AUTOR: Marcelo Alves Gomes

Introdução

O primeiro trabalho dessa disciplina consiste em usar nossos conhecimentos adquiridos na aula afim de desenvolver funções de rasterização pedidas pelo professor usando a API OpenGL , são elas as: putPixel, DrawLine, DrawTriangle , tais funções são importantes de se aprender , pois serão a base para um desenvolvimento de curso e de conhecimento para as demais matérias seguintes.

Para começar , tem que se ter em mente que o processo utilizado para realizar tal tarefa , só se deu devido a uma framework feita pelo professor que simula o acesso a memória de vídeo feita pelo professor Christian , sem essa ferramenta , não seria possível acessar diretamente a memória.

PutPixel

Sabemos que um pixel é constituído pelo sistema RGBA de cores, e é localizado pela sua posição no espaço, sabendo isso, teremos que criar uma 2 estruturas para tal , afim de manter a organização e melhor compreensão. A estrutura que criamos primeiramente foi a da localização baseado no sistema cartesiano, em que o x se baseará no eixo das abscissas e o y no eixo das ordenadas da reta. A segunda estrutura será a da cor, baseada no sistema RGBA (três cores primarias somadas a propriedade de transparência Alpha), em que seus valores podem variar de 0 a 255, totalizando assim 256 possíveis estados para cada canal, tambem podemos afirmar que se pode utilizar 1 byte para cada canal , ficando assim com 4 bytes alinhados por cada pixel (destaque o “alinhados” pois o framebuffer “FBptr” utilizado no programa simulador consiste em uma matriz unidimensional, com tamanho relativo a dimensão da tela, para acessar o pixel que deseja-se pintar, o ponteiro FBptr move-se na direção horizontal, passando por um determinado número de bytes, até chegar ao pixel desejado ,isso será importante para compreender a formula seguinte) .Além disso , criamos uma cor padrão de estudo(vermelho) do tipo da estrutura que criamos anteriormente.

Com isso , temos que saber que a formula que faz acessar um pixel de coordenadas (x,y) baseado nas dimensões de uma tela de LARGURA X ALTURA é y*4*IMAGE_WIDTH+x*4 , onde IMAGEM_WIDTH é a largura da tela , não se utilizou a altura pois como explicado anteriormente , o framebuffer move em direção horizontal da largura e não da altura.

Continuando , após aplicar a formula , sabemos a localização exata do pixel , agora basta alterar cada canal do pixel de acordo com o que o ponteiro FBptr apontar , como são 4 canais , o ponteiro terá que se direcionar para cada um para que se possa completar uma cor requerida.Caso surja duvida a respeito

do FBptr ter variações de +1 , +2 ,+3 , isso se deve ao fato de escolher qual canal quer alterar para escrever na memória.

```
10  //1- Variaveis e estruturas que serão usadas para realizar as funcoes
11
12  //1.1 Essa será a estrutura que irá organizar a localização do pixel
13  struct pixel{
14      int x; // baseado no sistema cartesiano , o x estará direcionando a localizaçã do e
15      int y; // baseado no sistema cartesiano , o y estará direcionando a localizaçã do e
16  };
17  //1.2 Essa será a estrutura que irá cuidar da parte da cor do pixel ,vale saber que a m
18  struct cor{
19      int r ; // variação do vermelho
20      int g ;// variação do verde
21      int b ; //variação do azul
22      int a ;// transparencia
23
24  };
25  cor corVermelho= {255,0,0,1}; //Declarei um exemplo para testar
26  //          { R  G B A  }
27
28  //2- Função putPixel , sua entrada será a localização do pixel nas coordenadas , seguid
29      void putPixel(int x, int y,cor cor){
30          //pc = PixelCompleto
```

