This paper is author's private version of the paper published as follows:
Proc. of 7th Int'l Conf. on Signal-Image Technology and Internet-Based Systems (SITIS2011), pp.253–262

DOI: 10.1109/SITIS.2011.25

Tiling with Different Spatial Resolutions for Pseudo Real-Time Video Processing Library RaVioli

Katsuhiko KONDO*, Ami ONO*, Takafumi INABA*, Tomoaki TSUMURA* and Hiroshi MATSUO*

*Nagoya Institute of Technology
Gokiso, Showa, Nagoya, Japan
Email: camp@matlab.nitech.ac.jp

***Abstract*—The performance of general purpose computers is increasing rapidly, and now they are capable of running video processing applications. However, on general purpose operating systems, real-time video processing is still difficult because there is no guarantee that enough CPU resources can surely be provided. A pseudo real-time video processing library RaVioli has been proposed for solving this issue. RaVioli conceals two resolutions, frame rate and number of pixels, from programmers and provides a dynamic and transparent resolution adjustability. Using RaVioli, pseudo real-time video processing can be achieved easily, but output precision may be roughened for reducing processing load. To prevent this situation, this paper proposes a method for automatically dividing whole video frame into several subframes, or tiles, and changing the resolutions of each tile individually. We have implemented the method on RaVioli and have made some evaluations with a sample program. The result shows that the proposed method can keep the resolution of video frames higher than traditional RaVioli.**

I. Introduction

Real-time video processing applications such as surveillance systems, smoke detection systems or automatic vehicle collision avoidance systems are increasingly popular and now in demand. Therefore, the demand of the systems, which highly requires such real-time video processing, is rapidly increasing. Furthermore, the processors for general-purpose computers have been developed drastically. The performance of the processors has been improved, and the cost has been reduced. Therefore, it is also expected that the performance improvement and the cost reduction will promote real-time video processing on the general-purpose computers and operating systems. In spite of the advances of the general-purpose processors, it is difficult to realize the real-time video processing on general-purpose operating systems, because it should keep a certain output framerate. The main reasons for the difficulty are the fluctuations in the computation load of each frame and in the amount of the available CPU resource.

To solve this problem, we have proposed a high-level video processing library *RaVioli* (Resolution-Adaptable Video and Image Operating Library)[1], [2] which guarantees pseudo real-time processing on general-purpose computing systems. RaVioli can regulate the throughput rate by automatically modifying spatial resolution and frame rate according to CPU usage and load. For such dynamical modification of the resolutions, a programming fashion which is independent of the resolutions is required. RaVioli conceals two resolutions, frame rate and the number of pixels, from programmers for changing the resolutions at run-time. It enables RaVioli to exclude the concept of resolutions from video processing programming, and developers can write video processing programs more intuitively.

With RaVioli, developers can implement real-time processing without considering the fluctuations in the computation load of each frame and the amount of the available CPU resource, but sometimes the output precision may be too much roughened. This occurs due to reducing resolutions for load-adjustment, and it is difficult to ensure keeping output precision high. However, developers expect the input image to be processed with as possible as high resolutions.

In this paper, we propose a new model of load-adjustment by tiling with different spatial resolutions. It divides the whole image into several tiles, and roughens the spatial resolution of each tile according to how important the region enclosed in the tile is. It can reduce the total amount of the computation load, and the spatial resolution of the region which should be processed precisely may be prevented from being roughened. To achieve this, we have implemented a method for automatically dividing whole video frames into several subframes and changing each spatial resolution individually on RaVioli.

II. Related Works

A. Real-Time Video Processing

So far, several real-time video processing applications have been developed. For example, Garcia-Martin et al.[3] have presented a moving people detection for surveillance video systems. Kim et al.[4] have proposed a method for early smoke detection. Lin et al.[5] have presented a real-time eye detection algorithm.

For such a real-time video processing, the scheduling of image processing and adjusting the processing load are very important. Some scheduling methods for image processing have been developed. Lee et al.[6] have proposed a static scheduling scheme for optimizing the parallel time of image processing operations involving multiple stages, loops, and data dependent operations. Although this scheduling method can reduce parallel execution time significantly, it does not ensure that image processing operations can be processed in a given constant time. Kywe et al.[7] have proposed an adaptive scheduling method using anytime algorithm, under such a

condition that the processing time is restricted in the duration of one frame. With the scheduling technique, the maximum performance can be achieved in the restricted time. However, it is necessary to modify conventional image processing algorithms to anytime algorithmic image processing.

On the other hand, some methods for adjusting the processing load also have been developed. Writing multiple routines with different algorithms has been the most-used solution for the load adjustment. One example is the imprecise computation model (ICM)[8], [9]. In this model, computation accuracy varies corresponding to the given computation time limit. With the confidence-driven architecture, which is based on the ICM, developers have to troublesomely implement multiple routines with different algorithms and different loads, and the confidence-driven architecture selects suitable routine dynamically and empirically among them.

