Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação

Disciplina: Computação Gráfica

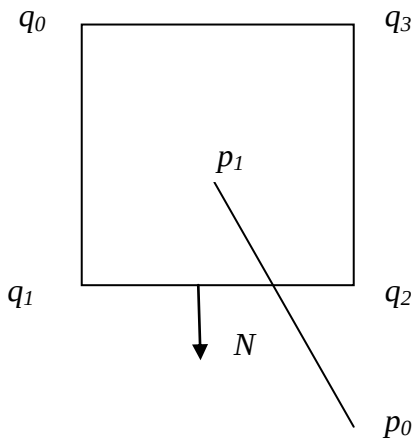
AD1 - 1° semestre de 2013.

1) Sob quais condições um dado de terreno pode ser compreendido como uma imagem? (1.0 ponto).

   Um terreno somente pode ser visto como uma imagem quando ele pode ser modelado como uma função z = f(x,y). Se houver ao menos um ponto do terreno com mais de um valor de altura, por exemplo, no caso de cavernas, então o modelo de imagem não é válido, já que não será possível definir uma função nesta situação.

2) Mostre como determinar se um segmento de reta dado por dois pontos p_0 e p_1, que intersecta uma janela retangular com vértices q_0, q_1, q_2 e q_3, tem sentido de penetração (entrada). Considere que a interseção ocorre no lado determinado pelo segmento q_1q_2. (1.0 ponto)

   Para isto basta determinar o vetor normal N ao lado q_1q_2 (vetor perpendicular que aponta para fora da janela) e calcular o produto interno entre o vetor $(p_1\text{-}p_0)$ e N. Se $\langle N, p_1\text{-}p_0\rangle$ for negativo, então o ponto de interseção é um ponto de entrada, já que $p_1\text{-}p_0$ tem sentido oposto ao da normal, caso contrário, terá sentido de saída. Se $\langle N,p_1\text{-}p_0\rangle = 0$ então o segmento é paralelo ao lado q_1q_2.

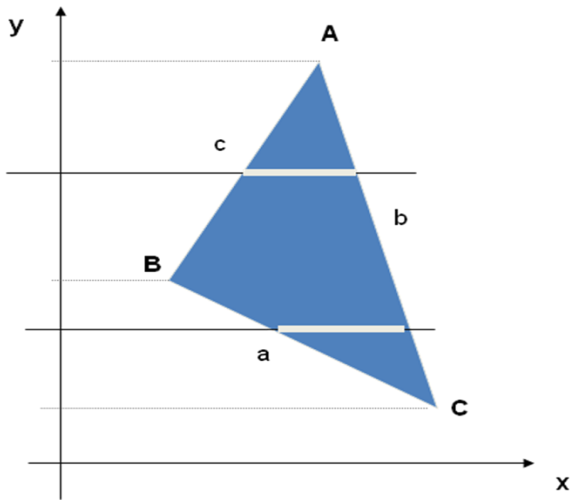


3) Implemente um algoritmo de rasterização de polígonos (não usar o rasterizador do pipeline do OpenGL) (1.0 ponto).

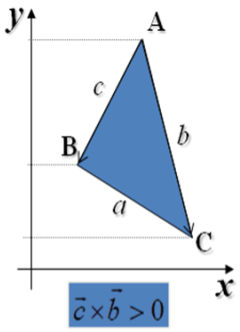
Ver solução no final do documento.

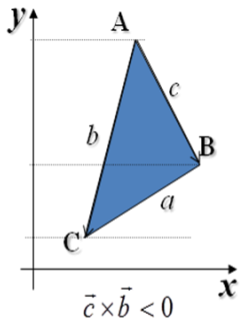
4) Faça uma versão do algoritmo de rasterização adaptada especialmente para triângulos (não usar o rasterizador do pipeline do OpenGL) (1.0 ponto).

A versão é basicamente uma simplificação do algoritmo implementado na questão 3. A diferença fundamental é que triângulos tem somente duas arestas ativas, não necessitando de uma lista para armazená-las.