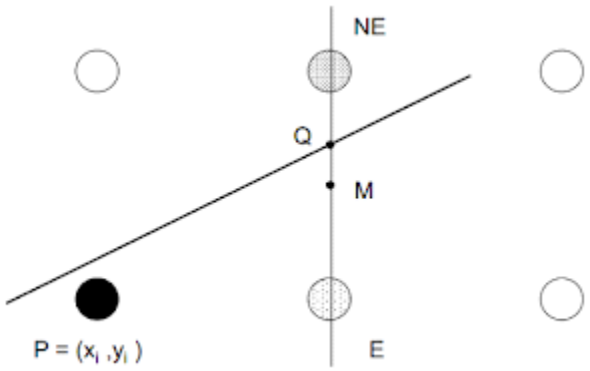
```
28  //2- Função putPixel , sua entrada será a localização do pixel nas coordenadas , seguid
29  void putPixel(int x, int y,cor cor){
30      //pc = PixelCompleto
31      int pc = y*4*IMAGE_WIDTH+x*4; // Essa é a formula para achar a posição do pixe
32      //como já explicado , os bytes do sistema RGBA estao alinhados em sequencia de
33
34      FBptr[pc] =   cor.r; // posicao 1 irá armazenar o valor 255 na variavel r
35      FBptr[pc+1] =   cor.g;
36      FBptr[pc+2] =   cor.b;
37      FBptr[pc+3] =   cor.a;
38  }
39
```

```
1   #include "main.h"
2
3   //--------------------------------------------------------------------------------
4   void MyGlDraw(void)
5   {
6       //************************************************************************
7       // Chame aqui as funções do mygl.h
8       //************************************************************************
9
10      putPixel(255,255,corVermelho);
11      putPixel(255,300,corVermelho);
12      putPixel(300,300,corVermelho);
13      putPixel(255,210,corVermelho);
14
15
16
17
18
19
```

(Resultado de saida)

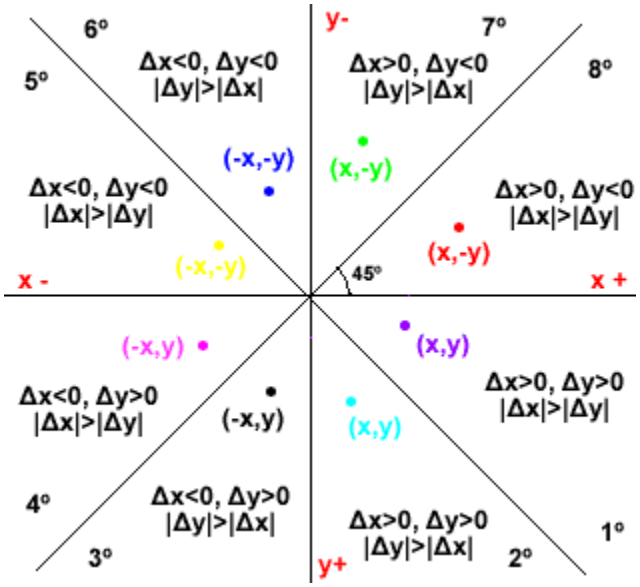
DrawLine

A rasterização de linhas nesse caso se dá pelo clássico algoritmo de brezenham, esse algoritmo é muito famoso no que se baseia em eficiência, pois ele trabalha na questão de otimização, sem utilizar-se de multiplicações sucessivas, ao invés disso , trabalha com incremento e decremento , sem forçar assim o grande trabalho da memória e da maquina. A compreensão do algoritmo de Brezenham é importante para que se possa ter entendimento de sua variação que será feita seguidamente.Seu objetivo principal se baseia em determinar pixels que serão acesos seguidamente para formar uma reta , do principio do ponto inicial até o ponto final.Com isso , a forma de como os pixels serão desenhados será dada a partir de uma variável de decisão d ,em que sua formula se baseia no seguinte d = 2*(dy-dx) em que dy = (Y1-Y0) e o dx = (X1-X0) ,caso d <=0 o pixel seguinte a ser pintado será o de leste , caso d>0 o próximo pixel a ser pintado será o nordeste.Cada vez que se pintar um pixel, a incrementação será feita em referencia a d , ou seja , caso o pixel a ser pintado estiver localizado na posição leste , só o x será incrementado , já se no caso o pixel a ser pintado estiver localizado na região nordeste , será incrementado tanto o x quanto o y , a imagem a seguir dará melhor entendimento ao explicado:



(O pixel inicial é dado por P=(Xi,Yi), o próximo a ser pintado irá depender da relação da reta com o ponto médio , se ela estiver acima , o próximo pixel será o NE , se não , o próximo será o E.)

