Content-Aware Precision Control on a Real-Time Video Processing Library

Takuya MATSUNAGA*, Shinji OHIRA*, 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 addressing 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 solve this problem, this paper proposes a new load adjustment method for RaVioli, which can divide whole video frame into several sub-frames, and can process each sub-frame with appropriate resolution or precision automatically.**

***Index Terms*—Real-time video processing, load adjustment, video processing library.**

I. Introduction

Real-time video processing applications such as surveillance systems, smoke detection systems or automatic vehicle collision avoidance systems are now in demand. On the other hand, 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. However, it is still difficult to realize the real-time video processing on general-purpose systems. 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 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 dynamic modification of the resolutions, a programming fashion which is independent of the resolutions is required. RaVioli conceals two resolutions, *spatial resolution* (i.e. pixel rate) and *temporal resolution* (i.e. frame rate), from programmers for changing the resolutions at run-time. This can exclude the concept of resolutions from video processing programming, and developers can write video processing programs more intuitively.

However, resolutions can not be reduced without limitation, because too low processing precision may make its processing result worthless. It is ineluctable that the output precision is somewhat reduced for achieving realtimeness, but the precision should be kept as high as possible.

In this paper, we propose a new load adjustment model for RaVioli by managing resolution in each sub-area of input frames separately, considering the feature of each sub-area.

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.

As described in section I, it is difficult to implement real-time video processing applications on general-purpose systems. This is because that the amount of the available CPU resources or computation load can fluctuate. To solve this problem, 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)[6], [7]. In this model, computation accuracy varies corresponding to the given computation time limit. With the confidence-driven architecture [7], which is based on 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. On the other hand, RaVioli can apply load-adjustment to any video processing applications automatically.

B. Libraries and Programming Languages

On the other hand, many image/video processing libraries have been developed. For example, VIGRA[8], OpenCV[9], OpenIP[10] and Pandore[11] 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[12], 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.

Halide[13] is another programming language specialized for image processing. In Halide programs, the algorithm for image processing and parallelization procedure are separated, and developers can try several scheduling for parallelization without modifying the core algorithm for image processing. However, this means that Halide requires developers to have knowledge of not only image processing but also efficient parallelization.

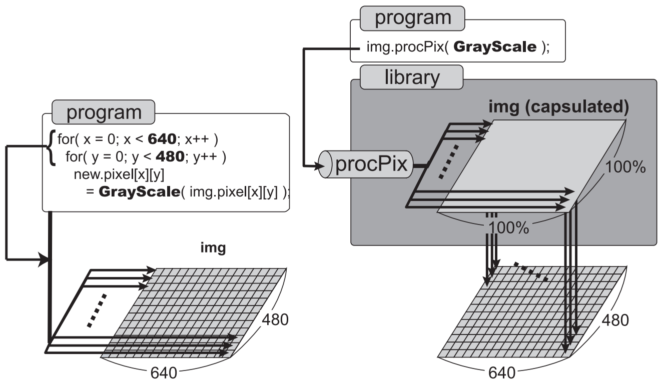
The approach of the library RaVioli[1], [2] is completely different from existing computation models or image/video processing libraries and languages. RaVioli allows programmers to be unaware of the existence of pixels and frames through their video processing programming. Concealing pixels and frames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

III. Overview of RaVioli

A. Abstraction of Image and Video Processing

Two resolutions, spatial resolution and temporal resolution of a video, are derived from the necessity of quantization on computers, and are not natural. We human beings naturally have no concept of resolutions through our visual recognition. For example, developers should consider resolutions for implementing a motion object detection application on computer systems, even though we can recognize object motion in our view without any pixel or frame. Hence, the presence of resolutions makes video processing programs unintuitive.

To solve these problems, RaVioli provides a new programming paradigm which conceals these two resolutions from programmers. Hence, with RaVioli, developers can implement video processing applications without considering pixels and frames. Developers also can easily implement real-time video


```
for( x = 0; x < 640; x++ )
  for( y = 0; y < 480; y++ )
    new.pixel[x][y]
      = GrayScale( img.pixel[x][y] );
```
```
img.procPix( GrayScale );
```

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

Fig. 1. Digital image processing.

