|  |  |
| --- | --- |
|  | 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.

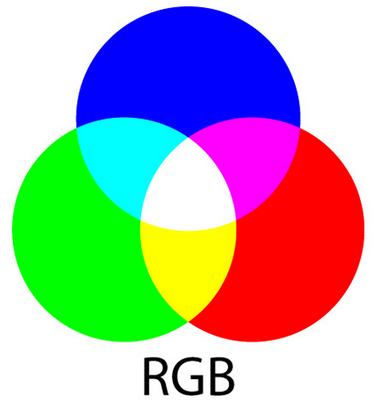


Figura 1 Modelo RGB

El modelo HSV, define un modelo de color en términos de sus componentes. De esta forma se puede realizar la transformación para conocer los valores básicos RGB en el dominio HSV:

|  |
| --- |
| H = \begin{cases} \mbox{no definido}, & \mbox{si } MAX = MIN \\ 60^\circ \times \frac{G - B}{MAX - MIN} + 0^\circ, & \mbox{si } MAX = R \\ &\mbox{y } G \ge B \\ 60^\circ \times \frac{G - B}{MAX - MIN} + 360^\circ, & \mbox{si } MAX = R \\ &\mbox{y } G < B \\ 60^\circ \times \frac{B - R}{MAX - MIN} + 120^\circ, & \mbox{si } MAX = G \\ 60^\circ \times \frac{R - G}{MAX - MIN} + 240^\circ, & \mbox{si } MAX = B \end{cases} |
| S =  \begin{cases} 0, & \mbox{si } MAX = 0 \\ 1 - \frac {MIN} {MAX}, & \mbox{en otro caso} \end{cases} |
| V = MAX \, |
| Figura 2. Relaciones de modelo RGB a HSV |

Donde disponemos de 360 grados dónde se dividen los 3 colores RGB, eso da un total de 120º por color, sabiendo esto podemos recordar que el 0 es rojo RGB(1, 0, 0), 120 es verde RGB(0, 1, 0) y 240 es azul RGB(0, 0, 1). Para colores mixtos se utilizan los grados intermedios, el amarillo, RGB(1, 1, 0) está entre rojo y verde, por lo tanto 60º. Se puede observar cómo se sigue la secuencia de sumar 60 grados y añadir un 1 o quitar el anterior. Para el color blanco se puede poner cualquier color y saturación, siempre que se establezca el valor (de luminosidad) máximo. Asimismo, para el color negro se puede poner cualquier color y saturación, siempre que se ponga un valor de 0.

|  |
| --- |
| cono |
| Tomada de .Wikipedia |
| Figura 3. Modelo HSV |

Para retornar a los valores de canales de color RGB se utilizaran las relaciones:

|  |
| --- |
| H_i = \left\lfloor \frac { H }{ 60 }  \right\rfloor  \mbox{ mod } 6; H \le 360 |
| f = \left( \frac { H }{ 60 } \mbox{ mod }6 \right)  - H_i |
| p = V (1 - S) |
| q = V (1 - f S), \mbox{ } |
| t = V (1 - (1 - f) S) |
| \mbox{si }H_i = \begin{cases} 0, &R = V \\ &G = t \\ &B = p \\ 1, &R = q \\ &G = V \\ &B = p \\ 2, &R = p \\ &G = V \\ &B = t \\ 3, &R = p \\ &G = q \\ &B = V \\ 4, &R = t \\ &G = p \\ &B = V \\ 5, &R = V \\ &G = p \\ &B = q \\ \end{cases} |
| Figura 4 Conversión HSV a RGB |

Como se puede observar el canal V del dominio HSV representa la intensidad lumínica en la escena, y S la saturación de color en alguno de los puntos, al hacer la diferencia entre y se pretende representar la escena basados en la absorción de luz de cada uno de sus elementos, con este valor se eleva al cuadrado para obtener un valor positivo menor que el valor , de esta forma los valores de color menos su nivel de absorción, da una matriz que reduce los pixeles donde se da gran intensidad de luz. Al hacer la diferencia entre la unidad y los valores normalizados de estas relaciones, se tiene que se destacan los valores con tonos similares a los de iluminación (rojo y amarillo) y se reducen tonos azules. El filtro mencionado tiene la forma

El presente proyecto pretende pasar a código C el preprocesamiento de imágenes de nadadores utilizado en Matlab, desarrollado dentro de las actividades del grupo Hdsp, en algoritmos de segmentación para nadadores. Por este motivo se incluyen 3 tipos de imágenes, “018.bmp”, “coral.bmp” y “lena.bmp”, dos de ellas bajo el agua y una de ellas la imagen de prueba “lena”, aclarando no es de disposición publica o para divulgación científica, dado que aún no tiene las publicaciones de rigurosidad que apoye su validez, por lo que las pruebas son netamente académicas-experimentales, para dar validez al mismo.

