OpenGL 总结

Table of Contents

1. OpenGL 概述

OpenGL 来自于一个称为 GL(Graphics Library 的缩写)的接口，因为简单易用和功能强大得到了最广泛的认可。OpenGL 能为多种图形硬件所支持，使用 OpenGL 编写的程序可被一直到任何支持该接口的计算机。几乎所有的计算机和操作系统都有 OpenGL 的相应实现，这些实现方式充分发挥了目前最先进的硬件加速功能。此外，OpenGL 还具有高度稳定性，保证了程序具有很长的生命期。

OpenGL 包含 200 多个函数。这些函数可分为图元函数（指定要生成屏幕图像的图元，如多边形，位图等），属性函数（负责控制图元的外观，如颜色等），观察函数（指定摄像机属性，如位置和朝向等），控制函数（控制使用 OpenGL 的特性，如光照等），查询函数（查询 OpenGL 的状态值），输入与窗口函数（用于使用鼠标和键盘控制屏幕中的窗口）。这些函数包括在三个库中，分别是 GL，GLU，GLUT 库。GL 库是 OpenGL 的核心库，GLU 库是使用 GL 库函数编写的新函数，GLUT 库主要包括输入与窗口函数。这些库中的函数大多都以库名为前缀，以参数个数和参数类型为后缀。

2. OpenGL 绘制的大概流程

step1: 定义 GLCanvas， 这给定绘制的画布
step2: 定义 GLContext，这给定绘制的工具
step3: 从 GLContext 可以得到类似画笔的 GL
利用 GL 绘制出各类图形，然后显示

实例代码:

```
GLData data = new GLData();
data.doubleBuffer = true; // 双缓冲技术，防止闪烁现象
GLCanvas canvas = new GLCanvas(          // step1:
        composite,                       // 画布所在的面板
        style,                           // 显示风格
        data);
GLContext context = GLDrawableFactory.  // step2:
        getFactory().
        createExternalGLContext();
GL gl = gontext.getGL();  // step3:
```

3. OpenGL 知识点

3.1 基本图元的绘制(点, 直线段, 多边形)

3.1.1 点

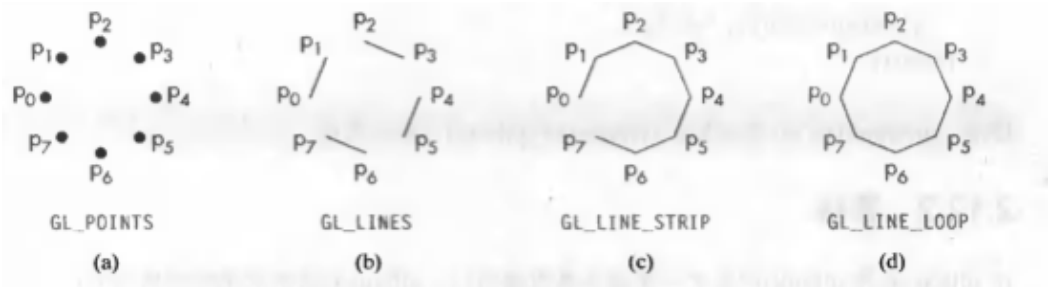
```
gl.glBegin(GL.GL_POINTS);
gl.glVertex3f(x,y,z);          // 可以绘制多次
gl.glEnd();
```

3.1.2 直线段

可选参数:

GL_LINES	- n 个点画 n/2 条线段
GL_LINE_STRIP	- n 个点画 n－1 条线段
GL_LINE_LOOP	- n 个点画 n 条首尾相连的线段

效果如下:

