Universidad Complutense de Madrid

Facultad de Informática

Ingeniería en Informática

Aplicación de Graph Cuts en la edición de imágenes y vídeo

Sistemas Informáticos

Curso 2010-2011

Alumnos del grupo:

Adrián Gustavo Flores Hernández

María Trinidad Martín Campos

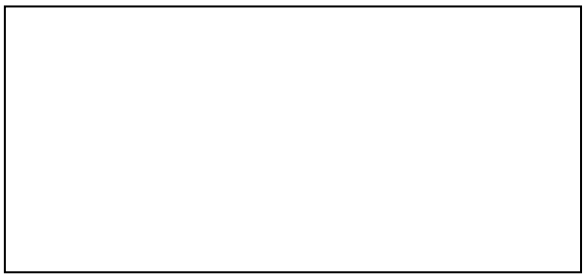
Luca Renna

Dirigido por D. Pedro J. Martín de la Calle

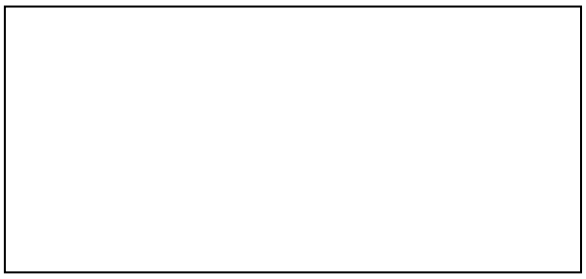
Autorización

Autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Adrián Gustavo Flores Hernández



María Trinidad Martín Campos



Luca Renna

Resumen

Siempre hemos oído hablar o hemos utilizado programas de retoques fotográficos que modifican las imágenes y vídeos aplicándoles muchos efectos gráficos de manera intuitiva, sin darnos cuenta de la verdadera complejidad que está detrás de estos procesos.

El objetivo del proyecto es la creación de un editor de imágenes y vídeos que permita reconocer objetos como el *background* (fondo) y objetos en el *foreground* (primer plano), y aplicarle un cierto efecto como la sustitución de esas regiones por otra imagen.

Nuestro fin ha sido el de entender cómo funcionan algunos algoritmos complejos para conseguir estos efectos visuales y artísticos, a partir de conocimientos sobre informática gráfica moderna.

Palabras claves: Graph Cuts, Minimización de Energías, Background, Foreground, Imágenes, Vídeos,

Abstract

We have always heard that graphic editing programs allow people to manipulate a photograph or a video frame and apply any kind of visual effect in an easy manner. But we inadvertently ignore the complexity behind these processes.

The main goal of this Project is the creation of a graphic editing program that recognizes the background object or foreground object and apply to these items various kinds of effects such as automatic picture substitution of the recognized region.

Our last ambition was to understand how some of these complex algorithms work so that we could apply them to achieve certain visual and artistic effects.

Key words: Graph Cuts, Fast Energy Minimization, Background, Foreground, Photograph, Pictures, Frames.

Índice

Capítulo IV

Introducción

El objetivo del proyecto es la creación de un editor de imágenes y vídeos que permita reconocer objetos como el *background* (fondo) y el *foreground* (primer plano), y aplicarle un cierto efecto, como la sustitución de estos elementos por otras imágenes. Para ello hemos estudiado algunos algoritmos basados en *graph cuts* (cortes de grafos). Estos métodos se aplican mucho en el campo de la visión por computador ya que resuelven una amplia variedad de problemas de *visión artificial* de manera eficiente, como el suavizado de imágenes, problemas estereoscópicos y en general, cualquier problema de visión computacional que puede ser formulado en términos de minimización de energías.

Motivación

Siempre hemos oído hablar o hemos utilizado programas de retoques fotográficos que modifican las imágenes y vídeos aplicándoles muchos efectos gráficos de manera intuitiva, sin darnos cuenta de la verdadera complejidad que está detrás de estos procesos.

La elección de este proyecto es debida al interés que tenemos en el mundo de la informática gráfica y a la intención de aplicar conocimientos adquiridos en un campo en continuo desarrollo.

Nuestro fin ha sido el de entender cómo funcionan algunos algoritmos complejos para conseguir un producto correcto que incluya algunos efectos visuales y artísticos interesantes a partir de nuestros conocimientos sobre informática gráfica.

A lo largo de esta memoria trataremos temas como el background en 2D aplicado a geometrías, simulación/animación de objetos, imágenes y renderización.

Memoria

La presente memoria está estructurada de la siguiente manera.

- En el Capítulo I se introducen las nociones teóricas sobre la minimización de energía mediante graph cuts.

- En el Capítulo II se presenta una breve descripción de los graph cuts y una especificación detallada de los algoritmos usados en el proyecto.

- En el Capítulo III se explican los pasos seguidos para el desarrollo de la aplicación.

- En el Capítulo VI detallamos las funcionalidades de la aplicación y los resultados obtenidos.

Capítulo I

Minimización De Energías Mediante Graph Cuts

Introducción

El problema de reconocer regiones diferentes en una imagen consiste, como en muchos problemas de visión por computador, en asignar una etiqueta a cada píxel. Si dos píxeles de una imagen pertenecen a etiquetas diferentes, es equivalente a decir que los dos píxeles perteneces a regiones diferentes.

Como se explica muy bien en [FAEMGC], estas tareas se resuelven, de forma natural, si consideramos la asignación de etiquetas en términos de minimización de la energía.

La minimización global de estas funciones de energía es NP-difícil, incluso en los casos de problemas simples. Los algoritmos basados en graph

cuts proporcionan una solución aproximada al problema de manera más eficiente.

En el proyecto se utilizan dos algoritmos que se presentan en [FAEMGC] basados en recortes de grafos que, de manera eficiente, encuentran un mínimo.

Por el contrario, muchos algoritmos estándar, utilizan pequeños movimientos donde sólo pueden cambiar la etiqueta de un píxel al mismo tiempo. El algoritmo de expansión encuentra un sistema de etiquetado a partir de unos datos iniciales del mínimo global, mientras que el algoritmo swap soporta más funciones de energía.

Minimización de la energía en problemas de visión

Como se ha mencionado anteriormente a cada píxel perteneciente a una imagen se le debe asignar una etiqueta dentro de un conjunto finito . Para nosotros el conjunto de etiquetas son los números naturales; al ejecutar el programa se define cuántas etiquetas se quieren hallar para la imagen que se está considerando.

El objetivo es encontrar un etiquetado que asigne a cada píxel p una etiqueta perteneciente al conjunto , donde es, a la vez, suave y consistente para los datos observados. Esto quiere decir que a un píxel se le asigna una etiqueta sólo si es parecido a los píxeles que tienen etiqueta . Este problema de visión puede, formularse en términos de minimización de la energía.

Se busca el etiquetado que minimiza la energía de la manera siguiente:

Aquí, mide el grado en que no es suave, es decir, cuantos más píxeles parecidos hay en , más bajo es el valor de . Mientras que la medida mide el desacuerdo entre y los datos observados.

