Using graph cuts in GPUs for color based human skin segmentation

Lucas Lattari[*], Anselmo Montenegro, Aura Conci, Esteban Clua[a], Virginia Mota, Marcelo Bernardes Vieira[b], Gabriel Lizarraga[c]
[a]*Instituto de Computação, Universidade Federal Fluminense, llattari@ic.uff.br, anselmo@ic.uff.br , aconci@ic.uff.br, esteban@ic.uff.br, Niterói, Rio de Janeiro, Brazil.*
[b]*Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora, virginia.fernandes@ice.ufjf.br, marcelo.bernardes@ice.ufjf.br, Juiz de Fora, Minas Gerais, Brazil.*
[c] *Computer Science Department, Florida International University, gamaliz@gmail.com, Miami, USA.*

Abstract. In this paper we propose a new method to deal with the problem of automatic human skin segmentation in RGB color space model. The problem is modeled as a minimum cost graph cut problem on a graph whose vertices represent the image color characteristics. Skin and non-skin elements are assigned by evaluating label costs of vertices associated to the weight edges of the graph. A novel approach based on an energy function defined in terms of a database of skin and non-skin tones is used to define the costs of the edges of the graph. Finally, the graph cut problem is solved in Graphics Processing Units (GPU) using the Compute Unified Device Architecture (CUDA) technology yielding very promising skin segmentation results for standard resolution video sequences. Our method was evaluated under several conditions, indicating when correct or incorrect results are generated. The overall experiments have shown that this automatic method is simple, efficient, and yields very reliable results.

Keywords: graph cuts, skin segmentation, GPU computing, pixel-based classification; RGB color space, push-relabel algorithm, seeds database;

1. Introduction

Color Segmentation is an important problem in image processing [38] that presents a great number of applications as encoding [9] and identification. Human identification is mainly based in skin segmentation, which is a research field with many applications such as video surveillance, face or hand gesture recognition, content-based visual information retrieval (CBVIR), filtering on the web among others. It is a fundamental task for any application that searches for human sequences on image and video streams. In many situations, very little human intervention must occur (as when the data to be processed is represented by a huge database).

Most skin segmentation methods use a color based approach and introduce a color metric which considers the distance of the color of a pixel to a specific color [1, 3, 18, 24, 38, 42, 49, 51]. It is not a trivial problem to solve, since objects and backgrounds exist in a large variety of colors, including skin tones.

In our previous work, we presented a new automatic per-pixel human skin segmentation method, which is highly customizable and yields good results [34]. It is based on the premise that skin colors form a small and unique subset of the RGB color space, which makes it easier to solve this specific case of segmentation [49]. The segmentation problem is modeled as the minimization of a new energy function with an intuitive semantics, being automatically computed from a database of skin tones. The minimization solution is done by employing the Graph Cuts method which guarantees the robustness and good quality of the results.

[*]Corresponding author. E-mail: llattari@ic.uff.br.

However, the main flaw of our previous solution is the inefficiency of our energy function computation based on database user marked seeds when it is done sequentially. The main contribution of this paper is an implementation and adaptation of our original method in [34] for Graphics Processing Units using Compute Unified Device Architecture, improving the speed of the algorithm convergence. With a faster convergence it is possible to compute the automatic human skin segmentation efficiently not only for images, but also videos with adequate frames per second for standard video resolution.

Differently from the previous work in which we used a sequential Boykov and Kolmogorov algorithm [4] to compute the minimization of the energy function, we implemented a GPU version of another algorithm based on Vineet et al [53], called *Push-Relabel* algorithm. This also improves significantly the time efficiency of our solution.

We obtained good segmentation results for standard resolution video sequences in near 16 frames per second using a Nvidia Tesla C1060 with 240 stream processors. All issues concerning the implementation with the acceleration strategies used to increase the algorithm performance are detailed in the paper.

Another contribution of this work is a more complete study of the behavior of our method proposed in [34] when we modify some energy function parameters, as the total number of mean colors of object and background regions of images in the input database. More accurate visual results can be obtained when it is appropriately adjusted. We evaluate in our experimental results the accuracy of our solution by checking the results with user marked ground truth images. A comparison of our method with a classic skin segmentation based on RGB [32] is also performed in this paper.

The rest of the paper is organized as follows. Section 2 summarizes the main works related to ours. Section 3 describes the basis of the segmentation based on Graph Cuts, a fundamental issue in our paper. The main algorithms used to solve the maximum-flow problem associated to the Graph Cuts method are described in section 4. The parallel version of the Push-Relabel method used in our implementation is shown in the Section 5. In Section 6, we describe how the energy function is defined in terms of the database and how it is used in our model. The methodology is explained in the Section 7. Our results are presented in Section 8 and finally, in Section 9, conclusions and future works are outlined.

2. Related Works

This work is related to three main topics: models and methods for skin segmentation, segmentation methods based on graph-cuts and acceleration strategies for implementing graph-cuts in Graphics Processing Units. We summarize below some of the main contributions associated to such subjects.

2.1. Models for skin segmentation

Many methods and models have been proposed to segment human skin. Hu et al [57] obtained skin regions after a detection step based on a single Gaussian Model and a Gaussian Mixture Model, representing respectively skin and non-skin. After that, three main steps are applied to produce the final result: noise suppression, segmentation and filtering. The segmentation method used in Hu's work is based on Graph Cuts, similarly to our work. Their results are very good, even when illumination varies.

Color analysis of skin and non-skin regions is widely used. Cheddad et al [10] presented a novel color space for skin tone detection that works mostly with the luminance of the RGB image. The hypothesis of this work is that luminance inclusion does increase separability of skin and non-skin clusters. The data of this new space admits a distribution that could be fit into a Gaussian curve using Expectation Maximization. Their results are compared with three other methods and showed lower false negative rates.

Sigal et al [49] considered time-varying illumination with multiple sources and colors, using a Markov model to predict the evolution of skin histogram on HSV color space over time also obtaining good results.

Also with a clustering method, Ravichandran and Ananthi [43] obtained skin segmentation using *K*-means. The idea is to convert the RGB image into a Lab color space and then find *K* color regions. The skin detection is made using simple boolean decision rules for each *K* region. The results presented in [57] seem to be more robust in terms of segmentation quality. However, the Ravichandran and Ananthi's work proposes a more efficient approach in terms of the computational cost.

Jedynak et al [23] compared three approaches for skin detection. Each one relies on a maximum entropy model based on constraints. The first is a baseline model in which pixels are considered independent and measured by a Receiver Operating Characteristic curve. The second is a Markov Random Field that

forces smoothness in the solution. The final model used is based in color gradient. The MRF and the color gradient approaches were better, resulting in approximately 84% true positive detection rate.

2.2. Energy minimization via Graph-cuts

One of the first works solving global minimization energies for computer vision was the one proposed by Greig et al [17] for image restoration. It was the first work to discover that powerful graph-based algorithms for combinatorial optimization can be used to minimize certain important energies in computer vision. They created an image-based two terminal *s*-*t* graph whose construction is based on the Gibbs Energy [3, 4, 6, 15, 17, 21, 22, 28, 31, 33, 35, 37, 41, 44, 45, 46, 53, 54] such that the minimum cost graph cut gives the optimal binary labeling solution. Previously, such energies could not be solved by exact minimization methods. One alternative was to use metaheuristics like Simulated Annealing [27].

The Graph Cuts concept was preceded by a number of graph-based methods for image clustering that used combinatorial optimization algorithms or approximate spectral analysis techniques, like normalized cuts [48]. These works are mentioned in Boykov and Funka-Lea [2]. The goal of some of these approaches is to produce a completely automatic high-level grouping of image pixels. This means that they divide an image into “blobs” or “clusters” using only generic cues of coherence or a measure of affinity between pixels. Differently, Graph Cuts integrate more appropriately model-specific visual cues and contextual information in order to define more accurately particular objects of interest. This is also related to other categories of segmentation methods like Snakes [26], Active Contours [55], Intelligent Scissors [20] and Level-Sets [47].

Researchers have been creating or comparing approaches for the minimization step. Well known examples are Snakes [26], Intelligent Scissors [55] and Level-Sets [47]. Techniques like Gradient Descent [8, 56] can be applied to any energy function of continuous variables and others like Simulated Annealing can be used in any function of discrete variables. However, these generalities can imply in very poor results since they get stuck in the local minima or take an extremely long time to converge.

Boykov and Jolly [3] developed a technique to solve binary image segmentation by minimizing the Gibbs Energy using Graph Cuts. It was the first efficient work for *N*-dimensional applications, comprising *region* and *boundary* properties of elements. It is a general-purpose segmentation, needing initial user mark cues to characterize which elements probably belong to an object or a background set. Their solutions allows new additional user marks even after the initial segmentation, without recalculating them from scratch. It was tested with 2D images and 3D volumes giving good and stable results even when initial seeds were changed after the final result, performing with good speed. They motivated other works to approach different problems with similar methods.

