Aluno: José Pedro Henrique Maul dos Santos 2016083447

Introdução

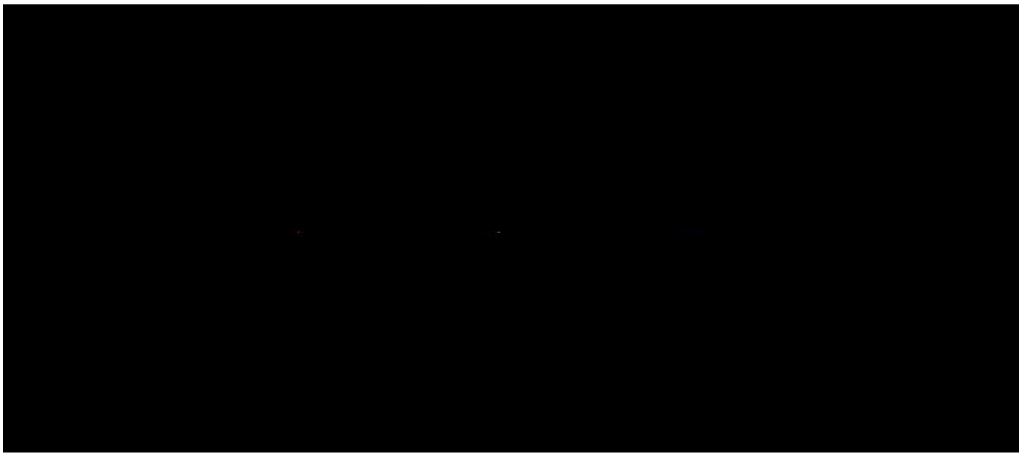
Devemos implementar um algoritmo para a rasterização de pontos e linhas. Além de fazer triângulos que serão ser desenhados através da rasterização das linhas que compõem suas arestas.
A rasterização destas primitivas será feita através da escrita direta na memória de vídeo. Como os sistemas operacionais atuais protegem a memória quanto ao acesso direto, utilizaremos um framework que simula o acesso à memória de vídeo.
Usaremos um framework desenvolvido pelo prof. Christian Azambuja Pagot pois os sistemas da atualidade não permitem acesso direto à memória de vídeo. Ele tem como objetivo simular a memória de vídeo, o frame buffer e o color buffer.

PutPixel

Na memória do computador a tela é representada de forma contínua, ela é uma linha que tem 4*largura*altura bytes de tamanho, como queremos pintar um pixel de coordenadas (x,y) é necessário fazer 4*x + 4*y*largura , acessamos o primeiro elemento do pixel , ou seja , o vermelho.

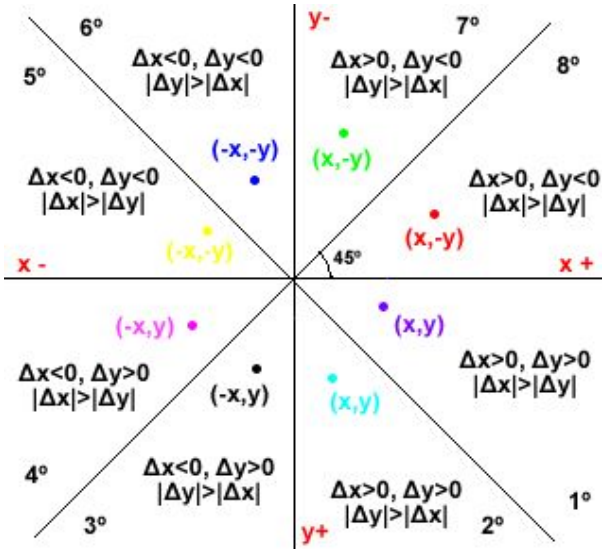
```
void PutPixel(int coordenadaX, int coordenadaY, tCor *cor){
  int byte = (coordenadaX* 4) + (coordenadaY * IMAGE_WIDTH * 4);
  FBptr[byte] = cor->red;
  FBptr[byte + 1] = cor->green;
  FBptr[byte + 2] = cor->blue;
  FBptr[byte + 3] = cor->alpha;
}
```

Colocando os conceitos em algoritmo, temos o exemplo anterior onde Fbptr é um ponteiro para o início da memória e Color um dos struct que criei e contém 4 floats e cada um é um componente RGBA



DrawLine

Para a implementação de função de rasterizar linhas utilizei o algoritmo de Bresenham. Esse algoritmo otimiza os custos de operações para rasterizar linhas pois é baseado apenas na incrementação dos pontos.
Notei que este algoritmo apenas permite a rasterização de linhas apenas entre 0º a 45º, no primeiro quadrante, então foi preciso fazer pequenas condições para que conseguisse rasterizar em todos os quadrantes.



Interpolação de Cores

O método de Interpolação de cores foi bem simples de fazer, inicialmente peço duas cores, uma inicial e outra final, assim que é inserido utilizamos uma variável auxiliar para receber a cor inicial.

