J Sign Process Syst (2013) 72:17–33
DOI 10.1007/s11265-012-0691-5

Real-Time 3D Depth Generation for Stereoscopic Video Applications with Thread-Level Superscalar-Pipeline Parallelization

Guo-An Jian · Cheng-An Chien · Peng-Sheng Chen · Jiun-In Guo

Received: 24 August 2011 / Revised: 18 May 2012 / Accepted: 29 August 2012 / Published online: 12 September 2012

Abstract We propose a low-complexity algorithm for stereoscopic video applications that generates a high-quality 3D image depth map from a single 2D image. Based on their characteristics, 2D images are classified into one of three categories before being processed by the proposed low-complexity algorithm to generate corresponding depth maps. We also extend the 3D depth algorithm to construct a parallel 3D video system. A thread-level superscalar-pipelining approach is developed to parallelize the 3D video system. Experimental results for HD1080 resolution images demonstrate that the algorithm can generate high-quality depth maps with an average reduction in the computational complexity of 98.2 % compared with a conventional algorithm. The parallel 3D video system can achieve a processing speed of 63.66 fps for HD720 resolution video.

Keywords 3D image/video · Depth map · Stereo · Parallel computing

G.-A. Jian · C.-A. Chien · P.-S. Chen (✉)
Department of Computer Science and Information Engineering, National Chung Cheng University,
Chia-Yi, Taiwan
e-mail: pschen@cs.ccu.edu.tw

G.-A. Jian
e-mail: chienka@cs.ccu.edu.tw

C.-A. Chien
e-mail: cca95m@cs.ccu.edu.tw

J.-I. Guo
Department of Electronics Engineering, National Chiao Tung University,
HsinChu, Taiwan
e-mail: jiguo@nctu.edu.tw

1 Introduction

3D images/videos are becoming increasingly popular because people desire more realistic visual effects than those currently provided by 2D image/videos. The major difference between 2D and 3D content is depth map information, which represents the relative distances of individual objects in 2D content with respect to the viewer. Consequently, depth map information is important for 3D display technology.

Several methods have been proposed for generating depth maps. Cheng et al. [1] used two or more images from different views to generate a depth map; this technique requires using a stereo camera to simultaneously obtain images from different views. Most commercial cameras are only capable of obtaining images from a single view, but it is considerably more difficult to obtain an accurate depth map from a single 2D image than from multiple 2D images. Thus, considerable researches have been done to generate depth maps from single 2D images and several methods have been proposed to achieve this, including image classification [2, 3], vanishing point (VP) detection [2, 4], mean shift segmentation [5], and a post-processing method using a joint bilateral filter (JBF) [6]. However, all these methods suffer from high computational complexity, which makes them difficult to apply in real-time applications. Thus, there is a strong demand to develop a low-complexity algorithm for generating depth maps from single images.

In this study, we investigate the characteristics of 2D images and propose a low-complexity algorithm for generating depth maps. Experimental results for HD1080 images demonstrate that the proposed algorithm can generate high-quality depth maps with an average reduction in the

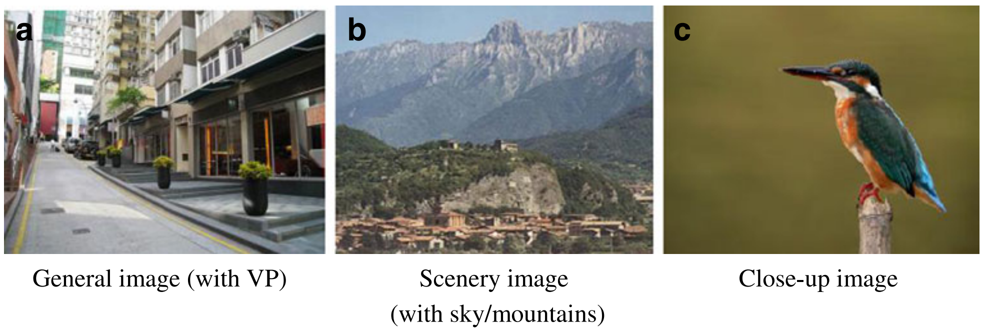


Figure 1 Sample images of the three image categories defined in the proposed algorithm.

computational complexity of 98.2 % compared with a conventional algorithm. In addition, the proposed algorithm is highly feasible for both hardware and software implementation, which makes it suitable for a variety of stereo applications.

We construct a parallel 3D video system based on the proposed 3D depth algorithm. This system can accept various video formats (after decoding by a video decoder), and it can generate the corresponding depth information. Although the 3D depth algorithm has low complexity, the whole system still requires a large amount of computation to achieve real-time processing. In the past few years, single-core processors have been gradually replaced by multicore processors that employ parallelization technology [7, 8], which give significantly improved performance. Accordingly, we adopt a thread-level superscalar-pipeline parallelization approach to implement the proposed algorithm on multicore processors. Experimental results demonstrate that the 3D video system can achieve a processing speed of 63.66 fps for HD720 resolution.

1.1 Contributions

This paper makes the following contributions:

- **Algorithm**. We proposed a low-complexity depth map generation algorithm for real-time 3D applications. The proposed algorithm is described in detailed. The computational complexity of the proposed algorithm is appropriately represented in a target-independent way, so that it could be a good reference for the evaluation of the proposed algorithm being considered to be realized in different platforms.
- **Implementation**. The algorithm was implemented by a thread-level superscalar-pipelining approach for real-time processing. Besides the parallel programming model, two synchronization approaches were developed to appropriately handle synchronization issues. These approaches are beneficial to program developers to reduce the synchronization problems when developing multi-thread parallel programs for video processing applications.

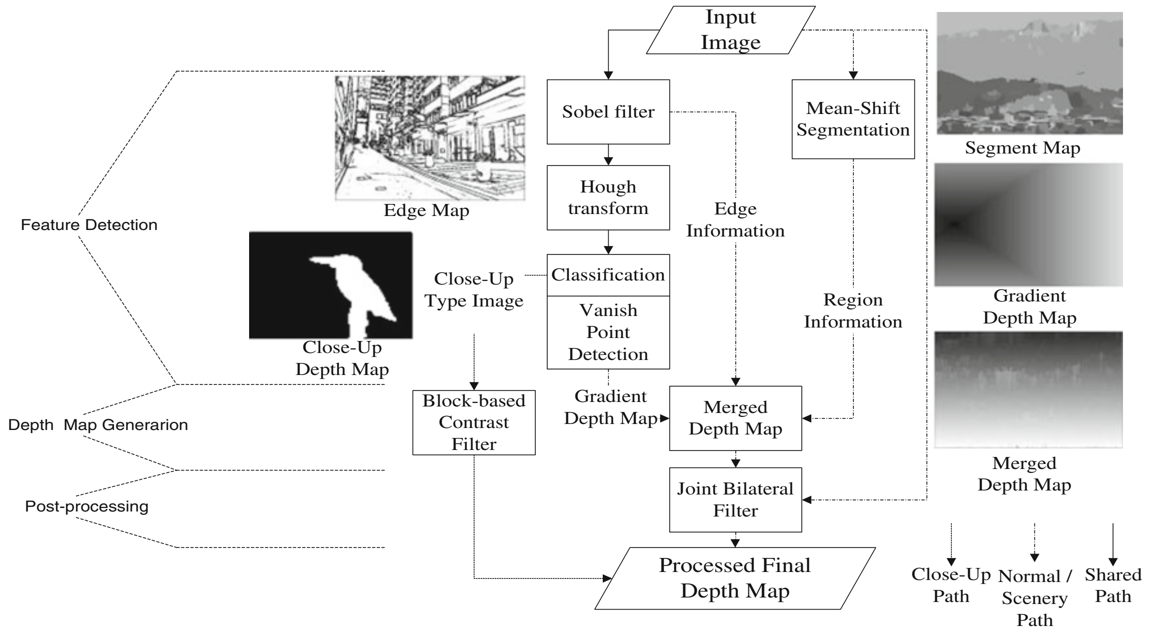


Figure 2 Proposed original depth map generation algorithm.

