EECS 1710
Programming for Digital Media

Lecture 22 :: PShape & PGraphics

YORK UNIVERSITÉ UNIVERSITY

- PShape
  - Data type for storing shapes (especially non-primitive shapes)
  - Can specify vertexes for complex polygons
  - createShape(), beginShape(), endShape(), vertex()
  - loadShape(), hold geometry loaded from a file (e.g. SVG)

- PGraphics
  - Can be used as an off screen graphics buffer
    - i.e. like an additional app window (not visible)
  - Can be used to render to different outputs (e.g. pdf files or svg files, alongside the main app)

- createShape() used to define a new shape
  - can create primitive shapes (similar to draw methods) but this method creates an object that can be stored

Syntax

```
createShape()
createShape(type)
createShape(kind, p)
```

Parameters

kind	(int)	either POINT, LINE, TRIANGLE, QUAD, RECT, ELLIPSE, ARC, BOX, SPHERE
p	(float[])	parameters that match the kind of shape

Return

PShape

```
ArrayList<PShape> shapes = new ArrayList<PShape>();

void setup() {
  size(400,400);

  square = createShape(RECT,10,300,50,50);
  square.setFill(color(0, 0, 255));
  square.setStroke(false);
  shapes.add(square);

  oval = createShape(ELLIPSE,200,100,80,40);
  oval.stroke(color(0,255,0));
  shapes.add(oval);

  arc = createShape(ARC,50, 55, 50, 50, 0, HALF_PI);
  arc.setFill(color(255,0,0));
  arc.stroke(color(0,0,0));
  arc.setStroke(true);
  shapes.add(arc);

  triangle = createShape(TRIANGLE,300, 300, 332, 80, 344, 300);
  triangle.setFill(color(128,128,128));
  triangle.setStroke(false);
  shapes.add(triangle);

}


void draw() {
  background(255,255,255);
  for (int i=0; i<shapes.size(); i++) {    // special version of loop on a collection
    shape(shapes.get(i));
  }
}
```

Interpreting vertices as “kinds” of shapes

- createShape() used to define a unique shape
  - can create sets of vertices to be interpreted a certain way
  - uses beginShape()

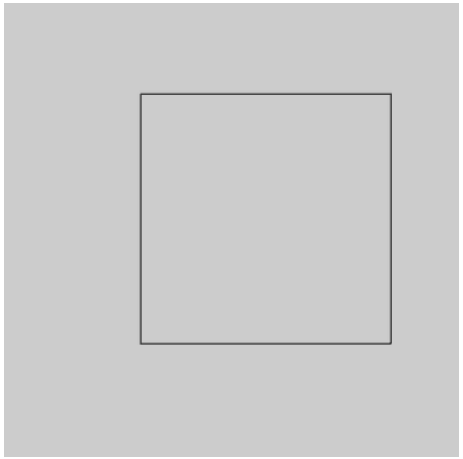
Syntax	`beginShape()`	
	`beginShape(kind)`	
Parameters	**`kind`** `(int)`	Either POINTS, LINES, TRIANGLES, TRIANGLE_FAN, TRIANGLE_STRIP, QUADS, or QUAD_STRIP
Return	`void`	

  - uses endShape()

PShape

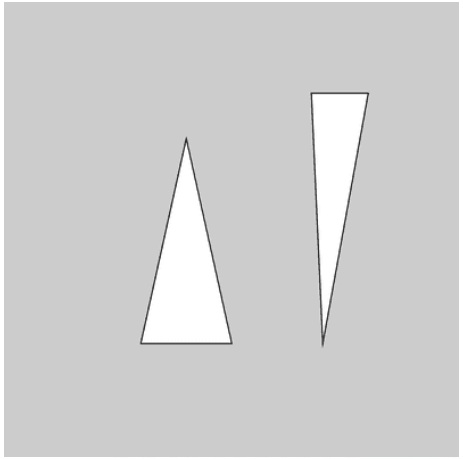
```
noFill();
beginShape();

vertex(120, 80);
vertex(340, 80);
vertex(340, 300);
vertex(120, 300);

endShape(CLOSE);
```

