Title	Quality-guided phase unwrapping implementation: an improved indexed interwoven linked list
Author(s)	Zhao, Ming; Kemao, Qian
Citation	Zhao, M., & Kemao, Q. (2014). Quality-guided phase unwrapping implementation: an improved indexed interwoven linked list. Applied Optics, 53(16), 3492-3500.
Date	2014
URL	http://hdl.handle.net/10220/20058
Rights	© 2014 Optical Society of America. This paper was published in Applied Optics and is made available as an electronic reprint (preprint) with permission of Optical Society of America. The paper can be found at the following official DOI: http://dx.doi.org/10.1364/AO.53.003492. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper is prohibited and is subject to penalties under law.

Quality-guided phase unwrapping implementation: an improved indexed interwoven linked list

Ming Zhao and Qian Kemao*

School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore

*Corresponding author: mkmqian@ntu.edu.sg

Received 3 March 2014; revised 26 April 2014; accepted 2 May 2014;
posted 5 May 2014 (Doc. ID 207057); published 27 May 2014

Quality-guided phase unwrapping (QGPU) is a widely used technique, and an adjoin list plays a very important role in the QGPU process. Indexed interwoven linked list (I2L2) is a data structure for implementing the adjoin list. In this paper, we propose three improvements on the I2L2. The first improvement is resumed searching, which records the highest nonempty level in the I2L2 and reduces the computational redundancy; the second is an adaptive mapping between the quality values and the I2L2 levels, which reduces the effect of concentrated quality value distribution. Last, I2L2-H, a new variant of the I2L2 combining the advantages of both the I2L2 and heap, is developed. With these three improvements, the improved I2L2 is over 6 times faster than the original one in the best cases, and it can process large phase maps in almost real time. © 2014 Optical Society of America

OCIS codes: (100.2650) Fringe analysis; (100.5088) Phase unwrapping; (120.2830) Height measurements.

http://dx.doi.org/10.1364/AO.53.003492

1. Introduction

In many phase-measuring techniques, such as optical interferometric methods [1], fringe projection profilometry [2,3], and interferometric synthetic aperture radar [4], the retrieved phase is wrapped within a range from $-\pi$ to π, that is, modulo 2π. The wrapped phase needs to be unwrapped to obtain a continuous phase. This process is called phase unwrapping [5].

Many phase unwrapping algorithms have been proposed, including spatial phase unwrapping methods, such as the Goldstein's branch cut method [6], the quality-guided method [7], the Flynn's minimum weighted discontinuity method [8], and the minimum L^p-norm method [9]; and temporal phase unwrapping methods, such as the dynamic unwrapping method [10], the multifrequency method [11–13], and the heterodyne method [14]. Spatial methods only need one wrapped phase map, but they have a restriction that they all assume the phase difference between neighboring pixels lies within the range $-\pi$ to π. They cannot handle the phase discontinuity problem that the true phase difference is out of this range. Meanwhile, spatial methods also suffer from the noise problem. On the other hand, temporal methods are robust to the discontinuity problem and not sensitive to noise [15], but they require multiple frames of wrapped phase along the time axis or multifrequency fringe patterns.

The quality-guided phase unwrapping (QGPU) technique has been proved as an effective, efficient, and automatic spatial phase unwrapping method [5,7]. It uses a quality map to represent the goodness of pixels, and good pixels are processed prior to the bad ones. Noisy pixels and pixels around discontinuous areas are usually considered as bad pixels and assigned low quality values. Many quality maps have been proposed to distinguish good and bad pixels [16].

The QGPU maintains a list called the adjoin list. Pixels whose neighboring pixels have been unwrapped will be stored in the adjoin list as the

1559-128X/14/163492-09$15.00/0

unwrapping candidates. Only the pixel with the highest quality value in the adjoin list will be selected for unwrapping. The QGPU process can be described as follows.

Step 1: choose the pixel with the highest quality value in the quality map as the seed pixel, and insert it into the adjoin list.

Step 2: unwrap the pixel with the highest quality value in the adjoin list. After that, remove this pixel from the adjoin list, and insert its unprocessed neighboring pixels into the adjoin list.

Step 3: repeat step 2 till the adjoin list is empty.

A data structure can be used to implement an adjoin list if it supports the following three operations: inserting a new pixel, deleting the unwrapped pixel, and searching the pixel with the highest quality value. In [16], we introduced a data structure named indexed interwoven linked list (I2L2), which satisfies these requirements very well. The I2L2 achieves very high unwrapping speed. In this paper, we further improve its performance from three aspects. First, because the original I2L2 has redundant computations on searching the highest nonempty level, we add a step to record it to reduce the redundancy and thus accelerate the searching. Second, we find that a highly concentrated quality value distribution slows down the unwrapping speed. An adaptive mapping method is proposed to reduce the effect of quality value distribution. Last, we combine the advantages of I2L2 and heap and create a new structure called I2L2-H. Heap is a widely used data structure with a characteristic of efficient element insertion [17]. The I2L2-H obtains the best results in most test cases, and we believe it is a good choice to implement the adjoin list.

The rest of paper is organized as follows. In Section 2, the original I2L2 is first introduced; I2L2 with searching redundancy removal and adaptive mapping is described; and a comparison between the improved I2L2 and original I2L2 is provided. In Section 3, the concept, operations, and implementation of I2L2-H are introduced, followed by a comparison with I2L2 and heap. The paper is concluded in Section 4.

