Real-time segmentation of video on a multiprocessor platform

Constantin Arapis *, Simon Gibbs, Christian Breiteneder

GMD, Institute for Media and Communications, Visualization and Media Design, Schloss Birlinghoven, 53721 Sankt Augustin, Germany

Received 4 November 1996; revised 23 May 1997

Abstract

This paper presents a multiprocessor implementation of a segmentation algorithm for video streams. The segmentation procedure is part of an augmented-reality telepresence prototype. It has been implemented on a multiprocessor computer, equipped with specialized hardware for real-time graphics and video processing. The goal of this paper is to assess our experience in implementing an image processing procedure intended to meet the prototype's real-time requirements rather than present a new image processing technique. © 1997 Elsevier Science B.V.

Keywords: Multiprocessing; Image segmentation; Real time video processing; Teleconferencing; Augmented reality

1. Introduction

Image analysis and image processing techniques constitute basic building blocks for the development of many applications in the areas of multimedia, virtual reality, augmented reality and telepresence. However, existing image processing techniques may not satisfy the requirements of an application falling in the aforementioned application classes for the following two reasons: first, new increasingly stringent requirements characterize the design of such applications. Second, the solutions proposed for most basic image processing problems are not universally applicable [5] since they are domain dependent and sometimes application dependent. It is therefore likely that basic image processing techniques either would have to be tailored and/or tuned to an application's specific needs or new techniques would have to be devised.

For the development of an augmented reality telepresence prototype system, named...
TELEPORT [2], we had to implement an image segmentation procedure for video streams (separating background from foreground images). Our intention for writing this paper is to assess our experience in implementing an image processing algorithm intended to meet the prototype's real-time requirements rather than present a new image processing technique. The contents of this paper constitute an extended version of [1]. It has been enriched with the description of an additional implementation variation of the segmentation algorithm. We have also included more accurate performance measurements for both implementations of the segmentation algorithm.

This paper is structured in the following way. In Section 2 we give a brief description of TELEPORT by pointing out the usage of the segmentation procedure during operation and the physical conditions under which video stream images are captured and displayed. In the Section 3 we describe both the available hardware configuration for operating the segmentation procedure and the requirements we expect the segmentation procedure to satisfy. We present a high level description of the segmentation algorithm and discuss a set of issues related to the quality of the segmented images. Finally, we justify the need for a parallel implementation of the segmentation algorithm according to the performance measurements we have obtained. Section 4 describes the multiprocessor implementation of the algorithm, and Section 5 describes an improved implementation variation of the algorithm. Section 6 presents our conclusion.

2. TELEPORT, an augmented reality telepresence system

The goal of TELEPORT is to provide an enhanced communication channel between participants located in two distant sites A and B that wish to have a face-to-face meeting. TELEPORT has not the ambition to replace physical face-to-face meetings but to provide a good alternative when physical face-to-face meetings cannot be held due to various reasons including long distances, high costs and security reasons.

In each site there is a TELEPORT station (TS) which is a box having the dimensions of a 'real' office: 2.25 m (h) \times 3 m (l) \times 3 m (w). A TS serves as a metaphor for future offices designed for integrating CSCW functionality. In such offices we expect among other facilities a wall (or part of a wall) to be a display surface. There is no need for all TSs to have the same dimensions. Also they can be furnished differently and painted in different colors. One of the vertical TS's faces is a rear projection surface. A participant (b) may enter the TS A (b) from a door made in one of the vertical faces (see Fig. 1). A TS is set up in GMD Bonn in Germany. Another one is planned to be constructed at

![Fig. 1. The TELEPORT prototype application.](image-url)
the University of Geneva. A projector above the box projects video images on a mirror placed behind the projection surface. The mirror reflects the images on the projection surface. Without the use of the mirror, the projector would have to be placed behind the projection surface at a distance $2d$, where $d$ is the actual distance between the mirror and the projection surface.

