

# PicWall: Photo Collage On-the-fly

Zhipeng Wu and Kiyoharu Aizawa

Department of Information and Communication Engineering, the University of Tokyo, Tokyo, Japan.

E-mail: zhipengwu@hal.t.u-tokyo.ac.jp

E-mail: aizawa@hal.t.u-tokyo.ac.jp

**Abstract**— Photo collage, which constructs a compact and visually appealing representation from a collection of input images, provides the best convenient and impressive user experience. Previous approaches for automatic collage generation are always analogized as optimization problems, in which the researchers are trying to find the best balance between maximizing the visibility of photos’ salient areas as well as compactly arrange the collage canvas layout. However, automatic saliency detection can sometimes be harmful since we cannot guarantee all of the user’s interest areas are well-kept. On the other hand, the rapid development of mobile technology also calls for a robust solution of fast collage generation without any computation-expensive processes such as saliency detection and graph-cut. The PicWall approach proposed in this paper offers real-time collage generation. Given the expected canvas sizes, it tightly packs the input images while keeping their aspect ratios and orientations unchanged. Experiments show that it costs less than 0.5ms for a 100-photo collage generation. Besides, various PicWall-based applications are also demonstrated.

## I. INTRODUCTION

“A picture is said to be worth a thousand words.” With the rapid development of multimedia technology and the popularity of digital devices, people can conveniently access to any kinds of digital photographic devices (e.g. digital camera, mobile phone, tablet PC, and web camera) and we are getting used to log our daily lives by taking photos rather than using text. Nevertheless, without advanced photo organization and visualization strategy, the huge amount of image resources such as hundreds of vacation photos in a single folder or thousands of images returned by a web query may draw a potential problem with respect to information access and acquisition. Recently, a new image visualization technique, namely photo collage, provides a compact and pleasing representation for summarizing and displaying a set of photos on single canvas. The main advantage of collage is that it allows the user to efficiently browse multi-images at once while still keeping the important details of them.

For collage generation, one typical idea is to minimize the viewing areas of the input images first, only leaving the ROIs (Region of Interest), and then create a layout and organize them into the main canvas. As a representative work, researchers first extracted the salient regions of each image and then arrange them by solving a Maximum a Posterior (MAP) problem [1]. They introduced a very efficient Markov chain Monte Carlo (MCMC) method for the optimization and illustrated potential applications such as desktop image browsing

as well as image search summarization. Xiao et al. presented an image segmentation based saliency detection algorithm [2]. They first reduced the image to around 20 representative color clusters. Then, the clusters were assigned with a back/foreground probability and further classified as “subject”, “background”, or “distraction”. The final ROIs were defined based on the region labels with an expanded margin. In [3], the saliency computation was accelerated by introducing an integral map (similar to integral image) thus it was easy to get the sum of importance values in a rectangular region. In “AutoCollage” [4], saliency areas were extracted by graph-cut with a special respect to particular objects (e.g. faces). To create a seamless transition between input images that are adjacent in the canvas, alpha-poisson blending was also included in the collage generation.

Meanwhile, since automatic ROI extraction is usually time-consuming and it may have the potential harm for losing image’s content information, some non-saliency detection based approaches were proposed as alternatives. Intuitively, they proposed to use pre-defined layout templates [5]. The collage was then filled by matching the metadata of photos to the template cells based on an optimization algorithm. Although templates were always carefully designed which make the final generated collage compact in composition and the canvas area to be fully utilized, while coping with a set of images with different aspect ratios (width/height) and orientations, the main drawback was that it requires pre-processing steps such as cropping and shrinking. On the other hand, in [6], Atkins proposed BRIC (Block Recursive Image Composition) which ensured to keep the original aspect ratios without cropping the input photos. BRIC adopted the “slicing structure” used in floorplan design [7]. It solved two linear equations of  $N$  (number of input images) variables in  $O(N^3)$  complexity. In [8], Fan further improved BRIC under the framework of genetic algorithm. His work included a “slicing structure”-based fast computation of photo layout as well as a new definition of cost function.

As a novel image visualization technique, photo collage has been applied to many fields and applications. For consumer photo album, a genetic algorithm was proposed to generate the personalized album pages [9]. And in [10], applications for photo-on-photo composition were considered. It is also worth mentioning “Tiling Slideshow”, in which they created music-driven photo collages for images having similar characteristics [11]. For video summary, “video collage” was generated by first selecting representative frames, extracting

ROIs, and seamlessly arranging on the canvas [12]. Comparing with traditional video summarization schemes, “video collage” enabled a more compact and visually appealing presentation of video content, and thus it is seen as an effective and efficient presentation for video browsing and understanding. In [13], the media files were formed into a matrix-like representation according to when and where they were taken. The user can easily browse the media collection by scrolling the time or space axis. On mobile platform, “iPhoto-book” was proposed as a solution to the problem of photo book creation on mobile devices. It constructed mobile-oriented collages and help the user better organize photos with great convenience [14].

In this paper, we propose a fast photo collage generation method – PicWall. As a general solution oriented to multiple usage contexts and device platforms, PicWall prevails other approaches in the following points.

1. **Fast:** Given a set of input images, PicWall can generate photo collage on-the-fly, which is particular suitable for real-time applications such as image retrieval service, online games, and human-computer interaction. According to experimental results, it costs less than 0.5ms for a 100-input-photo collage generation (excluding the time for image reading), and less than 0.1ms for 20-input-photo collage.
2. **User-Adjustable:** PicWall allows the user to customize the size of collage by setting canvas height and width.
3. **Content-Preserved:** PicWall assures to fully preserve the visual content of input images. Although these photos can be stretched, their aspect ratios are strictly kept, and there is no cropping as well as changing of orientations.

In Section II, we are going to introduce the basics of binary tree based collage generation for non-size-adjustable canvas. Section III and IV further extends this algorithm and show how to create a size-adjustable collage. The experimental results and demo applications are shown in Section V. Finally, we give the conclusion in Section VI.

## II. NON-SIZE-ADJUSTABLE COLLAGE GENERATION

Our collage generation algorithm is enlightened by the idea of “slicing structure” and full binary tree [6], [8]. The notion of “slicing structure” originates from floorplan design in VLSI circuit layout [7]. We have the following definitions.

- **Canvas dissection:** A subdivision of a given rectangle canvas by horizontal (“H”) cut and vertical (“V”) cut into a finite number of non-overlapping tile boxes.
- **Horizontal cut:** As shown in Fig. 1, by drawing a horizontal cut line on the canvas, it is divided into two non-overlapping tile boxes.
- **Vertical cut:** Same to horizontal cut. A vertical cut line is drawn on the canvas to divide it into two tile boxes.

A “slicing structure” is a canvas dissection that can be obtained by recursively cutting rectangles into smaller rectangles. Intuitively, we can map a full binary to any kinds of “slicing structure”. The full binary tree requires each of its

nodes to be either a leaf or a node with exactly two children. Fig. 1 illustrates a simple slicing structure and the corresponding full binary tree.

