DOI : 10.3745/JIPS.2009.5.2.105

GPU-based Stereo Matching Algorithm with the Strategy of Population-based Incremental Learning

Dong-Hu Nie*, Kyu-Phil Han and Heng-Suk Lee****

Abstract: To solve the general problems surrounding the application of genetic algorithms in stereo matching, two measures are proposed. Firstly, the strategy of simplified population-based incremental learning (PBIL) is adopted to reduce the problems with memory consumption and search inefficiency, and a scheme for controlling the distance of neighbors for disparity smoothness is inserted to obtain a wide-area consistency of disparities. In addition, an alternative version of the proposed algorithm, without the use of a probability vector, is also presented for simpler set-ups. Secondly, programmable graphics-hardware (GPU) consists of multiple multi-processors and has a powerful parallelism which can perform operations in parallel at low cost. Therefore, in order to decrease the running time further, a model of the proposed algorithm, which can be run on programmable graphics-hardware (GPU), is presented for the first time. The algorithms are implemented on the CPU as well as on the GPU and are evaluated by experiments. The experimental results show that the proposed algorithm offers better performance than traditional BMA methods with a deliberate relaxation and its modified version in terms of both running speed and stability. The comparison of computation times for the algorithm both on the GPU and the CPU shows that the former has more speed-up than the latter, the bigger the image size is.

Keywords: *Image filtering, Performance Evaluation, General-Purpose Computation Based on GPU, GPU, Population-Based Incremental Learning*

1. Introduction

Binocular stereo-matching algorithms perform a group of operations repeatedly on each pixel point or block of pixel points to identify the corresponding points between the source image and the target image, which require a considerable amount of CPU time. Therefore, parallelism possibly benefits the efficiency of these algorithms. The graphics processor units (GPU) consist of multiple multi-processors, which are very efficient, to perform parallel computation. They were originally designed to give a fast rendering of geometric primitives for computer games and image generation. They are now available in all personal computers and many handheld devices. Current GPUs also support floating point arithmetic. Moreover, GPU performance has been improving at a faster rate than Moore's law [1], by about 2-3 times a year. However, we have to solve certain problems which arise in GPU implementation.

Recently, an evolutionary computation strategy known as the genetic algorithm has been used to solve stereo-matching problems [2-4]. However, the general problems with genetic algorithms, such as memory consumption and search inefficiency, are more critical the larger image size is. Han's algorithm [5] tried to solve those problems to some extent, but its memory cost was a little high due to the use of the probability vector. Then, Han et al. proposed a simplified version, which produced the same results as the initial version, to decrease the memory cost of a serial manner CPU algorithm, but it required at least three layers of loops to perform these computations. For more improvements, Han's algorithm is needed to consider that the GPU-based scheme should be designed to perform repetitive operations based on a group of vertices or fragments and a stereo-matching algorithm with a PBIL [6] strategy executed on commodity graphics card to solve these problems. Therefore, we describe the algorithm of the stereo algorithm using PBIL and simplify it by not using a probability vector. Then, the model of the algorithm run on GPU is given. Finally, we evaluate the implementation by comparing the runtime between the CPU and the GPU. The experimental results show that the GPU algorithm had more than twice the speedup of the CPU algorithm.

Manuscript received 1 October, 2008; first revision 24 February, 2009; second revision 8 April, 2009; accepted 25 May, 2009.

Corresponding Author: Kyu-Phil Han

* Coll. of Computer Science and Technology, Harbin Engineering University, Harbin, China (niedonghu@hrbeu.edu.cn)

** Dept. of Computer Engineering, Kumoh National Institute of Technology, Gumi, South Korea (kphan@kumoh.ac.kr, hslee@kumoh.ac.kr)

2. Related Work

The existing techniques for stereo matching are roughly grouped into two categories. One is feature-based and the

other is intensity or area-based. Since the feature-based methods use relatively sparse primitives, a complicated interpolation process including occlusion modeling and disparity continuity must be taken into account to obtain a dense disparity map. Conversely, intensity-based methods use dense low-level features and intensity values themselves, thus a feature extraction and an interpolation process are not necessary for a dense disparity map to be obtained; unfortunately, they are sensitive to noise and small differences of intensity. Consequently, other enhancements of stereo approaches using chromatic information [7, 8], windowed Fourier phase [9], and transformed images [10], etc. have been studied. However, these methods possess the nature of feature- and intensity-based techniques, thus the problems mentioned above occur in the enhancements as usual.

Genetic algorithms are efficient search methods based on the principles of natural selection and population genetics [11]. They have been used to solve the stereo matching problem. The matching environment is considered as an optimization problem in this approach and finds the optimal solution under a pre-defined condition. Since the matching and the relaxation processes are used at the same time in this method, there are some improvements in the output quality. However, general problems, such as memory consumption and search inefficiency occur in genetic algorithms. The problems are more critical the bigger the image size is. Han proposed a genetic stereo- matching scheme using PBIL. The PBIL is a modified search technique for genetic algorithms using stochastic search and competetive learning based on a probability vector (PV). Its structure is much simpler than that of other algorithms such as serial and parallel genetic methods. The algorithm can avoid using a gene pool, crossover and mutation, while preserving the important rules of evolution. Han's method improved search efficiency and the matching performance remarkably, but led to only a slight enhancement of the memory consumption because of the PV and loop structure with three layers. In other words, it has to perform kernel computation repeatedly. So, we can achieve higher speedup in parallel by using the GPU, which is composed of multiple multi-processors.

Since nVidia made the GPU highly programmable in 2001, the programmability of GPU has steadily increased. Many high level shader languages, such as Cg, Brook, and so forth, which can be used to make programs on the GPU, have also been developed. Moreover the cheaper cost and increasing rate more than Moore's law are also attracting more and more researchers to use the parallelism of GPU in both graphics and non-graphics applications. General purpose computations have become a popular area of research by using commodity GPU. Many classical algorithms in the areas of image processing and numerical computation, etc. have been implemented on GPU, such as FFT [12, 13], convolution [12], numerical algorithms [14], sparse matrix solvers [15] and so on. So far, no research has been conducted on a stereo matching algorithm running on GPU.

3. Stereo Matching with PBIL

3.1 Population-Based Incremental Learning

PBIL is a variation of genetic algorithms using a stochastic search based on a probability vector. It transforms the survival degree of a chromosome into a probability in the range of [0, 1]. In each generation, the production of chromosomes is based on the probability value. The generated chromosomes change their survival probabilities owing to the adaptation and disappear. In the next generation, chromosomes are also produced by the updated probability. Through the recursive process, if the probability converges, the process will be terminated. Therefore, unlike conventional genetic algorithms, it is not necessary for survival chromosomes to be saved. The probability vector holds all the survival information.

