PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 05/08

Using Quadtrees for Energy Minimization Via Graph Cuts

Cristina Nader Vasconcelos Asla Medeiros e Sá
Paulo Cezar Pinto de Carvalho Marcelo Gattass

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 05/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena February, 2008

Using Quadtrees for Energy Minimization Via Graph Cuts [1]

Cristina N. Vasconcelos[1], Asla Medeiros e Sá[2], Paulo Cezar Pinto de Carvalho[3] and Marcelo Gattass[1,2]

[1] Depto. de Informática - Pontifícia Universidade Católica (PUC-Rio).
Rua Marquês de São Vicente, 225. 22453-900 - Gávea, Rio de Janeiro, RJ, Brasil
[2] Tecgraf (PUC-Rio). Rua Marquês de São Vicente, 225. 22451-900 - Gávea, Rio de Janeiro, RJ, Brasil
[3] Instituto de Matemática Pura e Aplicada (IMPA).
Estrada Dona Castorina, 110. 22460 - Jardim Botânico, Rio de Janeiro, RJ, Brasil

crisnv@inf.puc-rio.br, asla@tecgraf.puc-rio.br, pcezar@impa.br,
mgattass@tecgraf.puc-rio.br

Abstract.

Energy minimization via graph cut is widely used to solve several computer vision problems. In the standard formulation, the optimization procedure is applied to a very large graph, since a graph node is created for each pixel of the image. This makes it difficult to achieve interactive running times. We propose modifying this set-up by introducing a pre-processing step that groups similar pixels, aiming to reduce the number of nodes and edges present in the graph for which a minimum cut is to be found.

We use a quadtree structure to cluster similar pixels, motivated by fact that it induces an easily retrievable neighborhood system between its leaves. The resulting quadtree leaves replace the image pixels in the construction of the graph, substantially reducing its size.

We also take advantage of some of the new GPGPU concepts and algorithms to efficiently compute the energy function terms, its penalties and the quadtree structure, allowing us to take a step toward a real time solution for energy minimization via graph cuts. We illustrate the proposed method in an application that addresses the problem of image segmentation of natural images by active illumination.

Keywords: Graph Cuts; Quadtrees; GPGPU

Resumo.

[1]This work has been sponsored by Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

O método de minimização de energia via corte de grafo é amplamente utilizado para resolver diversos problemas de visão computacional. Em sua formulação tradicional, o procedimento de optimização é aplicado a um grande grafo, uma vez que é criado um nó do grafo para cada pixel da imagem, tornando difícil atingir taxas de processamento em tempo real. Propomos uma modificação de dessa formulação, introduzindo um passo de pré-processamento que agrupa pixeis semelhantes no intuito de reduzir o número de nós e arestas presentes no grafo para o qual o corte mínimo será encontrado.

Utilizamos a estrutura de uma quadtree para clusterizar pixeis semelhantes, motivados pelo fato de que ela induz a um sistema de vizinhança entre suas folhas facilmente recuperável. As folhas de quadtree resultante substituem os pixeis da imagem na construção do grafo, reduzindo seu tamanho substancialmente.

Conceitos de programação genérica utilizando o hardware gráfico (GPGPU) também são incorporados para computar eficientemente os termos da função de energia, suas penalidades e a estrutura da quadtree, permitindo avançar na direção de uma solução em tempo real para minimização de energia via o Graph Cuts. O metodo proposto [e ilustrado em uma aplicação que aborda o problema de segmentação de imagens naturais por iluminação ativa.

Palavras-chave: Corte de Grafo; Quadtrees; GPGPU

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introduction

Many important problems in image analysis can be posed as optimization problems involving the minimization of some kind of energy function. For some of those problems, methods based on computing the minimum cut on graphs offer the possibility of finding global minimum for some classes of energy functions [8].

These methods explore the fact that algorithms for computing minimum cuts in polynomial time have been known for some time [3].

Much research has been done in setting the mathematical requirements for the energy functions that justify the use of Graph Cut minimization for both exact and approximate cases [3],[2],[8]. The applicability of the technique has also been shown by several papers in themes like image segmentation [14], foreground/background extraction [12], clustering [16], texture synthesis[9], photo composition[1] and so on.

However, the use of graph-cut methods for real-time applications has been limited by the size of the graph in which optimization must take place. In this paper we propose a preprocessing of the input images, in order to produce a new set of nodes and edges, instead of the image pixels and its neighborhood commonly used for the graph construction. The proposed sets are considerably smaller, inducing a significant reduction on the running time of the graph-cut procedure. We call Quad Cut the use of graph cut minimization in

