UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Paradigmas de la programación

Informe de laboratorio N°1

Software de edición o manipulación de imágenes

Andres Zelaya

Profesor Gonzalo Martínez

26 de septiembre 2022

El presente informe corresponde al laboratorio N°1 del curso; donde en cada experiencia se trabajará utilizado un paradigma de la programación distinto.

Siendo utilizado en esta ocasión el Paradigma Funcional, donde es ocupado el lenguaje de programación Scheme a través del compilador Dr. Racket.

En el informe sigue se cubrirán los siguientes tópicos:

- Descripción breve del problema
- Introducción al paradigma
- Objetivos del proyecto
- Análisis del problema,
- Diseño de solución del problema
- Aspectos de implementación
- Instrucciones y ejemplos de uso
- Resultados y autoevaluación
- Conclusión

Descripción del problema

El proyecto de este semestre es la creación de un software de edición y/o manipulación de imágenes digitales, es decir, que permita a un usuario realizar distintas operaciones sobre éstas. Ejemplos de este tipo de software son GIMP y Adobe Photoshop.

Existen distintos tipos de operaciones tales como: rotar una imagen, invertirla, rotarla, retocarla, transformarla y redimensionarla, entre otras.

A diferencia de los softwares antes señalados, este proyecto se concentrará en trabajar en imágenes RGBD o RGB-D, esto es, imágenes que además de tener información en el espacio de colores (R)ed, (G)reen, (B)lue, contiene información de la profundidad (D)epth en un espacio tridimensional.

De esta forma, en la imagen bidimensional de la Figura 1 es posible distinguir los colores en el espectro RGB. Al incorporar la dimensión D (profundidad) capturada a través de una cámara especializada, sería posible saber más sobre los detalles del rostro, proyección de la nariz, sombrero, distancia del espejo en la parte posterior, etc. Incluso sería posible construir una representación tridimensional del rostro, como se ilustra en la Figura 2.

Descripción del paradigma

El Paradigma Funcional, tal como su nombre lo dice, tiene como unidad de abstracción y programación básica a las funciones.

Una función corresponde a una transformación de elementos, donde al igual que en las funciones matemáticas, se tiene un dominio y un recorrido, siendo el primero los elementos de entrada y que serán transformados por el proceso de la función, y el segundo respectivamente es el elemento de salida una vez se ha realizado la transformación.

En el Paradigma Funcional se tienen algunos elementos bastante importantes a la hora de construir un código utilizando este paradigma, como, por ejemplo:

Funciones Anónimas: Provienen directamente del Cálculo Lambda, donde estas funciones se expresan sin nombre, bajo una entrada y una transformación de esta que se aplica a la función.

Composición de funciones: Consiste en una especie de operación donde dos funciones generan una tercera función, algo bastante útil cuando se tienen funciones que se complementan entre sí.

Recursividad: Es fundamental en este paradigma, pues permite realizar una gran cantidad de operaciones y procesos donde este se van utilizando soluciones pequeñas del problema.

Objetivos

Como objetivos del laboratorio se tiene aprender sobre el Paradigma y la programación funcional, para así obtener la habilidad de programar de otra forma distinta a la que se tiene costumbre actualmente.

Otro objetivo es programar correctamente en Scheme y aprender a utilizar las herramientas de Dr. Racket para completar el proyecto de laboratorio y así poder tener una base para los futuros laboratorios y otros proyectos que en un futuro se desarrollen con la Programación Funcional.

Análisis del problema

Primero que todo, debido a la forma de trabajarse el paradigma y al no existir el uso de variables, para que una transformación de una imagen sea “almacenada” se debe definir, en caso contrario, solo se mostrara una salida de esta por consola sin ser “permanente”.

Básicamente todas las funciones tienen como dominio una imagen y parámetros que varían dependiendo de la operación que se realice, las únicas excepciones a esto son los constructores de las imágenes, las cuales reciben las dimensiones, el tipo de imagen y los pixeles que la componen, ya sean tipo bit, RGB o hexadecimales.

Para poder trabajar correctamente una imagen, es decir, aplicarle funciones esta debe ser previamente definida o incluirse como una función interior de la aplicación deseada.

También es necesario implementar distintos tipos de datos abstractos (TDA) para cada tipo de píxel que compone una imagen, pero esta última se puede trabajar como conjunto.

Diseño de solución

Para diseñar la solución, no solo es necesario utilizar los tipos de datos nativos de Scheme, sino que también se deben crear tipos de datos específicos, a través de lo que se conoce como TDA, utilizando la siguiente estructura:

- Representación
- Constructores
- Funciones de Pertenencia
- Selectores
- Modificadores
- Otras funciones asociadas al TDA.