A frame skipping method, one of the other solutions, has been proposed in [10]. This is based on the fact that not all frames are equally important for the overall video quality. If there is not enough resources for fully processing the incoming video, the processing of less essential frames is omitted according to a quality-aware frame skipping rule. Another solution also has been used in the eye detection[5]. In the detection algorithm, by adjusting the scale to enlarge the frame, the detection speed can be increased. Here, the scale refers to the zoom scale of the frame. After the detection in each frame, the scale is automatically changed according to the area of detected regions. These two studies are similar to RaVioli, however, RaVioli can apply both solutions to any video processing applications automatically.

B. Image/Video Processing Libraries and Languages

On the other hand, many image/video processing libraries have been developed. For example, VIGRA[11], OpenCV[12], [13], OpenIP[14] and Pandore[15] are well-known image/video processing libraries. Adopting template techniques similar to the C++ STL, VIGRA allows developers to easily adapt given components to their programs. OpenCV provides many typical image/video processing algorithms as C functions or C++ methods. OpenIP provides a set of interoperable, open source libraries, satisfying the demands of image processing and computer vision in education, research and industry, as well. Pandore provides a set of executable image processing operators. It is dedicated to image processing experts because skills on image processing operations are needed to use this library. These libraries provide high-level descriptivity of image/video processing, but adjusting computation load is difficult to be implemented with them.

Some programming languages for image processing also have been developed. A loopless image processing language[16], for example, allows developers to implement image processing for embedded devices without any knowledge about the processors or memory architectures. With this language, developers can operate arrays without using loops in programs with some special operators. However, developers have to write programs with a formula editor and consider array sizes. A whole image processing language called picture processing languages (PPL)[17] is also developed. The general idea of PPL is to encapsulate image processing algorithms. Thus, programmers do not have to deal with each individual pixel when implementing image processing algorithms, and this enables programmers, who know little about digital image processing, to write image processing programs. Although PPL incorporates common image processing algorithms, developers should extend the PPL if they want to use a new image processing algorithm. Using the API provided by PPL, developers can add more customized functions or methods, however, it is not easy for the end user.

The approach of the library RaVioli[1], [2] is completely different from existing computation models or image/video processing libraries and languages. RaVioli allows programmers to be unaware of the existence of pixels and frames through their video processing programming. Concealing pixels and frames from programmers, RaVioli can change spatial/temporal resolutions and can adjust processing load dynamically and automatically.

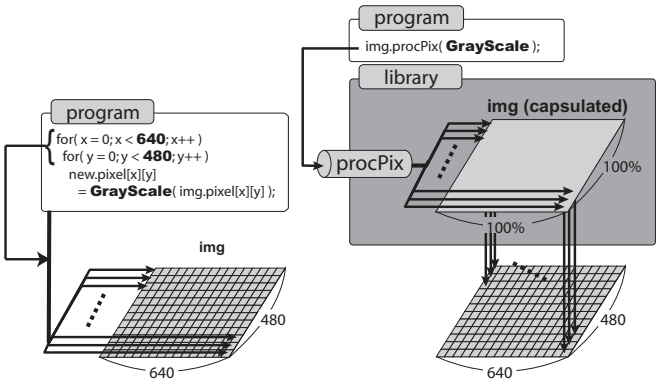
III. Overview of RaVioli

A. Abstraction of Video Processing

RaVioli proposes a new programming paradigm with which programmers can write video processing applications intuitively. RaVioli conceals *spatial resolution* (pixel rate) and *temporal resolution* (frame rate) of a video from programmers. We human beings naturally have no concept of resolutions through our visual recognition. For example, we can recognize object motion in our view without any pixel or frame. However, pixels and frames are indispensable for motion object detection programs on computer systems.