However, one of the main problems of the Graph Cuts approach is the computational complexity of its execution. Some of these minimization algorithms are very expensive for instances with a large number of pixels. Many works have studied the practical efficiency of it in Computer Vision and proposed major improvements [4].

Boykov and Kolmogorov [4] provided an experimental comparison between different Graph Cuts algorithms for computer vision applications. They analyzed the complexity of methods based on Goldberg and Tarjan [16, 19], Ford and Fulkerson [13, 14] and a new method created by them. Their method was significantly faster than the previous ones. It is based on the original Ford and Fulkerson algorithm but it builds two simultaneous search trees for terminal nodes, reusing data at each iteration. Their implementation is the current sequential state-of-art method for Computer Vision implementations of Graph Cuts.

Other works have proposed improvements on the cost function task of the energy minimization framework. Li et al [35] developed a method for interactive image cutout, focused on user usability without loss of performance. Their method consists of two steps: an object marking task, like presented in [3] and a pre-segmentation computation, followed by a simple boundary editing process, creating the output segmentation. The nodes from the graph are not single pixels, but similar color regions generated by a Watershed algorithm [52] and becoming *superpixels*. Their experiments yielded remarkable results in usability case studies, where users took overall less than 60% of the time using their software than traditional Magnetic Lasso feature present at common image processing software. The energy function used in our work is related to the one they proposed.

2.3. Graph-cuts in GPUs

Some works attempted to improve the efficiency of the Graph Cuts method proposing versions running in parallel processors. Our work belongs to this category, running on Graphics Processing Units.

Many parallel approaches were made using GPU Computing, after they became more popular and easy to use, mainly because of the development of programming libraries like CUDA [40]. Examples of works in this area are the ones proposed by Vinnet and Narayanan [53], Hussein et al [20], Garret and Saito [15] and Yildiz and Akgul [56].

Hussein et al [20] proposed the first implementation of the Graph Cuts algorithm in CUDA for general graphs. It is a GPU version of the Push-Relabel algorithm, making two important changes: the only labeling scheme is the Global Relabeling heuristic [12] and sending flow from a node is an operation divided into two phases: *Push* and *Pull*. The first only sends flow, storing an amount on a temporary memory and the *Pull* updates all entire flow pushed before. This is necessary to avoid *Read-After-Write* hazards. They also introduced two optimizations. The first one is a lockstep Breadth-First Search which performs Prefix Sum Operations when traversing each depth level which is necessary for general graphs. It can be considered a lockstep operation because only a single direction is traversed at a time, while the others are blocked. The second optimization is basically the emulation of a cache. Instead of loading the data of a single node in the global memory, it loads a 2D tile which is added into a lattice, a data structure created during the Breadth-First Search stage containing only visited nodes. That strategy enables coalesced memory access, improving the algorithm speed. The speedup obtained is in the range of 1.7-4.5 over the CPU version proposed in [4].

Vinnet and Narayanan [53] proposed an implementation of Graph Cuts in CUDA for binary segmentation using the Push-Relabel algorithm. They presented two versions of the same algorithm, using atomic and non-atomic operations. They also proposed a new heuristic called *Stochastic Cut*. This heuristic relies on the fact that after a few iterations, the processing done on the majority of the nodes is finished. If at a given iteration, an entire thread block does not modify the residual graph by pushing flow, then the block is considered inactive and delayed for 10 iterations. Their implementation is very similar to the one described in [20], including the use of GPU shared memory and *Push-Pull* operations. One of the characteristics of their work is that it does not use prefix sum operations. Finally, they defined a simple dynamic Graph Cuts implementation for video segmentation, always reusing solutions from the previous frames. They achieved a level of performance 10-12 times faster than [4] and approximately 3 times faster than [20] for images. However, their approach is only applicable for grid graphs and not for general maximum flow calculations.

Some GPU-based works do not use the Graph Cuts method directly. Yildiz and Akgul [56] formulated the Graph Cuts optimization as a gradient descent solution on the GPU. Working differently from the previous Maximum Flow approaches, this solution is given by the Minimum Cut energy function formulation, solving the labeling problem directly without graph processing. It is based in Linear Programming and decreases spatial complexity. It is modeled by a Lagrange dual model and a modified approximate objective function which is differentiable at any point. Their method needs less memory than the standard Maximum Flow methods, but gives some small errors at smooth regions because the used function is an approximation. For some examples, their method converges much faster than [4].

3. Segmentation by Graph Cuts

The basic foundations of our solution are featured in this section.

3.1. Energy Function

It is possible to find a characteristic function of an object defined in a given domain by minimizing an objective function, i.e., given a set *V*, we have to find the characteristic function *X* which is the minimum argument of a function [3] and the partition sets. A widely used objective function in image segmentation is the Gibbs Energy [3, 17, 35] defined as

$$E(X) = \sum_{x_i \in V} E_1\left(X(x_i)\right) + \lambda \sum_{x_i, x_j \in \varepsilon} E_2\left(X(x_i), X(x_j)\right), \quad (1)$$

where x_i and x_j are elements of the set to be segmented, V is the set of elements, ε is the set of connected elements and λ is a weight. E_1 is the term that defines the cost for each x_i to belong to one of the sets. Aiming to minimize the objective function,

this cost should be inversely proportional to the probability of x_i belonging to the set. It can be given as

$$\begin{array}{lll} E_1(X(x_i)=1)=0 & E_1(X(x_i)=0)=\infty & \forall x_i \in O \\ E_1(X(x_i)=1)=\infty & E_1(X(x_i)=0)=0 & \forall x_i \in B \\ E_1(X(x_i)=1)=\phi(\rho_o) & E_1(X(x_i)=0)=\phi(\rho_b) & \forall x_i \in N \end{array} \quad (2)$$

where *O* is the set of object elements, *B* is the set of the background elements, *N* is the set of pixels whose labels are unknown and ϕ is a function inversely proportional to its parameters terms ρ_o and ρ_b.

E_2 is a term that defines a penalty for labeling two connected elements with different labels. This penalty depends on the similarity of both elements: very similar elements have high probability of belonging to the same set. In this case the resulting cost must be high; otherwise it has a small value.

The minimization of some classes of energy functions can be considered a NP-Hard problem, needing special methods to efficiently solve it. In our work, we will minimize the Gibbs Energy using the Graph Cuts method. The next section describes how energy functions can be minimized in the context of Graph Cuts theory. For more details, see Boykov et al [6] and Kolmogorov and Zabih [31].

3.2. Graph-cuts

The energy function in (2) is used to define the costs $w \geq 0$ for each edge $(u,v) \in E$ on a directed graph $G=(V,E)$ [4]. Two terminal nodes are defined: a source *s* and a sink *t*, corresponding to the labels that can be assigned to pixels. Each non-terminal node in the graph will have an edge connected to *s* and another edge to *t*. We assume that every vertex lies on some path from *s* to *t*.

Two types of edges are then defined: *T-links* and *N-links*. *T-links* connects pixels to terminals. *N-links* connects pairs of neighbor pixels. Their costs are based in the terms E_1 and E_2, respectively.

The minimum cut of a graph is the set of edges that removed from *G* creates two disjoint subsets: a set *S* and a set T. These sets are subgraphs disposed so that $s \in S$ and $t \in T$ [12]. An important theorem due to Ford and Fulkerson [14] states that the solution of the minimum cut of a graph is equivalent to the maximum flow problem. Considering this, it is used maximum flow algorithms to solve the minimum cut problem.

In the next section, we summarize the main algorithms used to solve the minimum cut/maximum flow problem.

4. Minimum Cut / Maximum Flow Algorithms

Two most known classes of algorithms were proposed to solve the maximum flow problem for Graph Cuts minimization. One category is based on the Ford and Fulkerson original idea [14] which enforces flow conservation during the whole process. Another one formulated by Goldberg and Tarjan [16] breaks the flow conservation rule until convergence. Examples of these will be described below.

4.1. Algorithms based on augmenting paths

The classical maximum flow algorithm based on the notion of augmenting paths was proposed by Ford and Fulkerson [14].

Boykov and Kolmogorov [4] proposed a new method to solve the maximum flow associated to Computer Vision problems using an approach based on the Ford and Fulkerson algorithm. It uses two search trees on the residual graph, one with its root at the source *s* and another starting at the sink *t*. Each tree grows from its own terminal node.

When the two trees touch each other, an augmenting path is found. Flow is sent in this path as much as possible. After that, the residual graph is updated and new paths are searched, reusing the trees. The algorithm is finished when no other path can be found.

