Práctica 3: Affine Transformation

Visión por Computadora

Elaborado por:
José Baltazar Ramírez Rodríguez

Dra María Susana Ávila García

19 de febrero del 2019

I. DESCRIPCIÓN DEL PROBLEMA

En algunas ocasiones al obtener alguna imagen a través de un dispositivo, podemos adquirirla con ciertas irregularidades. Estos pequeños defectos en la imagen pueden ser corregidos mediante operaciones geométricas/Transformaciones Afín. Ya sea que queramos obtener una perspectiva diferente de la imagen, esta puede ser rotada a cierto ángulo para poder apreciarla mejor. Que la imagen se quiera trasladar en distintas coordenadas; que una imagen quiera ser escalada para aumentar su tamaño. Todo este tipo de tratamiento de imágenes se realizan mediante operaciones geométricas/Transformaciones Afín.
En estos ejercicios, se trabaja con una imagen (matriz) binaria. Sobre ella se van a aplicar las operaciones de traslación, escalar, rotación.

II. ALGORITMO UTILIZADO PARA RESOLVER EL PROBLEMA

El programa consiste en varias funciones que realizarán las operaciones sobre la matriz binaria. Para cada transformación que se desea realizar, existe una matriz con la que se estará operando.

La primera operación que se realizó es la de **traslación**. La traslación ocurre cuando se quiere desplazar la matriz cierta cantidad de unidades en x y en y. Para esta operación se requiere de un vector con las nuevas coordenadas, las coordenadas actuales y una matriz de transformación. La operación quedaría de la siguiente manera:

$$[coordenadasNuevas] = [coordenadas] \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tx & ty & 1 \end{matrix}$$

Donde tx y ty es la cantidad de desplazamientos que hará nuestra matriz. Para el vector de coordenadas nuevas se utilizó un auxiliar para acumular la suma de cada nueva coordenada. En la función main del programa, se recorre la matriz binaria hasta encontrar un uno.

```
for(i = 0; i < filas; i++)
{
    for(j = 0; j < columnas; j++)
    {
        if(matriz[i][j] == 1)
        {
            trasladar(i,j);
            escalar(i,j);
            rotar(i,j);
        }
    }
}
```

Una vez que lo encuentra, manda llamar a la función trasladar, que recibe como parámetros las coordenadas i y j de la matriz original.

La función $trasladar$ luce de la siguiente forma:

```
void trasladar(int i, int j)
{
    int auxPos = 0;
    int coordX = 0;
    int coordY = 0;
    int vectorCoordenada[3] = {i,j,1};

    int vectorCoordenadaNueva[3] = {0};


    int matrizTransformacion[3][3] = {{1, 0, 0},
                                      {0, 1, 0},
                                      {0, 2, 1}};

    int x = 0;
    int y = 0;
```

Donde $auxPos$ es un acumulador que irá realizando la suma de la multiplicación de $vectorCoordenada[i,j]$ y la $matrizTrasnformacion$. El $vectorCoordenadaNueva$[3] almacenará nuestro acumulador $auxPos$ para obtener las nuevas coordenadas de la matriz. Y por supuesto la $matrizTrasnformacion$, que, en este caso, $tx = 0$ y $ty = 2$. Además, se declaran dos variables para un ciclo anidado para realizar la multiplicación de $vectorCoordenada[i,j]$ y la $matrizTrasnformacion$.

```
    for(x = 0; x < 3; x++)
    {
        for(y = 0; y < 3; y++)
        {
            auxPos += vectorCoordenada[y] * matrizTransformacion[y][x];

        }
        vectorCoordenadaNueva[x] = auxPos;
        auxPos = 0;
    }


//Para sacar coordenadas X y Y nuevas

    coordX = vectorCoordenadaNueva[0];  //Para trasladar en filas
    coordY = vectorCoordenadaNueva[1];  //Para trasladar en columnas

    matrizTrasladada[coordX][coordY] = 1;
```

Se declara un ciclo anidado para realizar la multiplicación entre los elementos que mencionaron anteriormente y se acumula la suma en $auxPos$. Se asigna al $vectorCoordenadaNueva$[x] la suma de la multiplicación llevada a cabo para obtener la nueva coordenada. Después de obtienen la coordenada x y y para agregarse a una

$matrizTrasladada[coordX][coordY]$ con las nuevas posiciones y así, haber trasladado la matriz. Esta $matrizTrasladada[filas][columnas]$ ha sido declarada antes de la función principal, junto con los prototipos de las funciones.