Fig. 1 shows the basic outline of PBIL, where l and n denote the number of all possible chromosomes and the size of the population, respectively. Initial probabilities of all possible chromosomes in the probability vector are set to 0.5 or $1/l$ in step 1), and then n chromosomes are randomly generated by a production function. The function generates two random numbers, which are the chromosome index and its survival probability. If the stored value in the generated index of the probability vector is greater than the generated survival probability, the chromosome can be produced. After the production step, the n chromosomes are evaluated by the fitness function and the probability vector is updated according to the fitness value. This process is called *learning*. A mutation altering the probability value may be inserted after the learning process. Finally,

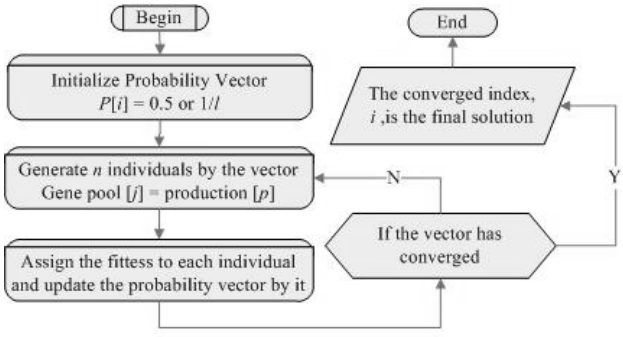


Fig. 1. Outline of PBIL.

the convergence of the vector is checked in step 5). If the vector has converged, the recursive process is terminated. Otherwise the process will be continued.

As seen in Fig. 1, the fittest or the winner, among the generated chromosomes in each generation updates the probability vector, and then disappears. Since the probability vector implies the survival information of chromosomes, it is not necessary to store the survival chromosomes explicitly. The algorithm using PBIL may be very compact if the solution space is small.

3.2 Probability Model

The objective of stereo matching is to find the corresponding point of a reference pixel in a target image. Since it is assumed that the epipolar constraint is satisfied, the search-range of the matching is limited to only the horizontal direction. If not, a rectification process is required before stereo matching can begin [16, 17]. A pixel, or a center of a block, in the reference image can be matched to a certain pixel on the target image within a given 1-D range, as shown in Fig. 2(a) where $fr(\cdot)$ and $ft(\cdot)$ are the gray levels of the reference and the target image, respectively. That is to say, each pixel of the target image within the search range has the probability of being matched to a reference one, as seen in Fig. 2(b). In this way, the matching probability vector of one point (0-dimension) has to be a 1-D array. Hence, the dimension of the probability vector can simply be expanded for 2-D images. The probability that the pixel (i, j) on the reference image matches to the pixel $(i, j+k)$ of the target image can be represented as $p(i, j, k)$. Thus, a total 3-D probability vector is needed for the 2-D image matching. In this probability model, stereo matching corresponds to finding the disparity k which has the maximum matched probability for all image points (i, j). That is, there is a 1-D probability vector per matching point, which is to be a pixel for dense matching or a center of a block for a skipped block matching algorithm (BMA).

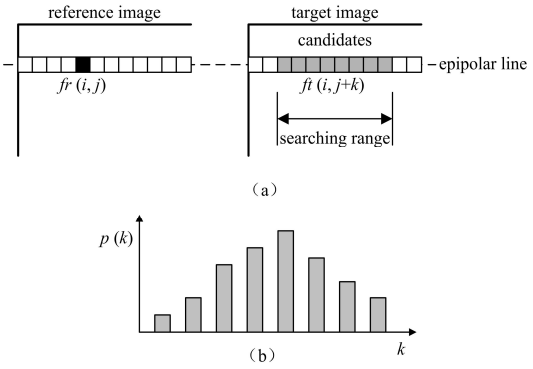


Fig. 2. Probability mapping for pixel matching. (a) General stereo matching environment (b) Matched probabilities of candidates.

In the PBIL model, the number of probability vectors depends on all possible solutions, so the space of the solutions may become immense according to the type of problem. However, the number of all possible solutions concerning a pixel in the stereo matching problem is the same as the size of the search range, thus is quite a small and finite number. Furthermore, since PBIL uses the chromosome index and its probability during evolution, the proposed stereo matching strategy using PBIL is very effective.

To obtain a disparity map from this model, the 3-D probability vector is initialized and updated by PBIL. Then, the disparity index having the maximum probability along the k-directions is selected as the solution for the pixel. The flow diagram of the matching algorithm is shown in Fig. 3. Steps 1) and 2) are equal to that of Fig. 1 except for its dimensions. Since each image point has an independent 1-D PBIL structure, the algorithm is executed as a raster-scanning procedure. The number of generated chromosomes in PBIL must be greater than 1 because the fittest must be selected. Also, the number depends on the convergence rate. The larger the n used, the faster the convergence rate obtained, and with less diversity. After production of the chromosomes, the generated chromosomes are evaluated by a problem-dependent function. In conventional GAs, after fitness allotment, the fitness values of all individuals should be transformed into survival probabilities according to the degree of their fitness, in order to prepare natural selection. This transformation is not a trivial task because it is difficult to assign a probability amount relative to cost or error measurements. However, in PBIL only the fittest needs to be identified as the winner by comparing the

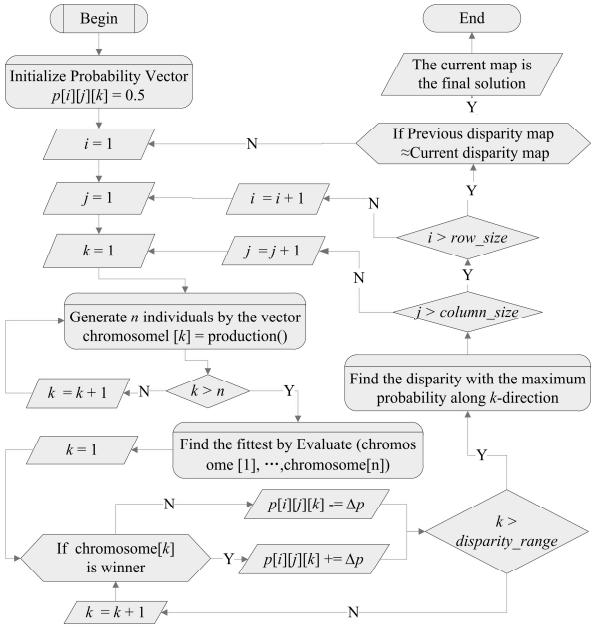


Fig. 3. Flow diagram of the proposed matching algorithm.

fitness values, which makes the matching structure simple because the transformation is not necessary. Additionally a constant learning rate is used in the probability update as shown in step 2). In step 3), the current disparity having the maximum probability in each generation is checked with that of the previous generation. If there is little or no change, then the recursive process is terminated.