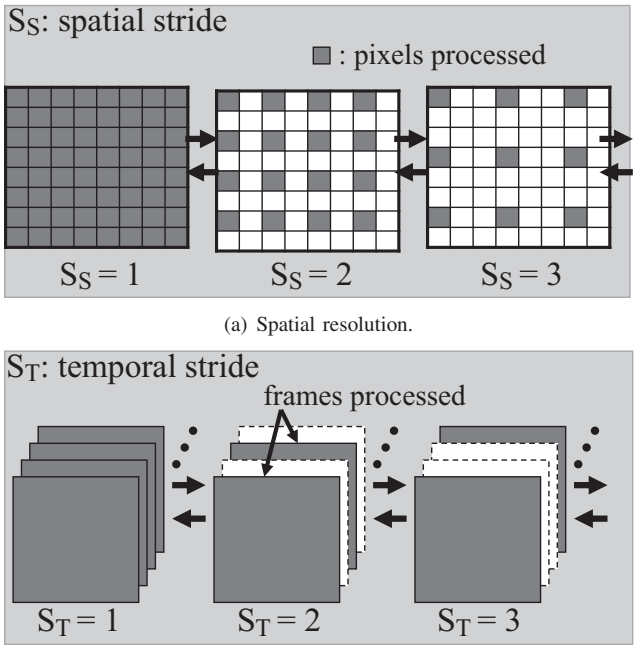
For example, motion object detection programs are sometimes implemented by using a block matching algorithm, which searches for the most similar block between current window and previous one. The similarity between image windows will be calculated by SAD (sum of absolute differences) or another alternative method, and the method should be implemented by cumulative pixel value differences. Resolutions are delivered from the requirement of quantitativeness on computers. Hence, developers have to manage resolutions in their programs although resolutions are not required essentially for vision. In other words, the presence of resolutions makes video processing programs unintuitive.

Generally, loop iterations are heavily used in video processing programs. When converting a color image to grayscale, for example, each pixel will be converted to grayscale in innermost iteration, and the process is repeated for every pixels by loop nests as shown in Fig. 1(a). On the other hand, with RaVioli, an image is encapsulated in an RV_Image instance, and this repetition for all pixels is done by RaVioli automatically, so developers should only write a routine for one pixel as shown in Fig. 1(b). GrayScale() in Fig. 1(b) is the routine defined by the developer. What developer should do are defining a function which processes one pixel and passing the function to one of the image instance's public methods. In this



(a) Traditional program. (b) Program with RaVioli.

Fig. 1. Digital image processing.



(a) Spatial resolution.

(b) Temporal resolution.

Fig. 2. Resolution changes.

example, the method is procPix(), which is defined as a higher-order method of the RV_Image class. It applies a function passed as its argument to all pixels in the RV_Image instance one after another. This framework allows developers to be released from resolutions and the number of iterations. Not only procPix(), RaVioli also provides some higher-order methods for several processing patterns; such as template matching, k-neighbor processing, and so on. As same as images, videos are also encapsulated in RV_Streaming instances in RaVioli. Frames, the components of an RV_Streaming instance, are concealed from developers. An RV_Streaming instance also has several higher-order methods. Developers should only define a *component function*, which manages one frame, and pass the function to an appropriate higher-order method for video processing.

B. Self-Adjustment of Computation Load

RaVioli can dynamically change video resolutions considering processing load. RaVioli periodically compares the frame capturing interval and the processing time for one frame. When the processing time becomes larger than the capture interval, RaVioli considers it is overloaded and reduces resolutions. There are two resolutions; spatial resolution and temporal resolution in videos. Spatial resolution refers the number of pixels contained in each frame, and temporal resolution refers the frame rate. RaVioli applies component functions to frames or pixels skipping on a certain stride in higher-order methods mentioned above. Roughening resolutions can be done by raising the stride value, and it leads to decreasing the computation load. Fig. 2(a) shows which pixels are processed when spatial stride increases, and Fig. 2(b) shows which frames are processed when temporal stride increases.

Priorities can be specified for telling RaVioli which resolution (spatial or temporal) should be kept. In a real-time video application, top priority will be given to temporal resolution, and RaVioli reduces spatial resolution. In other applications such as face authentication, top priority will be given to spatial resolution, and RaVioli reduces temporal one. What developers should do for load adjustment is only specifying priorities.

The resolution priority is specified by a tuple of two values (P_S, P_T) called a *priority set*. P_S represents the priority of spatial resolution, and P_T represents the priority of temporal resolution. When (P_S, P_T) = (3, 7) is specified, the priority ratio of P_S and P_T is recognized as 3:7, and RaVioli manages to keep spatial stride and temporal stride in the ratio of 7:3. Therefore a video processing application, which fulfills the performance demand and realizes real-time processing, can be easily implemented.

This algorithm for reducing resolutions is very simple and naive. However, this simplicity is very important. Many complement algorithms such as bi-linear, hyper-cubic, and so on are well known and they can be used. However, notice that the function of changing resolutions of RaVioli aims at reducing calculations. Adding calculations for changing resolutions makes no sense.

An application written with RaVioli can achieve real-time processing without considering the fluctuations in the computation load of each frame and the amount of the available CPU resource, but sometimes the output will have low quality. Although this result occurs due to reducing resolutions, developers expect the input image to be processed with high resolutions. For this problem, developers already can control the inconvenience from quality loss by defining priority set appropriately. The resolution which is given top priority can be kept relatively high. On the other hand, the other resolution is roughened a lot when computation load is high. Thus this remains as a problem for RaVioli.