O problema encontrado na questão da implementação do algoritmo de Bresenham foi que ele só consegue escrever segmentos de retas no octante 1 em que a variação de graus vai de 0° até 45° , para isso , foi feito alterações no código e implementações de simetria para que os outros 7 pudessem ser representados , abaixo temos uma imagem que representa um resumo da questão de simetria dos octantes e em seguida mostra-se como foi feito sua implementação por meio do código



(Sabendo o caso principal em que o algoritmo de Bresenham trabalha , pode-se usar conhecimentos de simetria nos quadrantes e octantes para que um complemente o outro , por exemplo , sabe-se que o 8°e 5°octante mostrado na imagem são um espelho um do outro , se soubermos por exemplo o delta x e delta y em questão , e trocando seus valores , achamos os seus respectivos espelhos.Faremos esse mesmo principio na implementação do código)

cg_framework\main.cpp | *cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

```
44  void DrawLine(int x0,int y0,int xl,int yl, cor cor1,cor cor2){
45      int adc_y =1;
46      bool n = false;//variavel para controlar os casos
47      int aux; // variavel aux que será utilizada nas trocas de variaveis dos if's
48      int dx=(xl-x0); // deltaX
49      int dy=(yl-y0); // deltaY
50
51      if(dy>dx){ //Troca as coordenadas dos pontos caso dy>dx , esse caso pode ser chamado de espel
52
53          aux=x0;
54          x0=y0;
55          y0=aux;
56
57          xl=aux;
58          xl=yl;
59          yl=aux;
60
61          aux=dx;
62          dx=dy;
63          dy=aux;
64
65          n=true; // n fica true para possa ser executado quando for trocar ,
66      }
67
```

cg_framework\main.cpp | *cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

```
68
69      if(xl<x0){ // quando DeltaX<0 , inverte a direção da seta
70          //troca(yl,y0)
71          aux=yl;
72          yl=y0;
73          y0=aux;
74          //troca(xl,x0)
75          aux=xl;
76          xl=x0;
77          x0=aux;
78      }
79
80      if(dy<0){// decrementa os valores de 1 e da variavel de decisão
81          dy=-1*dy;
82          adc_y=-1;
83      }
84
85      int d=2*(dy-dx); // variavel de decisao
86      int incr_e=2*dy; // não pode declarar essas variaveis em cima , pois pode ter alteração de vi
87      int incr_ne=2*(dy-dx);
88
89      //valores iniciais
90      int y=y0;
91      int x=x0;
```

cg_framework\main.cpp | *cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

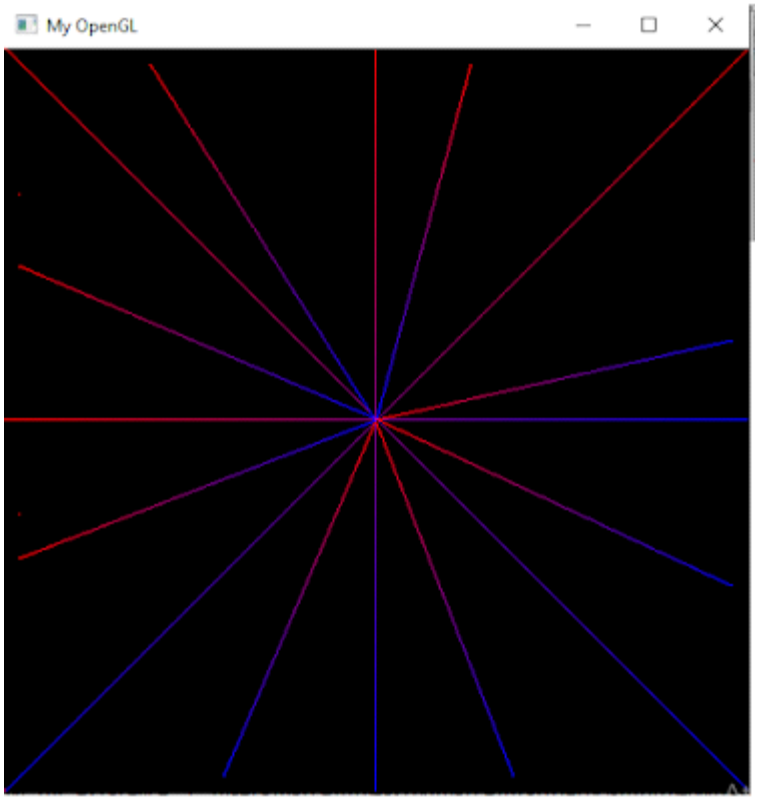
```
89          //valores iniciais
90          int y=y0;
91          int x=x0;
92
93          //INTERPOLAÇÃO DE CORES
94          cor cor = cor1;
95          float Gverde=(cor2.g-cor1.g)/dx;
96          float Rvermelho = (cor2.r-cor1.r)/dx;
97          float Bazul=(cor2.b-cor1.b)/dx;
98
99          putPixel(x,y,cor); //ponto inicial em que a reta vai se basear
100
101             while(x<xl){//para poder fazer a reta , necessita-se de um laço de repeticao
102
103                 cor.g+=Gverde;
104                 cor.r+=Rvermelho;
105                 cor.b+=Bazul;
106
107                 if(d<=0){ // escolha do leste , só o x será incrementado
108                     d+=incr_e;
109                     x++;
110             }
111                 else{
112                     d+=incr_ne; // escolha da regiao nordeste , x e y serão incrementados
```