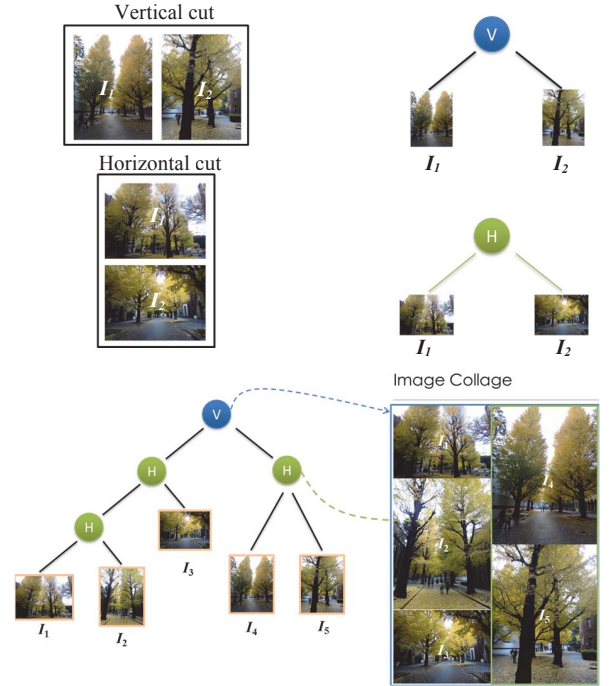


Fig. 1 Mapping a full binary tree to “slicing structure”.

This observation motivates us to generate a random binary tree first and then map the tree into a collage layout. Given  $N$ -image dataset, the generated binary tree should have exactly  $N$  leaves. Meanwhile, for all the other  $N - 1$  inner nodes, either “H” or “V” label is assigned. Once a full binary tree is constructed, how can we place all the input tile boxes into the canvas? The layout generation has two steps: (1) Bottom-up recursively calculating aspect ratio  $ar$ . (2) Top-down propagating image position.

The aspect ratio  $ar_{parent}$  of a parent rectangle (inner node) can be calculated from its left and right child  $ar_{left}$  and  $ar_{right}$ . It is proved that:

$$\text{“V” cut: } ar_{parent} = ar_{left} + ar_{right}. \quad (1)$$

$$\text{“H” cut: } 1/ar_{parent} = 1/ar_{left} + 1/ar_{right}. \quad (2)$$

Before showing the details of collage generation algorithms, we first define the data structure of a binary tree node.

TABLE I  
DATA STRUCTURE OF BINARY TREE NODE

SYMBOL	DATA TYPE	DESCRIPTION
<i>Split</i>	char	Denote the dissection type. “H” for horizontal, “V” for vertical.
<i>ar</i>	float	The current aspect ratio of this node.
<i>ar<sub>exp</sub></i>	float	The expected aspect ratio of this node.
<i>l<sub>child</sub></i>	TreeNode*	Pointer to its left child (if exists).
<i>r<sub>child</sub></i>	TreeNode*	Pointer to its right child (if exists).
<i>parent</i>	TreeNode*	Pointer to its parent (if exists).

---

**Algorithm 1: RECURSIVELY CALCULATE ASPECT RATIO**

---

```
1 Function RecurCalcAR(node)
   Input: Tree node : node
   Output: Node aspect ratio : node.ar
2 if node is not tree leaf then
3    $ar_{left} \leftarrow \text{RecurCalcAR}(\text{node}.l_{child})$ 
4    $ar_{right} \leftarrow \text{RecurCalcAR}(\text{node}.r_{child})$ 
5   if node is 'V'-split then
6      $\text{node}.ar \leftarrow ar_{left} + ar_{right}$ 
7   else
8      $\text{node}.ar \leftarrow \frac{ar_{left} \times ar_{right}}{ar_{left} + ar_{right}}$ 
9   end
10 end
11 return node.ar
12 end
```

---

Once the aspect ratios are calculated, we can set the size of the canvas accordingly. For example, we have a pre-defined canvas width ( $w = 800\text{px}$ ), and the calculated aspect ratio (Algorithm 1) for the whole canvas is 2, which indicates the height is  $400\text{px}$  ( $h = w / ar$ ). Then, the positions for each of the inner and leaf nodes can be propagated top-to-down, from the tree-root to the bottom layer (Algorithm 2).

---

**Algorithm 2: TOP-DOWN PROPAGATE POSITION**

---

```
1 Function TopDownCalcPos(node)
   Input: Tree node: node
2 if node's parent is 'V' split then
3    $\text{node}.height \leftarrow \text{node}.parent.height$ 
4    $\text{node}.width \leftarrow \text{node}.height \times \text{node}.ar$ 
5 else
6    $\text{node}.width \leftarrow \text{node}.parent.width$ 
7    $\text{node}.height \leftarrow \frac{\text{node}.width}{\text{node}.ar}$ 
8 end
9 if node is left child then
10    $\text{node}.x \leftarrow \text{node}.parent.x$ 
11    $\text{node}.y \leftarrow \text{node}.parent.y$ 
12 else
13   if node's parent is 'V' split then
14      $\text{node}.x \leftarrow \text{node}.parent.x +$ 
15        $\text{node}.parent.width - \text{node}.width$ 
16      $\text{node}.y \leftarrow \text{node}.parent.y$ 
17   else
18      $\text{node}.y \leftarrow \text{node}.parent.y +$ 
19        $\text{node}.parent.height - \text{node}.height$ 
20      $\text{node}.x \leftarrow \text{node}.parent.x$ 
21 end
22 end
```

---

For  $N$  input images, the full binary tree has  $(N - 1)$  inner nodes and  $N$  leaf nodes. Generating random tree, calculating aspect ratio, and propagating position have  $O(N)$  complexity, respectively. In all, non-fixed-aspect ratio generation can be finished in  $O(N)$  time.

### III. SIZE-ADJUSTABLE COLLAGE GENERATION

The algorithms in Section II enable photo collage generation. However, many applications have a size requirement on the canvas, and such a "random-size" collage generation seems to be useless. For instance, some mobile displays restrict the resolution to be  $320 \times 480$  or  $240 \times 320$ . And it turns to be  $1024 \times 768$  on Tablet PC and Laptop. In this section, we are going to improve the approaches in Section II and make the canvas size / aspect ratio adjustable for the user.

Given  $N$  input images and an expected canvas size, the proper generation of binary tree based photo layout is not easy. For instance, we have 20 images, the generated full binary tree has:

$$\begin{aligned} n_0 &= 20 & (0\text{-child leaf-node}) \\ n_1 &= 0 & (1\text{-child inner-node}) \\ n_2 &= n_0 - 1 = 19 & (2\text{-child inner node}) \end{aligned}$$

The total node number is  $n = n_0 + n_2 = 39$ . For  $n$ -node full binary tree, the number of different tree structures  $H(n)$  satisfies a modified Catalan sequence:

$$H(n) = \begin{cases} n & n = 0, 1 \\ H(0) \cdot H(n-1) + \dots + H(n-1) \cdot H(0) & n \geq 2 \end{cases} \quad (3)$$

$H(39) = 1767263190$ . For each of the 19 inner nodes, its split-type can be either 'H' or 'V'. The total tree number  $T(n) = 1767263190 \times 2^{19} \approx 927$  trillion. Noted  $T(n)$  exponentially grows as  $n$  gets larger, how to efficiently find a suitable solution among extremely large search spaces (trillions or even thousands of trillions) is the problem to be solved.

