High Speed Skin Color Detection and Localization on a GPU

M. Seshadrinathan and K. Dempski

Accenture Technology Labs, Chicago, IL, USA

Abstract

In this paper, we present a simple yet novel method for high speed skin color detection and localization on images and video using the Graphics Processing Unit (GPU) and common graphics programming techniques. Our approach is innovative in that it provides a system for skin recognition and localization on the GPU at a speed much higher than that on the CPU. We also test in detail, the performance of different classes of cards and data buses.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications

1. Introduction

It is often desirable to perform skin detection in real-time in order to analyze a video stream at 30 frames per second (fps). There have been approaches towards creating specialized hardware to solve the problem of Real-time skin detection. In [KLACS01] a single chip digital camera with a massively parallel processor is used to achieve a speed of 30 fps. Currently, no approaches take advantage of the capabilities of the GPU. Parallelism and data independence make the GPU a much faster alternative to the CPU. GPUs have been used in many areas, such as fluid dynamics, particle systems and Computer Vision. Mapping general-purpose computation onto the GPU is not easy as there are constraints to programming the GPU that makes generalization difficult [Har05]. Our system takes advantage of the parallelism of the GPU and achieves higher than real time speeds on the card. Compared to 3 - 13 fps [ZG05] at the CPU (30 being real-time), we achieve speeds averaging 830 fps - at least 30 times faster than the CPU.

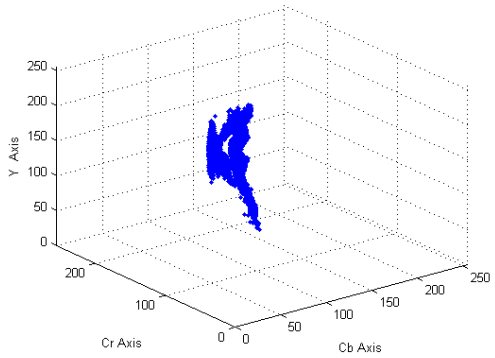

Figure 1: *Plot of Y-Cb-Cr values of skin samples from various races*

2. Skin detection

Many different approaches to the problem of skin detection such as learning networks, normalized lookup tables and parametric modeling have been explored [VSA03]. We follow the approach taken by [JR99] to build statistical color models for skin color detection since this approach lends favorably to implementation on the GPU. To build the model, we use a set of 1000000 skin color samples in the RGB color space. The Y-Cb-Cr (luminance - chromaticity) color space is commonly used in image processing. Human skin colors, though differing from person to person, are clustered together in a very small area on the Cb-Cr plane as shown in Figure 1 (different colors correspond to different races). We convert the RGB color space to the Y-Cb-Cr color space using a simple set of equations. Further, in the Y-Cb-Cr color space the luminance (Y) is decoupled from color information.

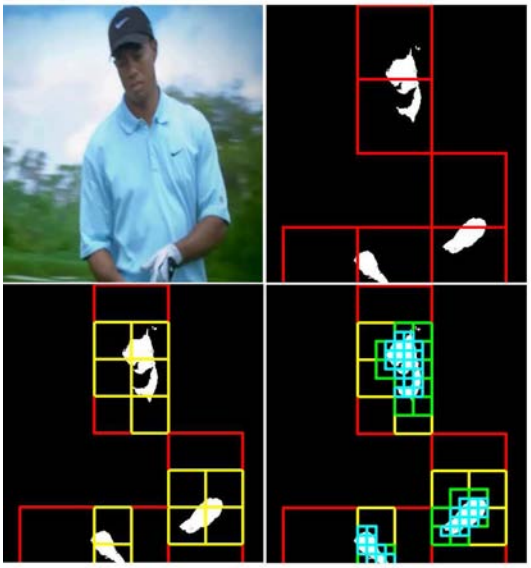

Figure 2: *Process of skin detection - From top - Initial Frame, First pass (128 x 128), second (64 x 64) and fifth (16 x 16)*

As seen from the plot, most of the spread is in the luminance. Hence, we use just the Cb-Cr color space for skin detection. We build a histogram with 240 bins per channel in the Cb-Cr color space. We convert these histogram counts into a probability distribution by dividing the count in each bin by the total number of samples. Equation 1 describes our statistical model for skin color in the Cb-Cr color space.

$$P(Cb,Cr) = C(Cb,Cr)/Tc \quad (1)$$

where, Cb,Cr are the chromaticity values, $P(Cb,Cr)$ is the probability of that chromaticity value being skin, $C(Cb,Cr)$ is the count of pixels that were indexed in that bin and Tc is the total number of samples.

3. Skin Detection on the GPU

We take advantage of several features of the GPU and graphics programming. We convert the probability model into a texture of size 240 x 240 pixels. The value of each pixel in the texture corresponds to the probability of that particular coordinate being skin color. We upload the probability texture onto the GPU as a input lookup texture. Each frame of data, on which it is desired to perform skin detection, is also uploaded onto the GPU texture memory. We then write a shader to convert the RGB value of each pixel in the input image to Cb-Cr values using a set of arithmetic instructions. After conversion, we use the (Cb,Cr) values as pixel coordinates to look up the value in the probability texture. We then assign the probability value to the alpha channel of the output pixel. The output contains all the regions detected as human skin in the incoming frame of data.