3.3 Chromosome production function

The chromosome production process shown in Fig.3 based on the probability vector uses a random function. It generates two random numbers as mentioned in Section 2.1.1: One is a chromosome index denoting the disparity value, the other is its survival probability that will be generated. When a duplicated chromosome occurs or the when the probability of the vector is less than the survival one, another chromosome is produced. The flow diagram of the production stage is shown in Fig. 4, where MAX_NUM denotes the maximum number generated by the random function. In general, the chromosome index in PBIL differs from its chromosome itself. The index number only stands for a certain encoded chromosome. However, the chromosome index is equal to the chromosome value, i.e., its disparity value, in the proposed algorithm. That is to say, the chromosome structure of the proposed algorithm has only one gene and the gene value is the disparity.

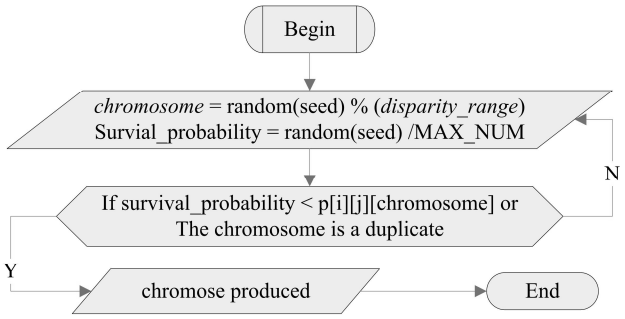


Fig. 4. Flow diagram of the chromosome production function.

3.4 Evaluation Function

To identify the winner of each generation, the evaluation function, or fitness function must assign a degree of fitness to the generated chromosomes. Therefore, it has to operate as a matching criterion function. In stereo matching, several constraints - such as intensity similarity and disparity smoothness - are commonly used to find the best match. The intensity similarity implies that the reference pixel or block is similar to the target one. The disparity smoothness denotes that disparities are smooth across neighboring pixels. Thus, the fitness function of the proposed algorithm is constructed by taking into account both similarity and smoothness terms. The mean of the absolute intensity difference between the reference and the target blocks is defined as the intensity similarity measure. Therefore, the mean intensity difference of the k-th chromosome at the considering point (i, j) is represented by

$$m_{ik}(i, j) = \frac{1}{S(W)} \sum_{(m,n)\in W} \sum \left| \begin{matrix} f_r(i+m, j+n) \\ - f_t(i+m, j+n+C_k) \end{matrix} \right| \quad (1)$$

where $S(W)$ is the size of window W, f_r and f_t are the intensity values, and C_k denotes the chromosome value, or disparity value, of the k-th chromosome. Another important constraint is disparity smoothness. There are many false matches in stereo matching when only using a similarity measure, so they should be carefully replaced with consistent disparities according to their adjacent values. The disparity ordering and uniqueness constraints may be inserted into the fitness function. The smoothness is only considered because it is a more dominant factor than the ordering constraint. And, also, the uniqueness term is automatically included in the proposed scheme. Since one winner along the disparity axis is selected at an image point, the matched point is unique. Therefore, we can say that three constraints are used in the fitness function of the proposed algorithm, which are intensity similarity, disparity smoothness, and uniqueness. If all disparities are known, the mean of the absolute disparity difference between the current and the 8-neighbor's disparities, i.e.

$$m_d(i, j) = \frac{1}{8} \sum_{m=-1}^{1} \sum_{n=-1}^{1} |d(i, j) - d(i+m, j+n)| \quad (2)$$
$$m \neq 0, n \neq 0$$

can be used as the measure of smoothness. However, since the proposed algorithm is executed in a raster-scan order, 4 disparities out of 8 neighbors, namely the lower 3 pixels and the right one of the current pixel, are not known at that moment, as shown in Fig. 5. Therefore, the disparities of the previous generation are used for the smoothness check of the current pixel, except in the first generation. The disparity having the maximum probability at each generation is temporarily stored and used for smoothness and convergence checking in the next generation. Also, in order to obtain a wide-area consistency of disparity and to include a coarse-to-fine strategy, the distance of a neighbor pixel is controlled by a scale factor, s, in the proposed scheme. Thus, the final disparity smoothness function about the k-th chromosome can be rewritten as

$$m_{dk}(i, j) = \frac{1}{8} \sum_{m=-1}^{1} \sum_{n=-1}^{1} |C_k - d^p(i+s\cdot m, j+s\cdot n)| \quad (3)$$
$$m \neq 0, n \neq 0$$

d	d	d	d	d	d	d	d	d
d	d	d	d	d	d	d	d	d
d	d	d	d	×	n	n	n	n
n	n	n	n	n	n	n	n	n
n	n	n	n	n	n	n	n	n

Fig. 5. Stereo matching sequence in raster-scan order, where × denotes the current matching point, and d and n are the pixels by which disparities are determined or not-determined at each point, respectively.

where d_p denotes the disparity value having the maximum probability at the pixel in the previous generation. The larger the scale factor used, the coarser the result obtained. Thus, the scale factor should be set to decreasing order to obtain a finer output. In the proposed algorithm, three steps of the scale factor, i.e., 4, 2, and 1, are used for a certain generation. Fig. 6 shows the 8 neighbors according to the scale factor in the sense of chess-board distance. The symbol X and each number denote the considering point and the scale factor, respectively. Four or more steps of scaling can be used to obtain a wide range of disparity.

The evaluation function of the *k*-th chromosome is defined as

$$E(i) = \begin{cases} m_{ik}, & \text{generation} = 1 \\ w_i m_{ik} + w_d m_{dk}, & \text{generation} > 1 \end{cases} \quad (4)$$

where w_i and w*d* are the weights of the similarity and the smoothness, respectively, and $w_i + w_d = 1$. In the first generation, the intensity similarity is only used because the previous disparity does not exist. Since Eq. (4) is composed of differences in intensity and disparity, it is referred to as an error function, and the fitness function about the error may be represented with the reciprocal of Eq. (4). Normally, a fitness value would be transformed into a survival pro

	4				4			4	
			2		2		2		
				1	1	1			
	4		2	1	**X**	1	2	4	
				1	1	1			
			2		2		2		
	4				4			4	

Fig. 6. Eight neighbors participating in disparity smoothness according to each scale factor.

bability in conventional GAs because of natural selection, as mentioned above. Since the relation between fitness and survival probability cannot be modeled clearly, there are many difficulties. However, this process of transformation from an error into a fitness and survival probability value is not essential in the proposed scheme, because only the fittest, or the winner, needs to be identified.

4. Simplified Scheme without PV

In this section, an alternative version of the proposed algorithm is presented to obtain the fastest convergence and the smallest memory space. If we allow all the possible chromosomes to always be generated at every generation, the convergence speed will be maximized while diversity will be minimized and the production function eliminated, because it can be assumed that all chromosomes have already been generated. This modification makes the proposed algorithm simple and fast. Next, in order to reduce the memory space, the 3D probability vector can be eliminated if the disparity is deterministically decided as the solution at each generation. The survival probability does not need to be saved in this case, thus this second modification is neither a PBIL nor a genetic algorithm. However, it will be shown that the performance due to these modifications is similar to that of the original version of the proposed algorithm in experiments. Fig. 7 shows the flow diagram of the alternative scheme including the two modifications.