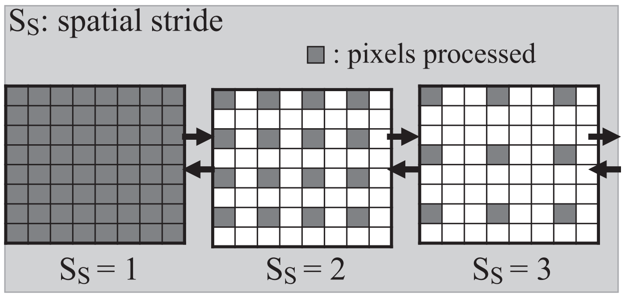
processing applications because RaVioli can automatically vary resolutions for adjusting computation load and achieving realtimeness.

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 pixel by loop nests as shown in Fig. 1(a). In the case of neighborhood processing such as blur or edge enhancement, the processed unit is a set of a pixel and its neighbour pixels, and in the case of template matching, the processed unit is a small window in each frame. These units are processed in innermost iteration, and the process is applied repeatedly by loop structures.

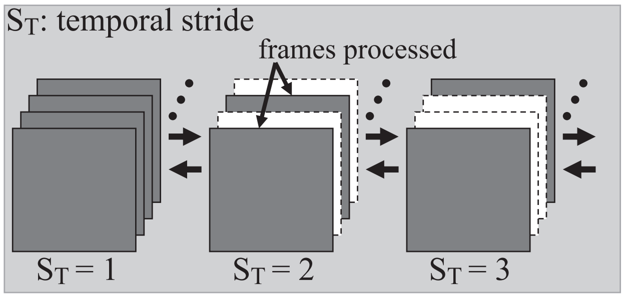
For using a loop structure for image processing, programmers should know the height and the width of the image for defining the number of iterations of the loop. On the other hand, with RaVioli, an image is encapsulated in an RV_Image instance, and this repetition for all pixels is applied by RaVioli 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 developers should do are defining a function which processes one pixel and passing the function to one of the image instance's public methods. We call this function a *component function*. In this 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, a video is also encapsulated in an RV_Streaming instance 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



(a) Spatial stride and resolution.



(b) Temporal stride and resolution.

Fig. 2. Changing processing resolution by stride access.

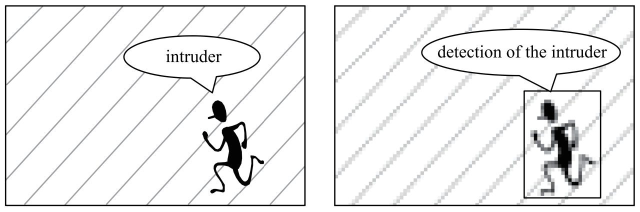
frame, and pass the function to an appropriate higher-order method for video processing.

B. Real-Time Processing by Adjusting Computation Load

On general purpose systems, multiple processes are running concurrently. Hence, it is difficult to implement real-time video processing applications on such systems, because there is no guarantee that enough CPU resources can surely be provided. One solution for this problem is reducing computation load by roughly processing the video data, or reducing the resolution of the video. RaVioli can change spatial and temporal resolutions dynamically and automatically according to the available CPU resources, because RaVioli conceals resolutions from programmers.

RaVioli has two internal parameters; *spatial stride* (S_S) and *temporal stride* (S_T), and RaVioli changes resolutions by varying these parameters. Fig. 2(a) shows the relation between the value of spatial stride and the processing spatial resolution. Initially the value of spatial stride $S_S = 1$, and all pixels in a frame are processed. When the value of spatial stride is increased to $S_S = 2$, every other pixel is processed and spatial resolution is roughened, and the whole processing load is reduced to $1/4$. For reducing the load more, spatial stride is increased to $S_S = 3$ and the load is reduced to $1/9$.

Likewise, the relation between the value of temporal stride and the temporal resolution is shown in Fig. 2(b). Initially, the value of temporal stride $S_T = 1$ and all frames are processed. When the value of temporal stride is increased to $S_T = 2$, temporal resolution is reduced by processing every other frame, and the whole computation load is also reduced to $1/2$. If the value of temporal stride becomes $S_T = 3$, the whole processing load is reduced to $1/3$.



(a) input (b) output

Fig. 3. An example of a surveillance system.

Programmers can specify *priorities* for telling RaVioli which resolution (spatial or temporal) should be kept high. In a hard 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.

C. Problem with RaVioli

RaVioli can automatically change resolutions for adjusting processing load according to the currently available CPU resources. Hence, the output sometimes will have low quality. Although this can not be avoided for achieving realtimeness, developers expect the input video frames to be processed with as high resolutions as possible.

