```
// Solução possível para o efólio-B

import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
import com.sun.opengl.util.GLUT;

public class efolioB extends J1_4_Line{
    GLU glu = new GLU(); // interface para a biblioteca GLU

    // Definição de palete de cores
    float[] colorBlack  = {0.0f,0.0f,0.0f,1.0f};
    float[] colorCyan   = {0.0f,0.5f,0.5f,1.0f};
    float[] colorWhite  = {1.0f,1.0f,1.0f,1.0f};
    float[] colorGray   = {0.6f,0.6f,0.6f,1.0f};
    float[] colorGreen   = {0.0f,1.0f,0.0f,1.0f};
    float[] colorRed    = {1.0f,0.0f,0.0f,1.0f};
    float[] colorBlue   = {0.0f,0.0f,0.5f, 1.0f};
    float[] colorYellow = {1.0f,1.0f,0.0f,1.0f};
    float[] colorLightYellow = {.5f,.5f,0.0f,1.0f};
    float[] colorMagenta = {1.0f, 0.0f, 1.0f, 1.0f};

    // Tonalidades para a luz
    float blackish[] = {0.3f, 0.3f, 0.3f, 0.3f};
    float whitish[] = {0.8f, 0.8f, 0.8f, 1};

    // Localização de fontes e orientação do spot, localizações infinitas
    float position[] = {1, 1, 1, 0};
    float posLight1[] = { 1.0f, 1.0f, 1.0f, 0.0f };
    float posLight2[] = { 2.0f, 2.0f, 10.0f, 0.0f };
    float spotDirection[] = { -1.0f, -1.0f, 0.0f };

    // Controle de alternância entre ligar/desligar fontes
    boolean move = true;

    // Controle para rotação da fonte de luz 2
    float spin = 0.0f;

    public efolioB() {
        // utiliza o construtor da classe mãe e ativa duplo buffering
        capabilities.setDoubleBuffered(true);
    }

    public void init(GLAutoDrawable glDrawable) {

        super.init(glDrawable);

        // Ativa a luz e demais funcionalidades para sua boa operacionalização
        gl.glEnable(GL.GL_LIGHTING);
        gl.glEnable(GL.GL_NORMALIZE);
        gl.glEnable(GL.GL_CULL_FACE);

        // Parametriza a fonte de luz 0
        gl.glEnable(GL.GL_LIGHT0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, position,0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, whitish,0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, colorWhite,0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, colorWhite,0);
```

```
        // Parametriza a fonte de luz 1 - é um foco de luz
        gl.glEnable(GL.GL_LIGHT1 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_POSITION, posLight1, 0);
        gl.glLightf(GL.GL_LIGHT1, GL.GL_SPOT_CUTOFF, 60.0F);
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPOT_DIRECTION, spotDirection, 0 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_AMBIENT, colorGray, 0 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE, colorGray, 0 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPECULAR, colorWhite,0 );
        gl.glLightf(GL.GL_LIGHT1, GL.GL_CONSTANT_ATTENUATION, 0.2f );

        // Parametriza a fonte de luz 1 - é um foco de luz
        gl.glEnable(GL.GL_LIGHT2 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_POSITION, posLight2, 0);
        gl.glLightf(GL.GL_LIGHT2, GL.GL_SPOT_CUTOFF, 30.0F);
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_SPOT_DIRECTION, spotDirection, 0 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_AMBIENT, colorYellow, 0 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_DIFFUSE, colorYellow, 0 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_SPECULAR, colorWhite,0 );
        gl.glLightf(GL.GL_LIGHT2, GL.GL_CONSTANT_ATTENUATION, 0.2f );

        // Materiais para a iluminação padrão para todas
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_AMBIENT, colorBlack,0);
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_DIFFUSE, whitish,0);
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, colorWhite,0);
        gl.glMaterialf(GL.GL_FRONT, GL.GL_SHININESS, 100.0f);
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, colorBlack,0);
    }

    public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {

        // Define o view port, inicializa a pilha de matrizes de projeção e define a mesma em
        // perspectiva
        gl.glViewport(0, 0, WIDTH, HEIGHT);
        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glLoadIdentity();
        glu.gluPerspective(45, WIDTH/HEIGHT,1.0, 800);
    }

    public void display(GLAutoDrawable drawable) {

        // Ativa buffers para gestão de cor e redesenho de profundidade