Esto quiere decir que a cada píxel de una imagen se le asigna una etiqueta (inicialmente de manera aleatoria) y se calcula su energía asociada según la ecuación (1). Cuando ésta energía en mínima, hemos encontrado una solución buena para el nuestro problema, es decir un buen etiquetado para la imagen.

La forma de se traduce a:

donde mide cómo de bien se ajusta la etiqueta del píxel p, según los datos observados en la imagen.

En nuestro proyecto la función , donde es la intensidad de en la imagen, esto es la media de los canales R, G y B de , y es la media de las intensidades de todos aquellos píxeles de la imagen que tienen asignada la etiqueta .

La forma de es del tipo:

donde es la función de penalización que mide cómo de bien se ajustan las etiquetas asignadas a píxeles adyacentes. Es decir, se añade una penalización a la energía cuando dos píxeles adyacentes, y similares en cuanto a intensidad, tienen etiquetas diferentes.

La elección de es una cuestión crítica, ya que se pueden usar muchas funciones diferentes. En nuestro proyecto usamos una función truncada cuadrática: para el primer algoritmo. Para el segundo algoritmo usamos el *Potts Model:* , donde es igual a 1 si su argumento es cierto, y 0 si n caso contrario. es una constante, que vale 5 en nuestra implementación.

En definitiva, la energía de un conjunto de etiquetas se calcula mediante la fórmula:

En el siguiente capítulo se describen los dos algoritmos que se han utilizado en el proyecto: el primer algoritmo se llama swap. El segundo algoritmo, se llama expansion.

Capítulo II

Descripción de los algoritmos

Introducción

Los algoritmos que detallamos en esta sección generan un etiquetado que es un mínimo local de energía para dos tipos de grandes movimientos: el expansion y el swap. En la siguiente sección se explica brevemente que es un graph cuts y los grandes movimientos. Luego se detallará la implementación que hemos hecho de los dos algoritmos mencionados anteriormente.

Graph Cuts

Antes de describir los algoritmos de expansion y el swap, vamos a revisar los graph cuts. Sea un grafo con dos vértices distinguidos llamados terminales (en la figura). Un corte en el grafo es un conjunto de aristas de tal manera que los terminales están separados en el grafo inducido por . Además, ningún

subconjunto propio de separa los terminales de . El costo del corte , denotado por , es igual a la suma de pesos de sus aristas. El problema de *reducción mínima* es encontrar las aristas con menor peso, entre todos los cortes, que separan los terminales.

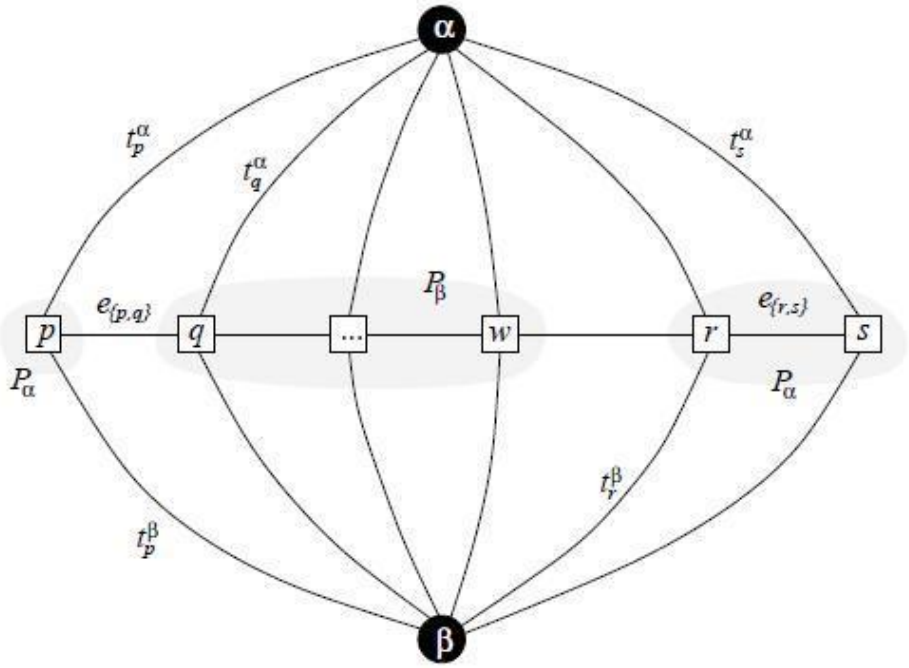


Figura 1: Ejemplo grafo para una imagen de 1D. El conjunto de píxeles de la imagen es donde y .

Se puede notar que cuando se hace un corte en el grafos, todos los píxeles que se separan de , se le asignará la etiqueta , por el contrario a todo los píxeles que se separan de se le asignará la etiqueta . Al contrario que los movimientos estándar (p.e. enfriamiento simulado) estos movimientos permiten que un gran número de píxeles puedan cambiar sus etiquetas de forma simultánea (gracias a un solo corte en el grafo). Esto hace que el conjunto de etiquetados en un solo movimiento sea exponencialmente grande. Por ejemplo, los movimientos de expansión son tan fuertes que cualquier etiquetado localmente óptimo con respecto a estos movimientos, se encuentra dentro de un factor conocido del mínimo global. Esto quiere decir, como demostrado en [FAEMGC], que cuando el algoritmo expansion encuentra una solución está es la óptima.

Los algoritmos expansion y el swap difieren por los distintos modos de construcción del grafo a cortar, estos se detalla en la siguiente sección.

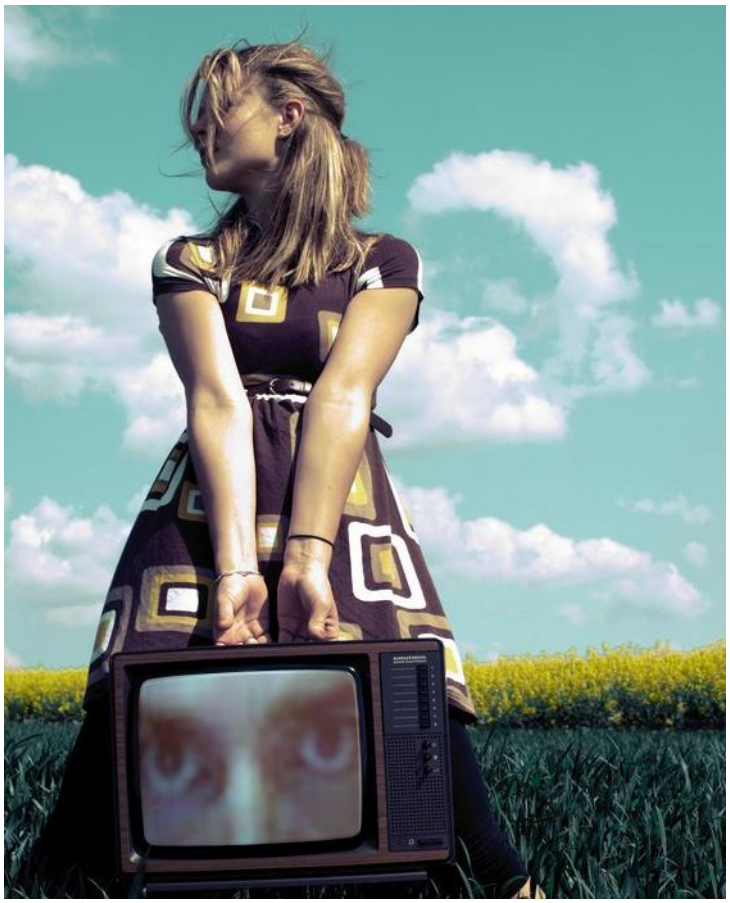
Algoritmo Alpha expansion

Hemos dividido la implementación del algoritmo en seis pasos, para que se entienda mejor el código.