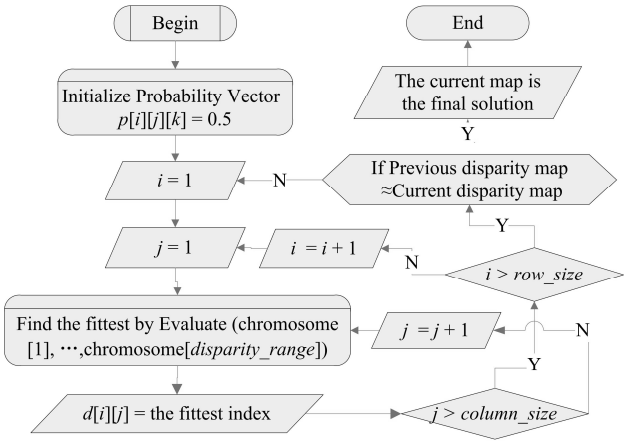


Fig. 7. Flow diagram of the alternative scheme of the proposed matching.

5. Overview of GPU

Seen from the GPU pipeline in Fig.8, vertices are passed to a vertex shader, which can compute positions, colors,

texture coordinates, and other attributes. These results are then interpolated to each fragment bounded by the resulting vertices. The interpolated results and textures are input into the fragment shader, which uses them to obtain the final fragment color. The graphics pipeline in Fig. 8 shows that computation on GPU is based on data streaming. The input data, vertex, texture, vertex index, etc. are represented as data streams. The kernels or shaders then input into the vertex processor and fragment processor perform computations on those data streams, including vertex, texture etc. A kernel or shader can execute similar computations on each record of those streams. Actually, we can easily think of GPU as SIMD.

The details at each stage of the GPU pipeline, such as the vertex processor, primitive assembly, interpolation, rasterization, fragment processor, etc. are not listed here, because one can find them in many materials about GPU. In the following, we will provide some information about texture as input data, which are helpful to understanding our algorithm.

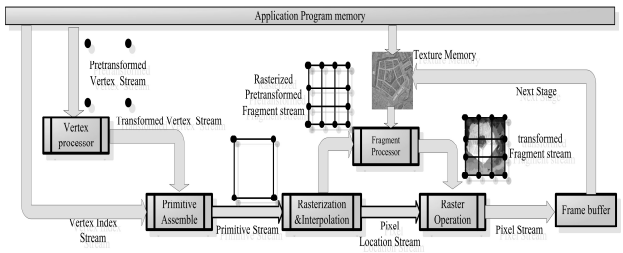


Fig. 8. Overview of the GPU Pipeline

5.1 Texture Coordinate and texture sampling

One-dimensional arrays constitute the native CPU data layout. Higher-dimensional arrays are typically accessed by offsetting coordinates in a large 1-D array. An example of this is the row-wise mapping of a two-dimensional array a[i][j] of dimensions M and N into the one-dimensional array a[$i \times M+j$].

For GPU, the native data layout is a two-dimensional array. Of course, one- and three-dimensional arrays are also supported. Arrays in GPU memory are called textures or texture samplers. Any data in textures can be obtained by sampling with the coordinates of that point.

The model for the proposed algorithm on the GPU comprises a few shaders. For each shader, multiple input textures of different sizes are used, but the viewport size (which is also the range of the sampling coordinate indices) is only the same as the output texture size. That being the case, for those input textures, the sampling coordinate for each point is different. Actually, the whole algorithm is separated into a few shaders according to the input and output texture size.

5.2 Texture Coordinate and texture sampling

Since GPU data stored in textures are updated by a rendering operation, a special projection that maps from the 3-D world (world or model coordinate space) to the 2-D screen (screen or display coordinate space) and which additionally performs 1:1 mapping between the pixel (to which we want to render) and the texel (from which we access data) are needed to precisely control the data elements for computing or accessing from the texture memory. The key to success here is to choose an orthogonal projection and a proper viewport that will enable 1:1 mapping between the geometry coordinates (used in rendering), the texture coordinates (used for data input), and the pixel coordinates (used for data output). The mapping is based on the only value that has been available to us so far, or the size (in each dimension) we allocate to the textures.

5.3 Texture Coordinate and texture sampling

One key functionality for achieving good performance rates is the possibility of using textures not only for data input, but also for data output. Internally, GPU schedule rendering tasks into several pipelines work in parallel, independently of each other. During the process of rendering into texture, the output texture can be used directly as the input texture of the next shader without data transfer from CPU again, which decreases the cost of transmission.

6. Modeling on GPU for the Proposed Algorithm

GPU comprises multiple multi-processors which can perform parallel algorithms. Thus, we have to change the algorithm for CPU into that for GPU. In order to implement the proposed GPU algorithm to obtain a disparity map, the algorithm has to be divided into a few independent computational kernels (each kernel is a shader). They are as follows: production, evaluation, learning and checking, except for initialization. The complete data flow diagram is shown in Fig. 9.

The input data of the input texture can be obtained only by sampling texture in the correct coordinates. Before each shader is executed, the attributes of the viewport have to be set. It has to retain the same size as output texture. That is to say, the viewport can be covered wholly by the output texture. In this way, the computation results of each shader can be output into the correct positions of the output texture. In addition, some of the shaders have multiple input textures whose sizes differ from the viewport. In this case, it means that the map of corresponding sampling coordinates between the input texture and the output

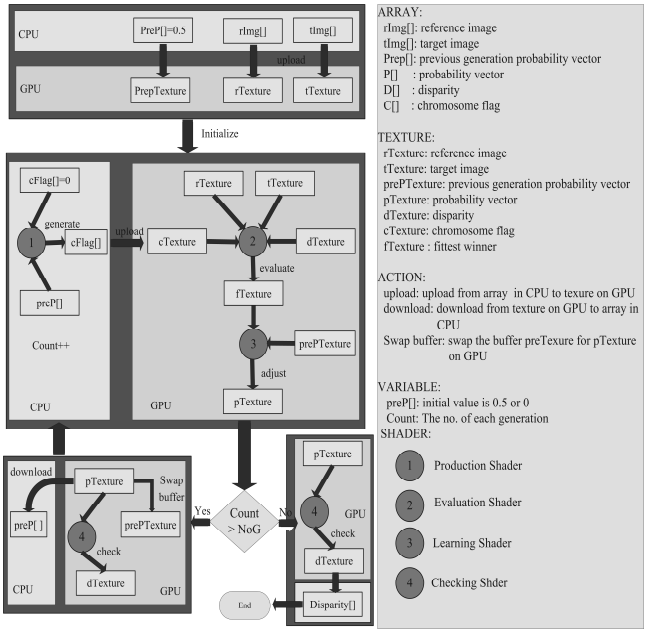


Fig. 9. The data flow diagram for PBIL stereo matching on GPU.

texture is multiple to one. Then, the input data can be obtained by transforming the sampling coordinates of the output texture.