Além disso, triângulos admitem somente duas configurações possíveis: uma em que a aresta mais longa AC, começando do vértice superior A, encontra-se à esquerda e na outra em que tal aresta encontra-se à direita. Um teste de posicionamento relativo entre arestas, baseado no produto vetorial permite distinguir os dois casos.





$$\vec{c}\times\vec{b}=(x_B-x_A)(y_C-y_A)-(x_C-x_A)(y_B-y_A)$$

A detecção de quando uma aresta passa a ser ativa pode ser feita de forma simples. Haverá mudança nas arestas ativas quando a *scanline* alcançar o vértice intermediário B, considerando a coordenada y. Neste momento uma das arestas, ou a esquerda, ou a direita deixará de ser ativa e uma nova aresta será ativada, no caso BC. Observe que sempre a aresta AB será desativada, dando lugar a aresta BC. A única diferença é que AB pode ser a aresta ativa à esquerda ou à direita do triângulo, dependendo da configuração.

5) Explique porque o algoritmo Cohen-Sutherland não funciona para janelas que não sejam retangulares e alinhadas com os eixos cartesianos. (2.0 pontos).

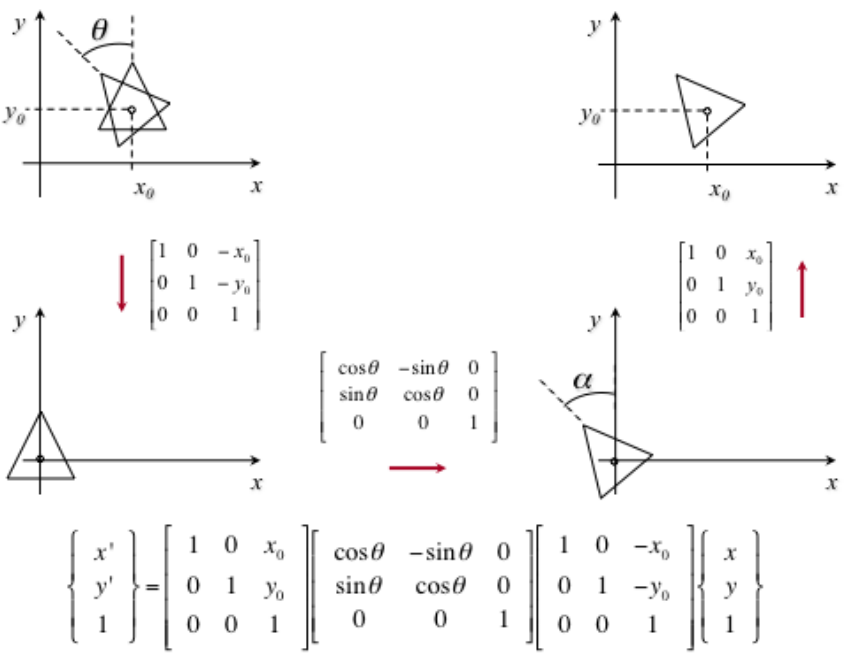
Anulada

Porque o algoritmo depende de uma subdivisão do plano, que contém uma janela, em 9 subregiões, onde para cada uma delas atribuímos um código de 4 bits, indicando a posição relativa dos pontos em relação as retas suportes de cada lado. A codificação do posicionamento relativo é feita com base em um simples teste nas coordenadas x_{min}, x_{max}, y_{min} e y_{max} da janela e, portanto, ele funcionará apenas se os lados estiverem alinhados com os eixos x e y.

6) Explique como efetuar uma rotação de θ graus sobre um objeto bidimensional, em torno de uma posição genérica (x_0,y_0). (1.0 ponto).

Anulada

Basta transladar o objeto de $(-x_0,-y_0)$, efetuar a rotação e desfazer a translação inicial , conforme apresentado na figura a seguir:



$$\begin{Bmatrix} x' \\ y' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \\ 1 \end{Bmatrix}$$

7) Considere dadas as cores c_0, c_1 e c_2, definidas em três vértices p_0, p_1 e p_2, de um triângulo no plano $p_0p_1p_2$, determine a cor em um ponto (x,y) no interior do triângulo (2.0 ponto).

