Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación

GRAFICACIÓN

TAREA 3: POLIGONO ANIMADO

Docente:
Prof. Iván Olmos Pineda

Alumno:
Jesús Huerta Aguilar

Matricula:
202041509

NRC: 10592
Sección: 001

CUARTO SEMESTRE

Puebla, Pue. Fecha de entrega: 09/09/2022

INTRODUCCIÓN

Un algoritmo preciso y efectivo para la generación de líneas de rastreo, desarrollado por Bresenham, convierte mediante rastreo las líneas al utilizar sólo cálculos incrementales con enteros que se pueden adoptar para desplegar circunferencias y otras curvas.
El algoritmo de línea de Bresenham se basa en probar el signo de un parámetro entero, cuyo valor es proporcional a la diferencia entre las separaciones de las dos posiciones de píxel de la trayectoria real de la línea.
En este trabajo se implementara el algoritmo de Bresenham para ahorrar costos computacionales al momento de imprimir una gráfica de una función polar, además de reajustar la resolución de la grafica para tener puntos pivote, los cuales serán la entrada de el algoritmo.

CONCEPTOS DESARROLLADOS

Es un algoritmo creado para dibujar rectas en los dispositivos de gráficos rasterizados, como por ejemplo un monitor de computadora, que determina que pixeles se rellenaran, en función de la inclinación del ángulo de la recta a dibujar.
Es un algoritmo preciso para la generación de líneas de rastreo que convierte mediante rastreo las líneas al utilizar solo cálculos incrementales con enteros que se pueden adaptar para desplegar circunferencias y curvas. Los ejes verticales muestran las posiciones de rastreo y los ejes horizontales identifican las columnas de píxel.

ANALISIS EMPIRICO

Reserva memoria.

En esta función reservamos dinámicamente memoria para las matrices.

```
///RESERVA MEMORIA
//Reservamos memoria dinamica, ya que asi se pasa por referencia los
elementos de la matriz
float** reservarMemoria(size_t FIL, size_t COL) {
    float **matriz = new float *[FIL];

    for (size_t f(0); f < FIL; ++f) {
        matriz[f] = new float[COL];
    }
    return matriz;
}
```

Multiplica matrices 3x3.

Función que multiplica 2 matrices de 3x3 y guarda los valores por referencia en la primera matriz de entrada, la función recibe como parámetro una matriz flotante dinámica y una matriz flotante estática.

```
//MULTIPLICA MATRICES 3X3
void multMatriz(float **A,float B[3][3]){
    float C[FIL][COL];
    for (int i = 0;i < FIL;i++){
        for (int j = 0;j < COL;j++){
            C[i][j]=0;
            for (k = 0;k < FIL;k++){
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
            }
        }
    }
    for(int i = 0; i < FIL;i++){
        for(int j = 0;j < COL;j++){
            A[i][j] = C[i][j];
        }
    }
}
```

Multiplica matrices puntos.
Función que multiplica dos matrices, la primera de 3x3 y la segunda de 3x1 y guarda los resultados en la segunda matriz de 3x1 que pasara por referencia, la función recibe como parámetro dos puntos enteros los cuales serán T' de la matriz de modelado, además también recibe dos matrices flotantes dinámicas, la primera será la matriz de modelado 3x3 y la segunda la matriz de puntos 3x1.

```
//MULTIPLICA MATRICES PUNTOS
void multMatrizPoints(int p1,int p2,float **A,float **B){
    int points[3][1] = {{p1},
                        {p2},
                        {1}};
    float C[3][1];
    for(int i = 0;i < 3;i++){
        for(int j = 0;j < 1;j++){
            C[i][j]=0;
            for (int k = 0;k < 3;k++){
                C[i][j] += A[i][k] * points[k][j];
            }
        }
    }
    for(int i = 0; i < 3;i++){
        B[i][0] = trunc(C[i][0]);
    }
}
```