La aplicación de este algoritmo a nuestro proyecto ha sido la siguiente:

1. Se genera un etiquetado aleatorio
2. Obtenemos la energía actual de la imagen
   2.1. Inicializamos las variables
3. Mientras la energía obtenida no mejore a la actual, y no supere al número de etiquetas hacemos:
   3.1. Se ejecuta el graph cut en su versión expansion.
   3.2. Recalculamos el nuevo etiquetado
   3.3. Asignamos nuevas etiquetas
4. Al salir del bucle for
   4.1. Obtenemos nueva energía después de los graph cuts
   4.2. Si la energía obtenida es mejor, se la asginamos a la energía actual
   4.3. Si no es mejor, variable booleana del bucle while a true
5. Al salir del while, mostramos la imagen con las nuevas etiquetas halladas.

Input: Foto original



Nota: los métodos resaltados en negrita, se mostraran y explicaran de forma independiente

```
void execAlphaExpansionAlgorithm()
{
    setArbitraryLabeling();
```

En el etiquetado "arbitrario" obtenemos el cociente entre el número total de píxeles y el número total de etiquetas (elegidas por el usuario). Este cociente delimita sectores dentro de la imagen de forma secuencial, donde iremos asignando etiquetas del mismo valor para cada sector.

```
    bool fin = false;
    float Ecurr = getCurrentEnergy();
    int j = 0;
```

```
while (!fin)
{
        for (int i = 0; i < numlabels && !fin; i++)
        {
                createAndExecGraphCutExpansion(i);
                recalculateLabelingValuesExpansion(i, true);
```

Recorremos el vector Pixel, consultamos su campo etiqueta y, para todas aquellas que sean iguales, calculamos su media aritmética (sumatorio de sus intensidades entre su número de etiquetas). Una vez calculada la media, recorremos el vector de labels y asignamos a cada etiqueta el valor de la media.

```
                swapLabelsExpansion(i);
```

Asignamos las nuevas etiquetas, a las antiguas, debido a que los valores de las etiquetas han sido modificados tras la ejecución del graph cuts.

```
        }

        float Enew = getNewEnergy();
        if (Enew < Ecurr)
        {
                Ecurr = Enew;
        }
        else
        {
                fin = true;
        }
}
showLabelingImage();
```

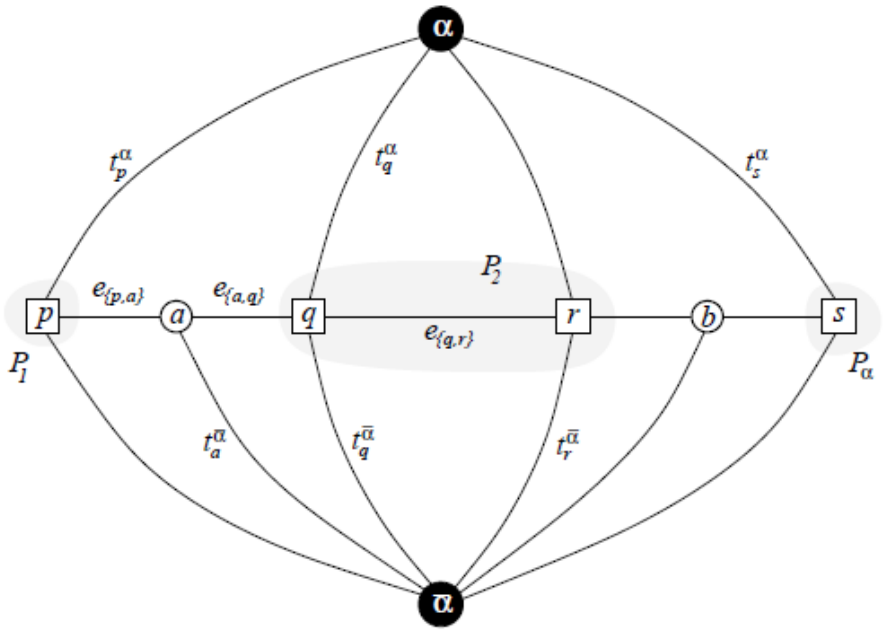
Almacenamos la matriz de etiquetas en una variable para luego ser mostrada por pantalla y así enseñar el resultado del algoritmo α-expansion.

}

Output: Imagen de etiquetas

void **createAndExecGraphCutExpansion**(int alpha)

A través de este método, la idea es conseguir un grafo que tenga la siguiente forma y aplicarle el min-cut/max-flow (método específico de graph cuts que usamos):



Los pesos asignados a las aristas son los que se reflejan en la siguiente tabla:

edge	weight	for
$t_p^{\bar{\alpha}}$	∞	$p \in \mathcal{P}_\alpha$
$t_p^{\bar{\alpha}}$	$D_p(f_p)$	$p \notin \mathcal{P}_\alpha$
t_p^{α}	$D_p(\alpha)$	$p \in \mathcal{P}$
$e_{\{p,a\}}$	$V(f_p, \alpha)$	$\{p,q\} \in \mathcal{N},\ f_p \neq f_q$
$e_{\{a,q\}}$	$V(\alpha, f_q)$	
$t_a^{\bar{\alpha}}$	$V(f_p, f_q)$	
$e_{\{p,q\}}$	$V(f_p, \alpha)$	$\{p,q\} \in \mathcal{N},\ f_p = f_q$

Tabla 1 pesos asignados a las aristas para el expansion

De forma que:

```
for (int i = 0; i < numpixels; i++)
    {
            createNlinksExpansion(g, i, alpha);
```

Creación de todas las aristas y (vistas en el dibujo) para el grafo.

```
            if (pixels[i].label == alpha)
            {

              g -> add_tweights(i, 0, INT_INF);
```

La arista que une un nodo consigo mismo tendrá peso infinito.

```
            }
```

```
        else
        {
                float weightnalpha = getDValue(i, pixels[i].label);
```

El peso de cada una de las aristas será la media calculada anteriormente menos la intensidad del píxel (también calculada), lo que se conoce en términos estadísticos como Varianza.

```
                g -> add_tweights(i, 0, (int) weightnalpha);
```

Se asigna el peso calculado a las aristas.

```
        }
        if (pixels[i].label >= 0)
        {
                float weightalpha = getDValue(i, alpha);
                g -> add_tweights(i, (int) weightalpha, 0);
```

Mismo procedimiento anterior pero calculando las .

```
        }
    }
    g -> maxflow();
```

Aplicamos el algoritmo al grafo creado.

```
float getNewEnergy()
{
    return getEnergySmoothValue(true) + getEnergyDataValue(true);
```

Para cada par de píxeles vecinos, se calcula la diferencia de sus intensidades en valor absoluto ( ), se compara dicha diferencia con una constante empírica de valor 5 y, si es menor, se asigna una penalización de (donde es otra constante empírica) debido a la disparidad de píxeles; sino, la penalización tendrá valor :

$$U(|I_p - I_q|) = \begin{cases} 2K & \text{if } |I_p - I_q| \leq 5 \\ K & \text{if } |I_p - I_q| > 5 \end{cases}$$

}

Algoritmo Alpha-Beta

Hemos dividido la implementación del algoritmo en seis pasos, para que se entienda mejor el código.