Here, a cluster of alphabets with the postfix "[]" expresses an array, while the postfix "Texture" expresses a texture. ImgWidth and ImgHeight denote the width and the height of an image. DisparityRange denotes the range of disparity. In Fig. 9, rImg[] and tImg[] are arrays for the reference and target images, with the dimension of 1 by ImgWidth×ImgHeight. Prep[] and P[] are the arrays of the probability vectors for the previous and the present generation. C[] is the array for the chromosome flag, with a dimension of 1 by DisparityRange×ImgWidth×ImgHeight. D[] is the array for disparity with the dimensions of ImgWidth×ImgHeight. The rTexture, tTexture, preTexture, pTexture, dTexture, and cTexture are textures corresponding to the arrays mentioned above. The fTexture denotes the texture containing the fittest winner of each matching point. If the disparity is computed pixel by pixel, the dTexture and fTexture will have the same dimensions - ImgWidth by ImgHeight - as rTexture and tTexture. Their data layout is similarly shown in Fig. 10(a). If the disparity is computed using block by block with the size of BlockSize, the data layout of fTexture and dTexture is shown as Fig. 10(b). Another three textures have the same structure. Their size is expanded up to the dimension of DisparityRange from each matching point, as shown in Fig. 11(a).

By using some specific OpenGL instructions, the dimension of textures does not have to match a power of 2 in each dimension. However, for some cases, the dimension of three textures - preTexture, pTexture and cTexture - may

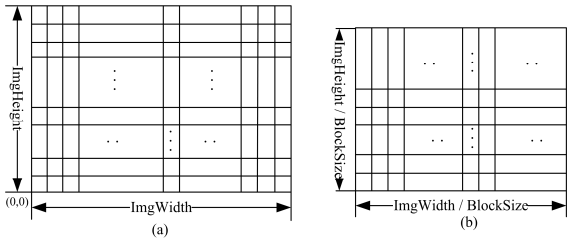


Fig. 10. The data layout of the initial image texture and the disparity texture: (a) Data layout of image, disparity and fittest winner texture with pixel by pixel dimension; (b) Data layout of disparity and fittest winner textures with block by block dimension.

exceed the maximum value supported by the graphics hardware. Currently, this maximum value is 2048×2048 or 4096×4096. To go beyond that, the following scheme is adopted:

For example, let the DisparityRange = $M \times N$, where M and N are integers and denote the maximal displacement of disparity in the direction of the vertical and horizontal respectively, and the maximum value supported by GPU is 4096×4096. If the constraints M×ImgWidth ≤ 4096 and N× ImgHeight ≤ 4096 can be satisfied, then $M \leq$ 4096/ImgWidth and $N \leq$ 4096/ImgHeight. The texture with the size of those textures = M×ImgWidth by N×ImgHeight can be supported by hardware. Fig. 11(b) shows the data layout of those three textures. If letting M = 1, which means a rectified stereo matching environments, then N = DisparityRange, and the texture layout is the same as that shown in Fig. 11(a). The texture data of Fig. 10(a) is expanded and stored row-wise with a dimension of 1 by DispaityRange, as shown in Fig. 11(a). However, in the case of 2-D search environments such as motion estimation, the data can be stored by a local block-wise texture with the dimensions of

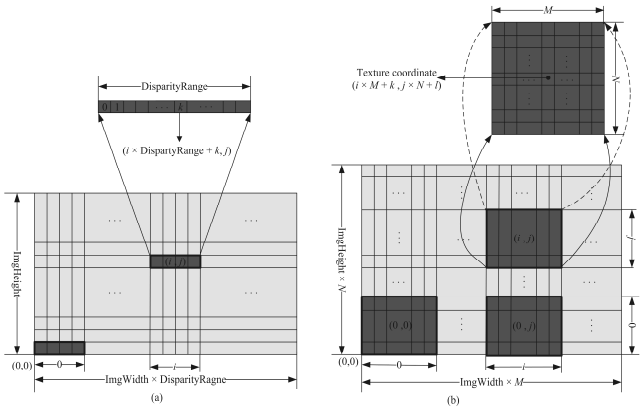


Fig. 11. Data layout of chromosome producing flag and probability vector textures for the previous and the present generation. (a) Arranged row-wise with dimensions of 1 by DisparityRange. (b) Arranged block-wise with dimensions of $M \times N$.

M by *N*, as shown in Fig. 11(b).

Suppose that the coordinate of one pixel point in the viewport is (*i*, *j*). The value of the vector corresponding to that matching point (shown in Fig. 11(a)) can be obtained by sampling with the coordinate of (*i*×DisparityRange+*k*, *j*), where *k* is opposite to the *k*-th disparity. In Fig. 11(b) the coordinate will become (*i*×*M*+*k*, *j*×*N*+*l*), and d = *k*×*M*+*l* will thus be opposite to the *d*-th disparity.

Uploading an array onto texture or downloading data into an array from texture can be implemented easily by using some OpenGL instructions, but transmission between the CPU and the GPU is costly. Therefore, it should be avoided as much as possible although it is necessary in some steps.

In the following sections, we will use (*i*, *j*) to denote the coordinate of a pixel in the viewport corresponding to the matching point (*i*, *j*).

6.1 Initialization

The initialization operation on the GPU is different from that used for the CPU. The reference and the target image data saved in the arrays rImg[] and tImg[] will be uploaded into rTexture and tTexture on the GPU by using a few OpenGL commands. The data layout of the two textures is shown in Fig. 10(a). The loop variable count is initialized to 0. The array Prep[] is the probability vector for all matching points in the image rather than for one matching point. It is first initialized to the value 0.5 and then has to be uploaded onto PrepTexture, for which the layout is shown in Fig. 11(a).

6.2 Shader1: Generating the Chromosome Flag Array

The computation in this section will use a random function which cannot be supported very well by the GPU, so the random function will still be computed on the CPU. For that reason, the routine should not be called a shader. However, because it is included in the loop of the proposed algorithm, as shown in Fig. 9, we call it a shader temporarily. The computed result is to assign value 0 or 1 into the array cFlag[]. The 1 denotes whether the chromosome for the next generation can be produced or not. The array cFlag[] is then uploaded onto cTexture on GPU as shown in Fig. 10.

6.3 Shader2: Evaluating the Fittest Winner

As shown in Fig. 9, the input of shader 2 includes four textures, rTexture, tTexture, cTexture and dTexture. The output is the fTexture. In this process, off-screen rendering is used. The result of shader 2 is not shown on-screen, but rendered into texture on the GPU. Thus, the output fTexture can be used as the input directly to shader 3, which avoids data transfer between the CPU and the GPU.