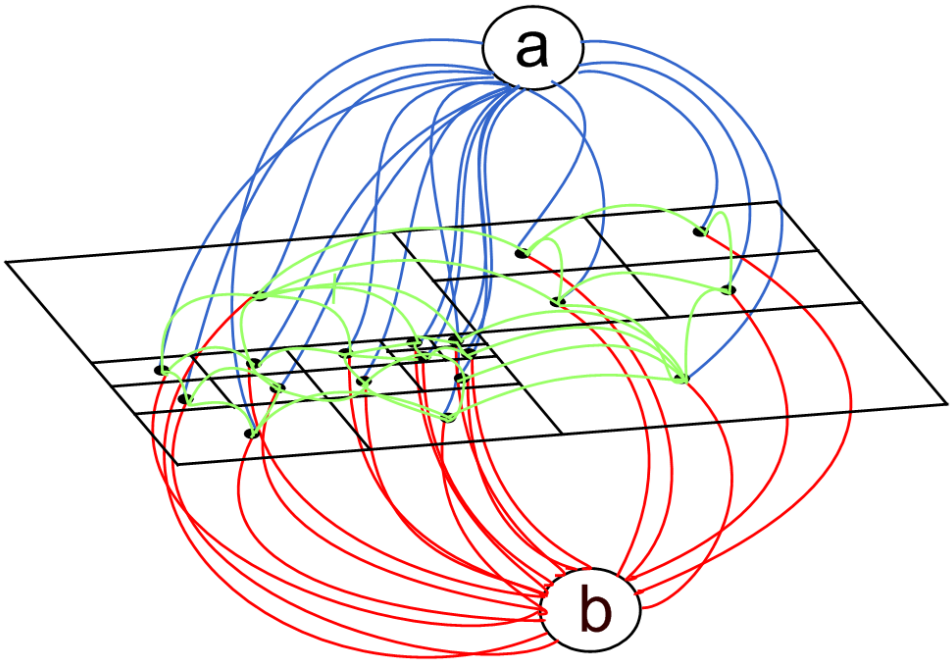

Figure 1: Quad Graph

this modified way, the concept is illustrated in Figure 1.

The idea of the preprocessing is to group similar pixels, but in a way that creates a well known neighborhood system. For that reason, we choose to group them into Quadtree nodes. The metric used for grouping should be a similarity criteria appropriate to the context being analyzed by the energy function.

After constructing the quadtree, its leaves are used, instead of image pixels, as the basis for the construction of the graph. An appropriate energy function and neighborhood relationships are created to be used in this new procedure.

As we are interested in offering a fast approximation for the computer vision problems that rely on computing the minimum cut on an appropriately constructed graph, in addition to reducing the graph size, we also explore graphics hardware to efficiently compute energy function terms, its penalties and the Quadtree structure. Inspired by [17], we can take advantage of the Graphics Processing Unit (GPU) parallelism to compute all the preprocessing steps, including an efficient construction of a Quadtree with all the information needed for the optimization algorithm, leaving the CPU free to minimize the Graph constructed with the quad leaves.

Figure 2: Example of minimization via Graph Cuts to the image segmentation of natural images aided by active illumination. In (a) and (b) the input images are shown. In (c) the initial segmentation provided by active illumination is compared to the final optimized segmentation shown in (d). The composition result (using parameters $\sigma_L = 0.25$, $\sigma_C = 0.05$) is shown in (e).

As an application, we address the problem of foreground/background image segmentation aided by active illumination, in which graph cuts are used to compute an optimal binary classification, starting with an initial background/foreground separation, provided by the difference in intensity levels for two different illumination levels [12]. Figure 2 illustrates the application. Observe that the quality of the binary segmentation produced can be used for matting.

The paper is organized as follows: some applications that use energy minimization via graph cuts in vision are reviewed in the next Section; Section 3 briefly describes the basic concepts for energy minimization via Graph Cuts; then, in Section 4 we argue that

grouping pixels into the Quadtree structure is useful to substantially reduce the nodes of the final graph to be cut. An GPU implementation to construct the quad tree structure is discussed in section 5. In Section 6 we present an illustrative implementation to accelerate the active illumination segmentation problem. Results are discussed in Section 6.4 followed by conclusions and future work.

2 Related Work

In the Computer Vision and Graphics context, the graph cut method, can be interpreted as a clustering algorithm that works in a image feature space to produce spatially coherent clusters as result. Several recent works creatively models different applications as a labeling problem, then uses graph cuts to optimize the proposed labeling. This is the case in [1], where a framework for composing digital photos into a single picture, called "digital photomontage", is described. Having n source images $S_1, ..., S_n$ to form a photo composition, the problem is posed as choosing a label for each pixel p, where each label represents a source image S_i. The proposed method extends the applicability of graph cuts to compute selective composites, photo extended depth of field, relighting, stroboscopic visualization of movement, time-lapse photo mosaics and panoramic stitching.

In [16], the spatial clustering problem is modeled as a labeling problem. The spatial coherence is guaranteed by the penalty imposed for neighboring pixels to have different labels, that are used as weights for the edges between neighbor pixels in the graph.

In [9], texture synthesis is modeled as labeling. The method generates textures by copying input texture patches into a new location, the graph-cut technique is used to find the optimal region inside the patch to be transferred to the output image. Such patch fitting step is a minimum cost graph cut problem using a matching quality measure for pixels from the old and new patch.

The problem of monochrome image colorization is modeled as a segmentation problem in [15]. The input image is partitioned interactively while the user specifies input colors, maintaining smoothness almost everywhere except for the sharp discontinuity at the boundaries in the image.

