SaintsDraw

`SaintsDraw` allow you to draw arrow, circle, and arc in Unity, using Gizmos or LineRenderer

Deveoped by: TylerTemp , 墨瞳

Unity: 2019.1 or above

Installation

- Using OpenUPM

  ```
  openupm add today.comes.saintsdraw
  ```

- Using git upm:

  add this line to manifest.json in your project

  ```
  {
      "dependencies": {
          "today.comes.saintsdraw": "https://github.com/TylerTemp/SaintsDraw.git"
          // your other dependencies...
      }
  }
  ```

- Using a unitypackage:

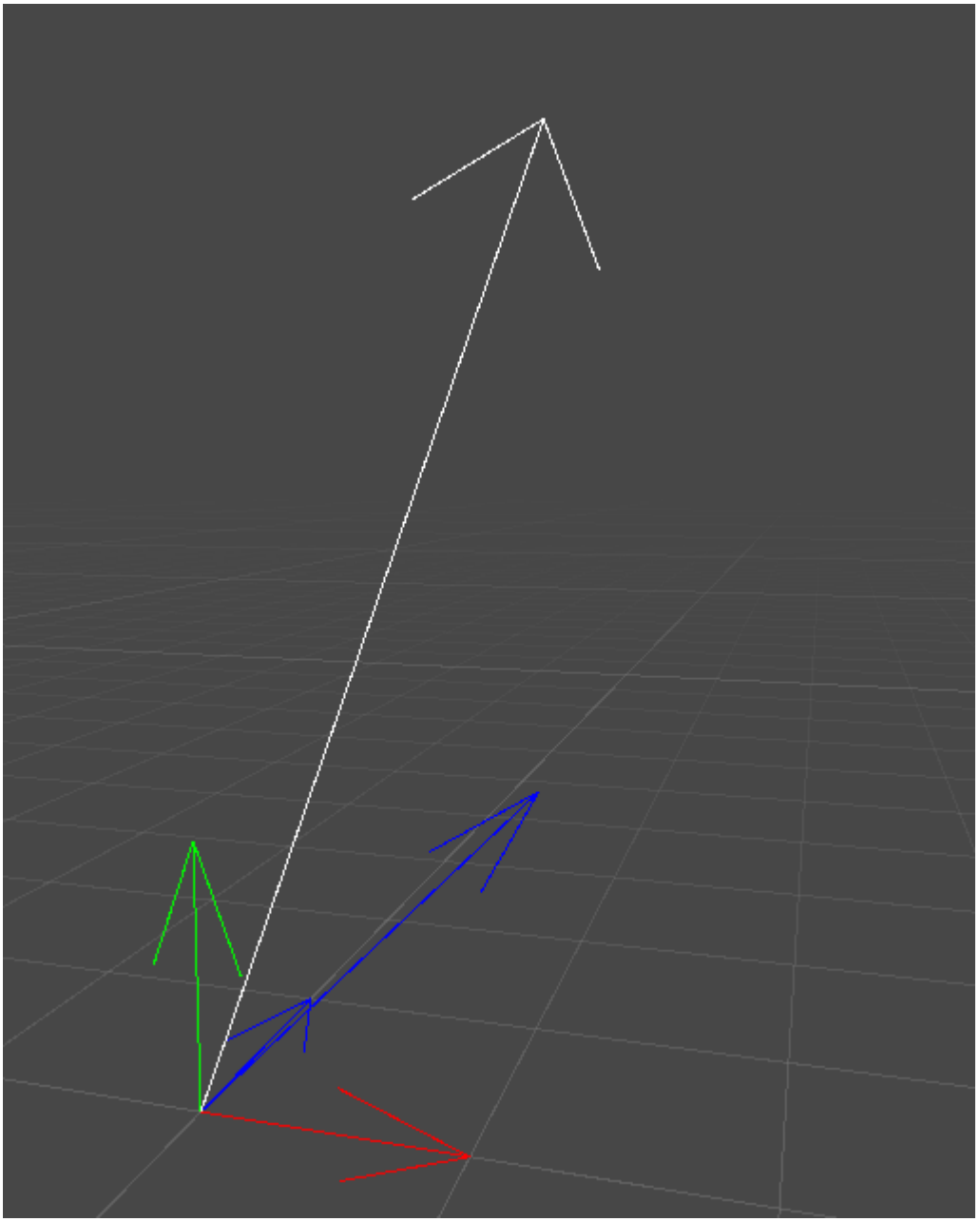
  Go to the Release Page to download a desired version of unitypackage and import it to your project