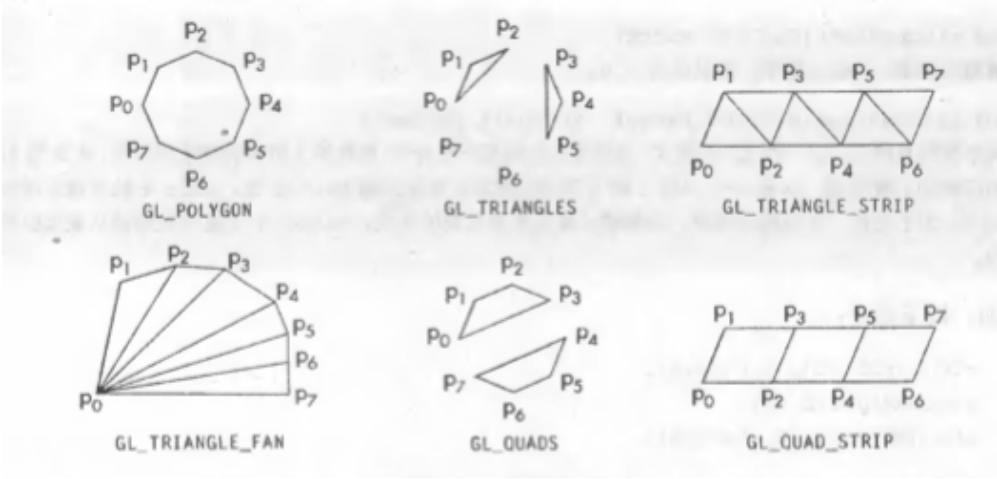


3.1.3 多边形(填充图元)

可选参数:

GL_POLYGON	- 多边形
GL_TRIANGLES	- n 个点画 n/3 个三角形
GL_TRIANGLE_STRIP	- n 个点画 n－2 个三角形
GL_TRIANGLE_FAN	- 除多边形外，每个点都与第一个点相连
GL_QUADS	- 每 4 个点定义一个四边形
GL_QUADS_STRIP	- 每 2 个点对之间画一个四边形

效果如下:



3.1.4 2D 矩形

```
gl.glRect(x1,y1,x2,y2);
```

3.2 常用设置

点尺寸: gl.glPointSize(size);

线宽: gl.glLineWidth(width);

颜色:

背景色:

```
gl.glClear(mode);            // 清除画布背景
gl.glClearColor(r,g,b,c);    // 设置背景色
```

前景色:

```
gl.glColor();
```

启用和禁用特性:

```
gl.glEnable(feature);
gl.glDisable(feature);
```

直线点画模式:

```
gl.glEnable(GL.GL_LINE_STIPPLE);
```

```
gl.glLineStipple(factor, pattern);
// pattern 的 01 串复制 factor 模式
```

多边形点画模式:

```
gl.glEnable(GL.GL_POLYGON_STIPPLE);
gl.glPolygonStipple(mask);
```

3.3 文本绘制

3.3.1 字符模式

位图字符: glut.glutBitmap(font, char);
笔画字符: glut.glutStroke(font, char);
font: 可以用大多数窗口系统所提供的字体,
这样程序可移植性降低
GLUT 自带字体:
GLUT_BITMAP_TIMES_ROMAN_10
GLUT_STROKE_MONO_ROMAN...

3.3.2 位置设定

```
gl.glRasterPos4d(x,y,z,w);
// 注意,这里坐标是自定义坐标系内的坐标
```

3.3.3 位图字符 V.S.笔画字符

位图字符绘制较快,不能进行缩放,用位矩阵实现
笔画字符存储空间大,绘制比较慢,用直线和曲线实现

3.4 GLU 二次曲面(球体, 圆柱体, 圆盘)

3.4.1 曲面对象的创建或删除

```
glu.gluNewQuadric();      // 创建新的二次曲面对象
glu.gluDeleteQuadric(GLUquadricObj);// 删除二次曲面对象 obj
```

3.4.2 绘制风格设定

```
glu.gluQuadricDrawStyle(
        GLUquadricObj obj,      // 曲面对象
        GLenum style            // 绘制风格:
        GLU_POINT, GLU_LINE, GLU_FILL, GLU_SILHOUETTE
        );
```

3.4.3 法线、纹理坐标生成

```
glu.gluQuadricNormals(
        GLUquadricObj obj,
        GLenum mode    // 法线模式:
        GL_NONE, GLU_FLAT, GLU_SMOOTH
        );
glu.gluQuadricTexture(
        GLUquadricObj obj,
        GLboolean mode // 是否有纹理坐标
        );
```

3.4.4 球体

```
glu.gluSphere(
        GLUquadricObj obj,
        GLdouble radius,        // 半径
```

```
        GLint slices,           // 经线行数
        GLint stacks            // 纬线行数
        );
```