The evaluation is performed by computing similarity and disparity smoothness measures which are similar to those on the CPU. Therefore, the key is how to obtain data from all the input textures by sampling and to set the viewport size. Fig. 12 shows the data flow diagram of the computation of one matching point for shader 2. Since the computation is theoretically run in parallel on the GPU [16], other points are also processed in the same way. According to the rule in which the size of the viewport must be the same as the output texture, the coordinate range of shader 2 should be (0, 0) – (ImgWidth, ImgHeight). The three input textures of rTexture, tTexture, and dTexture have the same dimensions as the output texture. Then, we can identify whether their texture coordinate is a one-to-one mapping. However, the size of cTexture is *M*×*N* times of the output texture. As can be seen in Fig. 12, a local block of texture with the dimensions of *M*×*N* corresponds to a texel in fTexture. If the output texture coordinate is (*i*, *j*), then the coordinate of the texel attended for the computation for rTexture, tTexture, and dTexture, is also (*i*, *j*), but for cTexture it is a block of texture with the coordinates ranging from (*i*×*M*, *j*×*N*) to ((*i*+1)×*M*-1, (*j*+1)×*N*-1).

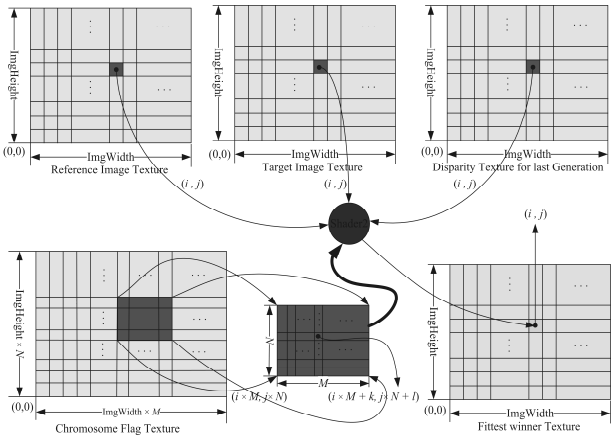


Fig. 12. Data flow diagram of shader 2.

The dTexture is the disparity texture map for the previous generation. In the first generation phase, the disparity smoothness measure has not yet been computed. Therefore, the dTexture will not be used until the 2nd generation, which is the result of the 1st generation phase.

6.4 Shader3: Adjusting Probability Vector

Shader 3 accepts the output texture of fTexture from shader 2, and prePtexture as the input texture for updating or adjusting the probability vector. Fig. 13 shows the data

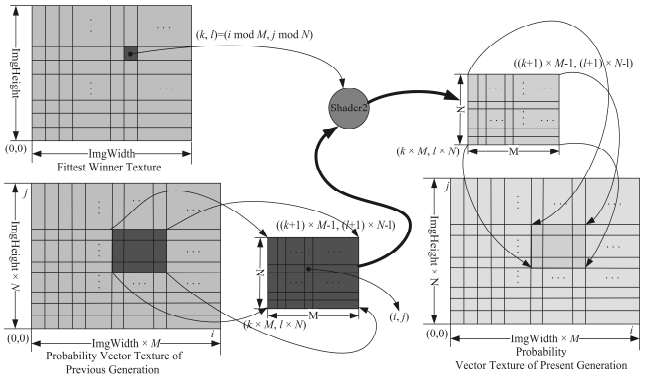


Fig. 13. The data flow diagram for shader 3.

flow diagram for shader 3. It is also using the off-screen rendering approach. As seen in Fig. 13, the relationship of the coordinates between prePtexture and pTexture is that of one-to-one mapping, and between fTexure and pTexture is that of one-to-a-block-of-texture mapping, with the coordinates ranging from (k×M, l×N) to ((k+1)×M-1, (l+1)×N-1). It is supposed that the output texture coordinate is (i, j), while the corresponding coordinate in prePTexture is also (i, j), but (i mod M, j mod N) for fTexture. Only to identify those relations between the input textures and the output texture, the shader can be run exactly.

Here, two textures for saving the probability vectors of the previous and the present generations are used, because the textures cannot be written and read at the same time in a shader. Therefore, before executing the next-generation computation, the buffers for the two textures are simply swapped by OpenGL instruction.

6.5 Shader4: Checking Disparity according to PV

Shader 4 checks the disparity for the fittest by using the updated probability vector textures from the output of shader 3. The index, k, at the point or block which has the maximum probability will be selected as disparity. The relationship of the coordinates between dTexture and pTexture is that of one-to-a-block-of-texture, with the coordinates ranging from (i×M, j×N) to ((i+1)×M-1, (j+1)×N-1). Fig. 14 gives the details of the processing steps for this shader.

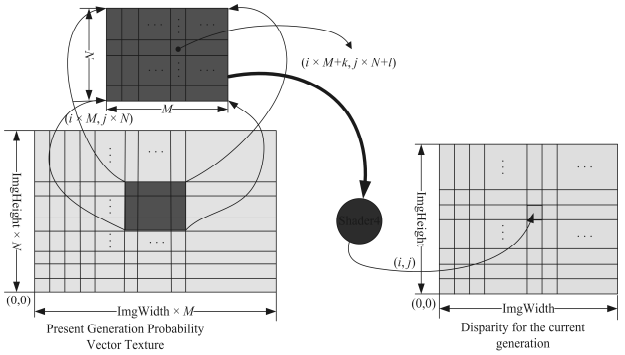


Fig. 14. The data flow diagram of shader 4.

6.6 Convergence Condition

If the loop count does not change or is more than the NoG (no. of generations), the whole computation ends or else the frame buffer swaps prePTexture for pTexture, and then downloads prePTexture into the array preP[], i.e. the data is transferred from the GPU to the CPU. The array prep[] will be used to produce chromosomes for the next generation.

6.7 Simplified Scheme

All chromosomes in each generation are produced in the alternative version, as mentioned in section 3.3. The simplified version does not need the probability vector, so there is no texture which has to be expanded in size. Thus, it is easy to implement in a shader. The details are not listed here for reasons of scope.