La aplicación de este algoritmo a nuestro proyecto ha sido la siguiente:

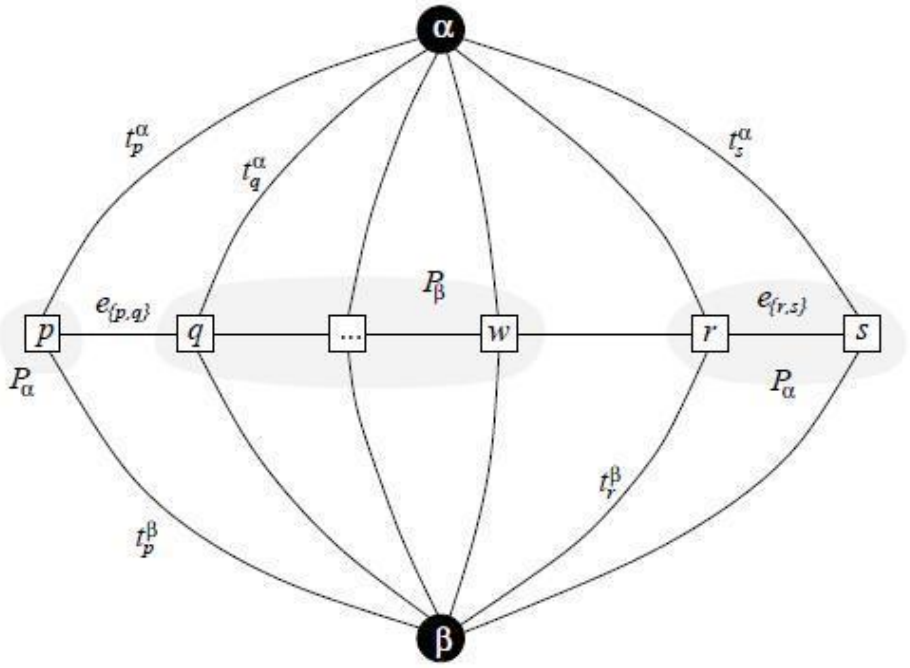
1. Se genera un etiquetado aleatorio
2. Obtenemos la energía actual de la imagen
   2.1. Inicializamos las variables
3. Se recorre la matriz de píxeles por encima de la diagonal (de forma triangular):
   3.1. Se ejecuta el graph cut en su versión swap.
   3.2. Recalculamos el nuevo etiquetado.
   3.3. Obtenemos la nueva energía.
   3.4. Si es menor que cero, terminamos el bucle.
   3.5. Si es menor que la actual (mejor) cambiamos etiquetado antiguo por el nuevo.
   3.6. Si no, no hemos encontrado un etiquetado mejor, terminamos el bucle.
4. Al salir del bucle, mostramos la imagen de etiquetas.

Input: Foto original

Nota: los métodos resaltados en negrita, se mostraran y explicaran de forma independiente

```
void execSwapAlgorithm()
{
    setArbitraryLabeling();
    bool fin = false;
    float Ecurr = getCurrentEnergy();
    int j = 0;
    while (!fin)
    {
            for (int i = 0; i < numlabels && !fin; i++)
                    for (int j = i + 1; !fin && j < numlabels; j++)
                    {
                            createAndExecGraphCutSwap(i, j);
```

A través de este método, la idea es conseguir un grafo que tenga la siguiente forma y aplicarle el min-cut/max-flow (método específico de graph cuts que usamos):



Los pesos asignados a las aristas son los que se reflejan en la siguiente tabla:

edge	weight	for
t_p^α	$D_p(\alpha) + \sum_{\substack{q \in \mathcal{N}_p \\ q \notin \mathcal{P}_{\alpha\beta}}} V(\alpha, f_q)$	$p \in \mathcal{P}_{\alpha\beta}$
t_p^β	$D_p(\beta) + \sum_{\substack{q \in \mathcal{N}_p \\ q \notin \mathcal{P}_{\alpha\beta}}} V(\beta, f_q)$	$p \in \mathcal{P}_{\alpha\beta}$
$e_{\{p,q\}}$	$V(\alpha, \beta)$	$\substack{\{p,q\} \in \mathcal{N} \\ p,q \in \mathcal{P}_{\alpha\beta}}$

Tabla 2 pesos asignados a las aristas para el swap

```
                    recalculateLabelingValues(i, j, true);
                    float Enew = getNewEnergy();

                    if (Enew < 0)
                            fin = true;

                    if (Enew < Ecurr)
                    {
                            swapLabels(i, j);
```

Análogo a swapLabelsExpansion(i), con la diferencia de que, en este caso, hay dos etiquetas.

```
                            Ecurr = Enew;                    }
                    else
                    {
                            if (j > numlabels)
                                    fin = true;
                    }
            }

        if (j > numlabels)
                fin = true;

        j++;
    }
    showLabelingImage(false);
}
```

Output:Imagen de etiquetas

```
void createAndExecGraphCutSwap(int alpha, int beta)

for (int i = 0; i < numpixels; i++)
    {
            if (pixels[i].label == alpha || pixels[i].label == beta)
            {
                    createNlinksSwap(g, i, alpha, beta);
```

Calcula el peso de las aristas . Consulta en el grafo "g", el píxel "i", obtiene sus vecinos (izquierda, derecha, abajo, arriba), calcula la varianza para cada uno de ellos y asigna ese valor al peso de la arista que los une con el píxel "i".

```
                    float  weightalpha = getDValue(i, alpha) +
getVNeighborsTlinkWeightValue(i, alpha, alpha, beta);
                    float  weightbeta = getDValue(i, beta) +
getVNeighborsTlinkWeightValue(i, beta, alpha, beta);
```

getVNeighborsTlinkWeightValue es análogo al método createNlinksSwap, aplicándose a alpha y beta. Calcula .

```
                    g -> add_tweights(I, (int) weightalpha, (int) weightbeta);
                }
            }

            g -> maxflow();
}
```

Capítulo III

Desarrollo de la Aplicación

Introducción

El proyecto se ha desarrollado usando el lenguaje de programación C++ porque consideramos la manipulación de imágenes y vídeos como una fase crítica en términos de tiempo, y sólo los lenguajes no interpretados pueden garantizar una velocidad de ejecución rápida.