3.4.5 画圆柱

```
glu.gluCylinder(
        GLUquadricObj obj,
        GLdouble base,          // 底面半径
        GLdouble top,           // 顶面半径
        GLdouble height,        // 高
        GLdouble slices,        // 模拟圆的多边形边数
        GLdouble stacks         // 圆环层数
        );
```

3.4.6 画圆盘

```
glu.gluDisk(
        GLUquadricObj obj,
        GLdouble inner,         // 内环半径
        GLdouble outer,         // 外环半径
        GLdouble slices,  // 模拟圆的多边形边数
        GLdouble rings          // 圆环层数
        );
```

3.4.7 画局部圆盘

```
glu.gluDisk(
        GLUquadricObj obj,
        GLdouble inner,         // 内环半径
        GLdouble outer,         // 外环半径
        GLdouble slices,        // 模拟圆的多边形边数
        GLdouble rings          // 圆环层数
        GLdouble start,         // 开始角度点
        GLdouble angle          // 移除角度长
        );
```

3.5 曲线的绘制

定义求值器

```
gl.glMap1f(entity, uo, u1, stride, order, data);
```

使用求值器

```
gl.glEnable(entity);
```

画曲线

```
方法 1:
gl.glMapGrid1f(num, uo, u1);                    // 区间[uo,u1]等分成 num 份
gl.glEvalMesh1(GL.GL_LINE, first, last);    //将所有点用线段连接
方法 2:
gl.glBegin(GL.GL_LINE_STRIP);
for (int i = 0; i <= num; ++i) {
        gl.glEvalCoord1f((float) i / (float) num);
} gl.glEnd();
```

3.6 观察设置

3.6.1 视口

采用画布的哪些范围

gl.glViewport(x,y,w,h);

3.6.2 坐标系与变换

由模型视图矩阵和投影矩阵决定

设置分三步：

step1. 指定修改的矩阵

step2. 将矩阵设为单位矩阵

step3. 修改当前矩阵

例如二维取景需要修改投影矩阵，如下：

```
gl.glMatrixMode(GL.GL_PROJECTION);
gl.glLoadIdentity();
GLU glu = new GLU();
glu.gluOrtho2D(xLow, xHigh, yLow, yHigh);
```

3.6.3 二维取景

在范围内的内容会被显示

```
glu.gluOrtho2D(left, right, bottom, top);
```

3.6.4 三维取景

```
gl.glOrtho(left, right, bottom, top, near, far);
```

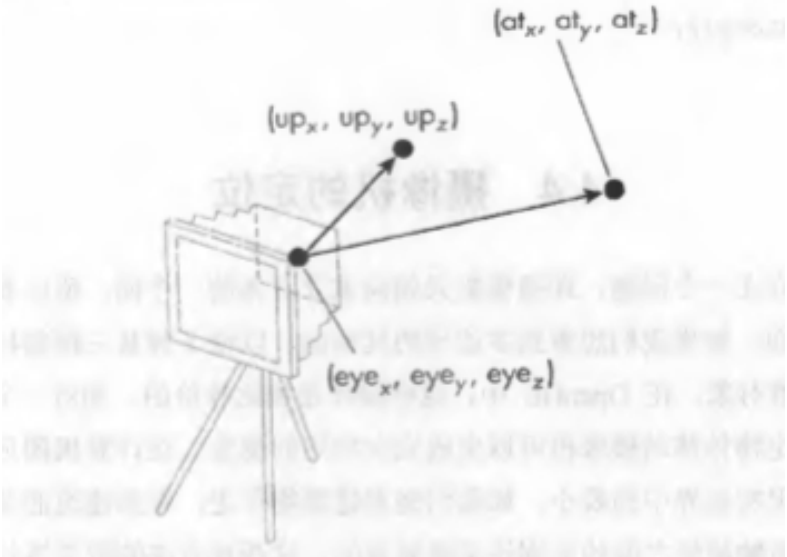
3.6.5 透视投影

```
gl.glMatrixMode(GL.GL_PROJECTION);
gl.glLoadIdentity();
GLU glu = new GLU();
glu.gluPerspective(fov, aspect, near, far);
// fov 是摄像机的夹角，aspect 是 width/height，near，far 是离摄像机点的距离
```