Image segmentation problem can also be solved by minimization via graph cuts. The main work lies in defining the energy function that models the specified application. In particular, background/foreground segmentation can be solved by means of Graph Cuts. In [11], [14] and [10] the user has to indicate coarsely the foreground and the background pixels, as initial restrictions for a minimization process. Then, graph cuts are used to find automatically the globally optimal segmentation for the rest of the image.

Similarly to our algorithm, [10] proposed the use of the image uniform regions as the nodes used in the graph construction in stead of the image pixels. They group similar pixels into such regions segmenting the original image using the watershed method. We believe that such segmentation do not provide a neighborhood system neither a boundary perimeter and area as easy to compute as the one presented in our proposal provided by the quadtree structure.

In this paper we will concentrate on applying graph cuts for image foreground-background segmentation aided by *active illumination*, as in [12]. Active illumination consists of using an additional light source in the scene that illuminates the foreground objects more strongly than the background. This gives *a priori* clues of the foreground. The information derived

from this difference in illumination replaces the indication of object and background pixels by the user. These initial clues are then used as seeds for an optimization procedure in order to obtain a high quality segmentation. Potentially, the approach could be used for video capture, since a projector can be controlled to produce alternating illumination conditions at 60 Hz.

3 Basic concepts in Energy Minimization via Graph Cuts

In Computer Vision and Graphics, energy functions minimization is commonly computed using the min-cut/max-flow algorithms. The general goal for using the min-cut/max-flow algorithms is to find a labeling L, that assign each variable $p \in P$ (usually associated with the pixels of the input image) to a labeling $L_p \in L$, which minimizes the corresponding energy function.

The number of possible values assumed by the variables of the energy function is assumed finite, and modeled as a set of labels L, each label representing a possible output value.

The energy function to be optimized can be generally represented as [2]:

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{p,q \in N} V_{p,q}(L_p, L_q), \tag{1}$$

Traditionally, $N \subset P \times P$ is a neighborhood system on pixels, $D_p(L_p)$ is a function that measures the cost of assigning the label L_p to the pixel p, while $V_{p,q}$ measures the cost of assigning the labels $\{L_p, L_q\}$ to the adjacent pixels p and q and is used to impose spatial smoothness.

The method of Graph Cuts to minimize (1) is applied by the creation of a graph normally containing nodes corresponding to each of the image pixels and some additional special nodes, called terminals, corresponding to each of the possible labels. There are two types of edges in the graph: n-links and t-links. N-links are the edges connecting pairs of neighboring pixels, representing the neighborhood system in the image, while t-links are edges connecting pixels with terminals nodes. All edges in the graph are assigned some weight or cost related to the energy function terms. The cost of a t-link corresponds to a penalty for assigning the corresponding label to the pixel, derived from the data term D_p in (1). The cost of a n-links corresponds to a penalty for discontinuity between the pixels. These costs are usually derived from the pixel interaction term $V_{p,q}$ in (1).

The Graph Cut finds a minimum of the energy function (1), providing an optimal labeling for the graph nodes [2].

4 Grouping Pixels into Quadtrees Leaves

When modeling computer vision problems as a energy-minimization problem, one can use different kinds of image features (e.g., luminance, color, gradient, frequency) and different metrics (e.g., statistical functions, differences between images, min/max relations). However, whatever the image feature or the metric used in the energy function, most natural images have areas of pixels presenting similar values according to them. Those pixels are expected to receive the same label in the energy minimization output. Our approach takes

advantage of this fact, grouping pixels of such uniform areas, thus decreasing the graph size on which the min-cut algorithm is to be applied.

One more question arises here. If, on one hand, grouping pixels reduces the size of the graph, on the other hand, it may cause its adjacency topology to be more complex than the usual 4- or 8-connected pixel neighborhood systems. This may lead to spending considerable time both to find suitable clusters of pixels and to compute their adjacency relationships, overcoming the benefits by the smaller graph size.

Driven by these observations, we propose the use of a quadtree structure for grouping pixels into regions using a similarity criteria, while, at the same, creating a manageable neighborhood system between the quadtree leaves, in which adjacency relationships are easily retrievable.

In the next subsections we show how a graph for energy minimization can be constructed using quadtree leaves. The construction of the quadtree itself is discussed in section 5.

4.1 Graph Cuts using Quadtree Leaves

Using the quadtree leaves as the input data for the energy minimization via graph cuts, our goal is to find a labeling L, that assigns a label $L_t \in L$ to each leaf $t \in T$ of the quadtree, that minimizes the energy function adopted. The same set of the labels L may be used here. The modified energy function can be generally represented as:

$$E(L) = \sum_{t \in T} \alpha * D_t(L_t) + \sum_{t,u \in N} \beta * V_{t,u}(L_t, L_u), \tag{2}$$

Where $N \subset T \times T$ is a neighborhood system on the quadtree leaves, $D_t(L_t)$ is a function that measures the cost of assigning label L_t to leaf t, and $V_{t,u}$ measures the cost of assigning labels $\{L_t, L_u\}$ to the adjacent leaves t and u. The α and β terms are weights for balancing the energy function, explained below.

In such energy function model, the energy variables represent the quadtree leaves. Thus, graph cut minimization is applied to a graph containing nodes corresponding to each leaf of the quadtree and terminal nodes corresponding to each of the possible labels. Now, the n-links connect pairs of neighboring leaves, while t-links connect leaves with terminals nodes.

