, Kim, J. Learning to discover cross-domain relations with generative adversarial networks. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. 2017, p. 18571865.

[30] Zhu, JY, Park, T, Isola, P, Efros, AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: IEEE International Conference on Computer Vision. 2017, p. 22232232.

[31] Vinker, Y, Horwitz, E, Zabari, N, Hoshen, Y. Deep single image manipulation. ArXiv 2020;abs/2007.01289.

[32] Salembier, P, Oliveras, A, Garrido, L. Antiextensive connected operators for image and sequence processing. IEEE transactions on image processing : a publication of the IEEE Signal Processing Society 1998;7:555–570.

[33] Berger, C, Graud, T, Levillain, R, Widynski, N, Baillard, A, Bertin, E. Effective component tree computation with application to pattern recognition in astronomical imaging. In: 2007 IEEE International Conference on Image Processing; vol. 4. 2007, p. IV – 41.

[34] Souza, R, Tavares, L, Rittner, L, Lotufo, R. An overview of max-tree principles, algorithms and applications. In: 2016 29th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T). 2016, p. 15–23.

[35] Ballester, C, Caselles, V, Monasse, P. The tree of shapes of an image. ESAIM: Control, Optimisation and Calculus of Variations 2003;9:1–18.

[36] Najman, L, Couprie, M. Building the component tree in quasi-linear time. IEEE Transactions on Image Processing 2006;15(11):3531–3539.

[37] Wilkinson, M, Gao, H, Hesselink, W, Jonker, JE, Meijster, A. Concurrent computation of attribute filters on shared memory parallel machines. IEEE transactions on pattern analysis and machine intelligence 2008;30:1800–1813.

[38] Carlinet, E, Graud, T. A comparative review of component tree computation algorithms. IEEE Transactions on Image Processing 2014;23(9):3885–3895.

[39] Souza, R, Rittner, L, Lotufo, R, Machado, R. An array-based node-oriented max-tree representation. In: 2015 ICIP. 2015, p. 36203624.

[40] Donoser, M, Bischof, H. Efficient maximally stable extremal region (mser) tracking. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition; vol. 1. 2006, p. 553–560.

[41] Benediktsson, JA, Bruzzone, L, Chanussot, J, Mura, MD, Salembier, P, Valero, S. Hierarchical analysis of remote sensing data: Morphological attribute profiles and binary partition trees. In: Proceedings of the 10th International Conference on Mathematical Morphology and Its Applications to Image and Signal Processing. 2011, p. 306319.

[42] Westenberg, MA, Roerdink, JBTM, Wilkinson, MHF. Volumetric attribute filtering and interactive visualization using the max-tree representation. IEEE Transactions on Image Processing 2007;16(12):2943–2952.

[43] Passat, N, Naegel, B, Rousseau, F, Koob, M, Dietemann, JL. Interactive segmentation based on component-trees. Pattern Recognition 2011;44:2539–2554.