This algorithm outperforms Ford-Fulkerson when grid-based graphs are used. This occurs because the search trees are reused during algorithm iteration. However, the critical point of this algorithm lies on the management of the trees, for example, in growing or updating it to obtain a short augmenting path. It does not guarantee to find the shortest path as it uses some heuristics to search for the shortest one. The use of such heuristics produces good quality paths while keeping the efficiency of the overall process. In fact, the method has a compromise between choosing any path or the shortest one. The theoretical time bound complexity of this algorithm is $O(VE|C|)$, but in practice it is almost linear.

Boykov-Kolmogorov is considered the best method to compute Graph Cuts in Computer Vision for

sequential machines. However, new parallel approaches using Push-Relabel approaches have given best times over it.

4.2. Algorithms based on Preflow

The Push-Relabel algorithm works in a more localized manner than the augmenting-paths methods making it a strong candidate for parallelization. Instead of examining the entire residual network $G_f = (V,E)$ to find an augmenting path, generic Push-Relabel algorithms work on one vertex at a time, analyzing and operating at its neighbors in the residual network. Furthermore, unlike the augmenting-paths based methods, it does not maintain the flow conservation property throughout the execution.

The basic intuition of this method is very different from those based on augmenting paths. Each node has two additional properties, defined as *excess flow* and *height label*. All nodes start with a height and an excess flow equal to zero, except the height of the source s that is fixed at |V| and the excess of *s* which is infinite.

Consider the nodes $\{u,v\} \in G_f$. Two operations are also defined: the one that pushes excess flow of a node *u* into a neighbor $v \in N_u$ is called *Push*. Eventually, the algorithm will try to push the excess flow of *u* into its neighbors, but none of these nodes $v \in N_u$ has height label below the height of *u*. To rid an overflowing vertex *u* of its excess flow, it is necessary to increase its height. Such operation is called *Local Relabel*. The height of *u* is increased by one unit above the height of the lowest neighbor that has an unsaturated edge connecting them. After the local operation, the excess of *u* can be pushed. When all paths to *t* are saturated, the algorithm has to send the remaining excess flow in the system back to the source *s*, by continuously increasing height of the vertices with excess flow until they achieve |V|. After this point, the *preflow* becomes a legal flow and also a maximum flow.

The *Local Relabel* and the *Push* are the operations done in the basic Push-Relabel algorithm, but the procedure as described above has poor practical performance [12]. Much unnecessary processing is done until convergence, because the heights are only updated locally, not considering the global picture of the distances. However, heuristics can be employed to discharge the excess nodes to the sink *s* faster, once the paths to *t* are saturated. Here we describe two heuristics: the *Global Relabeling* and the *Gap Relabeling*. These heuristics check the entire residual graph and correct heights globally.

The *Global Relabeling* operation updates the distance function defined on the residual graph by computing shortest path distances in the residual graphs from all nodes to the sink. This can be done in linear time by using a Backwards Breadth-First Search starting at the sink *t* node, adjusting exactly all heights.

The *Gap Relabeling* tries to find disconnected nodes from *t* in the graph *G*. It is based on the following statement. Suppose that $g \in N$ and $0 < g < |V|$. At a certain stage of the algorithm there may be no nodes $n \in V$ with distance $h(n) = g$, but there are nodes *u* with $g < h(u) < |V|$, a situation defined as a *gap*. These nodes *u* are converging to *s*, and the sink *t* is no more reachable from any of these vertices *u*. Therefore, the label of such nodes may be increased to $|V| + 1$ directly.

The addition of the *Gap Relabeling* heuristic to the Push-Relabel outperforms the practical efficiency of the pure Push-Relabel method, although usually not as much as by adding the Global Relabeling heuristic. These heuristics are not independent, considering that the Global Relabeling discovers nodes disconnected from *t* and makes gaps less likely. However, the Gap Relabeling iteration has small overhead compared to the Global Relabeling. Thus even if no gaps are discovered in a run of an implementation that uses both heuristics, the running time is almost the same as in the implementation that uses only Global Relabeling. In some graphs instances, many gaps are found and the former implementation is faster than the latter.

The Push-Relabel algorithm is one of the most efficient algorithms to compute a maximum flow. The general algorithm has $O(V^2E)$ theoretical time complexity. As evaluated by [4], the Push-Relabel method behaves more efficiently for dense graphs, differently from augmenting-path based approaches that converges rapidly for sparse graphs.

Considering only sequential architectures, it is more adequate to use Boykov-Kolmogorov algorithm for grid-based sparse graphs and Push-Relabel approach for grid-based dense graphs, like based in 3D data. However, the computational power of manycore architectures like GPUs makes appropriate the use of Push-Relabel approaches even for sparse graphs with

very competitive results. This is detailed in the next section.

5. Graph Cuts in GPU

In this section we describe in details our framework that implements the Push-Relabel algorithm in the GPU. First we explain the idiosyncrasies of GPUs that we had to consider in our implementation. In the sequel, we describe how the graph is constructed, how the algorithms are implemented and which heuristics were applied to accelerate the overall performance. Our implementation is loosely based in the [15, 20, 53] works and was developed with the CUDA architecture.

5.1. GPU Computing

Graphics Processing Units were initially developed as devices dedicated to graphics processing, improving the efficiency and the power of the graphic pipeline. With the advent of the new GPU models after the GeForce series 8 and the architectures like CUDA, the implementation of GPU computing applications became easier to be done.

This architecture consists on a unified arrangement of cores, which simplifies the GPU programming model by treating it as a typical manycore processor. Further it improves the GPU model by removing memory restrictions or graphical idiosyncrasies for each processor. Data representation is also improved by providing friendly data structures to the programmer. The memories available in the CUDA device can be accessed by all processors with no restriction on its representation, though the access times may vary depending on the memory type used.

The CUDA environment is based on the SIMD parallel architecture, where program kernels process data grids, dividing multiple blocks in threads. It is important to obtain maximum performance by executing the same operation simultaneously on different data elements, avoiding code flow divergence. Divergent code produces poor performance because the CUDA model cannot deal efficiently with different instruction flows at a given moment. Another important fact is the lack of GPU memory lock. This brings restrictions on how threads can modify shared memory space.

Each thread can use a number of private registers for its computation. A collection of threads is called a block and runs on the same multiprocessor at a given time. The threads of each block have access to a small amount of common shared memory. Synchronization barriers are also available for all threads of a block. A group of blocks can be assigned to a single multiprocessor but their execution is time-shared. The available shared memory and registers are split equally among all blocks that timeshare a multiprocessor. Multiple groups of blocks are also time shared on the multiprocessor for execution. The collection of all blocks in a single execution is called a grid.

Each thread executes a single instruction set called kernel. For each thread and block is given a unique ID that can be accessed within the thread during its execution. An algorithm may use multiple kernels, which share data through the global memory and synchronize their execution at the end of each kernel. Threads from multiple blocks can only synchronize at the end of the kernel execution by all threads.

5.2. Maximum Flow/Minimum Cut in GPUs

The first challenge to implement Graph Cuts in the GPU is to devise a way to implement the graph neighborhood. Considering that graphs based on images naturally have a grid structure, we can define a specific model to store their neighborhood structure in the GPU. This is important not only to simplify the algorithm, but also to reduce the total use of global memory and the number of accesses, improving our solution.

It is recommendable to reduce the use of global memory as much as possible. Creating auxiliary data structures to represent neighborhood, like adjacency lists or adjacency matrices can impose excessive use of memory and increase the number of accesses and the processing times.

The Graph Cuts model is well suited to the GPU architectures, because each thread can operate on exactly one pixel in a SIMD fashion. Images have a grid format, making the mapping to this architecture very simple. Each vertex can have a 4 or 8-Neighborhood System connectivity. Two schemes are usually employed in GPUs for neighborhood representation: adjacency lists and grid structures. In this work we use only grid structures.

The data associated to the vertices and edges is represented by arrays, where each array index *i* stores the data referent to a single node *i*. This data structure

is appropriate to the SIMD model, which may be treated as an array of processors. Memory coalescence and data manipulation efficiency is improved considering that all threads will access and handle contiguous data. The array size of the non-terminal vertices and the edges' data is always |V|.

In order to represent the vertices, two arrays are necessary: one to store the excess flow and another to represent the height labels that estimate the distances to the target node. To represent edges, six arrays storing residual capacities are sufficient, two for source and sink capacities, and four to the north, south, west and east directions. Even nodes in the border will have a total of six edges represented in the arrays. However, the edges that do not exist for the border nodes will have a residual capacity equal to zero. This is done to guarantee memory coalescence.