cg_framework\main.cpp | *cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

```
104                 cor.r+=Rvermelho;
105                 cor.b+=Bazul;
106
107                 if(d<=0){ // escolha do leste , só o x será incrementado
108                     d+=incr_e;
109                     x++;
110             }
111                 else{
112                     d+=incr_ne; // escolha da regiao nordeste , x e y serão incrementados
113                     x++;
114                     y+=adc_y;
115             }
116                 if(n){ // if(n) quer dizer que se o valor de n for verdade , então executa , ou seja ,
117                     putPixel(y,x,cor); //permutação/troca irá ocorrer novamente caso elas tenham sido
118             }
119                 else{
120                     putPixel(x,y,cor);
121             }
122         }
123
124     }
125 #endif // _MYGL_H_
126
127
```

*cg_framework\main.cpp | cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

```
16      DrawLine(255,0,255,510,corVermelho,corAzul);//eixo x
17      DrawLine(0,255,510,255,corVermelho,corAzul);//eixo y
18      DrawLine(0,0,510,510,corVermelho,corAzul);//y=-x
19      DrawLine(510,0,0,510,corVermelho,corAzul);//y=x
20      DrawLine(255,255,500,370,corVermelho,corAzul);//1°octante
21      DrawLine(255,255,350,500,corVermelho,corAzul);//2°octante
22      DrawLine(255,255,150,500,corVermelho,corAzul);//3°octante
23      DrawLine(10,350,255,255,corVermelho,corAzul);//4°octante
24      DrawLine(10,150,255,255,corVermelho,corAzul);//5°octante
25      DrawLine(100,10,255,255,corVermelho,corAzul);//6°octante
26      DrawLine(320,10,255,255,corVermelho,corAzul);//7°octante
27      DrawLine(255,255,500,200,corVermelho,corAzul);//8°octante
28
29
```



(Resultado de saida)

Interpolação de cores

Entende-se que interpolação de cores de um segmento de reta é a rasterização de uma linha em que sua cor inicial se modifica ao longo do percurso da reta, trazendo incrementalmente novas cores desde o inicio até o final , com isso já sabemos que teremos que definir uma cor inicial e final na chamada da função e na sua implementação .Sabe-se que o algoritmo de rasterização mostrado anteriormente funciona em etapas , da forma pixel por pixel , ou seja , só podemos modificar a cor a cada novo pixel criado.Logo , define-se que o incremento de cor para cada canal(RGBA) do pixel se dará pela formula Gverde=(cor2.g-cor1.g)/dx , onde Gverde é o incremento do canal verde assim como Bazul(incremento do canal azul), Rvermelho(incremento do canal vermelho) ,cor2.g é o valor do canal verde no fim da linha e cor1.g é o valor do canal verde no inicio da linha e dx é o numero de vezes que o algoritmo de rasterização irá rodar (tal lógica se aplica pois o while(x<x1) é justamente o dx).Um detalhe que deve-se ressaltar foi que devido a alguns erros de compilação da interpolação de cores com canais do tipo int estavam dando erro , pois não interpolavam as linhas localizadas especificamente nas abscissas e nas ordenadas , só interpolavam os octantes diagonas , logo teve-se que alterar para tipo float , com isso , solucionou o problema.Sabendo disso , o que foi feito foi apenas colocar as incrementações no final do ciclo while(x<x1), pois seguindo a didática , cada fez que se termina um ciclo , uma variação dos canais de cores será feita até que se possa chegar na cor final.Uma observação a ser feita , foi que para ficar mais organizado modifiquei o local das incrementações para no final do while , onde seria mais fácil a compreensão lógica , segue as imagens das modificações.

```
cg_framework\main.cpp | *cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

 7    // Defina aqui as suas funções gráficas
 8    //*********************************************************************
 9
10    //1- Variaveis e estruturas que serão usadas para realizar as funcoes
11
12    //1.1 Essa será a estrutura que irá organizar a localização do pixel
13    struct pixel{
14        int x; // baseado no sistema cartesiano , o x estará direcionando a localizaçã do eixo das abcis
15        int y; // baseado no sistema cartesiano , o y estará direcionando a localizaçã do eixo das ordenad
16    };
17    //1.2 Essa será a estrutura que irá cuidar da parte da cor do pixel ,vale saber que a maneira de como
18    struct cor{
19        float r ; // variação do vermelho
20        float g ;// variação do verde
21        float b ; //variação do azul
22        float a ;// transparencia
23
24    };
25    //Cores predefinidas para a pessoa usar quando for fazer a chamada da função
26    cor corVermelho= {255,0,0,1}; //Declarei um exemplo para testar
27    cor corVerde= {0,255,0,1}; //Declarei um exemplo para testar
28    cor corAzul= {0,0,255,1}; //Declarei um exemplo para testar
29
30    //          { R G B A }
```