IV. Tiling with Different Spatial Resolutions

A. Outline of a New Load-Adjustment Model

As described in III-B, RaVioli achieves load self-adjustment by changing resolutions. However, there is a limit on reducing resolutions. If an input frame is processed with drastically roughened spatial resolution by RaVioli, the output may not be the result which developers expect. To prevent this situation, this paper proposes a new model of load-adjustment which focuses on the characteristics of input video frames in real-time video processing.

In real-time video processing, input video frames are captured and processed with a constant time interval, but not all frames need to be processed precisely. In a real-time intruder detection system, for example, if there is no motion object in an input video frame, the changeless frame is not necessary to be processed precisely. Even when there are some motion objects in a frame, most part of the frame should be changeless. Such a part is not also necessary to be processed precisely. Similarly, the parts which do not have to be processed precisely would be found in many other real-time video processing applications. Processing such parts precisely leads to waste of the CPU resources. When enough CPU resources cannot be provided, RaVioli tries to adjust processing loads by roughening the resolutions, so processing these useless parts leads to low resolution all around the frame.

Hence, this paper proposes a new model, with which each input video frame is divided into several subframes, and the spatial resolution of each subframe is changed individually based on whether the subframe needs to be processed precisely. Besides the spatial stride of the whole frame, the new RaVioli handles the spatial strides of each subframe individually. The spatial stride for the whole frame is used as for subframes which need to be processed precisely. We call this *base stride*. For the other subframes, RaVioli manages another greater stride value than the base stride. We call this *rough stride*. In this way, it is possible to reduce wasted processing, and RaVioli can achieve real-time video processing without excessively roughening resolutions.

B. Processing Model

Now, we will illustrate the new processing model with a surveillance application, and compare the new model to the traditional processing model of RaVioli. How both surveillance applications based on the traditional and the new model work is shown in Fig. 3. The input video stream is shown in the upper part of Fig. 3. For ease of explanation, we focus on four frames in the stream. The frames are labeled from #1 to #4 according to the captured order. The output stream with traditional RaVioli and that with the new model are shown in the middle and the lower part of Fig. 3 respectively.

Suppose that RaVioli processes all frames of input video stream in order not to miss intruders. Thus, RaVioli changes only the spatial resolution for adjusting the processing load.

To begin with, let us see how a surveillance application with the traditional model works. It is shown in the middle part of Fig. 3. The traditional model tries to process the all input video frames with as possible as high spatial resolution, so even the frame #1, where there is no change, is processed precisely. If the amount of the available CPU resource decreases when processing the frame #1, the next frame #2 is processed with rougher spatial resolution. Because RaVioli keeps on incrementing the spatial stride while the processing time is larger than the capture interval, roughening the spatial resolution often continues through several input video frames. In this example, the frame #3 and #4 is processed with roughened spatial resolution, and it is difficult to detect a intruder precisely because of low image quality.

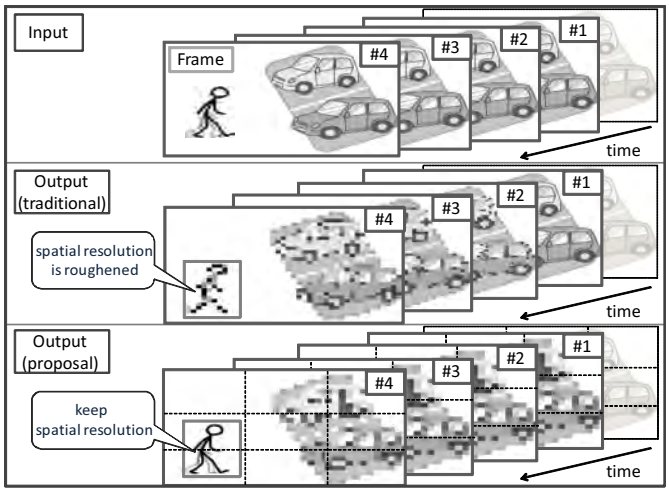


Fig. 3. Surveillance application based on traditional RaVioli and new RaVioli.

On the other hand, in the new model, each input video frame is divided into several subframes, or *tiles*, in accordance with the parameter defined by developers. Then, RaVioli determines whether each tile needs to be processed precisely. The base stride is used for the tiles each of which covers some variable region, and the rough stride is used for the other tiles which have no changes in them.