- Using a git submodule:

  ```
  git submodule add https://github.com/TylerTemp/SaintsDraw.git Assets/SaintsDraw
  ```

Draw

Arrow



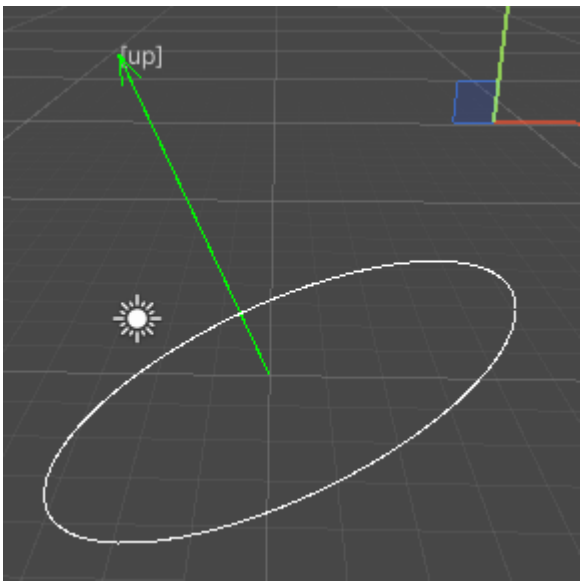
Using `Arrow.Draw` to draw an arrow, which has parameters:

- `Vector3 from` point where the arrow starts (tail)
- `Vector3 to` points where the arrow ends (head)
- `float arrowHeadLength = 0.5f`
- `float arrowHeadAngle = 20.0f`
- `Vector3? up = null` up direction of the arrow, default is `Vector3.up`. This is useful when you have some rotation on the arrow. The arrow is always perpendicular to this `up` direction.

Append a `LineRenderer` as the first parameter to draw the arrow using `LineRenderer`

```
using SaintsDraw;
Arrow.Draw(Vector3.zero, Vector3.one);
```

Circle (Disk)



Using `Circle.Draw` to draw an circle (disk), which has parameters:

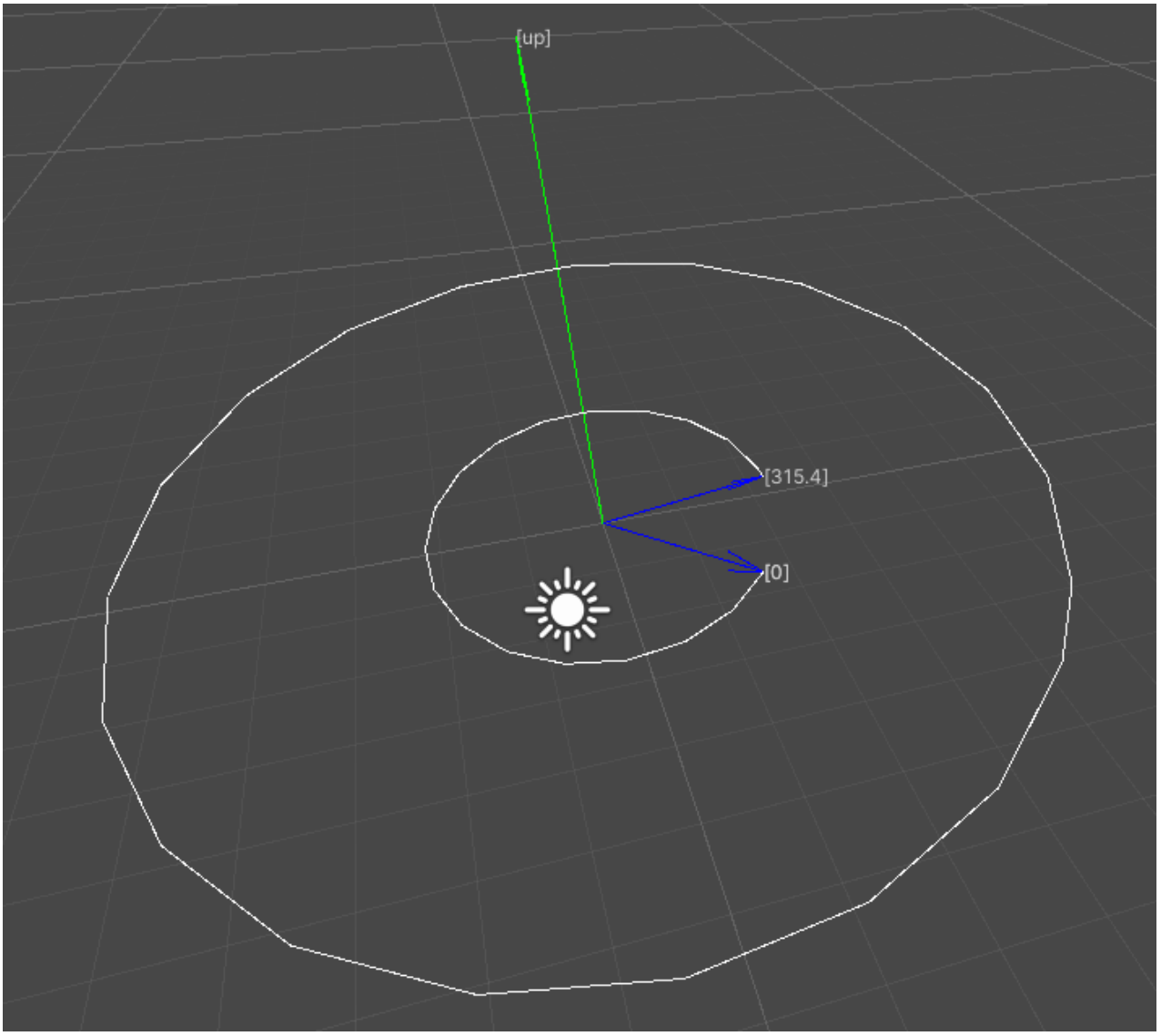
- `Vector3 center` center of the circle
- `float radius` radius of the circle
- `Vector3 upward` up direction of the circle. The circle is always perpendicular to this value. Usually `Vector3.up` is used
- `int numSegments` how many segments to draw for the arc. The bigger it is, the smoother the arc is