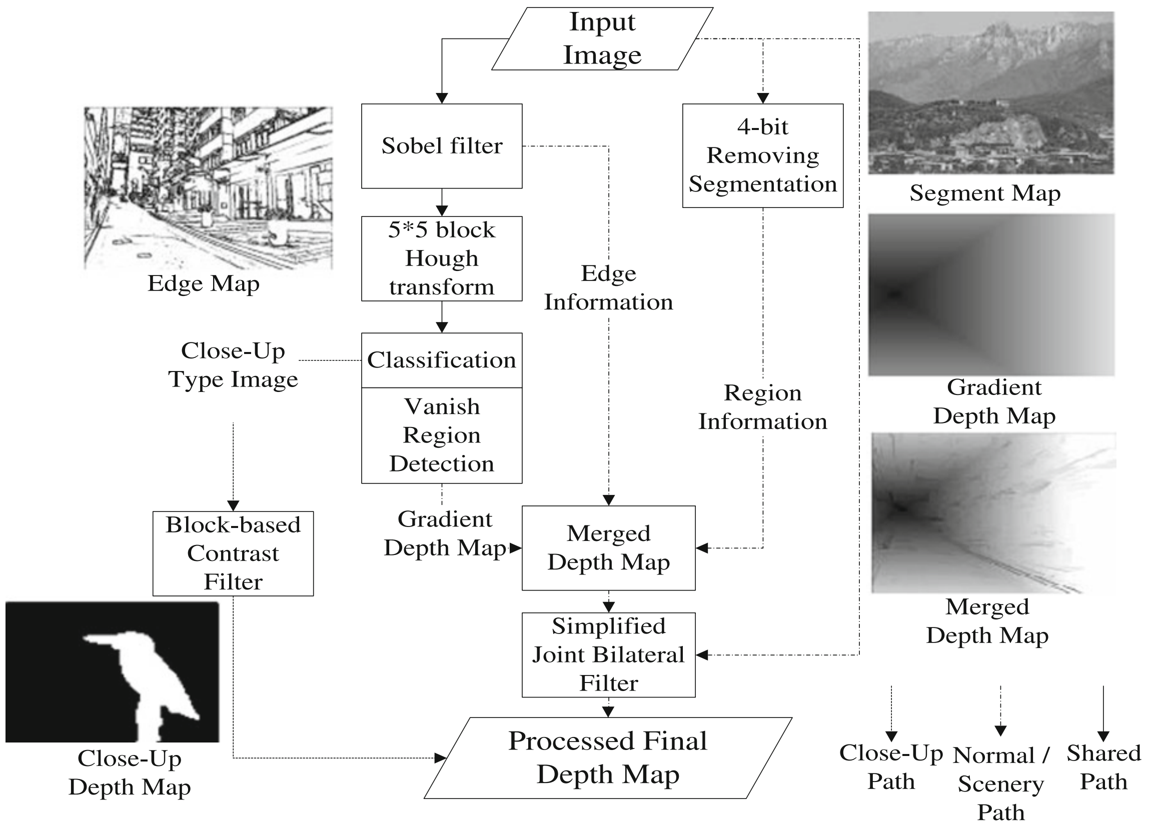


Figure 3 Proposed low complexity depth-map generation algorithm.

- **Experimental results**. The parallel depth map generator was integrated with a 3D video system for the evaluation. Experimental results for HD1080 resolution images demonstrate that the algorithm can generate high-quality depth maps with an average reduction in the computational complexity of 98.2 % compared with a conventional algorithm. The parallel 3D video system can achieve a processing speed of 63.66 fps for HD720 resolution video.

The remainder of this paper is organized as follows. Section 2 describes the proposed low-complexity depth-map-generation algorithm, and Section 3 describes the thread-level superscalar-pipeline parallelization approach adopted in this study. Section 4 reports the results of simulations and measurements of the quality of the algorithm, as well as a performance evaluation of the thread-level superscalar-pipeline approach. Finally, conclusions are presented in Section 5.

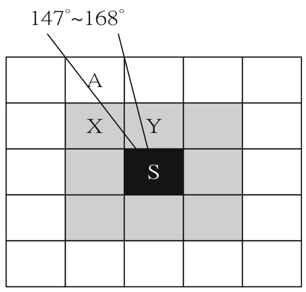


Figure 4 Example of a 5×5 Hough transform in the proposed algorithm.

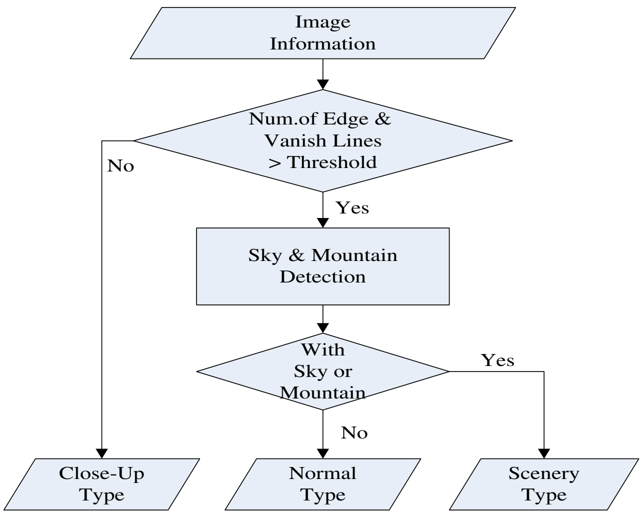


Figure 5 Proposed image classification approach.

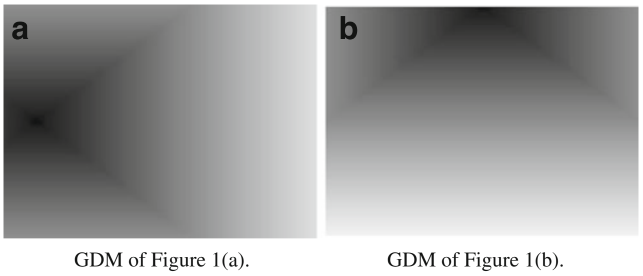


Figure 6 GDMs of the images in Fig. 1(a) and (b).

2 Depth Map Generation Algorithm

We observed that the characteristics of a single 2D image dramatically affect the computational complexity when generating a depth map from it. The processes employed in the generation algorithm can be adjusted based on the characteristics of an image to simultaneously realize a low computational complexity and generate a high-quality depth map. This concept represents the design philosophy of the proposed algorithm. The algorithm can operate on three types of input images: images of general scenes with a vanishing point (VP), images of scenery containing sky and mountains, and close-up images. Figure 1 shows a sample image for each category. Images are classified based on several image characteristics. Images containing few vanishing lines are identified as close-up images. Images containing a sufficient number of vanishing lines are classified as general images. Finally, images containing a sufficient number of vanishing lines and sky and mountains are classified as scenery images. Figure 2 depicts the original proposed algorithm for generating depth maps from single 2D images. The algorithm classifies input images according to the three categories and generates depth maps based on the following techniques: edge detection using Sobel filtering, line detection using Hough transform, vanishing region detection, mean shift segmentation, depth map merging, depth map post-processing by joint bilateral filtering (JBF), and block-based contrast filtering for identifying foreground objects in close-up images.

To make the algorithm suitable for real-time applications, we optimized these processing steps to reduce the computational complexity while preserving high image quality. Figure 3 shows the proposed low complexity depth map generation as compared to the original one (shown in Fig. 2). In the proposed algorithm, we simplify the complexity of Sobel edge detection, Hough transform, mean shift segmentation, and JBF to make it suitable for embedded applications. When reducing the algorithm complexity, we need to face the design trade-off in balancing the depth map quality and complexity. In fact, it is hard to measure the quality of depth map during the process of developing the proposed depth generation algorithm because stereo effect for 3D content is a perceptual feeling. Therefore, we try to balance between the complexity and the quality in a heuristic way. Each time when we decide to simplify some operation, we perform subjective quality measurement that adopts DSCQS (Double Stimulus Continuous Quality Scale) established by ITU-R and widely used in image and video research area. With an acceptable subjective quality as compared to the existing designs like [9, 10], we repeat the complexity reduction until all of the main computational intensive components are optimized. Below, we describe each processing step in detail.

2.1 3×3 Sobel Filter

A Sobel filter is used to obtain edge information from the input image, which is used to detect vanishing lines in the next step. The algorithm employs a 3×3 Sobel filter. The input pixels for the Sobel filter can be the average value of pixels in RGB format or the Y pixels in YUV format. In Sobel filtering, we apply horizontal and vertical filters to respectively obtain the values of G_x and G_y, as expressed by (1). We next compute the final value of G from G_x and G_y by using (2). To construct an edge information map, we set the current pixel to 1 if the value G exceeds a certain threshold and to 0 otherwise. After Sobel filtering, the algorithm obtains edge map information (see

Figure 7 Visual comparison between the mean shift segmentation and the proposed 4-bit segmentation.

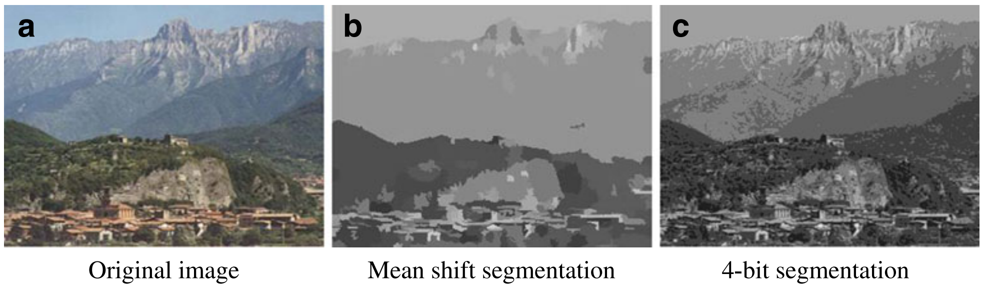


Fig. 3), which it uses to detect vanishing lines in the next step.

$$\frac{(R+G+B)}{3} * \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = Gx \quad , \quad \frac{(R+G+B)}{3} * \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} = Gy \tag{1}$$

$$G = \sqrt{(Gx^2 + Gy^2)} \tag{2}$$

To reduce the computational complexity of Sobel filtering in (2), we replace the square and square root operations by the sum of the absolute value operation, as shown in (3), since we found that the results generated by (3) are similar to those generated by (2). This reduces the computational complexity of Sobel filtering by about 65 %.