For alleviating this problem, developers can control the inconvenience from quality loss by defining priority set appropriately. The resolution which is given top priority can be kept relatively higher than the other resolution. However, the other resolution is roughened a lot when computation load is high. Thus this remains as a problem for RaVioli.

Now, assume that a surveillance system is implemented with RaVioli. This application is required to detect an intruder in the input video stream, and to show the features of the intruder precisely. Hence, both of the spatial resolution and temporal resolution are very important for this application. However, the available CPU resources are limited. It is a severe problem if an intruder is missed because of too roughened temporal resolution, and the developer should give top priority to temporal resolution.

Fig. 3 shows an input and an output of this surveillance system. In the input frame shown in Fig. 3(a), an intruder is included. If this input frame is processed with a particularly low spatial resolution, the features of the intruder can not be

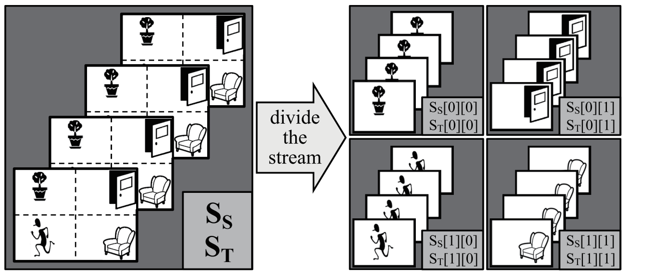


Fig. 4. Dividing video stream.

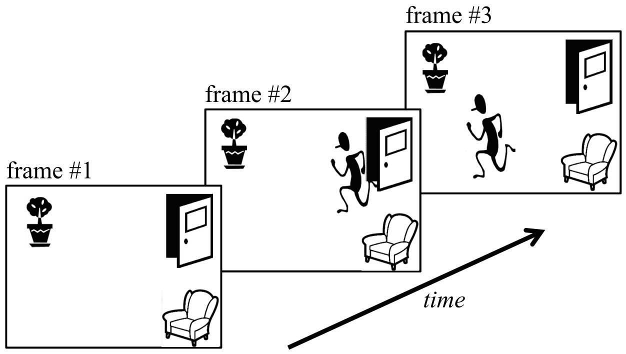


Fig. 5. An example of surveillance system.

clear in the processing result as shown in Fig. 3(b), and this result may be useless for the user of this application.

As described above, RaVioli achieves load-adjustment by changing resolutions. However, there is a limit on reducing resolutions. If an input frame is processed with a drastically roughened spatial resolution, the output may not serve developers purpose. To prevent this situation, this paper proposes a new load-adjustment model for RaVioli, considering the characteristics of input video frames in real-time video processing. With the new RaVioli, each input video frame is divided into several sub-frames, and each sub-frame can be processed with appropriate resolution or precision automatically, according to its importance for the processing. The proposed model eliminates wasteful processing in real-time video processing. Hence, this model alleviate the problem caused by reducing spacial/temporal resolutions.

IV. Adjusting Resolutions of Each Sub-frame

A. Basic Concept

In real-time video processing, some part of a frame will be important and should be processed precisely, but other part is not necessary to be processed precisely. For such perspective, we have proposed a tiling method with different spatial resolutions[2]. In this paper, we will improve this method.

Fig. 4 shows an example of dividing a video stream into four sub-streams. S_S and S_T represent spatial stride and temporal stride respectively. Each sub-stream has its own spatial resolution and temporal resolution, and can vary them independently of other sub-streams. Important areas should be processed precisely, hence corresponding sub-streams have high resolutions, or low stride values. On the other hand, unimportant areas should be processed in low resolutions with high stride values. In this way, the processing result with high precision can be compatible with lower computation load.

However, this method has a problem that it can not adapt rapidly to the change in the input video streams. An unimportant sub-stream has a low temporal resolution, and there will be a time lag from when a change emerges to when the change is detected in the input video. Moreover, such an unimportant sub-stream also has a low spatial resolution, and the processing result will have insufficient precision.

In response to this, we focus on the characteristics of "important areas." Each area does not become "important" at random completely, but there is a certain tendency. For example in Fig. 5, the programmer can predict that an intruder will appear from the door. Then, the intruder should move into one of the adjacent sub-area, unless he can teleport himself. Consequently, considering input video streams for real-time video processing, it can be predictable which sub-area will become important soon.