Look back at the collage generation algorithm in Section II. The result is a "random-sized" collage which is not adjustable by the user. This is because of the following random factors:

- a) Leaf nodes are *randomly* associated with images.
- b) Inner nodes are *randomly* set with a split type, either "H" (Horizontal cut) or "V" (Vertical cut).

Rather than cover all the search spaces, the demand of real-time processing requires us to rapidly generate an acceptable near-optimal solution. On the other side, we improve the "random" algorithm in Section II and propose a "guided" tree generation scheme by using "divide-and-conquer" paradigm

A divide-and-conquer algorithm recursively breaks down the original problem into sub-problems of the same type, until it becomes simple enough to be solved directly. To be more specific, we have the following steps.

- A. **Preparation:** Prepare the input image list  $L$ , expected aspect ratio  $ar_{exp}$ , create the initial tree root *node*, and set the number of associated images  $N$ .
- B. **Divide step:** Given input tuple  $(L, ar_{exp}, N, \text{node}^*)$ , assuming  $N \neq 1$  and  $N \neq 2$  (base cases):
  - 1) Randomly generate a split type ("V" or "H") for the current node.
  - 2) Create child nodes  $l\_node$  and  $r\_node$
  - 3) Set the number of associated images with the newly generated child nodes:
$$N_{left} = N_{right} = N / 2$$



- 4) Set split type (“V” or “H”) to *node*
- 5) If split type is “V”, initialize:
 
$$ar_{exp-left} = ar_{exp-right} = a_{exp} / 2$$
 Else:
 
$$ar_{exp-left} = ar_{exp-right} = ar_{exp} \times 2$$

C. **Conquer step:** Solve sub-problems recursively:  
 $(L, ar_{exp-left}, N_{left}, l\_node^*)$   
 $(L, ar_{exp-right}, N_{right}, r\_node^*)$

In any recursive algorithm, *base cases* serve as the terminator for recursion and there is considerable freedom in the choice of the *base cases*. Without loss of generality, since the number of associated images is decreasing in a geometrical ratio, we select *base cases* when it is 1 or 2. As you can find in Table II, the two *base cases* exist when we are going to generate leaf nodes for the binary tree. In other words, it performs the action of dispatching images from the input list to the tree leaves.

**Case  $N = 1$ :** Given the expected aspect ratio and the input list  $L$ , we need to select one image  $i$  with respect to the expected aspect ratio  $ar_{exp}$  and remove  $i$  from  $L$ . This can be down by pre-sorting  $L$  at the initialization step and do binary search when it is invoked.

**Case  $N = 2$ :** Given the expected aspect ratio and the input list  $L$ , we need to select two images  $i$  and  $j$  to best fit  $ar_{exp}$ . Considering the split type of parent *node\**, we have the following situations.

$$\text{“V” cut: } ar_{exp} \approx ar_i + ar_j \quad (4)$$

$$\text{“H” cut: } 1/ar_{exp} \approx 1/ar_i + 1/ar_j \quad (5)$$



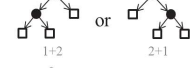

#### Algorithm 3: FIND IMAGE PAIR

```

1 Function FindImgPair( $\alpha_{exp}, L$ )
   Input: Expected aspect ratio:  $\alpha_{exp}$ , Aspect ratio list:  $L$ 
   Output: Index for image pair  $\langle i, j \rangle$ 
2    $p_{front} \leftarrow 0$ 
3    $p_{rear} \leftarrow L.size - 1$ 
4    $min_{diff} \leftarrow |L[p_{front}] + L[p_{rear}] - \alpha_{exp}|$ 
5   while  $p_{front} \leq p_{rear}$  do
6     if  $L[p_{front}] + L[p_{rear}] > \alpha_{exp}$  then
7       if  $|L[p_{front}] + L[p_{rear}] - \alpha_{exp}| < min_{diff}$  then
8          $min_{diff} \leftarrow |L[p_{front}] + L[p_{rear}] - \alpha_{exp}|$ 
9          $i \leftarrow p_{front}$ 
10         $j \leftarrow p_{rear}$ 
11      end
12       $p_{rear} = p_{rear} - 1$ 
13    else if  $L[p_{front}] + L[p_{rear}] < \alpha_{exp}$  then
14      if  $|L[p_{front}] + L[p_{rear}] - \alpha_{exp}| < min_{diff}$  then
15         $min_{diff} \leftarrow |L[p_{front}] + L[p_{rear}] - \alpha_{exp}|$ 
16         $i \leftarrow p_{front}$ 
17         $j \leftarrow p_{rear}$ 
18      end
19       $p_{front} = p_{front} + 1$ 
20    else
21       $i \leftarrow p_{front}$ 
22       $j \leftarrow p_{rear}$ 
23    end
24  end
25   $L.erase(i)$ 
26   $L.erase(j)$ 
27  return  $\langle i, j \rangle$ 
28 end

```

TABLE II  
TREE GENERATION BY USING DIVIDE-AND-CONQUER

$N$	Node type	Example	Description
1	leaf node		<leaf node>: select one image $i$ from set $L$ . The aspect ratio of $i$ should best fit the $ar_{exp}$ .
2	leaf node		<leaf node>: select two image $i, j$ from set $L$ . Their aspect ratio should best fit the $ar_{exp}$ .
3	inner node		<inner node>: split the number $N$ ( $3 = 1 + 2$ ). Create two child nodes and recursively calculate.
4	inner node		<inner node>: split the number $N$ ( $4 = 2 + 2$ ). Create two child nodes and recursively calculate.
$\vdots$			

Once we sort the input images according to their aspect ratios, the solution for  $i$  and  $j$  can be found by traversing the sorted list with two pointers  $P_{front}$  and  $P_{rear}$ . Noticing if the images are sorted in  $ar$  ascending order, they are also sorted in  $(1/ar)$  descending order, without loss of generality, we only consider the “V” cut situation in Algorithm 3.

The complexity for finding two images by Algorithm 3 is  $O(N)$ . Finding one image (case 1) by binary search is  $O(\log N)$ . Thus, generating a  $N$ -leaf tree can be finished in  $O(N^2)$ .

#### IV. TREE ADJUSTMENT

Although the algorithm in Section III improves the tree generation by using divide-and-conquer paradigm, it still cannot guarantee that the generated canvas meets the user’s size (aspect ratio) requirement. In this section, we propose a novel top-down adjustment algorithm to refine the generated binary tree. We achieve this by traversing the inner nodes and adjusting their split types (“H” or “V”). The algorithm runs with the complexity of  $O(N)$ .