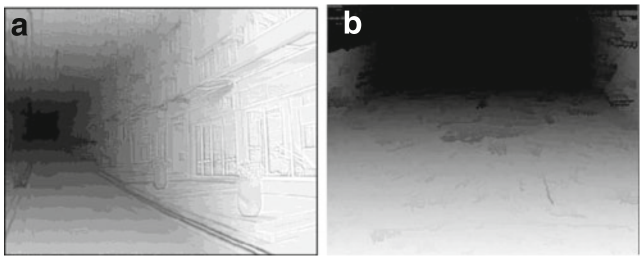
$$G = |Gx| + |Gy| \tag{3}$$

2.2 5×5 Simplified Hough Transform

The Hough transform is used to detect lines in the images. The input data for the Hough transform is the edge map from the output of the Sobel filtering. The original Hough transform algorithm [11] detects lines by searching all pixels between 0° and 180°. The coordinates of each pixel are transformed from Cartesian coordinates (x, y) to polar coordinates (ρ, θ) by using the following expression:

$$x\cos\theta + y\sin\theta = \rho. \tag{4}$$

The appearance at each set of (ρ, θ) values is then recorded. To reduce the complexity of the Hough transform, we used a 5×5 block as the basic processing unit and detected only the pixels with non-zero values in the edge map. Figure 4 shows an example in which pixel S is the current pixel with a non-zero value. We first search all the white pixels to check whether any of them has a value of 1. If pixel A has a value of 1, we will further search the gray pixels X and Y. If pixel X or Y has a value of 1, we perform the Hough transform to obtain the coordinates (ρ, θ) for between 147° and 168° in 1° intervals. Using the 5×5 Hough transform reduces the complexity by about 56 % because it is not necessary to search all the pixels between 0° and 180°. After processing all the pixels in the edge map, the algorithm sorts these (ρ, θ) data and selects the line with the most pixels as the vanishing line. On average, the algorithm records about 24 lines as vanishing lines in an image, which are sufficient to determine the vanishing region.



Depth map for Figure 1(a). Depth map for Figure 1(b).

Figure 8 Final depth maps for images in Fig. 1(a) and (b).

2.3 Vanishing Region Detection with Classification

To reduce the complexity of detecting the VP, we propose the concept of vanishing region detection (VRD) for detecting the vanishing region in the image based on the vanishing line information obtained from the Hough transform. The reason for detecting the vanishing region instead of the VP is that the vanishing region is usually close to its associated VP and the vanishing region can be located from fewer vanishing lines. The algorithm first determines the image classification based on the vanishing line information and it then detects the vanishing region in terms of the image classification. Figure 5 shows the proposed image classification approach. First, the algorithm determines the number of points and lines obtained by applying Sobel filtering and the Hough transform. If this number is less than a predefined threshold, the image is classified as a close-up image. The algorithm then analyzes the image to detect whether it contains sky and mountain. If it does, it is classified as a scenery image. Otherwise, the image is classified as a general image.

In VRD, the procedure used to detect the vanishing region depends on the classification of the image. For general images, all intersection points of vanishing lines are calculated and then an 8×8 region is used to group the nearest

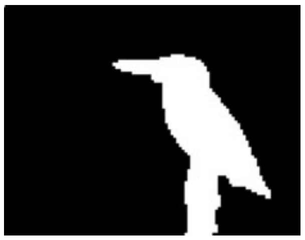


Figure 9 Depth map for image shown in Fig. 1(c).

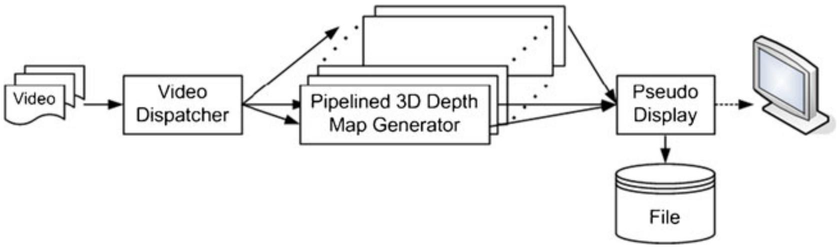


Figure 10 Proposed parallel 3D video system.

points in the image; this region is defined as a vanishing region. A vanishing region may be located inside or outside the image. If the vanishing region is outside the image, it is replaced by a new one located on the closest image margin. For general images, the algorithm generates a gradient depth map (GDM) based on the distance between each pixel and the vanishing region. Figure 6(a) shows the GDM for the image in Fig. 1(a). Since the sky and mountains in scenery images are usually at the top of the images, we set the top of these images as the vanishing region to compute the GDM. Figure 6(b) shows the GDM of the image in Fig. 1(b).

2.4 4-Bit Segmentation

Segmentation is used to identify objects inside images. Mean shift segmentation is the most commonly used segmentation algorithm in 3D image processing [5]. Mean shift segmentation improves the quality of the depth map, but it increases the computational complexity, making it difficult to apply in real-time applications. For this reason, we employ color-based grouping that involves extracting 4-bit color from each pixel and grouping pixels with similar colors into the same object. This 4-bit segmentation reduces the computational complexity by about 99.8 % relative to that using mean shift segmentation. Figure 7 shows the images processed with these two algorithms. In the proposed depth generation algorithm, segmentation is responsible for providing region information. Although the proposed 4-bit segmentation only produces rough segmentation, it has provided enough region information for following processing and significantly improves the feasibility of real-time segmentation.

2.5 Merging Depth Map

Segmentation and edge detection are used to obtain several intermediate depth maps. Below, we integrate them into a merged depth map, as shown in Fig. 3. We use the GDM as the fundamental depth map. However, since it is a preliminary depth map, it cannot represent the shapes of the objects in the original image. To identify the objects in the original image, we adjust the depth values in the same color-group region by referencing the results of color-based grouping from the phase of 4-bit removing segmentation. The depth values of pixels in the same color-group region are assigned to the deepest value among the depth values in the region in the GDM.

2.6 Simplified Joint Bilateral Filtering

Simplified JBF (SJBF) is used to refine the merged depth map by increasing the edge information of objects in the original image. In the original JBF algorithm [12], the target image is produced through fine-tuning the source image by a reference image. The proposed algorithm employs a slightly modified formula, as shown in (5), from the one used in the original JBF algorithm. In (5), D, D', and I indicate the source, target, and reference images, respectively; p represents the position of the current pixel and q represents any position in the region S, a predefined region around pixel p. For example, I_q represents the pixel data at position q in the reference image I. In addition, $W_{p,q}$ is a weighting function that includes a spatial distance function g_d for computing the distance between p and q and an intensity range function g_r for computing the difference between D_p and I_q.

The intensity range function g_r functions as an edge-stopping function in the original JBF algorithm; however, it is not necessary to detect edges in the SJBF algorithm because the edge information is included in the merged depth map. Therefore, we can reduce the

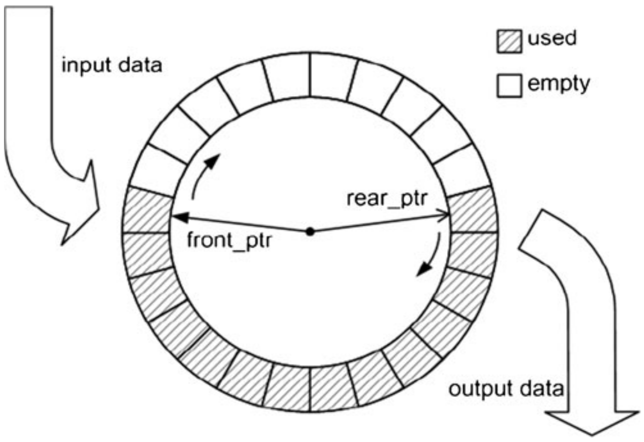


Figure 11 Proposed synchronized FIFO.

Figure 12 Pseudo code for synchronization at the producer end of the synchronized FIFO.

```
// DATA represents the data will be pushed into the FIFO.
Producer (DATA)
{
   // check whether FIFO is full.
   if (fifo.check_full() == TRUE) {
       fifo.wait_until_not_full();  // wait if FIFO is full.
   }

   // get the address of the front end of the FIFO.
   data_buffer = fifo.get_front_end();

   // write data into data_buffer.
   fifo.write(data_buffer, DATA);

   // increase front_ptr, FIFO length, and
   // issue a signal to consumer end.
   fifo.produce_update();
}
```

computational complexity by about 26 % by removing edge-stopping operations from the intensity range function g_r. Figure 8(a) and (b) shows the depth maps produced by SJBF for the images in Fig. 1(a) and (b), respectively.

$$D'_p = \sum_{q\in s} W_{p,q}\cdot D_q \Big/ \sum_{q\in s} W_{p,q},\ where \qquad (5)$$
$$W_{p,q} = g_d(p,q)\cdot g_r(D_p, I_q)$$

2.7 Block-Based Contrast Filtering

Block-based contrast filtering is used to differentiate between foreground and background objects in close-up images. We choose a 3×3 block as the basic processing unit for detection. Contract filtering is specified in (6), where I_{max} and I_{min} are respectively the largest and smallest values in the 3×3 block. This algorithm uses this contrast value to determine whether the 3×3 block belongs to a foreground or background object.

$$contrast = \frac{I_{\max} - I_{\min}}{I_{\max} + I_{\min}} \qquad (6)$$

After detection, it assigns the depth value 0 to background pixels, which indicates that they are at the furthest location in the image, and assigns the depth value 1 to foreground pixels. For close-up images, the depth map contains only two layers, namely foreground and background. Figure 9 shows the final depth map obtained using 3×3 contrast filtering for the close-up image shown in Fig. 1(c).