        gl.glClear(GL.GL_COLOR_BUFFER_BIT|GL.GL_DEPTH_BUFFER_BIT);

        // Modo de sobreamento é o smooth
        gl.glShadeModel(GL.GL_SMOOTH);

        // Liga e desliga fontes 0 e 1
        if (move)
        {
            gl.glDisable(GL.GL_LIGHT0);
            gl.glEnable(GL.GL_LIGHT1);
            drawScene();
        }
        else {
            gl.glDisable(GL.GL_LIGHT1);
            gl.glEnable(GL.GL_LIGHT0);
```

```
            drawScene();
        }

        // Garante a pilha da model-view estar sem transformações indesejáveis acumuladas
        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glLoadIdentity();

        // Especifica posição do observador da cena e do ponto para o qual olha
        glu.gluLookAt(0,70,200, 3*WIDTH/6, 3*HEIGHT/6, 0.0f, 0,0,1);

        // torna refrescamento de desenho mais lento
        try {
            Thread.sleep(100);
        } catch (Exception ignore) {
            }

    }

    // Desenha a cena
    public void drawScene() {
        if (spin > 360.0f) spin = 0.0f;
        ++spin;
        move = !move;
        // Desenha o plano em verde, posicionando e definindo material
        gl.glPushMatrix();
            gl.glTranslatef(3*WIDTH/6 - 30, 3*HEIGHT/6 - 20, 0.0f);
            gl.glScalef(350.0f, 250.0f, 2.0f);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, colorGreen, 0);
            glut.glutSolidCube(1);
        gl.glPopMatrix();

        // Desenha o cone em vermelho, posicionando e definindo material
        gl.glPushMatrix();
            gl.glTranslatef(3*WIDTH/6, 3*HEIGHT/6, 10.0f);
            gl.glScalef(40.0f, 40.0f, 40.0f);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, colorRed,0);
            glut.glutSolidCone(1, 1, 20, 20);
        gl.glPopMatrix();

        // Desenha a esfera em amarelo, posicionando e definindo material
        gl.glPushMatrix();
            gl.glTranslatef(3*WIDTH/6 + 150, 3*HEIGHT/6, 15.0f);
            gl.glScalef(40.0f, 40.0f, 40.0f);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, colorLightYellow,0);
            glut.glutSolidSphere(1, 20, 20);
        gl.glPopMatrix();

        // Desenha o cilindro em magenta, posicionando e definindo material
        gl.glPushMatrix();
            gl.glTranslatef(3*WIDTH/6 - 150, 3*HEIGHT/6, 15.0f);
            gl.glScalef(40.0f, 40.0f, 40.0f);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, colorCyan, 0);
            glut.glutSolidCylinder(1, 1, 20, 20);
        gl.glPopMatrix();

        // Desenha uma esfera acima do plano com os demais objetos e associa uma fonte de
        luz móvel
```

```
        gl.glPushMatrix ();
            gl.glTranslated (3*WIDTH/6 + 300, 3*HEIGHT/6, 100.0f);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, colorMagenta, 0);
            gl.glScalef(20.0f, 20.0f, 20.0f);
            gl.glPushMatrix ();
                gl.glRotatef (spin, 1.0f, 0.0f, 0.0f);
                gl.glLightfv(GL.GL_LIGHT2, GL.GL_POSITION, posLight2, 0);
                gl.glTranslated (0.0, 0.0, 1.5);
                gl.glDisable (GL.GL_LIGHTING);
                gl.glColor3d (0.0, 1.0, 1.0);
                glut.glutWireCube (1); // indicador da posição da luz
                gl.glEnable (GL.GL_LIGHTING);
           gl.glPopMatrix ();
           glut.glutSolidSphere(1, 20, 20);
        gl.glPopMatrix ();

        gl.glFlush();
    }

    public static void main(String[] args) {
         efolioB f = new efolioB();

         f.setTitle("Efólio-B");
         f.setSize(WIDTH, HEIGHT);
         f.setVisible(true);
    }

}


package uab;

import java.util.Random;
import javax.media.opengl.*;

import com.sun.opengl.util.GLUT;

import uab.Peixe;

public class Cardume {
  int numero; // atributos da classe
  Peixe [] cardume; // array de Peixes e nº total
  Random gerador; // geração de valores aleatórios
  float pos []; // posição inicial do cardume
  float [] ar = {0.9f,0.9f,0.9f, 0.1f};
  GL gl; // interface to OpenGL
  GLUT glut; // interface para a biblioteca GLUT

// Construtor
  public Cardume (int n, GL gl, GLUT glut) {
    numero = n;
    cardume = new Peixe[numero]; // inicialização
    float tamanho;
    gerador = new Random(); // inicializa gerador aleatório