2. I2L2

As discussed above, the adjoin list involves three types of operations. To achieve high efficiency, we assume that pixels in an adjoin list are ordered; that is, they are queued according to their quality values [16]. Thus, searching the highest quality value can be directly achieved by selecting the first pixel in the list. Thus we only need to consider how to insert a new pixel at a proper location in the list and how to delete the unwrapped pixel from the list.

A. Original I2L2

The I2L2 proposed in [16] is originated from the linked list [17]. If we use a linked list to implement the adjoin list, deleting the unwrapped pixel is very simple, but it is time consuming to insert a new pixel into the adjoin list at its proper location, especially when the adjoin list is very long. Our idea is to cut the whole linked list into several levels based on the quality values. Each level forms a short linked list, and these short linked lists are connected through an array, which stores their head indices. This data structure is shown in Fig. 1(a). The quality values decrease downward from the higher short linked lists to the lower ones. Meanwhile, in the same short linked list, quality values decrease from the head to the tail [i.e., from left to right in Fig. 1(a)].

With the data structure shown in Fig. 1(a), the program needs to allocate or release memory dynamically when a pixel is inserted or removed from the adjoin list. This problem can be avoided by preallocating memory for the short linked lists. A one-dimensional (1D) array [as shown in Fig. 1(b)] is created and shared by all short linked lists. The index of an element in the 1D array has a one-to-one relationship with the location of a pixel in the two-dimensional (2D) quality map. This data structure is the I2L2. The I2L2 is over 35% faster than the indexed linked list [16]. Because the data structures shown in Figs. 1(a) and 1(b) have an equivalent strategy, the former is used for explanation in the following sections, although the latter is used for real implementation.

B. Improved I2L2 with Resumed Searching

1. *Recording the Highest Nonempty Level*

In the QGPU, we need to find the pixel with the highest quality value for unwrapping. This pixel is the first pixel in the highest nonempty level in the I2L2 data structure. We thus need to search this highest nonempty level repeatedly. In our original implementation, the search always starts from the top of the index array until a nonempty level is found. This strategy is illustrated in Fig. 2(a). To reduce the searching cost so as to improve the I2L2, we keep a record of the highest nonempty level. When inserting

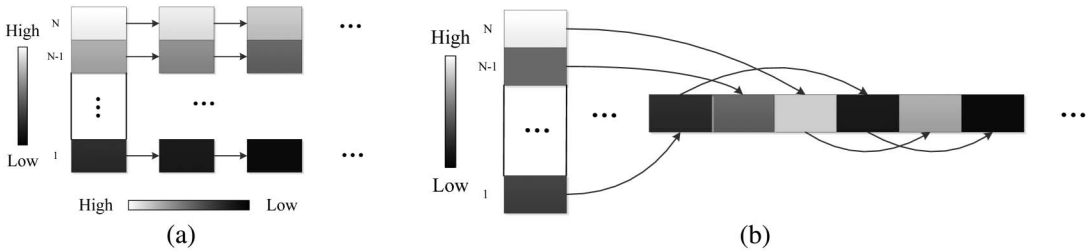


Fig. 1. (a) Indexed linked list and (b) I2L2.

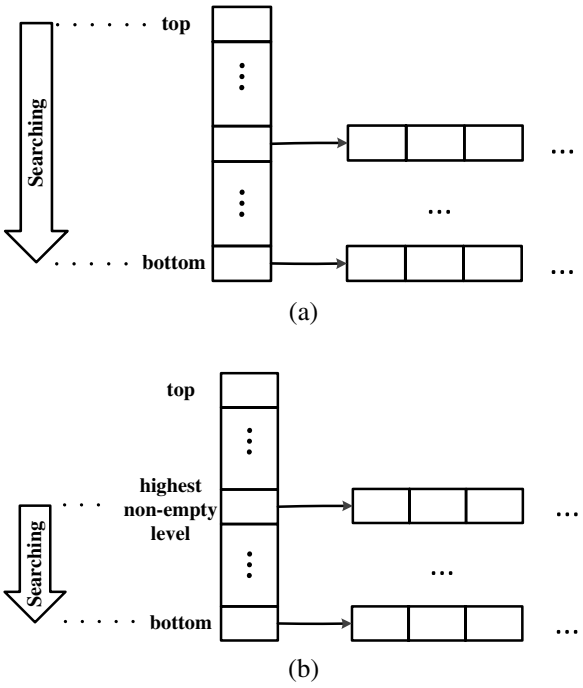


Fig. 2. Searching the highest nonempty level. (a) Refreshed searching and (b) resumed searching.

a new pixel into the adjoin list, the algorithm will check the level of this pixel. If the level of this pixel is higher than the recorded highest nonempty level, the record is updated. With this record, in our current implementation, the I2L2 starts the searching from the recorded position each time, instead of the top of the index array [see Fig. 2(b)]. To differentiate the original and current implementations, we call them refreshed searching and resumed searching, respectively. Obviously the resumed searching reduces the computing redundancy, and thus reduction of time cost can be expected.

In the original I2L2, the level number L is determined by the user, while $L = (MN)^{1/2}$ is recommended where M and N are the row and column numbers of a phase map [16]. Because the resumed searching strategy only performs a partial searching, we can increase the level number to improve the overall searching performance. Thus $L = k \cdot (MN)^{1/2}$ can be set where $k \geq 1$ (typically 16) is a factor.