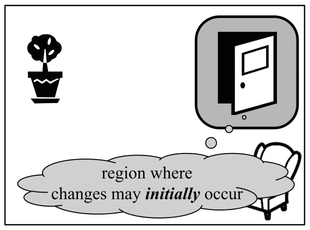
Based on this concept, we propose a new load-adjustment method considering changes in input video streams. In the proposed method, we define the areas, which may become important soon, as *preferential areas*, and make the resolutions of them be kept relatively high. Real-time video processing application written with RaVioli can rapidly adapt to the change in the input video streams with this method.

B. Preferential Areas

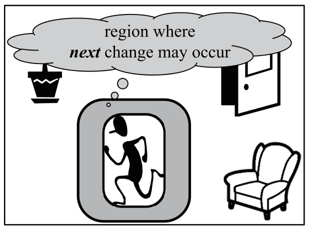
1) Areas where Some Changes will Occur: We regard the areas where some changes occur as important areas for real-time video processing, as described in the previous section. The location of important areas will vary because the contents of input video frames are changing every moment. However, there are some characteristics of such important areas.

First, it is somewhat predictable where input change will occur initially. Now, assume that the input frame shown in Fig. 6(a) is being processed. There is no important area in this input frame, and whole of the frame can be processed in low quality. However, if the resolution becomes too low, some changes in the input cannot be detected immediately, and the sub-frames which contain the changes can not be processed in appropriate precision for a little while. Now, notice that programmers will predict that the change in input will be brought by someone coming from the door. In real-time video processing applications, programmers will often predict where some changes will occur in the input frame.

Next, it is probable that there will occur some changes in the areas around the current important areas. Now, assume that the input frame shown in Fig. 6(b) is being processed. In this input frame, the area covering the running person is important, and should be processed precisely. The location of the important area will change, because the person is moving. However, it is predictable that the important area just after

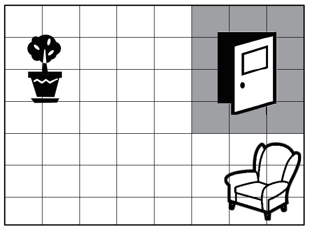


(a) Predictable area where some changes will occur initially.

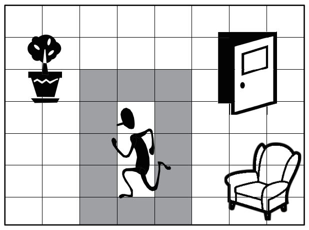


(b) Probable area where changes will occur next.

Fig. 6. Areas where changes will occur.

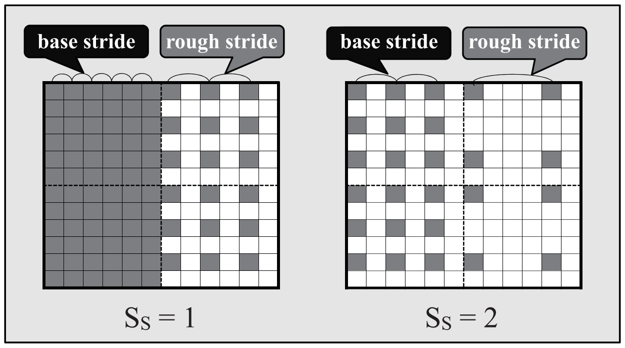


(a) Manual definition of preferential sub-frames.

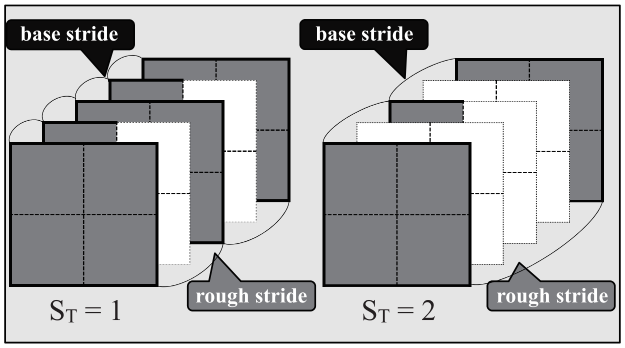


(b) Automatic location of preferential sub-frames.

Fig. 7. How to specify preferential sub-frames.



(a) Base and rough spatial strides.



(b) Base and rough temporal strides.

Fig. 8. Base stride and rough stride.

this frame will be located around the current important area, because the person can not disappear or appear suddenly.