Now, let us see how a surveillance application with the new tiling model works. It is shown in the lower part of Fig. 3. In this figure, vertical and horizontal dashed lines in a frame represent borderlines of each tile. This example shows the case where the developer indicates that each video frame should be divided into 3×3 tiles. Unlike the traditional RaVioli, the new tiling model applies the rough stride to the tiles which need not to be processed precisely. Thus, the total amount of the computation load can be reduced. Therefore, even if the amount of the available CPU resource decreases, the base stride can be prevented from being drastically roughened. When processing the frame #4, the left-middle and the left-lower tiles where a prowling intruder is captured are decided to be processed precisely, and the base stride is used for them. The base stride with the new model will be smaller than the spatial stride for the frame #4 with the traditional

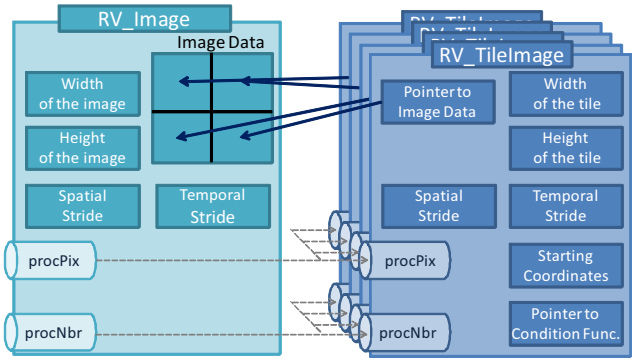
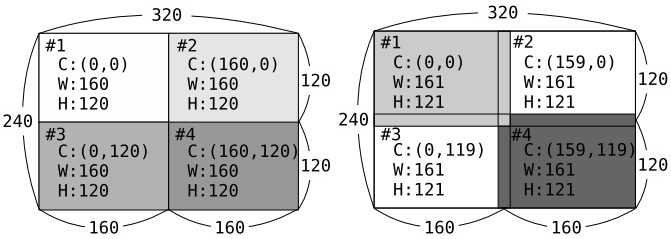


Fig. 4. Outline of RV_TileImage.



(a) procPix, procImgComp. (b) procNeighbor.

Fig. 5. Tiling in several higher-order methods.

RaVioli, because the new model can reduce the computation load during the frame #1 through to the frame #3. Thus, it is possible to detect the intruder more precisely.

As described above, the new model enables RaVioli to reduce the total computation load by roughening the spatial resolution of the region which do not have to be processed precisely. Therefore, the spatial resolution of the region which should be processed precisely can be prevented from being drastically roughened.

V. Implementation

In this section, an implementation for the tiling model with different spatial resolutions will be described. First, a new class RV_TileImage which encapsulates a divided subframe will be described. Next, modification for higher-order methods of RV_Image class will be described, and the function for deciding whether a subframe needs to be processed precisely will be described. We call this function *condition function*. Finally, the flow of image processing with the new model will be described.

A. RV_TileImage Class

In the new model, each video frame is divided equally in order to change the spatial resolution of each subframe, or tile, individually. To accomplish this, a new class RV_TileImage which represents a divided tile is implemented on RaVioli. The outline of RV_TileImage class is shown in Fig. 4. Each RV_TileImage instance is associated with an RV_Image instance which it belongs to. As shown in Fig. 4, an RV_Image instance has width and height of the image, both spatial and temporal stride, space for holding image data, and some higher-order methods as its member. On the other hand, an RV_TileImage instance has width and height of the tile, both spatial and temporal stride, some higher-order methods, starting coordinates which are the coordinates of the upper-left of the tile, and a pointer to a condition function as its member. Instead of holding pixel data, RV_TileImage instances have a pointer to the space for image data which is allocated by its associated RV_Image instance.

When a video frame is processed with the new model, RV_TileImage instances are constructed as many as number of tiles. Each instance processes its responsible region in a video frame, so whole of the frame is processed properly. A region which is processed by an RV_TileImage instance is defined in terms of the starting coordinates, width and height of the tile held by the RV_TileImage instance. An RV_TileImage instance applies a component function repeatedly by means of a suitable higher-order method as same as an RV_Image instance in the traditional RaVioli. In this way an RV_TileImage instance processes the corresponding region of image data which is held by an RV_Image instance.

B. Higher-order Methods

The new model should be implemented so that developers do not have to rewrite component functions which are already written for the traditional RaVioli. Therefore, when an RV_Image instance applies a component function to its video frame, the instance should invoke RV_TileImage instances' higher-order methods instead of executing its own.

Now, we have modified RV_Image instance's higher-order methods. For changing the spatial resolution of each tile individually, an RV_Image instance creates RV_TileImage instances as many as the number of tiles and processes whole frame by means of several RV_TileImage instances as described above.