4.1.1 Weighting the Quadtree Nodes

The α and β factors were added to equation (2) in order to balance the energy metric according to leaves topology. The number of pixels inside a leaf t is $(2^{level(t)})^2$, while the number of pixels in the border between two neighboring leaves t and u is $2^{\min(level(t),level(u))}$. Therefore, we can rewrite (2) by taking α, that represents the weight for the regional term, as the leaf area, and β, that represents the weight for the boundary term, as the number of neighboring pixels between the two leaves.

With the suggested weights, we ensure that larger leaves have greater impact than smaller ones, while also enhancing the neighborhood influence of larger borders.

$$E(L) = \sum_{t \in T} (2^{level(t)})^2 * D_t(L_t)$$

$$+ \sum_{t,u \in N} 2^{\min(level(t),level(u))} * V_{t,u}(L_t, L_u), \quad (3)$$

5 Efficiently computing the Quadtrees

In this section we describe how the quadtree can be constructed efficiently using graphics hardware.

5.1 Quadtrees in GPGPU

The increasing use of the Graphics Processing Unit (GPU) for general-purpose computation (GPGPU) is motivated by its newest capability of performing more than the specific graphics computations which they were designed for.

In the context of our proposal, the GPU can be used for efficiently computing the energy function terms and also for constructing the quadtree whose leaves will be used as nodes in the graph cut minimization. For saving the partial results, we apply the useful concept of "Playing Ping-Pong with Render-To-Texture" [5], rendering to Frame Buffer Objects (FBO) [6] when 32-bit floating-point precision is necessary.

A solution for constructing a quadtree structure for general purposes in GPU is presented in [17]. A reduction operator is described that creates an image pyramid called QuadPyramid. The operator writes in each fragment of the pyramid texture whether it represents a grouping of similar pixels or if it should be threaded as a quadtree internal node, in this case saving the number of leaves covered by the region represented by the fragment.

In a second shader, they identify the quadtree leaves reading the pyramid texture repeatedly, simulating tree traversals from root to leaves. Relative counters, read from the pyramid texture, are used to control such traversals. The origin and size of the found leaves are saved in a output texture, organized as a point list. To construct such list for a quadtree of m leaves over a square image of N pixels, their algorithm may need $(m * \log(\sqrt(N)))$ texture accesses in the worst case.

For our purposes, the resulting quadtree leaves will be used in CPU for graph construction. In addition to the origin and size of the leaves, we will also need leaf values that are used as the graph weights. We propose a simpler image pyramid operator for quadtree construction than the used in [17] and a new algorithm for identifying leaves from the pyramid texture. Next sections explain our methods for quadtree construction and leaves identification.

5.2 Quadtree Construction

Once a similarity criteria has been selected, the input image should be transformed to the adopted metric space, previously to the quadtree construction. For example, when grouping pixels by luminance, the original image should be transformed to the luminance space.

Here, as in [17], the quadtree construction starts by a reduction operator, creating an image pyramid. For each fragment in the pyramid level being constructed, the operator reads four texture samples from the previous pyramid level, representative of its four children in the quadtree. If the samples represent similar nodes, then, the fragment is

classified as a leaf, grouping them into a single node that receives its children mean value. Otherwise, the fragment is classified as a tree internal node. The reduction operator is performed until the pyramid top level (1×1 pixel dimension) is reached.

Our algorithm is simpler than the one presented in [17]. While grouping leaves, [17] also computes relative counters in fragments representing internal nodes. Those counters indicate how many leaves are covered by the internal node being processed. In our case, we do not count the existing leaves inside a internal node region because this information is not needed in our leaves isolation solution.

For our purposes, the pyramid texture is used for saving the grouping decision (alpha channel) and the leaves values (RGB channels). Figure 3 shows an image pyramid found using the example application of section 6.

Figure 3: image pyramid found using the example application

5.3 Identifying Final Leaves

In order to identify the quadtree leaves in the pyramid texture, we propose a leaf isolation method that does not require computing several texture transversals, as used in [17], and, as a consequence, does not impose the use of a GPU supporting several nested branches.

Using the pyramid image as input, this processing step produces a texture whose pixels contain the data corresponding to a quadtree leaf (its size, position and representative value), or a color associated with empty data. This texture saves all the data needed for building the graph *a posteriori*.

Our algorithm erases texels representing other than leaf nodes in the pyramid texture. For that, we use a new fragment shader that reads our pyramid texture and discards all fragments that should not be leaf nodes in the final tree. This shader produces the output texture in a single rendering pass that makes at most two texture accesses per fragment.

The cleanup shader initially reads the fragment classification (leaf/non-leaf) from the alpha channel of the pyramid texture. If the sample is already classified as non-leaf, the fragment is immediately discarded. Otherwise, the pyramid texture is queried again, now on its corresponding parent texture coordinate. When the parent was classified as a leaf, this means that this fragment was grouped with its neighbors into a higher level leaf, so it can also be discarded. However, in the case of a non-leaf parent, this means that the previous shader could not group this node with its neighbors and that the fragment represents a leaf in the final tree.