What appears on the projection surface in TS $A$ ($B$) is a 3D graphics model, rendered in real-time, simulating a virtual extension of TS $A$ ($B$) (see Fig. 2). The 3D graphics model projected in box $A$ is not necessarily the same as that projected in $B$. For example, the virtual extension of the real table in box $A$ could have a rectangular shape, while the virtual extension of the real table in box $B$ could have a cyclic shape. A flat textured image of participant $b$ ($a$) is positioned in the virtual extension of TS $A$ ($B$), vertically to the $z$-axis, thus creating the illusion to participant $a$ ($b$) that he/she shares the office with participant $b$ ($a$).

The participants position (more precisely the position of their head) is tracked by means of an infrared tracking sensor. The $(x, y, z)$ position of participant $a$ ($b$) serve two purposes: First, it is used for rendering the virtual extension of box $A$ ($B$) from $a$'s ($b$'s) perspective. Second, the $z$-coordinate it is used for depth positioning the video texture image of $a$ ($b$) in ($b$'s) ($a$'s) virtual extension.

The source of the video images of participant $a$ ($b$) is the camera positioned on the table which is placed against the projection surface in TS $A$ ($B$). The position of the camera has been chosen so as to provide the best compromise of image perspective for the remote user. The result of the segmentation procedure are images whose transparency channel pixel values are set to values corresponding to either opacity or transparency depending on whether the pixel is part of the foreground, or part of the background. Typically, the background is the image the camera is capturing when no person is in the TS. The foreground is the image of the user and eventually of objects the user brings with him in the TS.

3. TELEPORT's segmentation algorithm

3.1. Hardware configuration

The real time aspects of TELEPORT require the use of specialized high performance hardware. More precisely the hardware platform for operating TELEPORT is an SGI Onyx computer with four MIPS R4400 processors running at 150 Mhz. The parallel architecture of the computer follows the shared memory paradigm: several processors
sharing the same memory. Additional equipment attached to the machine includes two Reality Engine 2 (RE2) graphic pipelines and two Sirius boards for video I/O. The input circuit of the Sirius board is delivering digitized video fields whose size is $720 \times 288$ pixels (thus the frame size is $720 \times 576$ pixels) having four 8-bit channels: one channel for transparency, and three channels for red, green and blue colors.

### 3.2. Requirements

We expect the segmentation procedure to satisfy the following requirements:

- Once the background scene is chosen, it should be possible to separate foreground from background without making any further assumptions on background or foreground object properties [4]. For example, restricting the detection of foreground objects to moving objects only, uniformly colored objects or objects having specific shapes is not suitable for TELEPORT.

- Process images in real time. Rates less than 12 images/s becomes noticeable to the user.

- Process images introducing small delay. Introducing delays greater than 250 ms becomes disturbing for the users.

- The segmented images should be of good quality. More precisely, the number of pixels that may be wrongly classified either as foreground or background should be small enough for not being noticeable to the user. This requirement has not been expressed in terms of numbers. We operated the segmentation procedure and we subjectively judged whether the result was satisfactory.

Finally, there are two important assumptions we can make which considerably reduce the complexity of the segmentation algorithm. The first is that all camera parameters, including position, orientation, zoom and focus are fixed. The second is that illumination conditions remain stable during operation.

### 3.3. A simple segmentation algorithm

The first three requirements in the previous subsection suggest a general, simple and low process-time consuming solution for the segmentation algorithm. The basic idea of the segmentation algorithm is to record a reference image $R$ containing the background scene. In fact, several images of the same scene are captured and the average image (average made between pixel values belonging to different images) becomes the reference image. Then, for each subsequently captured image $F$ the segmentation procedure calculates the absolute difference of the RGB values between pixels in $F$ and $R$. For those pixels in $F$ for which the absolute difference exceeds (is less than or equal) a given threshold $T$ are classified as foreground (background) pixels and their transparency component is assigned a value corresponding to opacity (transparency). Note that there is no need for updating the reference image over time according to the assumptions we cited in the previous section.