Intuitively, “V” cut makes the node’s aspect ratio larger as it is the sum of its child nodes’ aspect ratios. On the contrary, “H” cut makes the node’s aspect ratio smaller. Since we have already defined the expected aspect ratio for the whole canvas (the top-level root node in the binary tree), the basic idea is:

- If current node’s  $ar$  is too large than  $ar_{exp}$ , the split type should be “H”, and we can set  $ar_{exp}$  for its child nodes accordingly.
- If current node’s  $ar$  is too small than  $ar_{exp}$ , the split type should be “V”, and we can set  $ar_{exp}$  for its child nodes accordingly.

Algorithm 4 shows the proposed fast tree adjustment. It is understandable that certain values of aspect ratio are impossible to reach (e.g. given 3 images with aspect ratio [1.0, 1.3, 1.5] but the target aspect ratio is 5.0). We skip over these cases and set an acceptance range (e.g.  $[ar_{exp} \pm 5\%]$ ) for near-optimal solution. The PicWall system iteratively runs tree adjustment until the result aspect ratio is acceptable. In most cases, this scheme performs successfully. However, another round of tree generation is invoked again under the following cases:

- 1) The adjustment number exceeds the maximum allowed value (defines as 100 in current implementation).
- 2) After tree adjustment, no node has been updated.

**Algorithm 4: FAST TREE ADJUSTMENT**

```

1 Function AdjustTree(node, th)
  Input: Tree node: node, Adjustment threshold: th
  if node is not a tree leaf then
    if node.ar > node.arexp × th then
      node.split = 'H'
    end
    if node.ar < node.arexp / th then
      node.split = 'V'
    end
  end
  if node is 'V' split then
    node.lchild.arexp ←  $\frac{\text{node.ar}_{\text{exp}}}{2}$ 
    node.rchild ← node.lchild
  else
    node.lchild.arexp ← node.arexp × 2
    node.rchild ← node.lchild
  end
  AdjustTree(node.lchild, th)
  AdjustTree(node.rchild, th)
2 end

```

## V. EXPERIMENTS

In this section, we first show the performance comparison with state-of-the-art approaches [6, 8]. Then a comprehensive test of efficiency is given. Finally, we illustrate several PicWall-based applications on various device platforms.

### A. Performance Comparison

The first experiment compares PicWall with BRIC [6] and FAST [8]. Since all of the three approach provides content-preserved collage generation as well as user-personalized canvas size and aspect ratio, we simply use the image dataset and evaluation criterion proposed in [8].

- **Dataset:** [8] introduced two datasets. One is called “San Francisco” dataset which includes 10 images with the following aspect ratios {2.05, 1.53, 1.49, 1.74, 0.54, 1.58, 0.67, 1.20, 2.08, 1.46}. Another one is 25-photo Hawaii dataset with aspect ratios as {0.88, 1.33, 1.44, 0.94, 1.84, 1.82, 1.61, 1.35, 1.17, 1.87, 1.65, 1.49, 1.49, 1.73, 1.65, 0.64, 1.91, 0.50, 0.88, 1.74, 1.49, 0.50, 1.70, 1.77, 1.43}.
- **Canvas setting:** For San Francisco dataset, the canvas is set as 13×19 (inches<sup>2</sup>). For Hawaii dataset, it is 24×16 (inches<sup>2</sup>).
- **Evaluation metric:** [8] proposed two cost functions  $C_1$  and  $C_2$ .  $C_2$  measures the coverage of the canvas.

$$C_2 = 1 - \sum_{i=0}^{N-1} s_i \quad (6)$$

where  $N$  is the number of input images,  $s_i = (w_i \times h_i)/S$  is the normalized size of the  $i$ -th image ( $S$  is the canvas size).

In [6] and [8], the user is allowed to select several highlighted photos to show them in bigger tile boxes. For instance, in San Francisco dataset, the desired size for “Golden Gate Bridge” image is up to 5 and others are set to 1. In Hawaii

dataset, the “Waipio Valley Lookout” is set to 5 and others are set to one. PicWall approach also enables the user to set desired sizes to a sub-set of input images. As shown in Fig. 2, the region I (green) is generated based on the sub-set images with respect to their desired sizes. After laying region I at the top-left corner of the main canvas, we can further generate two collages to fill the blank areas (region II and region III). The difference between Fig. 2-(a) and (b) is the generation order of region II and III - whether to fill the horizontal blank first or the vertical blank first.



Fig. 2 Collage with user-customized image sizes.

Cost function  $C_1$  is proposed for evaluating the matching of image sizes to the user’s expectation.

$$C_1 = \sum_{i=0}^{N-1} k_i (s_i - t_i)^2 \quad (7)$$

where  $t_i$  is the normalized desired size of the  $i$ -th image. And  $k_i$  equals to 5 if  $(s_i/t_i) < 0.5$ . Otherwise,  $k_i$  equals to 1. Experimental results are shown in Table III and IV ( $T$  is calculated excluding the time of image I/O).

TABLE III  
PERFORMANCE COMPARISON ON 10-IMAGE DATASET

Run	BRIC			FAST			PicWall		
	$C_1$	$C_2$	$T(s)$	$C_1$	$C_2$	$T(s)$	$C_1$	$C_2$	$T(\text{ms})$
1	0.32	0.23	0.16	0.0065	0.035	0.49	0.07294	0.04461	0.055
2	0.56	0.30	0.17	0.0058	0.038	0.49	0.07321	0.03538	0.050
3	0.022	0.19	0.16	0.0098	0.090	0.48	0.07270	0.04769	0.049
4	0.56	0.30	0.16	0.0200	0.048	0.47	0.07321	0.03538	0.031
5	0.058	0.03	0.17	0.0054	0.047	0.48	0.07109	0.03833	0.048
Avg	0.30	0.21	0.16	<b>0.0096</b>	0.052	0.48	0.07263	<b>0.040278</b>	<b>0.047</b>

TABLE IV  
PERFORMANCE COMPARISON ON 25-IMAGE DATASET

Run	BRIC			FAST			PicWall		
	$C_1$	$C_2$	$T(s)$	$C_1$	$C_2$	$T(s)$	$C_1$	$C_2$	$T(\text{ms})$
1	0.092	0.068	3.8	0.0092	0.034	4.2	0.05287	0.01583	0.129
2	0.12	0.032	3.8	0.018	0.039	4.1	0.04698	0.02000	0.118
3	0.18	0.065	3.8	0.014	0.027	4.1	0.04292	0.02125	0.113
4	0.13	0.065	3.8	0.012	0.037	4.2	0.06548	0.00333	0.172
5	0.007	0.26	3.8	0.011	0.035	4.2	0.04747	0.01750	0.110
Avg	0.092	0.099	3.8	<b>0.013</b>	0.035	4.2	0.05114	<b>0.01558</b>	<b>0.128</b>

Both FAST and PicWall perform better than BRIC. FAST shows its advantage with respect to user-desired sizes ( $C_1$ ). However, the proposed PicWall can generate a more accurate

canvas with precise width and height. One thing worth to mention is that the processing time for both BRIC and FAST grows rapidly while the number of input images increases. Even for the FAST approach, when the number of images is up to 25, it requires several seconds for collage generation and that seems to be unacceptable in scenarios such as online image retrieval and human-computer interaction. On the other hand, PicWall runs  $10^4$  times faster, which shows its capacity for real-time applications. Fig. 3 shows the photo collage generated by our approach.