A cor c(x,y) em um ponto p com coordenadas (x,y) interior a um triângulo de vértices p_0, p_1 e p_2 pode ser determinada via interpolação através de coordenadas baricêntricas.

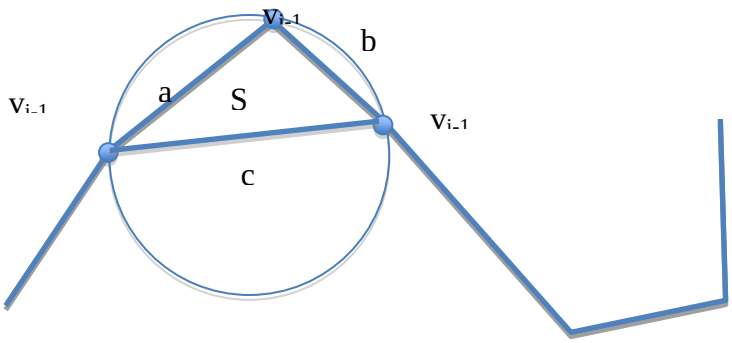
Sejam λ_0, λ_1 e λ_2, onde $\lambda_0+\lambda_1+\lambda_2 =1$, as coordenadas baricêntricas de p. A cor c(x,y) em um ponto (x,y) podem ser dadas por $c(x,y) = \lambda_0c_0+\lambda_1c_1+\lambda_2c_2$, onde

$$l_0 = \frac{\begin{vmatrix} x & p_1.x & p_2.x \\ y & p_1.y & p_2.y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} p_0.x & p_1.x & p_2.x \\ p_0.y & p_1.y & p_2.y \\ 1 & 1 & 1 \end{vmatrix}}, \quad l_1 = \frac{\begin{vmatrix} p_0.x & x & p_2.x \\ p_0.y & y & p_2.y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} p_0.x & p_1.x & p_2.x \\ p_0.y & p_1.y & p_2.y \\ 1 & 1 & 1 \end{vmatrix}}, \quad l_2 = \frac{\begin{vmatrix} p_0.x & p_1.x & x \\ p_0.y & p_1.y & y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} p_0.x & p_1.x & p_2.x \\ p_0.y & p_1.y & p_2.y \\ 1 & 1 & 1 \end{vmatrix}}$$

8) Determine um modo de estimar a curvatura de uma curva poligonal (1.0 ponto).

Pode-se estimar a curvatura *k* de uma linha poligonal em cada vértice v_i, determinando o raio do círculo que passa por três pontos consecutivos v_{i-1}, v_i e v_{i+1} na linha poligonal. O raio R de tal círculo é dado pela fórmula

$R = \frac{abc}{S}$, onde S é área do triângulo e a estimativa da curvatura é dada pelo inverso do raio $\tilde{k} = \frac{1}{R}$



Referência: http://www.mpi-inf.mpg.de/~ag4-gm/handouts/06gm_curves.pdf

Solução do problema de rasterização de polígonos

Código baseado em Donald Hearn, M. Pauline Baker. Computer Graphics, C Version (2nd Edition). Prentice Hall; 2 Sub edition (May 24, 1996), pg. 117.

Uma referência para o algoritmo implementado abaixo pode ser encontrada em http://www.ugrad.cs.ubc.ca/~cs314/notes/scanconv.html (acessado em 11/03/2013).