Concerning the last requirement expressed in Section 3.2 there are mainly three important factors deteriorating the quality of the segmentation: shadows, noise, background/foreground color blended areas and complexity of the background scene.
Shadows of users change the illumination of various parts of the scene thus making them visible as foreground although they should have been recognized as part of the background and therefore be transparent. The problem cannot be solved by increasing the value of the threshold $T$. This will cause parts of the scene to be identified as transparent (background) while being part of the foreground. We have not found a satisfactory computer-based solution to this problem. Our strategy is rather aiming in reducing the shadow problem by careful choice of furniture and painting of TSs. For example, in order to avoid shadows on the floor we have chosen a dark colored carpet. However, there are situations where it is not possible to remedy the shadow problem. For example, when a user is standing near a wall, the area on which the user's shadow falls, will be detected as foreground area. In this case we cannot expect users to paint their offices with dark colors. The shadow problem could eventually be softened with appropriate lightening. However, this is not so easy to accomplish in a TS and in real offices. The usage of bright lights may worsen the brightness perception of the image displayed on the projection surface.

The level of noise present in each image greatly affects the result of the segmentation. In our setting the main source of noise is the camera. We actually use a non-intrusive micro-camera that does not disturb the users' visual communication. We have not tried to reduce the noise level of the captured video stream. More precisely, we could apply a low pass filter on the captured images. However, this turned out to be expensive in terms of processing time even when that was performed by the graphics pipeline. Instead we provide users simple means for improving the quality of the segmentation result: the ability to inverse the transparency value of a small sized region when it is surrounded from a region with the inverse transparency value.

Another factor degrading the quality of the resulting image is the blending of foreground color and background color at the borders of the foreground regions. The cause for this color blending might be focus adjustment. Recall that the focus of the camera is fixed during operation. Therefore, foreground objects may be slightly out of focus depending on their distance with respect to the camera. This effect gives the user the impression that a narrow but noticeable region around the border of foreground regions is misidentified as foreground region while being a background region. In fact, due to color blending it is not possible to classify such pixels as foreground or background. The solution we propose is to enable the user to subtract from foreground pixels in $F$, a portion of the RGB values of pixels in $R$. Color subtraction is not a completely satisfactory solution but in many cases attenuates the impression of wrongly segmented regions. Note however that this color subtraction is applied on all foreground pixels and is not limited to foreground border pixels. Identifying border pixels was computationally expensive in our case.

The quality of the segmentation also depends on the complexity of the background scene. The more color-variation in the background, the highest the probability that background regions may be wrongly identified as foreground and vice versa. With a uniformly colored background scene, like blue studios, it is much more easier to avoid wearing clothing with the background color. Surfaces having a high reflection coefficient, like glasses and mirrors, are another problem. They change constantly the images they reflect when users move. Mobile objects which are part of the background may be
moved by the user from one place to another. In such cases, both the moving object and the area the object were placed before, will be identified as foreground objects. Finally, objects laying in the middle of the background scene, for example a chair, may hide the user when he moves behind them. There are no obvious solutions to the problems we have just described. Rather one may try to make them as rare as possible.

3.4. Need for a multiprocessor implementation

Even though the available hardware platform is considered to be a powerful one it did not allow us to meet the real-time requirements of TELEPORT with a monolithic software implementation. Indeed, the design and implementation of the various software components should attempt to optimize the use of the available hardware. The need for a multiprocessor implementation was confirmed by running a simple test program on a single processor while being the only user logged in the machine. The program consisted of a single loop computing pixel value differences between two images. Each image was stored as an array of pixel values. A pixel value consists of four 8 bit integer values. The image difference was computed on images having the size of a field and on images having the size of a frame. When segmentation is operated on images of field size the final image would be magnified by interlacing two times the same field thus obtaining segmented image having the size of a frame. For a given image size we executed the program three times. Each execution computed the absolute difference on one channel, two channels and three channels respectively. After the calculation of the difference an if-statement is assigning a value of 1 to the fourth channel when the difference was greater than a constant value otherwise the value 0 was assigned to the fourth channel. Table 1 shows the image rates at which the image difference was computed. The last row contains mean image rates of each column. Mean images rates are representatives of image difference calculations where in each iteration the absolute difference of a color channel is computed and when the difference is found to be greater than a given threshold the computation proceeds to the next iteration without computing differences of remaining color channels.