Para la correcta implementación y desarrollo de este laboratorio, se emplean distintos TDAs que logran realizar de forma correcta las distintas operaciones requeridas para esta instancia de laboratorio. Estos fueron:

- TDA pixbit-d
- TDA pixrgb-d
- TDA pixhex-d
- TDA image

Vale mencionar que como la mayoría de las funciones requeridas son para una imagen, estas estarán en el TDA imagen.

Primero hay que hablar de pixeles, ya que son la unidad menor de una imagen, cada uno tendrá:

- **Tipo:** Puede variar dependiendo de si es tipo bit, RGB o hexadecimal

- **Posición:** Ya que la representación de imagen utilizada es una matriz con pixeles, cada uno debería tener una posición X e Y, que será representada mediante 2 valores que hacen el rol de posición X e Y respectivamente.

- **Contenido:** El contenido de cada píxel puede variar dependiendo de su tipo, si es bit tendrá solo un valor que podría ser 0 o 1, si es tipo RGB tendrá tres valores que oscilarán entre el 0 y 255, y finalmente si es tipo hexadecimal tendrá un respectivo código indicado un color en este formato.

- **Profundidad:** Como se menciona anteriormente, esta implementación contara con posibilidad de manejo de imágenes en 3D, por lo que este argumento está pensado para representar la profundidad en cada píxel de una imagen.

Aspectos de implementación

Para este proyecto es necesario el compilador Dr. Racket, específicamente de versión 6.11 o superior Dr. Racket tiene bastantes herramientas útiles, como por ejemplo debug.

Se pueden utilizar todo tipo de funciones de Scheme y Racket para la elaboración de los TDA, todo esto mientras se respete el paradigma funcional, por ejemplo, no se puede simular el uso de variables.

Para todos los TDAs, se hace uso de “(provide (all-defined-out))”, lo cual permite que se puedan utilizar todas las funciones creadas dentro de ese archivo en otro archivo, sin necesidad de estar importando funciones una tras una, mientras que para el archivo main, se hace uso de todos los TDAs creados importándolos a través de REQUIRE.

Instrucciones de uso

Primero, se debe verificar que se tengan los cinco archivos en una misma carpeta, ya que, de lo contrario, el archivo main no se podrá ejecutar si falta un TDA. Luego de eso, se puede compilar, ejecutando el programa a través de la opción “Run”.

Una vez ejecutado, en el archivo main se tienen varios ejemplos para cada una de las operaciones, sin embargo, se pueden realizar todas las operaciones disponibles de forma individualmente siempre que se entreguen los parámetros correctos para esta.

Se espera que todas las operaciones en el main.rkt funcionen sin problemas

Resultados esperados

Se espera poder trabajar exitosamente con imágenes de distintos tipos, donde la implementación del programa sea completamente funcional y sin errores.

Todo esto logrado mediante un buen uso de listas y recursiones, que son fundamentales para la programación funcional.

Finalmente se espera una implementación correcta de los TDA.

Posibles errores

El error más probable es que al ejecutar el archivo de pruebas no se cuenten con todos los archivos .rkt en la misma carpeta, en este caso, verificar su existencia.

Resultados obtenidos

Los resultados obtenidos fueron los esperados, ya que se logró crear las funciones obligatorias y la gran mayoría de las funciones opcionales. El programa es completamente funcional.

Se hicieron múltiples pruebas con distintos ejemplos para probar de que no hubiera fallos en la ejecución del código y que el código hiciera lo correcto, lográndose la implementación de 13/20 funciones.

Autoevaluación

Las funciones que si fueron implementadas funcionan en la totalidad de las veces probadas.

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces.

Se adjunta una taba de autoevaluación individual por funciones como anexo en la tabla 1.

Conclusiones

Luego de realizar y casi completar el proyecto, se puede concluir que se cumplieron los objetivos principales, ya que fue posible aprender a utilizar correctamente Scheme y Racket para poder completar el proyecto de laboratorio correspondiente.

Lo más difícil de esta experiencia fue poder implementar correctamente las distintas recursiones utilizadas para las funciones creadas, ya que muy pocas veces cumplían con su propósito al primer intento.

Por otro lado, no se experimentaron complicaciones por el lado de la instalación o uso del compilador ni por el aspecto del versionamiento con Git.