Además hemos decidido poder ejecutar el programa en múltiples plataformas, lo cual implica la elección de las librerías portables que se detallan a continuación. Además se describen también algunos detalles de implementación y se enumeran las clases creadas.

Entorno de ejecución

Para que la compilación del código sea lo más homogénea posible, hemos utilizados un entorno de ejecución portable: Eclipse con el

compilador GCC. Aunque este entorno es más optimizados para el lenguaje de programación Java, tiene un soporte para C++ suficiente para nosotros.

Un pequeño inconveniente que hemos tenido con Eclipse en Windows ha sido que el sistema no lleva el compilador C++ por defecto. Este problema se ha resuelto instalando *MinGW* (Minimalist GNU for Windows), que es una implementación de los compiladores GCC para la plataforma Win32.

Implementación del Graph Cuts

Al principio del trabajo, se había optado por la implementación directa del graph cuts según se explica en [CORMEN] en particular la implementación del algoritmo *push relabel*.

Debido a problemas de tiempo y a resultados muy pobres, decidimos abandonar su desarrollo en favor de una librería que ya implementa dicha funcionalidad. Al principio preferimos usar la librería Boost de C++, pero nos resultó muy complicada, y al final elegimos una librería mucho más sencilla, desarrollada por Yuri Boykov y Vladimir Kolmogorov.

Clases utilizadas

En esta sección se detallan y describen las clases que se han creado con el desarrollo de la aplicación.

- VideoProcessing:
  Esta clase permite cargar y guardar los vídeos, y permite aplicar cualquiera de los algoritmos que hemos implementado sobre cada frame.
  Métodos más importantes:

- load: este método carga un vídeo en memoria, para ello descompone el vídeo en secuencia de imágenes que se guardan en un array.
- processFrame: extrae una imagen del vídeo.
- applingToAll: aplica por cada frames uno de los algoritmos vistos anteriormente.
- applingToAllThreaded: hace lo mismo que el método anterior pero con hilos para aprovechar la potencia de los procesadores multicore.
- getOneFrame: devuelve un frame.
- convertFrame: convierte cada frame del vídeo al formato interno que usamos nosotros.
- copyToFrame: una vez ejecutado uno de los dos algoritmos de graph cuts, se usa este método para reinsertar la imagen modificada en el vídeo.

- SaveThreaded: Esta clase permite la creación de un hilo por cada imagen que se quiere procesar usando la librearía thread de posix.

- ImageProcessing: Esta clase edita una imagen o, que es lo mismo, un frame. Permite aplicar directamente sobre ella los dos algoritmos de graph cuts, y cambiar el fondo por otro. La imagen es un array de caracteres.

  Métodos más importantes:

  - setArbitraryLabeling: asigna a cada píxel una etiqueta arbitraria.
  - getCurrentEnergy: calcula la energía actual que tiene la imagen.
  - getEnergySmoothValue: calcula el término .
  - getEnergyDataValue: calcula el término .
  - getPixelIntensity: devuelve la intensidad de un píxel.
  - swapLabels: intercambia las etiquetas de los píxeles afectados si la ejecución del graph cuts ha producido un decremento de energía, es decir ha mejorado.
  - createAndExecGraphCutSwap: crea el graph cuts para el algoritmo swap y lo ejecuta.
  - createAndExecGraphCutExpansion: crea el graph cuts para el algoritmo expansion y lo ejecuta.

- Graph: Esta clase permite la creación de un grafo y la ejecución del corte sobre el mismo.

  Métodos más importantes:

  - add_node: crea un nodo en el grafo.
  - add_edge: crea una arista entre nodos que no sean terminales.
  - add_tweights: crea una arista entre un nodo terminal y uno no terminal.

Librerías utilizadas

Unos de los objetivos que nos hemos impuesto es conseguir que el programa se pueda ejecutar en múltiples plataformas. Por ello, hemos intentado utilizar librerías portables. A continuación se explicarán y listarán brevemente.

- SDL (Simple DirectMedia Layer): es una librería multi-plataforma desarrollada en el lenguaje de programación C que proporciona funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, y carga y gestión de imágenes.

  Fueron desarrolladas inicialmente por Sam Lantinga, un desarrollador de videojuegos para la plataforma GNU/Linux. Para utilizar las librerías se han utilizado los tutoriales en [SDLTU].
- OpenGL (Open Graphics Library): es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

  Fue desarrollada originalmente por Silicon Graphics Inc. (SGI) en 1992. Aprendimos a usarla en la asignatura de Informática Gráfica.
- FFmpeg: es un conjunto de librerías de software libre multi-plataforma desarrolladas en el lenguaje de programación C que permiten cargar y guardar cualquier tipo de vídeos (teniendo los respectivos codecs instalados).

Fueron desarrolladas inicialmente por Fabrice Bellard, hemos aprendido a usarlas gracias a los tutoriales de [FFMTU].

- Graph cuts: es una librería gratuita para fines no comerciales que permite la creación de graph cuts. Desarrollada por Yuri Boykov y Vladimir Kolmogorov [YBVK].
- Posix Thread: es una librearía estándar Posix para crear programación multi-threading.

Organización del trabajo

En esta sección se presenta la organización del trabajo, utilizando una herramienta CASE (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering, Ingeniería del Software Asistida por Computador*) como ScrumDo, que ayuda a dividir el trabajo en diferentes iteraciones llamadas Sprints (con una fecha de inicio y otra de fin). Cada sprint incluye varias "historias" que representan las diferentes tareas por efectuar durante la realización del proyecto.

A continuación presentamos las tareas de cada sprint, incluyendo las correspondientes capturas de la herramienta ScrumDo.

➢ Sprint1

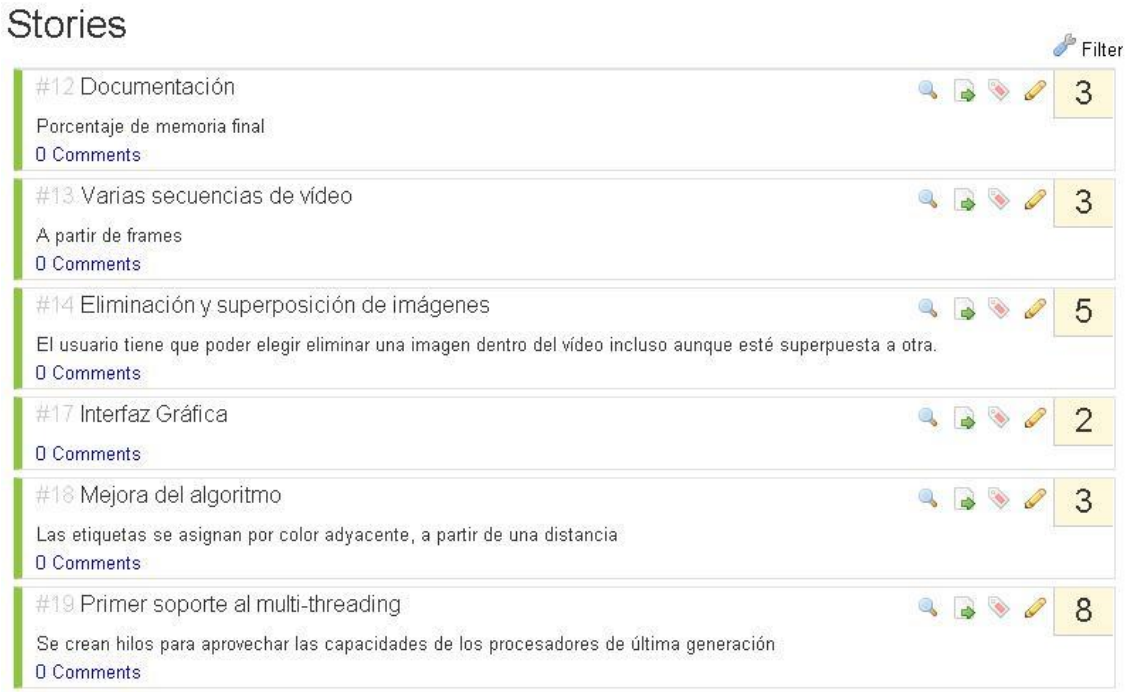
- Cargador de imágenes jpg y bmp
- Cargador de vídeos
- Algoritmo Foto-Grafo
- Implementación Graph Cut
- Aplicación de etiquetas
- Algoritmo alpha-beta swap
- Algoritmo alpha expansión