DESARROLLO DEL PROYECTO

En primera instancia se sigue una organización de módulos que puede ser aproximada de la siguiente forma:

|  |
| --- |
| D:\Users\edsf5\Desktop\cosita\clase\segundo semestre\MULTIDISCIPLINAR 2\doc1.jpg |
| Figura 5 Conversión HSV a RGB |

Para cargar las imágenes *\*.BMP* , se utilizó una librería disponible en el repositorio <https://github.com/jake92m/loadBMP>, posterior a la carga de la imagen, el archivo Main, para la lectura de la imagen utiliza un bucle que llama las funciones de conversión *RGB a HSV*, calcula el *Filtro* y retorna los valores filtrados *HSV a RGB.*

Para utilizar este filtro, se tienen los siguientes archivos:

|  |
| --- |
|  |
| Figura 6. Estructura carpeta del proyecto |

Readme: contiene la información que expone las instrucciones y página principal del proyecto en https://github.com/mavcod/ProyectoFinal.

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

|  |
| --- |
|  |
| Figura 7. Estructura de Carpetas Principal |

Imágenes utilizadas para desarrollar el proyecto:

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

“coral.bmp” : imagen utilizada para las pruebas. “Imagen acuática”

“lena.bmp”: imagen utilizada para las pruebas iniciales. “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.c : 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 Make para la ejecución del proyecto.

Al ejecutar el archivo Makepro se pueden realizar diferentes acciones, estos comandos permiten la ejecución del proyecto de diferentes maneras:

* ../make –f Makepro “compila el proyecto para generar ejecutable” con el ejecutable main.exe se puede generar la transformación de la imagen.

|  |
| --- |
|  |
| Figura 8. Archivo generado al compilar |

* ../make –f Makepro clean “elimina archivos generados” los archivos que elimina al ejecutar el “clean” son el ejecutable y la imagen de salida img.bmp.
* ../make –f Makepro gdbmake “ejecuta debugging” ejecuta el archivo main.exe y el debugger.
* ../make –f Makepro Valval “valgrind con las opciones de memcheck” ejecuta el archive main.exe y el análisis de fuga de memoria.
* ../make –f Makepro calcal “ejecuta el valgrind con opciones callgrind” se utiliza para generar el archivo callgrind, posterior a esto se utiliza el comando .../kcachegrind “callgrind name file” para analizar con kcachegrind.

main.exe: archivo ejecutable generado mediante el make.

Para ejecutar el proyecto, compilar con make y ejecutar el archivo main.exe, posteriormente se solicitará el nombre de la imagen a procesar. Para este caso las imágenes disponibles dentro de la carpeta del proyecto deben ser llamadas escribiendo su nombre y extensión, que para el caso todas son **.*BMP*** finalmente el procesamiento arrojará como resultado la creación de una imagen con el nombre ***img.bmp****.*

Durante las diversas pruebas del desarrollo del proyecto, se generó una imagen test en el domino HSV. Para llegar al resultado final, una muestra de los resultados obtenidos durante el procesamiento es la transformación a la imagen lena.bmp, quien en el dominio HSV puede no ser fácilmente apreciable, pero al retornar al dominio RGB se nota claramente el cambio en tonos brillantes y la atenuación sobre la piel, así como el cambio en la tonalidad de los azules que es la banda de interés a afectar en las fotos acuáticas donde el fondo se ve fusionado con el objeto.

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.

|  |
| --- |
| D:\Users\edsf5\Desktop\cosita\ProyectoFinal\Main\lena.bmp |
|  |
| D:\Users\edsf5\Desktop\cosita\ProyectoFinal\imagenes para informe\img4.bmp |
| Figura 9. Imagen HSV generada |

Inicialmente se manejaron variables uint8\_t pero para divisiones y demás operaciones matemáticas fue necesario utilizar otros tipos de variables, manejar float en conjunto con las variables uint8\_t se dio como necesidad observado durante el debugging.