Posteriormente en la función main, se manda llamar a la función $mostrarMatrizTrasladada()$ que es de tipo $void$.

```
void mostrarMatrizTrasladada()
{
    //Visualizar matriz
    for(i = 0; i < filas; i++){
        for(j = 0; j < columnas; j++){
            printf(" %d ", matrizTrasladada[i][j]);
        }
        printf("\n");
    }
}
```

La matriz binaria de $[10][10]$ que se ha utilizado para prueba con todas las funciones es la siguiente:

```
Matriz Original
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

El resultado que se obtuvo con $tx = 0$ y $ty = 2$ fue:

```
Matriz Original
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

```
Matriz Trasladada
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

Donde se puede apreciar que efectivamente, la matriz ha sido trasladada dos posiciones en $ty = 2$ y $tx = 0$. Se agregan otra prueba más donde $ty = 2$ y $tx = 2$.

```
Matriz Original
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  1  1  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  1  1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

```
Matriz Trasladada
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  1  1  1  1  0  0
 0  0  0  0  1  1  1  1  0  0
 0  0  0  0  1  1  0  0  0  0
 0  0  0  0  1  1  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
```

La segunda operación que se realizó es la de **escalar**. Este proceso ocurre cuando se realiza una multiplicación de un vector con escalares sx y sy en una matriz de transformación. Para esta operación se requiere de un vector con las nuevas coordenadas, las coordenadas actuales y una matriz de transformación. La operación quedaría de la siguiente manera:

$$[coordenadasNuevas] = [coordenadas] \begin{matrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{matrix}$$

La función escalar luce de la siguiente forma:

```
int sX = 1;
int sY = 1;
int auxPos = 0;
int coordX = 0;
int coordY = 0;
int vectorCoordenada[3] = {i,j,1};
int vectorCoordenadaNueva[3] = {0};

int matrizTransformacion[3][3] = {{sX, 0, 0},
                                  {0, sY, 0},
                                  {0, 0, 1}};

int x = 0;
int y = 0;
```

Al igual que para la función trasladar, $auxPos$ es un acumulador que irá realizando la suma de la multiplicación de $vectorCoordenada[i,j]$ y la $matrizTrasnformacion$. El $vectorCoordenadaNueva$[3] almacenará nuestro acumulador $auxPos$ para obtener las nuevas coordenadas de la matriz. Y por supuesto la $matrizTrasnformacion$, que, en este caso, $sx = 1$ y $sy = 1$. Además, se declaran dos variables para un ciclo anidado para realizar la multiplicación de $vectorCoordenada[i,j]$ y la $matrizTrasnformacion$.

```
for(x = 0; x < 3; x++)
{
    for(y = 0; y < 3; y++)
    {
        auxPos += vectorCoordenada[y] * matrizTransformacion[y][x];

    }
    vectorCoordenadaNueva[x] = auxPos;
    auxPos = 0;
}

//Para sacar coordenadas X y Y nuevas

coordX = vectorCoordenadaNueva[0];
coordY = vectorCoordenadaNueva[1];
```

Se declara un ciclo anidado para realizar la multiplicación entre los elementos que mencionaron anteriormente y se acumula la suma en $auxPos$. Se asigna al $vectorCoordenadaNueva$[x] la suma de la multiplicación llevada a cabo para obtener la

nueva coordenada. Después cuando la matriz original en la posición i, j sea = 1, se aplican algunas condiciones anidadas.