The same format can be extended to represent non-grid graphs for SIMD architectures. This is done by modifying the original graph to become regular, inserting null edges until all nodes have the same number of neighbors as the node with the highest degree. This is necessary to represent the graph in a more friendly SIMD format [15], being able to perform the same operation across multiple data locations. These null edges inserted in the original system do not modify the graph solution, because their residual capacities will always be zero.

The implementation described in this section is based in the ideas presented in [15] and [53] with a few modifications. The size of the CUDA grid is equal to the image dimensions, where each single thread is mapped by a pixel or a graph vertex.

Each node has the following data: excess flow and height, the same attributes defined in the original sequential implementation. These are stored as appropriate-sized arrays in the GPU global memory, becoming accessible to all threads. Auxiliary memories are also used, principally shared memory. It is a simple tiny memory available at each multiprocessor. Each multiprocessor handles a thread block with 512 threads. This value can vary with the specific hardware.

Two kernels are implemented: *Parallel Push* and *Parallel Local Relabel*. These two kernels do the same as the Push and Relabel operations of the sequential Push-Relabel algorithm. Respectively, one updates the excess and pushes flow to its neighbors and the other applies a local relabeling operation to adjust height labels. For parallel correctness, the Parallel Push stage (Figure 1) is divided into two phases, defined as *Push* and *Pull*. This is necessary because of the potential Read-After-Write hazard. The Parallel Push stage is implemented in a kernel in which each node sends flow to its neighbors but only modifies the edges residual capacities and not the excess of the neighbor nodes. The updated flow is stored temporarily in an auxiliary array. The *Parallel Pull* stage (Figure 2) is executed in another kernel, usually implemented together with the relabel operation.

Parallel Push Operation Kernel on a node u

Load $h(u)$ and $e(u)$ from the global memory to the shared memory of the block.
Synchronize threads to ensure load completion.
Push flow from $e(u)$ to the neighbors $v \in N_u$ that satisfies height properties and without violating $c_f(u,v)$ capacities.
Update the residual capacities of edges (u,v).
Update the excess flow $e(u)$.
Store the flow pushed to each edge in a temporary global array F.

Fig. 1. Parallel Push operation in pseudocode.

Parallel PullRelabel Operation Kernel on a node u

Load $h(u)$ and $e(u)$ from the global memory to the shared memory of the block.
Synchronize threads to ensure load completion.
Update excess flow $e(u)$ of each vertex and the residual capacities $c_f(u, v)$ with the flow from global array F.
Synchronize threads to ensure completion load.
Compute the minimum height of N_u.
Write the new height to global memory $h(u)$.

Fig. 2. Parallel Pull and Relabel operations in pseudocode.

The *Parallel Pull* function updates the excess of the Parallel Push phase from the temporary array data. The Push and Pull stages implemented in two different kernels are unnecessary when Atomic Operations are employed, because this prevents the occurrence of the RAW inconsistency in the hardware level.

The first basic necessary optimization adds null edges in the image boundary nodes, similar to the idea in [15]. This is necessary because boundary nodes have fewer neighbors than central nodes. Moreover, the CUDA Model is SIMD, restricting single instructions execution in a massive data grid. It is important to execute the same operations in each thread in a block, avoiding divergence. Divergence results in serialization of the instructions and a reduction in performance. It is appropriate for the Push and Relabel kernels to check six edges in each node, even

if the boundary nodes do not contain six neighbors. Obviously, the null edges will make no difference in the final result.

A *Parallel Global Relabel* kernel based on the original work from [12], [14] and [53] is also necessary to accelerate the algorithm (Figure 3). It is basically a Breadth-First Search starting from the sink *t* node and the only nodes that are visited are those with unsaturated neighbor edges adjacent to *t* and their neighbors. The first iteration checks which nodes have unsaturated sink edges. Such nodes are added to a list of nodes to be visited.

For each iteration, all nodes in this list have their heights updated based on the distance from *t*. Then, they are removed from the list and their neighbors are added. Each node is also marked as a visited node in another list. Only nodes never visited and added as a new node to be visited have their heights updated. This heuristic helps on quickly Push-Relabel convergence, but every execution of it is very slow. It is important to execute it only a few times during the execution.

Global Relabeling kernel starting on a node t on a graph G_f

Start a frontier array F_a, a temporary array F_{ta} and a visited nodes array V_a.
$F_a[t] \leftarrow$ true.
$LevelBFS \leftarrow 0$.
while $\ni F_a[x] = true$, *such that* $x \in V$ do
  for each $u \in V$ in parallel do
    if $F_a[u]$ = true and $V_a[u]$ = false then
      $V_a[u] \leftarrow$ true.
      $F_a[u] \leftarrow$ false.
      for each $v \in N_u$ do
        if $c(v,u) > 0$ then
          $H(v) \leftarrow LevelBFS + 1$.
          $F_{ta}[v] \leftarrow$ true.
        end if
      end for
    end if
  end for
  $LevelBFS \leftarrow LevelBFS + 1$.
  $F_a \leftarrow F_{ta}$.
end while

Fig. 3. Global Relabeling operation in pseudocode.

Another heuristic implemented in our GPU Push-Relabel is a parallel version of the Gap Relabel (Figure 4), based on the original heuristic from [16]. In this heuristic, we change in parallel the nodes height to the maximum limit if they are disconnected from the residual graph. We maintain a binary list with the size of all possible heights that the nodes could assume, storing binary values if there is at least one node with that height. All positions of this list are checked simultaneously by a Gap Relabel kernel and if there is at least one position with 0 data, but the next position is 1, a gap is found and this index is the gap height. Another kernel is executed, where all nodes with height greater than this gap are changed to the maximum instantly. This heuristic improves the convergence of the method. The main advantage of this compared to the Global Relabel is the small overhead and speed of a single iteration.

Gap Relabeling kernel operation on a graph Gf

Start a binary gap array G_a with false and a gap variable with zero.
Check all $h(u), \forall u \in V$. If exists at least one node $h(u) = n$, then $G_a[n]$ = true.
Synchronize all threads.
Find the minimum n such as $G_a[n]$ = true and $G_a[n+1]$ = false.
$gap \leftarrow n$.
if $gap > 0$ then
  for each $u \in G_f$ in parallel do
    If $h(v) > gap$ then
      $h(v) \leftarrow h(s) + 1$.
    end if
  end for
end if

Fig. 4. Gap Relabeling operation in pseudocode.

6. Human Skin Database

In this section we present our color database which is crucial to obtain the automatic seeds necessary to define the weights of the edges in *G*.

In order to define an energy function for skin detection, we propose the use of *n* images where skin regions and non-skin regions are explicitly marked by some user. The idea is to use these multiple images to infer the likelihood of a pixel color to represent a skin or non-skin region.

In each image I_i, $0 < i \leq n$, of the database, n_o pixels are marked as skin, forming an object region $O \subset I_i$, and n_b pixels are marked as background, forming a region $B \subset I_i$. Note that $O \cup B$ is not necessarily a partition of I_i. This

means that the user does not have to label all pixels of the image.

The database is then defined by *n* sets, each one composed by k_o colors of O_i and k_b colors of B_i. These k_o and k_b colors are colors computed using a quantization method capable of finding the best colors that represents each gamut subregion of each image in database. Each image will be represented by mean colors that try to represent the entire user marked region. In our work, we use the K-means method [36] for simplicity.

7. Proposed Method

Now we describe how our color database is used to define the energy terms in our problem.

Let $CO_{m,i}$ be a *m*-th object color defined by the quantization method of the *i*-th image of the database and $CB_{l,i}$ be the *l*-th background color of the same *i*-th quantized image of the database, where $0 < m \leq k_o$, $0 < l \leq k_b$ and $0 < i \leq n$. It is possible to define the similarities by the following functions

$$\begin{aligned} d_{o,i} &= \min_m \left\| C_p - CO_{m,i} \right\| \\ d_{b,i} &= \min_l \left\| C_p - CB_{l,i} \right\| \end{aligned} \tag{3}$$

where C_p is the color of the pixel *p* that we need to label as skin or not, $d_{o,i}$ and $d_{b,i}$ are the minimum distance between C_p and the colors of the *i*-th object set and the colors of the *i*-th background set in the database, respectively.

Similar to [35], our energy terms for (2) are then defined as

$$\phi(\rho_o) = E_1\left(X(p)=1\right) = \frac{\sum_{i=1}^{n} \frac{d_{o,i}}{d_{o,i}+d_{b,i}}}{n} \tag{4}$$

$$\phi(\rho_b) = E_1\left(X(p)=0\right) = \frac{\sum_{i=1}^{n} \frac{d_{b,i}}{d_{o,i}+d_{b,i}}}{n} \tag{5}$$

$$E_2\left(X(p), X(q)\right) = \frac{\left|X(p) - X(q)\right|}{\left\|C_p - C_q\right\|^2 + 1} \tag{6}$$

$$\lambda = 1,$$

where C_p and C_q are the colors of the neighbor pixels *p* and *q*. The term $\left|X(p) - X(q)\right|$ makes $E_2\left(X(p), X(q)\right) = 0$ whenever *p* and *q* belong to the same set.