Bibliografía y referencias

1. Gonzales, R. (2022). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. https://docs.google.com/document/d/1D6g3S3vLC-zziOsSprLIBPypkd89s2QEkJs1F_kbHm4/edit?pli=1
2. Gonzales, R (2022) “Laboratorio 1 (Paradigma funcional – Scheme)” Paradigmas de la programación. https://docs.google.com/document/d/1hUAooKwBj3TWv-yuzBZtNbuaC8iNkzOZdbLpD8P9B8c/edit?pli=1#heading=h.w59fjctai3ln
3. Chacón, S. y Straub, B. (2020). “Pro Git – Todo lo que necesitas saber sobre Git’’. Libro Online. Recuperado de: https://drive.google.com/file/d/1mHJsfvGCYcIhdrmK-lBI6a1WS-U1AAPi/view
4. Flatt, M. y Bruce, R. (2022). “The Racket Guide’’. The Racket Reference. Documentación Online. Recuperado de: https://docs.racket-lang.org/guide

Anexos

Figura 1. Lenna, una de las imágenes más empleadas (desde 1973) como imágenes estándar de prueba de algoritmos de compresión
Fuentes:
https://taglang.io/blog/post/Understanding_Kernel_Convolution_Part_2/

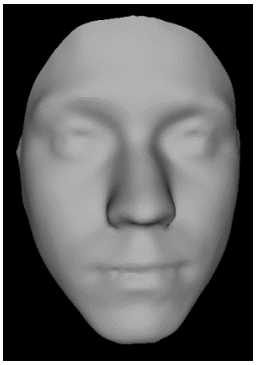

Figura 2. Reconstrucción sintética de un rostro a partir de una imagen RGBD.
Fuente:
https://www.researchgate.net/publication/316442804_Facial_depth_map_enhancement_via_neighbor_embedding/figures?lo=1

Figura 3.

Estructura de la implementación – Creación propia

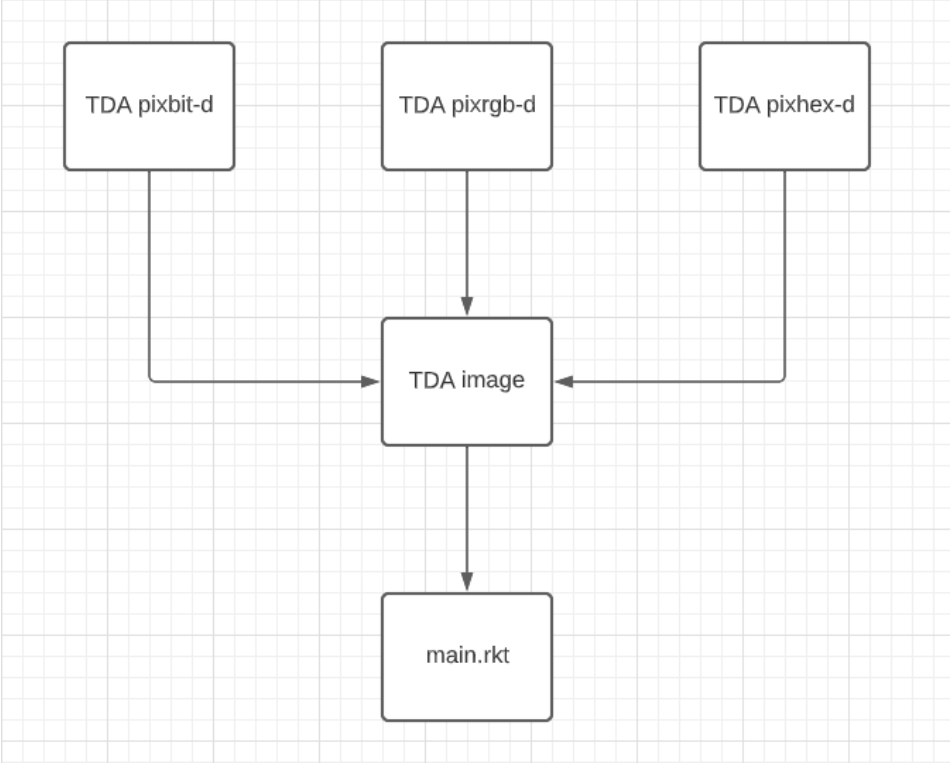

Tabla 1.

Autoevaluación de requerimientos funcionales.

TDAs	1
TDA image - constructor	1
TDA image - bitmap?	1
TDA image - pixmap?	1
TDA image - hexmap?	1
TDA image - compressed?	1
TDA image - flipH	1
TDA image - flipV	1
TDA image - crop	1
TDA image - imgRGB->imgHex	1
TDA image - histogram	1
TDA image - rotate90	1
TDA image - compress	1
TDA image - edit	0
TDA image - invertColorBit	0
TDA image - invertColorRGB	0
TDA image - adjustChannel	0
TDA image - image->string	0
TDA image - depthLayers	0
TDA image - decompress	0