Referência para o uso de PBO pode ser obtida em http://www.songho.ca/opengl/gl_pbo.html (acessado em 11/03/2013).

scanconversion.h

```
#pragma once


typedef struct tEdge {
   int yUpper;
   float xIntersect, dxPerScan;
   struct tEdge * next;}Edge;

typedef struct tdcPt{
   int x;
   int y;
}dcPt;

void scanFill (int cnt, dcPt * pts, int width, int height, unsigned char * image);
```

scanconversion.cpp

```
#include <stdlib.h>

#include "scanconversion.h"

/* Insere aresta em uma lista em ordem crescente do campo xIntersect */
void insertEdge (Edge * list, Edge * edge)
{
   Edge *p, *q = list;

   p = q->next;

   while (p != NULL){
      if (edge->xIntersect < p->xIntersect)
                p = NULL;
             else {
                    q = p;
                    p = p->next;
             }
   }
   edge->next = q->next;
   q->next = edge;
}


/* Para cada í-ndice, retorna a coordenada y da próxima linha não horizontal */
int yNext (int k, int cnt, dcPt * pts){
       int j;

       if ((k+1) > (cnt-1))
              j = 0;
       else
              j=k+1;

       while (pts[k].y == pts[j].y)
          if ((j+1)>(cnt-1))
             j=0;
          else
                j++;

       return (pts[j].y);
}


/*Armazena a menor coordenada y e o inverso da inclinação de cada aresta. Ajusta e
armazena a
 coordenada y superior para arestas que são o membro inferior de uma par de arestas
monotonicamente
 crescentes ou decrescentes */

void makeEdgeRec(dcPt lower, dcPt upper, int yComp, Edge * edge, Edge * edges[]){

   edge->dxPerScan = (float)( upper.x - lower.x) / (upper.y - lower.y);

   edge->xIntersect= lower.x;
```

```
       if (upper.y < yComp)
      edge->yUpper = upper.y - 1;
   else
      edge->yUpper = upper.y;

       insertEdge (edges[lower.y] , edge);
}

void buildEdgeList (int cnt, dcPt * pts, Edge * edges[]){

   Edge * edge;
   dcPt v1, v2;

   int i, yPrev = pts[cnt - 2].y;

   v1.x = pts[cnt-1].x;
   v1.y = pts[cnt-1].y;

   for (i=0; i<cnt; i++) {
     v2 = pts[i];
     if (v1.y != v2.y){ /* segmento nÃ£o horizontal */

              edge = (Edge *) malloc (sizeof (Edge));

        if (v1.y < v2.y) /* aresta em sentido ascendente */
          makeEdgeRec (v1, v2, yNext (i, cnt, pts), edge, edges) ;
        else /* aresta em sentido descendente */
             makeEdgeRec (v2, v1 , yPrev, edge, edges);
        }
        yPrev = v1.y;
     v1 = v2;
   }
}

void buildActiveList (int scan, Edge * active, Edge * edges[]){

   Edge *p, *q;

   p = edges[scan]->next;

   while (p) {
      q = p->next;
      insertEdge (active, p);
      p=q;
   }
}

void setPixel(int i, int scan, int width, int height, unsigned char * image){
       image[3*(width*scan+i)+0]=255;
       image[3*(width*scan+i)+1]=255;
       image[3*(width*scan+i)+2]=255;
}

void fillScan(int scan, Edge * active, int width,int height, unsigned char * image){
   Edge *p1, *p2;
   int i;

   p1 = active->next;
```

