PrimitiveUI

# User Manual

## 1. Introduction and Getting Started

Primitive UI offers you the ability to draw shapes and lines directly onto a new component called a Primitive Canvas. The `PrimitiveCanvas` class is subclassed from the `UI.MaskableGraphic` class and is thus fully compatible with the new Unity UI system.

As opposed to the built-in Unity UI objects you need to first make sure to have at least one Canvas object in your scene. After that, to add a Primitive Canvas to your scene you can either right-click in your Hierarchy view and select UI > Primitive Canvas, or navigate to GameObject > UI > Primitive Canvas using the top menu bar.

## 2. Coordinate System and Measurements

Primitive UI follows the same coordinate system as Unity UI, with (0,0) being the bottom-left and (1,1) being the top-right, regardless of actual dimensions. However, this means that a rectangle drawn with dimensions 0.5 x 0.5 will not produce a uniform square unless the Primitive Canvas itself is also square.

In some cases this is desirable; drawing a center line through any axis will always be located at position 0.5 on the orthogonal axis. In other cases, however, this might seem counter-intuitive. There are two rules that are always followed internally:

- When passing parameters pertaining to (a) specific axis/ axes, those axes' local coordinates are used.
- When passing a parameter pertaining to either two axes at once (like a circle's radius) or an unspecified axis (like a stroke's width), **the X-axis' local coordinates are used**.

For example, let's say we have a Primitive Canvas that has a 2:1 (width:height) aspect ratio. Drawing a circle with any radius would internally treat all coordinates to effectively multiply them by 2 on the Y-axis. Thus, drawing a circle with radius 1 would draw a circle exceeding the Primitive Canvas' vertical bounds, but drawing one with radius 0.5 would make it fit snugly against the edges.

The circle's position, however, is given by a `Vector2`, making it applicable to rule number one, and thus using each individual axis' local coordinates.

Giving a line a relative thickness of 0.05 would give the entire stroke a thickness of 1/20th the width of the Primitive Canvas' width.

## 3. Solid Shapes

Primitive UI allows you to draw some incredibly complex shapes, but also comes with a pre-defined set of methods that makes getting shapes on the screen extremely fast and easy.

All of these convenience methods feature optional parameters and are also overloaded a number of times, so you only have to pass in what you care about and can expect sensible defaults on everything else.

All shape methods contain overloads accepting either a fill color, a stroke style, or both. Passing in a fill color will cause that shape to be filled in, while passing in a stroke style will cause an outline of the shape to be stroked. You can find more information on stroke styles in the next chapter *(4. Lines and Paths).*

The available shape methods are:

**DrawSquare** `(Vector2 center, float size[, float rotation, Color? fillColor, StrokeStyle strokeStyle])`
*Draws a square using a uniform size, specified as relative to the Primitive Canvas' width.*

**DrawRectangle** `(Rect rect[, float rotation, Color fillColor, StrokeStyle strokeStyle])`

*Draws a rectangle in dimensions specified per axis.*

**DrawCircle** `(Vector2 center, float radius[, float stepSize, float startAngle, float endAngle, Color? fillColor, StrokeStyle strokeStyle])`

*Draws a circle using a uniform radius, specified as relative to the Primitive Canvas' width.*

**DrawEllipse** `(Vector2 center, Vector2 radii[, float stepSize, float rotation, float startAngle, float endAngle, Color? fillColor, StrokeStyle strokeStyle])`

*Draws an ellipse using radii specified per axis.*

**DrawRegularSolid** `(Vector2 center, float radius, int sides[, float rotation, Color? fillColor, StrokeStyle strokeStyle])`

*Draws a regular solid using a uniform radius, specified as relative to the Primitive Canvas' width. A regular solid is a shape of which all sides are of equal length, and angles are of equal sides. Examples are triangles, squares and pentagons. This is identical to calling `DrawCircle()` with a `stepSize` of `360 / number of sides`.*

**DrawIrregularSolid** `(Vector2 center, float[] radii[, float rotation, Color? fillColor, StrokeStyle strokeStyle])`

*Draws an irregular solid using uniform radii, specified as relative to the Primitive Canvas' width. Produces results similar to `DrawRegularSolid`, with the exception that each point of the polygon can be at a different distance from the specified center.*

**DrawPolygon** `(Vector2[] points[, Color fillColor, StrokeStyle strokeStyle])`

*Draws an arbitrary closed polygon in dimensions specified per axis. After passing in an array of `Vector2` points the Primitive Canvas will tesselate a surface to fill the shape and/or stroke surrounding it.*

Unity 4.6 - 5.1 only:
**DrawRawVBObject** `(Vector2[] points, Color fillColor)`

*Draws the passed quadrilateral polygons in dimensions specified per axis. Directly adds an element to the canvas with its vertices set to those passed in. Note that polygons need to be passed in as quads, meaning a list would look like:*
`[quad1p1, quad1p2, quad1p3, quad1p4, quad2p1, quad2p2, quad2p3, quad2p4, ...].`
*Whenever you need a triangle, simply set that polygon's fourth vertex to be at the same position as its first.*

Unity 5.2+ only:
**DrawRawMesh** `(Vector2[] points, int[] triangles, Color fillColor)`
*Draws the passed mesh in dimensions specified per axis. Directly adds an element to the canvas with its vertices and triangles set to those passed in.*

# 4. Lines and Paths

Besides drawing solid shapes Primitive UI is also capable of line rendering, drawing lines and stroking paths right onto the Primitive Canvas.

There are two methods for putting lines on the Primitive Canvas:

**DrawLine** `(Vector2 point1, Vector2 point2[, StrokeStyle strokeStyle])`
*Draws a simple line between two points specified per axis. This method does not support line chaining, meaning lines are drawn naively and do not weld together.*

**DrawPath (**`Vector2[] points[, StrokeStyle strokeStyle, bool closePath]`**)**
*Draws a series of lines between points in dimensions specified per axis. Individual lines are welded together using the miter technique so avoid overdraw and ensure a constant line thickness.*

Where the solid shapes expected a fill color to be passed in, lines and paths expect something called a `StrokeStyle`. This is a simple class that holds information about the strokes, including the color. These properties are:

**color**
*The color of the line or path.*

**scaleMode**
*The way the line or path stroke scales when the Primitive Canvas is resized, or when the resolution changes.*

**thickness**
*The thickness of the line or path.*

*When `scaleMode` is set to `Absolute` the thickness is measured in pixels. An absolute-scaling line with `thickness` 4 will always be 4 pixels wide, no matter what.*

*When `scaleMode` is set to `Relative` the thickness is measured as being relative to the Primitive Canvas' width, just like a circle's radius for example. A relative-scaling line with `thickness` 0.05 will always have a stroke width of 1/20th the width of the Primitive Canvas. The line thickness will then scale along with the Primitive Canvas and resolution changes.*