The processing patterns of higher-order methods are different each other such as point processing, k-neighbor processing, template matching, and so on. Therefore, the size of region, especially the overlap width, for one RV_TileImage instance is different among higher-order methods. Thus, the way of dividing whole frame into several tiles should be different in each higher-order method. Now, we will show some representative higher-order methods, and explain how a frame should be divided for the methods. In the following descriptions, it is assumed that a video frame is divided into four(2×2) tiles.

procPix(void(*CF)(RV_Pixel *P))
: This higher-order method receives a pointer to a component function CF which processes one pixel P and applies it to all pixels one after another. Thus, a video frame is divided into four tiles simply as shown in Fig. 5(a). Here, in Fig. 5(a), C, W and H represent the starting coordinates, the width and the height of the tile respectively.

procImgComp(void(*CF)(RV_Pixel *P, P_c), RV_Image I_c)

This higher-order method receives a pointer to a component function *CF* which processes one pixel *P*, using the corresponding pixel P_c in a image I_c as reference and applies it to all pixels one after another. This method can be used for calculating frame-to-frame difference, calculating degree of similarity between two frames, and so on. Because the processing pattern of procImgComp is same as that of procPix(), two video frames are divided as shown in Fig. 5(a).

procNeighbor(void(**CF*)(RV_Pixel **P*, **P_{nbr}*, int *k*))

This higher-order method receives a pointer to a component function *CF* which processes one pixel *P*, using an array of RV_Pixel instance P_{nbr} which contains *k* nearest neighbor pixels (*k*=8) as reference, and applies it to all pixels one after another. With this method, a video frame is divided as shown in Fig. 5(b). The region #1 contains the top left quarter of the image and one pixel margin on the right and lower side of the region. Similarly, the region #2, #3 and #4 contain each quarter image accompanying with its one pixel margin on the left and lower side, right and upper side, and left and upper side respectively.

C. Condition Function

To change resolutions of all tiles automatically and individually, each RV_TileImage instance needs to determine whether the tile should be processed precisely or not. In the new model, *condition functions* are applied to video frames using the processing mechanism as same as component functions. Developers can define a condition function which determines whether a tile needs to be processed precisely, and pass the function to an RV_Streaming instance through its higher-order method. RaVioli uses this function for changing spatial resolution of each tile. Furthermore, RaVioli provides developers with some predefined condition functions. Condition functions should return 1 if the tile needs to be processed precisely, otherwise return 0.

Now, we show a sample video processing program implemented with the new RaVioli model in Fig. 6. In this sample program, the component functions `pixF`, `imgF` and a condition function `FleshDetect` are defined by the developer. In main function, as a first step, an RV_Streaming instance `video` is declared at the line 9. Then, the priority set is designated by means of the `setPriority` method of the RV_Streaming instance at the line 10. The instance `video` changes both spatial and temporal resolutions according to the value of the priority set. Next, a condition function should be defined and passed to the higher-order method `setCondFunc`. Now, the specifications of condition functions and a higher-order method `setCondFunc` are as follows.

void setCondFunc(int(**CdF*)(RV_Image **F_c*))

void setCondFunc(int(**CdF2*)(RV_Image **F_c*,**F_p*))

This method receives a pointer to a condition function *CdF* which uses the current processing frame F_c, or a pointer to a condition function *CdF2* which uses both F_c and the previous frame F_p. This assigns the function pointer to RV_TileImage instance's pointer variable shown in Fig. 4.

```
1  void pixF(RV_Pixel *pix){...}
2  void imgF(RV_Image *img){
3      img->procPix(pixF);                  // a higher-order method
4  }
5  int FleshDetect(RV_Image *Curr, RV_Image *Prev){
6      /* determine whether precisely or not */
7  }
8  int main(int argc, char* argv[]){
9      RV_Streaming video;
10     video.setPriority(7, 3);             // set priority set
11     video.setCondFunc(FreshDetect);      // set condition func.
12     // video.setCondFunc(FrameDiff);     // select condition func.
13     video.setTileNum(3, 4);              // divide into 3x4 tiles
14     video.procStream(imgF);              // higher-order method for streaming
15 }
```

Fig. 6. Description of new RaVioli program.

In this sample program, the programmer-defined condition function `FleshDetect` is passed to the higher-order method `setCondFunc` at the line 11. A predefined function can be designated instead of programmer-defined condition function. `FrameDiff` in line 12, which is commented out, is one of the predifined condition functions. It returns 1 if the difference between the tiles in adjacent frames is greater than a certain threshold, otherwise returns 0. Then, how many tiles the whole frame is divided into is designated at the line 13. The component function `imgF`, which is applied to all video frames, is passed to `video`'s higher-order method `procStream` at the line 14. In the `imgF`, the higher-order method procPix() is invoked at the line 3. In the method procPix(), each RV_TileImage instance changes the spatial stride by means of the condition function and processes the corresponding region with the spatial stride. As described above, the process for a video frame can be applied to all frames.