➢ Sprint2

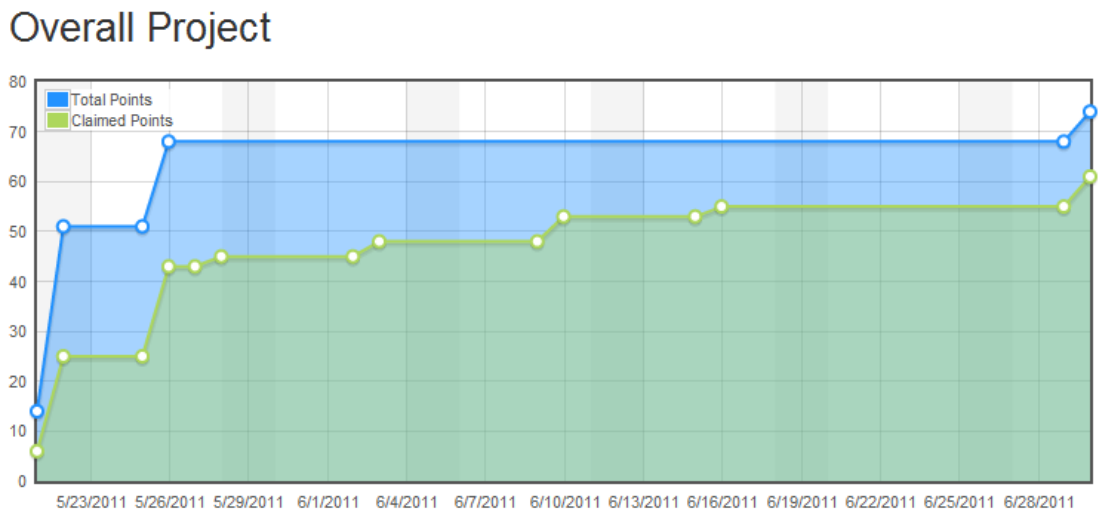
- Soporte múltiples etiquetas
- Selección asistida
- Sustitución imagen del background
- Búsqueda de vídeos
- Implementación Graph Cut

➢ Sprint3

- Mejora del algoritmo
- Interfaz Gráfica
- Eliminación y superposición de imágenes
- Primer soporte al multi-threading
- Varias secuencias de vídeo
- Documentación



➢ Progreso general del proyecto



Como se puede observar en los diferentes Sprints, cada una de las historias tiene asignado un estado (elaborándose, hecho, en revisión, o sin asignar estado) que se irá modificando a lo largo de los días y con respecto al trabajo realizado por los componentes del equipo.

Otro dato significativo son los llamados puntos de historia, que aparecen representados en una pila dentro del Sprint, y etiquetas con números. Un punto de historia corresponde a un día de trabajo ideal de una persona: donde el integrante del grupo es completamente efectivo y trabaja sin distracciones.

Esta metodología se corresponde con las conocidas metodologías Agile, que proporcionan una forma de estructurar el trabajo de forma rápida para proyectos de poca envergadura y favorecen el trabajo en grupo y la finalización del proyecto en las fechas indicadas en los Sprints.

Capítulo IV

Aplicaciones y Resultados

Introducción

En ésta sección detallamos la funcionalidad de nuestra aplicación, que como hemos comentado está basada en la minimización de energía mediante graph cuts. Además mostramos algunos ejemplos gráficos que hemos obtenido aplicando las técnicas mencionadas.

Aplicaciones

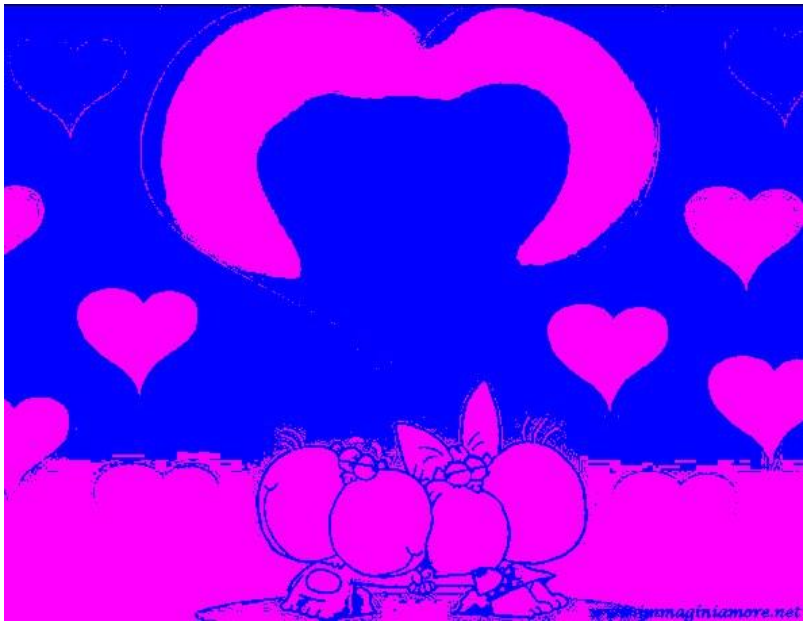
Se ofrece al usuario la posibilidad de etiquetado múltiple, es decir, es posible usar un número de etiquetas entre 2 y 9, siendo 2 el valor por defecto.

Una vez elegido este número se da la opción de usar los dos algoritmos diferentes para recortar la misma foto, a saber, alpha-expansion o alpha-

beta swap. Por otra parte, permitimos la selección asistida que se explicará más adelante.

FOTO ORIGINAL:

Ejemplo con 2 etiquetas:



Ejemplo con 5 etiquetas:

Ejemplo con 9 etiquetas:

Asistencia al etiquetado

Una característica destacable de nuestra aplicación es la ***asistencia al etiquetado*** por medio de la selección manual por parte del usuario. Esto facilita el reconocimiento de determinados objetos dentro de la imagen. Por ello, se puede decir que la aplicación es interactiva a la hora de elegir los píxeles que queremos agrupar.

