Jzy3d Quick Start Guide

This quick start guide helps you immediatly display standard 3d elements in a chart. The complete sample code can be downloaded from www.jzy3d.org.

For further details on 3d chart customisation, please purchase the official Jzy3d documentation. A summary of its content is available on the website.

Summary

Drawing functions: the 3d Surface Demo

Drawing functions allow to programmatically set:

- a z=f(x,y) function:

```
Mapper mapper = new Mapper(){
 public double f(double x, double y) {
  return 10*Math.sin(x/10)*Math.cos(y/20)*x;
 }
};
```

- a pair of range for X and Y values to plot, and a number of steps for both ranges:

```
Range xrange = new Range(-150,150);
Range yrange = new Range(-150,150);
int  nsteps = 50;
```

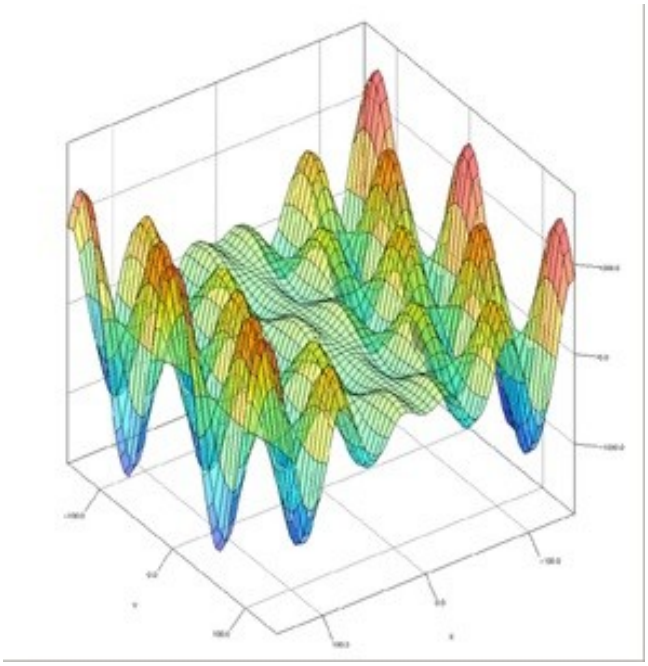
- a surface builder that will apply the z=f(x,y) function to the grid and compute colored 3d polygons:

```
Shape surface = (Shape)Builder.buildOrthonormal(new OrthonormalGrid(range,
steps, range, steps), mapper);
```

- you can add few rendering parameters for your surface object:

```
surface.setColorMapper(new ColorMapper(new ColorMapRainbow(),
surface.getBounds().getZmin(), surface.getBounds().getZmax(), new
Color(1,1,1,.5f)));
surface.setFaceDisplayed(true); // draws surface polygons content
surface.setWireframeDisplayed(true); // draw surface polygons border
surface.setWireframeColor(Color.BLACK); // set polygon border in black
surface.setFace(new ColorbarFace(surface)); // attach a 2d panel
annotating the surface: a colorbar
surface.setFace2dDisplayed(true); // opens a colorbar on the right part of
the display
```

which will produce:



Here is the complete listing of the org.jzy3d.demos.charts.surface.SurfaceDemo:

```
public static Chart getChart(){
 // Define a function to plot
 Mapper mapper = new Mapper(){
  public double f(double x, double y) {
   return 10*Math.sin(x/10)*Math.cos(y/20)*x;
  }
 };

 // Define range and precision for the function to plot
 Range range = new Range(-150,150);
 int steps  = 50;

 // Create the object to represent the function over the given range.
 surface = (Shape)Builder.buildOrthonormal(
            new OrthonormalGrid(range, steps, range, steps), mapper);

 surface.setColorMapper(new ColorMapper(
                           new ColorMapRainbow(),
                              surface.getBounds().getZmin(),
                              surface.getBounds().getZmax(),
                              new Color(1,1,1,.5f)));
 surface.setWireframeDisplayed(true);
 surface.setWireframeColor(Color.BLACK);
 surface.setFace(new ColorbarFace(surface));
 surface.setFaceDisplayed(true);
 surface.setFace2dDisplayed(true); // opens a colorbar

 // Create a chart
 Chart chart = new Chart();
 chart.getScene().getGraph().add(surface);
 return chart;
}
```