3 Proposed Parallelization Methodology

This section presents a parallel 3D video system based on the proposed 3D depth map generation algorithm. For generality, this system is designed to perform real-time processing on a

Figure 13 Pseudo code for synchronization at the consumer end of the synchronized FIFO.

```
// DATA represents the data will be popped off the FIFO.
DATA Consumer()
{
   // check whether FIFO is empty.
   if (fifo.check_empty() == TRUE) {
       fifo.wait_until_not_empty(); // wait if FIFO is empty.
   }

   // get the address of the rear end of the FIFO.
   data_buffer = fifo.get_rear_end();

   // read data into data_buffer.
   DATA = fifo.read(data_buffer);

   // increase rear_ptr, decrease FIFO length, and
   // issue a signal to producer end.
   fifo.consume_update();
   return DATA;
}
```

multicore platform by employing thread-level parallelization techniques, and the target multicore platform does not rely on any hardware-specific accelerations. This strategy makes the proposed system can be smoothly migrated to a multicore platform which supports thread-like APIs. Figure 10 shows the proposed parallel 3D video system. The system consists of a video dispatcher, a pseudo display, and several pipelined 3D depth map generators. At the front end of the proposed system, a video dispatcher receives videos decoded by a video decoder. It splits an input video into several individual frames and dispatches each frame to a pipelined 3D depth map generator. The pseudo display then collects all the depth maps in order. The following subsections describe thread-level superscalar-pipeline parallelization in detail.

3.1 Thread Synchronization

To ensure that the proposed parallel 3D video system operates correctly, it is necessary to establish appropriate synchronization mechanisms between the threads. Two synchronization approaches are developed and adopted in the system. One is a synchronized first-in, first-out (FIFO) buffer. We use the synchronized FIFO to connect any two threads once if one of them has to deliver data to the other. As shown in Fig. 11, the proposed synchronized FIFO is essentially a circular FIFO carried out based on a producer–consumer mechanism. The front and rear ends of the synchronized FIFO are connected to threads that function as the producer and the consumer, respectively. The producer can write data to the FIFO except when the FIFO is full. Similarly, the consumer can read data from the FIFO except when the FIFO is empty. Using this synchronization mechanism, synchronization can be easily realized in the proposed parallel 3D video system.

Below, we describe how the proposed synchronized FIFO handles synchronization between two threads. Figure 12 is the pseudo code for synchronization at the producer end of the synchronized FIFO. The thread at the producer end checks whether the FIFO is full; if it is, the thread waits until the FIFO is not full. After obtaining access permission, the thread starts to write its data to the FIFO. The thread then calls a confirmation function that updates the information recorded in the FIFO and sends a notification signal to the thread at the consumer end of the FIFO. We consider synchronization at the consumer end. Figure 13 shows the pseudo code for synchronization at the consumer end of the synchronized FIFO. The thread at the consumer end of the synchronized FIFO checks whether FIFO is empty; if it is, the thread waits until the FIFO is not empty. After obtaining access permission, the thread can start to read data from the FIFO. The thread also updates the information of the FIFO by calling a confirmation function and it sends a notification signal to the thread at the producer end of the FIFO.

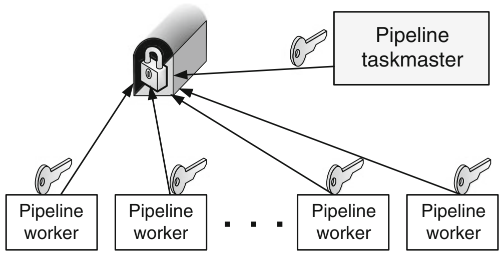


Figure 14 Proposed synchronized mailbox.

An alternative synchronization approach is a synchronized mailbox, which is used to synchronize threads. Figure 14 depicts the proposed synchronized mailbox. The pipeline workers execute the designated tasks and the pipeline taskmaster monitors the processing status of all the pipeline workers. Each pipeline worker puts a message in the mailbox and sends a notification signal to the taskmaster. The taskmaster then accesses the message in the mailbox. Only one taskmaster or worker can access the mailbox at a time.

3.2 Task Partition

This section describes the concept of thread-level pipeline parallelization. We assume that the entire process consists of a sequence of similar computations and that each computation can be divided into several small tasks. In a similar manner as the pipeline mechanism, the tasks of different computations can be processed simultaneously. The working performance of a pipeline depends on the processing speed of the longest pipeline

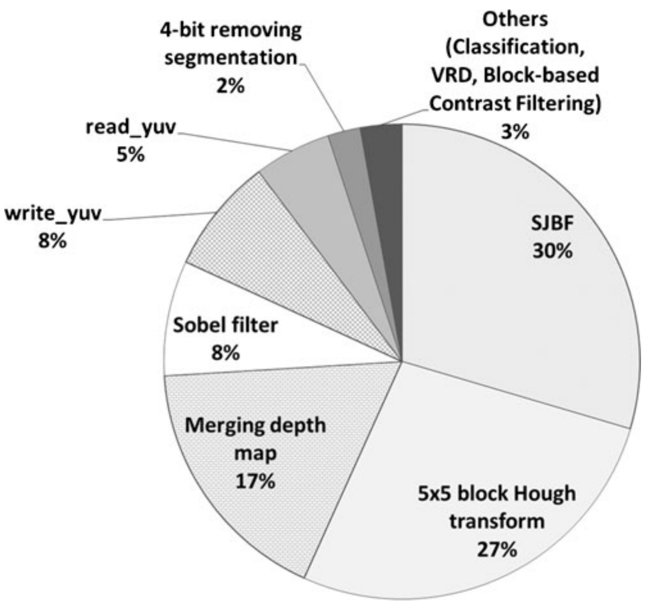


Figure 15 Workload distribution for the proposed 3D depth algorithm.

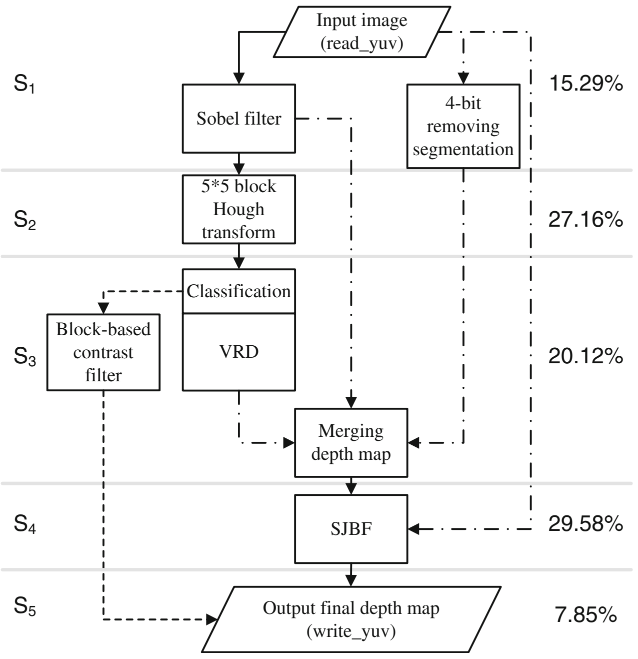

Input image (read_yuv)
S₁
Sobel filter
4-bit removing segmentation
15.29%
S₂
5*5 block Hough transform
27.16%
S₃
Classification
VRD
Block-based contrast filter
20.12%
Merging depth map
S₄
SJBF
29.58%
S₅
Output final depth map (write_yuv)
7.85%

Figure 16 Task partition for the proposed 3D depth algorithm.

stage. Therefore, the workload for each task has to be balanced to maximize the performance of the whole pipeline.

For such a purpose, we analyzed the workload distribution for the proposed 3D depth map algorithm (see Fig. 15). Using this data, we can appropriately partition the processing flow of the proposed 3D depth algorithm to balance the load. The working flow of the proposed 3D depth generator is partitioned into five stages, S_1–S_5, as shown in Fig. 16. Although the workload has been distributed as evenly as possible, the workloads in some stages still differ considerably from those in other stages. As depicted in Fig. 16, the workload of stage S_5 is much lower than those in the four other stages. Stage S_5 cannot be merged with stage S_4 because stage S_4 has one of the largest workloads.

3.3 Data Packetization

To realize thread-level pipeline parallelization, we use the synchronized FIFOs mentioned in Section 3.1 to connect the stages to construct a pipeline. As shown in Fig. 17, each FIFO between two stages functions like a conveyor. At each pipeline stage, extra buffers are required to temporarily reference data for subsequent stages. In addition, all related buffers are bundled into a packet, which is transferred between stages. By this technique, the pipeline worker at each stage can easily access relevant data from the current packet and perform the designated task on it.

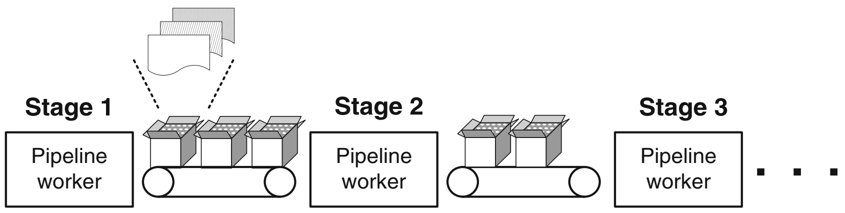
3.4 Pipeline Control

Pipeline control plays an important role in thread-level pipeline parallelization. Figure 18 depicts the pipeline control system used in the proposed pipelined 3D depth map generator. Each pipeline worker is responsible for executing a designated task. The pipeline taskmaster monitors the processing status of all the pipeline workers. The control system includes two kinds of control signals. The first control signal is an inter-stage control signal. Each pipeline worker is connected with its neighbors through the synchronized FIFOs. Each pipeline worker can deliver the data and control signal from its preceding neighbor or to its following neighbor. The second control signal is the over-stage control signal. It is used between the pipeline taskmaster and all the pipeline workers. This control signal is transferred through the synchronized mailbox described in Section 3.1.