Considering these two tendencies of important areas, we introduce a concept of *preferential area*, and keep the resolution for preferential areas relatively high even if there occurs no change in the area. Preferential areas can be defined manually and automatically.

2) Specifying Preferential Sub-frames:

a) How to Define Manually: For defining preferential areas, or preferential sub-frames, RaVioli provides an user interface for programmers. The interface is described in section V-B later. Using this interface, programmers can define some areas where changes will occur initially as preferential sub-frames. For example on the frame shown in Fig. 7(a), 12 sub-frames covering the door can be defined as preferential sub-frames by the programmer.

b) How to Locate Automatically: We installed a faculty of locating preferential areas automatically in RaVioli. Now RaVioli can manage the sub-frames around the current important sub-frames as preferential sub-frames, and keep their resolutions higher than unimportant areas. Hence, real-time video processing application implemented RaVioli can adapt to the change in the input video streams, and can generate high precision result. For example on the frame shown in Fig. 7(b), 12 sub-frames around the 3 sub-frames covering the running person are automatically managed as preferential sub-frames.

V. Implementation

A. Load-Adjustment

For achieving both of realtimeness of video processing and high precision processing results of important areas, multiple options of spatial/temporal stride should be provided for each video sub-stream. In this paper, we provide three strides; *base stride*, *rough stride*, and *medium stride* for important area, unimportant area, and preferential area respectively.

1) Base Stride and Rough Stride: Base stride is a stride option for important areas. Important areas should be processed as precise as possible, and the value of base stride is set as small as possible.

On the other hand, *rough stride* is a stride for unimportant areas. Unimportant areas should be processed in low resolution to reduce whole processing load for achieving real-time video processing. Hence, the value of rough stride is defined as larger than base stride.

Now, Fig. 8(a) and Fig. 8(b) shows how base stride and rough stride are applied to spatial and temporal resolutions. In both figures, a video stream is divided into 2×2 sub-streams, and the boundaries are depicted by dashed lines.

For ease of explanation, the value of rough stride is twice as base stride, and assume that left two sub-areas are important and right two sub-areas are not important, in this example. First, when spatial base stride $\mathrm{S_S} = 1$, unimportant sub-areas are processed with rough spatial stride 2, as shown in Fig. 8(a) In this case, the whole processing load of a frame is reduced to $5/8$, compared with previous RaVioli which processes the whole frame with $\mathrm{S_S} = 1$. When $\mathrm{S_S} = 2$, the value of rough stride is 4, as shown in the right side of Fig. 8(a). In this case, the whole processing load of a frame is reduced to $5/32$.

On the other hand, when temporal base stride $\mathrm{S_T} = 1$,

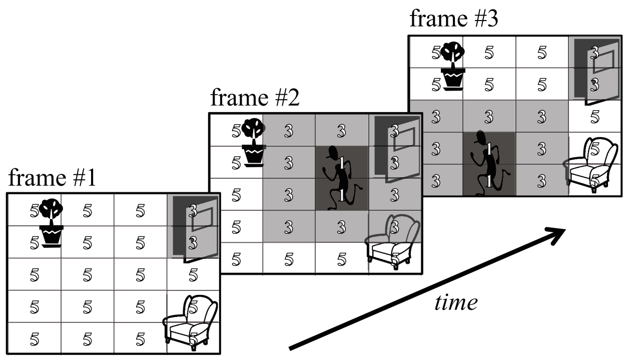


Fig. 9. An example of locating medium stride areas.

```
1  void ForPixel(RV_Pixel *pixel){
2    /* Processing for each pixel. */
3  }
4  void ForImage(RV_Image* Frame){
5    Frame−>procPix(ForPixel);
6  }
7  int FrameDiff(RV_Image *Curr, RV_Image *Prev){
8    /* determine whether precisely or not */
9  }
10 int main(int argc, char* argv[]){
11   RV_Streaming video;
12   video.setPriority(7,3);            // priority set
13   video.setCondFunc(FrameDiff);  // condition func.
14   video.setTileNum(5,6);             // divide input into 5x6 tiles
15   video.setTilePriority(0,4,1,5);    // specify preferential sub−areas
16   video.RunCapture();                // start capturing
17   video.StreamProc(ForImage);     // higher−order method for streaming
18   return 0;
19 }
```

Fig. 10. A sample program using the proposed method.

unimportant sub-streams are processed with rough temporal stride 2, as shown in Fig. 8(b). In other words, the unimportant areas in one of the each two frames will not be processed and the result of the previous frame is used. In this case, the whole processing load of this video is reduced to $3/4$. When base temporal stride $\mathrm{S_T}$ becomes 2, rough temporal stride becomes 4, as shown in the right side of Fig. 8(b). In this case, the whole processing load is reduced to $3/8$.

