# Shapes

Quick Start Guide

Website • Online Documentation • Changelog • Feedback

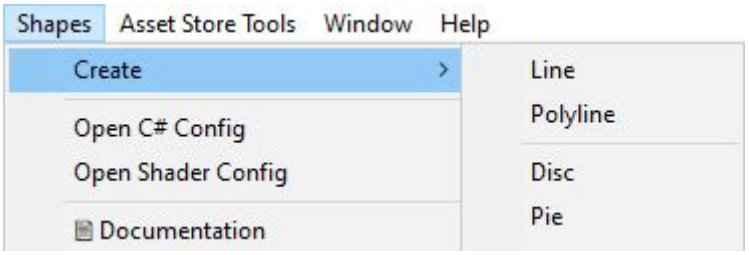
---

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 Shapes/Create/...




## Immediate-Mode Drawing

`Shapes.Draw` is the main class for immediate mode drawing, and has two core parts.
Drawing functions like `Draw.Line, Draw.Disc, Draw.Torus` etc. and static global properties for subsequent `Draw.X` calls, such as `Draw.Color, Draw.LineThickness, Draw.BlendMode`...

Example: drawing 4px world xyz basis vectors in 3D at the origin in OnDrawGizmos:

```
void OnDrawGizmos(){
    // set up all statics - these are used for following Draw 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  );
}
```

In this case, we used global properties for many of the line properties, but we set color per-line, overriding the global `Draw.Color`

## Drawing Contexts

Drawing in immediate mode with Shapes is equivalent to using Material.SetPass and Graphics.DrawMeshNow. This means that the timing and the context in which you draw matters a lot. Here are a few places you can call immediate mode drawing:

- MonoBehaviour.OnDrawGizmos, as shown above, useful for editor gizmos
- MonoBehaviour.OnPostRender, useful for drawing from scripts attached to your camera
- Camera.onPostRender, static event to subscribe to, receiving *all* camera OnPostRender callbacks, to ensure it's being drawn regardless of what camera is being used

## Polylines

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

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

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

```
PolylinePath p;
void Awake(){
    p = new PolylinePath();
    p.AddPoint( -1, -1 );
    p.AddPoint( -1,  1 );
    p.AddPoint(  1,  1 );
    p.AddPoint(  1, -1 );
}
void OnPostRender() => Draw.Polyline( p, closed:true ); // Draws here
void OnDestroy() => p.Dispose(); // Disposing of mesh data happens here
```