The fragments that pass through those tests are considered as final quadtree leaves and are written in the output texture, saving in its channels all the data to be associated with the leaf that the fragment represents (see figure 4). By doing this, we guarantee that subsequent steps of the graph construction do not have to query any other texture.

Figure 4: Quadtree Leaf Texture

All the information necessary for graph-cut computing is contained in this texture. For illustration, in figure 5 we reconstruct the entire quadtree using only the leaf texture shown in figure 4). Each leaf is painted according to its level.

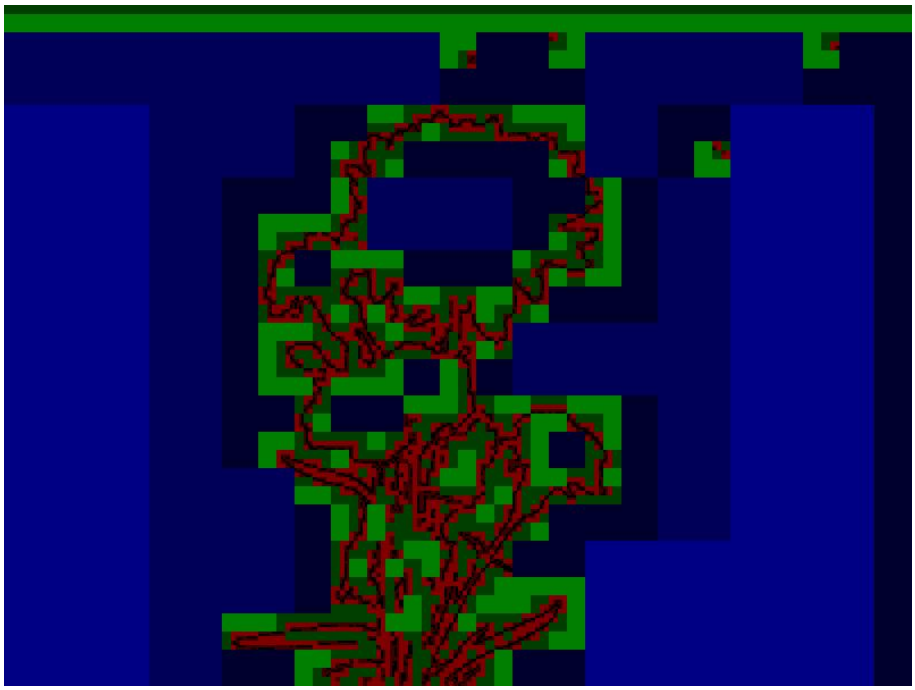

Figure 5: Found Quadtree (leaf color according with its level).

6 Application to Active Segmentation

In this section we describe in detail an application of the proposed method to the problem of image segmentation by active illumination using graph cuts.

Segmentation using active illumination employs a single, intensity-modulated light source that stays in a fixed position between shots, as proposed in [12]. The two shots, differently illuminated, are used to obtain an initial segmentation used as a seed, referred as *segmentation seed*, and to attribute weights to the pixels that are used in graph cut optimization step to produce a improved final segmentation.

6.1 Energy Function Definition

The objective function adopted is the same proposed in [12]. The regional term considers the luminance difference between the two input images and the object color histogram as information that characterize the segmentation. The luminance difference for background pixels is considered to have Gaussian distribution, with density

$$\mathrm{p}_B(p) = \frac{1}{\sqrt{2\pi}\sigma_L}\exp(\frac{-|L_{I_2}(p) - L_{I_1}(p)|^2}{2\sigma_L^2}), \tag{4}$$

where σ_L is the standard deviation of the luminance differences, illustrated in figure 6 b.

The segmentation seed is defined as $O = \{p \mid \mathrm{p}_B(p) < t\}$, where t is a small threshold.

The color histogram of these initial foreground pixels are used to characterize the object as in [11]. In this work, only the components a and b of the Lab color systems are considered to characterize the object color distribution. For simplicity, the histogram is defined over a uniform partition.

The object distribution function is modeled as

$$\mathrm{p}_O(p) = \frac{n_k}{n_O} \tag{5}$$

where n_k is the number of pixels assigned to the bin k and n_O is the number of pixels in the object region O.

Observe that only one of the input images is used to construct the histogram information, since mixing different images may distort color information. In most cases, we use the image corresponding to the lowest projected intensity.

The regional term of the energy function is:

$$R(x_p) = \begin{cases} -\log(\mathrm{p}_O(p)), & \text{if } x_p \text{ is } 1 \\ -\log(\mathrm{p}_B(p)), & \text{if } x_p \text{ is } 0 \end{cases} \tag{6}$$

where 1 is foreground and 0 is background.

The likelihood function for neighboring boundary pixels given by

$$\mathrm{B}(p,q) = 1 \; - \; \exp(\frac{-(||Lab(p) - Lab(q)||)^2}{2\sigma_C^2}), \tag{7}$$

where $Lab(p)$ denotes the color at point p and σ_C is the standard deviation of the L^2-norm of the color difference.

The boundary term for neighboring pixels p, q is given by $-|x_p - x_q| \log \mathrm{B}(p, q)$, where points q are the neighbors of p.

