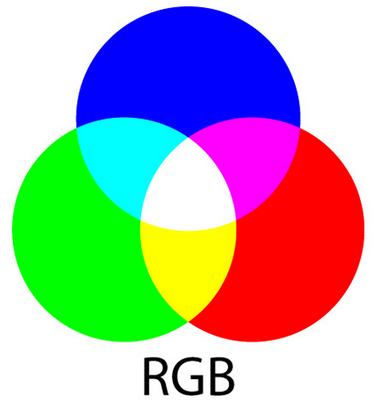
|  |  |
| --- | --- |
|  | j0319484.wmf PROYECTO DE CLASE (GIT)Desarrollo en C para la conversión de imágenes RGB a HSV y HSV a RGBedson mojicamanuel villamizarGABRIEL PEDRAZADocente |

**DEFINICIÓN DEL MODELO RGB Y HSV**

RGB es un modelo de que se representa mediante la mezcla por adición de los tres colores de luz primarios rojo, verde y azul.

**Figura** 1 **Modelo RGB**



**HSV es un modelo**

El modelo de color HSV es una transformación no lineal del modelo RGB en coordenadas cilíndricas de manera que cada color viene definido por los siguientes elementos:

Tinte o matiz, Saturación, Brillo

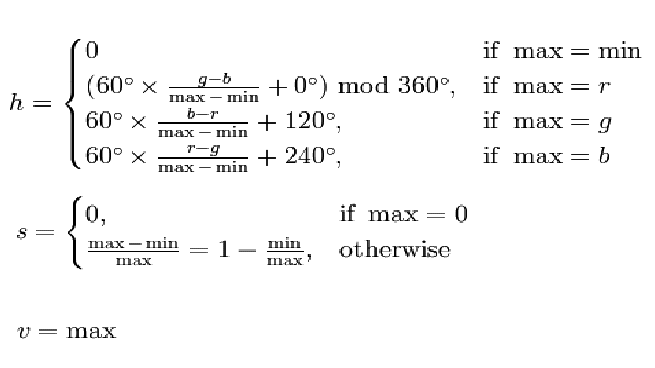
**Figura** 2**. Modelo HSV**



Tomada de .Wikipedia

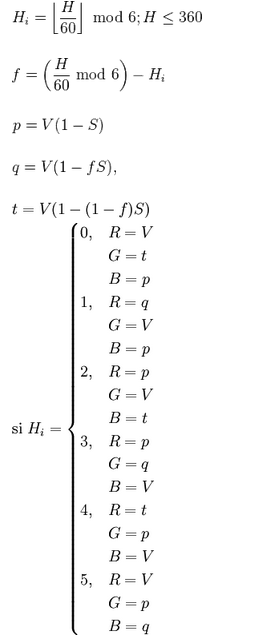
**FORMULACIÓN MATEMATICA PARA LA CONVERSION DE RGB TO HSV**

**Figura** 3**. Conversión HSV to RGB**



**FORMULACIÓN MATEMATICA PARA LA CONVERSION DE HSV TO RGB**

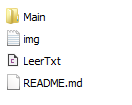
**Figura** 4 **Conversión RGB to HSV**



**DESARROLLO DEL PROYECTO**

**Estructura de carpetas**

**Figura** 5**. Estructura carpeta del proyecto**



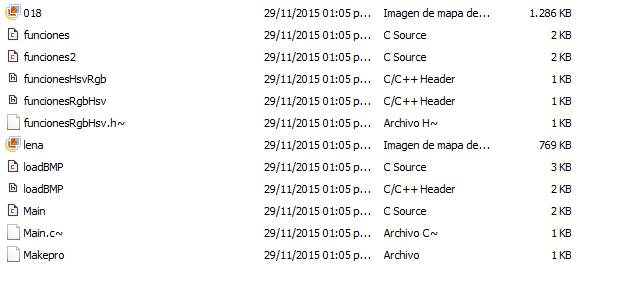
**Readme:** contiene la información e instrucciones del proyecto.

**LeerTxt:** Archivo inicial que lee un txt de representa los datos de uno de los tres componentes RGB y lo maneja en matrices.

**Img:** archivo txt con los datos de una de las componentes de RGB.

**Main:** Contiene los archivos con el desarrollo del proyecto, se detallan a continuación.

**Figura** 6**. Estructura de Carpetas Principal**



Imágenes utilizadas para desarrollar el proyecto:

**018**: imagen utilizada para las pruebas. “Imagen acuática”

**lena**: imagen utilizada para las pruebas inicialies. “imagen de lena”

Archivos

**funcionesHsvRgb**: contiene los prototipos del nombre de las funciones que realizan la conversión de RGB a HSV.

**funcionesRgbHsv**: contiene los prototipos de las funciones que realizan la conversión de HSV a RGB.

**loadBMP**: clase utilizada para cargar la imagen.

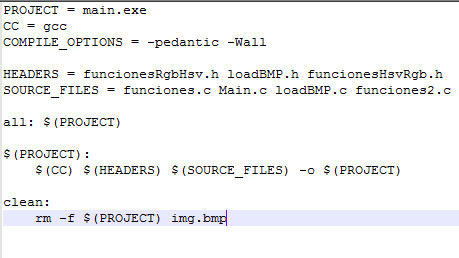
**Main.Archivo**: principal del proyecto.

