Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

Proyecto de Laboratorio 2022 - Paradigma Funcional

Paradigmas de Programación

Lucas Mesias Soza
Sección: 13310-0-A-1
Profesor: Roberto Gonzalez Ibañez

Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

Índice

Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

1. Introducción

Un programa de edición de imágenes permite modificar o alterar imágenes existentes y crear los elementos visuales con los que interactuamos día a día, como una simple fotografía con filtros simples o ajustes de color, afiches, eliminación de fondos, o directamente el programa se usa como herramienta de dibujo digital, programas como Adobe Photoshop, GIMP, Photopea, entre otros, son las herramientas con las que se trabaja para crear este mundo digital al que estamos acostumbrados.

2. Problema

Se requiere crear un software de manipulación de imágenes simplificado en el lenguaje Scheme, bajo el paradigma funcional, es decir, con operaciones básicas, interactuando con el usuario a través de la consola de comandos, usando las funciones nativas del lenguaje scheme o racket, y trabajar teniendo en cuenta las diferentes características del paradigma funcional, entre ellas la incapacidad de usar variables o control de flujo, para esto, se trabaja de forma declarativa, se hace uso extendido de la recursión, funciones envoltorio, currificación, etc.

Los pixeles serán la base del trabajo de manipulación, estos deben ser representados en un TDA que permita guardar la siguiente información: posición, valor de color y profundidad. Las imágenes se deben representar en un TDA adecuado, que permita trabajar pixel a pixel, además habrán 3 tipos distintos de pixeles, diferenciandose en el valor de color que almacenan, estos serán bit (0 o 1), RGB y hexadecimal, además entre las operaciones que debe tener el software, debe ser capaz de comprimir y descomprimir imágenes, por lo que la implementación de la imágen debe soportar estas características.

3. Solución

Los TDAs implementados para los pixeles fueron estructurados en un TDA principal llamado “pixel”, el cual almacena el tipo de pixel, las coordenadas X, Y, el valor de color, y la profundidad en una lista de scheme. Luego, se crearon 3 sub-TDAs, uno por cada tipo de valor de color, “pixbit” para almacenar el número que representa el bit, “pixrgb” para almacenar 3 números enteros con los valores (R G B) y “pixhex” para almacenar el valor RGB en formato hexadecimal “#RRGGBB”, cada uno de estos es un pixel que guarda información diferente, de esta forma, se pueden implementar funciones básicas para pixeles genéricos, simplificando la implementación de funciones que no dependen del tipo de pixel.

El TDA imagen guarda un identificador, las dimensiones de la imágen, un valor de compresión y una lista de pixeles, los cuales no necesitan estar en un orden particular, las funciones implementadas no dependen del orden de esta lista, pero todos los pixeles de la imagen deben estar presentes o habrá inconsistencias a la hora de la descompresión.

Para las funciones que identifican el tipo de imagen se revisa primero si fueron comprimidas, si es así, el valor de compresión nos permite inferir el tipo de imagen, si la imagen no está comprimida, se revisa que cada pixel de la lista almacenada sea del mismo tipo.
Histograma: Fue implementada una función recursiva de cola, la cual arma el retorno en cada recursión, cuenta las veces que un color se repite en una imagen, lo agrega a la lista de retorno y borra el color de la siguiente recursión. Para implementar de manera simple para el usuario, fue necesaria una función envoltorio que se encarga de entregar los parámetros iniciales necesarios para la recursión.

Compresión: Usando la información entregada por Histograma, se elimina de la imagen los pixeles con el color más común usando un filtro, luego se guarda en el valor de compresión el color eliminado.

Descompresión: Al ingresar una imagen comprimida, se rellenan los pixeles que faltan en la imagen con pixeles nuevos, con el valor de compresión. Cabe destacar que con esta implementación se pierde totalmente la información de profundidad en los pixeles comprimidos.

Transformación de imagen a string: Se recorre la imagen de salida en orden, buscando el pixel que va en cada posición y se transforma su valor de color a string, debido a esto, el orden en el que los pixeles están en la lista no importa.

Capas de Profundidad: Primero se extrae de la imagen todas las profundidades de los pixeles sin repetición y se guardan, luego se genera una imagen por cada profundidad, y con un filtro se guardan sólo los pixeles que pertenecen a dicha profundidad.