The final objective function combines both the regional and the boundary term and is given by:

$$E(\mathbf{X}) = \sum_{p \in I_1} R(x_p) - \sum_{p,q \in I_1} |x_p - x_q| \cdot \log \mathrm{B}(p, q), \tag{8}$$

As shown in [12], the proposed energy function is regular, which means that it can be minimized by graph-cuts. This remains valid for the modified energy function defined on quadtrees leaves. As a consequence, Quad-Cuts can be applied to minimize the modified energy function.

6.2 Energy Function in GPU

The next sections describe how shaders can be used to compute efficiently the regional and boundary terms of the active illumination energy function applying GPGPU.

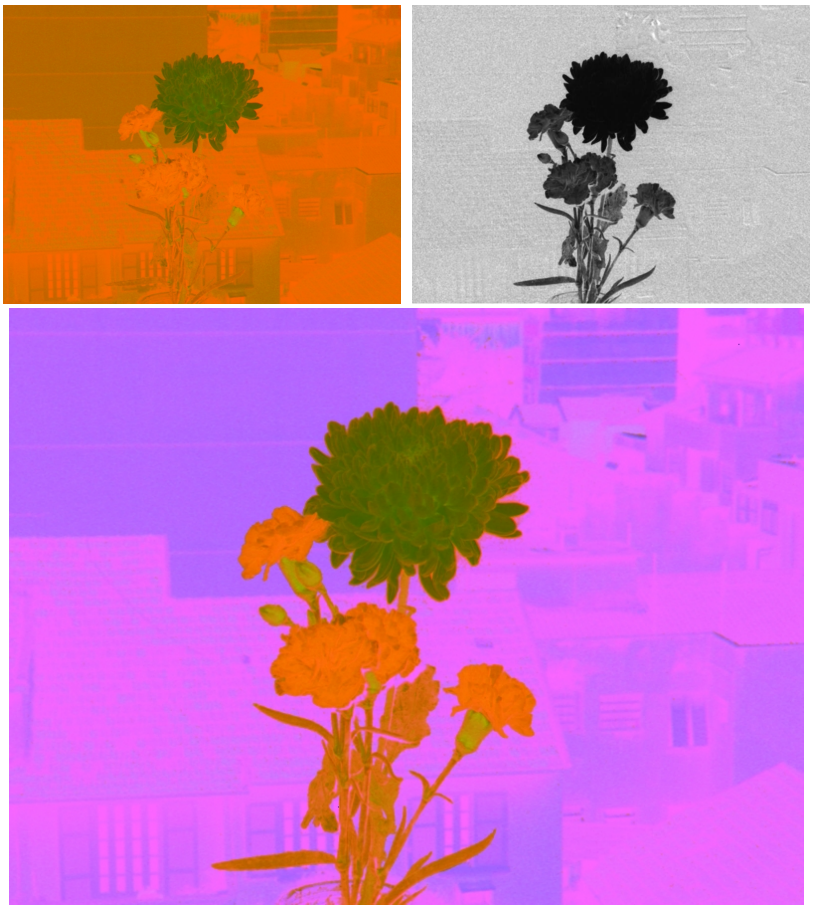

Figure 6: Stratified Texture.

To pass the computed data efficiently across the algorithm we create what we call a *Stratified Texture*, illustrated in Figure 6.

The Stratified Texture is generated by saving, in its different channels, red, green, blue and alpha, all the data needed for the following steps of our algorithm. In this example application, the red and green channels are used for storing the a and b channels of the input image converted to Lab color space, the blue channel for storing the initial seed segmentation obtained by thresholding the luminance difference, and the alpha channel for storing the background distribution.

6.2.1 Color Space Conversion

The input images are converted from RGB to CIE Lab color space, to exploit metrics in a perception-based color space presenting orthogonality properties between luminance and chrominance information.

Shaders for color space conversion have been used intensively by GPGPU programs. However, in order to efficiently compute the RGB to Lab conversion with high precision we also take advantage of the concept of rendering to texture with 32 bit floating point internal format using frame buffer objects (FBO) [6]. We save the Lab a and b computed channels in the resulting texture r and g channels, as illustrated in figure 6(a).

6.2.2 Background Probability

The background probability is computed in GPU according to equation (4), measuring the distribution of the luminance difference of the lit and unlit images. The result is illustrated in Figure 6(b).

For efficiently using the GPU parallelism, we pre-compute the constants $1/\sqrt{2\pi}\sigma_L$ and $1/(2\sigma_L^2)$ of equation (4) for a fixed σ_L. Those values are passed to the shader, avoiding repeatedly calculating it for every fragment.

6.2.3 Computing the Color Distribution

In order to compute the object distribution function using equation (5), we construct the histogram of the a and b channels from Lab color space (saved in stratified texture red and green channels), distinguishing object pixels using the object seed (from the stratified texture blue channel).

Motivated by its performance in computing histograms with a large set of bins, we choose to adapt [13] to our application context. Originally, that approach was proposed for monochromatic histograms, computing the histogram bin selection in a vertex shader, by loading the texture using either vertex texture fetches or by rendering the input image pixels into a vertex buffer, according to the graphics hardware capability.