(linha 19- mudança feita do int para o float)

```
cg_framework\main.cpp | cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

85          int d=2*(dy-dx); // variavel de decisao
86          int incr_e=2*dy; // não vamos declarar essas variaveis em cima , pois pode ter alteração da va
87          int incr_ne=2*(dy-dx);
88
89          //valores iniciais
90          int y=y0;
91          int x=x0;
92
93          //INTERPOLAÇÃO DE CORES
94          cor cor = cor1;
95          float Gverde=(cor2.g-cor1.g)/dx;
96          float Rvermelho = (cor2.r-cor1.r)/dx;
97          float Bazul=(cor2.b-cor1.b)/dx;
98
99          putPixel(x,y,cor); //ponto inicial em que a reta vai se basear
100
101             while(x<x1){//para poder fazer a reta , necessita-se de um laço de repeticao
102                 if(d<=0){ // escolha do leste , só o x será incrementado
103                     d+=incr_e;
104                     x++;
105             }
106                 else{
107                     d+=incr_ne; // escolha da regiao nordeste , x e y serão incrementados
108                     x++;
```

(linha 94 -Formula de incremento de cor para cada canal)

```
97          float Bazul=(cor2.b-cor1.b)/dx;
98
99          putPixel(x,y,cor); //ponto inicial em que a reta vai se basear
100
101             while(x<x1){//para poder fazer a reta , necessita-se de um laço de repeticao
102                 if(d<=0){ // escolha do leste , só o x será incrementado
103                     d+=incr_e;
104                     x++;
105             }
106                 else{
107                     d+=incr_ne; // escolha da regiao nordeste , x e y serão incrementados
108                     x++;
109                     y+=adc_y;
110             }
111                 if(n){ // if(n) quer dizer que se o valor de n for verdade , então executa , ou seja
112                     putPixel(y,x,cor); //permutação/troca dos valores novamente caso algo tenham sido
113             }
114                 else{
115
116                     putPixel(x,y,cor);
117             }
118                 cor.g+=Gverde; //incrementações de cores no final do ciclo do while , cada vez que el
119                 cor.r+=Rvermelho;
120                 cor.b+=Bazul;
```

(linha 117 - começa a parte de incremento da mudança de cor)