The energy terms are used for defining costs for each edge in the graph. E_1 defines costs for a pixel belonging to a foreground (4) or background (5) region and E_2 set penalties when *p* and *q* nodes are assigned with different labels (6). The $\lambda = 1$ term is an empirical value that associates a penalty to the boundary term E_2. Increasing this constant, we penalize adjacent pixels grouped in different sets, raising the possibility of neighbor tones being represented in the same set.

It is not necessary to mark pixels as object or background in the image to be segmented. The energy terms were conceived to gather skin and non-skin information only from the database images. Thus, the user presents an image which is segmented automatically by the method based on the database colors.

8. Experimental Results

Our tests were evaluated considering time computation and visual quality of the output segmentation. They will be presented separately in this section.

8.1. Visual quality

The tests featured in this section were obtained with different k_o and k_b mean colors parameters. On the database, 20 images are user marked with skin and non-skin regions. Some examples of user marks are presented in Figure 5. Some images of our data-

base belong to the Berkeley Image Segmentation Database [39]. Naturally, when more images and consequently skin and non-skin colors are stored, more samples of different tones are considered, improving the average energy and leading to better results. However, with the increase of the database, the time computation of the energy terms increases significantly.

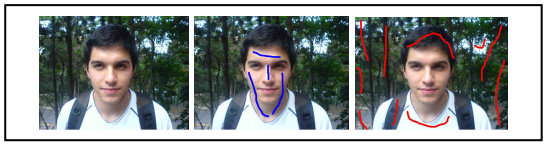

Fig. 5. Examples of user marks on the database stored images.

Our method was applied to images of people belonging to different ethnic groups. The results are consistent because the possible skin colors forms a small subset of the RGB space. Consequently, a skin pixel tends to have small distances for all images in the database, minimizing the cost in Equation (4).

The segmentation performs better when the database has an adequate number of images with heterogeneous color tones. When more background pixels are marked, fewer potential errors can occur in the segmentation of an image. This is shown in Figure 6.

In our tests, the object color parameter is k_o = 32. In other words, each user marked image in the database has 32 mean colors computed by a quantization method that belong to the set *O*. It is not necessary to increase this value because the skin color set does not vary significantly along with the database samples. However, the entire RGB color space can be stored in the background set, including even the skin color set. Consequently, it is recommendable to use a higher background color parameter k_b . Many tests were evaluated with k_b = 64, k_b = 128 and k_b = 256.

Figure 6 shows an image with examples of their automatic human skin segmentation. Figure 6(a) is the original image of a soldier, where cloth tones of background can be confused with human skin colors. In Figure 6(b) the parameter k_b = 64 was insufficient to represent many skin-like background regions. Increasing this value to k_b = 128 reduces the errors, as depicted in Figure 6(c), because more background tones similar to cloth are considered. Finally, with k_b = 256 in Figure 6(d) the errors are reduced.

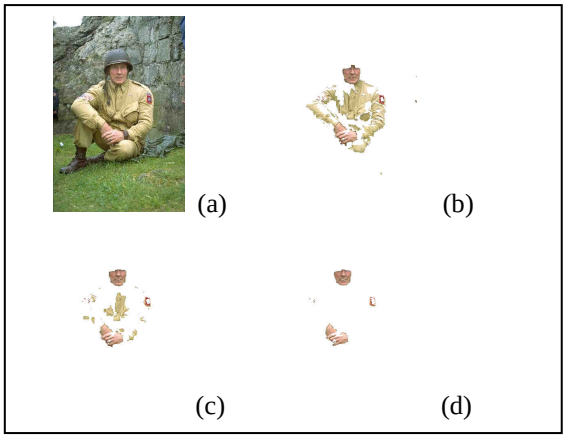

Fig. 6. Example of segmentation results with skin-like tones in the background. Figure 6(a) is the original image. Figure 6(b) is a segmentation with k_b = 64, Figure 6(c) has k_b = 128 and Figure 6(d) has k_b = 256.

The method is dependent on the database images. It is important to mark well the different background tones as much as possible, to reduce segmentation mismatch. The distance metric is appropriate to reduce errors, because if a background color of the segmented image is not well represented in the database background set, then probably its Euclidean distance to another database background color will be small. However, errors may occur when a background skin-like tone on the segmented image is not well represented in the database. A similar example of skin-like background region is shown in the flowers of Figure 7.

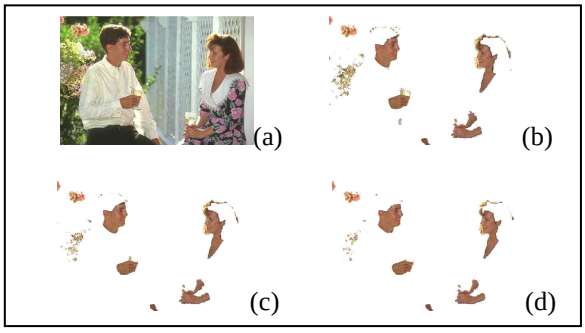

Fig. 7. Other examples of segmentation results with skin-like tones in the background. Figure 7(a) is the original image. Figure 7(b), 7(c) and 7(d) has respectively k_b = 64, k_b = 128 and k_b = 256.

In some cases it is practically impossible to remove errors because the background tones are identical to the skin tones represented in the database. This is shown in some pieces of clothes in Figure 8.

Even including the image with background marks could not be sufficient to solve this problem. This means that color based segmentation may not be sufficient for certain situations.

Increasing the k_b value is highly recommended to improve the visual results of our method, but it is not adequate to use a huge value. A larger k_b constant compared to the k_o include more pixels to the background and not to the object skin set. Considering that many possible background regions will have skin-like tones, many correct skin regions will be labeled incorrectly.

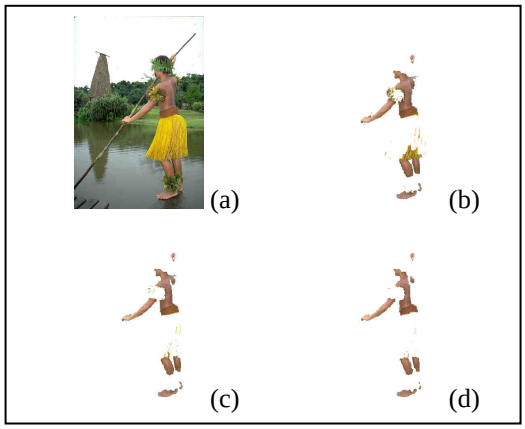

Fig. 8. Example of segmentation where some background píxels are not removed even increasing the mean background color parameter. Figure 8(a) is the original image. Figure 8(b), 8(c) and 8(d) has respectively k_b = 64, k_b = 128 and k_b = 256.

Figure 9 shows an example where the increase of the k_b parameter inserts errors in the correct skin region, making the labelling inadequate.

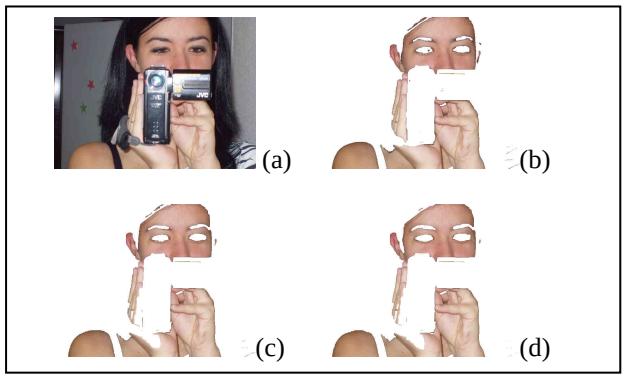

Fig. 9. Example of segmentation where the errors would increase with a huge k_b . Figure 9(a) is the original image. Figure 9(b) is a segmentation with k_b = 64, Figure 9(c) has k_b = 128 and Figure 9(d) has k_b = 256.

A comparison of our method with a classical skin segmentation based on RGB threshold [32] is presented as follows. This method was chosen because it is a standard technique very employed in this area. Ten images with manually user marked ground truth were used to evaluate tests comparing the method from [32] with ours. The images used in this experiment are presented in Figure 10.