7. Experiments and Results

In order to evaluate the proposed algorithm, the conventional BMA with a relaxation scheme is compared with the proposed algorithm, because the strategy of the proposed scheme is very close to the BMA with relaxation. 30%, 50% random dot stereogram (RDS) images, “tsukuba” and “pentagon” are used in the experiments, as shown in Figs. 15 to 18. Experimental constants are shown in Table 1. 10% and 20% random noise are added to the 30% and 50% RDS and there is no added noise in the two real-world scene images. Three steps of distance for disparity smoothness checking, 4-, 2-, and 1-pixel, are applied in all experimental images. 0.8, 0.6 and 0.4 are used as the three values of wd and the number of generations for each step are set to 6, 4, and 2, respectively. It is also assumed that all the disparity candidates are generated to obtain the maximum convergence speed in all experiments. Fig. 19 shows the results of the conventional BMA with a 5×5 window. Fig. 20 shows the improved results with the relaxation process using fixed neighbors for disparity smoothness, which is a modified version [18] of the cooperative algorithm proposed by Marr and Poggio [19]. The results of Han’s algorithm are shown in Fig. 21. Fig. 22 shows each result of the proposed method without PV. As compared with the results of Fig. 20, the proposed algorithms clearly give faster and more stable results, due to the use of controlling smoothness distances of 8 neighbors from being coarse to be fine. The results of Figs. 21 and 22 are almost the same, where even the memory size of each scheme is definitely different. Table Ⅱ shows the comparison of the computation time for each step on an Intel(R) Pentium-4 PC with a

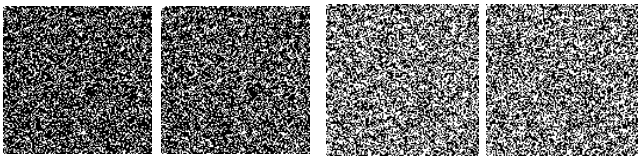
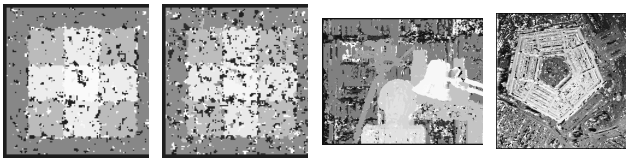


Fig. 15. 30% RDS. 10% dots of the right image are randomly décor-related.

Fig. 16. 50% RDS. 20% dots of the right image are randomly décor-related.

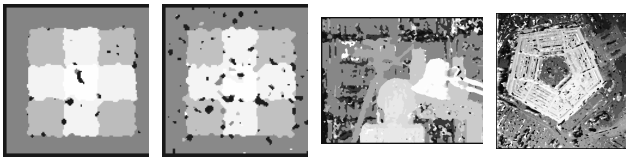
Fig. 17. "Tsukuba" stereo image pair.

Fig. 18. "Pentagon" stereo image pair.



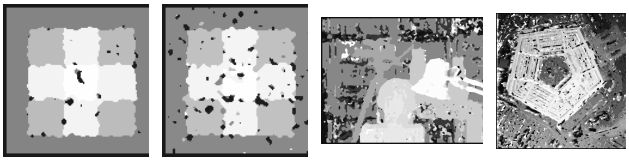
(a) (b) (c) (d)

Fig. 19. Conventional BMA results using a 5×5 window. (a) 30% RDS (b) 50% RDS (c) "tsukuba" (d) "pentagon".



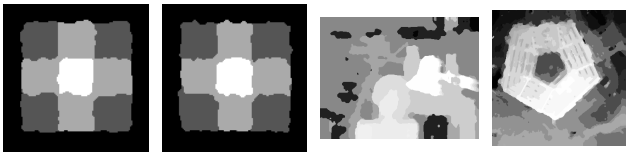
(a) (b) (c) (d)

Fig. 20. Relaxed BMA results after 12 iterations. (a) 30% RDS (b) 50% RDS (c) "tsukuba" (d) "pentagon".



(a) (b) (c) (d)

Fig. 21. The results of Han's algorithm after 12 generations. (a) 30% RDS (b) 50% RDS (c) "tsukuba" (d) "pentagon".



(a) (b) (c) (d)

Fig. 22. The results of the proposed algorithm after 12 generations. (a) 30% RDS (b) 50% RDS (c) "tsukuba" (d) "pentagon".

3.00GHz CPU and 512MB of main memory. It is shown that the proposed algorithm requires the least computation time among the three methods and produces good results compared to those of the original one. The good results of the full resolution PBIL-based method only require a little additional computing time compared to the relaxed-BMA, but provide significantly better results.

The experiment for the GPU-based approach was executed on an nVidia Geforce 7300GS graphic card with 256MB of memory. The fragment shaders were programmed with OpenGL and the Cg language. The proposed algorithm and its alternative on the graphics card were implemented as outlined above. Due to the GPU's features, the running speed was greatly improved for both the original and the proposed version. Table III shows a comparison of the running time on the GPU. Compared with Table II, Han's version of the GPU takes even less time than the simplified

Table 1. Experimental Constants

Methods / Images	Artificial Image		Real Image	
	30% RDS	50% RDS	"Tsukuba"	"Pentagon"
Image size	128×128	128×128	288×384	512×512
Image type	Gray	Gray	Color	Gray
Noise type	10% random	20% random	None	None
Actual disparity	0 ~ 3	0 ~ 3	About 0~20	About –10~10
Range of gene value (disparity range)	-4~7	-4~7	-10~30	-15~15
Window size for intensity difference	5×5	5×5	5×5	5×5
No. of steps of scale factor(s values)	3(4, 2, 1)	3(4, 2, 1)	3(4, 2, 1)	3(4, 2, 1)
No. of generation of each step (Total)	6, 4, 2(12)	6, 4, 2(12)	6, 4, 2(12)	6, 4, 2(12)
wd according to s (wi = 1 - wd)	0.8, 0.6, 0.4	0.8, 0.6, 0.4	0.8, 0.6, 0.4	0.8, 0.6, 0.4
Δ p	0.05	0.05	0.05	0.05

Table 2. Comparison of Computation Time on CPU (in seconds)

Methods / Images	BMA with 12 relaxations	Han's algorithm (12 generations)			Simplified algorithm (12 generations)		
		3-step	2-step	1-step	3-step	2-step	1-step
RDS	1.6	2.7	1.9	0.5	1.5	1.1	0.4
"tuskuba"	35.4	40.4	26.1	7.2	18.1	12.3	5.4
"pentagon"	75.1	76.5	49.8	14.5	39.7	28.5	12.2

Table 3. Comparison of Computation Time on GPU (in seconds)

Methods / Images	Han's algorithm (12 generations)			Simplified algorithm (12 generations)		
	3-step	2-step	1-step	3-step	2-step	1-step
RDS	2.02	1.54	0.50	1.05	0.64	0.41
"tuskuba"	33.48	15.25	6.23	16.78	7.77	4.05
"pentagon"	35.18	16.66	6.47	18.86	8.69	4.04

algorithm on the CPU. When the image sizes are smaller, the running time is similar, but when the image size is larger, more time is gained by implementing the GPU. For example, the computing time taken for the "pentagon" image at each step decreases by twice that taken by the CPU.

8. Conclusion

A hardware specific stereo matching algorithm using PBIL was presented in this paper to improve the running efficiency of genetic-based matching algorithms while preserving its compact structure. The stereo-matching problem has been modeled as a probability model and the PBIL strategy has been adapted for stereo matching. As a result, the matching structure of the proposed algorithm has been simplified and is comparable to a BMA, plus the relaxation scheme. Moreover, the 8-neighbor's distance participating in the disparity smoothness is gradually changed by 3 steps in order to obtain a wide-area consistency of disparities. Because of this distance control, the proposed algorithm can produce good results while using only a small fixed-size matching window of 5×5. It has been shown that the multi-step distance control for smoothness plays a dominant role in matching. At the same time, an alternative version including two modifications to the proposed algorithm, without using the probability vector and the production function, has also been presented for a simpler set-up. Experimental results have shown that the proposed matching algorithm improves both the convergence rate and the output quality over previous approaches. The alternative variant has been proposed as a good and effective choice with even less memory space and computation time cost.