    pos = new float[3];
    pos[0] = gerador.nextFloat() * 100.0f;
    pos[1] = gerador.nextFloat() * 100.0f;
```

```
    pos[2] = gerador.nextFloat() * 10.0f;

    this.gl = gl;
    this.glut = glut;

 // Preenche cada objeto Peixe do array
    for (int i = 0; i < n; ++ i) {
 // cor de tamanho são aleatórios
      float [] cor = {gerador.nextInt(255)/255.0f, gerador.nextInt(255)/255.0f,
      gerador.nextInt(255)/255.0f, 1.0f};
      tamanho = 1.0f + gerador.nextInt(2)/1.0f; // altera o tamanho
      cardume[i] =  new Peixe(cor, tamanho, gl, glut, pos);
      pos[0] = gerador.nextFloat() * 100.0f; // altera posição matriz para não haver
      sobreposições
      pos[1] = gerador.nextFloat() * 500.0f;
      pos[2] = pos[2] + 100.0f;
    }
  }
   // Desenha os peixes um a um
   void desenha(float x, float y){

      for (int i = 0; i < numero; ++ i) {
          float oscila = gerador.nextFloat() * 10.0f; // faz com que os peixes oscilem
          cardume[i].desenha(x, y + oscila);
      }
   }

// Desenha bolhas de ar quando é premida a tecla SHIFT
    void bolhas(float x, float y) {
        gl.glPushMatrix();
            gl.glTranslatef(x, y, 0.0f);
            gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, ar, 0);
            glut.glutSolidSphere(10.0f, 10, 10);
        gl.glPopMatrix();

        gl.glFlush();
    }
}


package uab;

import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Random;

import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCanvas;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.glu.GLU;
```

```
import com.sun.opengl.util.Animator;
import com.sun.opengl.util.GLUT;

import uab.Cardume;

@SuppressWarnings("serial")
public class Aquario extends Frame implements GLEventListener, MouseMotionListener {

    // Gestão da janela e eventos OpenGL
        static GLCanvas canvas; // drawable in a frame
        static GLCapabilities capabilities;
        static GL gl; // interface to OpenGL
        static GLUT glut;
        static GLU glu;
        static int HEIGHT = 600, WIDTH = 600;
        static Animator animator; // drive display() in a loop

    // Atributos da classe
        Cardume cardume;
        float offset = 50.0f;
        float scaleVal = 35.0f;
        float anguloInc = (float) Math.PI/28;
        float angulo = 0.0f;
        float x, y, z, xRato, yRato;
        float h, w;
        int n;
        int contador;
        boolean bolha;
        Random gerador; // geração de valores aleatórios

    // Atributos para iluminação ambiente
        // Definição de palete de cores
        float[] preto  = {0.0f,0.0f,0.0f,1.0f};
        float[] branco  = {1.0f,1.0f,1.0f,1.0f};
        float[] cinza   = {0.6f,0.6f,0.6f,1.0f};
        float[] amarelo = {1.0f,1.0f,0.0f,1.0f};


    // Tonalidades para a luz
        float escurecido[] = {0.3f, 0.3f, 0.3f, 0.3f};
        float esbranquicado[] = {0.8f, 0.8f, 0.8f, 1};

    // Localização de fontes e orientação do spot, localizações infinitas
        float pos[] = {1, 1, -800, 0};
        float posFonte1[] = { 1.0f, 1.0f, 1.0f, 0.0f };
        float posFonte2[] = { 2.0f, 2.0f, 10.0f, 0.0f };
        float spotDirecao[] = { -1.0f, -1.0f, 0.0f };

    // Controle de alternância entre ligar/desligar fontes
        boolean alterna = true;

    public Aquario(int n) {
// utiliza o construtor da classe mãe e ativa duplo buffering
        capabilities = new GLCapabilities();
        canvas = new GLCanvas();
        capabilities.setDoubleBuffered(false);
        canvas.addGLEventListener(this);
```

```
        canvas.addMouseMotionListener(this);
        add(canvas, BorderLayout.CENTER);
        gl = canvas.getGL();
        glut = new GLUT();
        glu = new GLU();

        angulo = 0;
        h = 500.0f; w = 500.0f;
        x = 0;
        y = h/2;
        z = 1.0f;
        this.n = n;
        contador = 0;
        bolha = false;
        gerador = new Random(); // inicializa gerador aleatório

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                animator.stop(); // stop animation
                System.exit(0);
            }
        });
    }

    public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
        WIDTH = width;
        HEIGHT = height;