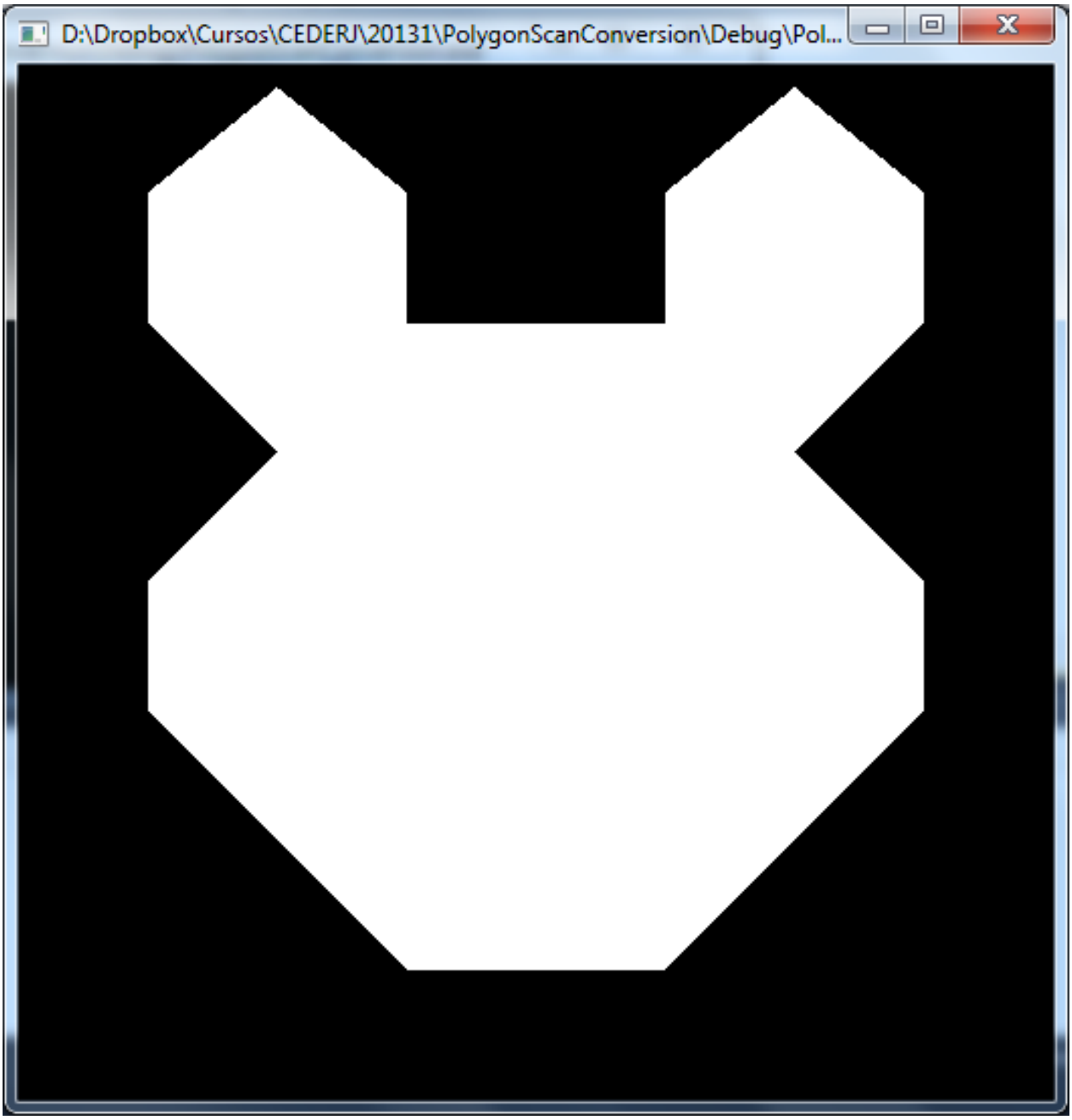
```
   while (p1){
      p2 = p1->next;
         for (i=p1->xIntersect; i<p2->xIntersect; i++)
        setPixel((int)i, scan, width, height, image);
      p1 = p2->next;
   }
}

void deleteAfter (Edge * q){
   Edge * p = q->next;

   q->next = p->next;
   free (p);
}

/* Deleta arestas completadas. Atualiza o campo xIntersect para as outras */
void updateActiveList(int scan, Edge * active){
   Edge * q = active, *p = active->next;

   while (p)
      if (scan >= p->yUpper){
         p = p->next;
               deleteAfter(q);
         }
         else {
         p->xIntersect = p->xIntersect + p->dxPerScan;
         p = p->next;
         }

}

void resortActiveList (Edge * active){

   Edge *q, *p = active->next;
   active->next = NULL;

   while (p){
      q = p->next;
      insertEdge (active, p);
         p = q;
   }
}

/* Rotina que faz a rasterização do poligono armazenado em pts*/
/* Os pixels são acesos na imagem image com widthxheight pixels */

void scanFill (int cnt, dcPt * pts, int width, int height, unsigned char * image){

   Edge ** edges, /* Array de listas encadeadas de arestas.
                             /* Existe uma potencial lista para cada linha da imagem*/
            *active; /* Lista encadeada de arestas ativas */

   int i, scan;

   edges = (Edge**)malloc(height*sizeof(Edge*));

   /*Aloca uma lista de arestas por linha da imagem */
   /*Todas as listas neste código são lista simplesmente*/
```

```
   /*encadeadas com nó cabeça. Logo sempre tem pelo menos um nó */

   for (i=0; i<height; i++){
     edges[i] = (Edge *)malloc(sizeof(Edge));
     edges[i]->next =  NULL;
   }

   /* Monta a lista de arestas */
   buildEdgeList (cnt, pts, edges);

   /* Aloca a lista de arestas ativas*/
   /* As arestas ativas são as que de fato serão */
   /* rasterizadas em uma scanline, podendo não haver nenhuma ativa*/
   active = (Edge *)malloc(sizeof(Edge));
   active->next = NULL;

   for (scan=0; scan<height; scan++){
         /* Monta a lista de arestas ativas para um determinada scan line */
      buildActiveList (scan, active, edges);
         /* Se a lista de arestas ativas não for vazia.*/
         /* Observe que desconsideramos o nó cabeça*/
         if (active->next){
                /* Preenche os pixels entre pares de aresta ativas*/
         fillScan(scan, active, width, height, image);
               /* Atualiza a lista de arestas ativas */
         updateActiveList(scan, active) ;
               /* reordena a lista de arestas ativas */
         resortActiveList(active);
         }
   }
}
```

