David Berga - 146716
Oriol Ferrer - 122145
Sergi Cabanyes - 122185

Práctica 1. Rapid texture image inpainting

Introducción

En esta práctica, hemos formulado un programa que a partir de texturas o imágenes, repara los espacios que hayamos asignado a una máscara. Para ello hemos utilizado el lenguaje c++ usando librerías de ImageMagick para utilizar la lectura, escritura y manipulación de las imágenes.

Los patrones de reparación de los píxeles asignados por la máscara son el cálculo de semejanza entre los vecinos según ventanas de 3x3 (las diferencias de color) al cuadrado y la distancia entre los pixeles asignados.

Cómo funciona el programa?

Algorismo (función principal)

-Previamente, en el main ya guardaremos una matriz de pixeles donde leerá la imagen de máscara considerando los píxeles blancos como agujeros (valor 1) y el fondo negro (valor 0).

-Empezamos el algorismo, con dos for recorriendo la matriz de mascara, si el valor es 1 (agujero) recorremos toda la matriz en busca del pixel correcto a copiar

-Para eso, recorremos la matriz otra vez con dos for, considerando que el valor sea 0 (no agujero) y para cada uno guardamos la semejanza para ROJO,VERDE y AZUL (RGB)

-Sumamos estas semejanzas y las multiplicamos por la distancia que tiene respecto la coordenada del pixel de mascara cogido, utilizando sqrt((x1-x2)^2+(y1-y2)^2)

-Una vez tengamos ese valor de semejanza considerando la distancia, miramos si ese valor es menor al mejorvalor de semejanza (cuanto menor sea, mejor)

-Entonces si ha dado que es el mejor valor, guarda esa coordenada en mejorsemejaX y mejorsemejaY esperando que acabe de recorrer la matriz

-Una vez acabado el recorrido de pixels normales lo que hace es copiar el pixel que se ha dado de mejor semejanza (que habiamos guardado sus coordenadas), despues marcamos que esa coordenada ya la hemos pintado en la matriz de mascara, asignamos el valor inicial de mejorsemejanza y el for de busqueda de pixels de mascara sigue su recorrido

Funcion de cálculo de semejanza de píxeles

-La metodologia es sencilla. Primero guardamos los vecinos del pixel de mascara y del pixel normal.

-Entonces, según cómo sea dado el píxel de máscara, ejecutaremos una cosa u otra. Es decir, si es la esquina superior izquierda, que compare sólo los vecinos que sean de fuera de la máscara, es decir, que valgan 0 en la matriz de pixeles, que corresponden a los vecinos v1,v2,v3,v4,v7. Y así para cada caso. Véase el kernel usado (3x3)

v1	v2	v3
v4		v6
v7	v8	v9

-Para comparar estos valores, lo que hemos hecho es restar el valor de cada vecino del pixel de mascara al pixel normal (elevándolo todo al cuadrado) y finalmente sumar cada caso. Entonces la función devolverá éste valor de suma. Cuando menor sea ése valor, mejor semejanza habrá entre el píxel de máscara y el píxel normal encontrado.

Definición de variables usadas

En el algorismo

float distancia : es la distancia entre un pixel de mascara y un pixel encontrado
int mejorsemejaX, mejorsemejaY : coordenadas donde guardaremos el pixel que mejor se asemeja
float semejROJO, semejAZUL, semejVERDE : es la semejanza de cada color entre dos pixeles
float semej : es la semejanza total entre dos pixeles (suma de semejanzas por distancia)
float smejorCOLOR : es el menor valor de semejanza encontrado en el recorrido total de la matriz
Imagen uR, uG, uB : es la matriz de píxeles leída (que contriene una estructura de matriz de float)
Imagen uTR, uTG, uTB : es la matriz de píxeles de salida
int matriupixels[][] : es la matriz de píxeles de máscara (valen 1 o 0)
Imagen maskFLOT : es la matriz de máscara, pero en floats, que aseñalaremos para decir que está rellena, porque es la que introducimos en la funcion de semejanza
int i, j : corresponde al recorrido de coordenadas donde buscamos el pixel de mascara
int x, y : corresponde al recorrido de coordenadas donde buscamos el pixel a copiar

En la funcion de cálculo de semejanza de pixeles

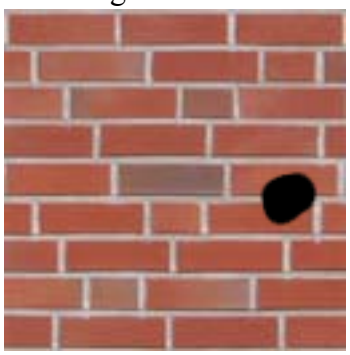
int maskposX, maskposY : coordenadas del pixel de máscara encontrado
int x, int y : coordenadas del pixel encontrado que queremos comparar
Imagen COLOR : es la imagen de color concreto (matriz de floats) de entrada
Imagen MASCARA : es la imagen de máscara (unos y ceros en floats) actual
float semej : es el valor final que guardaremos (cuanto menor sea, mas se asemeja el pixel)
float colorkernel[3][3] : corresponde a la matriz de ventana donde pondremos los vecinos del x e y
float v1,v2,v3,v4,v5,v6,v7,v8,v9 : son los valores de los vecinos de la mascara (valen 1 o 0)
float unR, dosR, tresR, quatreR, sisR, setR, vuitR, nouR : son los valores de diferencia de color entre cada vecino del pixel de mascara y el pixel normal

Experimentos

Hemos probado diferentes tipos de imágenes y a veces nos variaba según la foto haya sido con mucho o poco colorido, pero con el programa actual que tenemos, creemos que trabaja bastante bien con las texturas. Hemos ido probando con diferentes tamaños al dibujar la máscara.