We use alpha testing to determine which pixels are rendered. Alpha tests allow us to set a comparison function that determines if a pixel is drawn or discarded. The GPU compares the alpha value of each pixel to a reference value. If the pixel does not pass the test, it is discarded. In our approach, we set the alpha test reference value to some threshold. Hence, using the alpha test, we discard all those pixels whose probability of being skin is less than a reference value.

Hardware occlusion queries make it possible for an application to query the hardware as to whether any pixels would be drawn if an object was rendered. [WB05] We use the occlusion queries to determine the number of skin pixels in the region drawn. The command to start the occlusion query is issued before the block of data is rendered. Using the alpha test we draw just those pixels whose alpha values are greater than a given reference. After the rendering is done, the command to end the occlusion query is issued. The results of the occlusion query come back with the total number of pixels drawn in that call. This corresponds to the number of skin pixels in that particular block of image data. Figure 2 the detection and the bounding boxes at different levels of localization. Figure 3 shows the flowchart of our method.

4. The Block Tree approach

If we were to process the entire image, we would be able to detect skin regions globally. However, we still would not be able to localize the skin regions. To solve this problem we use the block tree approach. We start by stepping through the image in blocks of size 128 x 128 pixels. The vertex shaders make it easier to render blocks of the input image. This increases the efficiency since the GPU is concerned with rendering just the sub segment that is being processed and not the whole image.

We used a starting block size of 128 x 128 pixels since it gave us almost the same performance when compared with a pure quad tree approach. The occlusion queries give us the number of skin pixels in that block. Each block that returns containing skin, is further divided into four blocks, each of which is further processed for skin.

To render a block of image pixels, the CPU issues a DrawPrimitve call. However, more DrawPrimitive calls translate to slower speed. It is faster to render a block of 100 x 100 pixels with a single DrawPrimitive call than issuing 100 calls to draw 10 x 10 pixel blocks. We stop at blocks of 8 x 8 pixels for localization. To increase the efficiency of our algorithm, if a block returns with no skin pixels, we do no further processing of the block. If a block returns with fully populated

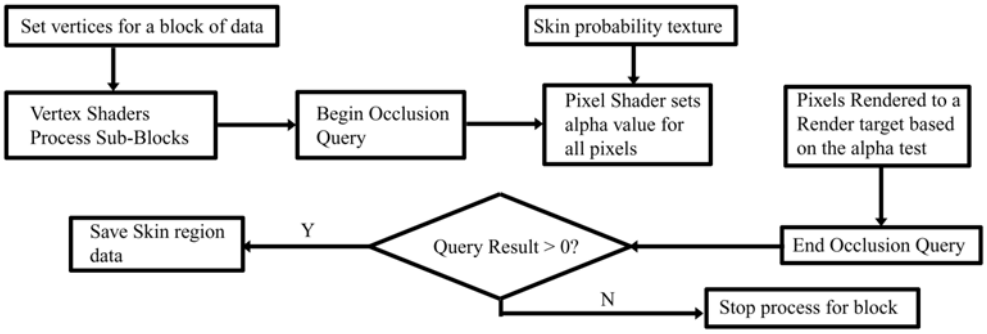

Figure 3: *The flowchart of our approach*

Card	Speed in frames per second		Bus
	Not getting data back on CPU	Getting data back on CPU	
GeForce 7800	830	495	PCI - Express
Quadro 4400	374	299	PCI - Express
Quadro 3000	339	145	AGP
CPU	22	22	N/A

Figure 4: *Performance of different cards - Average speed for 80 frames*

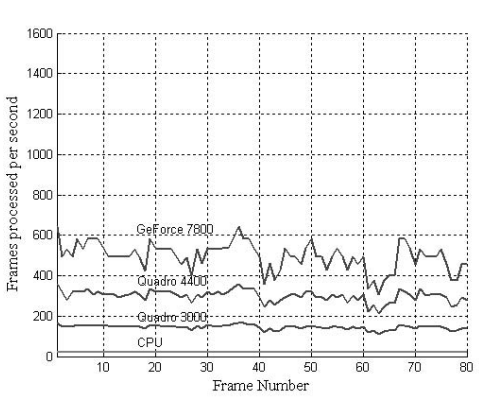

Figure 5: *Performance of different cards - Getting RT data back from CPU*

skin regions, then we index all sub blocks as skin and do no further processing.

The image generated by the GPU is written onto a Render Target. The render target resides on the GPU memory. Getting the data from the card to CPU is comparatively expensive. Since we process down to 8 x 8 blocks in the image, this gives us sufficiently fine bounding box data and we can operate at higher speeds by not getting the actual data from the card onto the CPU.