3.6.6 摄像机定位

```
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();
glu.gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);
```

其中，三个参数向量如下：



3.7 几何变换

3.7.1 概念

变换将顶点和向量映射到另一些顶点和向量
旋转和平移不会改变物体的基本性质(尺度和尺寸)，称为刚体变换
OpenGL 中有两类线性变换比较重要，仿射变换和射影变换
仿射变换包括：平移，旋转，比例变换
射影变换包括 2.6 中的透视变换

3.7.2 平移

```
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glTranslatef(dx, dy, dz); //  在坐标系中的度量的向量(dx,dy,dz)
```

3.7.3 旋转

```
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glRotatef(angle, dx, dy, dz);  // 旋转轴(dx,dy,dz)，旋转角度 angle
```

旋转角方向: 从旋转轴的正端往原点看，按逆时针旋转的角度为正

3.7.4 比例变换

```
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glScalef(sx,sy,sz);          // sx,sy,sz 为缩放因子
```

3.7.5 注意

由于每次定义的变换都通过自右乘作用于当前矩阵
所以变换的应用次序与其在程序中出现的次序相反

3.8 两类事件监听

通过给 GLCanvas 注册监听器，可以用鼠标和键盘控制图形的显示
鼠标事件：MouseMoveListener，从 MouseEvent 得到(x,y)坐标
键盘事件：KeyListener，从 KeyEvent 得到 keyCode 按键

4. 基于 SWT 与 OpenGL 小项目(曲线图，柱状图，直方图，扇形图)

4.1 项目设计

4.1.1 画板的设计

画板类(Paint)继承 Composite 类，这样它可以直接当 SWT 里 Composite 使用，里面组合了 OpenGL 绘制所需要基本工具（GLCanvas，GLContext 等），还有要绘制的内容 List<Drawable>。Drawable 接口如下：

```
public interface Drawable{
        void draw(GL gl);
}
```

将需要实现了 Drawable 的图表类加入到 List，然后调用画板类里的 draw()方法，在 Draw 方法里面会遍历 List，调用元素 Drawable 的 draw(GL gl)方法绘制所有需要显示的内容。画板的 draw 方法主要如下：

```
public void draw() {
```