Table 1

<table>
<thead>
<tr>
<th>Number of channels</th>
<th>Image size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frame size</td>
</tr>
<tr>
<td>Three channels</td>
<td>2.56 F/s</td>
</tr>
<tr>
<td>Two channels</td>
<td>3.57 F/s</td>
</tr>
<tr>
<td>One channel</td>
<td>5.55 F/s</td>
</tr>
<tr>
<td>Mean</td>
<td>3.89 F/s</td>
</tr>
</tbody>
</table>

There are two important remarks related to the performance of the segmentation procedure. First, restricting image difference only to one or two channels may lead to poor quality segmentation results. We had, therefore, to operate the segmentation on all color channels. Second, when the segmentation is operated on fields rather than frames, the final interlacing operation is performed by the graphics pipeline. No additional
Table 2

<table>
<thead>
<tr>
<th>Number of channels</th>
<th>Image size, field size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three channels</td>
<td>2.85 f/s</td>
</tr>
<tr>
<td>Two channels</td>
<td>3.44 f/s</td>
</tr>
<tr>
<td>One channel</td>
<td>4.16 f/s</td>
</tr>
<tr>
<td>Mean</td>
<td>3.48 f/s</td>
</tr>
</tbody>
</table>

computational effort is requested from the generalized processors for operating the interlacing operation. Thus, we obtain an important saving of computational power required for computing the segmentation of fields.

Image difference alone is not sufficient for providing satisfactory segmentation results. There is the need to further process the segmented image for improving the quality of the segmentation. Further processing lowers the image rate below 3.5 f/s. Table 2 shows image rates obtained when in addition to the image difference calculation we execute two identical loops over the whole image. In each iteration two equality tests are performed between the current pixel and its south neighbor and between the current pixel and its west neighbor. The equality test compares only the fourth channel values of pixels. Note that the processing time needed for executing these two loops is much less than the processing time necessary for improving the quality of the segmented images.

4. A multiprocessor implementation of the segmentation algorithm

The segmentation module includes the following processes: master, consumer, grabber, synchronizer and one or several slave processes. All these processes communicate through shared memory.

Fig. 3 presents the architecture of the segmentation module. Ovals are used to represent processes. The big rectangle represents the shared memory. Thin arrows represent V operations on semaphores [3]. The tail of the arrow is the process calling the V operation. The head of the arrow points to the process to which the semaphore is associated. For example, the thin arrow from the consumer process to the grabber process means that consumer executes a \( V(\text{grabber}_{-}\text{sem}) \). Thick arrows show the image data transfers. Dashed arrows are used for showing the path of user commands and parameter values updates.

4.1. Shared memory layout

The shared memory includes the following parts. An area containing pointers to the various other areas of the shared memory. Each process accessing the shared memory may obtain the address of this area and then following the pointers stored at predetermined offsets access the other areas. An area large enough for storing a fixed number of images comprising the reference image \( R \), a work image useful for storing intermediate results and a minimum of three images used as an area of image buffers (AIB). The AIB
is used for storing the captured images in order to be segmented. The segmented images remain in the AIB until they are copied to texture memory. The second part of the shared memory area holds a set of parameter values shared by the various processes. A third part of the shared memory holds the various semaphores used for process synchronization.