[44] Tavares, LA, Souza, RM, Rittner, L, Machado, RC, Lotufo, RA. Interactive max-tree visualization tool for image processing and analysis. In: 2015 IPTA. 2015, p. 119–124.
[45] Vachier, C. Extinction value: a new measurement of persistence. In: IEEE Workshop on Nonlinear Signal and Image Processing. 1995, p. 254257.
[46] Tavares, LA, de Souza, RM, Rittner, L, Machado, RC, de Alencar Lotufo, R. A max-tree simplification proposal and applications for the interactive max-tree visualization tool. 29th SIBGRAPI 2016;:313–320.
[47] Kruskal, JB, Landwehr, JM. Icicle plots: Better displays for hierarchical clustering. The American Statistician 1983;37(2):162–168.
[48] Fekete, JD. The infovis toolkit. In: Proceedings of the IEEE Symposium on Information Visualization. 2004, p. 167174.
[49] Bostock, M, Heer, J. Protovis: A graphical toolkit for visualization. IEEE transactions on visualization and computer graphics 2009;15:1121–1128.
[50] van Wijk, J, van de Wetering, H. Cushion treemaps: visualization of hierarchical information. In: Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99, San Francisco CA, USA, October 24-29, 1999). United States: IEEE Computer Society. ISBN 0-7695-0433-7; 1999, p. 73–78. doi:10.1109/INFVIS.1999.801860; 1999 IEEE Symposium on Information Visualization (InfoVIS'99) ; Conference date: 24-10-1999 Through 29-10-1999.
[51] Lommerse, G, Nossin, F, Voinea, L, Telea, A. The visual code navigator: an interactive toolset for source code investigation. In: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005. 2005, p. 24–31. doi:10.1109/INFVIS.2005.1532125.
[52] Kovesi, P. Good colour maps: How to design them. CoRR 2015;abs/1509.03700. URL: http://arxiv.org/abs/1509.03700. arXiv:1509.03700.
[53] Kovesi, P. Cet perceptually uniform colour maps. 2022. https://colorcet.com/.
[54] Anonymous, . Interactive image manipulation supplementary material. 2022. Attached to this submission.
[55] Hecht, S. The visual discrimination of intensity and the weber-fechner law. The Journal of General Physiology 2003;7:235–267.
[56] Wang, J, Terpstra, M, Kosinka, J, Telea, A. Quantitative evaluation of dense skeletons for image compression. Information 2020;11(5):274–292.
[57] Pal, NR, Pal, SK. A review on image segmentation techniques. Pattern Recognition 1993;26(9):1277–1294.
[58] Pham, DL, Xu, C, Prince, JL. Current methods in medical image segmentation. Annual Review of Biomedical Engineering 2000;2:315–337.
[59] Zaitoun, NM, Aqel, MJ. Survey on image segmentation techniques. Procedia Computer Science 2015;65:797–806.
[60] Hahn, H, Peitgen, HO. The skull stripping problem in mri solved by a single 3d watershed transform. In: Lecture Notes in Computer Science; vol. 1935. 2000, p. 134–143.
[61] Doshi, J, Erus, G, Ou, Y, Gaonkar, B, Davatzikos, C. Multi-atlas skull-stripping. Academic Radiology 2013;20(12):1566–1576.
[62] Huang, CH, Wu, JL. Attacking visible watermarking schemes. IEEE Transactions on Multimedia 2004;6(1):16–30.
[63] Pei, SC, Zeng, YC. A novel image recovery algorithm for visible watermarked images. Information Forensics and Security, IEEE Transactions on 2007;1:543 – 550.
[64] Cheng, D, Li, X, Li, WH, Lu, C, Li, F, Zhao, H, et al. Large-scale visible watermark detection and removal with deep convolutional networks. In: Pattern Recognition and Computer Vision. 2018, p. 27–40.
[65] Qin, C, He, Z, Yao, H, Cao, F, Gao, L. Visible watermark removal scheme based on reversible data hiding and image inpainting. Sig Proc: Image Comm 2018;60:160–172.
[66] Xu, C, Lu, Y, Zhou, Y. An automatic visible watermark removal technique using image inpainting algorithms. In: 2017 4th International Conference on Systems and Informatics (ICSAI). 2017, p. 1152–1157.
[67] Wang, J, Joao, L, Falcão, A, Kosinka, J, Telea, A. Focus-and-context skeleton-based image simplification using saliency maps. In: VISAPP; vol. 4. 2021, p. 45–55.
[68] Shorten, C, Khoshgoftaar, TM. A survey on image data augmentation for deep learning. J Big Data 2019;6:60. URL: https://doi.org/10.1186/s40537-019-0197-0. doi:10.1186/s40537-019-0197-0.
[69] Perez, L, Wang, J. The effectiveness of data augmentation in image classification using deep learning. CoRR 2017;abs/1712.04621. URL: http://arxiv.org/abs/1712.04621. arXiv:1712.04621.
[70] Benato, BC, Telea, AC, Falc£o, AX. Iterative pseudo-labeling with deep feature annotation and confidence-based sampling. In: 2021 34th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI). 2021, p. 192–198. doi:10.1109/SIBGRAPI54419.2021.00034.
[71] Deng, L. The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine 2012;29(6):141–142.
[72] Cohen, G, Afshar, S, Tapson, J, van Schaik, A. Emnist: Extending mnist to handwritten letters. In: 2017 International Joint Conference on Neural Networks (IJCNN). 2017, p. 2921–2926. doi:10.1109/IJCNN.2017.7966217.