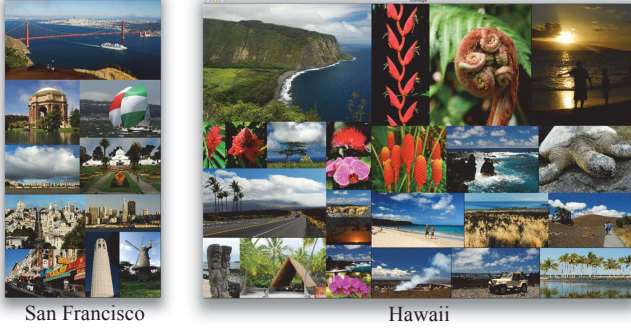


Fig. 3 Generated collages by PicWall.

### B. Efficiency Analysis

We then show the evaluation with respect to the efficiency of PicWall. Our dataset includes 100 images returned by Google image search. Their aspect ratios range from 0.67 to 2.00.

The first evaluation aims to find the relationship between the number of input images and algorithm efficiency (num. of tree adjustments, num. of tree generations, and processing time). We fix the expected canvas aspect ratio to 1.5 and change the number of input images from 20 to 100. Table V shows the experimental details (Results in rows are the average values of 1000 runs).

TABLE V  
EVALUATION FOR DIFFERENT NUMBER OF INPUT IMAGES.

Number of Input images	Tree Adjustment	Tree Generation	Processing Time (ms)
20	13.035	1.104	0.055785
30	45.094	1.407	0.127750
40	29.645	1.259	0.144606
50	9.543	1.071	0.131016
60	1.883	1.000	0.136629
70	2.165	1.004	0.171236
80	14.561	1.121	0.261470
90	23.611	1.206	0.349438
100	25.182	1.218	0.413838

Intuitively, the processing time is closely related to the number of tree adjustments and tree generations. And the tree generation depends on a random factor since we randomly set the split type to a newly generated node. According to the experimental results, the number of tree generations and adjustments not only depends on the number of input images but also depends on input aspect ratios and the expected canvas

size. In all, what we noticed is that the processing time roughly increases when the number of input images gets larger. Besides, the proposed PicWall can swiftly generate photo collages. It requires less than 0.5ms for a 100-image input and less than 0.1ms for 20-image input. Fig. 4 illustrates the results of collages with different number of input photos.

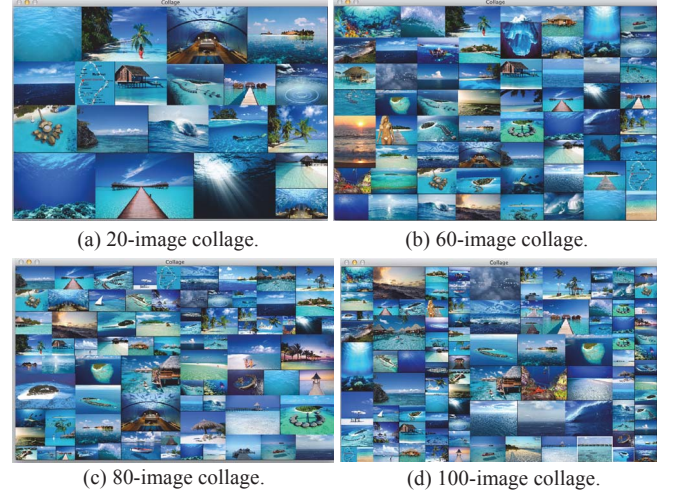


Fig. 4 Collages generated by different number of images.

In accordance with previous analysis, processing time is influenced by many factors such as the input images and the expected canvas aspect ratio. We also find that when the input image number is fixed, the trend of processing time is in correspondence with the trend of tree adjustment and tree generation (Fig. 5).

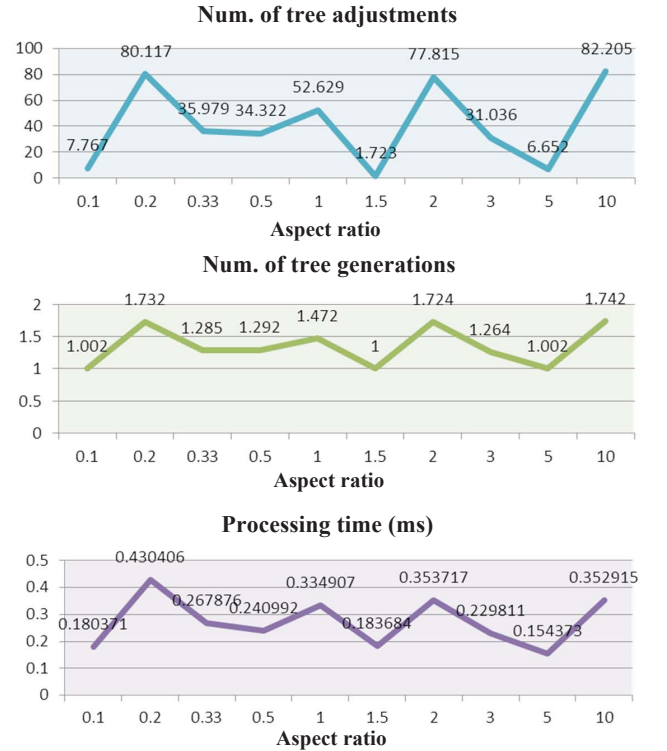


Fig. 5 Evaluation for collages with different number of images.



It is worth noticing that the processing time is roughly in direct proportion with the number of tree generations. Comparing with tree adjustment ( $O(N)$ ), the complexity of tree generation is much higher ( $O(N^2)$ ), and it accounts for the majority of the total processing time when we chose a large  $N$  (here,  $N=60$ ). Besides, we still notice that the proposed algorithm has the ability to deal with rare aspect ratios such as 0.10 and 10.00. In all, the canvases can be generated efficiently, usually in less than 0.5ms.

### C. Applications

As a real-time photo collage generation algorithm, PicWall shows its potential to be applied in various usage contexts and devices platforms. We demonstrate five applications with respect to (1) Retrieval result presentation; (2) Personal photo album visualization; (3) Mobile image browsing; (4) Video summarization; (5) Image to manga conversion.