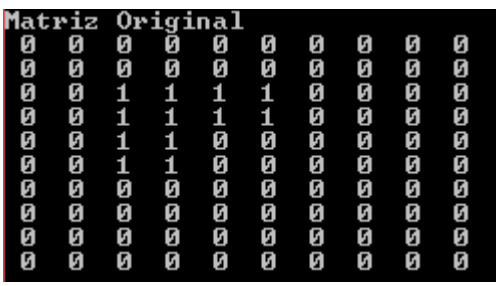
```
if(matriz[i][j] == 1)
{
    if(matriz[i][j+1] != 0)
    {
        matrizEscalada[i][j] = 1;

    }
    else if(matriz[i][j+1] == 0)
    {
        for(x = 0; x <= sX; x++)
        {
            matrizEscalada[i][j+x] = 1;
        }
    }

    if(matriz[i+1][j] != 0)
    {
        matrizEscalada[i][j] = 1;
    }
    else if(matriz[i+1][j] == 0)
    {
        for(y = 0; y <= sY; y++)
        {
            matrizEscalada[i+y][j] = 1;
        }
    }

}
```

Verifica que la matriz en la siguiente posición en columnas sea diferente de 0, o sea que no sea un 1, para agregar un 1 en la $matrizEscalada[i][j]$. Con esto garantiza que ponga los 1's en las mismas posiciones que la matriz original. Si la matriz original en la siguiente posición de columnas es igual a 0, entonces de define un bucle que vaya de 0 al sx para agregar 1's a partir de esas posiciones. De esta forma es escala en x esa cantidad de veces. Lo mismo ocurre para las filas.

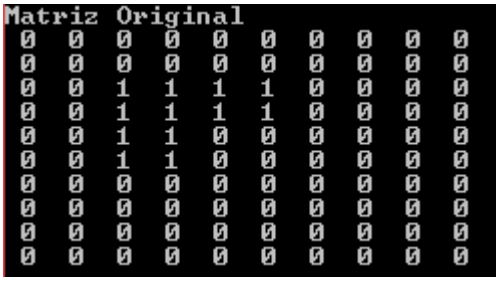
Posteriormente en la función main, se manda llamar a la función $mostrarMatrizEscalada( )$ que es de tipo $void$.

```
void mostrarMatrizEscalada()
{
    //Visualizar matriz
    for(i = 0; i < filas; i++){
        for(j = 0; j < columnas; j++){
            printf(" %d ", matrizEscalada[i][j]);
        }
        printf("\n");
    }
}
```

El resultado que se obtuvo con $sx = 1$ y $sy = 1$ fue:



Donde se puede apreciar que efectivamente, la matriz ha sido escalada en $sx = 1$ y $sy = 1$. Se agregan otra prueba más donde $sx = 3$ y $sy = 2$.



Desafortunadamente no se supo implementar para la fecha de entrega, la función de rotar y shear. Respecto a la función de rotar, se considera igual una matriz de transformación.

```
void rotar(int i, int j)
{
    double auxPos = 0;
    double coordX = 0;
    double coordY = 0;
    double vectorCoordenada[3] = {i,j,1};
    double vectorCoordenadaNueva[3] = {0};
    double q = 90;
    double val = PI / 180.0;
    int matrizTransformacion[3][3] = {{cos(q*val), sin(q*val), 0},
                                      {-sin(q*val), cos(q*val), 0},
                                      {0, 0, 1}};
```

Se utilizó un factor de conversión para convertir los radianes a grados y al igual se efectúa la operación de la matriz transformación.

```
for(x = 0; x < 3; x++)
{
    for(y = 0; y < 3; y++)
    {
        auxPos += vectorCoordenada[y] * matrizTransformacion[y][x];

    }
    vectorCoordenadaNueva[x] = auxPos;

    auxPos = 0;
}


//Para sacar coordenadas X y Y nuevas

coordX = vectorCoordenadaNueva[0];
coordY = vectorCoordenadaNueva[1];
//printf(" %d , %d \n", (int)coordX, (int)coordY);

//matrizRotada[(int)coordX][(int)coordY] = 1;
```

Se trabajará para comprender e implementar las funciones de rotar y shear, para el próximo avance de parcial poder utilizar estos algoritmos en la interfaz.