|  |
| --- |
|  |
| Figura 10. Ejecutando el archivo Make |

|  |
| --- |
|  |
| Figura 11. Utilizando el GDB |

|  |
| --- |
| Screenshot from 2015-11-29 21:43:20.png |
| Figura 12. 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 se investigó el porqué de esos warning y se identificó en las variables de la librería de lectura y escritura de los archivos bmp, encontrando que:

|  |
| --- |
| Screenshot from 2015-11-29 21:52:34.png |
| Screenshot from 2015-11-29 21:52:34.png |
| Figura 13. Utilizando el Valgrind |

Se tenían variables que utilizaban malloc pero no se estaba liberando la memoria. Para este caso se agregaron dos comandos extra a la librería, un *“free(image);”* en la línea 81, *“*free(img\_data);” en la línea 80 y un “fclose(fh);” para reducir los problemas. Desafortunadamente se incrementaron los errores. Por lo que se quitó el *“*free(img\_data);”

|  |
| --- |
|  |
| D:\Users\edsf5\Desktop\maldito gato\aeliminar\Screenshot from 2015-11-30 10-39-43.png |
| Figura 14. Depurando con el Valgrind |

UTILIZANDO KCACHEGRIND

Finalmente se ejecuta el make con la opción “calcal” para generar el archivo de entrada al kcachegrind el cual revela que la distribución de instrucciones para el proyecto, los flujos de llamados y datos sobre la ejecución de una forma más clara.

|  |
| --- |
| Screenshot from 2015-11-30 22:31:46.png |
| Figura 15.Diagrama KcacheGrind 1 |

O con el mapa de llamados donde el 33% se divide en 3 acciones básicas, color filtro saturación detección de mínimo y máximo lo cual compone un módulo de operación y alrededor del otro 33% lo tienen las instrucciones para pasar de HSV a RGB, el resto de operaciones están asignadas a las tareas de recorrido en el main.

|  |
| --- |
|  |
| Figura 16.Diagrama KcacheGrind 2 |

Finalmente la distribución, sus asignaciones de llamados y relaciones se pueden apreciar mejor en un árbol de llamados, para esto se disponen los principales módulos en un despliegue tipo árbol, con la cantidad de llamadas, o procedimientos involucrados a cada uno de estos módulos.

|  |
| --- |
|  |
| Figura 17. Diagrama KcacheGrind 3 y detalles |

CONCLUSIONES

Con la implementación del algoritmo propuesto se logra identificar que al pasar una imagen de rgb a hsv y aplicar un filtro se pueden cambiar ciertas características sin afectar en la totalidad el color relacionado a los puntos de interés en una imagen.

En las imágenes con fondo azul, se tiende a resaltar los objetos que se encuentran en tonos distintos al azul.  Esta apreciación es de gran aplicabilidad a imágenes acuáticas, donde se tiene exceso de tonalidades azules. A continuación se muestran imágenes que evidencia el proceso anteriormente descrito.

|  |
| --- |
|  |
| Figura 18.Imagen rgb |
|  |
| Figura 19.Imagen hsv |

|  |
| --- |
|  |
| Figura 20.Imagen en escala de grises |
|  |
| Figura 21.Diagrama rgb |

De las imágenes expuestas se puede observar el objeto de una forma más clara, como resultado del filtro, el cual resalta y permite la separación de los elementos con tonalidades de color distintas al azul en la escena.

Las imágenes resultantes permiten observar el objeto de una forma más clara, esto se debe a que el filtro resalta o separa los elementos diferentes al azul; la imagen final en RGB muestra el fondo anteriormente en azul con un color rojo y las figuras o demás objetos completamente separados por tonalidades similares a la imagen original.

La figura en escala de grises presenta el objeto con mayor intensidad de iluminación respecto al fondo de la imagen saturada en tonos azules, pero con el filtro se hace de una mejor forma la distinción objeto fondo. estas apreciaciones son resultado del análisis de las imágenes finales para los diferentes ejemplos utilizados.

Usualmente al crear algoritmos para darle solución a problemas es común que al conseguir el objetivo se dé por finalizado, pero en este trabajo final se busca no solo resolver el problema sino hacerlo de forma eficiente, haciendo uso de las herramientas existentes para analizar mejor los algoritmos en busca de realizar mejores prácticas.

Utilizando gdb, vallgrind y el visualizador se identificó de una forma clara los errores de asignación y de fuga de memoria, algunos de ellos aportados por las librerías utilizadas en la lectura y escritura de la imagen, después de realizar los cambios en el manejo de memoria, declaración y asignación de variables, al mitigar en lo posible los errores mencionados, se dio por finalizada la tarea.