#### (1) Retrieval result presentation

Modern online image search engines such as Google Images provide the best convenience for user to retrieval images based on text query and visual content. While focusing on the retrieval precision and recall, how to help users effectively visualize the returned results still remains a problem. As shown in Fig. 6, once a visual query is submitted, the retrieval results are listed in rows. Since these images usually have different orientations and aspect ratios, to compactly arrange them together, image resize and cropping are employed. As illustrated in Fig. 6, after entering the visual query, a list of images with different aspect ratios are returned. To effectively arrange the results in a user-friendly interface, cropping is adopted for some images. The photo on the top-right is the original image returned by the 20-th result. Because of the existence of image cropping (dashed box), some of the visual contents are lost.

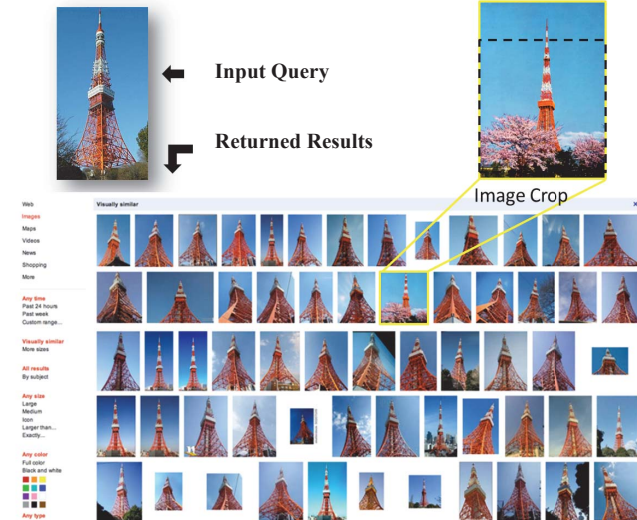


Fig. 6 Retrieval interface of Google Image.

As illustrated in Fig. 7, PicWall provides a novel representation for the returned images as well as preserves their origi-

nal content. Moreover, as shown in the yellow rectangle, the top- $K$  returned images are highlighted with bigger size and placed on the top-left corner. However, the downside for the collage-based image retrieval is that it loses the sequence information of the result images. The user cannot distinguish which one ranks exactly at the top-10 place and which one ranks at the top-100 place.

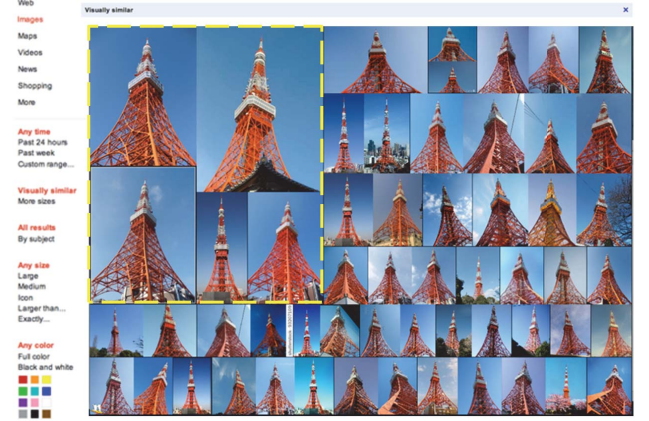


Fig. 7 Retrieval result presentation by PicWall.  
(The top-5 returned results are high-lighted with bigger size)

#### (2) Personal photo album visualization

Fig. 9 illustrates our demo for using photo collage to visualize the user's personal photo repository. Our demo system organizes all the local photos by their related social contacts/friends. At the very beginning, the user can chose one of the friends for photo browsing (Fig. 9-a). Then, all the events/folders related with that person are listed (Fig. 9-b). As shown in (Fig. 9-c), PicWall provides compact and palatable photo visualization with strong visual impact. If the user is interested in specific image, he can further view the details by clicking that photo (Fig. 9-d).

#### (3) "Shake & Show" – Mobile Image Browsing

Recently, the rapid development of mobile devices and the fast-growing of smartphone market have aroused great attention. One of the most remarkable advantages for mobile is its portability and rich human interaction such as swipe, pinch, and touch-hold. Based on the PicWall algorithm, we can provide an interesting feature called "Shake & Show" for browsing images on mobile phone. As shown in Fig. 8, when the user is browsing images, once he shakes his mobile, a new collage will be immediately generated and shown on the screen.



Fig. 8 Shake & Show for mobile image browsing.

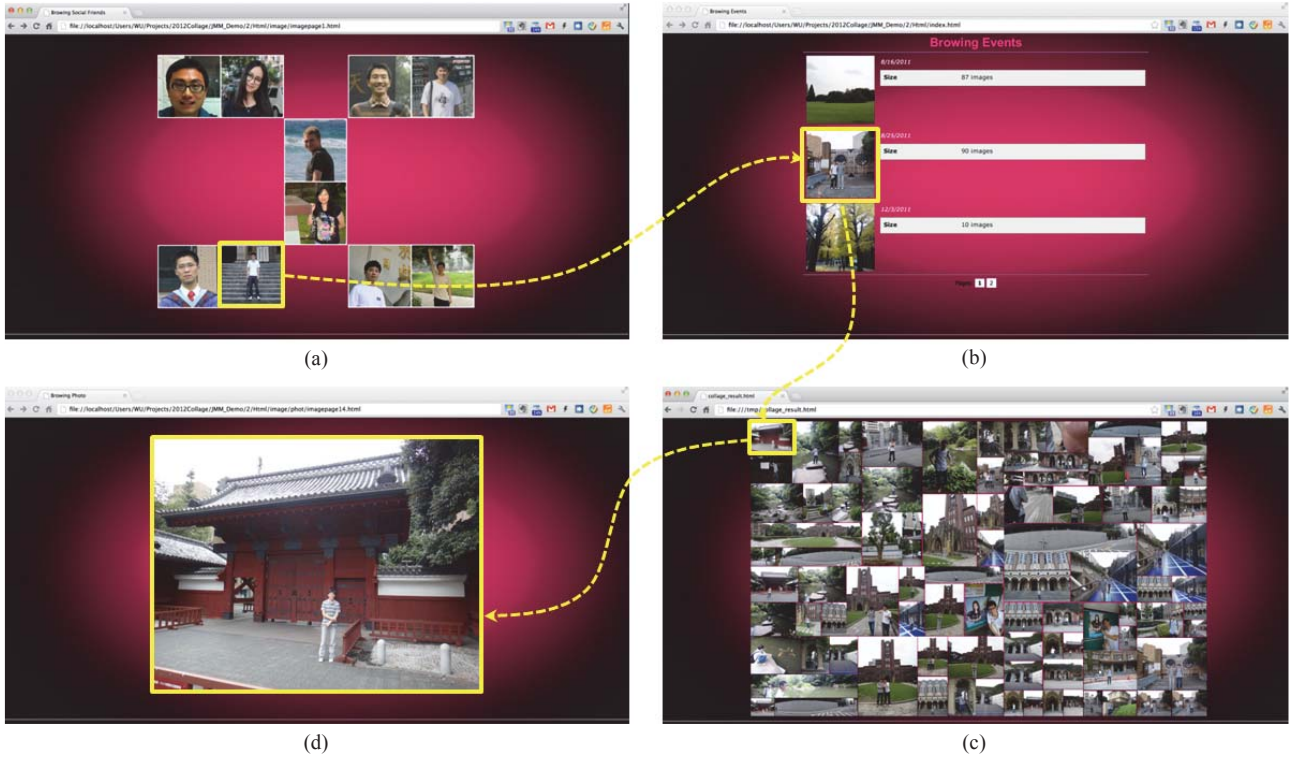


Fig. 9 Personal photo album visualization.