Figure 19 shows the work flow of the pipeline taskmaster. The pipeline taskmaster initializes the synchronized FIFOs and creates threads, which are designated as pipeline workers. All the pipeline workers then operate while the pipeline taskmaster enters waiting status. When a pipeline worker completes its task, it sends a notification signal to the pipeline taskmaster. When the pipeline taskmaster receives this signal, it checks whether $N_{finished} < N_{total}$, where $N_{finished}$ is the number of pipeline workers that have finished their tasks and N_{total} is the total number of pipeline workers. The pipeline taskmaster waits until all the pipeline workers have finished. Finally, it destroys the synchronized FIFOs and terminates.

Figure 20 shows the working flow of the pipeline workers. The pipeline workers can be classified into three types based on which pipeline stage they are located at: the head, the tail, and the body stages. Each pipeline worker checks its input FIFO. It then checks the termination flag delivered by the preceding pipeline worker. If the termination flag is not

Figure 17 Concept of data packetization in the proposed 3D depth map generator.



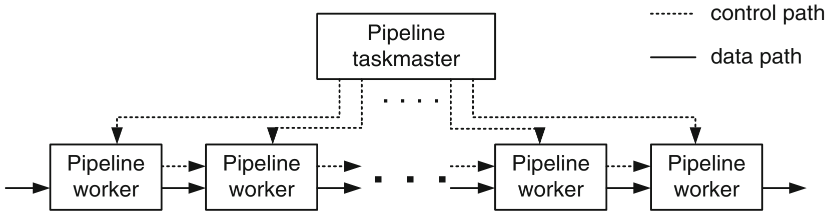


Figure 18 Pipeline control system in the proposed 3D depth map generator.

set, the pipeline worker will read data from its input FIFO and execute the designated task with the data. Finally, it checks its output FIFO and writes the processed data to it. Some additional steps have to be included in the work flow since each image packet is created by the head-stage pipeline worker and destroyed by the tail-stage pipeline worker. These steps are repeated until the termination flag is set. As soon as the termination flag is set, the pipeline worker exits the loop and writes the termination flag to its output FIFO to inform the next pipeline worker to terminate. This step is skipped by the tail-stage pipeline worker. Before terminating, each pipeline worker notifies the pipeline taskmaster of its termination.

3.5 Superscalar-Pipeline Parallelization

Although the performance of the algorithm was improved by employing thread-level pipeline parallelization, the throughput of the pipelined 3D depth map generator is much too low for real-time processing. To increase the throughput, we employed the design concept of superscalar processors [13]. Superscalar processors can improve the instruction-level parallelism because they use multiple pipelines. Consequently, we propose a thread-level superscalar-pipeline parallelization that is an extension of the pipeline-based parallelization described in the previous section. Figure 21 compares the execution behaviors of sequential, pipelined, and superscalar-pipeline computations. This figure assumes that the pipelined computation employs a five-stage (S_1–S_5) pipeline and the superscalar-pipeline computation uses three five-stage pipelines. T_1, T_2, and T_3 respectively represent the completion timestamps of processing three frames for sequential, pipelined, superscalar-pipeline computations. The superscalar-pipeline computation clearly employs higher parallelism and takes less time to achieve the same throughput as the sequential and pipelined computations.

3.6 Hardware-specific Acceleration

The proposed parallelization methodology is designed to exploit the multi-level parallelisms according to the features of the proposed 3D depth generation algorithm. In the coarse-grained view, the thread-pipelining technique exposes task-level parallelism, and the superscalar technique exploits data-level parallelism. In the fine-grained view, hardware/software approaches can be appropriately introduced to enhance the performance of the phases in the proposed 3D depth generation algorithm. If the performance of the phases is improved, developers are suggested to re-partition the processing flow into pipeline stages to obtain best performance benefits.

Several platforms provide hardware-specific mechanisms (e.g., DSP, GPU, and SIMD instruction) to speed up multimedia-related computations. These mechanisms can be used to enhance the performance of phases in the proposed algorithm, and then are seamlessly cooperated with

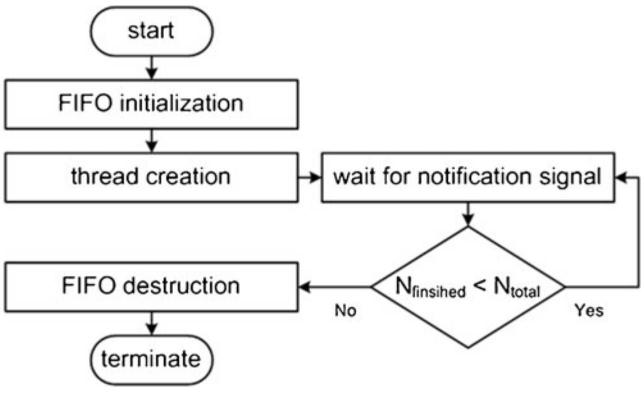


Figure 19 Work flow of the pipeline taskmaster.

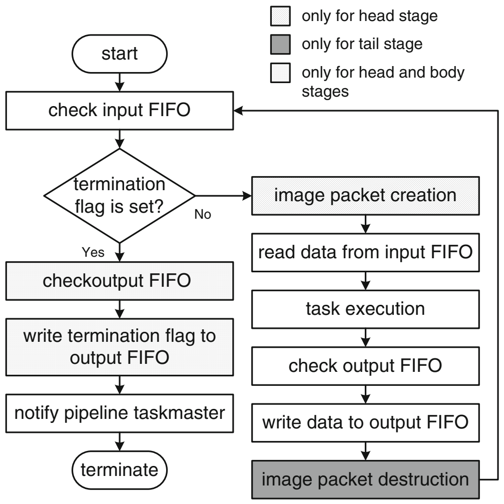


Figure 20 Work flow of the pipeline worker.

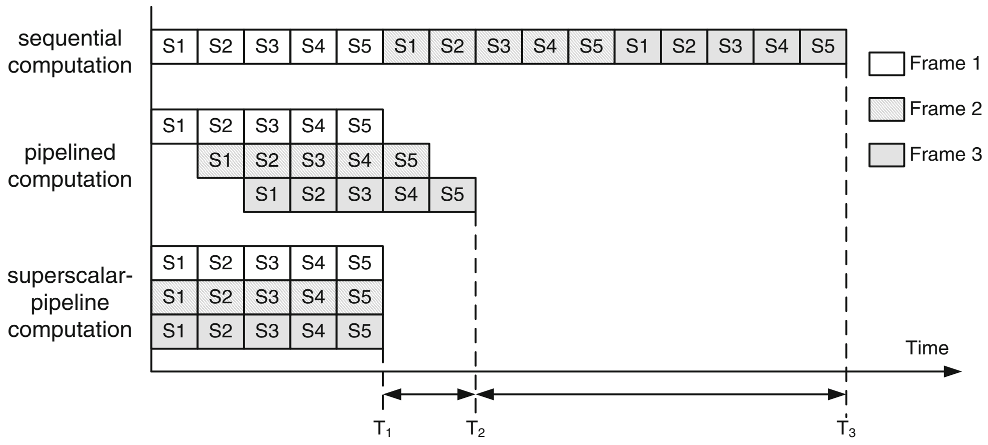


Figure 21 Comparison of the execution characteristics of sequential, pipelined, and superscalar-pipeline computations.

the proposed methodology. We describe these issues as follows:

- DSP and GPU: For a heterogeneous multicore system with DSPs or GPUs, the proposed methodology can be realized by multiple CPU cores, and DSPs or GPUs could be used to improve the performance of the phases in the proposed algorithm. Developers need to use vendor-provided low-level APIs to control DSPs or GPUs, and to appropriately handle I/O operations among CPU cores and DSPs or GPUs to obtain performance benefits.
- SIMD instruction: Some processors support SIMD instructions to exploit fine-grained data-level parallelism. These SIMD instructions are suitable for speeding up the phases in the proposed algorithm. In order to leverage SIMD instructions, developers need to pack and unpack data to maximize computation throughput. However, the overheads from these data arrangements usually nullify performance benefits from SIMD computations. The situation might be worse due to poor data locality after data arrangements. The complex data dependence relationships and conditional branches also decrease the opportunities of applying SIMD instructions. Table 1 shows applicable opportunities of SIMD instructions for the phases of the proposed 3D depth generation algorithm.

3.7 Performance Modeling

Consider the parallel 3D video system constructed by the thread-level superscalar-pipeline approach. The critical stage is defined as the stage that has the longest execution time of all the stages in the pipeline. Theoretically, the critical stage dominates the performance of a pipeline. $T_{pipelined}$ represents the total execution time of a single pipeline. $T_{pipelined}$ can be modeled by (7), where N_{stage} is the number of stages used in the pipeline, $N_{processed}$ is the total number of frames to be processed, $T_{latency}$ is the time from when the pipeline is empty to when it is full, and $T_{critical}$ is the execution time of the critical stage.

$$\begin{aligned} T_{pipelined} &= T_{latency} + (N_{processed} - 1) \cdot T_{critical} \\ &= N_{stage} \cdot T_{critical} + (N_{processed} - 1) \cdot T_{critical} \\ &= (N_{stage} + N_{processed} - 1) \cdot T_{critical} \end{aligned} \tag{7}$$

Equation (8) is used to compute $T_{critical}$, where T_{Si} is the execution time of stage S_i, $T_{sequential}$ is the execution time in sequential mode, and R_{Si} is the ratio between the execution time of stage S_i and the total execution time in sequential mode. For the case shown in Fig. 16, the

Table 1 Analysis of applicable opportunity for the proposed algorithm.