Using `Circle.DrawBySegCount` to draw an circle with fixed segment steps, which means each segment will have the same angle. It has the same parameters as `Circle.Draw` except `int numSegments` is replaced by `float segAngle`.

Append a `LineRenderer` as the first parameter to draw the arc using `LineRenderer`

```
using SaintsDraw;
Circle.Draw(Vector3.zero, 5f, Vector3.up, 40);
```

Arc



Using `Arc.Draw` to draw an arc, which has parameters:

- `Vector3 center` center of the arc
- `float radius` radius of the arc
- `float fromArc` angle to start
- `float toArc` angle to end
- `Vector3 upward` up direction of the arc. The arc is always perpendicular to this value. Usually `Vector3.up` is used
- `Vector3 plate` as the arc no has a plate which is perpendicular to the arc, this parameter is used to determine the plate's start point. It'll be automatically put on the plate defined by the `upward` direction.

  Usually `Vector3.left` or `Vector3.forward` is used
-

`int numSegments` how many segments to draw for the arc. The bigger it is, the smoother the arc is

Using `Arc.DrawBySegCount` to draw an with fixed segment steps, which means each segment will have the same angle. It has the same parameters as `Arc.Draw` except `int numSegments` is replaced by `float segAngle` .

Append a `LineRenderer` as the first parameter to draw the arc using `LineRenderer`

```
using SaintsDraw;
Arc.Draw(Vector3.zero, 5f, 60f, 120f, Vector3.up, Vector3.left, 40);
```

Some Tools

Gizmos Color

```
using (new ColorScoop(Color.green))
{
    Arrow.Draw(Vector2.zero, Vector2.up);
}
```

Gizmos Matrix

Useful if you want to draw gizmos in local space inheriting parent's scale and rotation

```
using (new MatrixScoop(transform.localToWorldMatrix))
{
    Arrow.Draw(Vector2.zero, Vector2.up);
}
```

Arc Tools

this will normalized your angle, which allow over 360 but will has no overlap

```
(float normFromArc, float normToArc) = Arc.NormalAngleRange(_fromArc, _toArc);
```

this will display an arrow from arc center to the angle you want to check, helpful when testing `upward` and `plate`

```
Vector3 startPos = Arc.GetDirection(_upward, _plate, angle).normalized * _arcRadis;
Arrow.Draw(Vector3.zero, startPos);
```