Open tickets for others
Tickets opened
High priority
Medium priority
Lower priority

(currently categorized, but not prioritized)

2D Mode Feature Parity

To help users make the leap to using WEBGL, most sketches should work when `createCanvas(400, 400);` becomes `createCanvas(400, 400, WEBGL);` (That is, changing the renderer from 2D to 3D shouldn't break the sketch or stop it from running)

Right now there are two reasons this might happen
1) some features are not implemented for the 3D renderer and
2) some features are implemented for 3D mode but do not function with 2D arguments, as Processing does (Processing fills in missing z-arguments as 0s, for example)

Missing API functionality:

- Arc
- Image
- Text
- Point
- begin/endContour
- bezierVertex, curveVertex, quadraticVertex

Features with 3D implementation but no 2D fallback in the API (or with 2D but no 3D equivalent):

- Line
- Bezier
- curve
- Triangle?
- Quad?

Stroke and Fill

Expand 3D mode to use the stroke and fill color for a shape, and to allow both to be used simultaneously on the same shape in both retain and immediate mode. Requires building out new shaders that can render both the fill and the line efficiently.

- strokeWeight
- stroke + fill shader

- resolve camera/transform needs for shader

Related issues on github:

- #1703 beginShape() not taking stroke() values in WebGL
- #1680 strokeWeight() not working in WebGL mode
- #1678 beginShape should not require call to fill() in order to render

Performance

- Shader efficiency and API
    - Abstract cache/hash lookup to API; don't access data structure directly
    - Organize preparation of geometry for shaders to use same approach in immediate and retain mode (similar API, at least); cut some repeated code.
    - See if any gains can be made by only changing shader when necessary
    - See also: proposed API for shader uniforms
- #1716 Firefox - slow handling of endShape in createGraphics

Control of Geometry Cache

In addition to drawing 3D primitives, allow the developer to create and cache their own custom shapes using something akin to Processing's PShape. PShape provides a variety of default geometries that mirror the 3D geometry primitives, as well as letting the user define their own custom shapes with begin/endShape and vertex methods.

Users could then cache their geometry in setup, and simply render in draw. This would prepare them to move on to a more complex/sophisticated toolkit for 3D graphics like three.js, which uses a scene graph model. Even if custom shapes aren't possible, having a handle for primitives would let people start modeling in the fashion of three.js

Porting Processing 3D Features

Processing users may expect some 3D-only features that do not yet have an implementation in p5.js.

- u,v texturing with begin/endShape and additional texturing features
    - textureMode (0-1 range uvs vs. pixel coordinates)
    - textureWrap
    - Related issues:
        - #1424 texture uv coordinates
        - #1561 webgl: texture not being set when using shapes
- Camera features
    - Set camera eye point, rotate camera
    - begin/endCamera

    - printCamera, printMatrix, printProjection, and other helpful debugging tools
    - Related issue: #1194 improve camera api
- Matrix implementation
    - Implement TODO items in p5.Matrix
- lights()
    - Easy default lighting
- Custom shaders
- Resolve coordinate system issues pertaining to y-axis and rotation direction
    - Related? #1696 Rotation in webgl mode is not clockwise, but counterclockwise

Documentation for Contributors

Extensive documentation of how each part of the code base works, specifically geared at encouraging future contributors to become oriented in the code more quickly.

General Bug Fixing or Feature Requests

- #1721 loadPixels / pixels[] array issues
- #1897 no texture bound to unit 0
- #1843 No shapes with WebGL and createGraphics
- #1553 add screenX, screenY, and screenZ
- #1622 Issues with positioning point lights