4.2. The master process

The master process allocates the shared memory, initializes the various semaphores and creates all other processes: grabber, consumer, synchronizer and a fixed number of slave processes. Each of the master, grabber, consumer and synchronizer processes is associated with a semaphore. Each slave process is associated with a pair of semaphores: slave_sem_start and slave_sem_stop. The semaphore associated with grabber is initialized to the image capacity of AIB. All other semaphores are initialized to 0.

In addition to the above task the master process gathers user input from a GUI allowing users to adjust the various parameters of the segmentation procedure. The new parameter values are placed in the region of the shared memory reserved for that purpose. Only the master process may modify the contents of this region of the shared memory. All other processes may only read the region. The master process executes according to the following pattern:

shared memory allocation
semaphore initialization
processes creation
while (TRUE) {
    if some user input event has been detected then
    get input values and store them in shared memory
    sleep some milliseconds
}

Note that the master blocks itself when executing the sleep call for a fixed number of milliseconds. From a functional point of view we could delete the line executing the sleep call. The consequence would be the quasi-exclusive use of the processor on which master is running for testing for input events. Therefore, the number of processors for executing segmentation task will be reduced to two. Testing for input events at the rate of image segmentation, that is, approximately 12–16 times per second, is satisfactory for our purposes and the computation load is sufficiently low thus allowing the use of the processor for both user input processing and segmentation.

4.3. The grabber and consumer processes

The grabber gets a new image from the Sirius board and transfers it to AIB. The grabber stores images in the AIB one after the other until it reaches the end of the AIB whereupon it starts storing data again at the beginning of the AIB. It executes according to the following pattern:

while (TRUE) {
    P(grabber_sem)
    get an image from the Sirius board and transfer it in the queue
    V(synchronizer_sem)
}

Grabber blocks on semaphore grabber_sem when executing the P operation and the value of the semaphore is not positive. Grabber_sem is initialized to the image capacity of the AIB. In other words, grabber blocks when the AIB is full. V operations on grabber_sem are executed from consumer each time an image from the AIB is copied to texture memory. After the transfer of an image into the AIB, grabber executes a V operation on synchronizer_sem thus warning synchronizer that there are images to be segmented.

The consumer fetches images residing in AIB one after the other and interlaces two times the segmented image for obtaining an image having the size of a frame. Then the resulting image is copied to texture memory. It executes according to the following pattern:

while (TRUE) {
    P(consumer_sem)
    get next image from AIB and copy it to texture
    V(grabber_sem)
}
Consumer blocks when executing the P operation on consumer_sem which is initialized to 0. In other words, consumer blocks until there is a segmented image to be copied to texture memory. Since the segmentation is controlled by the synchronizer, V operations on consumer_sem are executed from synchronizer each time an image has been segmented.

4.4. The slave processes

Each slave process is allocated an rectangular area of an image upon which it applies, in sequence, various processing steps. Optimum use of processing power is achieved when the number of slave processes is equal to the number of available processors. When the number of slaves is less than the number of processors then some processors remain idle. When the number of slaves is greater than the number of available processors the some processors may execute several slave processes. The image subdivision in regions is user adjustable: the user may specify the desired number of columns and rows. In order to distribute as evenly as possible the processor load we subdivided the image into one column and four rows. This subdivision distributes quite evenly the processor load since the silhouette of the user will likely appear in all subdivisions. This may not be the case if the image were subdivided in one row and four columns and the user is sitting on the right or left portion of the image.

The ith slave is executes according to the following pattern:

```c
while (TRUE) {
    /* 1st synchronization point */
    P(slave_i_sem_start)
    assign transparency values to the transparency channel of an image F by computing the difference of the RGB value between F and R
    V(slave_i_sem_stop)

    /* 2nd synchronization point */
    P(slave_i_sem_start)
    eliminate foreground noise
    V(slave_i_sem_stop)

    /* 3rd synchronization point */
    P(slave_i_sem_start)
    eliminate background noise
    V(slave_i_sem_stop)

    /* 4th synchronization point */
    P(slave_i_sem_start)
    subtract background color from foreground color
    V(slave_i_sem_stop)
}
```
The execution pattern of each slave process comprises one or several synchronization points. To each synchronization point corresponds a processing step. Each processing step should be terminated before executing the processing step corresponding to the next synchronization point. The reason for forcing all slave processes to finish with step $i$ before processing step $i+1$ is that a step may be dependent on the results of previous step in regions that are processed by other slave processes.

