Shapes

Quick Start Guide

This guide is a short version of the longer, more comprehensive online documentation
I recommend giving it a read! It's better than this lil guide~

Shapes has two main ways of drawing, either using Shape Components or Immediate-Mode Drawing

Shape Components

To create shapes in your scene, go to Tools/Shapes/Create/...
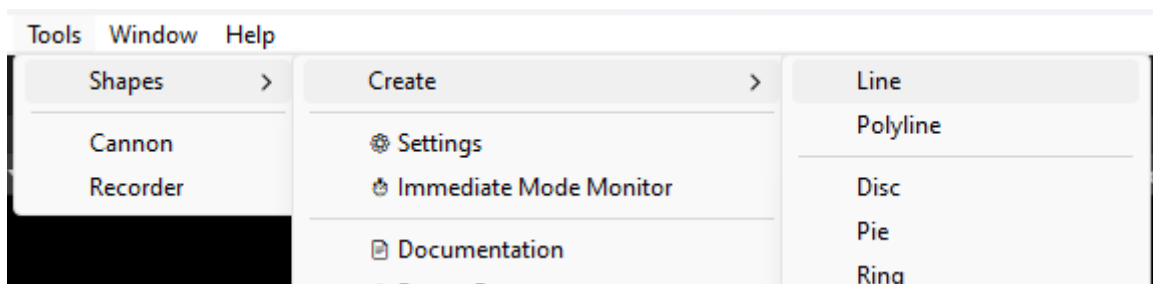
These Shapes work like any other mesh renderers - you can tweak their parameters in the inspector and move them around your scene with game objects! You can also modify their parameters from scripts using GetComponent<T>, and assigning to all the available properties.

Immediate-Mode Drawing

Drawing in immediate mode is useful for shapes that are only temporary, procedural, or where you don't want to deal with the overhead of many game objects. Immediate mode is instantaneous - it renders only once in the camera you draw in, so if you want to continuously draw, you draw every frame! This means it's very easy to create dynamic setups where what is being drawn can always change with no overhead of destroying or creating GameObjects

Example

```csharp
[ExecuteAlways] public class MyScript : ImmediateModeShapeDrawer {

    public override void DrawShapes( Camera cam ){

        using( Draw.Command( cam ) ){

            // Sets up all static parameters.
            // These are used for all following Draw.Line calls
            Draw.LineGeometry = LineGeometry.Volumetric3D;
            Draw.ThicknessSpace = ThicknessSpace.Pixels;
            Draw.Thickness = 4; // 4px wide

            // draw lines
            Draw.Line( Vector3.zero, Vector3.right,   Color.red   );
            Draw.Line( Vector3.zero, Vector3.up,      Color.green );
            Draw.Line( Vector3.zero, Vector3.forward, Color.blue  );
        }

    }

}
```

Attach this script to any object, and it will draw three 4px wide 3D lines along each axis at the center of the world, with different colors. Let's break it down:

ImmediateModeShapeDrawer

This is a helper class you can inherit from, that will make drawing in immediate mode a little bit easier when overriding the `DrawShapes()` method. This method is called once for every active camera, including the scene view if your script is marked with `[ExecuteAlways]`. Internally, this is just a wrapper for the camera OnPreRender callback, subscribed to in OnEnable and unsubscribed from in OnDisable. If you don't use this class, then it is highly recommended that you use the appropriate OnPreRender callback for your render pipeline when issuing draw commands

Draw.Command(...)

`using( Draw.Command( myCamera ) ){}` sets up a scope in which you can issue draw commands to the specified camera.

- Consecutive draw commands of the same type inside of this scope will be GPU instanced, so avoid alternating between types of shapes if you want to use less draw calls
- I highly recommended setting up draw commands in Camera.onPreRender or RenderPipelineManager.beginCameraRendering
- `ImmediateModeShapeDrawer` does this for you
- While you *can* set up draw commands in Update(), it is not recommended. The reason for this is because it's important to not add more commands than you're rendering, as this will stack draw calls. If you're worried about this, you can always go to `Shapes/Immediate Mode Monitor` in Unity, to make sure nothing is leaking

Shapes.Draw (static properties)

`Shapes.Draw` is the main class for immediate mode drawing, and has two core parts. First, it contains static properties that will configure how all following Shapes should be drawn. For example:

- `Draw.Color = Color.red;` will make all following Shapes default to red
- `Draw.Matrix = transform.localToWorldMatrix;` will make all following draw calls relative to that transform
- `Draw.LineGeometry = LineGeometry.Volumetric3D;` will make lines be drawn using 3D geometry instead of flat quads
- `Draw.ThicknessSpace = ThicknessSpace.Pixels;` will set the thickness space of lines to use pixels instead of meters
- `Draw.Thickness = 4;` will make all following lines have a width of 4 (pixels, in this case)