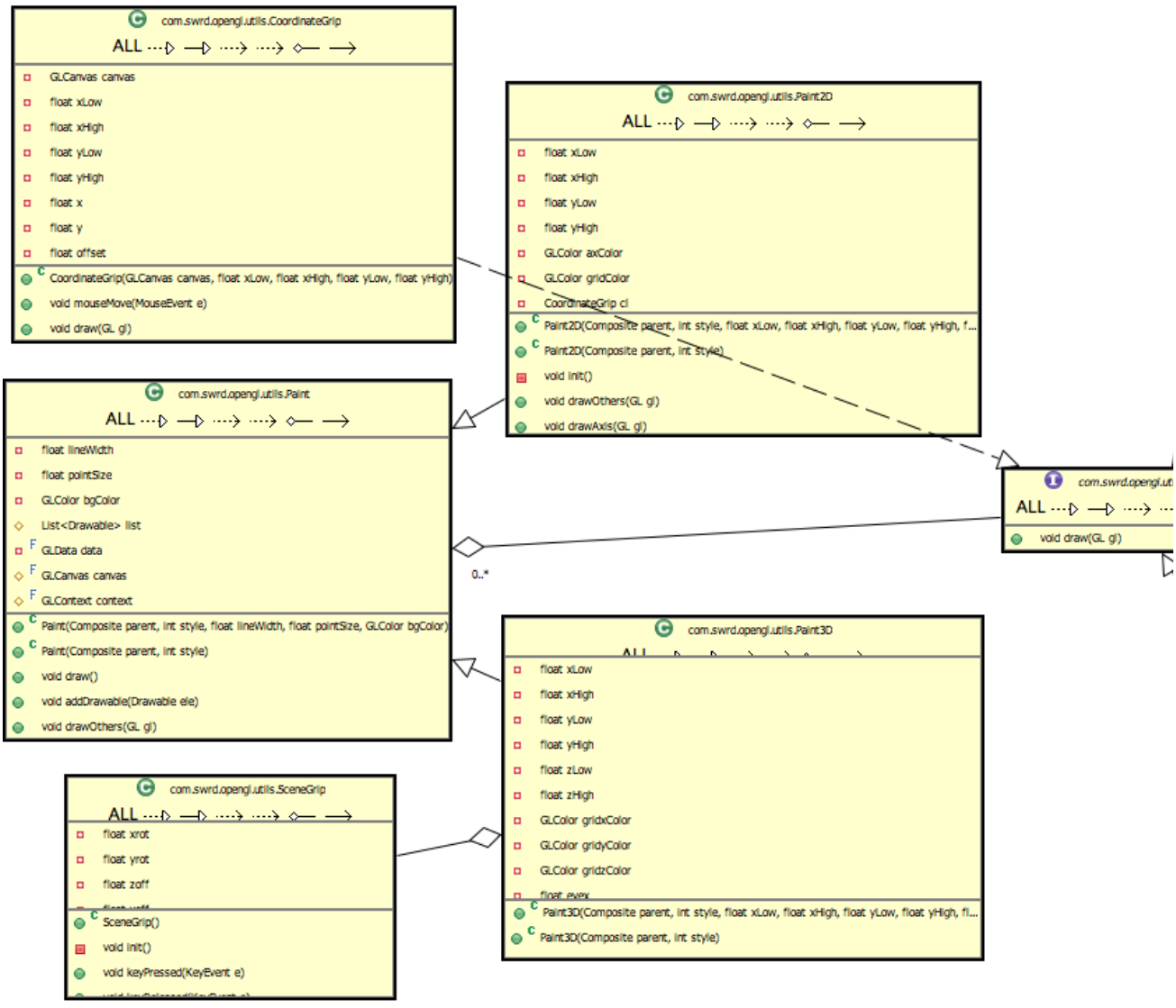
```
        GL gl = context.getGL();
        drawOthers(gl);
        for(Drawable obj : list) {
                obj.draw(gl);
        }
    }
```