Los algoritmos descritos anteriormente, basan el recorte de la imagen en la intensidad de los píxeles, por tanto, en su color RGB. Inicialmente, cuando se procedía a la selección de un grupo de píxeles, todos los píxeles de la imagen con el mismo color pasaban a formar parte de la selección. Esto causaba efectos no deseados en el recorte, por ejemplo, píxeles con el mismo color eran asociados al segundo plano. Por ello, hemos aplicado la siguiente mejora: aunque los píxeles tengan el mismo color, si la distancia que los separa es lo suficientemente amplía, no se incluyen ambos en la selección, sino que sólo el que está seleccionado.

Una vez hecho el recorte de la imagen, habilitamos el cambio del background por otro definido con anterioridad. Este cambio puede llevarse a cabo de dos maneras:

- Instantánea: Cambia de golpe la imagen de fondo, por tanto, el usuario no ve dicho cambio, sino que observa el resultado final.
- Progresiva: Cambia la imagen paulatinamente, por tanto, el usuario va viendo el proceso.

Mejoras aplicadas al programa:

Resultados

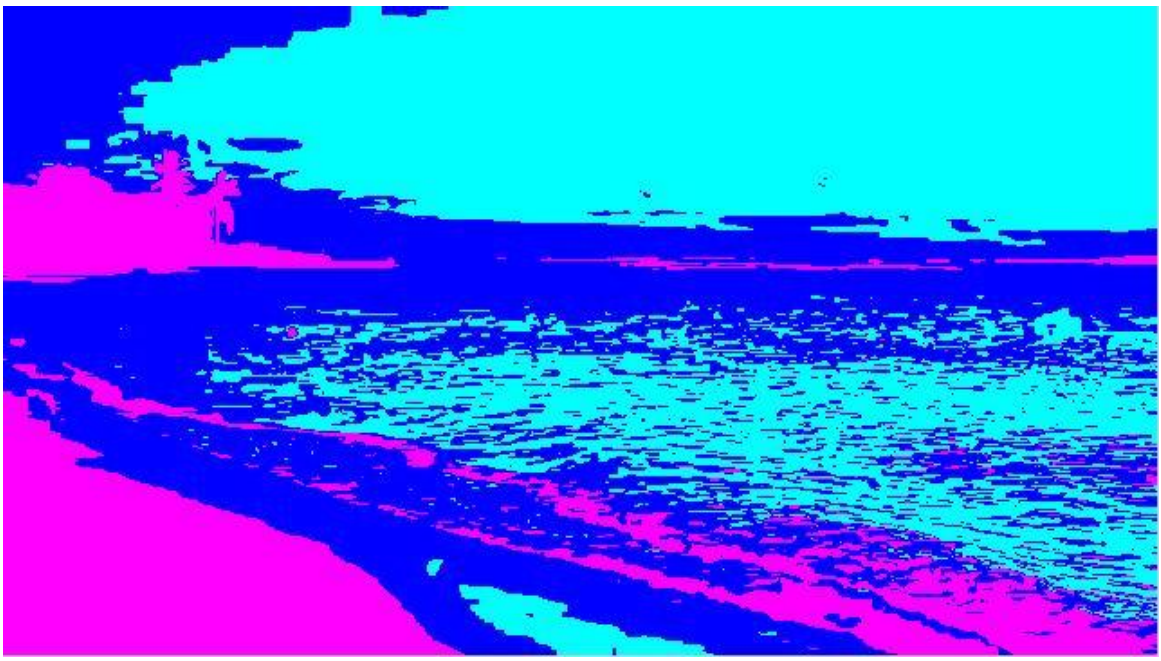
A Continuación se muestras algunas imágenes y los resultados aplicando los algoritmos presentados anteriormente.

Todas las pruebas se han realizado con éxito sobre máquinas multiprocesador con sistemas operativos Linux, Windows XP y Windows7.

Imagen número uno:

Aquí presentamos la imagen original:

Aquí presentamos la imagen usando el algoritmo expansion:



En esta imagen se han usado 3 etiquetas para que se puedan notar las diferencias en la imagen, podemos ver la arena, con etiqueta rosa, el cielo y el mar, con etiquetas de color azul. Las etiquetas del cielo y del mar se confunden debido a la semejanza entre las intensidades de sus píxeles.

Aquí presentamos la imagen 1 con expansion usando assisted labeling:

Con el *assisted labeling* se fuerza a que algunos píxeles pertenezcan a una etiqueta específica. Al asistir a la aplicación indicando dónde se encuentra el fondo (etiqueta rosa), se ve que separa el cielo del resto, es decir de la arena y el agua, así podemos separar con éxito los tres elementos de la imagen.

Imagen número dos:

Aquí presentamos la imagen original

Aquí presentamos la imagen usando el algoritmo swap con 2 etiquetas:

Se puede ver claramente que se separa el background del foreground, debido a la gran diferencia de intensidades entre los píxeles, a

pesar de usar solamente dos etiquetas. Una vez elegido el background, nuestra aplicación ofrece la posibilidad de cambiarlo por otra imagen preseleccionada.

Aquí se presenta la imagen con el fondo reemplazado por otro:

Teniendo una imagen preseleccionada para reemplazar el fondo, simplemente se ha cambiado el background (que tiene etiqueta azul en la imagen anterior) por la nueva imagen.

También es posible cambiar la imagen que está en primer plano (que tiene etiqueta rosa en la imagen anterior) y reemplazarla por otra. Éste es el resultado obtenido:

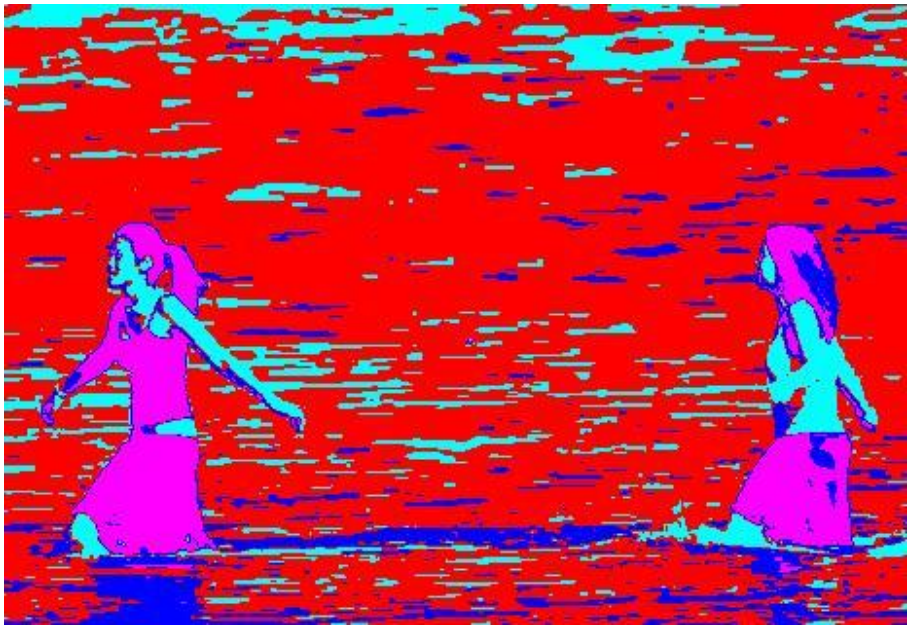
Imagen número tres:

Aquí presentamos la imagen original

Aquí presentamos la imagen con expansion usando 2 etiquetas

Los resultados no son los esperados debido a que hay más diferencia entre el color de la piel y el color de las indumentarias, que entre el color de la piel y el del mar.