DrawTriangle

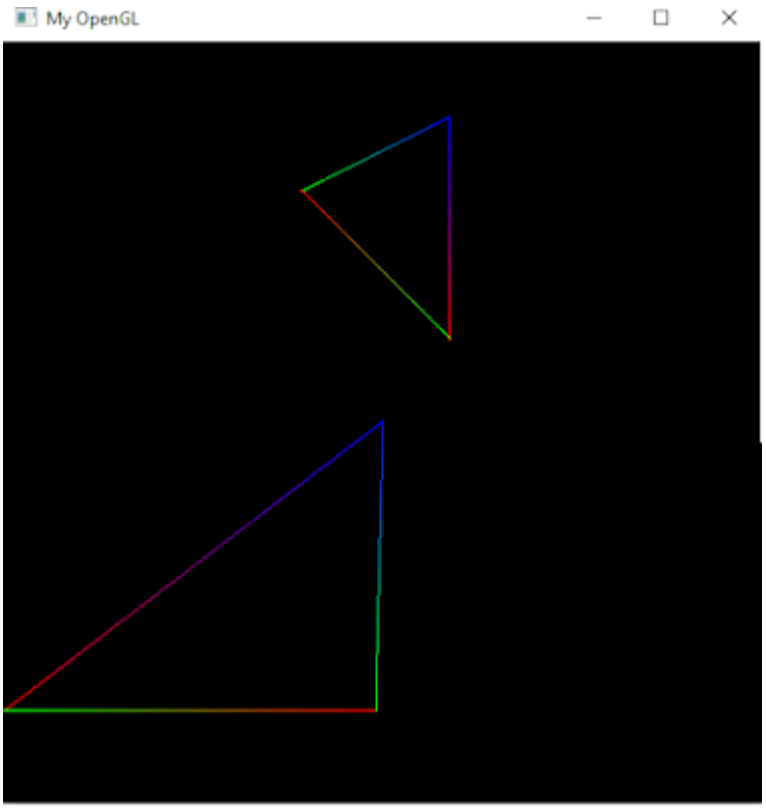
A compreensão da função DrawTriangle se dá pelo entendimento da função DrawLine , sabendo disso , para desenharmos um triangulo , basta chamarmos a função DrawLine 3 vezes , em que na primeira chamada desenhamos uma linha do (x0,y0) para (x1,y1) , depois uma linha do(x1,y1) para o (x2,y2) , e depois uma linha do (x0,y0) para o (x2,y2).Implementação junto com os testes está nas imagens a seguir:

```
109                 y+=adc_y;
110             }
111                 if(n){ // if(n) quer dizer que se o valor de n for verdade , então executa , ou seja
112                     putPixel(y,x,cor); //permutação/troca dos valores novamente caso algo tenham sido
113             }
114                 else{
115                     putPixel(x,y,cor);
116             }
117                 cor.g+=Gverde; //incrementações de cores no final do ciclo do while , cada vez que el
118                 cor.r+=Rvermelho;
119                 cor.b+=Bazul;
120
121         }
122
123     }
124     void DrawTriangle(int x0,int y0,int x1,int y1,int x2,int y2 ,cor cor1,cor cor2,cor cor3){
125         DrawLine(x0,y0,x1,y1,cor1,cor2);
126         DrawLine(x1,y1,x2,y2,cor2,cor3);
127         DrawLine(x0,y0,x2,y2,cor3,cor1);
128
129     }
130 #endif // _MYGL_H_
131
132
```

```
cg_framework\main.cpp | cg_framework\mygl.h | cg_framework\definitions.h | cg_framework\main.h

25      DrawLine(100,10,255,255,corVermelho,corAzul);//6°octante
26      DrawLine(320,10,255,255,corVermelho,corAzul);//7°octante
27      DrawLine(255,255,500,200,corVermelho,corAzul);//8°octante
28  */
29      DrawTriangle(0,450,255,255,250,450,corVermelho,corAzul,corVerde);
30      DrawTriangle(200,100,300,50,300,200,corVerde,corAzul,corVermelho);
31
32
33
34
35
36
37
38
39  }
40
41  //------------------------------------------------------------------------
42  int main(int argc, char **argv)
43  {
44      // Inicializações.
45      InitOpenGL(&argc, argv);
```

(Testes)



(Resultado de saida)

Bibliografia

Slides do professor Christian(UFPB)

http://disciplinas.ist.utl.pt/leic-cg/textos/livro/Rasterizacao.pdf -> **ajudou muito no entendimento do algoritmo de Bresenham**

https://pt.wikipedia.org/wiki/Octante_(geometria_espacial) **->ajudou muito a questão da simetria**

https://pt.wikipedia.org/wiki/Algoritmo_de_Bresenham

http://www.dca.fee.unicamp.br/courses/IA725/1s2006/notes/n4.pdf

http://www.univasf.edu.br/~jorge.cavalcanti/comput_graf04_prim_graficas2.pdf

https://stackoverflow.com/questions/25878029/smooth-color-interpolation-along-a-bresenham-line