画板类还提供了给子类绘制自己特定内容的 drawOthers(GL gl)方法。

2D 画板类(Paint2D)和 3D 画板类(Paint3D)继承画板类，增加了它们自己的特定内容（如坐标轴等），通过 drawOthers 方法进行绘制。

另外，给 2D 画板增加了获取鼠标事件，获取坐标值，给 3D 画板增加了键盘事件，转换视角。

它们的类图如下：

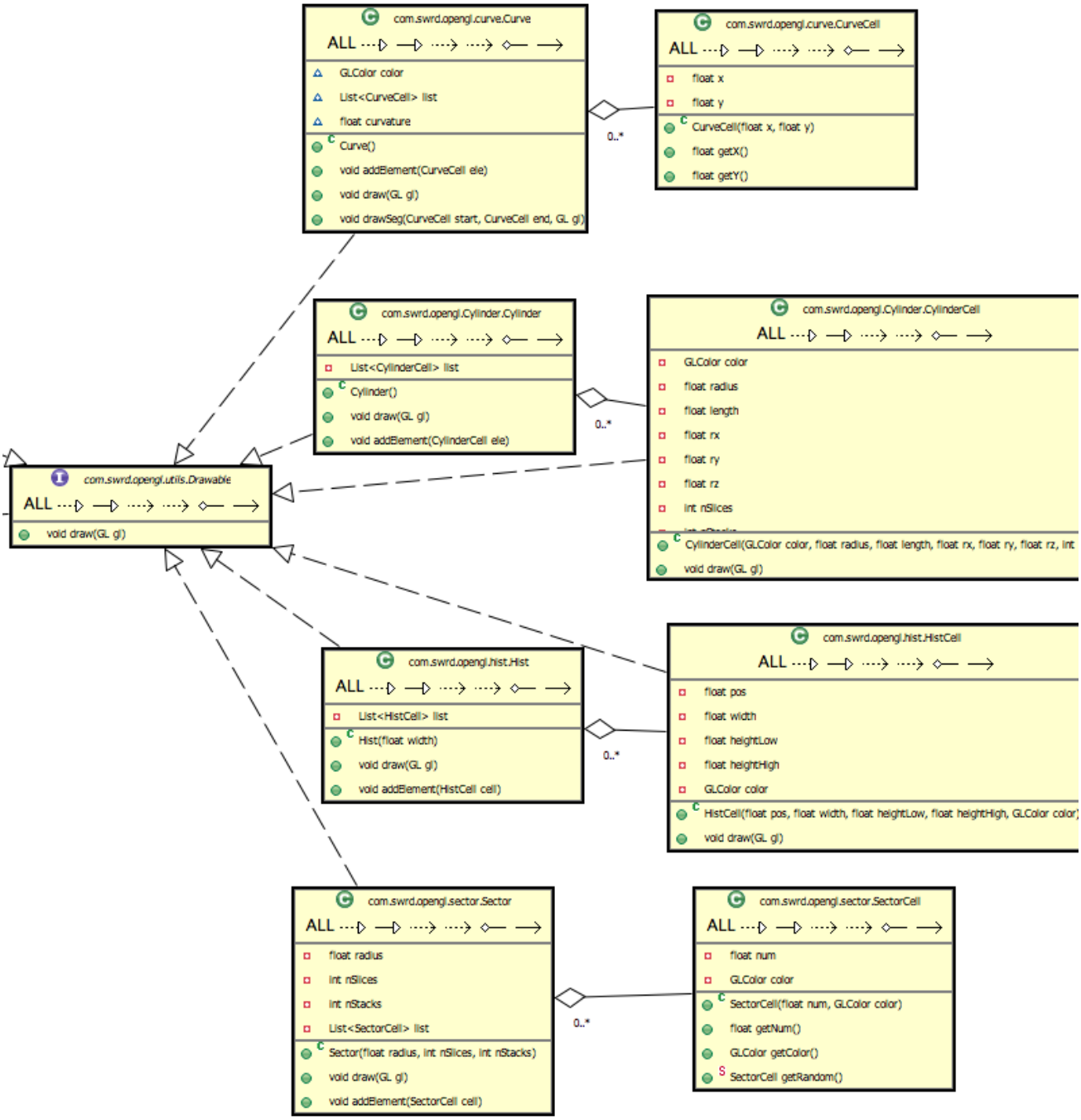


4.1.2 绘制内容的设计

图表均实现自定义的 Drawable 接口，实现各自的 draw(GL gl)方法。图表都有一个**Cell 的基本结构，如果能独立显示，**Cell 也实现 Drawable 接口，然后将图表的基本元素添加到 List<**Cell>里面。新建画板类之后，将图表添加到画板的 List<Drawable>列表内，通过画板的 draw()方法进行绘制。

四种图表的实现分别是：曲线图通过画连续两点间的贝塞耳曲线完成；柱状图通过圆柱体来画成；直方图通过填充矩形画成；扇形图通过局部圆盘画成。值得注意的是，扇形图需要在画之前归一化，这样才能确定各自的大小。