Recorrer la imagen: Para implementar la descompresión y transformación de imagen a string, fue necesario recorrer una imagen pixel a pixel, para esto se crearon las funciones *findPix* que encuentra un pixel dadas unas coordenadas X e Y, y *recorrerImg*, una función recursiva de cola que simula un ciclo *for* anidado, de esta forma permite aplicar funciones al área de la imágen coordenada a coordenada.

El proyecto se organizó en 3 archivos, otras_funciones.rkt contiene funciones genéricas no relacionadas con los TDAs, pixel.rkt e image.rkt, que incluyen las respectivas funciones propias de los TDA. No se usó ninguna librería externa, y se usó DrRacket para correr el programa.

4. Instrucciones de uso

Se debe tener el programa principal en la misma carpeta que los archivos del proyecto (image.rkt, pixel.rkt y otras_funciones.rkt) y añadir las siguientes líneas de código al inicio del mismo:

- *(require "pixel.rkt")*
- *(require "image.rkt")*

Luego se recomienda definir las imágenes con las que se va a trabajar frecuentemente:

- (define img1 (image …))

Una vez escritas las operaciones deseadas se ejecuta el código con el botón “run” de la interfaz de DrRacket o presionando CTRL+R, los resultados aparecerán en la consola de comandos y adicionalmente podrá realizar operaciones directamente en la consola. Si algún error surge al momento de ejecutar es recomendable revisar si el tipo de imagen/pixel que se está ingresando coincide con el tipo de operación.

Uso	Resultado
(image 2 2 (pixrgb-d 0 0 60 1 1 10) (pixrgb-d 0 1 2 80 2 20) (pixrgb-d 1 0 5 3 60 10) (pixrgb-d 1 1 145 90 123 1))	'("image" 2 2 -1 (("pixrgb-d" 0 0 (60 1 1) 10) ("pixrgb-d" 0 1 (2 80 2) 20) ("pixrgb-d" 1 0 (5 3 60) 10) ("pixrgb-d" 1 1 (145 90 123) 1)))
(display (image->string imgextra1 pixhex->string))	image.rkt:251:87: string-append: contract violation expected: string? given: '(60 1 1)
(display (image->string imgextra1 pixrgb->string))	#3C0101 #05033C #025002 #915A7B

5. Resultados

Se lograron implementar todas las funciones tanto obligatorias como opcionales, todas funcionan si la entrada es correcta, aunque se pudo haber hecho un esfuerzo mayor y revisar que las entradas sean correctas antes de procesar, este aspecto no fue una prioridad en el desarrollo del proyecto. Adicionalmente es posible la optimización del proyecto en varios aspectos, memoria, cambiando el identificador de tipo de pixel de string a un entero y representar los tipos en base a una enumeración, cambiar el uso de *append* por *cons*, entre otras mejoras.

6. Autoevaluación

La autoevaluación del proyecto con respecto a las funciones implementadas es de puntaje completo, las únicas pruebas donde el código falla es cuando la entrada es incorrecta, por lo tanto, las funciones cumplen si son usadas correctamente.

7. Conclusiones

El trabajo realizado en el lenguaje funcional scheme fue difícil de adoptar en un inicio, ya que es muy diferente a C o python, no por la sintaxis, sino por el cambio radical en la forma en la que uno afronta el problema planteado y trata de generar una solución, una preocupación adicional fue la restricción no especificada formalmente de algunas funciones como for o for/list, restricción que fue notificada verbalmente cuando se le preguntó al profesor en horario de clases específicamente por el uso de la misma, esto no forma parte del enunciado de laboratorio y genera mayores dudas sobre qué funciones se pueden o no usar.

8. Anexo

Puntajes Autoevaluación

Requerimientos funcionales	Autoevaluación
TDAs	1
image	1
bitmap?	1
pixmap?	1
hexmap?	1
compressed?	1
flipH	1
flipV	1
crop	1
imgRGB->imgHex	1
histogram	1
rotate90	1
compress	1
edit	1
invertColorBit	1
invertColorRGB	1
adjustChannel	1
image->string	1
depthLayers	1
decompress	1