Ejemplo1:

Imagen de entrada

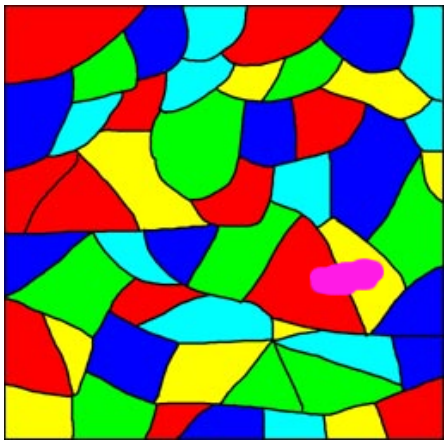


Máscara

Imagen restaurada

Ejemplo2:
(aquí sacaremos más o menos el pegote morado, usando una máscara con pegote blanco)

Imagen de entrada



Máscara

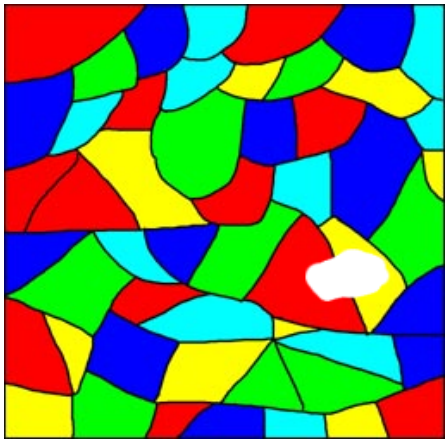
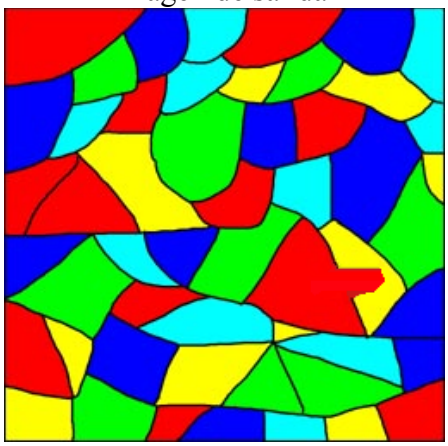
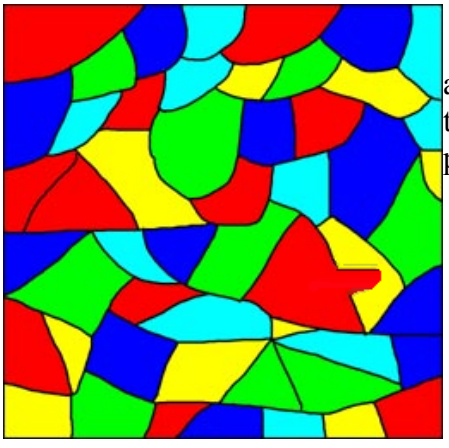


Imagen de salida



(y entonces hemos vuelto a aplicar el programa a la imagen de salida para mejorarla un poco más)

Imagen de salida2



Como podemos apreciar, a esa continuación del mosaico de color rojo le aplica la línea negra que sigue su camino. El resultado final no es totalmente perfecto, pero para el caso se tendrían que considerar patrones diversos, o bien, aplicar la máscara con otra forma.

Estructuración de los ficheros y ejecución

El programa usa básicamente 3 archivos.
-El “Makefile.make” dentro de src, que lo que hace es compilar el programa
-El “mainprueba.cpp” donde hemos programado todo el programa
-El “Imagen.cpp” donde hay definida la clase Imagen que usaremos para “mainpureba.cpp”

Para compilar y ejecutar el programa, hay que:
1. Borrar lo que haya dentro de la carpeta obj (archivos de ejecucion antiguos)
2. Ir a la carpeta src y compilar el makefile con el comando “make -k prueba”
3. Ir a obj y ejecutar el archivo de ejecución de prueba introduciendo sus parámetros con el comando “./prueba imagen_entrada imagen_mascara lista_secciones”

Una manera más sencilla es que hemos creado nuestro propio documento “sh” de compilación y ejecución directa que ya nos remueve el fichero de dentro de obj, después compila y despues ejecuta con los parámetros preterminados, donde:
-La imagen_entrada es AT_poster.png
-La imagen_mascara es mascara.png
-La lista es lista.txt (fichero no usado)

Conclusiones

Hemos tenido algunos problemas en lo que se refiere a que en casa no nos funcionava correctamente el ImageMagick por temas de que no encontraba librerías una vez instalado, o bien no encontraba funciones definidas... etc. Aunque una vez hemos entendido la idea de usar un doble recorrido sobre una imagen pixel a pixel, hemos entendido como hacer el programa. También hemos tenido algunos problemas en que no podíamos crear funciones con argumentos concretos (por ejemplo la matriz de enteros de la máscara , no la podíamos usar en la función de semejanza porque daba error, por lo que hemos usado un parámetro de entrada de Imagen de máscara).