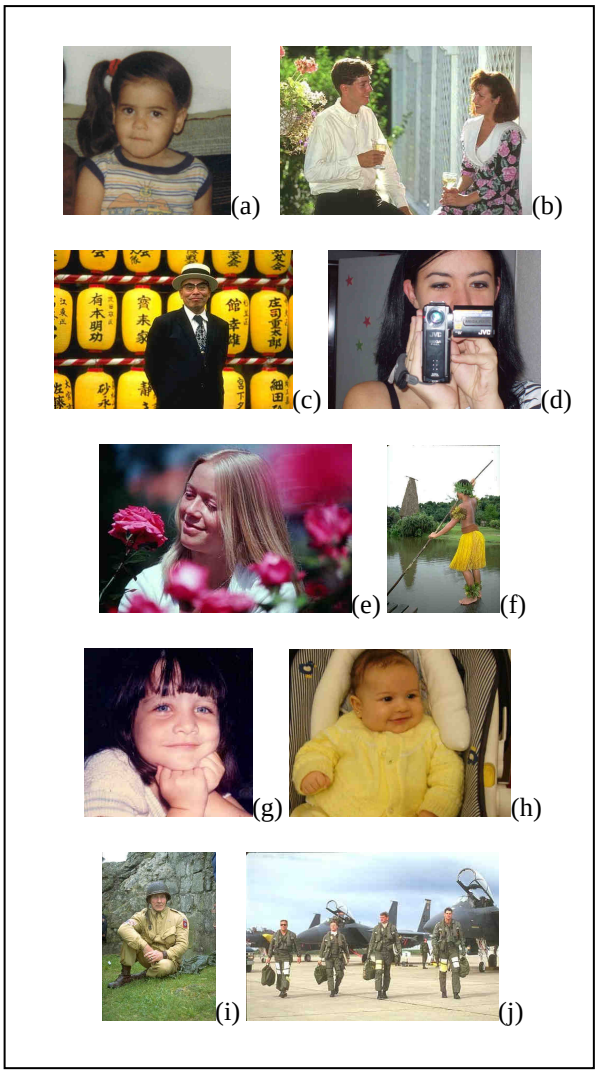

Fig. 10. Ten images used in tests comparison of the method from [32] with our method.

In order to compare the methods accuracy, all images of Figure 10 are analyzed by the corresponding classification errors: True Positives (TP), True Nega-

tives (TN), False Positives (FP) and False Negatives (FN). Four tables were generated based on these classifiers. Some examples of images with Ground Truths, segmentation results produced by our method and by the method in [32] are depicted in Figures 11 and 12. These images are represented by binary colors, such that black colors are background and white tones are skin. Table 1 evaluates the tests checking the accuracy of RGB skin threshold of [32]. Table 2 presents the results of our proposed method considering k_b = 64. Respectively, Table 3 and 4 shows the classifiers' rates with k_b = 128 and k_b = 256.

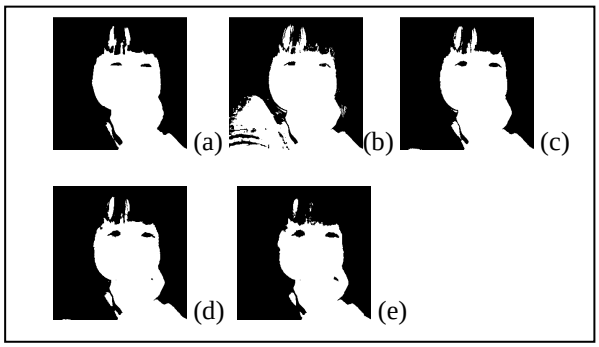

Fig. 11. Binary image segmentation of Figure 10(g). Black tones are identified as background and white tones are identified as skin. Figure 11(a) is the user marked ground truth. Figure 11(b) was generated with image segmentation method of [32]. Figure 11(c) is a segmentation using our method with k_b = 64, Figure 11(d) uses k_b = 128 and Figure 11(e) uses k_b = 256.

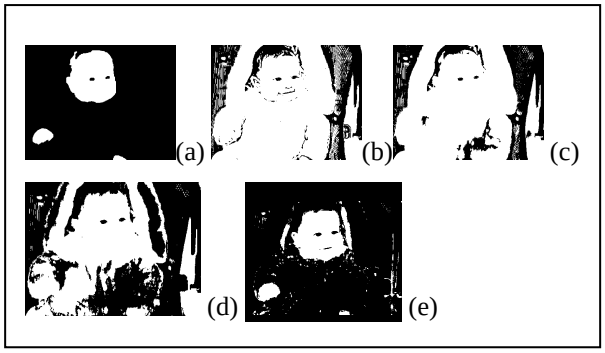

Fig. 12. Binary image segmentation of Figure 10(h). Black tones are identified as background and white tones are identified as skin. Figure 12(a) is the user marked ground truth. Figure 12(b) was generated with image segmentation method of [32]. Figure 12(c) is a segmentation using our method with k_b = 64, Figure 12(d) uses k_b = 128 and Figure 12(e) uses k_b = 256.

Table 1. Results generated by the method from [32] in total pixels percent. The first column indicates the images used from Figure 10. True positives, true negatives, false positives and false negatives are presented in percent in the next columns. Sixth column presents the total number of correct assigned pixels in the image and the last one shows the total number of wrong assigned pixels in the segmented image.

Img	TP	TN	FP	FN	Acc	Err
(a)	20.49	72.64	5.26	1.59	93.13	6.85
(b)	6.54	85.92	7.38	0.14	92.46	7.52
(c)	1.56	78.73	19.10	0.58	80.29	19.68
(d)	30.97	67.31	1.09	0.60	98.28	1.69
(e)	7.73	62.16	28.00	2.10	69.89	30.10
(f)	4.57	90.85	3.95	0.61	95.42	4.56
(g)	32.75	53.52	13.34	0.37	86.27	13.71
(h)	12.09	23.22	63.40	1.27	35.31	64.67
(i)	2.25	92.20	5.29	0.24	94.45	5.53
(j)	0.75	97.46	1.76	0.01	98.21	1.77

Table 2. Results of our skin method with k_b = 64 in total pixels percent. The first column indicates the images used from Figure 10. True positives, true negatives, false positives and false negatives are presented in percent in the next columns. The sixth column presents the total number of correct assigned pixels in the image and the last one shows the total number of wrong assigned pixels in the segmented image.

Img	TP	TN	FP	FN	Acc	Err
(a)	21.07	71.84	6.07	1.01	92.91	7.08
(b)	6.50	89.64	3.66	0.18	96.14	3.84
(c)	1.68	56.44	41.39	0.47	58.12	41.86
(d)	28.81	67.70	0.71	2.76	96.51	3.47
(e)	6.04	76.15	14.00	3.79	82.19	17.79
(f)	4.52	91.61	3.20	0.66	96.13	3.86
(g)	32.42	65.52	1.34	0.70	97.94	2.04
(h)	13.16	28.02	58.60	0.20	41.18	58.80
(i)	2.19	91.64	5.84	0.31	93.83	6.15
(j)	0.74	97.98	1.24	0.02	98.72	1.26

Table 3. Results of our skin method with k_b = 128 in total pixels percent. The first column indicates the images used from Figure 10. True positives, true negatives, false positives and false negatives are presented in percent in the next columns. The sixth column presents the total of correct assigned pixels in the image and the last one shows the total of wrong assigned pixels in the segmented image.

Img	TP	TN	FP	FN	Acc	Err
(a)	20.32	73.84	4.06	1.76	94.16	5.82
(b)	6.42	91.17	2.13	0.26	97.59	2.39
(c)	1.45	89.76	8.07	0.70	91.21	8.77
(d)	25.22	68.07	0.34	6.35	93.29	6.69
(e)	3.05	84.69	5.46	6.78	87.74	12.24
(f)	4.21	93.43	1.37	0.96	97.64	2.33
(g)	31.77	66.15	0.71	1.35	97.92	2.06
(h)	12.43	49.72	36.90	0.92	62.15	37.82
(i)	1.96	96.97	0.51	0.54	98.93	1.05
(j)	0.71	98.64	0.58	0.05	99.35	0.63

Table 4. Results of our skin method with k_b = 256 in total pixels percent. The first column indicates the images used from Figure 10. True positives, true negatives, false positives and false negatives are presented in percent in the next columns. The sixth column presents the total number of correct assigned pixels in the image and the last one shows the total number of wrong assigned pixels in the segmented image.

Img	TP	TN	FP	FN	Acc	Err
(a)	18.07	75.02	2.89	4.01	93.09	6.9
(b)	6.29	91.93	1.37	0.39	98.22	1.76
(c)	1.30	95.01	2.82	0.85	96.31	3.67
(d)	23.17	68.23	0.18	8.40	91.40	8.58
(e)	0.84	87.87	2.26	8.99	88.71	11.25
(f)	3.81	94.04	0.77	1.37	97.85	2.14
(g)	30.58	66.50	0.36	2.54	97.08	2.90
(h)	11.90	81.37	5.25	1.45	93.27	6.70
(i)	1.80	97.31	0.17	0.70	99.11	0.87
(j)	0.69	98.86	0.36	0.07	99.55	0.43

We can draw some conclusions from the experiments. Comparing Table 1 with Table 2, five segmented images have fewer errors with our method than with the classical one [32]. These images have background pixels with yellow and red tones that are skin-like tones. In our method, they are not easily confused because the database stores a heterogeneous set of background colors that reduces significantly the overall error.