main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include <memory.h>
#include "glext.h"

#include "scanconversion.h"

#define BUFFER_OFFSET(bytes) ((GLubyte*) NULL + (bytes))


#define imageWidth 512
#define imageHeight 512

static GLdouble zoomFactor = 1.0;
static GLint height;
static GLuint pixelBuffer;

dcPt polygon[20] =
{{192,64},{128,128},{64,192},{64,256},{128,320},{64,384},{64,448},{128,500},
{192,448},{192,384},{320,384},{320,448},{384,500},{448,448},{448,384},{384,320},
                    {448,256},{448,192},{384,128},{320,64}};


/* Este código usa Pixel Buffer Objects para escrever diretamente na memória da GPU */
/* A rasterização é feita em uma área de memória na GPU, o PBO, que depois é transferido
*/
/* para o framebuffer com glDrawPixels */

void init(void)
{
   glewInit();
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   /* Cria um pixel buffer object e o vincula ao rendering pipeline */
   glGenBuffers(1, &pixelBuffer);
   glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pixelBuffer);
    /* Aloca um buffer object do tamanho da imagem desejada */
   glBufferDataARB(GL_PIXEL_UNPACK_BUFFER_ARB, 3*imageWidth*imageHeight, 0,
GL_STREAM_DRAW_ARB);

}



void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glRasterPos2i(0, 0);
   /* Mapeia o buffer object para a memória do cliente (CPU)*/
   GLubyte* ptr = (GLubyte*)glMapBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB,GL_WRITE_ONLY_ARB);
   if(ptr)
   {
          /* Limpa a imagem com 0 (cor preta) */
```

```
         memset(ptr,0,3*imageWidth*imageHeight);
       /* Faz a rasterização direto no pixel buffer object */
      scanFill (20, polygon, imageWidth, imageHeight, ptr);
      glUnmapBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB); // Termina o mapeamento
   }
   /* Transfere os dados do PBO para o framebuffer */
   glDrawPixels(imageWidth, imageHeight, GL_RGB, GL_UNSIGNED_BYTE, BUFFER_OFFSET(0));
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   glOrtho(0, w, 0, h, -1.0, 1.0);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void finish(){
       /* Libera o PBO */
       glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, 0);
}


int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(imageHeight, imageWidth);
   glutInitWindowPosition(imageHeight, imageWidth);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
   finish();
   return 0;
}
```


D:\Dropbox\Cursos\CEDERJ\20131\PolygonScanConversion\Debug\Pol...