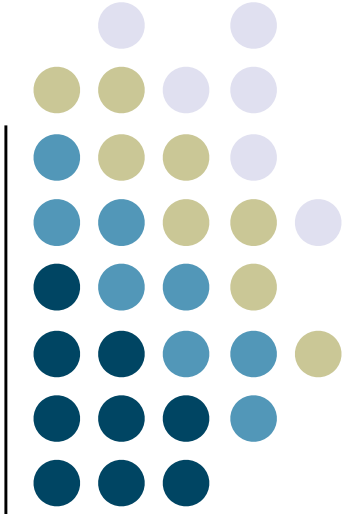
Some Advanced Topics in Processing

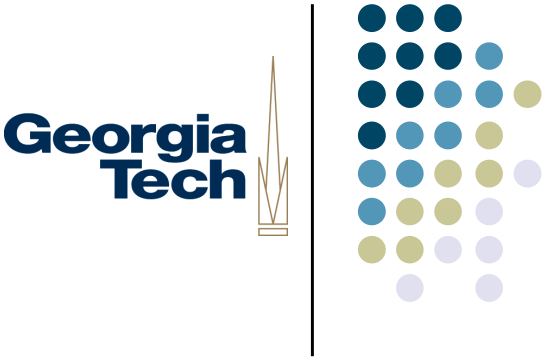


Georgia Tech

Getting Stuff In and Out of Processing

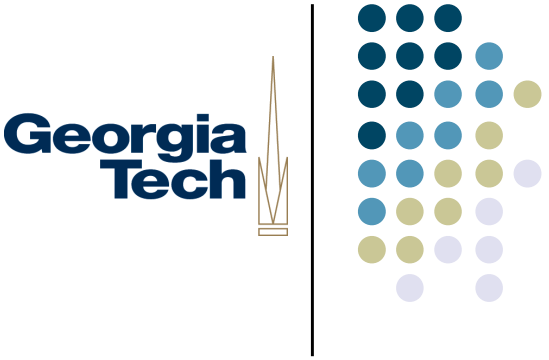
- Exporting your sketch as a stand-alone application
  - Select File -> Export Application
  - Create an app for your platform
  - Option to bundle Java Runtime Environment
    - More likely your code will run the same as on your computer
    - Resulting app is ~100MB larger!
    - Caution: don't have a method named main() in your sketch

Getting Stuff In and Out of Processing



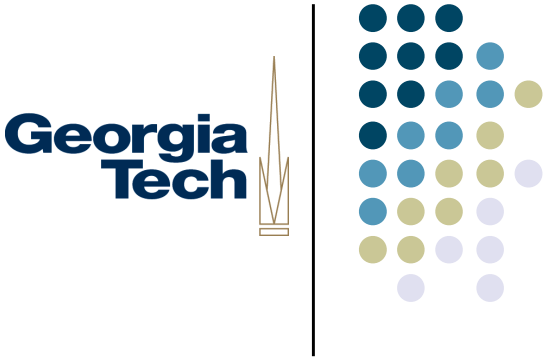
- Importing libraries
  - Sketch -> Import Library… -> Add Library…
  - Lets you select from libraries that are known to the Processing developers
  - Can add libraries "manually" also
    - Download, put them under the "libraries" folder in your sketch path

Getting Stuff In and Out of Processing



- The data folder
  - Functions that deal with images and files (e.g., loadImage() and loadString() look for files inside a folder named **data,** inside your sketch's folder
- E.g., String[] lines = loadStrings("foo.txt");
  - Expects data/foo.txt

Getting set up for HW4

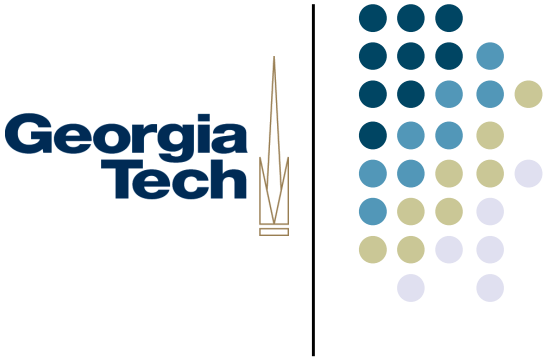


- Download Unfolding Maps
  - **IMPORTANT:** Use the Beta version on t-square, not the one from the website!
  - Un-zip and install under the "libraries" folder
  - Restart processing
- Grab the tweets.json file
  - Create, name, and save your HW4 sketch
  - put tweets.json under data under your sketch's folder