However, many false positives were detected with k_b = 64, as shown in the images from Figure 10(e) and 10(h). This happened because many background tones in the segmented images are not well represented in the skin database. It is possible to reduce this error by increasing the total sample k_b of background tones considered. Consequently, more tones of image database will be considered, for instance, these mismatched yellow and red skin-like tones.

It can be easily seen that with the increase of the parameter k_b = 128, practically the total pixels errors of all images is dramatically reduced. This is obvious, because a more complete background description is considered during the assignment of a given pixel.

However, it is not guaranteed that with the increase of the background parameter to k_b = 256 the errors will be more reduced. As said before, the non-skin database contains the whole RGB set, including skin colors. With the increase of k_b, more correct skin pixels can be assigned to the database, because the considered non-skin background set of the database will be very huge compared to the skin object set and the propensity of set assignment is to include pixels to the background and not the object set, except if the analyzed pixel has skin tones very similar to the whole skin set. This situation occurs in some considered images as, for instance, in Figures 10(a), 10(d), 10(g). But for some image examples this increase can be extremely advantageous, as in Figure 12, where with the increase of the k_b, many yellow skin-like tones were removed, reducing the total errors.

8.2. Time complexity

Another problem that can influence significantly the final solution is the time complexity of the implementation. The computation of the proposed energy can be very slow depending on the total number of database images, user marks or on the k_b and k_o parameters. Considering these factors, it is very hard to do real-time segmentation on sequential machines. However, if we implement it on the GPU architecture, we can obtain efficient segmentation even for videos.

A comparison of the times to compute the cost function terms is featured in Table 5 using the same energy parameters both in the CPU and the GPU. It is also featured the computation of the graph-cut in Table 6. The CPU used is a Core 2 Duo with 2.53 Ghz and the GPU used in this specific experiment is a Nvidia Tesla C1060 with 240 stream processors. This configuration gives a good visual result but is slow to converge. This test was done using three different images and dimensions. These images are depicted in Figure 13.

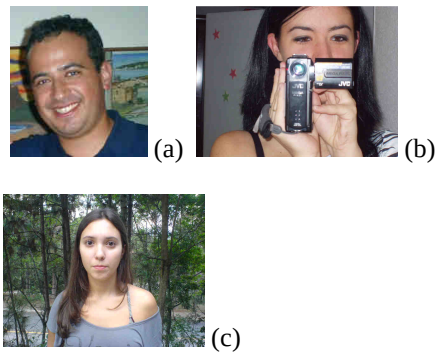

Fig. 13. Images used in the energy terms construction experiments.

Table 5. Comparison between CPU and GPU times for energy construction.

Image	Dimensions	CPU Time	GPU Time
(a)	188 x 205	29.5s	37ms
(b)	640 x 480	241.28s	239ms
(c)	1280 x 960	957.266s	897ms

Table 6. Comparison between CPU and GPU times for graph-cut computation.

Image	Dimensions	CPU Time	GPU Time
(a)	188 x 205	16ms	24ms
(b)	640 x 480	172ms	78ms
(c)	1280 x 960	1.73s	243ms

The construction of the energy terms for Figure 13 in Table 5 was extremely more efficient to compute in the GPU when compared to the CPU. The construction of the energy terms consists of only arithmetic operations, becoming more appropriate to be performed in the GPU than the CPU. A comparison of the graph-cut computation of the images in Figure 13 using both the CPU and GPU approaches is depicted in Table 6. As it can be seen, the graph cut computation in GPU is not as efficient as the energy construction, but it is possible to obtain a significant speedup.

Finally, we evaluate experiments with video segmentation. For this we defined a different database to improve the speed of the construction of the energy terms. Tests were done with video sequences of sizes equal to 320 x 240, 640 x 480 and 1280 x 720. Table 7 shows the results in terms of the number of frames per second for these videos. Considering that the energy construction is an expensive computational step, it is not appropriate to compute it at each frame. We did tests constructing the energy at every 5 frames, obtaining good visual results for low resolution video sequences (320 x 240), achieving approximately 41 FPS. However, with the increase of the resolution of the video, the number of frames per second reduced significantly because the energy construction and the graph-cut became more complex to compute. For high resolution videos the computation is slow, obtaining 6 FPS. We also computed the variance of the data in order to show that the data dispersion data is not significant. An example of segmented video sequences used in these tests is featured in the Figure 14.

Fig. 14. Video Segmentation results.

Table 7. FPS analysis of video sequences of different dimensions.

Video	Dimensions	Mean Time (ms)	Variance Time(ms)	FPS
(a)	320 x 240	24	0.0001	41
(b)	640 x 480	67	0.0001	15
(c)	1280 x 960	176	0.0008	6

9. Conclusion and Future Works

This works presents a new approach for efficient automatic human skin segmentation for image and videos using Graph-Cuts in GPUs. On traditional implementations of Graph Cuts, the energy function needs to be assigned by user marked seeds, what in this work it is not needed. The method uses a database of marked images which gives clues for the algorithm on what regions are skin or non-skin. As we presented, our method can yield good results when compared with traditional color segmentation techniques.

On future works, we intend to use new color spaces with the Graph Cuts approach as the HSL and Lab. This is necessary because the RGB color space, used entirely in this work, cannot deal with drastic variations, for instance, due to illumination. Also as we have shown, the color metric is sometimes not sufficient to yield an exact segmentation. In those cases, new features could be employed to our database approach, like textures.

Temporal coherence is another property that can also be used to improve the overall performance of

the method. Reusing the results obtained in a previous frame calculation as an input to the next frame can improve significantly the algorithm speed [25]. Also, real-time can be achieved by using GPU specific techniques like Loop Unrolling [40].

10. Acknowledgments

Anselmo Montenegro is grateful for the fund from FAPERJ under process number E-26/171.208/2006. Marcelo Bernardes Vieira is thankful for the fund from FAPEMIG. Esteban Clua is grateful for the fund from FAPERJ. The others authors are grateful to CNPq and CAPES Projects Pro CAD 224/2007.

References

[1] Baozhu, W.; Xiuying, C.; Cuixiang L. A Robust Method for Skin Detection and Segmentation of Human Face. In Proc. the International Conf. Intelligent Networks and Intelligent Systems, pp. 290-293, Tianjin, Nov. 2009.

[2] Boykov, Y.; Funka-Lea, G. Graph cuts and efficient n-d image segmentation. Int. J. Comput. Vision, Kluwer Academic Publishers, Hingham, MA, USA, v. 70, n. 2, pp. 109-131, 2006. ISSN 0920-5691.

[3] Boykov; Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. Proc. IEEE Int. Conf. on Computer Vision, pp. I:105-112, 2001.

[4] Boykov, Y.; Kolmogorov, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Anal. Mach. Intell., IEEE Computer Society, Washington, USA, v. 26, n. 9, pp. 1124-1137, 2004.

[5] Boykov, Y.; Veksler, O. Graph cuts in vision and graphics: Theories and applications. Springer-Verlag, 2006. pp.79-96.

[6] Boykov, Y.; Veksler, O.; Zabih, R. Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 23, p. 2001, 1999.

[7] Bray, M.; Kohli, P.; Torr, P. H. S. Posecut: Simultaneous segmentation and 3d pose estimation of humans using dynamic graph-cuts. In: ECCV. [S.l.: s.n.], 2006. p. 642-655.

[8] Burges, C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; Hullender, G. Learning to rank using gradient descent. In: ICML '05: Proceedings of the 22nd international conference on Machine learning. New York, NY, USA: ACM, 2005. pp. 89-96.

[9] Carro-Calvo, L.; Salcedo-Sanz, S.; Ortiz-García, E. G. and Portilla-Figueras A. An incremental-encoding evolutionary algorithm for color reduction in images. In Journal of Integrated Computed-Aided Engineering, IOS Pres, v. 17, n. 3/2010, pp.261-269, 2010.

[10] Cheddad, A. and Condell, J. and Curran, K. and Kevitt, P.M. A new colour space for skin tone detection. ICIP, p. 497-500, 2009.

[11] Cherkassky, B. V. A fast algorithm for computing maximum flow in a network. Collected Papers: Combinatorial Methods for Flow Problems, v. 3, pp. 90-96, 1979.

[12] Cormen, T. H., Leiserson, C. E. and Rivest, L. R. Introduction to Algorithms. MIT Press and McGraw-Hill, 1990.