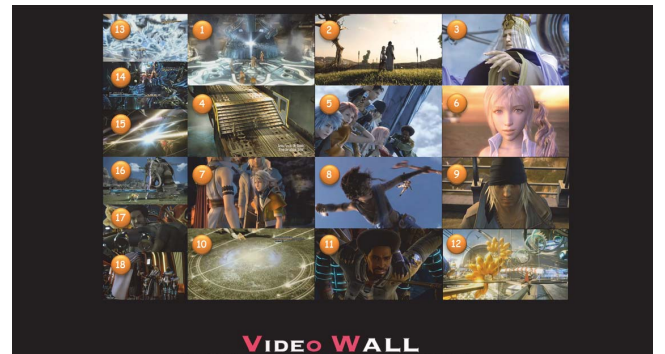
#### (4) Video Summarization and Visualization

Video is the collection of sequential frames/images. Previous works such as “Video Collage” [12] enabled the selection of representative frames, extraction of salient regions, and seamless ROI arrangement on a given canvas. In this paper, we propose a dynamic “Video Wall” which combines video collage and video player together. Given an input video, the generated video collage is a kind of synthesized image that enables the user to quickly learn the whole story. Meanwhile, since the video player has already been embedded in *Video Wall*, the user is able to skip to any parts of the video by clicking the tile boxes. Currently, we have two kinds of video collages: *Shots Wall* and *Highlights Wall*.

- The *Shots Wall* splits the video into shots and organizes the cover frames into the collage canvas. At the very beginning, the input video is automatically segmented into  $m$  shots, where  $m$  refers to the user customized shots number. Then, representative cover frames are extracted and further organized together. All the shots are sorted sequentially according to their timestamps. As shown in Fig. 10 (a), the pre-sorted video shots are arranged from top to bottom and from left to right.
- In *Highlights Wall*, the video highlights are firstly detected based on the motion feature. Then, all the highlight shots are ranked according to their ‘highlight scores’. In the step of collage generation, shots with higher scores are intended to be matched to larger tile boxes. As shown in Fig. 10 (b), the numbers in the orange circles stand for the rank of highlight scores.



(a) *Shots Wall*: The whole video is segmented into 30 shots and all the sequential shots are arranged from top to bottom/from left to right. (Numbers in the blue circles)



(b) *Highlights Wall*: A series of highlight shots is firstly extracted and used as video summary. For the collage composition, larger tile box corresponds to higher highlight score. (Numbers in the orange circles)

Fig. 10 Video Wall for video summarization and visualization.



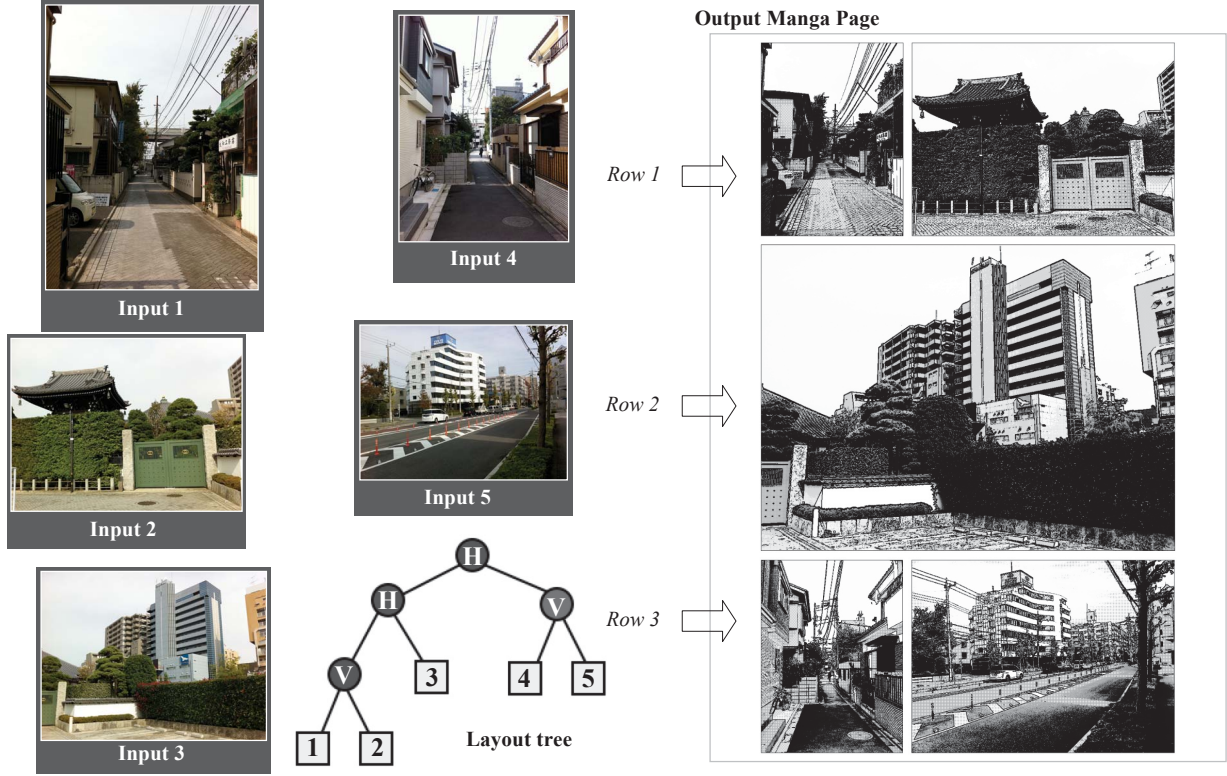


Fig. 11 Image to manga conversion.

##### (5) Image to Manga Conversion

With the rapid progress of multimedia technology, amateur-oriented image to manga conversion has caused wide research interest recently [15, 16]. Meanwhile, manga page layout has also become a core problem in computational manga [17].

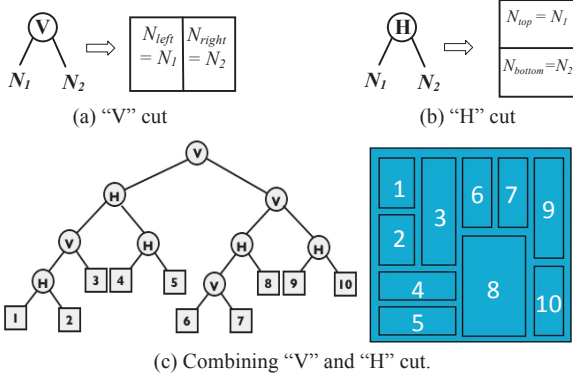


Fig. 12 Relationship between leaf order and image position.