Using Custom Renderers

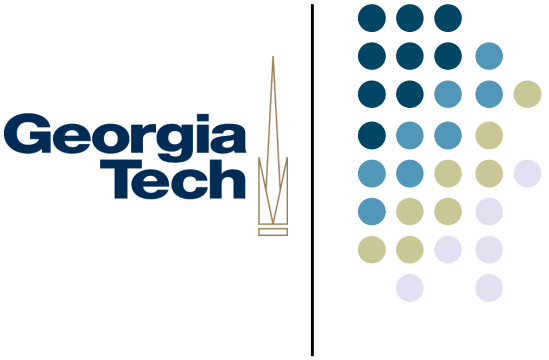
- Most programs never need to worry about this, but…
- Can set up different **renderers** that tell Processing how to optimize graphics.
- Unfolding Maps works best with the 2D renderer
  - In your program: size(400, 400, P2D);
  - Accelerated through OpenGL
  - Faster, but less accurate (we want fast for HW4)
- (There's also a P3D renderer that makes 3D graphics run better.)

Using JSON from Processing



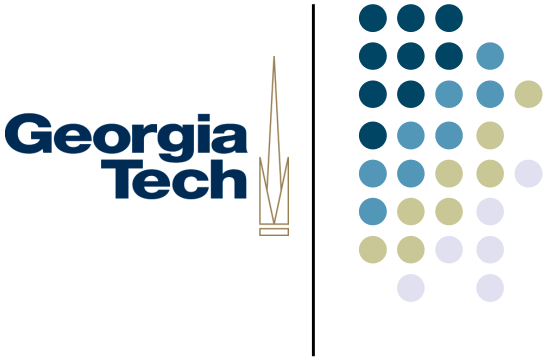
- Processing has JSON support similar to Python's
  - `JSONObject obj = loadJSONObject("filename.json")`
- Once you have a JSON object, you can pull out the data associated with specific attributes
  - One difference from the Python libraries is that you have to know the type of the data you're accessing
  - `String s = obj.getString("text");  // type of data is String`
  - `int i = obj.getString("count");  // type of data is int`
- For data that is itself a JSON object (like a Python dictionary), use getJSONObject()
  - `JSONObject userObj = obj.getJSONObject("user");`
- For list data, use getJSONArray() then access via index
  - `JSONArray statuses = obj.getJSONArray("statuses");`
  - `JSONObject s = statuses.getJSONObject(0);`

Getting Started with Unfolding



- Sketch -> Import Library
  - (If Unfolding is not there, you probably made a mistake installing it in the libraries folder)

Simple Maps



- Define a new map variable:
  - `UnfoldingMap map;`
- In your setup() function, create a new map and add the default event handler to the map

```
void setup() {
   size(800, 600, P2D);
   map = new UnfoldingMap(this);
   MapUtils.createDefaultEventDispatcher(this, map);
}
```

Simple Maps

- In your draw() function, tell the map to draw itself (O-O!)
  - UnfoldingMap map;

```
void draw() {
   map.draw();
}
```

- Try it!

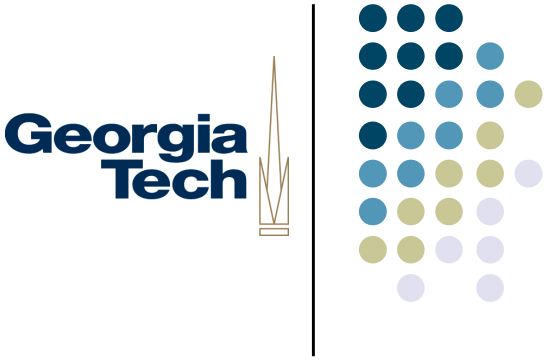
A Tip…

- Map data can be big.
- Preferences -> Increase maximum available memory
  - 1024MB may be a good figure, just in case