2. *Comparison between the Refreshed Searching and the Resumed Searching*

In order to evaluate the performance of the resumed searching in our current implementation, three wrapped phase maps (Fig. 3) with different sizes are used. These examples are peaks (512 × 512), coffee cup lid (960 × 1280), and toy duck (1000 × 1000). The peaks example is simulated by the MATLAB built-in *peaks* function, and normally distributed random noise is added. The other two examples are obtained by the fringe pattern projection technique. The quality map of coffee cup lid is its modulation map, and the quality maps of the peaks example and toy duck example are generated by the windowed Fourier ridges algorithm [18]. All the quality maps are shown in Fig. 4. As we only focus on the phase unwrapping process, all quality maps are provided prior to the unwrapping process. The time cost of generating quality maps is not included in the results.

Both the refreshed searching and the resumed searching are implemented by C++. The test program runs on a Dell Precision T3600 workstation. The CPU frequency is 3.20 GHz and the memory is 16 Gbytes. To reduce the effect of background programs, all tests are performed 10 times, and the average is used. The refreshed searching with $L = (MN)^{1/2}$, the resumed searching with $L = (MN)^{1/2}$, and the resumed searching with $L = 16 \times (MN)^{1/2}$ are tested. These different searching strategies give exactly the same unwrapping results (shown in Fig. 5) and thus are not discussed. However, as shown in Table 1, their time costs are different and thus compared.

From the results, we find that the resumed searching has a better performance than the original refreshed searching. In the coffee cup lid example, the resumed searching performs over 300% faster than the refreshed searching. However, the improvements of the peaks example and the toy duck example are much lower. This is because in these two examples, the qualities of the majority pixels are very high and thus the pixels in I2L2 are located in the levels close to the top level. In such cases, the computing redundancy of the refreshed searching is not high, resulting in only a small improvement by the resumed

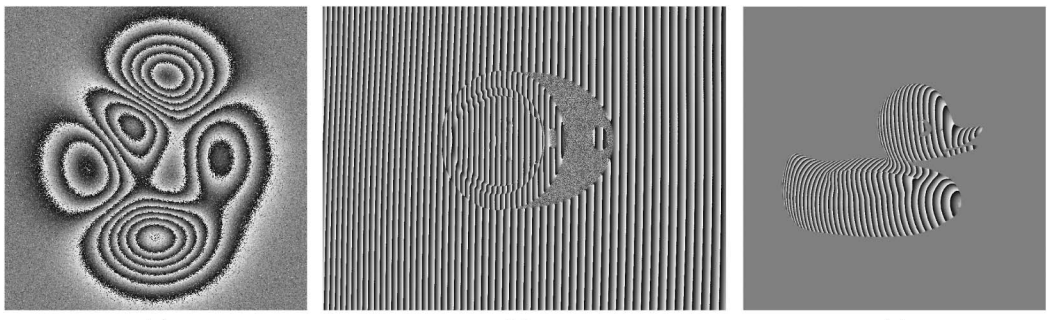
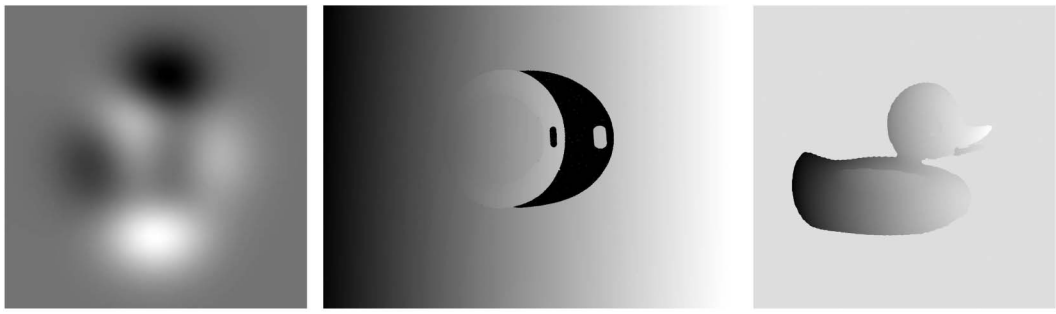


Fig. 3. Wrapped phase maps. (a) Peaks (512 × 512), (b) coffee cup lid (960 × 1280), and (c) toy duck (1000 × 1000).

(a) (b) (c)

Fig. 4. Quality maps. (a) Peaks, (b) coffee cup lid, and (c) toy duck.



(a) (b) (c)

Fig. 5. Unwrapped phase maps. (a) Peaks, (b) coffee cup lid, and (c) toy duck.

searching. In the following sections, the resumed searching is used as the default implementation.

C. Improved I2L2 with Adaptive Levels

1. Assigning Levels Adaptive to Quality Value Distribution

The intention of the I2L2 is to cut a long linked list into several short linked lists to make the searching more efficient. To achieve it, in the original I2L2, the whole range of quality values is equally cut into L small ranges, where L is the level number of I2L2 determined by the user. All pixels in the quality map are assigned a level index by linear mapping. In this implementation, each short linked list represents a small fixed range of quality values.