Comparing with general photo collage, manga layout generation has the following uniqueness.

1) *Fixed reading sequence.* Rather than presenting a collection of static images, the images in a manga page are deliberately arranged in order to tell a story. This means that the sequential order of the input images should be strictly preserved.

As shown in Fig. 12, for each of the inner node, its left child  $N_{left}$  is guaranteed to locate on the left side (or top) of its right child  $N_{right}$ . The conventional reading order (left-to-right and top-to-bottom) can be naturally preserved by sequentially assigning input image to tree leaves.

It is also worth mentioning the *RIGHT-TO-LEFT* reading order of Japanese manga. Traditional Japanese manga starts at the back of the book. It is read from right-to-left. Then skip down to the next row. Speech bubbles, words and sound effects are also read from right-to-left. To mimic this, the image arrangement scheme should be slightly changed. For each of the “V” cut node, its left child should be placed on the right side of its right child. Fig. 13 illustrates the comparison between general cartoon reading order and Japanese manga reading order.

2) *Portrait manga page & vertically aligned rows.* Manga pages are divided into several rows (Fig. 11). Each of these rows is read from right-to-left, and then skips down to the next row. The composition of vertically aligned rows determines the portrait print orientation of manga pages. In other words, the aspect ratio is always less than 1.0 and the slit type for top-level root node should be fixed to “H”.

For commercial manga, the number of images displayed on single canvas is usually less than 10. According to our experimental results, the binary tree based layout generation runs super-fast. The computation can be finished in less than one millisecond.

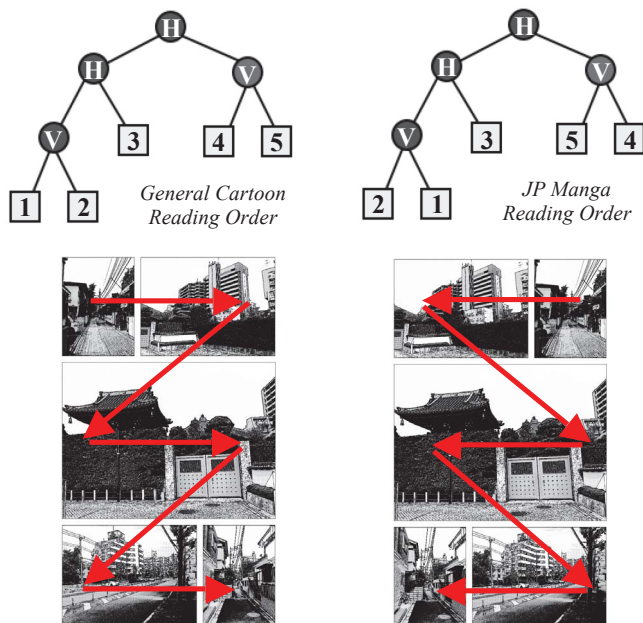


Fig. 13 JP manga reading order V.S. general cartoon reading order.

## VI. CONCLUSION

We propose PicWall, a real-time photo collage solution in this paper. Our approach characterizes on three points: (1) Fast collage generation; (2) User-customized canvas size; (3) Content-preserved layout. Motivated by the “slicing structure” in floorplan design, we first generate a full binary tree and then it can be mapped into a collage composition. To generate collages with user-customized sizes, we further improve the tree generation scheme by introducing “divide-and-conquer” paradigm as well as a fast tree adjust algorithm. In the experiments, we compare our approach with existing collage generation schemes with respect to efficiency and effectiveness. Moreover, several PicWall-based applications have also been discussed.

## REFERENCES

[1] T. Liu, J. Wang, J. Sun, N. Zheng, X. Tang, and H.-Y. Shum, “Picture Collage,” *IEEE Transactions on Multi-media*, vol. 11, no. 7, pp. 1225–1239, Nov. 2009.

[2] J. Xiao, X. Zhang, P. Cheatle, Y. Gao, and C. B. Atkins, “Mixed-initiative photo collage authoring,” in *Proceeding of the 16th ACM international conference on Multi-media*, 2008, pp. 509–518.

[3] Y. Yang, Y. Wei, C. Liu, Q. Peng, and Y. Matsushita, “An improved belief propagation method for dynamic collage,” *The Visual Computer*, vol. 25, no. 5–7, pp. 431–439, Mar. 2009.

[4] C. Rother, L. Bordeaux, Y. Hamadi, and A. Blake, “AutoCollage,” *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 847–852, Jul. 2006.

[5] N. Diakopoulos and I. Essa, “Mediating photo collage authoring,” in *Proceedings of the 18th annual ACM symposium on User interface software and technology*, 2005, pp. 183–186.

[6] C. B. Atkins, “Blocked recursive image composition,” in *Proceedings of the 16th international conference on Multimedia*, 2008, pp. 821–824.

[7] D. F. Wong and C. L. Liu, “A New Algorithm for Floor-plan Design,” in *ACM/IEEE Design Automation Conference*, 1986, pp. 101–107.

[8] J. Fan, “Photo Layout with a Fast Evaluation Method and Genetic Algorithm,” in *2012 IEEE International Conference on Multimedia and Expo Workshops*, 2012, pp. 308–313.

[9] J. Geigel and A. Loui, “Using genetic algorithms for album page layouts,” *IEEE Multimedia*, vol. 10, no. 4, pp. 16–26, Oct. 2003.

[10] A. Tian, X. Zhang, and D. R. Tretter, “Content-aware photo-on-photo composition for consumer photos,” in *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 1549–1552.

[11] W.-T. Chu, J.-C. Chen, and J.-L. Wu, “Tiling Slideshow: An Audiovisual Presentation Method for Consumer Photos,” *IEEE Multimedia*, vol. 14, no. 3, pp. 36–45, Jul. 2007.

[12] T. Mei, B. Yang, S.-Q. Yang, X.-S. Hua, “Video Collage: Presenting a Video Sequence Using a Single Image,” *The Visual Computer*, vol. 25, pp. 39–51, Jan. 2009.

[13] Q. Xu, Z. Wu, G. Li, L. Qin, S. Jiang, and Q. Huang, “Memory matrix,” in *ACM Multimedia*, 2010, pp. 927–930.

[14] J. Xiao, N. Lyons, C. B. Atkins, Y. Gao, H. Chao, and X. Zhang, “iPhotobook,” in *Proceedings of the international conference on Multimedia*, 2010, pp. 1551–1554.

[15] Y. Qu, T.-T. Wong, and P.-A. Heng, “Manga colorization,” *ACM TOG*, vol. 25, 2006.

[16] Y. Qu, W.-M. Pang, T.-T. Wong, and P.-A. Heng, “Richness-preserving manga screening,” *ACM TOG*, vol. 27, no. 5, Dec. 2008.

[17] Y. Cao, A. B. Chan, and R. W. H. Lau, “Automatic stylistic manga layout,” *ACM TOG*, vol. 31, no. 6, Nov. 2012.