Drawing parametrized functions: the Animated Surface Demo

The animation package introduce a sample ParametrizedMapperthat let you change the behaviour of your z=f(x,y) function according to a p parameter. A basic thread change the mapper parameter, and remap the whole shape. The idea behind remapping, is that it's not necessary to compute the tesselation again since it won't change. Instead, we can simply update z values according to the ParametrizedMapper' state:

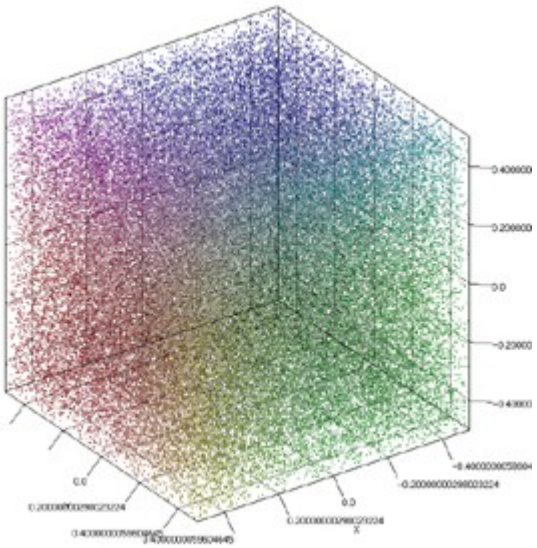
```
protected static void remap(Shape shape, Mapper mapper){
 List<Drawable> polygons = shape.getDrawables();
 for(Drawable d: polygons){
  if(d instanceof Polygon){
   Polygon p = (Polygon) d;
   for(int i=0; i<p.size(); i++){
    Point pt = p.get(i);
    Coord3d c = pt.xyz;
    c.z = (float) mapper.f(c.x, c.y);
   }
  }
 }
}
```

Drawing 3D scatter plots

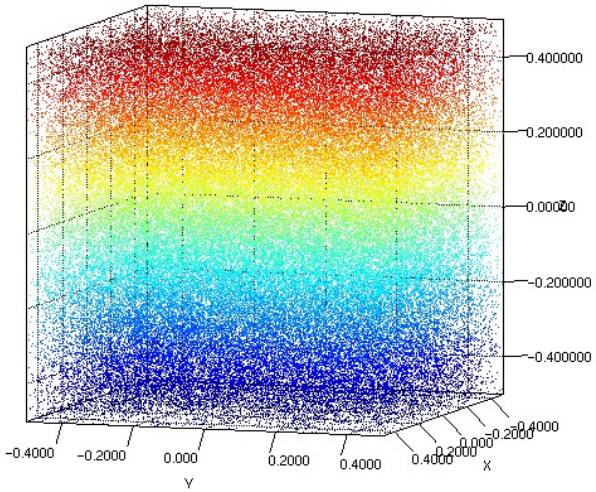
When you wish to draw a 3d scatter plot, you may fall into three cases concerning the scatter coloring:

- The color is known for each point.
- The color can be given by a function, and you would like to see the colormap of this function.

The first case is illutrated by the ScatterDemo, which produce:



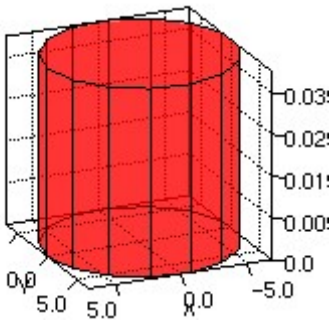
The second case is illustrated by the MultiColorScatterDemo, which produce points colored according to their Z value. Please note that using a Colormap for coloring the scatter is the only way to have a colorbar in the chart: indeed the colorbar requires a function to draw the color gradient.