Connects with lines (CLOSE draws a line back to first vertex)



```
beginShape(TRIANGLES);

vertex(120, 300);
vertex(160, 120);
vertex(200, 300);

vertex(270, 80);
vertex(280, 300);
vertex(320, 80);

endShape();
```

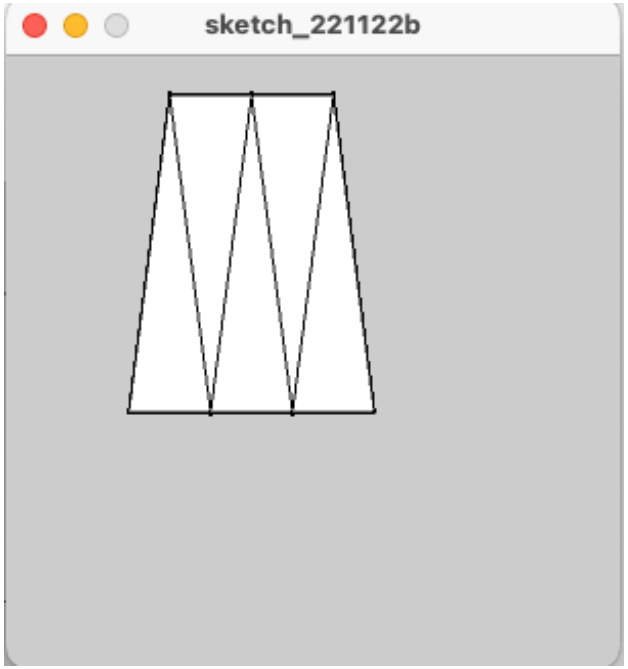
Interprets every 3 vertices as points of a triangle, next 3 as next triangle, etc..



https://processing.org/reference/beginShape_.html

PShape

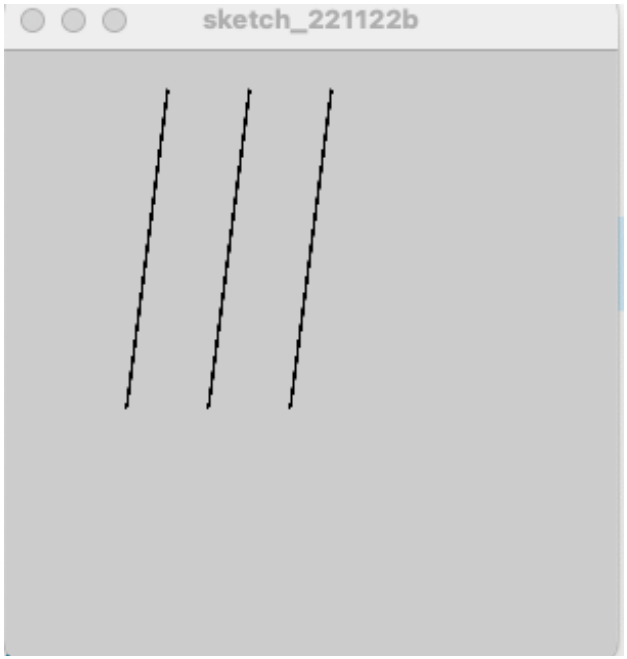
```
PShape s;

void setup() {

  size(100, 100, P2D);
  s = createShape();
  s.beginShape(TRIANGLE_STRIP);
  s.vertex(30, 75);
  s.vertex(40, 20);
  s.vertex(50, 75);
  s.vertex(60, 20);
  s.vertex(70, 75);
  s.vertex(80, 20);
  s.vertex(90, 75);
  s.endShape();
}

void draw() {
  shape(s, 0, 0);
}
```



https://processing.org/reference/beginShape_.html

PShape

```
PShape s;

void setup() {

  size(100, 100, P2D);
  s = createShape();
  s.beginShape(LINES);
  s.vertex(30, 75);
  s.vertex(40, 20);
  s.vertex(50, 75);
  s.vertex(60, 20);
  s.vertex(70, 75);
  s.vertex(80, 20);
  s.vertex(90, 75);
  s.endShape();
}

void draw() {
  shape(s, 0, 0);
}
```