In addition, an implementation exploiting the high performance of the GPU for a general purpose computation has been presented, which demonstrated significant performance improvements over the CPU-based algorithms. The experimental results using the GPU show savings in computing time by a factor of two over the CPU implementation for larger image sizes. Actually, GPU chips with higher speedup than the ones used in our experiment are now on sale, and would offer even better results.

References

[1] D.Luebke, M. Harris, J.Krüger, T. Purcell, N.Govind araju etc.. GPGPU: general purpose computation on graphics hardware, in ACM SIGGRAPH 2004 Course Notes, ACM, New York, NY, 2004, pp.33.

[2] P. H. Winston, Artificial Intelligence-3rd edition, New York: Addison-Wesley Publishing Co., pp.505-528, 1993.

[3] Fang-Chih Tien, Te-Hsiu Sun.Solving Line-Feature Stereo Matching with Genetic Algorithms in Hough Space..Journal of the Chinese Institute of Industrial Engineers,Vol.21, No.5, pp.516-526, 2004

[4] Régis Vaillant and Laurent Gueguen. Genetic algorithms applied to binocular stereovision. Computer Vision-ECCV '94. Vol.801, pp.193-198, 2006

[5] Kyu-Phil Han, "A Simple Stereo Matching Algorithm Using PBIL and its Alternative," Korea Information Processing Society, Vol.12-B, No.4, pp.429-436, Aug., 2005.

[6] Shumeet Baluja, "Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," Technical reports CMU-CS-94-163, Carnegie Mellon Univ., Jun., 1994.

[7] Q. Yang, L. Wang, R. yang, H. Stewenius, and D. nister, "Stereo Matching with Color-Weighted correlation, hierarchical Belief Propagation and occlusion Handling", Proceedings of the 2006 IEEE Computer society Conference on Computer Vision and Pattern recognition (CVPR'06), 2006, pp.2347-2354.

[8] John R. Jordan and Alan C. Bovik, "Using Chromatic Information in Edge-based Stereo Correspondence," CVGIP: Image Understanding, Vol.54, No.1, pp.98-118, 1991.

[9] John (Juyang) Weng, "Image Matching Using the Windowed Fourier Phase," International Journal of Computer Vision, Vol.11, No.3, pp.211-236, 1993.

[10] Yong-Suk Kim, Jun-Jae Lee, and Yeong-Ho Ha, "Stereo Matching Algorithm Based on Modified Wavelet Decomposition Process," Pattern Recognition, Vol.30, pp.929-952, 1997.

[11] Kyu-Phil Han, Kun-Woen Song, Eui-Yoon Chung, Seok-Je Cho, and Yeong-Ho Ha, "Stereo Matching Using Genetic Algorithm with Adaptive Chromosomes," Pattern Recognition, Vol.34, No.9, pp.1729-1740, 2001.

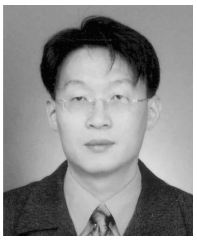
[12] Ondřej Fialka and Martin Čadik. CadÍk, "FFT and Convolution Performance in Image Filtering on GPU," Proceedings of the 10th International Conference on Information Visualisation, Los Alamitos, IEEE Computer Society, pp.609-614, 2006.

[13] Kenneth Moreland and Edward Angel, "The FFT on a GPU," SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003 Proceedings, San Diego, pp.112-119, 2003.

[14] J. KRÜGER AND R. WESTERMANN, Linear algebra operators for GPU implementation of numerical algorithms, in ACM SIGGRAPH 2005

Courses, ACM, New York, NY, 2005, p.234.
[15] J. BOLZ, I. FARMER, E. GRINSPUN, AND P. SCHRO ODER, Sparse matrix solvers on the GPU: conjugate gradients and multigrid, in ACM SIGGRAPH 2003 Papers, ACM, New York, NY, 2003, pp.917–924.
[16] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, R. Koch, "Visual Modeling with a Hand-held Camera," International Journal of Computer Vision, Vol.59, No.3, pp.207-232, 2004.
[17] D. V. Papadimitriou and T. J. Dennis, "Epipolar Line Estimation and Rectification for Stereo Image Pairs," IEEE Transactions on Image Processing, Vol.5, No.4, pp.672-676, 1996.
[18] Kyu-Phil Han, Tae-Min Bae, and Yeong-Ho Ha, "Hybrid Stereo Matching with a New Relaxation Scheme of Preserving Disparity Discontinuity," *Pattern Recognition*, Vol.33, No.5, pp.767-785, 2000.
[19] D. Marr and T. Poggio, A Computational Theory of Human Stereo Vision, Proc. Royal Soc. London, Vol. B204, pp.301-328, 1979.

Dong-Hu Nie
He received his B.S. degree in Econom-ics and h isM.S. degree in Computer Application from Harbin Engineering University, Harbin, China in 2001 and 2004 respectively. He has been a full-time lector at Harbin Engineering University since 2004. He is now studying as a Ph.D. course student at the Department of Computer Engineering of the Kumoh National Institute of Technology, Gumi, Korea. His interests lie in the fields of digital image and speech processing, computer vision, underwater acoustic signal processing. He is a member of the China Computer Federation and the Heilongjiang Province Computer Society.



Kyu-Phil Han
He received his B. S. and M. S., Ph. D. degrees in Electronic Engineering from Kyungpook National University, Daegu, Korea, in 1993 and 1995, and 1999, respectively. His main interests include digital image processing, image-based rendering, computer vision, and augmented reality. He was a researcher at Sindo Ricoh Advanced Research Institute, Seoul, Korea, from 1995 to 1996. He was awarded a bronze prize in the 5th Samsung Humantech Thesis Competition in Feb. 1999. In March 2000, he joined the School of Computer Engineering of the Kumoh National Institute of Technology, Gumi, Korea, as a full-time instructor, and is now an associate professor. He studied as a research professor in CG lab at the University of California, Irvine, from 2004 to 2005. He is a member of the Institute of Electronic Engineers of Korea, the Korean Institute of Communication Sciences, and the Korean Society for Imaging Science and Technology.

Heng-Suk, Lee
He received his BS, MS and Ph.D. degrees in Computer Engineering from Kumoh National Institute of Technology, Gumi, Korea, in 2001, 2003 and 2007, respectively. He is a contract professor in the School of Computer Engineering at the Kumoh National Institute of Technology, Korea. His research interests lie in the fields of Image Processing, Computer Vision, and Computer Graphics.