Unfortunately, in some wrapped phase maps, the quality values concentrate in a small range. Consequently, the majority of the pixels will be queued in a small number of short linked lists. These short linked lists are not really short any more. The peaks [shown in Fig. 6(a)] and toy duck [shown in Fig. 6(c)] serve as such examples. Furthermore, if we examine the coffee cup lid example [shown in Fig. 6(b)], this problem also exists. In fact, this is a problem in more or less all wrapped phase maps. We propose nonlinear and adaptive mapping from quality values to different levels, instead of the previous linear mapping. The basic idea is to build a histogram with a bin number much larger than the level number and properly combine bins into levels to realize a more dispersed quality values distribution. The idea is similar to but different from histogram equalization [19]. This adaptive mapping process is described in detail as follows.

Table 1. Comparison between the Refreshed Searching and Resumed Searching

	Searching	Level Number	Peaks	Coffee Cup Lid	Toy Duck
Time (ms)	Refreshed	$(MN)^{1/2}$	36.28	436.27	63.29
	Resumed	$(MN)^{1/2}$	26.92	140.93	58.39
	Resumed	$16 \times (MN)^{1/2}$	23.04	138.19	51.37

Step 1: build a histogram with $n \times L$ bins (n is an integer), and also assign the bin indices to all pixels in the quality map. Higher n reduces the bin width and obtains a better discrimination of the quality values, especially in the dense area. In our experiments, $n = 8$.

Step 2: reduce the bin number to L by merging the neighboring bins from left to right. Each bin should have no more than $M \times N/L$ pixels, except the last bin or the bins that have more than $M \times N/L$ pixels before the bin merging.

Step 3: assign the level indices to all pixels based on the relationship between bins and levels in step 2.

An example is given to explain the idea. Figure 7 shows a 10×10 quality map. The values of the pixels in this quality map are from 1 to 40. For easier demonstration, we set $L = 10$ and $n = 4$; that is, the I2L2 uses 10 levels, and the histogram has 40 bins. The histogram is computed and shown in Fig. 8. Following step 2, 40 bins are merged to 10 levels. The

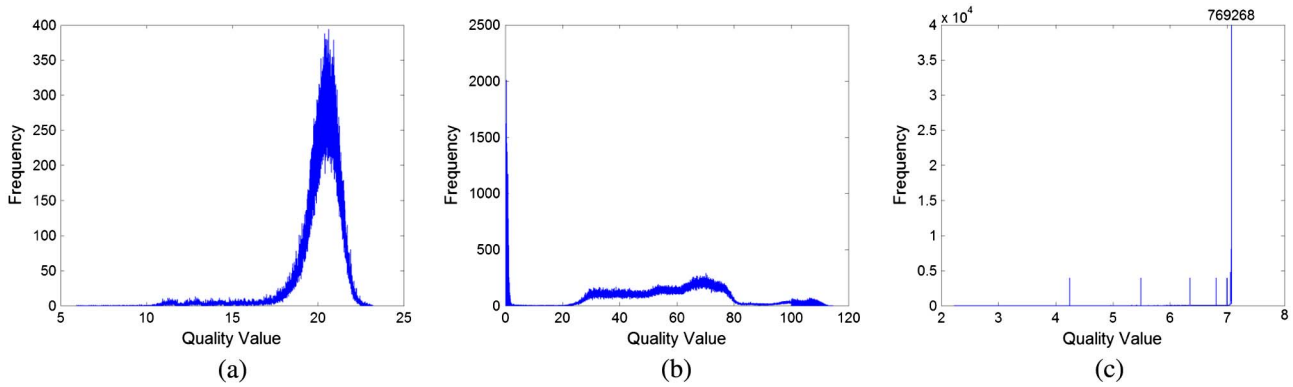


Fig. 6. Quality distribution curves. (a) Peaks, (b) coffee cup lid, and (c) toy duck.

1	30	19	3	13	5	8	6	6	2
1	28	11	5	7	5	8	4	15	8
7	5	3	6	1	2	7	7	4	1
33	5	13	2	8	4	1	6	8	4
38	23	5	31	6	35	3	8	25	2
38	2	4	3	4	6	3	8	35	8
5	23	11	3	8	14	6	3	7	6
5	9	7	4	2	3	2	6	7	25
7	20	15	5	4	18	6	2	5	4
40	19	8	1	8	1	1	1	2	1

Fig. 7. Quality map example (10 × 10).