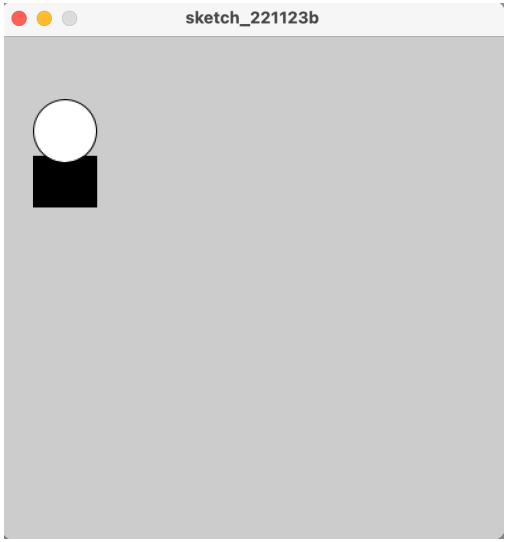
https://processing.org/reference/beginShape_.html

Custom shapes: grouping vertices

https://processing.org/tutorials/pshape/#custom-pshapes

```
PShape star;

void setup() {
  size(640,360,P2D);

  star = createShape();
  star.beginShape();
  star.fill(102);
  star.stroke(255);
  star.strokeWeight(2);

  // series of vertices
  star.vertex(0, -50);
  star.vertex(14, -20);
  star.vertex(47, -15);
  star.vertex(23, 7);
  star.vertex(29, 40);
  star.vertex(0, 25);
  star.vertex(-29, 40);
  star.vertex(-23, 7);
  star.vertex(-47, -15);
  star.vertex(-14, -20);

  star.endShape(CLOSE);
}

void draw() {
  background(51);
  translate(mouseX, mouseY);
  shape(star);
}
```

Custom shapes: grouping shapes



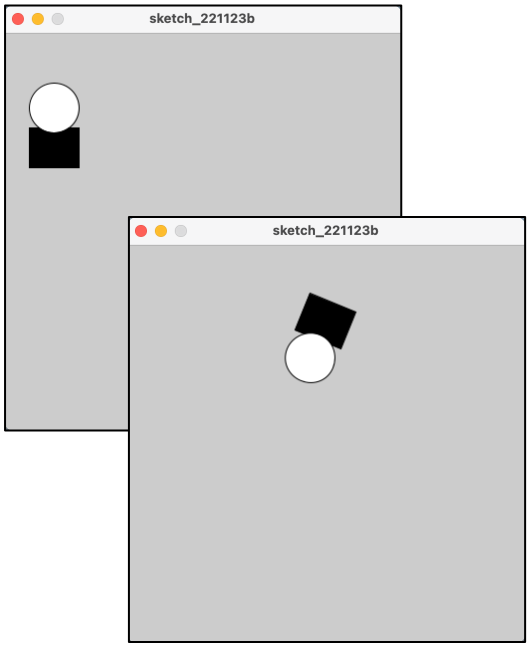

```
PShape alien, head, body;
PVector pos;
float angle;

void setup() {
  size(400, 400);

  // Create the shape group
  alien = createShape(GROUP);

  // Make two shapes
  ellipseMode(CORNER);
  head = createShape(ELLIPSE, -25, 0, 50, 50);
  head.setFill(color(255));
  body = createShape(RECT, -25, 45, 50, 40);
  body.setFill(color(0));

  // Add the two "child" shapes to the parent group
  alien.addChild(body);
  alien.addChild(head);

  shape(alien);     // draw group shape

}
```

Scaling shapes

```
void mousePressed() {

  println(mouseButton);

  if (mouseButton==37)
    star.scale(1.1);

  if (mouseButton==39)
    star.scale(0.9);

}
```

```
PShape star;

void setup() {
  size(640,360,P2D);

  // First create the shape
  star = createShape();
  star.beginShape();
  // You can set fill and stroke
  star.fill(102);
  star.stroke(255);
  star.strokeWeight(2);
  // Here, we are hardcoding a series of vertices
  star.vertex(0, -50);
  star.vertex(14, -20);
  star.vertex(47, -15);
  star.vertex(23, 7);
  star.vertex(29, 40);
  star.vertex(0, 25);
  star.vertex(-29, 40);
  star.vertex(-23, 7);
  star.vertex(-47, -15);
  star.vertex(-14, -20);
  star.endShape(CLOSE);
}

void draw() {
  background(51);
  translate(mouseX, mouseY);
  shape(star);
}
```