Traslación.
Creamos la matriz de traslación, creando una matriz de identidad y asignando en [0][1] = -tx y [0][2] = -ty

```
//TRASLACION
void traslation(float tx,float ty,float **in){
    for(int i = 0; i < FIL; i++){
        for(int j = 0;j < FIL;j++){
            if(i == j){
                in[i][j] = 1;
            }
            else{
                in[i][j] = 0;
            }
        }
    }
    in[2][0] = -1*tx;
    in[2][1] = -1*ty;
}
```

Rotación.
Creamos la matriz de rotacion y la multiplicamos con una matriz de entrada, la cual esta dada por referencia.

```
//ROTACION
void rotation(float theta,float **in){
    float rota[FIL][COL] = {{cos(theta),-1*sin(theta), 0 },
                            {sin(theta),  cos(theta) , 0 },
                            {     0    ,      0      , 1 }};
    multMatriz(in,rota);
}
```

Escalamiento.
Creamos la matriz de escalamiento y la multiplicamos con una matriz de entrada, la cual esta dada por referencia.

```
//ESCALAMIENTO
void scaling(float sx,float sy,float **in){
    float scal[FIL][COL] = {{ sx , 0  , 0 },
                            { 0  , sy , 0 },
                            { 0  , 0  , 1 }};
    multMatriz(in,scal);
}
```

Traslación inversa.
Creamos la matriz de traslacion inversa y la multiplicamos con una matriz de entrada, la cual esta dada por referencia.

```
//TRASLACION INVERSA (REGRESA POSICION ORIGINAL)
void traslationInversa(float tx,float ty,float **in){
    float trasInv[FIL][COL] = {{ 1 , 0 ,tx},
                               { 0 , 1 ,ty},
                               { 0 , 0 , 1}};
    multMatriz(in,trasInv);
}
```

Creación de Matriz de modelado.
Creamos la matriz de modelado, creamos matrices dinamicas y llamamos a las funciones de traslacion, rotacion y escalamiento para multiplicar todo por la matris de traslacion inversa y devolver la matriz de modelado por referencia.

```
//CREACION MATRIZ DE MODELADO
void modeladoCreation(float tx,float ty,float sx,float sy,float
theta,float **modelado){
    //MATRIZ DE TRASLACION
    float **tras = reservarMemoria(FIL,COL);
    //OP de matriz de modelado
    traslation(tx,ty,tras);
    rotation(theta,tras);
    scaling(sx,sy,tras);
    //T' de matriz de modelado
    traslationInversa(tx,ty,tras);
    for(int i = 0; i < FIL;i++){
        for(int j = 0;j < COL;j++){
            modelado[i][j] = tras[i][j];
        }
    }
}
```

Dibujar poligono.
Creamos dos matrices dinamicas, una para los puntos y otra la de modelado, llamamos la funcion para multiplicar la matriz de modelado con puntos y guardamos los valores en dos variables auxiliares para despues llamar al algoritmo de Bresenham e imprimir la linea, repetimos lo mismo con los demas puntos.

```
//DIBUJAR POLIGONO
void draw(float Dtx,float Dty,float Dsx,float Dsy,float Dtheta){
    float **points = reservarMemoria(3,1);
    float **MModelado = reservarMemoria(FIL,COL);
modeladoCreation(ax+Dtx,ay+Dty,scalex+Dsx,scaley+Dsy,finaltheta+Dtheta,MM
odelado);
    multMatrizPoints(ax,ay,MModelado,points);
    TP1 = points[0][0];
    TP2 = points[1][0];
    multMatrizPoints(bx,by,MModelado,points);
    //Bresenham
```