**funciones**.c: archivos con el cuerpo de las funciones para la conversión de RGB a HSV.

**funciones2**.c: archivos con el cuerpo de las funciones para la conversión de HSV a RGB.

**MakePro**: archivo que al ser ejecutado realiza la compilación del proyecto o borra lo generado dependiendo de la acción que se seleccione.

**Figura** 7**. Archivo Make**



Al ejecutar el archivo Makepro se compila y crea el archivo ejecutable main.exe:

**Figura** 8**. Archivo generado al compilar**



Luego al ejecutar main.exe se genera la imagen en modelo HSV como se ve en la siguiente figura.

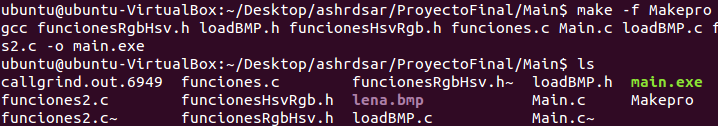
Figura 9. Figura HSV generada



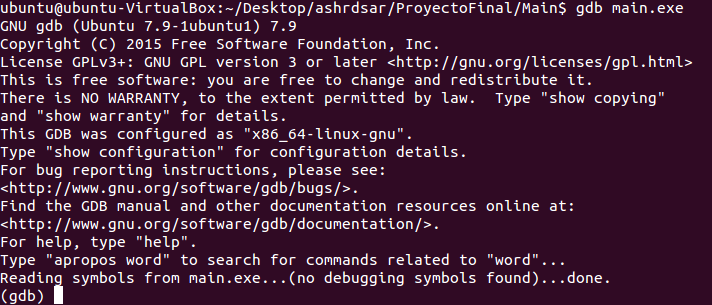
**UTILIZANDO GDB**

El utilizar GDB nos permitió detectar errores de asignación en las variables que no tenían el tipo correcto después de efectuada una operación, de esta forma logramos avanzar en la definición de los tipos de variables para todo el proyecto, inicialmente se manejó solamente uint8\_t pero para lograr un correcto funcionamiento fue necesario manejar float en conjunto con las variables uint8\_t.