Phase	Feature	Applicability
Sobel filter	Large similar MAC operations	Applicable
5×5 block Hough transform	Large similar MAC operations but followed by some counting operations	Partially applicable
VRD	Too many branch operations inside loops	Hardly applicable
4-bit removing segmentation	Too few operations	Hardly applicable
Merging depth map	Too many branch operations inside loops	Hardly applicable
SJBF	Large similar MAC operations	Applicable
Write yuv	RGB to YUV	Applicable
Read yuv	YUV to RGB	Applicable

Table 2 Average execution times (seconds) for the proposed algorithm.

Original algorithm				Proposed algorithm			
Resolution	QVGA	XGA	HD1080	Resolution	QVGA	XGA	HD1080
Sobel filter	0.024	0.223	0.627	3×3 Sobel filter	0.005	0.049	0.132
Hough transform	0.09	0.277	0.422	5×5 block Hough transform	0.025	0.073	0.107
VRD	0.001	0.001	0.001	VRD	0.001	0.001	0.001
Mean shift segmentation	0.84	9.39	24.6	4-bit removing segmentation	0.001	0.01	0.027
Merging depth map	0.002	0.026	0.065	Merging depth map	0.002	0.024	0.064
JBF	0.007	0.056	0.158	SJBF	0.005	0.048	0.134
Total	0.964	9.973	25.873	Total	0.039	0.205	0.465

critical stage in the proposed five-stage pipeline is S_4 and R_{S4}=0.2958.

$$T_{critical} = \max\{T_{s_i}\} = \max\left\{\frac{R_{s_i}\cdot T_{sequential}}{N_{processed}}\right\} \quad where\, i \in \{1,2,\ldots,N_{stage}\} \tag{8}$$

Equation (9) is used to compute the execution time $T_{superscalar}$ for the application of thread-level superscalar-pipeline parallelization. If $N_{pipeline}$ represents the number of used pipelines, $T_{superscalar}$ can be determined from $T_{pipelined}/N_{pipeline}$. From (7) and (9), we obtain a formula for predicting the performance of the whole system, $P_{superscalar}$, as given in (10).

$$\begin{aligned} T_{superscalar} &= \frac{T_{pipelined}}{N_{pipeline}} \\ &= \frac{\left(N_{stage}+N_{processed}-1\right)\cdot T_{critical}}{N_{pipeline}} \end{aligned} \tag{9}$$

$$\begin{aligned} P_{superscalar} &= \frac{N_{processed}}{T_{superscalar}} = \frac{N_{preocessed}\cdot N_{pipeline}}{T_{pipelined}} \\ &= \frac{N_{processed}\cdot N_{pipeline}}{(N_{stage}+N_{processed}-1)\cdot T_{critical}} \end{aligned} \tag{10}$$

In practice, the overheads from resource management and synchronization will affect the performance of a parallel application. To accurately predict the performance, we introduce $T_{overhead}$ that represents the overheads. $T_{overhead}$ can be approximately obtained from the difference of the execution times of sequential and sequentially pipelined computations, as shown in (11). A sequentially pipelined computation performs computations using the thread-level pipeline approach, but only one frame is input at a time. In other words, the next frame will not be input until the current frame exits the pipeline. In such a case, each frame is processed sequentially except for migration from one stage to another stage. $T_{sequentially\_pipelined}$ is the execution time for sequentially pipelined computation, and $P_{sequential}$ and $P_{sequentially\_pipelined}$ are the performances in sequential and sequentially pipelined computations, respectively.

$$\begin{aligned} T_{overhead} &= \frac{T_{sequentially\_pipelined}-T_{sequential}}{N_{processed}} \\ &= \frac{1}{P_{sequentially\_pipelined}} - \frac{1}{P_{sequential}} \\ &= \frac{P_{sequential}-P_{sequentially\_pipelined}}{P_{sequential}P_{sequentially\_pipelined}} \end{aligned} \tag{11}$$

Equation (12) gives an expression for $T'_{critical}$, which is the execution time at the critical stage after considering the effect of the overheads. By combining (10) and (12), we finally obtain the estimation model defined in (13),

Table 3 Speedup of the proposed algorithm relative to the original algorithm.

	Proposed algorithm		
Resolution	QVGA	XGA	HD1080
3×3 Sobel filter	4.8	4.6	4.8
5×5 block Hough transform	3.6	3.8	3.9
VRD	1.0	1.0	1.0
4-bit removing segmentation	840.0	939.0	911.1
Merging depth map	1.0	1.1	1.0
SJBF	1.4	1.2	1.2
Total	24.7	48.6	55.6

Table 4 Complexity reduction of the proposed algorithm as compared to the original one.

	Proposed algorithm		
Resolution	QVGA	XGA	HD1080
3×3 Sobel filter	79.2 %	78.0 %	78.9 %
5×5 block Hough transform	72.2 %	73.6 %	74.6 %
VRD	0.0 %	0.0 %	0.0 %
4-bit removing segmentation	99.9 %	99.9 %	99.9 %
Merging depth map	0.0 %	7.7 %	1.5 %
SJBF	28.6 %	14.3 %	15.2 %
Total	95.6 %	97.9 %	98.2 %

Table 5 Elementary operations required for the proposed depth generation algorithm.

Original algorithm						Proposed algorithm					
Phase	N_{MUL}	N_{ADS}	N_{SQRT}	N_{EXP}	N_{BRANCH}	Phase	N_{MUL}	N_{ADS}	N_{SQRT}	N_{EXP}	N_{BRANCH}
Sobel filter	22P	60P	1P	0	0	3×3 Sobel filter	20P	60P	0	0	2P
Hough transform	356P	534P	0	0	0	5×5 block Hough transform	6P	11P	0	0	40P
Mean shift segmentation	864P	432P	0	432P	9P	4-bit removing segmentation	0	11P	0	0	0
JBF	51P	238P	0	0	34P	SJBF	34P	170P	0	0	0

P = number of total pixels inside each frame = frame_width×frame_height

where $P'_{superscalar}$ is the actual performance of the whole system parallelized by the thread-level superscalar-pipeline approach. Using the performance model, users can predict the performance at an early stage of system

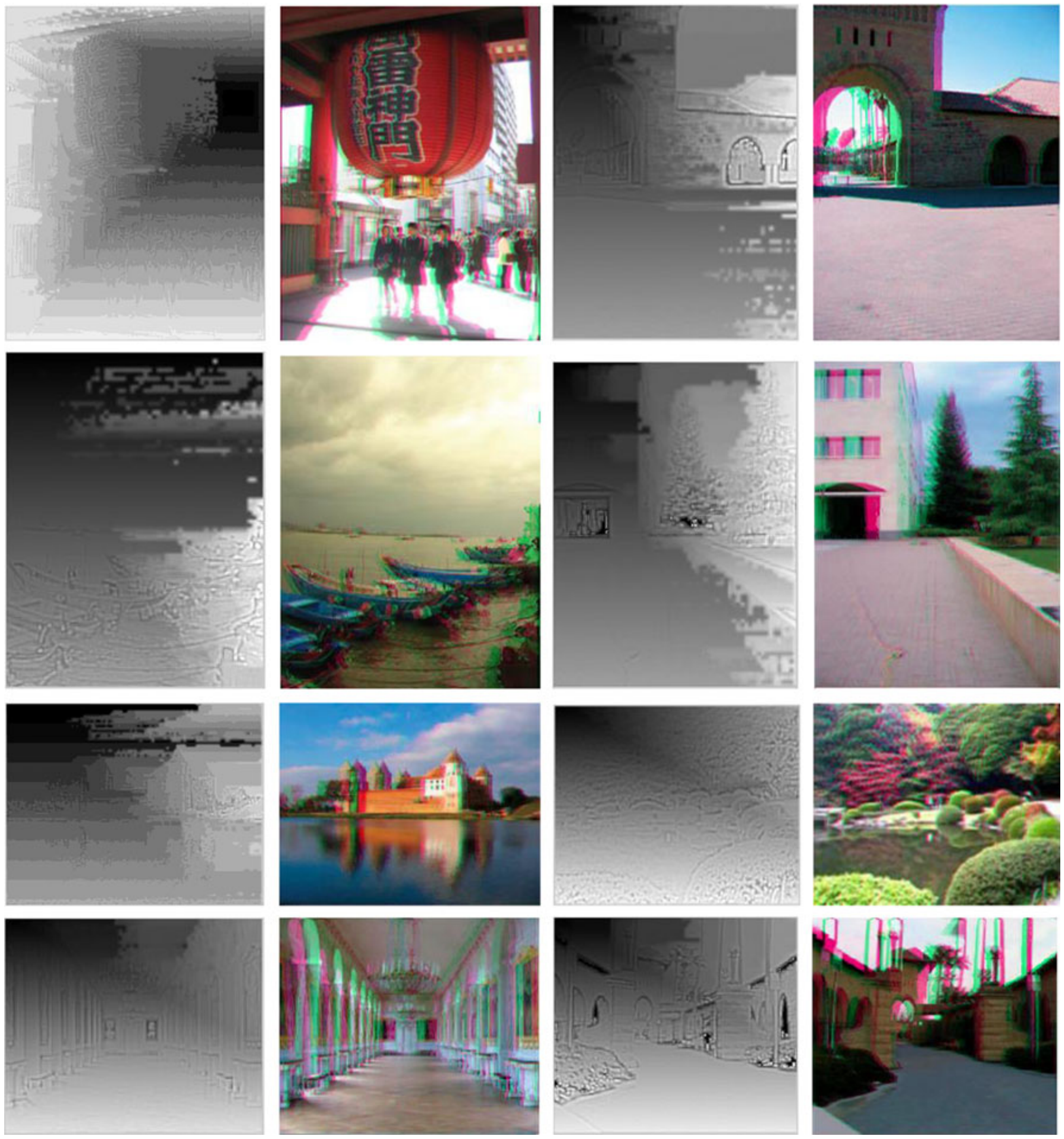


Figure 22 Examples of depth maps and corresponding stereoscopic images.