        gl.glViewport(0, 0, WIDTH, HEIGHT);
        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glOrtho(0, width, -1.0 * height*1.5, height*1.5, -1000, 1000.0);
        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glLoadIdentity();

/*      gl.glMatrixMode(GL.GL_PROJECTION);
         gl.glLoadIdentity();
         glu.gluPerspective(45.0f, h, 600.0, -600);
         gl.glMatrixMode(GL.GL_MODELVIEW);
         gl.glLoadIdentity();*/

    }

    // Invocado sempre para refrescar a cena
    public void display(GLAutoDrawable drawable) {
// Define o azul como cor de fundo
        if (contador <= 10) gl.glClearColor(0.8f, 0.89f, 1.0f, 1.0f);
        else if (contador <= 20) gl.glClearColor(0.6f, 0.8f, 1.0f, 1.0f);
            else if (contador <= 30) gl.glClearColor(0.4f, 0.69f, 1.0f, 1.0f);
                else contador = 0;
        ++contador;

        gl.glClear (GL.GL_COLOR_BUFFER_BIT|GL.GL_DEPTH_BUFFER_BIT);
        gl.glLoadIdentity();
    gl.glTranslated(0, 0, 500);

// Faz com que o movimento dos peixes seja segundo uma onda sinusoidal
        y = (float) (offset + (Math.sin(angulo) * scaleVal));
```

```
        if (x <= w) x += 5;
        else x = 0;

        angulo = angulo + anguloInc;

// Liga e desliga fontes 0 e 1
        if (alterna)
        {
            gl.glDisable(GL.GL_LIGHT0);
            gl.glEnable(GL.GL_LIGHT1);
            cardume.desenha(x, y); // redesenha o cardume em nova posição
        }
        else {
            gl.glDisable(GL.GL_LIGHT1);
            gl.glEnable(GL.GL_LIGHT0);
            cardume.desenha(x, y); // redesenha o cardume em nova posição
        }

        if (bolha) { // Se foi premida a tecla SHIFT, gera bolhas de ar...
            for (int i = 0; i <= 10; ++i) {
                cardume.bolhas(gerador.nextFloat() * xRato, gerador.nextFloat() * yRato);
            }
        }

        alterna = !alterna;

// torna refrescamento de desenho mais lento
        try {
            Thread.sleep(20);
        } catch (Exception ignore) {
            }
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        // TODO Auto-generated method stub

    }

// Qualquer movimento do rato, é capturado
    public void mouseMoved(MouseEvent e) {
        xRato = e.getX();
        yRato = e.getY();
        if (e.isShiftDown()) bolha = true;
        else bolha = false;
    }


    @Override
    public void displayChanged(GLAutoDrawable arg0, boolean arg1, boolean arg2) {
        // TODO Auto-generated method stub

    }

// Inicialização da máquina do OpenGL
    public void init(GLAutoDrawable arg0) {
```

```
// display() num ciclo
        animator = new Animator(canvas);
        animator.start(); // thread iniciado

// Teste de profundidade e preenchimento de polígonos
        gl.glEnable (GL.GL_DEPTH_TEST);
        gl.glPolygonMode(GL.GL_FRONT, GL.GL_FILL);

// Ativa a luz e demais funcionalidades para sua boa operacionalização
        gl.glEnable(GL.GL_LIGHTING);
        gl.glEnable(GL.GL_NORMALIZE);
        gl.glEnable(GL.GL_CULL_FACE);

// Parametriza a fonte de luz 0
        gl.glEnable(GL.GL_LIGHT0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, pos,0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, esbranquicado,0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, branco,0);
        gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, branco,0);

// Parametriza a fonte de luz 1 - é um foco de luz
        gl.glEnable(GL.GL_LIGHT1 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_POSITION, posFonte1, 0);
        gl.glLightf(GL.GL_LIGHT1, GL.GL_SPOT_CUTOFF, 60.0F);
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPOT_DIRECTION, spotDirecao, 0 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_AMBIENT, cinza, 0 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE, cinza, 0 );
        gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPECULAR, branco,0 );
        gl.glLightf(GL.GL_LIGHT1, GL.GL_CONSTANT_ATTENUATION, 0.2f );

// Parametriza a fonte de luz 2
        gl.glEnable(GL.GL_LIGHT2 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_POSITION, posFonte2, 0);
        gl.glLightf(GL.GL_LIGHT2, GL.GL_SPOT_CUTOFF, 30.0F);