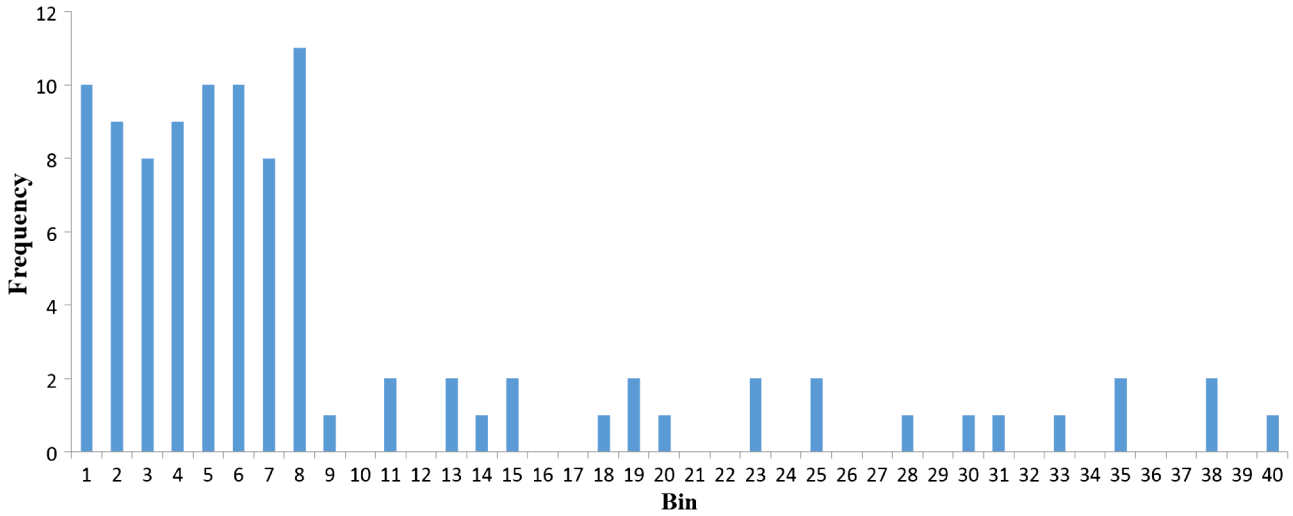


Fig. 8. Histogram with 40 bins.

obtained pixel distribution in levels is shown in Fig. 9(a). For comparison, the original nonadaptive pixel distribution is shown in Fig. 9(b). Obviously, the adaptive mapping provides better pixel distribution than the fixed mapping and thus an improved performance can be expected.

2. *Comparison between Fixed and Adaptive Mapping*

The three examples and configurations used in Section 2.B.2 are adopted again to examine the performance of the adaptive mapping. We first scrutinize the time costs to establish fixed and adaptive

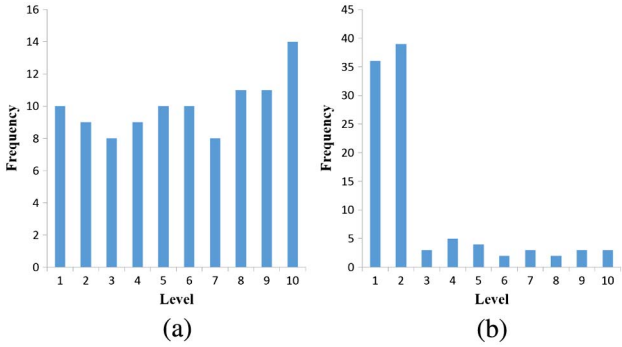


Fig. 9. (a) Histogram of quality distribution with adaptive mapping and (b) histogram of quality distribution with fixed mapping.

Table 2. Comparison of the Timing Cost for Assigning Level Numbers to Pixels between Fixed Mapping and Adaptive Mapping

		Peaks	Coffee Cup Lid	Toy Duck
Time (ms)	Fixed	2.79	13.44	10.96
	Adaptive	3.9	21.78	16.22

mapping. The results are shown in Table 2. It shows that although the adaptive mapping is more complicated than the fixed one, the additional time cost for establishing the mapping is insignificant. The quality value distribution curve comparisons are shown in Fig. 10. After establishing the adaptive mapping, the quality distribution curves in the peaks example and coffee cup lid example are flattened, except that some bins cannot be separated by the adaptive mapping. The toy duck example is an extreme case. Because this example has a large number of pixels with the same value, the adaptive mapping method cannot reduce the length of such bins, and only a small number of pixels are redistributed.

We next compare the phase unwrapping with two different mapping strategies. The results are listed in Table 3. It can be seen that the performance of adaptive mapping is over 120% of the fixed one. It is not as dramatic as we may hope. This is because the adaptive mapping cannot change a very concentrated quality distribution into a very wide distribution.

To summarize, after using both resumed searching and adaptive mapping, the I2L2 performance has improved explicitly. In the best case (coffee cup lid), the improved I2L2 reduces the time consumption from 436.27 to 85.34 ms, which is over 5 times faster. Even in the worst case (toy duck), the improved I2L2 still has 48% improvement.

3. I2L2-H: a New I2L2 Variant

In the I2L2, we need to insert new elements into an ordered linked list. Currently, a new element needs to be compared with all pixels larger than it until its proper location is found. Because a heap can archive more efficient insertion, a hybrid data structure combing the I2L2 and heap is proposed as the third improvement on the I2L2.

A. Proposed Hybrid I2L2 with Heaps

1. *Structure and Basic Operations of a Heap*

Heap [17] is a tree-based data structure. Each node of a heap has one pointer to its parent (i.e., a node in its upper level), and one pointer to each child (i.e., a node in its lower level). A heap has a property that all parent–child pairs in the data structure will follow the same value order. For instance, if it is a binary max heap (shown in Fig. 11), the values of parent nodes are always larger than or equal to the values of their child nodes, and the largest value can be found at the root position.