As mentioned above, the proposed method adjusts the whole computation load by reducing the precision for unimportant sub-areas, and keep the precision for important sub-areas.

2) Medium Stride: As mentioned in section IV-B, we define *preferential area* for adapting rapidly to the changes in input videos. Hence, the new stride *medium stride* is provided for preferential areas. The value of medium stride is set between the value of base stride and the value of rough stride. This means that, an area processed with medium stride can reduce computation load compared with base stride, and can detect changes rapidly in input videos compared with rough stride.

Now, let us see how medium stride is applied to sub-areas in Fig. 9. The digit on each sub-frame indicates the stride value of the sub-frame. In this example, the two sub-areas which include the door are manually defined as preferential areas, because it is predictable that some change may occur there. Assume that, the value of base stride is 1, rough stride is 5, and medium stride is 3. In the first frame, there is no change in the input, and the two preferential sub-frames use medium stride, and the other sub-frames use rough stride. In the second frame, an intruder appears from the door, and the input changes. Hence, base stride is applied to the sub-frames on the intruder, and medium stride is applied to the adjacent sub-frames. After that, if the intruder moves, appropriate strides are applied to all sub-frames according to the input as shown in the third frame in Fig. 9.

TABLE I
EVALUATION ENVIRONMENT.

OS	Fedora15
CPU	AMD Phenom II X4 965
Frequency	3.4GHz
Memory	8GB
Compiler	gcc 4.6.0
Compiler options	-O3

B. User Interface

For implementing applications with new RaVioli proposed in this paper, some user interfaces are defined. A sample program with new RaVioli is shown in Fig. 10. The condition function FrameDiff, and the component functions ForPixel and ForImage are defined by the programmer.

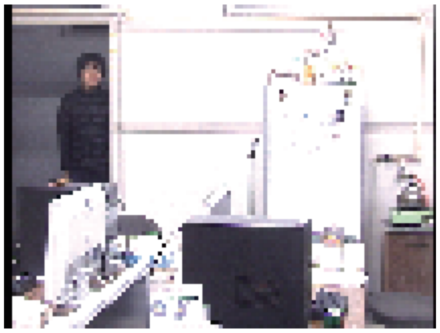
Condition functions take responsibility for deciding whether each area should be processed precisely or not, and the concept of them is described in [2] in detail. They should have one sub-frame or two temporary adjacent sub-frames as its argument(s), and return 1(true) if the sub-frame should be processed precisely or return 0(false) if not. For example, FrameDiff in this example is a condition function for deciding each sub-frame's preciseness based on the difference between temporary adjacent sub-frames.

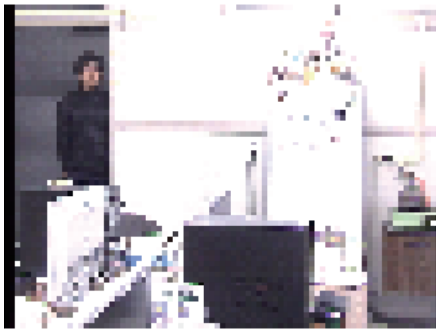
In `main` function at the line 10, the RV_Streaming instance `video` for handling the input video is generated (line 11), and priority set is defined (line 12). Next, the condition function `FrameDiff` is set (line 13), and the video stream is divided into sub-streams (line 14). Here, division is specified as 5×6, so the input video stream is divided into 30 sub-streams with five rows and six columns.

After that, preferential sub-frames are defined manually. Preferential sub-frames can be specified by calling the special function `setTilePriority` with the co-ordinates of upper-left and lower-right corner of the area (line 15). Finally, capturing is started with the description at the line 16, and video processing is started by passing the component function to a higher-order method of the instance `video` (line 17).

VI. Evaluation Results

We evaluated the new model of RaVioli proposed in this paper for confirming that important areas can be processed


(a) Output of the previous model.


(b) Output of the new model without medium stride.