**Figura** 10**. Ejecutando el archivo Make**

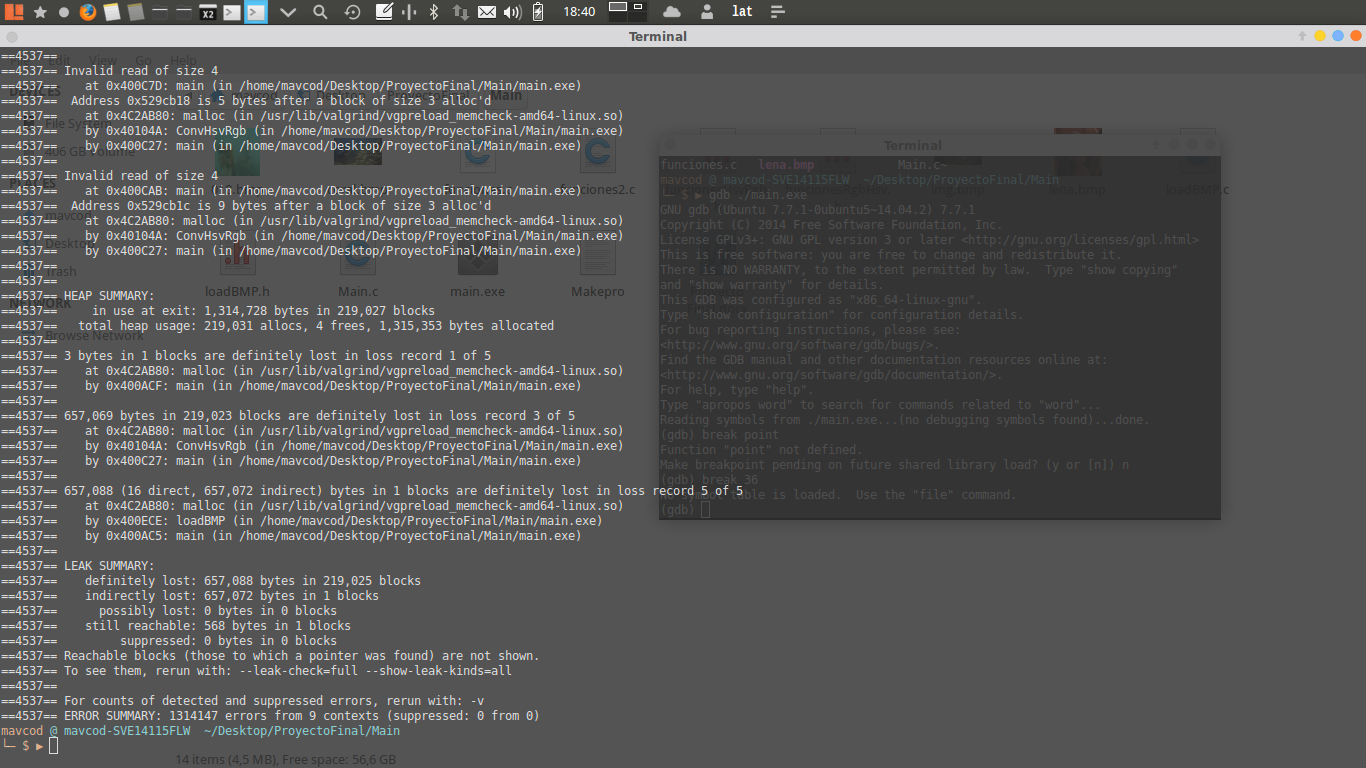


**Figura** 11**. Utilizando el GDB**



**UTILIZANDO VALGRIND**

Al utilizar el analizador de memoria en el proyecto se detectaron escrituras invalidas, las cuales no generan mal funcionamiento en los algoritmos pero si permitieron identificarlas y cambiarlas; los problemas relacionados con el manejo de memoria se observan en la siguiente imagen.



En los item relacionados al problema de memoria, que se pueden resaltar se observa el invalid read of size 4 y bytes perdidos en los bloques, el summary se detallan las perdidas definitivas, indirectas y posibles de bytes en la ejecución del proyecto

Definitely lost: 657,088 bytes in 219025

indirectly lost: 657072 bytes in 1 block

possibly lost: 0 bytes en 0 blocks.

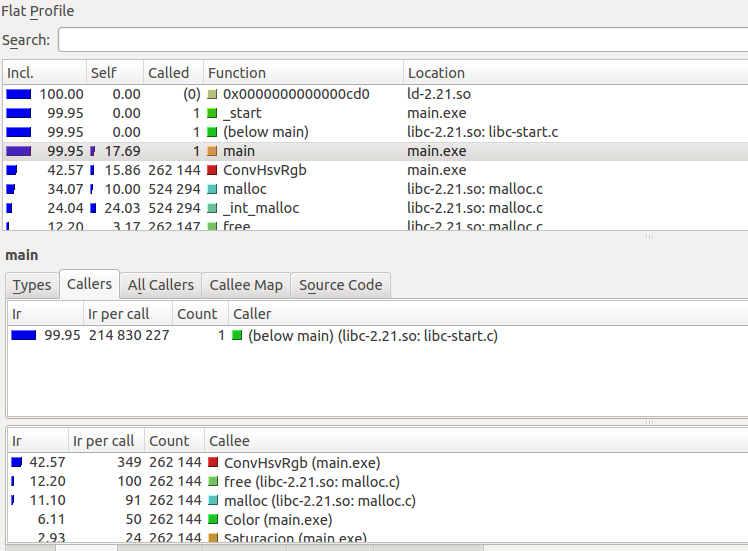
Still reachable: 568 bytes in 1 block

Los problemas de memoria se generaron al escribir en consola después de compilar el siguiente comando valgrind --tool=memcheck ./main.exe, estos problemas se analizaran detalladamente para ejecutar el proyecto sin problemas de memoria.

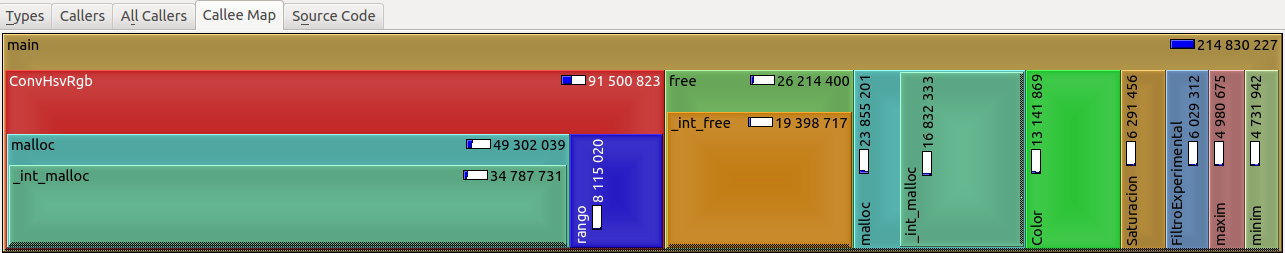
**UTILIZANDO KCACHEGRIND**

Después de ejecutar vallgrind con el visualizador KcacheGrind se logró identificar cada elemento y las líneas de códigos necesarias para su ejecución; por ejemplo el main y ConvHsvRgb; realizando un análisis de cada uno de los gráficos que brinda Kcachegrind se observan los flujos de llamados y datos sobre la ejecución de una forma mas clara.

**Figura** 12**.Diagrama KcacheGrind 1**



**Figura** 13**.Diagrama KcacheGrind 2**



**Figura** 14**. Diagrama KcacheGrind 3**