In our final version we had only four synchronization points. The first synchronization point corresponds to the calculation of the transparency channel. The second synchronization point corresponds to the elimination of small foreground regions surrounded by background regions. The third synchronization point corresponds to the elimination of small background regions surrounded by foreground regions. The last synchronization point corresponds to the subtraction of the background color from foreground areas.

This execution pattern has been proven very convenient for testing various filtering alternatives. Adding a new processing step required only to add the following lines:

```
P(slave_i_sem_start)
new processing step
V(slave_i_sem_stop)
```

Fig. 4 shows the segmentation and the filtering of an image. More precisely, Fig. 4(a) shows an image prior to its segmentation. The background scene is the same as in Fig. 4(a) but without the person standing in the right part of the image. Fig. 4(b) shows the result of identifying foreground and background regions. Fig. 4(c) shows the result after the execution of the foreground noise elimination step. Fig. 4(d) shows the result after the background noise elimination step. Note that the bottom part of the image that has not been correctly segmented is due to the shadow of the person falling on the table.

4.5. The synchronizer process

The synchronizer executes according to the following pattern:

```
while (TRUE) {
P(synchronizer_sem)
for (1<= i <= number_of_slave_sync_points) {
    for k in set_of_slave_processes V(slave_k_sem_begin)
    for k in set_of_slave_processes P(slave_k_sem_end)
}
V(consumer_sem)
}
```

The synchronizer expects a $V$ operation on semaphore synchronizer_sem to be executed from the grabber process meaning that there is an image to be segmented. The synchronizer continues the loop iteration after the $P$ operation synchronizing all slave process to their synchronization points. This is achieved by executing a pair of
loops for each synchronization point. The first loop executes a $V$ operation on all `slave_k_sem_begin`. The consequence of this loop will be to let all slaves start the execution step corresponding to the synchronization point. The second loop executes a $P$ operation on all `slave_k_sem_end`. The consequence of this loop will be to block the synchronizer until all slaves have completed any processing corresponding to the current synchronization point. Finally, the synchronizer executes a $V$ operation on `consumer_sem` thus letting the consumer process copies the segmented image to texture memory.

4.6. Performance measurements

Our measurements were obtained running the algorithm with no users logged in the machine. The input to the algorithm was a video loop. In order to guarantee that the same image is always chosen as reference image we did the following. We marked an image $M$ in the sequence by assigning a specific RGB value on specific image
coordinates. We ensured that no other image had this RGB value at the same position. Initially, the algorithm is just capturing images without performing any segmentation. When $M$ is found, the images immediately following $M$ are chosen for computing the $R$ image. Then the algorithm performs segmenting for $D$ min and displays the various performance numbers. We have executed the algorithm on four processors. The duration of the segmentation was 1 min, the key threshold parameter value was set to 20 and all filtering parameter values were set to 10. The segmentation operated at an image rate of 8.46 f/s and a delay of 318 ms.

5. An improved implementation of the segmentation algorithm

An improved version of the previous algorithm consists of segmenting an image by operating on a reduced resolution. Also expensive memory copy operations for reducing the size of the image and then restoring the image to its original size are avoided.

To satisfy this goal we proceed as follows. Each image is divided into a number of regions equal to the number of available processors in the same way as described in the previous section. Each region is divided into blocks: a rectangular area of pixels. Inside each block a rectangular area of pixels is selected named sample area.