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_SPOT_DIRECTION, spotDirecao, 0 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_AMBIENT, amarelo, 0 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_DIFFUSE, amarelo, 0 );
        gl.glLightfv(GL.GL_LIGHT2, GL.GL_SPECULAR, branco,0 );
        gl.glLightf(GL.GL_LIGHT2, GL.GL_CONSTANT_ATTENUATION, 0.2f );

// Materiais para a iluminação padrão para todas
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_AMBIENT, preto,0);
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_DIFFUSE, esbranquicado,0);
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_SPECULAR, branco,0);
        gl.glMaterialf(GL.GL_FRONT, GL.GL_SHININESS, 100.0f);
        gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, preto,0);

// Modo de sobreamento é o smooth
        gl.glShadeModel(GL.GL_SMOOTH);

        cardume = new Cardume(n, gl, glut); // Cria um cardume de peixes
    }

    public static void main(String[] args) {
        Aquario efolio = new Aquario(4);

        efolio.setTitle("Efólio-B");
```

```
        efolio.setSize(WIDTH, HEIGHT);
        efolio.setVisible(true);
    }
}
```

```
package uab;
import java.util.Random;

import javax.media.opengl.GL;

import com.sun.opengl.util.GLUT;

public class Peixe {
// Atributos de cada peixe
      float [] corCorpo = new float[4]; //colorido
      float [] corOlhos  = {0.0f,0.0f,0.0f,1.0f};
      float [] corRabo = {192.0f/255.0f, 192.0f/255.0f, 192.0f/255.0f, 1.0f};
      float [] normal = new float[3];
      float escala; // tamanho
      Random gerador; // geração de valores aleatórios
      int contador;
      GL gl; // interface to OpenGL
      GLUT glut; // interface para a biblioteca GLUT
      float posX, posY, posZ;

      public Peixe (float[] corC, float s, GL gl, GLUT glut, float [] pos) {
        corCorpo = corC;
        escala = s;
        gerador = new Random(); // inicializa gerador aleatório
        this.gl = gl;
        this.glut = glut;
// executa o desenho passando parâmetros aleatórios para a posição
        posX = pos[0];
        posY = pos[1];
        posZ = pos[2];

// Calcula a normal para o rabo do peixe
        float [] v1 = {-50, 0, 0};
        float [] v2 = {-100, 50, 0};
        float [] v3 = {-100, -50, 0};
        normal = calculaNormal(v1, v2, v3);
      }

// Método que desenha o corpo do peixe
      void desenha(float x, float y) {

        gl.glPushMatrix(); // push e pop para isolar efeitos de transformação
            gl.glTranslatef(x + posX, y + posY, posZ); // a posição y oscila ente +1 e -1
            gl.glPushMatrix(); // desenha o corpo do peixe
            //  gl.glScalef(escala + 0.5f, escala, escala);
                gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, corCorpo, 0);
                glut.glutSolidSphere((double)100, 50, 50);
            gl.glPopMatrix();
            gl.glPushMatrix(); // desenha os 2 olhos
```

```
                gl.glTranslatef(60.0f, 0.0f, 80.0f);
                gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, corOlhos, 0);
                glut.glutSolidSphere(10.0f, 10, 10);
                gl.glTranslatef(0.0f, 0.0f, -160.0f);
                glut.glutSolidSphere(10.0f, 10, 10);
            gl.glPopMatrix();
            gl.glPushMatrix();
                gl.glTranslatef(-50.0f, 0.0f, 0.0f);
                gl.glMaterialfv(GL.GL_FRONT, GL.GL_EMISSION, corRabo, 0);
                gl.glBegin(GL.GL_POLYGON); // desenha o rabo do peixe
                  gl.glVertex3f(-50, 0, 0);
                  gl.glVertex3f(-100, 50, 0);
                  gl.glVertex3f(-100, -50, 0);
                  gl.glNormal3f(normal[0], normal[1], normal[2]);
                gl.glEnd();
            gl.glPopMatrix();
        gl.glPopMatrix();
        gl.glFlush();
      }

// Calcula a normal de um plano
      float [] calculaNormal(float [] v1, float [] v2, float [] v3){
        float [] vn = new float[3];
        float qx, qy, qz, px, py, pz;

        qx = v2[0]-v1[0];
        qy = v2[1]-v1[1];
        qz = v2[2]-v1[2];
        px = v3[0]-v1[0];
        py = v3[1]-v1[1];
        pz = v3[2]-v1[2];

        vn[0] = py*qz - qz*qy;
        vn[1] = pz*qx - qx*qz;
        vn[2] = px*qy - qy*qx;

        return vn;
      }
}
```