Table 6 Average quality scores of the proposed algorithm.

	3D visual perception			Visual picture quality		
	[9]	[10]	Proposed	[9]	[10]	Proposed
General	8.2	8.4	8.45	8.4	8.55	8.5
Scenery	7.9	8.1	8.1	6.5	7.0	6.95
Close-up	7.5	7.7	7.6	7.9	8.2	8.25

development and then tune the design parameters to satisfy the requirements.

$$\begin{aligned}T'_{crirical} &= \max\{T_{s_i}\} + T_{overhead}\\ &= \max\left\{\frac{R_{s_i}\cdot T_{sequential}}{N_{processed}}\right\} + \left\{\frac{P_{sequential}-P_{sequentially\_pipelined}}{P_{sequential}P_{sequentially\_pipelined}}\right\}\\ &= \frac{\max\{R_{s_i}\}}{P_{sequential}} + \left\{\frac{P_{sequential}-P_{sequentially\_pipelined}}{P_{sequential}P_{sequentially\_pipelined}}\right\}\end{aligned} \tag{12}$$

$$\begin{aligned}P'_{superscalar} &= \frac{N_{processed}\cdot N_{pipeline}}{(N_{stage}+N_{processed}-1)\cdot T'_{critical}}\\ where\ T'_{critical} &= \frac{\max\{R_{s_i}\}}{P_{sequential}} + \left\{\frac{P_{sequential}-P_{sequentially\_pipelined}}{P_{sequential}P_{sequentially\_pipelined}}\right\}\end{aligned} \tag{13}$$

4 Performance Evaluation

This section evaluates the complexity reduction and quality measurement of the proposed 3D depth map generation algorithm. It also assesses the improved performance of the proposed parallel 3D video system.

4.1 Complexity Reduction and Quality Measurement

The proposed 3D depth map generation algorithm is implemented as a sequential program for the evaluation. Experiments were conducted on a computer with an Intel Core 2 Quad processor with 3 GB of memory. Fifty four images from Refs. 12 and 13, which had three different resolutions (1920×1080 (HD1080), 1024×768 (XGA), and 320×240 (QVGA)), were used to evaluate the performance. Table 2 shows the average execution times for images at each resolution. Table 3 lists the speedups relative to the original algorithm. Using the proposed algorithm, the phase of 4-bit removing segmentation yields speedups of 840 for QVGA, 939 for XGA, and 911.1 for HD1080. The phases of VRD and merging depth map give almost no improvement in performance, while the phase of SJBF results in slight improvement. Overall, the algorithm yields speedups of 24.7 for QVGA, 48.6 for XGA, and 55.6 for HD1080, respectively. Table 4 compares the complexity reduction of the algorithms, which were evaluated from the ratio of the reduced execution time relative to the original execution time. The experimental results show that the proposed algorithm achieves reductions of 95.6 %, 97.9 %, and 98.2 % in the computational complexity for QVGA, XGA, and HD1080, respectively.

In order to make the measurement method more useful for all platforms, we provide a complexity model defined in (14). Equation (14) calculates complexity based on elementary operations required to carry out the algorithm like multiplication, addition, and so on. Here N_{OP} means the number of operation *OP* while C_{OP} means the cost when executing one operation *OP*. All C_{OP} are evaluated based on the same evaluation unit. Operation *OP* includes multiplication, addition, shift, square root, and branch. We also use operation “ADS” standing for ADD and SHIFT operations because the costs to execute these two operations are almost the same. Table 5 shows elementary operations required for the original depth generation algorithm and the optimized one. The symbol “P” represents the number of total pixels inside each frame. For each elementary operation *OP* on a target platform, C_{OP} can be easily obtained by profiling. By feeding the C_{OP} and the corresponding operation frequency (shown in Table 5) into Eq. (14), developers can get correct complexity information on this target platform.

$$\begin{aligned}Complexity &= \sum (N_{OP}\times C_{OP})\\ &= N_{MUL}\times C_{MUL} + N_{ADS}\times C_{ADS} + N_{SQRT}\times C_{SQRT} + N_{EXP}\\ &\quad \times C_{EXP} + N_{BRANCH}\times C_{BRANCH}\end{aligned} \tag{14}$$

where $OP \in \{MUL, ADS, SQRT, EXP, BRANCH\}$

Next, we consider quality measurement. We apply a similar subjective quality evaluation approach as that used in Refs. [1] and [14] to the proposed 3D depth map generation algorithm. We use a depth-image-based rendering

Table 7 Technology summary of current depth generation researches.

	[1]	[2]	[3]	[14]	[9]	[10]	Proposed
Technology	Motion-based	Edge-based	Object-based	Motion-based	Learning-based	Edge-based	Object-based
Complexity	Medium	High	Medium	High	High	High	Low
Applicability	Video	Image	Image	Video	Image	Image / Video	Image / Video
Input requirement	Video sequence	Single image	Single image	Video sequence	Single image	Single image	Single image
Depth quality	Good	Fair	Fair	Fair	Fair	Good	Good

Table 8 Configuration of experimental environment.

	Item	Value
Hardware	CPU	AMD Opteron quad-core processor running at 2.3 GHz×8 (32 physical cores)
	Cache	L1: 64 KB, L2: 512 KB, L3: 2048 KB
	Memory	64 GB
Software	Operating system	Linux (kernel version: 2.6.9)
	Native C compiler	GCC 3.4.6 with –O2 option
	Thread library	POSIX thread library

algorithm [15] to produce 54 stereoscopic images based on depth maps. In the algorithm, each depth map and its original image are used to synthesize corresponding left- and right-view images. Figure 22 shows samples of depth maps and their corresponding stereoscopic images. To generate a stereo effect in vision, these stereoscopic images are displayed on a 42-inch 3D display connected to the PC. We asked 30 people to wear red/cyan glasses and to rank these stereo images on a scale of 1 to 10 in terms of 3D visual perception and visual picture quality, where 10 represents the strongest stereoscopic sense. Table 6 shows the average quality scores for the algorithms of Saxena et al. [9], Cheng et al. [10], and the present study. The proposed 3D depth map generation algorithm can obtain almost the same stereoscopic quality as the algorithms of Saxena et al. and Cheng et al. and most of the tested images processed by the algorithm obtain high scores. These results demonstrate that the depth information generated by the algorithm still has a good quality in vision even though we have simplified a lot of the computation in the proposed algorithm relative to that in the original algorithm. Table 7 shows the technology summary among the proposed 3D depth generation algorithm and the existing designs.

4.2 Performance on Thread-Level Superscalar-Pipeline Parallelization

We constructed a parallel 3D video system based on the proposed thread-level superscalar-pipeline approach. The implementation uses the POSIX thread library to manage multiple threads for portability. Table 8 lists the configuration of the experimental environment. The system was evaluated using a multicore platform that consists of eight AMD Opteron quad-core processors running at 2.3 GHz. Figure 23 compares the performances for different numbers of pipelines. The gray line represents the execution performance at runtime and the back line represents the execution performance predicted by the proposed performance model. The label "sequential execution" on the *x*-axis represents the performance of the sequential execution, while the other labels represent the number of pipelines. The tested video has HD720 resolution and contains 300 frames. The experimental results reveal that the system gives good scalability and performance. For the configuration with six pipelines, the proposed parallel 3D video system can achieve 63.66 fps. We also evaluated the accuracy of the performance model described in Section 3.7. The errors between the runtime performance and predicted performance are