Screen Coordinates and Mouse Coordinates

- Maps have their own built-in coordinate systems
  - Longitude and Latitude
- So does the screen
  - X & Y pixel coordinates
- How do we convert between them?
  - Screen to map:
    - `Location loc = map.getLocation(mouseX, mouseY);`
    - `println("Lat/Lon of mouse is" + loc.getLon() + "," + loc.getLat());`
  - Map to screen:
    - `ScreenPosition pos = map.getScreenPosition(loc);`
    - `println("X,Y coords of location " + loc + " are " + pos.x + "," + pos.y);`

Using Markers

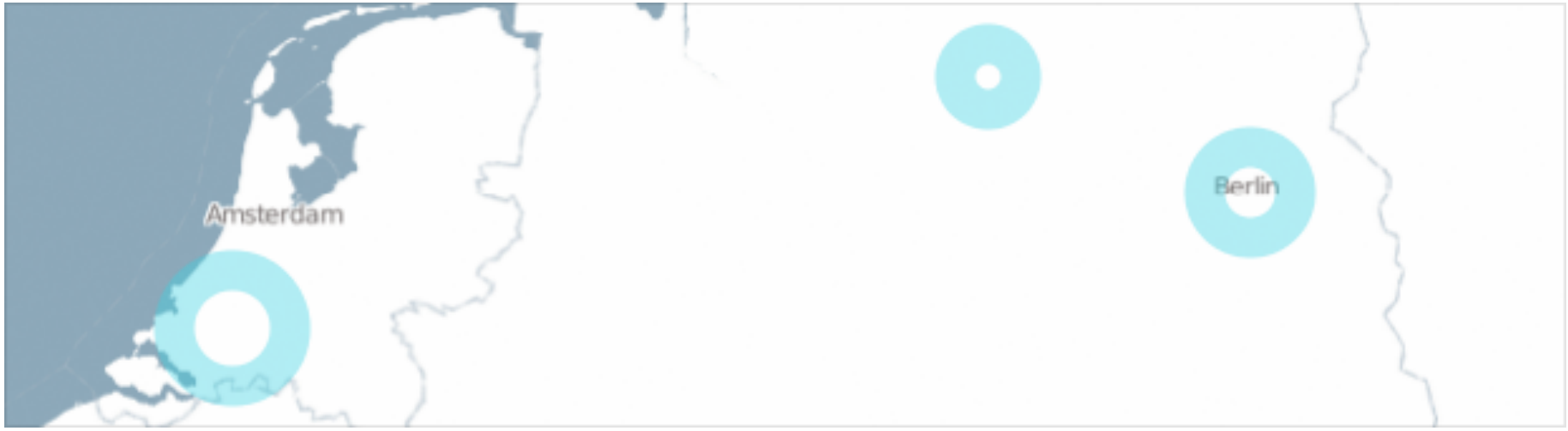


- Lots of different ways to use markers. Here's one simple way:
  - `Location berlinLoc = new Location(52.5, 13.4);`
  - `SimplePointMarker berlinMarker = new SimplePointMarker(berlinLoc);`
  - `map.addMarkers(berlinMarker);`
- Uses the default marker style.
- The map automatically manages drawing markers that have been added to it.

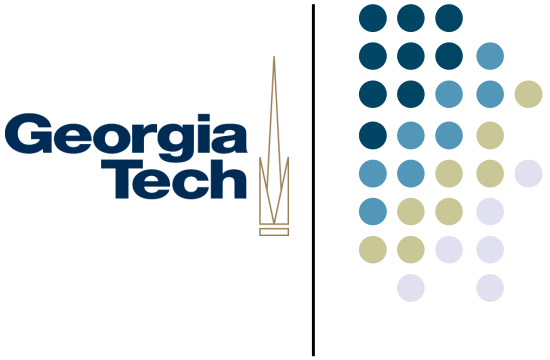
Styling Markers

- Some basic properties can be set directly on the markers themselves:

```
berlinMarker.setColor(color(255, 0, 0, 100));
berlinMarker.setStrokeColor(color(255, 0, 0));
berlinMarker.setStrokeWeight(4);
```



Styling Markers - Advanced Method



- Rather than have the marker automatically drawn, you can draw it yourself
  - Don't add it to the map…
  - … you're just using it as a way to keep track of the place.

```
void draw() {
  map.draw();

  ScreenPosition berlinPos = berlinMarker.getScreenPosition(map);
  strokeWeight(16);
  stroke(67, 211, 227, 100);
  noFill();
  ellipse(berlinPos.x, berlinPos.y, 36, 36);
}
```