D. Image Processing Flow

This subsection shows how a video processing program runs with the new model of RaVioli. First, a video frame is divided into several tiles. The number of tiles is defined by developers. Now, suppose that the number of tiles is designated as 2×2. In this case, four RV_TileImage instances are constructed to process the corresponding tile individually. This is illustrated in the left side of Fig. 7. Each RV_TileImage instance determines the tile should be processed with whether base stride or rough stride, and processes the tile with the appropriate spatial stride as shown in the center of Fig. 7. In this figure, each square represents a pixel, and a filled square represents a processed pixel. The case, where upper-left and lower-right tiles are processed with base stride and upper-right and lower-left tiles are processed with rough stride, is shown in this figure. The value of rough stride is now 2. Therefore,

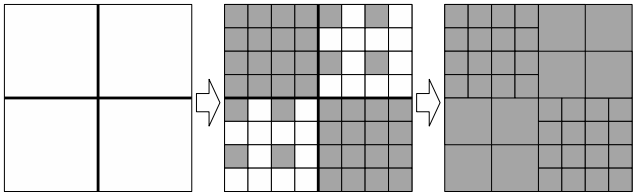


Fig. 7. Image Processing Flow.

TABLE I
EVALUATION ENVIRONMENT.

OS	Solaris10
CPU	Intel Core2 Ex. Quad
Frequency	3.0GHz
Memory	8GB
Compiler	Sun Studio 12 (Sun C++ 5.9)
Compile options	-fast

as you can see, there are some unprocessed pixels in upper-right and lower-left tiles. For these unprocessed pixels, the result of a left or upper or upper-left neighbor pixel is used for output instead of them. Thus, the output of the whole video frame is as shown in the right side of Fig. 7. By processing a video frame along this flow, a load-adjustment by tiling with different spatial resolutions is achieved.

VI. EVALUATIONS

We evaluated the new tiling model proposed in this paper. The evaluation environment is shown in TABLE I. We used a grayscale program for this evaluation. The input video stream is composed of 50 frames and the spatial resolution is 320×240. In this stream, there are changes in the input frames from the 30th frame through the 34th. Furthermore, from 20th frame to 40th frame, we executed another program which has heavy computation load for making a situation where available CPU resource decreases drastically. We have evaluated following three models,

(B) Traditional RaVioli.

($T_{3,4}$) Tiling model using *FrameDiff* described in V-C as a condition function, and a frame is divided into 3×4 tiles.

($T_{8,8}$) Tiling model as same as ($T_{3,4}$) but with 8×8 tiles.

The fluctuation of the base stride is shown in Fig. 8. As you can see, tiling model ($T_{3,4}$) and ($T_{8,8}$) can keep base stride 1 or 2 lower than model (B) from 30th to 34th frame. The output images at the 33rd frame are shown in Fig. 9. Fig. 9(a), Fig. 9(b) and Fig. 9(c) show the output image with model (B), model ($T_{3,4}$) and model ($T_{8,8}$) respectively. With model (B), the whole input image was processed with single spatial resolution, and the region which should be processed precisely was roughened as same as the other region. In contrast, with the new models ($T_{3,4}$) and ($T_{8,8}$), the region where a person moves was processed with finer spatial resolution than that for the other region. Especially, with model ($T_{8,8}$), more regions are processed with rough stride than model ($T_{3,4}$). Hence,

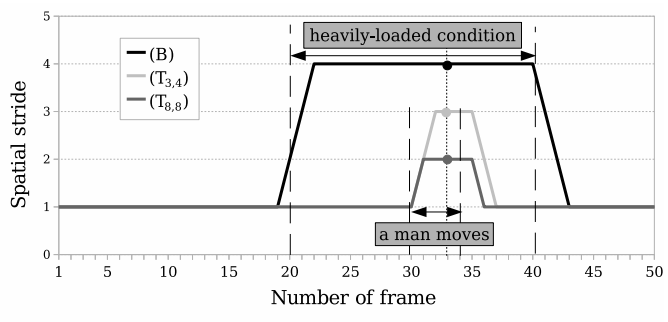


Fig. 8. Fluctuation of the spatial resolution.

model ($T_{8,8}$) was able to keep the base stride lower than the base stride of model ($T_{3,4}$).

As we can see in Fig. 8 and Fig. 9, tiling model can process the regions, where a person moves, with the base stride properly, and the spatial stride for the region with model ($T_{8,8}$) is up to 2 times smaller than that with model (B). Therefore, it is obvious that the region where a person moves is processed precisely. Thus, the output precision of the region is highly improved.

VII. CONCLUSIONS

In this paper, we proposed a new load-adjustment model by tiling with different spatial resolutions for pseudo real-time video processing library RaVioli. RaVioli changes resolutions automatically for adapting to currently available CPU resource. In the new tiling model, RaVioli divides whole video frame into several tiles, and can change the spatial resolution of each tile individually. Additionally, developers can use this new model without rewriting component functions for the traditional RaVioli. Through an evaluation, it is found that the new model can keep the spatial stride of regions, which should be processed precisely, up to 2 smaller than that of the traditional RaVioli.