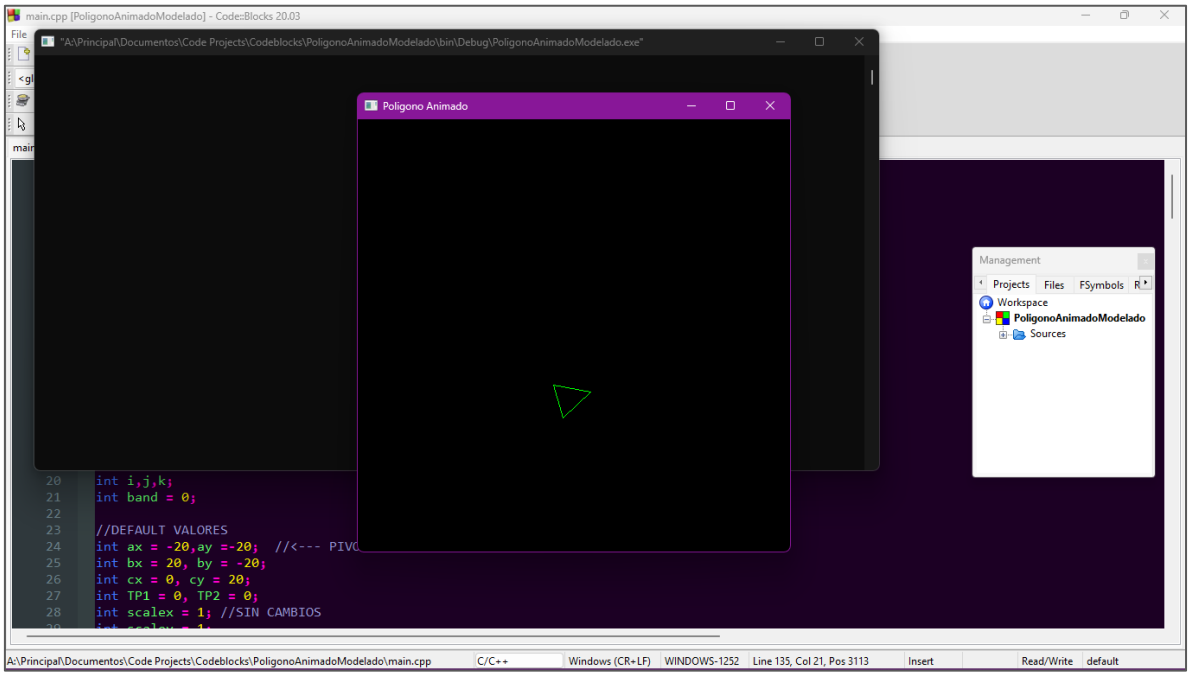
```
    Bresenham(TP1,TP2,points[0][0],points[1][0]);
    TP1 = points[0][0];
    TP2 = points[1][0];
    multMatrizPoints(cx,cy,MModelado,points);
    //Bresenham
    Bresenham(TP1,TP2,points[0][0],points[1][0]);
    TP1 = points[0][0];
    TP2 = points[1][0];
    multMatrizPoints(ax,ay,MModelado,points);
    //Bresenham
    Bresenham(TP1,TP2,points[0][0],points[1][0]);
}
```