Inserting new elements and deleting the root element are two basic operations of a heap. To insert a new element, the first step is to add the new element to the end of the heap, which is the leftmost available position in the bottom level of the heap. Then, the new element is compared with its parent. If it is larger than its parent, it is exchanged with its parent and compared with its new parent. Otherwise, the insertion operation stops. An example of insertion is shown in Fig. 12. A new element with the value of 30 is to be inserted. First, it is placed

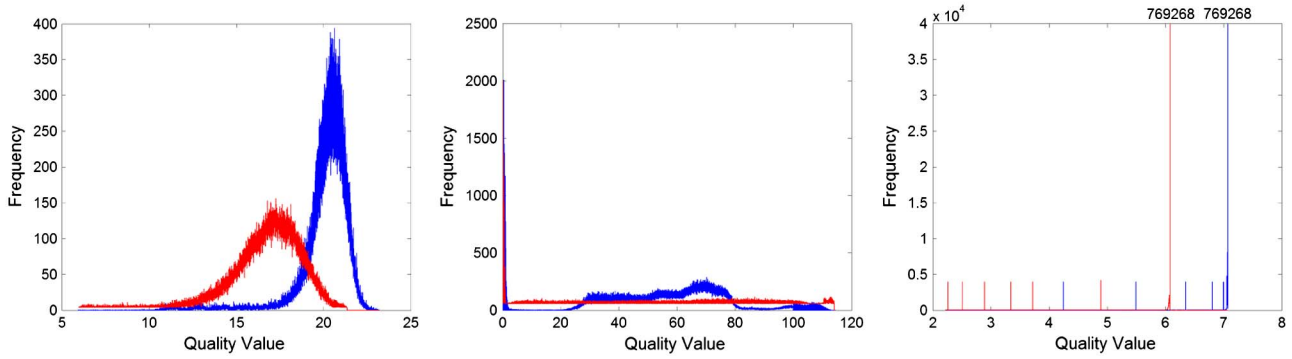


Fig. 10. Quality distribution curves comparison between the fixed and adaptive mapping. The blue curve represents the fixed mapping, and the red curve is the adaptive mapping. (a) Peaks, (b) coffee cup lid, and (c) toy duck.

Table 3. Comparison of Speed Among Refreshed Searching and Resumed Searching with Both Fixed and Adaptive Mappings

	Searching	Mapping	Level Number	Peaks	Coffee Cup Lid	Toy Duck
Time (ms)	Refreshed	Fixed	$(MN)^{1/2}$	36.28	436.27	63.29
	Refreshed	Adaptive	$(MN)^{1/2}$	32.26	417.36	61.57
	Resumed	Fixed	$16 \times (MN)^{1/2}$	23.04	138.19	51.37
	Resumed	Adaptive	$16 \times (MN)^{1/2}$	16.89	85.34	42.71

Table 4. Comparison among I2L2, I2L2-H, and Heap

		Peaks	Coffee Cup Lid	Toy Duck
Time (ms)	I2L2	16.89	85.34	42.71
	I2L2-H	12.84	66.19	54.27
	Heap	20.75	115.2	58.61

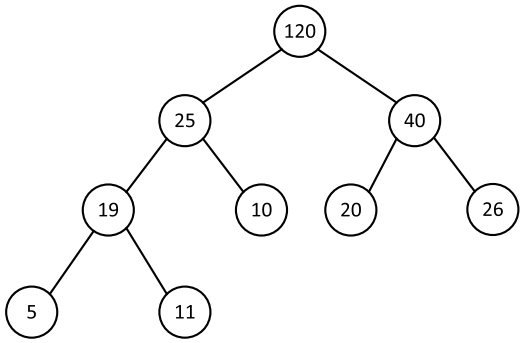


Fig. 11. Binary max heap data structure.

at the end of the heap [Fig. 12(a)]. It is then compared with its parent, whose value is 10. Because the new element is larger than its parent, they are swapped [Fig. 12(b)]. Then, the new element is continued for comparison with its new parent with a value of 25. Since the new element is still larger, they are swapped again. Finally, the new element is compared with the root element with a value of 120. Because the new element is smaller, the insertion stops [Fig. 12(c)]. Because the new element only recursively compared with its parents, a heap is faster than a linked list on inserting an element.

Deleting the root element is straightforward, but the heap order must be restored after the deletion. After removing the root element, the last element of the heap is selected as the new root. Then the new root is compared with its children. If the root is larger than its children, the operation terminates. If not, the root is swapped with its larger child and compared with its new children. Figure 13 demonstrates how to restore the order of a heap after the root element is removed. The last element with the value of 11 is chosen to be the new root [Fig. 13(a)]. Then it is compared with its children. Because the new root is smaller, it is swapped with its larger child with the value of 40 [Fig. 13(b)]. At last, the element is swapped with the element with the value of 26, and the restoring operation stops [Fig. 13(c)].

2. *Structure of I2L2-H*

Because a heap is faster than a linked list on inserting an element, in order to reduce the time of insertion, we propose to replace each short linked list by a heap. The new proposed structure is shown in Fig. 14 and named as I2L2-H in order to distinguish it from the previous I2L2. Nodes in heaps represent the pixels in the I2L2, and the values of nodes are the quality values. The root node of each heap is the largest pixel in the level, and the root