The segmentation of an image is processed in the following way. For each block the mean values of the RGB values of the pixels in the sample area are computed. Then all the pixels of a block are classified as background (foreground) if the absolute difference between mean values of sample areas of the current image and reference image is smaller (greater) than a given threshold. Paying the price for computing the mean RGB value of a sample area we save the computation resources needed for computing the absolute difference of all the pixels of a block.

The drawback of the above approach is that foreground objects of the segmented image have a staircase like border. Fig. 5 shows the result of segmentation based on sample areas of blocks. Black blocks are classified as background blocks. White blocks as foreground blocks. This quality of the segmented image is clearly not acceptable for the TELEPORT application. The exact border of foreground objects is reconstructed in the following way. First, blocks are identified that have a neighbor block classified

![Fig. 5. (a) Segmentation based on mean values computed from pixels in sample areas, (b) Identified border blocks for which segmentation is applied on pixel basis.](image-url)
differently. In Fig. 5 light gray (heavy gray) blocks are background (foreground) blocks having at least one foreground (background) neighbor block. Then for each border block the transparency value of its pixels is computed individually.

The new version of the $i$th slave is executing according to the following pattern:

```c
while (TRUE) {
    /* 1st synchronization point */
    P(slave_i_sem_start)
    compute mean values of sample areas
    V(slave_i_sem_stop)

    /* 2nd synchronization point */
    P(slave_i_sem_start)
    assign transparency values to the transparency channel of a image $F$ by computing the difference of the RGB value of sample areas between $P$ and $R$
    V(slave_i_sem_stop)

    /* 3rd synchronization point */
    P(slave_i_sem_start)
    identify border blocks
    V(slave_i_sem_stop)

    /* 4th synchronization point */
    P(slave_i_sem_start)
    compute transparency values for pixels in border blocks
    V(slave_i_sem_stop)

    /* 5th synchronization point */
    P(slave_i_sem_start)
    eliminate foreground noise
    V(slave_i_sem_stop)

    /* 6th synchronization point */
    P(slave_i_sem_start)
    eliminate background noise
    V(slave_i_sem_stop)

    /* 7th synchronization point */
    P(slave_i_sem_start)
    subtract background color from foreground color
    V(slave_i_sem_stop)
}
```

The choice of block size plays an important role for the performance of the segmentation procedure. When the size of the block is too small there are too many blocks for which the mean value of the sample area should be computed. When the
Table 3
Image rates obtained from executing the improved version of the segmentation procedure.

<table>
<thead>
<tr>
<th>Block size</th>
<th>Sample area size</th>
<th>Image rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>9.6 f/s (295 ms)</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>12.68 f/s (256 ms)</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>11.25 f/s (267 ms)</td>
</tr>
</tbody>
</table>

block size is big the image area covered by border blocks increases and may span over the major part of an image.

Table 3 shows image rates and within brackets delays we have obtained for various values of block sizes. We executed the algorithm with the same video loop and parameters we have described in the previous section. The dimensions of the sample area was \(2 \times 2\) pixels. When block dimensions were \(4 \times 4\) the algorithm executed the segmentation at slightly higher rates than the algorithm described in the previous section. We obtained better results with block dimensions \(16 \times 16\). Block dimensions \(8 \times 8\) gave us the best result: 35% better than the algorithm described in the previous section. Also, the performance requirements we have described in Section 3 were partially satisfied since the delay were slightly greater than 250 ms.

6. Conclusion

We have presented two multiprocessor implementations of an image segmentation procedure for real-time video streams used in a prototype telepresence system. Our presentation has not emphasized a new image segmentation technique. Rather we have concentrated on a flexible parallel implementation for a multiprocessor platform. By flexible we mean an implementation which, albeit complex due to parallelism, allowed us to easily plug in and out various alternatives without wasting precious programming effort.

Future plans for enhancing the speed of the segmentation procedure include taking into account previous segmentation results for dynamic image subdivision (instead of the current approach that subdivides the image only once at the beginning of the execution). This will eventually improve processor utilization and therefore processing speed.

References