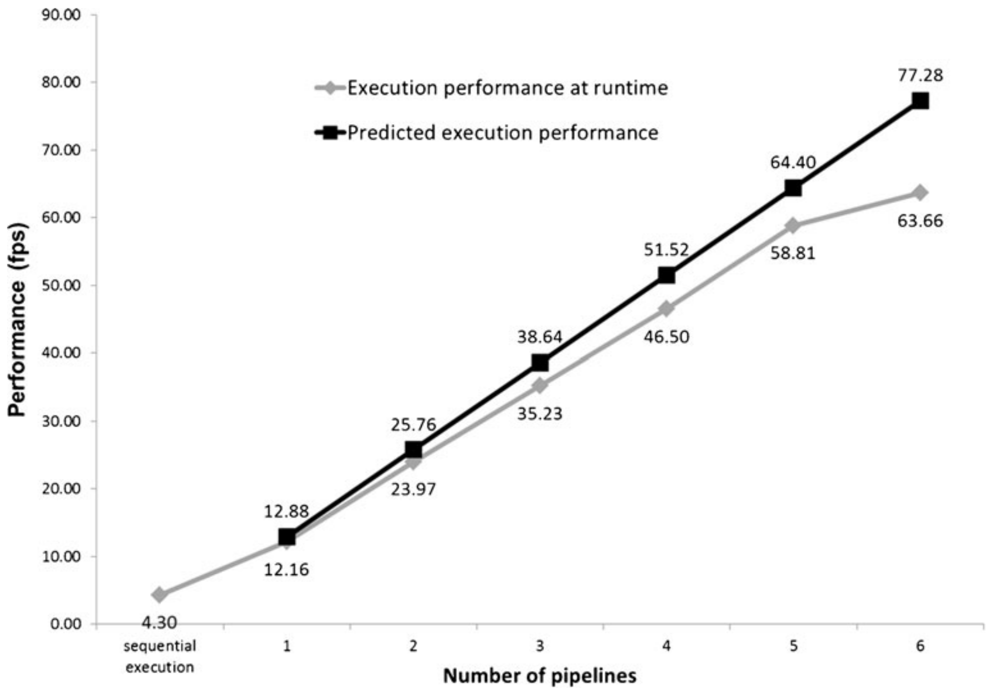


Figure 23 Performance evaluation of the proposed parallel 3D video system.

ranged from 5.9 % to 10.8 % for configurations with 1–5 pipelines. For the configuration with six pipelines, the runtime performance is slightly improved relative to the configurations with 1–5 pipelines and the error in the predicted performance increases to 21.4 %. The reason is because when more than five pipelines are created, the overheads of thread management become complicated and cannot be accurately predicted using the proposed performance model. In addition, the memory bandwidth at target platform limits the system performance when the memory requirement of the system cannot be satisfied.

5 Conclusion

We proposed a low-complexity 3D depth-map-generation algorithm for stereoscopic video applications. This algorithm is expected to generate high-quality depth maps based on 2D images. The processing steps in the algorithm have been optimized to reduce its computational complexity without sacrificing quality. We applied the proposed algorithm to construct a parallel 3D video system that was realized on a multicore computer based on a thread-level superscalar-pipeline approach. The experimental results demonstrate that for the tested images with HD1080 resolution, the proposed 3D depth map generation algorithm reduces the computational complexity by 98.2 % relative to the original algorithm while preserving good quality. Thus, the proposed parallel 3D video system can perform real-time processing of videos with HD720 resolution.

Acknowledgments The work was sponsored by the National Science Council of Taiwan under grants NSC-97-2221-E-194-028-MY2, NSC-99-2221-E-194-048, and NSC-99-2221-E-194-045.

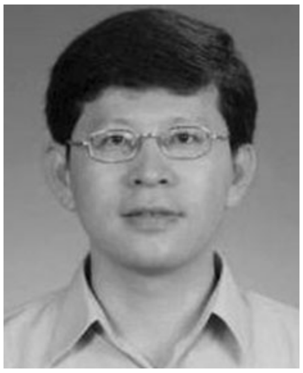
References

1. Cheng, C. C., Li, C. T., Huang, P. S., Lin, T. K., Tsai, Y. M., & Chen, L. G. (2009). A block-based 2D-to-3D conversion system with bilateral filter. *Proceedings of IEEE International Conference on Consumer Electronics*, 1–2.
2. Battiato, S., Capra, A., Curti, S., & La Cascia, M. (2004). 3D stereoscopic image pairs by depth-map generation. *Proceedings of International Symposium on 3D Data Processing, Visualization and Transmission*, 124–131.
3. Huang, Y. S., Cheng, F. H., & Liang, Y. H. (2008). Creating depth map from 2D scene classification. *Proceedings of International Conference on Innovative Computing Information and Control*, 69–72.
4. Tsai, Y. M., Chang, Y. L., & Chen, L. G. (2006). Block-based vanishing line and vanishing point detection for 3D scene reconstruction. *Proceedings of International Symposium on Intelligent Signal Processing and Communications*, 586–589.
5. Comaniciu, D., & Meer, P. (1999). Mean shift analysis and applications. Proceedings of IEEE International Conference on Computer Vision, 2, 1197–1203.
6. Gangwal, O. P., & Berretty, R. P. (2009). Depth map post-processing for 3D-TV. Proceedings of IEEE International Conference on Consumer Electronics, 1–2.
7. Mattson, T. G., Sanders, B. A., & Massingill, B. L. (2004). Patterns for Parallel Programming, 1st Edition. Addison-Wesley Professional.
8. Padua, D. A., & Wolfe, M. J. (1986). Advanced compiler optimizations for supercomputers. *Communication of ACM, 29*(12), 1184–1201.
9. Saxena, A., Sun, M., & Ng, A. Y. (2009). Make3D: learning 3D scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 31*(5), 824–840.
10. Cheng, C. C., Li, C. T., & Chen, L. G. (2010). A novel 2D-to-3D conversion system using edge information. *IEEE Transactions on Consumer Electronics, 56*(3), 1739–1745.
11. Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM, 15*(1), 11–15.
12. Petschnigg, G., Agrawala, M., Hoppe, H., Szeliski, R., Cohen, M., & Toyama, K. (2004). Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics, 23*(3), 664–672.
13. Hennessy, J. L., & Patterson, D. A. (2006). Computer Architecture: A Quantitative Approach, 4th Edition. Morgan Kaufmann Publishers.
14. Pourzad, M. T., Nasiopoulos, P., & Ward, R. K. (2010). Conversion of H.264-encoded 2D video to 3D format. Proceedings of IEEE International Conference on Consumer Electronics, 63–64.
15. Zhang, L., Tam, W. J., & Wang, D. (2004). Stereoscopic image generation based on depth images. Proceedings of IEEE International Conference on Image Processing, 5, 2993–2996.

Guo-An Jian was born in Taichung, Taiwan, in 1980. He received the B.S. and M.S. degrees in computer science from National Chung-Cheng University, Chia-Yi, Taiwan, in 2003 and 2005, respectively. He is currently a Ph.D. candidate in the Department of Computer Science and Information Engineering, National Chung-Cheng University, Chia-Yi, Taiwan. His research interests include software optimization and parallel computing for video technology.

Cheng-An Chien was born in Taipei, Taiwan, in 1983. He received the B.S. degree and M.S. degree in Department of Computer Science and Information Engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 2006 and 2008, respectively. He is currently working on his Ph.D. degree in Department of Computer Science and Information Engineering at National Chung Cheng University. His research interests include video processing algorithm, 3D display algorithm, VLSI architecture design, and digital IP design.

Peng-Sheng Chen was born in Tainan, Taiwan, in 1973, and received a B.S. degree in Computer Science from National Tsing-Hua University, Taiwan, in 1995, an M.S. degree in Computer Science and Information Engineering from National Cheng-Kung University in 1997, and a Ph.D. in Computer Science from National Tsing-Hua University in 2005. He is currently an Assistant Professor in the Department of Computer Science and Information Engineering, National Chung-Cheng University, Chia-Yi, Taiwan. His research interests include parallel programming, optimizing compilers, and computer architecture.



Jiun-In Guo was born in Kaohsiung, Taiwan, in 1966. He received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1989 and 1993, respectively. He is currently a Processor in the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan. He was a Processor in the Department of Computer Science and Information Engineering, National Chung-Cheng University, Chia-Yi, Taiwan, from 2003 to 2011. During the period, he joined the System-on-Chip (SOC) Research Center, to get involved in several Grand Research Projects on low-power, high-performance processor design, and multimedia IP/ SOC design. He was the Chairman of the Department of Computer Science and Information Engineering, National Chung-Cheng University, from 2009 to 2011 and was the Director of the SOC Research Center, National Chung-Cheng University, from 2005 to 2008. Also, he was the Research Distinguished Professor of National Chung-Cheng University from 2008 to 2010. He was an Associate Professor of the Department of Computer Science and Information Engineering, National Chung-Cheng University, from 2001 to 2003, and an Associate Professor of the Department of Electronics Engineering, National Lien-Ho Institute of Technology, Miaoli, Taiwan, from 1994 to 2001. Also, he was the Chairman of the Department of Electronics Engineering, National Lien-Ho Institute of Technology, from 1996 to 1999. His research interests include images, multimedia, and digital signal processing, VLSI algorithm/architecture design, digital SIP design, and SOC design. He is the author of over 160 technical papers on the research areas of low-power and low cost algorithm and architecture design for DSP/Multimedia signal processing applications.

Dr. Guo was the recipient of the National Science Council (NSC) Research Award in 1996 and 2001. He was the recipient of the 2003 MXIC Young Professor Award for his contributions to the course of low-power Multimedia/DSP silicon IP design. He was also the recipient of the 2004 Chinese Institute of Electrical Engineering (CIEE) Outstanding Youth Electrical Engineer Award, the recipient of the 2008 Chinese Institute of Electrical Engineering (CIEE) Tai-Chung Section Outstanding Engineering Professor Award, and the recipient of the 2010 Chinese Institute of Electrical Engineering (CIEE) Outstanding Electrical Engineering Professor Award to recognize his excellent contributions to Research and Development and service of electrical engineering. He was also the recipient of the 2006 Outstanding Research Award of National Chung Cheng University. His research team has won over 36 IC related student design contest awards from 2003 to 2011.