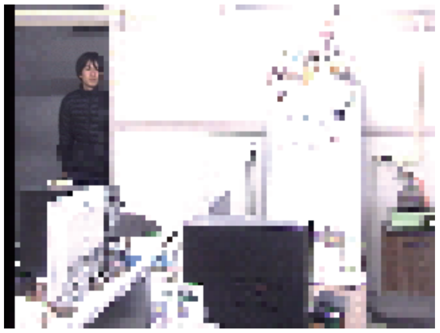
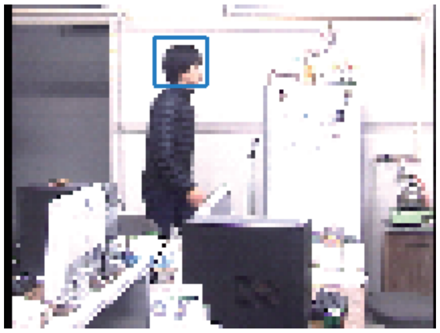
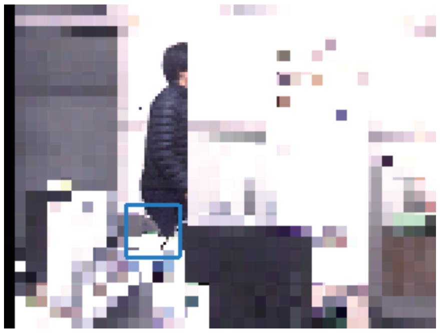

(c) Output of the new model with medium stride.

Fig. 11. Outputs of the 36th frames.


(a) Output of the previous model.


(b) Output of the new model without medium stride.

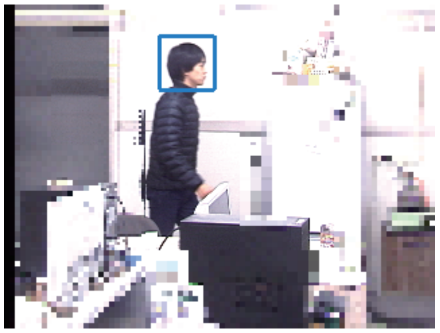

(c) Output of the new model with medium stride.

Fig. 12. Outputs of the 80th frames.

in finer resolution than the previous model. The evaluation environment is shown in TABLE I.

For this evaluation, we used a template matching program for searching a face in profile. The input video stream is composed of 120 frames and the spatial input resolution is 320×240. Each frame is divided into 9×9 sub-areas, and the function, which judges whether two adjacent frames have large difference or not, is used as the condition function. The priority set is defined as $(\mathrm{P_S}, \mathrm{P_T}) = (1, 1)$.

We evaluated and compared three models; (a) the previous RaVioli, (b) the new model without using medium stride, and (c) the new model with using medium stride. Fig. 11 shows each model's output of the 36th frame where some changes take place in the input frames, and Fig. 12 shows the each output of the 80th frame.

In the proposed models (b) and (c), the sub-frames around the door are manually defined as preferential areas. The results of template matching is drawn with blue boxes in Fig. 12.

First of all, in the 36th frames shown in Fig. 11, the person coming from the door can not processed precisely with both of (a) and (b). On the other hand with the proposed method (c), the appearance of the person can be distinguished, because the sub-frames around the door are manually defined as preferential and the processing can rapidly adapt the input changes.

Next, in the 80th frames shown in Fig. 12, the spatial resolution of the whole frame is drastically reduced for adjusting the computation load with (a). Hence, the face in profile of the walking person cannot be distinguished clearly. With (b), the person is just passing across a border between important area and unimportant area, and the template matching fails because the face part is not processed properly. On the other hand with the proposed method (c), the face in profile of the walking person is processed precisely. The above results lead to the conclusion that video processing applications with the proposed method can achieve both of realtimeness and high precision processing results.

Finally, the fluctuation of spatial base stride and temporal base stride are shown in Fig. 13 and Fig. 14 respectively. The horizontal axis shows the input frame indices, and the vertical axis shows the value of stride. Two vertical gray lines in each figure indicate the 36th and 80th frames shown in Fig. 11 and Fig. 12. With the proposed model (c) which employs medium stride, both of spatial and temporal base strides can be kept lower than the previous model. TABLE II shows the average values of spatial and temporal base strides of the previous model and the proposed model. As shown in this table, the precision of important areas can be kept higher with the proposed model using medium stride.