它们的类图如下：



4.2 运行结果

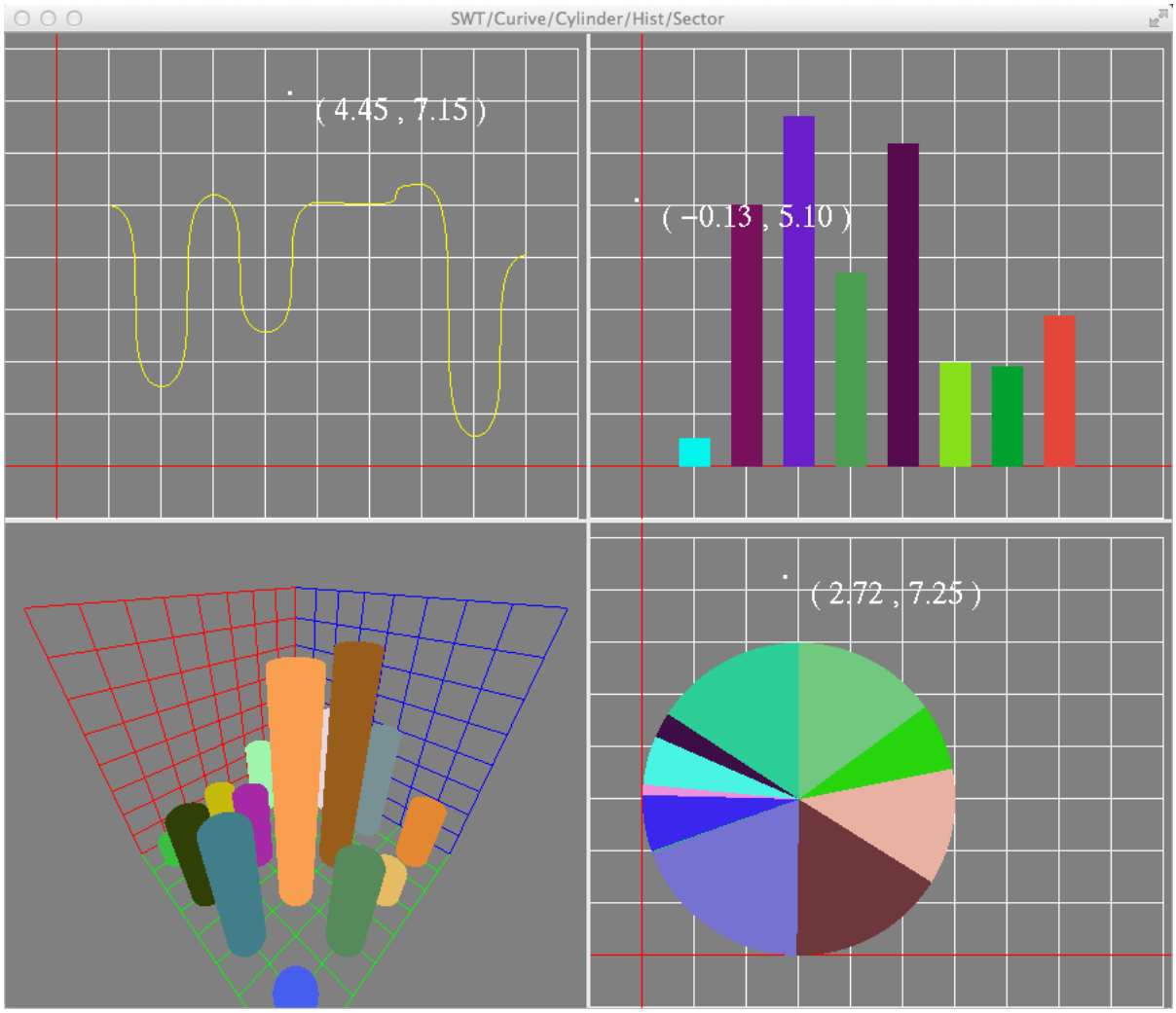
主程序大概流程如下：

step1: 用 SashForm 将 Shell 分成 4 块；
step2: 新建 4 个 Paint；
step3: 添加相关的图表；
step4: 用 Display 异步显示。

主要代码如下：

```
Shell shell = new Shell();
SashForm sashForm = new SashForm(shell, SWT.BORDER);
SashForm leftSashForm = new SashForm(sashForm, SWT.VERTICAL);
SashForm rightSashForm = new SashForm(sashForm, SWT.VERTICAL);

final Paint leftUp = new Paint2D(leftSashForm, SWT.NONE);
final Paint leftDown = new Paint3D(leftSashForm, SWT.NONE);
final Paint rightUp = new Paint2D(rightSashForm, SWT.NONE);
final Paint rightDown = new Paint2D(rightSashForm, SWT.NONE);

Curve curive = new Curve();
Cylinder cylinder = new Cylinder();
Hist hist = new Hist();
Sector sector = new Sector();

leftUp.addDrawable(curive);
leftDown.addDrawable(cylinder);
rightUp.addDrawable(hist);
rightDown.addDrawable(sector);

Display.getDefault().asyncExec(new Runnable() {
                @Override
                public void run() {
                        leftUp.draw();
                        leftDown.draw();
                        rightUp.draw();
                        rightDown.draw();
                }
        });
```

运行结果如下：



5. 参考

参考《OpenGL 基础编程》
API：http://download.java.net/media/jogl/jogl-2.x-docs/