One of our future works is merging this model and parallelization. RaVioli already can parallelize video processing by applying automatic block decomposition for each frame and by providing an easy-to-use pipelining interface which can automatically balance loads between some processing stages [1]. Because the execution time of a video processing program can be reduced by these parallelization methods, the resolutions will be prevented from being roughened, and it will lead to improving the output precision. On the other hand, the new tiling model should be applied to several useful real-time video processing programs. To achieve this, we should examine some new higher-order methods for them. Designing a new video programming language which cooperates with RaVioli is also left for our future work.

ACKNOWLEDGMENT

This research was partially supported by a Grant-in-Aid for Young Scientists (B), #21700028, 2009, from the Ministry of Education, Science, Sports and Culture of Japan.

(a) model (B) (b) model ($T_{3,4}$) (c) model ($T_{8,8}$)

Fig. 9. The output images with the traditional model and the new models.

REFERENCES

[1] H. Sakurai, M. Ohno, T. Tsumura, and H. Matsuo, "RaVioli: a Parallel Video Processing Library with Auto Resolution Adjustability," in *Proc. IADIS Int'l. Conf. Applied Computing 2009*, vol. 1, Nov. 2009, pp. 321–329.

[2] K. Kondo, T. Inaba, H. Sakurai, M. Ohno, T. Tsumura, and H. Matsuo, "RaVioli: a GPU Supported High-Level Pseudo Real-time Video Processing Library," in *Communication Papers Proc. 19th Int'l Conf. on Computer Graphics, Visualization and Computer Vision (WSCG2011)*, Jan. 2011, pp. 39–48.

[3] A. Garcia-Martin and J. M. Martinez, "Robust Real Time Moving People Detection in Surveillance Scenarios," in *Proc. 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, ser. AVSS '10. IEEE Computer Society, Aug. 2010, pp. 241–247.

[4] C. Kim, Y. Han, Y. Seo, and H. il Kang, "Statistical Pattern Based Real-time Smoke Detection Using DWT Energy," in *Proc. International Conference on Information Science and Applications*. IEEE Computer Society, Apr. 2011, pp. 1–7.

[5] K. Lin, J. Huang, J. Chen, and C. Zhou, "Real-time Eye Detection in Video Streams," in *Proc. Fourth International Conference on Natural Computation - Volume 06*. IEEE Computer Society, Oct. 2008, pp. 193–197.

[6] C. Lee, Y. Wang, and T. Yang, "Static global scheduling for optimal computer vision and image processing operations on distributed-memory multiprocessors," in *Computer Analysis of Images and Patterns*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1995, vol. 970, pp. 920–925.

[7] W. W. Kywe, D. Fujiwara, and K. Murakami, "Scheduling of Image Processing Using Anytime Algorithm for Real-time System," in *Proc. the 18th International Conference on Pattern Recognition - Volume 03*, ser. ICPR '06. IEEE Computer Society, 2006, pp. 1095–1098.

[8] J. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise Computations," in *Proceedings of the IEEE*, vol. 82, Jan. 1994, pp. 83–94.

[9] H. Yoshimoto, N. Date, D. Arita, and R. Taniguchi, "Confidence-Driven Architecture for Real-time Vision Processing and Its Application to Efficient Vision-based Human Motion Sensing," in *Proc. of the 17th Int'l. Conf. on Pattern Recognition (ICPR'04)*, vol. 1, 2004, pp. 736–740.

[10] D. Isovic, "Flexible Media Processing in Resource Constrained Real-Time Systems," in *Proc. Eighth IEEE International Symposium on Multimedia*, ser. ISM '06. IEEE Computer Society, Dec. 2006, pp. 363–370.

[11] U. Köthe, "Generic programming for computer vision: The vigra computer vision library," http://hci.iwr.uni-heidelberg.de/vigra/, Sep. 2011.

[12] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision With the OpenCV Library*. O'Reilly & Associates Inc, 2008.

[13] *Open Source Computer Vision Library*, Intel Corp., 2001.

[14] G. Kovács, J. I. Iván, A. Pányik, and A. Fazekas, "The openIP Open Source Image Processing Library," in *Proc. ACM Multimedia 2010*, ser. MM '10. ACM, 2010, pp. 1489–1492.

[15] "Pandore: A library of image processing operators (Version 6.4). [Software]. Greyc Laboratory," http://www.greyc.ensicaen.fr/˜regis/Pandore, 2011.

[16] J. Segawa and T. Kanai, "The Array Processing Language and the Parallel Execution Method for Multicore Platforms," *The First International Symposium on Information and Computer Elements*, 2007.

[17] S. Wang, Z. Dong, J. X. Chen, and R. S. Ledley, "PPL: A whole-image processing language," *Comput. Lang. Syst. Struct.*, vol. 34, pp. 18–24, Apr. 2008.