We propose to adapt [13] to a vertex shader that computes bin selection in a 2D mapping, modifying it to compute a histogram representing the frequencies of occurrence in both input channels. Our vertex shader computes the vertex position by reading the a and b channels, multiplying their normalized values by the number of bins in the corresponding dimension, and then transforming the resulting values to frame coordinates.

Observe that a histogram of a trichromatic image could also be computed in GPU using techniques for representing 3D arrays such as those proposed in [7].

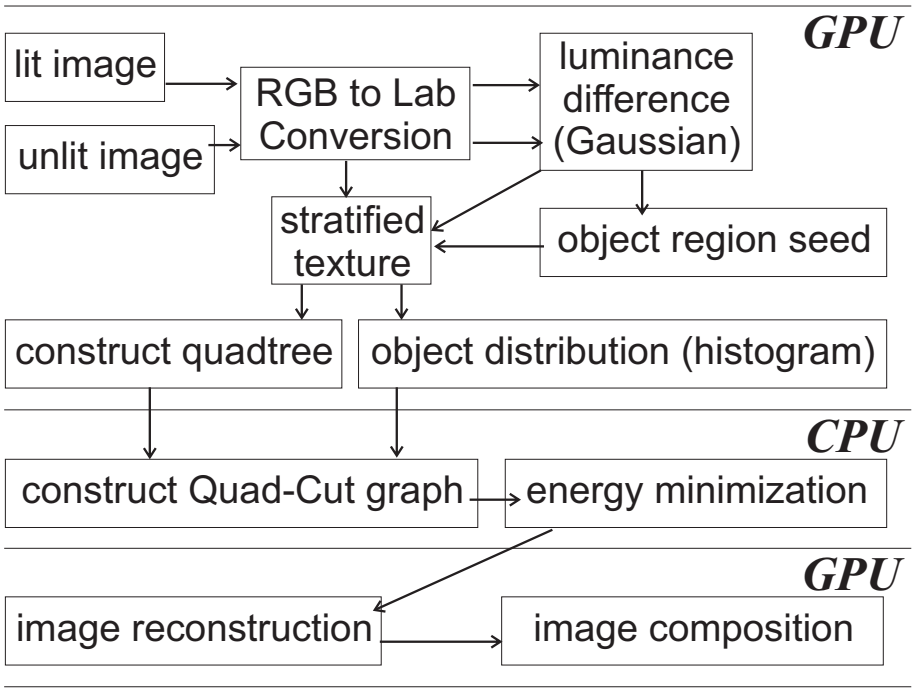

Figure 7: The proposed Quad-Cut method.

6.3 Application pipeline

The main steps of the example application are illustrated in Figure 7.

The lit and unlit input images are converted to Lab color space. Then, another shader computes the background distribution texture. The result of those shaders are grouped in the stratified texture as described in section 6.2 and illustrated in figure 6.

The object distribution function is obtained by computing the object histogram of the a and b channels read from the stratified texture, using only pixels that failed the background threshold test (read from its blue channel). This histogram is saved in a texture to be used later in the energy function construction.

Then, the quadtree is created using our reduction operator through the stratified texture. Following the method in section 5.3, the resulting pyramid texture in cleaned, generating a texture that contains only the leaf nodes. that contains all information needed about each leaf: its level, from its relative position in the texture; its a and b from LAB conversion saved in the red and green channels; and the luminance distribution, saved in the blue channel.

All the above steps are computed in GPU. After them, the graph is constructed in CPU by reading the data from the leaf texture (fig. 4) and from the histogram textures.

In CPU we store the quadtree leaves in a pointer less representation, as a linear quadtree. The leaves are associated with location codes for fast neighbor search as in [4].

The graph is constructed using the leaf data, which stores the previously computed terms of the objective function, according to the method explained in section 4, which is minimized by the Graph-Cut minimization as in [3].

The solution of the minimization provides the classification of the quadtree leaves as background or foreground. So, using the position and size of each leaf, we reconstruct the resulting image that represents the alpha mask solution.

Back to the GPU, for the final composition, a smooth shader is applied to the computed alpha mask. Finally, a blending operator $\alpha F + (1 - \alpha)B$ is applied to the segmented foreground and the new background.

6.4 Results

Segmentation results using Quad-Cuts and the final compositions are shown in figures 2 and 8. To illustrate the considerable reduction in the number of variables in the minimization problem, both figures 2 and 8 are originally 800×600 (480,000) pixels, while the computed quadtrees have 9,556 (2%) leaves and 30,036 (6%) leaves, respectively. Notice that the special characteristics of figure 8 (that presents many holes and thin structures) are automatically preserved through 15,992 leaves in the lowest level (1×1 pixel) and 8,718 in the next level (4×4 pixels).

We also measured the execution time of an background/foreground segmentation using graph-cut and active illumination with a Quad-Cut implementation with its preprocessing steps computed in GPU. A NVIDIA GeForce 7900 graphic card was used for the timings shown in Table 1.