[13] Edmonds, J.; Karp, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM, New York, NY, USA, v. 19, n. 2, pp. 248-264, 1972.

[14] Ford, L. R. and Fulkerson, A. D. R. Flows in Networks. Princeton Univ. Pr. 1962.

[15] Garret, Z. A.; Saito, H. Real-time online video object silhouette extraction using graph cuts on the gpu. Proceedings of the 15th International Conference on Image Analysis and Processing, p. 985-994, 2009.

[16] Goldberg, A. V.; Tarjan, R. E. A new approach to the maximum flow problem. Journal of the ACM, v. 35, pp. 921-940, 1988.

[17] Greig, D. M.; Porteous, B. T.; Seheult, A. H. Exact maximum a posteriori estimation for binary images. Journal of the Royal Statistical Society, 1989. Available in: <http://www.jstor.org/stable/2345609>.

[18] Han, J.; Award, G.M.; Sutherland, A.; Wu, H.; Automatic Skin Segmentation for Gesture Recognition Combining Region and Support Vector Machine Active Learning. IEEE International Conference on Automatic Face and Gesture Recognition, pp. 237-242, 2006.

[19] Hochbaum S. D. The Pseudoflow Algorithm and the Pseudoflow-Based Simplex for the Maximum Flow Problem. In 6th IPCO, pp. 325-337, 1998.

[20] Hussein, A. V. M.; Davis, L. On implementing graph cuts on cuda. First Workshop on General Purpose Processing on Graphics Processing Units, 2007.

[21] Ishikawa, H. Exact optimization for markov random fields with convex priors. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 25, pp. 1333-1336, 2003.

[22] Ishikawa, H.; Geiger, D. Segmentation by grouping junctions. Computer Vision and Pattern Recognition, IEEE Computer Society Conference, Los Alamitos, CA, USA, v. 0, p. 125, 1998.

[23] Jedynak, B.; Zheng, H.; Daoudi, M. “Skin detection using pairwise models,” IVC(23), n. 13, 29 Nov. 2005, pp. 1122-1130.

[24] Jones, M.; Rehg, J. Statistical color models with application to skin detection. Int. J. Comp. Vision, vol. 46, pp. 81-96, 2002.

[25] Juan, O.; Boykov, Y. Active graph cuts. In: CVPR, 2006. p.1023-1029.

[26] Kass, M.; Witkin, A.; Terzopoulos, D. Snakes: Active contour models. International Journal of Computer Vision, v. 1, n. 4, p. 321-331, 1988.

[27] Kirkpatrick, S.; Gelatt, D.; Vecchi, M. P. Optimization by simulated annealing. Science, v. 220, No. 4598, pp. 671-680, May 1983.

[28] Kohli, P.; Torr, P. H. S. Measuring uncertainty in graph cut solutions – efficiently computing min-marginal energies using dynamic graph cuts. In: ECCV. [S.l.: s.n.], 2006. pp. 30-43.

[29] Kolmogorov, V. Graph Based Algorithms For Scene Reconstruction From Two Or More Views. Phd Thesis - Cornell University, Ithaca, New York, USA, 2004.

[30] Kolmogorov, V.; Boykov, Y. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. Computer Vision, IEEE ICCV'05, v. 1, p. 564-571, 2005.

[31] Kolmogorov, V.; Zabih, R. What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 26, pp. 65-81, 2002.

[32] Kovac, P., Peer, P., and Solina, F. (2003). Human skin colour clustering for face detection. In EUROCON 2003.

[33] Kumar, M. P.; Torr, P. H. S.; Zisserman, A. Obj cut. In: CVPR '05: Proceedings of CVPR'05 Volume 1: IEEE, 2005. pp. 18-25.

[34] Lattari, L.; Conci, A.; Montenegro, A.; Clua, E.; Mota V.; Vieira, M. Colour based human skin segmentation using graph-cuts. Proceedings of IWSSIP 2010, ISBN 978-85-228-0565-5, Published by EdUFF, Editors F. R. Leta & A. Conci, 17-19 June 2010, Rio de Janeiro- Brazil. Number: 71, pp.223-226.

[35] Li, Y.; Jian, S.; Tang, C.; Shum, H. Lazy snapping. ACM Trans. Graph., ACM, New York, NY, USA, v. 23, n. 3, p. 303-308, 2004. ISSN 0730-0301.

[36] Lloyd, S. P. Least square quantization in PCM. IEEE Transact. Information Theory, vol. 28, n. 2, pp. 129-137, 1982.

[37] Lombaert, H.; Yiyong, S.; Grady, L.; Chenyang, X. A multilevel banded graph cuts method for fast image segmentation. (ICCV'05) Volume 1. Washington, DC, USA: IEEE Computer Society, 2005. pp. 259-265.

[38] López-Rubio, E.; Munõz-Pérez, J. and Gómez-Ruiz, J. A. A four-stage system for blind colour image segmentation. In Journal of Integrated Computed-Aided Engineering, IOS Pres, v. 10, n. 2/2003, pp. 127-137, 2003.

[39] Martin, D.; Fowlkes, D.; Tal, D.; Malik, J. A Database of Human Segmented Natural Images and Its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. Proc. Int'l Conf. Computer Vision, 2001.

[40] Nvidia. Nvidia CUDA Programming Guide. 2009.

[41] Park, A.; Jungwhan, K.; Seungki, M.; Sungju, Y.; Keechul J. Graph Cuts-Based Automatic Color Image Segmentation. In Proc. the International Conf. Digital Image Computing :Techniques and Applications (DICTA), pp. 564 - 571, Canberra, Dec. 2008

[42] Phung, S.; Bourzerdoum, A.; Chai, D. Skin Segmentation using color and edge information. Proc. Int. Symposium on Signal Processing and its Applications, pp.1-4, 2003.

[43] Ravichandran, K. S.; Ananthi , B.. Color Skin Segmentation Using K-Means Cluster. International Journal of Computational and Applied Mathematics, v. 4, n. 2, pp. 153-157. 2009.

[44] Rother, C.; Kolmogorov, V.; Blake, A. Grabcut: Interactive foreground extraction using iterated graph cuts. ACM Transactions on Graphics, v. 23, pp. 309-314, 2004.

[45] Roy, S.; Cox, I. A Maximum-Flow Formulation of the N-camera Stereo Correspondence Problem. In International Conference on Computer Vision, 1998.

[46] Sa, A.; Vieira, M.B.; Montenegro, A.A.; Carvalho, P.C.; Velho, L. Actively illuminated objects using graph-cuts. Computer Graphics and Image Processing, Brazilian Symposium on, IEEE Comp. Society, Los Alamitos, v. 0, pp. 45-52, 2006.

[47] Sethian, J. A. Level Set Methods and Fast Marching Methods - Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science. Cambridge University Press, 1999.

[48] Shi, J. and Malik, J. Normalized Cuts and Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 888-906, 2000.

[49] Sigal, L.; Sclaroff, S.; Athistos, V. "Estimation and prediction of evolving color distributions for skin segmentation under varying illumination," Proc. IEEE Conf. on Computer Vision and Pattern Recognition, vol. 2, pp. 152-159, 2000.

[50] Stalling, D.; Hege, H; Fur; K. Intelligent Scissors For Medical Image Segmentation. In B. Arnolds, H. Muller, D. Saupe, and T. Tolxdorff, editors, Proceedings of 4th Freiburger Workshop Digitale Bildverarbeitung in der Medizin, Freiburg, pages 32–36, March 1996.

[51] Vanhamel, I.; Pratikakis, I.; Sahli, H. Nonlinear multiscale graph theory based segmentation of color images, In Proc. the 18th International Conf. Pattern Recognition, pp. 407-411, Hong Kong, China, 2006.

[52] Vincent, L.; Soille, P. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Int., v. 13, pp. 583-598, 1991.

[53] Vineet, V.; Narayanan, P. Cuda cuts: Fast graph cuts on the gpu. CVPR'08 Workshops, p. 1-8, 2008.

[54] Veksler, O. Efficient Graph-Based Energy Minimization Methods In Computer Vision. Phd Thesis - Cornell University, Ithaca, New York, USA, 1999.

[55] Xu, N.; Ahuja, N.; Bansal, R. Object segmentation using graph cuts based active contours. Comput. Vis. Image Underst., Elsevier Science, New York, v. 107, n. 3, p. 210-224, 2007.

[56] Yildiz, A.; Akgul, Y. S. A gradient descent approximation for graph cuts. DAGM Symposium, pp. 312-321, 2009.

[57] Zhilan, H.; Wang, G.; Xinggang, L.; Hong Y. "Skin Segmentation Based on Graph Cuts," Tsinghua Science & Technology, Volume 14, Issue 4, August 2009, pp. 478-486.