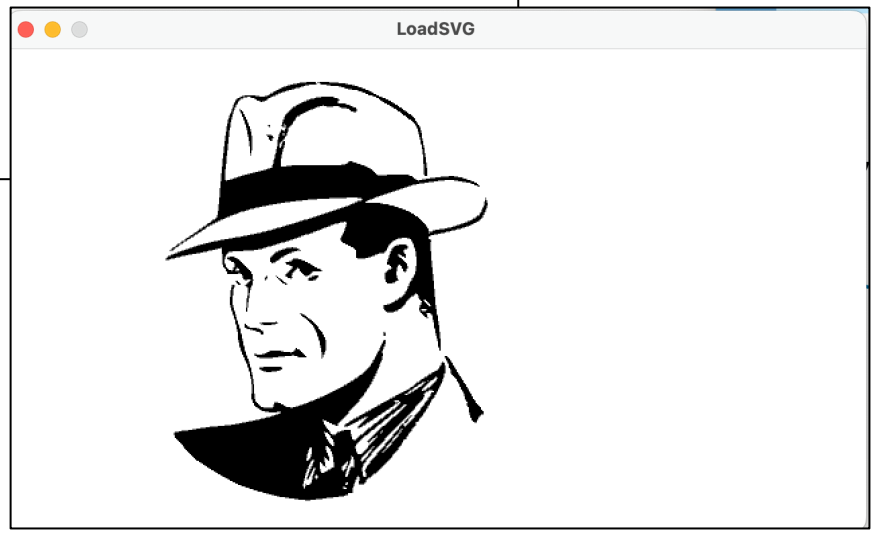
moving shapes




```
PShape alien, head, body;
PVector pos;
float angle;

void setup() {
  size(400, 400);

  // Create the shape group
  alien = createShape(GROUP);

  // Make two shapes
  ellipseMode(CORNER);
  head = createShape(ELLIPSE, -25, 0, 50, 50);
  head.setFill(color(255));
  body = createShape(RECT, -25, 45, 50, 40);
  body.setFill(color(0));

  // Add the two "child" shapes to the parent group
  alien.addChild(body);
  alien.addChild(head);

  // setup for animated movement of alien
  pos = new PVector(0,0);
  angle = 0.0;
}

void draw() {
  background(204);
  pos.add(1,1);
  angle++;

  translate(pos.x,pos.y);
  rotate(radians(angle));

  shape(alien); // Draw the group
}
```

Loading external shapes

```
PShape svg;

void setup() {
 size(640, 360, P2D);
 svg = loadShape("retro-detective.svg");
 println("w,h = " + svg.width +", " + svg.height);
}

void draw() {
 background(255);

 translate(mouseX, mouseY);
 shape(svg);
}
```



https://freesvg.org/detective-from-the-comic-book

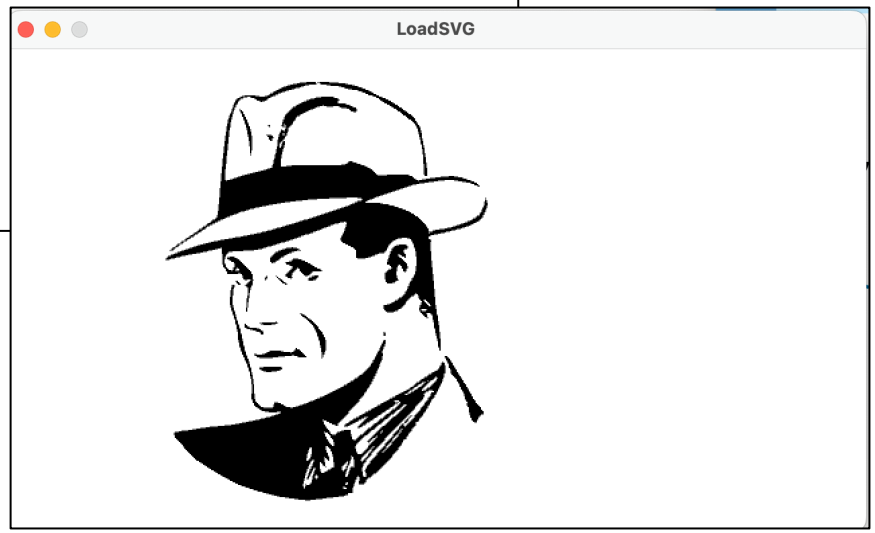
Accessing/Modifying vertices of a PShape

```
// assume s = star (the PShape object from slide 8)


for (int i = 0; i < s.getVertexCount(); i++) {

      PVector v = s.getVertex(i);
      v.x += random(-1, 1);
      v.y += random(-1, 1);

      s.setVertex(i, v.x, v.y);

}
```