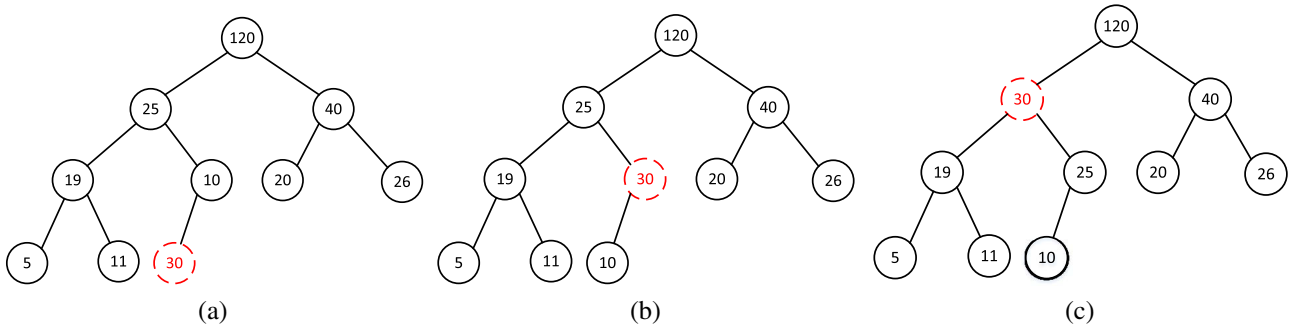


Fig. 12. Operations of inserting a new element. (a) The new element is inserted to the end of the heap. (b) and (c) Subsequent steps of moving the new element to the proper position.

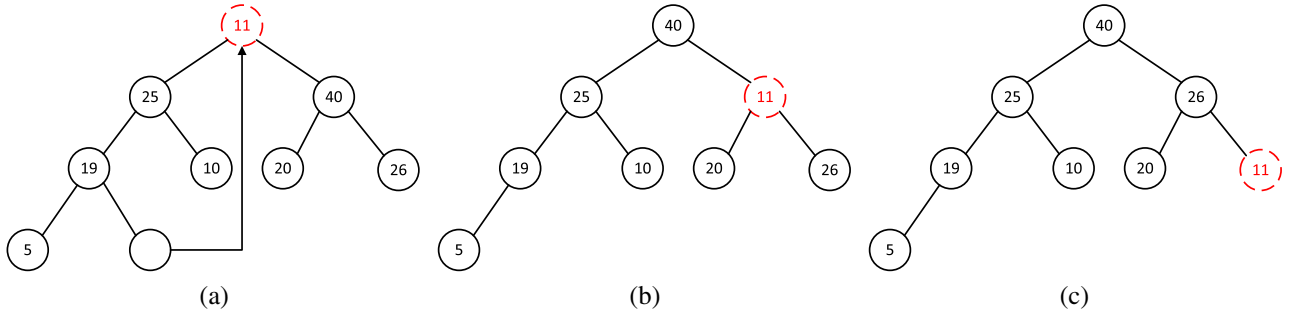


Fig. 13. Operations of restoring heap order after deleting the root element. (a) Replace the root with the last element. (b) and (c) Subsequent steps of restoring the order of the heap.

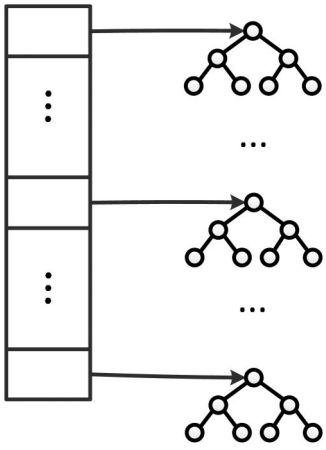


Fig. 14. Structure of I2L2-H.

of the highest nonempty level is selected to be unwrapped each time.

When a new pixel comes in for insertion, we first find its proper level in the I2L2-H, and then add it to the end of the corresponding heap. Then it is moved to its proper location following the inserting steps of a heap. To unwrap the pixel with the highest quality value, the root of the heap in the highest nonempty level is first selected and then removed from the heap. The heap restores its order following the deletion steps of a heap. Comparing with the I2L2, the time complexity of insertion is reduced from $O(n)$ of the I2L2 to $O(\log_2 n)$ of the I2L2-H, where n is pixel number in this level. However, the time complexity of removing one pixel is increased from $O(1)$ of the I2L2 to $O(\log_2 n)$ of the I2L2-H. The overall time consumption of the I2L2-H is expected to be lower than the I2L2.

It is noteworthy that, since a single heap data structure fulfills the requirements of the adjoin list mentioned in Section 1, the QGPU can also be implemented by a single heap [20,21]. Using a single heap for QGPU will be implemented and compared with both I2L2 and I2L2-H.

B. Comparison among I2L2-H, I2L2, and a Single Heap

The previous examples are used for comparisons. We use our own implementation of the heaps because it is about 30% faster than the implementation of the C ++ Standard Template Library. The results are shown in Table 4. For the peaks and coffee cup lid examples, the I2L2-H performs the best, as expected. However, for the toy duck example, the I2L2 performs better than the I2L2-H, due to its unusual quality value distribution. A single heap always performs the worst.

To summarize, these three data structures have their own advantages and drawbacks. With respect to concepts, the I2L2 converts 1D search to 2D and is intuitive. As for the heap, it is an existing data structure in textbooks, thus considered simple for programmers. I2L2-H is a hybrid between the I2L2 and the heap. With respect to implementation, the heap is less complex than the I2L2 and the I2L2-H.