Drawing custom composite objects: the 3d Histogram Demo

The Histogram Demo show you how to create a custom 3d object to be repeated. The Cylinder class is a Composite object made of a top and bottom polygon, and a serie of quads that will produce the tube. It just requires a setData(...) method so that you can set it up in one call:

```
public class Cylinder extends Composite{
    public void setData(Coord3d position, float height, float radius, int
slices, int rings, Color color){
        // Create sides
        top = new Polygon();
        low = new Polygon();

        for(int i=0; i<slices; i++){
            float angleBorder1 = (float)i*2*(float)Math.PI/(float)slices;
            float angleBorder2 = (float)(i+1)*2*(float)Math.PI/(float)slices;

            Coord2d border1 = new Coord2d(angleBorder1, radius).cartesian();
            Coord2d border2 = new Coord2d(angleBorder2, radius).cartesian();

            Quad face = new Quad();
            face.add(new Point(
new Coord3d(position.x+border1.x, position.y+border1.y, position.z)));
            face.add(new Point(
new Coord3d(position.x+border1.x, position.y+border1.y, position.z+height)));
            face.add(new Point(
new Coord3d(position.x+border2.x, position.y+border2.y, position.z+height)));
            face.add(new Point(
new Coord3d(position.x+border2.x, position.y+border2.y, position.z)));
            face.setColor(color);
            face.setWireframeDisplayed(false);

            // add the polygon to the cylinder
            add(face);

            // compute top and low faces
            low.add(new Point(new Coord3d(position.x+border1.x,
position.y+border1.y, position.z)));
            top.add(new Point(new Coord3d(position.x+border1.x,
position.y+border1.y, position.z+height)));
        }
        low.setColor(color);
        top.setColor(color);
        add(top);
        add(low);
    }

    private Polygon top;
    private Polygon low;
}
```



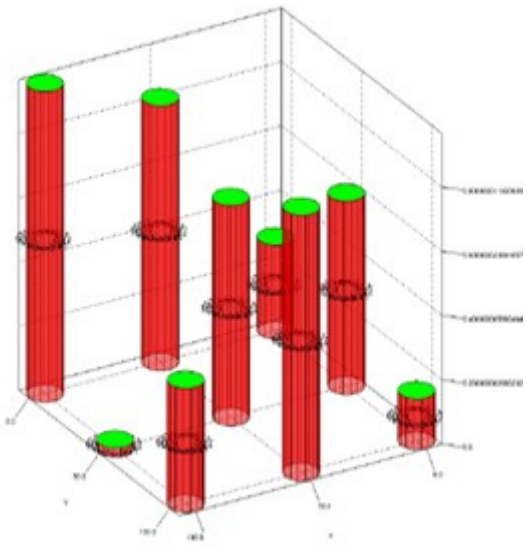
Note: OpenGL has some dedicated primitives for rendering cylinders, and this Cylinder implementation mainly shows how to create a Composite.

You will then wish to setup a scene object that will build, and register cylinder objects:

```
public class HistogramScene extends ChartScene {
     public HistogramScene(){
          ColorMap map = new ColorMapWhiteRed();
          map.setDirection(false);
          mapper = new ColorMapper(map, 0, 1);
     }

     public void addCylinder(double x, double y, double height){
          if(height>1 || height<0)
               throw new IllegalArgumentException("height is supposed to be a ratio");
          Color color = Color.RED; //mapper.getColor(new Coord3d(0,0,height));
          color.a = 0.55f;

          Cylinder bar = new Cylinder();
          bar.setData(new Coord3d(x, y, 0), (float)height, 7f, 15, 00, color);
          bar.setWireframeDisplayed(true);
          bar.setWireframeColor(Color.BLACK);
          graph.add(bar);
     }
     private ColorMapper mapper;
}
```

You then setup the scene content with data, and attach it to a Chart object, to be rendered in your UI:

```
final HistogramScene histogramScene = new HistogramScene();
for(int i=0; i<150; i+=50)
     for(int j=0; j<150; j+=50)
          histogramScene.addCylinder(i,j,Math.random());
// Create a chart for this scene
Chart chart = new Chart(){
     public HistogramScene initializeScene(){
          return histogramScene;
     }
};
```



Drawing dense 2d graphs

Rendering graphs with OpenGL let Jzy3d display dense graphs efficiently.