Note, not all PShapes have vertices

```
PShape svg;

void setup() {
 size(640, 360, P2D);
 svg = loadShape("retro-detective.svg");
 println("w,h = " + svg.width +", " + svg.height);
 println("svg -> vertices: " + svg.getVertexCount());
}

void draw() {
 background(255);

 translate(mouseX, mouseY);
 shape(svg);
}
```

Outputs zero



https://freesvg.org/detective-from-the-comic-book

PGraphics

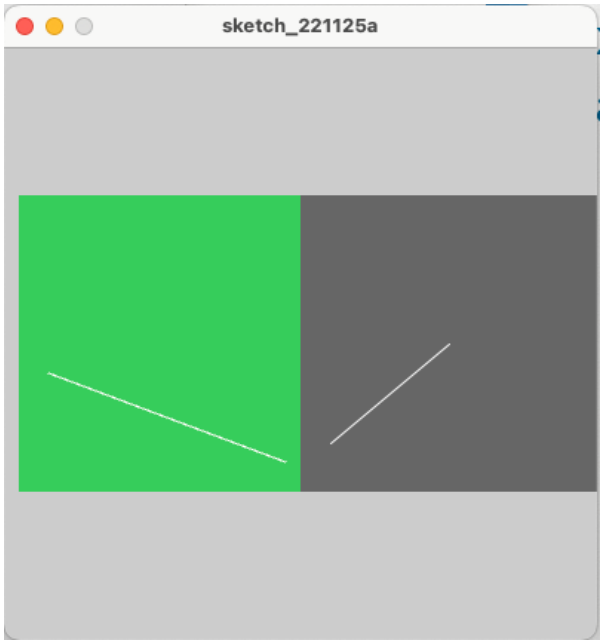
- createGraphics(x,y);
  - like createImage() but creates a new PGraphics object with a particular size (keep in mind this can be diff to app size)
  - Can have different “renderers” like size()
    - P2D – 2D graphics
    - P3D – 3D graphics
    - PDF – write graphics to a *.pdf file
    - SVG – write graphics to a *.svg file (scalable vector graphics)
- beginDraw() , endDraw()
  - All commands can be directed towards PGraphics object
  - beginDraw() – all commands after this are essentially

Keep draw properties separate ..
i.e. maintain different graphics “contexts”

```
PGraphics pg;

void setup() {
  size(100, 100, P2D);

  pg = createGraphics(80, 80, P2D);

  pg.beginDraw();

  pg.background(102);
  pg.stroke(255);
  pg.line(20, 20, 80, 80);

  pg.endDraw();
  noLoop();
}

void draw() {
  image(pg, 10, 10);
}
```

Two PGraphics objects?

```
PGraphics pg, pg2;

void setup() {
  size(400, 400, P2D);


  pg = createGraphics(200, 200, P2D);
  pg.beginDraw();
  pg.background(102);
  pg.stroke(255);
  pg.line(20, 120, 180, 180);
  pg.endDraw();

  pg2 = createGraphics(200, 200);


}

void draw() {
  pg2.beginDraw();
  pg2.background(102);
  pg2.stroke(255);
  pg2.line(pg.width*0.5, pg.height*0.5, mouseX, mouseY);
  pg2.endDraw();

  image(pg, 10, 100);
  image(pg2, 200, 100);
}
```



PGraphics

- Other uses
  - Maintain multiple ”screen layouts” and switch between them
  - Keep different areas of same layout separate (code responses to each separately)
  - Keep a context for rendering to PDF / SVG files (separate from what is shown on app window)
  - etc

Developers reference (full api for processing)

- http://processing.github.io/processing-javadocs/core/