Table 5. Comparison between the Original and Improved I2L2

		Peaks	Coffee Cup Lid	Toy Duck
Time (ms)	I2L2	36.28	436.27	63.29
	I2L2-H	12.84	66.19	54.27

Also, the heap has very rich resources on the Internet for researchers to utilize. With respect to performance, the I2L2-H is the best choice among the three data structures because it inserts elements faster than I2L2 and it needs less time to restore order than using a single heap.

C. Comparison between the Original and Improved I2L2

After three improvements introduced in previous sections, the final comparison results between the I2L2-H proposed in this paper and the original I2L2 proposed in [16] are listed in Table 5. We can find the improved I2L2 is over 6 times faster than the original I2L2 in the coffee cup lid example and nearly 3 times in the peaks example. Because of the unusual quality distribution of the toy duck example, the improved I2L2 is only 17% faster than the original one.

In addition, we can find that the I2L2-H is able to unwrap the coffee cup lid example and the toy duck example, which have 1 or more megapixels, in only 70 ms; that is, about 15 frames per second. It approaches the requirement of real-time phase unwrapping. For the peaks example, which has about a quarter of 1 megapixel, it achieves more than 75 frames per second, which exceeds the real-time requirement.

4. Conclusion

In this paper, three improvements of I2L2, the resumed searching, the adaptive mapping, and I2L2-H, are proposed. The resumed searching maintains a record of the highest nonempty level and reduces unnecessary comparisons. The adaptive mapping is used to solve the problem of concentrated distribution. It maps the quality values to the I2L2 levels dynamically instead of the original linear mapping and optimizes the quality value distribution. Heap is a data structure that is efficient on insertion. Thus heaps are used to replace the short linked list, resulting in the I2L2-H. From the experimental results, the I2L2-H performs better than the I2L2 and the heap, and it is over 6 times faster than the original I2L2 in the best case. We consider that the I2L2-H is a good choice to implement the adjoin list for phase unwrapping.

References

1. D. Malacara, *Optical Shop Testing*, 3rd ed. (Wiley, 2007).
2. M. Takeda and K. Mutoh, "Fourier transform profilometry for the automatic measurement of 3-d object shapes," Appl. Opt. **22**, 3977–3982 (1983).
3. V. Srinivasan, H. C. Liu, and M. Halioua, "Automated phase-measuring profilometry of 3-d diffuse objects," Appl. Opt. **23**, 3105–3108 (1984).

4. H. A. Zebker and R. M. Goldstein, "Topographic mapping from interferometric synthetic aperture radar observations," J. Geophys. Res. **91**, 4993–4999 (1986).
5. D. Ghiglia and M. Pritt, *Two-Dimensional Phase Unwrapping: Theory, Algorithms, and Software* (Wiley, 1998).
6. R. M. Goldstein, H. A. Zebker, and C. L. Werner, "Satellite radar interferometry: two-dimensional phase unwrapping," Radio Sci. **23**, 713–720 (1988).
7. X. Su and W. Chen, "Reliability-guided phase unwrapping algorithm: a review," Opt. Laser Eng. **42**, 245–261 (2004).
8. T. J. Flynn, "Two-dimensional phase unwrapping with minimum weighted discontinuity," J. Opt. Soc. Am. A **14**, 2692–2701 (1997).
9. D. C. Ghiglia and L. A. Romero, "Minimum L^p-norm two-dimensional phase unwrapping," J. Opt. Soc. Am. A **13**, 1999–2013 (1996).
10. G. Pedrini, I. Alexeenko, W. Osten, and H. J. Tiziani, "Temporal phase unwrapping of digital hologram sequences," Appl. Opt. **42**, 5846–5854 (2003).
11. J. M. Huntley and H. Saldner, "Temporal phase-unwrapping algorithm for automated interferogram analysis," Appl. Opt. **32**, 3047–3052 (1993).
12. H. Zhao, W. Chen, and Y. Tan, "Phase-unwrapping algorithm for the measurement of three-dimensional object shapes," Appl. Opt. **33**, 4497–4500 (1994).
13. J. Tian, X. Peng, and X. Zhao, "A generalized temporal phase unwrapping algorithm for three-dimensional profilometry," Opt. Laser Eng. **46**, 336–342 (2008).
14. C. Reich, R. Ritter, and J. Thesing, "White light heterodyne principle for 3d-measurement," Proc. SPIE **3100**, 236–244 (1997).
15. J. M. Huntley and H. O. Saldner, "Shape measurement by temporal phase unwrapping: comparison of unwrapping algorithms," Meas. Sci. Technol. **8**, 986 (1997).
16. M. Zhao, L. Huang, Q. Zhang, X. Su, A. Asundi, and Q. Kemao, "Quality-guided phase unwrapping technique: comparison of quality maps and guiding strategies," Appl. Opt. **50**, 6214–6224 (2011).
17. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. (MIT, 2009).
18. Q. Kemao, W. Gao, and H. Wang, "Windowed Fourier-filtered and quality-guided phase-unwrapping algorithm," Appl. Opt. **47**, 5420–5428 (2008).
19. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. (Prentice-Hall, 2006).
20. F. Li, Y. Zhan, D. Hu, and C. Ding, "A fast method for InSAR phase unwrapping based on quality guide," J. Radars **1**, 196–202 (2012) (in Chinese).
21. J. Wesner (personal communication, 2012). In an e-mail to the authors, dated 14 June 2012, Dr. J. Wesner asked whether we had tested using a heap for QGPU.