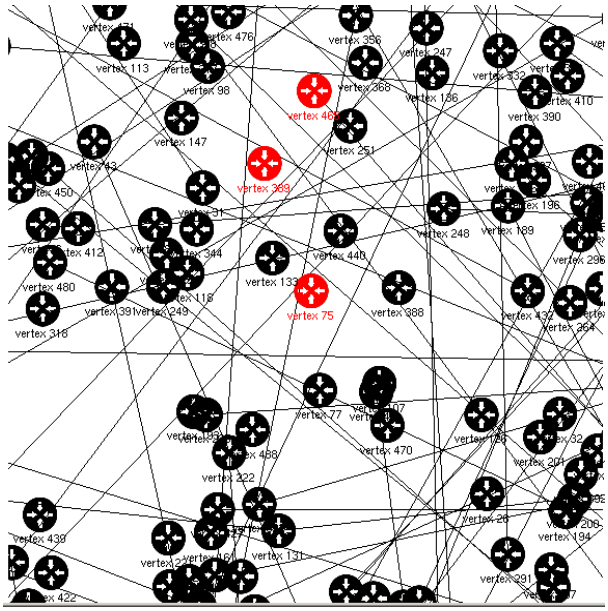
You just have to define an IGraph<V,E>, IGraphLayout2d<V,E>, and IGraphFormatter<V,E>, as follow in the example:

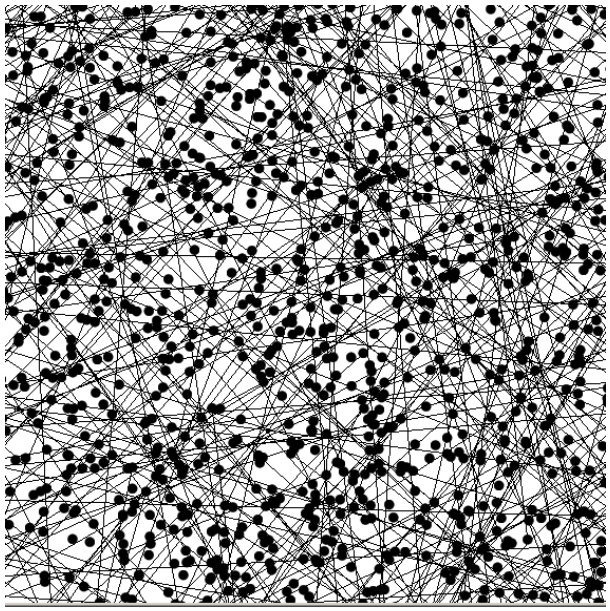
```
IGraph<String, String> graph = StringGraphGenerator.getGraph(NODES, EDGES);
IGraphLayout2d<String> layout = StringGraphGenerator.getRandomLayout(graph,
GL_LAYOUT_SIZE);
IGraphFormatter<String, String> formatter = new DefaultGraphFormatter<String, String>();
```

The formatter will let you setup colors and width of vertices, edges and their labels, in normal or highlighted status. You can define how to place the labels ({left|center|right} and {top|center|bottom}) relative to their anchor point.

You can use any IDrawableGraph2d<V,E>, that is any GL implementation you like for your graph. For the moment, you can use

- PointGraph2d<V,E>
- TextureGraph2d<V,E>





Each IDrawableGraph2d implementation may define a label screen offset, and a label scene offset. The first states how you shift your label from the vertex point based on a screen (2d) distance, while the later consider an offset in 3d coordinates, meaning in the GL space. These settings are GL-implementation dependant and not moved to the IGraphFormatter.

Note that the TextureGraph2dlet you load a PNG mask texture that you can later filter with a color. One may as well define a graph where the textures are based on any java BufferedImage.

To build a IDrawableGraph2d, you have to follow these steps:

```
PointGraph2d<String,String> drawableGraph = new PointGraph2d<String,String>();
drawableGraph.setGraphModel(graph, mouse.getPickingSupport());
drawableGraph.setGraphFormatter(formatter);
drawableGraph.setGraphLayout(layout);
```

As you have seen, you need to provide a picking support that registers a mapping between model data and representation data. Then you have to tell what to do when an object is picked. For example, you can set it as "highlighted" to have its color changed:

```
GraphChartMouseController<String,String> mouse = new
GraphChartMouseController<String,String>(chart, (int)VERTEX_WIDTH);
mouse.getPickingSupport().addVertexPickListener(new IObjectPickedListener<String>() {
 @Override
 public void objectPicked(List<String> vertices) {
  for(String vertex: vertices){
   System.out.println("picked: " + vertex);
   drawableGraph.setVertexHighlighted(vertex, true);
  }
  chart.render();
 }
});
```

See: org.jzy3d.demos.graphs.Demo

Warning: Edge labels are not implemented yet.

Warning: Jzy3d is slow at startup when displaying lot of labels, which is due to a known bug in JOGL-1.1.1-rc3. Rendering labels will go much faster when ported to JOGL2