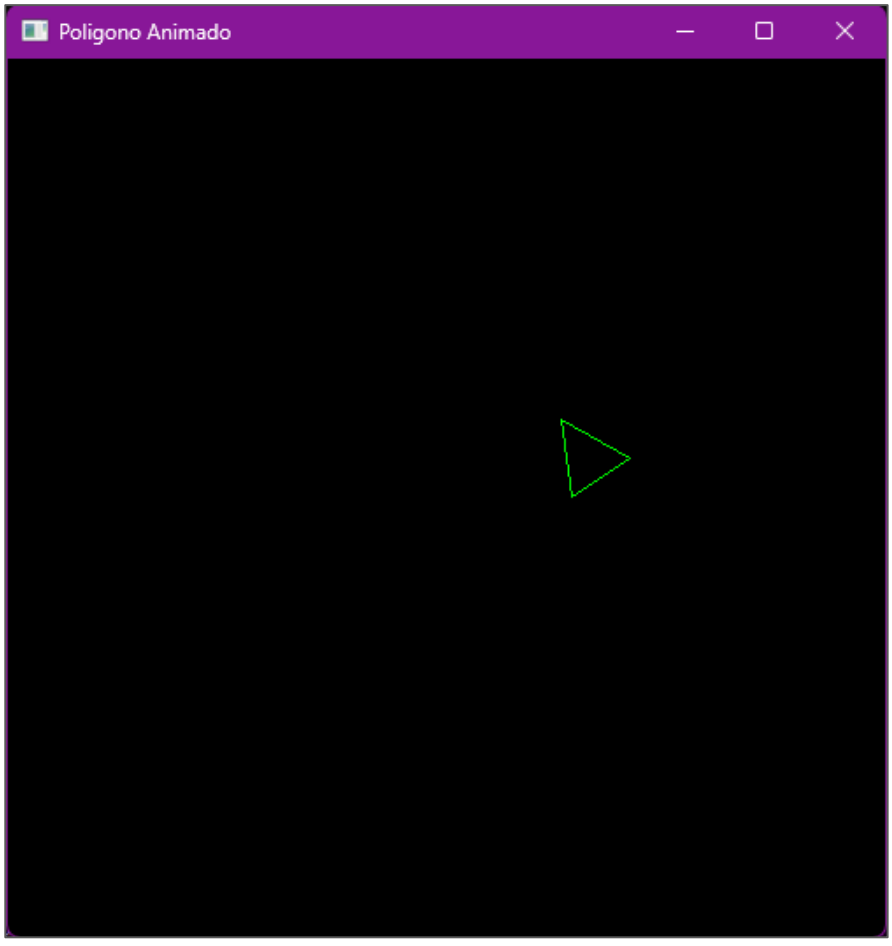
Dibuja (pero con variables de control).
Llamamos a la funcion de dibujado y controlomamos sus parametros con variables de control.

```
//Dibuja grafica
void dibujaGrafica(){
    glClear(GL_COLOR_BUFFER_BIT);  //borra la ventana de visualizacion
    //Dibujado
    draw(DIFtx,DIFty,DIFsx,DIFsy,DIFtheta); //Dibuja poligono
    ///Activamos las variables control
    //VC Traslacion
    if(band == 0){
        DIFtx++;
        if(DIFtx == 270)
            band = 1;
    }
    if(band == 1){
        DIFtx--;
        if(DIFtx == -270)
            band = 0;
    }
    //VC Rotacion
    if(DIFtheta < 6)
        DIFtheta+=0.2;
    else
        DIFtheta = 0;
    glFlush();
    Sleep(50);
}
```

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

EJECUCIONES





Jesús Huerta Aguilar

CONCLUSIONES

En conclusión, los pasos a seguir para la correcta implementación del algoritmo son: se almacena como primer punto el que se encuentra más a la izquierda y se pinta. Luego, se calculan los valores de incremento de x y de y, junto al parámetro de decisión, anotado p. Para cada x se comprueba la distancia genérica. Si ésta es menor o igual a cero, el siguiente punto a trazar será (x+1, y) y el parámetro de decisión del siguiente punto será igual al anterior más dos veces el incremento de y. Pero, si es mayor de cero, el siguiente punto a trazar será (x+1, y+1) y el parámetro de decisión se verá incrementado en 2dy-2dx.

BIBLIOGRAFIA

- Olmos, I. (2022, 8 septiembre). Reunion en «General». sharepoint. Recuperado 9 de septiembre de 2022, de https://correobuap.sharepoint.com/sites/Section_202235-CCOS261-10592-001/Shared%20Documents/Forms/AllItems.aspx?FolderCTID=0x01200042EC72C454FA1E45805A79EC6BAB0F21&isAscending=false&sortField=Modified&id=%2Fsites%2FSection_202235-CCOS261-10592-001%2FShared%20Documents%2FGeneral%2FRecordings%2FReunión%20en%20_General_-20220908_070554-Grabación%20de%20la%20reunión%2Emp4&parent=%2Fsites%2FSection_202235-CCOS261-10592-001%2FShared%20Documents%2FGeneral%2FRecordings
- Stackoverflow. (s. f.). C++ getting «invalid user-defined conversion» error. stackoverflow. Recuperado 9 de septiembre de 2022, de https://stackoverflow.com/questions/20314135/c-getting-invalid-user-defined-conversion-error