Table 1: Processing time

step	*in seconds*
Energy function on GPU:	
RGB to Lab	< 0.001
Background prob	< 0.001
Histogram	< 0.015
Quad on GPU:	
Pyramid Construction	0.047
Quad Leaves Isolation	< 0.001
Quad on CPU:	
Reading Texture to CPU	0.015
Leaf List	0.016
Neighborhood	0.014
Graph-cut Minimization	0.001
Answer Reconstruction	0.016

7 Conclusions

We propose to accelerate the computation of energy minimization using graph cuts by applying a pre-processing step for reducing the number of graph nodes and edges. In this pre-processing, pixels are grouped by a similarity criteria according to the problem context.

We argue in favor of using a quadtree structure for managing such clustering regions, motivated by the easily retrievable neighborhood system between its leaves. In order to

Figure 8: Composition Result 2 (using $\sigma_L = 0.25$, $\sigma_C = 0.05$).

support our claim, we present a general formulation of the energy function using the leaves as its variables, and we also presented a general graph-cut construction over the quadtree leaves.

We also show how the quadtree structure can be constructed using graphics hardware. Initially, we use a reduction operator for constructing an image pyramid that writes in each texel whether a similarity clustering was applied or not. Such shader is simpler than the one proposed in [17]. Then we propose a leaf isolation method that discards from the pyramid texture all the texels that do not represent a quadtree leaf, efficiently removing unneeded information of non-leaf nodes. The proposed method requires fewer texture readings than the method proposed by [17], due to fact that the algorithm that it employs for finding leaves does not compute tree traversals for discovering each leaf in the tree.

Our graph construction method does not compute a point list on GPU of the quadtree leaves, as [17] does. Instead, as explained before, we use the leaf texture data to save the weights of the computed energy function, and the leaf texture coordinates are used to set the leaf level, size and corner position. Saving all the data needed for the posterior steps into such leaf texture allows an efficient interplay between the result generated in GPU and the energy minimization on CPU.

We also presented an application of our method to the foreground/background segmentation problem. It can be observed from the presented results (figures 2 and 8) that the proposed method for grouping pixels into quad leaves conserved image fine grain details of the original image (by creating leaves as small as 1×1) while also featuring a good grouping rate, by creating large leaves in regions of similar pixels .

We also show that the efficient implementation of all preprocessing steps on GPU leads to reasonably fast processing rates. As a consequence, we believe that our method constitutes an important step towards real time segmentation and matting using active segmentation.

References

[1] Agrawala, A., Doncheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., Cohen, M.: Interactive digital photomontage. In: Computer Graphics Proceedings ACM SIGGRAPH, pp. 294–302 (2004)

[2] Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in computer vision. In: Proc. of Int'l Workshop Energy Minimization Methods in Computer Vision and Pattern Recognition (2001)

[3] Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Transactions on PAMI **23**, 1222–1239 (2001)

[4] Frisken, S.F., Perry, R.: Simple and efficient traversal methods for quadtrees and octrees. journal of graphics tools **7**(3), 1–11 (2002)

[5] Goddeke, D.: Playing Ping Pong with Render-To-Texture (2005)

[6] Green, S.: The opengl framebuffer object extension. Games Developers Conference (GDC (2005)

[7] Harris, M., Luebke, D., Buck, I., Govindaraju, N., Kruger, J., Lefohn, A., Purcell, T.: Gpgpu: General-purpose computation on graphics hardware. In: Tutorial at ACM SIGGRAPH 2005 (2005)

[8] Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? Proc. IEEE Transactions on Pattern Analysis and Machine Inteligence **26**, 147–159 (2004)

[9] Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video syntesis using graph cuts. In: ACM Transactions Graphics, Proc. SIGGRAPH (2003)

[10] Li, Y., Sun, J., Tang, C.K., Shum, H.Y.: Lazy snapping. In: Proceedings of ACM SIGGRAPH 2004, pp. 303–308 (2004)

[11] Rother, C., Kolmogorov, V., Blake, A.: Grabcut - interactive foreground extraction using iterated graph cuts. ACM Trans. Graph. **23**(3), 309–314 (2004). DOI http://doi.acm.org/10.1145/1015706.1015720

[12] Sá, A., Vieira, M., Montenegro, A., Carvalho, P., Velho, L.: Actively illuminated objects using graph-cuts. In: Proceedings of SIBGRAPI 2006 (2006)

[13] Scheuermann, T., Hensley, J.: Efficient histogram generation using scattering on gpus. In: SI3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games, pp. 33–37. ACM Press, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1230100.1230105

[14] Wang, J., Bhat, P., Colburn, R., Agrawala, M., , Cohen, M.: Interactive video cutout. Computer Graphics Proceedings ACM SIGGRAPH (2005)

[15] Yun-Tao, J., Shi-min., H.: Interactive graph cut colorization. The Chinese Journal of Computers. **29**(3), 508–513 (2006)

[16] Zabih, R., Kolmogorov., V.: Spatially coherent clustering using graph cuts. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04). **2**, 437–444 (2004)

[17] Ziegler, G., Dimitrov, R., Theobalt, C., Seidel, H.P.: Real-time quadtree analysis using histopyramids. In: B.E. Rogowitz, T.N. Pappas (eds.) IS&T and SPIE Conference on Electronic Imaging, Proceedings of SPIE-IS&T Electronic Imaging, pp. XX–XX. International Society for Optical Engineering (SPIE), SPIE and IS&T, San Jose, USA (2007)