5. Experimental Results

We performed experiments on several videos and image frames to test the efficiency of our algorithm. Our experiments show that we detect and localize skin color on the GPU at an average speed of 830 fps using our block tree approach. To discuss the results, we used a sample set of 80 frames of a video of resolution 640 x 480 pixels. Since, the process of skin detection is very fast on a GPU we process each frame a hundred times in order to better time our algorithm. In order to compare our results with the CPU, we implemented an identical skin detection algorithm on the CPU. The CPU implementation of the algorithm gave us a speed of 22 frames per second.

Pulling the data off the card onto the CPU takes time and hence, the speed of detection when getting the render target data is found to be significantly lower than while not getting the data. The speed of getting the data from the card to the CPU also depends on the bus. It is seen from Figure 4 that the PCI - Express bus contributes to higher speeds of detection when getting the render target data. The graph in Figure 5 compares the detection speeds on the CPU and different GPUs when we get the render target data back on the CPU.

However, we use the block tree approach and go down to 8 x 8 blocks (0.02 percent of the image) in our experiments. Since we have such fine localization it is not necessary to get the render target data from the card onto the CPU. The graph in Figure 6 shows the detection speed when we do not

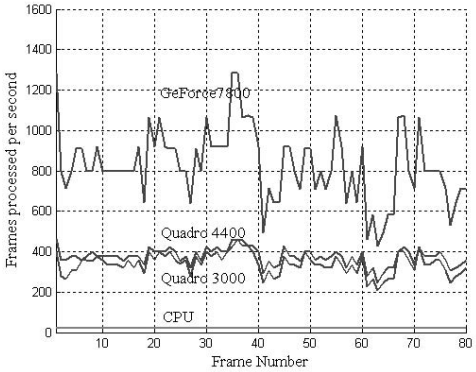

Figure 6: *Performance of different cards - Not getting the RT data*

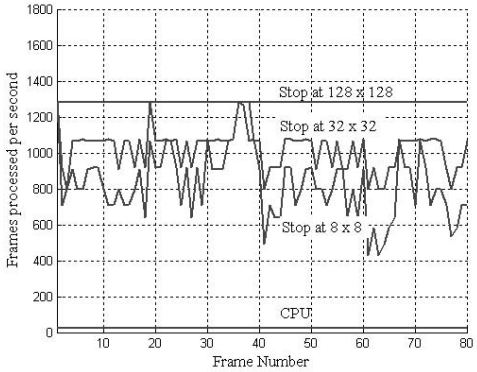

Figure 8: *Speed at various levels of localization*

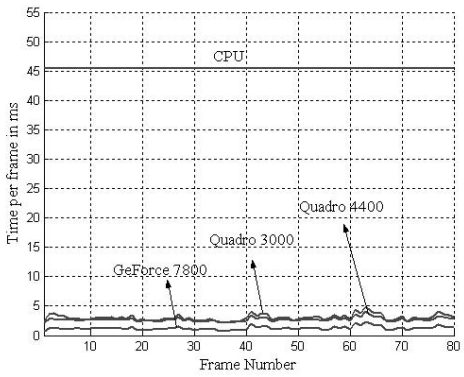

Figure 7: *Timing for different cards*

get the render target data. We see that there is a significant increase in speed.

Issuing more DrawPrimitive calls causes a reduction in speed but an increase in localization accuracy. Figure 8 shows the speed at various levels of localization (for the GeForce 7800 card). We see that the speed at 128 x 128 pixel blocks is much higher than that for localizing down at 8 x 8 pixel blocks.

From Figure 7, we see that it takes most GPUs from 1 - 5 ms for detecting skin regions as compared to 45 ms on the CPU. We also see from our experiments that the consumer end graphics cards, namely the GeForce 7800, perform better than the high-end graphics cards. This can be attributed to the fact that consumer end graphics cards are tuned for faster pixel fill rate.

We conclude that the GPU can detect and localize skin regions in an incoming video feed from a camera at speeds much higher than real time. Further, our method can be used for high speed detection and localization of any color by building a color model. We plan to integrate our method into existing face detection techniques to enable real-time face detection and recognition.

References

[Har05] HARRIS M.: *Mapping Computational Concepts to GPUs, in GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, 2005.

[JR99] JONES M. J., REHG J. M.: Statistical color models with application to skin detection. *Computer Vision and Pattern Recongition* (June 1999), 274–280.

[KLACS01] KLEIHORST R., LEE M.-S., ABBO A., COHEN-SOLAL E.: Real time skin region detection with a single-chip digital camera. *IEEE International Conference on Image Processing 3* (Oct. 2001), 306–309.

[VSA03] VEZHNEVETS V., SAZONOV V., ANDREEVA A.: A survey on pixel-based skin color detection techniques. In *Proc. Graphicon-2003* (2003), pp. 85–92.

[WB05] WIMMER M., BITTNER J.: *Hardware Occlusion Queries Made Useful, in GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, 2005.

[ZG05] ZHENG Q.-F., GAO W.: Fast adaptive skin detection in jpeg images. *Lecture Notes in Computer Science 3768* (Oct. 2005), 595–605.