(This is just a handful of the many static properties in there that you might find useful)

Shapes.Draw (draw calls)

`Shapes.Draw` also contains the actual draw calls - in this case, we are using `Draw.Line` in order to draw simple line segments, but it contains lots of other shapes

- `Draw.Line` Draws a line between two points
- `Draw.Polyline` Draws a polyline along a path (see usage below)
- `Draw.Disc` Draws a solid disc/filled circle
- `Draw.Ring` Draws a ring/circle with a given thickness
- `Draw.Pie` Draws a pie/circle sector
- `Draw.Arc` Draws a circular arc
- `Draw.Rectangle` Draws a rectangle
- `Draw.RegularPolygon` Draws a regular n-sided polygon
- `Draw.Polygon` Draws a filled polygon (see usage below)
- `Draw.Triangle` Draws a triangle given 3 vertex locations in 3D
- `Draw.Quad` Draws a quad given 4 vertex locations in 3D
- `Draw.Sphere` Draws a sphere
- `Draw.Cuboid` Draws a cuboid with a given size along each axis
- `Draw.Cube` Draws a cube with a given size
- `Draw.Cone` Draws a cone with a given radius and length/height
- `Draw.Torus` Draws a torus with a given radius and thickness
- `Draw.Text` Draws text mesh pro text

Now, there are way too many overloads to list here, but all Draw functions follow the same pattern:
`Draw.ShapeName( [Positioning], [Essentials], [Specials], [Coloring] )`

- Any unspecified parameters will use the static properties
- The first few parameters are always for the positioning of the object
- Second we have the essentials, the most common parameters of the shape, such as the radius of a sphere
- Third, we have more special case parameters, that you might want to overload that you usually skip
- Finally, the color override of the shape is always at the very end

Drawing without commands

In some special cases, you may want to draw without using `Draw.Command`. For example, if you want to draw gizmos using Shapes, you can issue draw calls directly:

```
void OnDrawGizmos(){
    // set up all static parameters.
    // these are used for all following Draw.Line calls
    Draw.LineGeometry = LineGeometry.Volumetric3D;
    Draw.LineThicknessSpace = ThicknessSpace.Pixels;
    Draw.LineThickness = 4; // 4px wide

    // draw lines
    Draw.Line( Vector3.zero, Vector3.right,   Color.red   );
    Draw.Line( Vector3.zero, Vector3.up,      Color.green );
    Draw.Line( Vector3.zero, Vector3.forward, Color.blue  );
}
```

However, it's important to keep a few things in mind

- GPU instancing is not supported without `Draw.Command`, which means each draw corresponds 1:1 to draw calls on the GPU, which can get expensive with many shapes
- This will literally draw as soon as you call each draw call, which means that *when* during the render pipeline these draw calls happen is *absolutely crucial*, and in many cases they will simply not work, especially in HDRP and URP
- Drawing like this is equivalent to using Material.SetPass and Graphics.DrawMeshNow
- Generally, this is a bit of a legacy/advanced/bad way of drawing, so like, pls use `Draw.Command` instead~

Polylines & Polygons

Polylines and Polygons can be drawn in several ways. The first and most flexible one is to create a temporary `PolylinePath`/`PolygonPath`, then specify its points, and finally drawing it. This method will automatically dispose of mesh data at the end of its scope, since it's in a `using` block

```
using( Draw.Command( cam ) ){
    using( var p = new PolylinePath() ){
        p.AddPoint( -1, -1 );
        p.AddPoint( -1,  1 );
        p.AddPoint(  1,  1 );
        p.AddPoint(  1, -1 );
        // Drawing happens here:
        Draw.Polyline( p, closed:true, thickness:0.1f, Color.red );
    } // Disposing of mesh data happens here
}
```

The above code will recreate the mesh every time you draw it. If you instead want a persistent one that you can modify instead of recreate all the time, you can create a `PolylinePath`/`PolygonPath` in, say, `Awake`, and then, importantly, dispose it when you're done with it, usually in `OnDestroy` or `OnDisable`. This ensures any mesh data is cleaned up properly.
(the following example below presumes you're using ImmediateModeShapeDrawer)

```
PolylinePath p;

void Awake(){
    p = new PolylinePath();
    p.AddPoint( -1, -1 );
    p.AddPoint( -1,  1 );
    p.AddPoint(  1,  1 );
    p.AddPoint(  1, -1 );
}

override void DrawShapes( Camera cam ){
    using( Draw.Command( cam ) ){
        // Drawing happens here
        Draw.Polyline( p, closed:true, thickness:0.1f, Color.red );
    }
}

void OnDestroy() => p.Dispose(); // Disposing of mesh data happens here
```