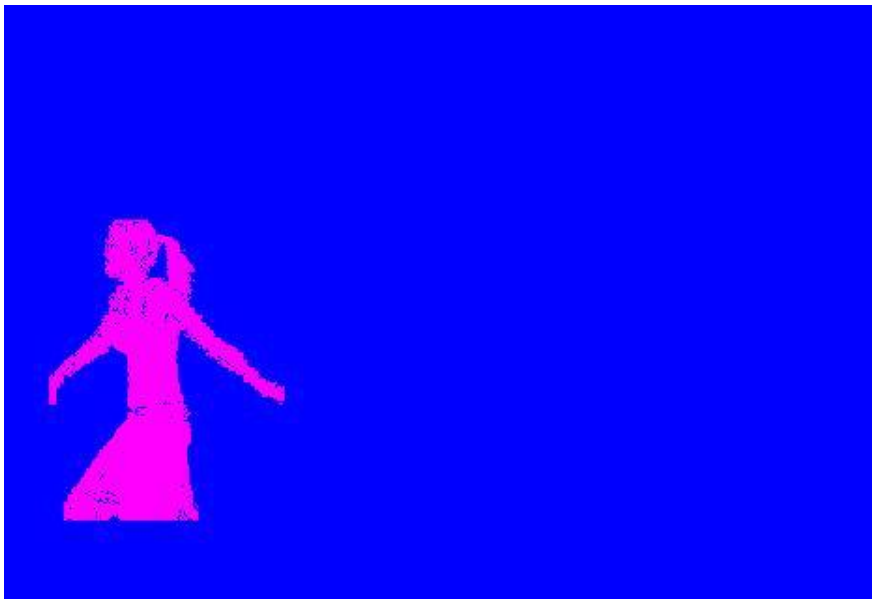
Aquí presentamos la imagen con 4 etiquetas:



Con más de dos etiquetas el algoritmo proporciona mejores resultados.

Aquí presentamos la imagen con 2 etiquetas y usando *assisted labeling*.

La finalidad de ésta prueba en concreto es poder eliminar automáticamente a una de las chicas de la figura, forzando a que la chica de la derecha forme parte del fondo.



Aquí podemos apreciar un efecto más elaborado. Después de reconocer la chica de la izquierda como parte del parecido foreground, sustituimos todo el foreground por un trozo del mar del background. Así hemos hecho desaparecer la chica completamente. No obstante, debido a la semejanza de intensidades, se necesitan técnicas más precisas para eliminar el ruido que aparece.

Vídeos

Se han hecho pruebas con secuencias de imágenes en las cuáles, para cada fotograma, se aplica uno de los algoritmos presentados.

Hemos obtenido resultados bastante satisfactorios, aun que mejorables. Los resultados son mejores cuando los fotogramas de la secuencia no cambian demasiado.

Uno de los inconvenientes es el tiempo de ejecución para procesar cada imagen de la secuencia. En realidad no podemos obtener resultados en tiempo real. Otro inconveniente, aunque no sea parte de nuestro proyecto, es el no poder utilizar la aplicación sobre todo tipo de vídeos, dado que los algoritmos no realizan seguimiento de objetos, por tanto, si en el vídeo se llevan a cabo demasiados movimientos de plano, los resultados no serán satisfactorios. Una posible mejora sería entonces la inclusión de *tracking* de objetos.

Por obvios motivos no se han incluido ejemplos de vídeos en esta memoria. No obstante, se pueden visualizar en la dirección http://www.imageprocessing.zobyhost.com

Hay que tener en cuenta que el procesamiento de todas las pruebas se ha realizado sobre un procesador Intel Core™ i3 (2 núcleos físicos y 2 núcleos lógicos) a 2,13GHz, y que cada foto tarda en procesarse aproximadamente entre 4 y 6 segundos.

Conclusiones

Minimización de energía mediante graph cuts, es sin duda una técnica muy compleja y muy interesante de estudiar, investigar y aplicar.

Se puede ver que teniendo una base matemática sólida, altos conocimientos de programación y mucho interés por la informática gráfica se pueden lograr técnicas de desarrollo para el procesamiento de fotografías muy interesantes, como el eliminado de un elemento, cambiar el fondo, diferenciar background de foreground; básicamente, efectos visuales que con más investigación se pueden llegar a realizar tareas muy complejas en tiempos aceptables, como se pudo ver en el capítulo anterior.

Hoy en día hay técnicas que realizan estas tareas con menor complejidad pero nos ha llamado la atención el hecho de que los métodos mencionados a lo largo de este trabajo (swap y expansion) tienen un alto índice de código que se puede paralelizar, esto significa que haciendo uso de tarjetas gráficas modernas podemos realizar las mismas tareas con en tiempos mucho más cortos.

Trabajos futuros

Los algoritmos presentados son una mejora en cuanto a optimalidad temporal respecto a otros métodos pero no alcanzan los niveles necesarios para una ejecución fluida en tiempo real (considerando vídeos).

Una mejora sería explotar las ventajas de las tarjetas gráficas (GPUs) frente a las CPUs de propósito general utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un altísimo número de hilos simultáneos. Es posible implementar el Graph Cuts aprovechando las tarjetas gráficas para mejorar su tiempo de ejecución.

Bibliografía

[FAEMGC] Yuri BoyKov, Olga Veksler and RaminZabih. Fast Aproximate Energy Minimization via Graph Cuts.

[SDLT] Tutorial SDL: http://lazyfoo.net/SDL_tutorials/index.php

[FFMTU] http://www.ffmpeg.org.
Tutorial FFmpeg: http://dranger.com/ffmpeg/tutorial01.html
OpenGL and ffmpeg:
http://www.gamedev.net/community/forums/topic.asp?topic_id=552132
http://stackoverflow.com/questions/3527584/ffmpeg-jpeg-file-to-avframe

[CORMEN] Thomas Cormen, Charles E. Leiserson, Ronald L.Rivest. Introduction to Algorithm, 2nd Edition, pp 669-680

[YBVK] Yuri Boykov y Vladimir Kolmogorov. Librería para graph cuts.
http://www.cs.adastral.ucl.ac.uk/~vnk/software.html

[EMFA] David P. Williamson, June Andrews. Efficiente max flow algorithms, Lecture 5 Septembre 18, 2007.

[PRA] Nilay Vaish. Push Relabel Algorithm.
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=maxflowPushRelabel

[BLIB] English version translated by Andreas Masur. The Boost C++Libraries, Charapter 8, Maximum Flow.