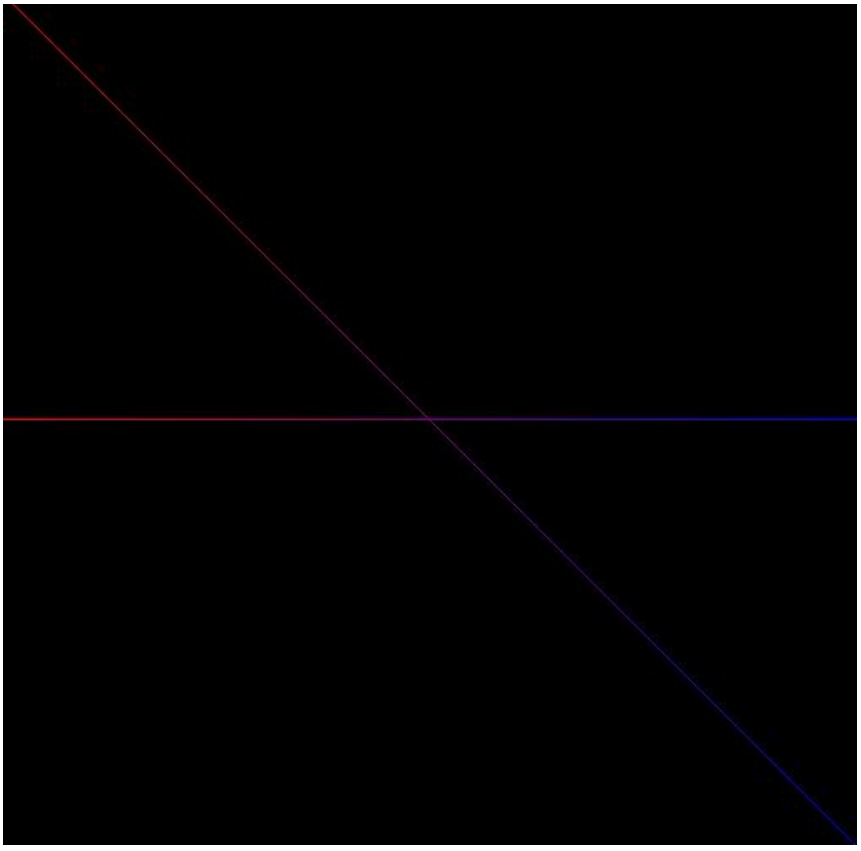
```
Color * corAuxiliar = (struct Color *) malloc(sizeof(Color));
        corAuxiliar->red = corInicial->red;
        corAuxiliar->green = corInicial->green;
        corAuxiliar->blue = corInicial->blue;
        corAuxiliar->aplha = corInicial->alpha;
```

Em seguida, determei o dR, dG, dB e dA, que é a variação de cor do ponto inicial ao ponto final, esses deltas são obtidos pela subtração dos valores finais pelos iniciais.

```
dR = (corFinal->red - corAuxiliar->red);
dG = (corFinal->green - corAuxiliar->green);
dB = (corFinal->blue - corAuxiliar->blue);
dA = (corFinal->alpha - corAuxiliar->alpha);
```

Após isso dividimos todos esses deltas obtidos pelo dX que é a variação da linha, e obtemos a quantidade de cor que varia de pixel para pixel de acordo com o desenho

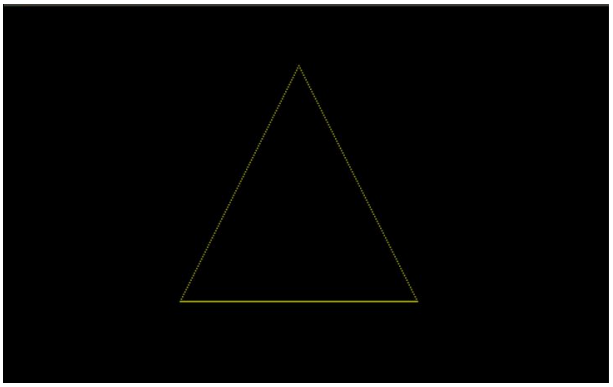
da linha, e podemos ter uma mudança de cor uniforme chegando do totalmente vermelho até o totalmente azul.



Rasterização de Triângulo

Esta função foi simples dado que já temos a implementação de rasterização de uma reta, com isso basta adicionar os pontos restantes para rasterizar um triângulo.

```
void drawTriangle(Coordenadas * pontoInicial, Color * cor1, Coordenadas * pontoIntermediario, Color * cor2,
Coordenadas * pontoFinal, Color * cor3) {
        DrawLine(pontoInicial,pontoIntermediario,cor1,cor2);
         DrawLine(pontoIntermediario,pontoFinal,cor2,cor3);
         DrawLine(pontoFinal,pontoInicial,cor3,cor1);}
```



Dificuldades Encontradas

A maior dificuldade encontrada neste trabalho foi na lógica de funcionamento e na implementação da função DrawLine, principalmente na generalização dos quadrantes.

Referência

Slides do professor Cristian
https://bitunico.wordpress.com/2012/12/16/rasterizacao-em-cc-algoritmo-de-bresenham/
http://fleigfleig.blogspot.com/