VII. Conclusion

In this paper, we proposed an improvement for real-time video processing library RaVioli. New RaVioli with this

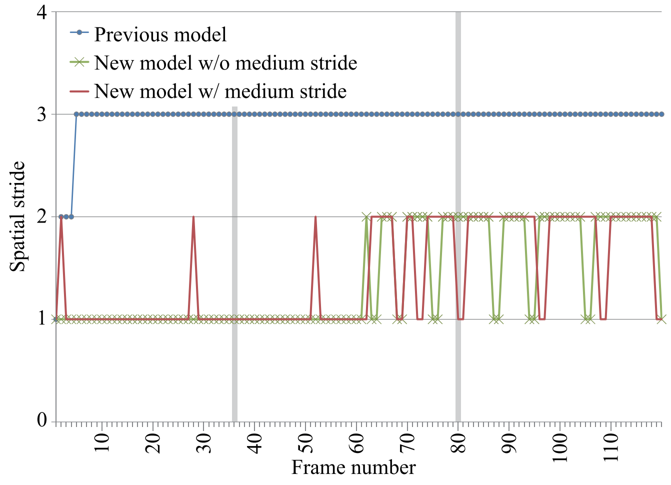


Fig. 13. The fluctuation of spatial base stride.

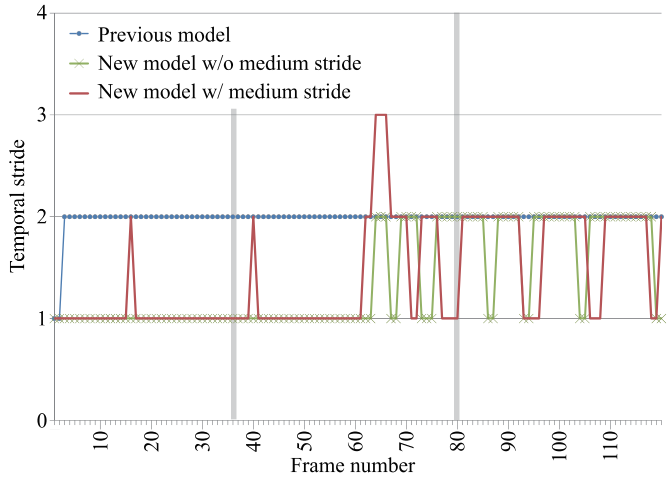


Fig. 14. The fluctuation of temporal base stride.

improvement can cope with appropriate load-adjustment for achieving realtimeness, both high quality output for important areas in video frames. With new RaVioli, the area which may become important can be managed as preferential area, and the preferential area can be defined manually and automatically.

Through an evaluation with a template matching program, it is found that the new RaVioli can achieve a high quality real-time processing by keeping the resolutions for important regions high and adapting the changes in input frames rapidly.

One of our future works is merging this improvement and an auto-parallelization mechanism on RaVioli. 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 further improvement of the output precision.

TABLE II
AVERAGE VALUE OF BASE STRIDES.

	spatial	temporal
Previous model	2.95	1.98
Proposed model w/ medium stride	1.40	1.40

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, A. Ono, T. Inaba, T. Tsumura, and H. Matsuo, "Tiling with Different Spatial Resolutions for Pseudo Real-Time Video Processing Library RaVioli," in *Proc. 7th Int'l Conf. on Signal-Image Technology and Internet-Based Systems (SITIS2011)*, Nov. 2011, pp. 253–260.

[3] A. Garcia-Martin and J. M. Martinez, "Robust Real Time Moving People Detection in Surveillance Scenarios," in *Proc. 7th IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance (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. Int'l Conf.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. 4th Int'l Conf. on Natural Computation*, vol. 06. IEEE Computer Society, Oct. 2008, pp. 193–197.

[6] 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.

[7] 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. 17th Int'l. Conf. on Pattern Recognition (ICPR'04)*, vol. 1, 2004, pp. 736–740.

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

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

[10] G. Kovács, J. I. Iván, Árpád Pányik, and A. Fazekas, "The openIP Open Source Image Processing Library," in *Proc. Int'l Conf. on Multimedia (MM'10)*. ACM, 2010, pp. 1489–1492.

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

[12] 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.

[13] J. Ragan-Kelley, A. Adams, S. Paris, M. Leboy, S. Amarasinghe, and F. Durand, "Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines," in *ACM Transactions on Graphics (TOG) - SIGGRAPH 2012 Conference Proceedings*. ACM, July. 2012.