Runtime API

All Paint In 3D features can be accessed from the C# API.

The main classes you will use are: P3D_Brush, P3D_Tree, and P3D_Painter.

In your code, the first thing you should do is create a brush and give it the settings you want, for example:

```
var myBrush = new P3D_Brush();
```

```
myBrush.Color = Color.red;
```

NOTE: If you don’t want to allocate a new brush, then you can use the `P3D_Brush.TempInstance` static property.

Next, you should make a painter and assign it a texture (canvas), for example:

```
var myPainter = new P3D_Painter();
```

```
myPainter.SetCanvas(myGameObjectThatHasAMeshRenderer, "_MainTex", 0);
```

or

```
myPainter.Canvas = myPaintableTexture;
```

NOTE: If you don’t want to allocate a new painter, then you can use the `P3D_Painter.TempInstance` static property.

Now, you can paint using 2D coordinates, for example:

```
myPainter.Paint(myBrush, 25, 50); // Pixel coordinates
```

or

```
myPainter.Paint(myBrush, new Vector2(0.1f, 0.5f)); // UV coordinates
```

If you want to do more complex painting (e.g. raycast), then you need to use a tree, for example:

```
var myTree = new P3D_Tree();
```

```
myTree.SetMesh(myGameObjectThatHasAMeshFilter, 0);
```

or

```
myTree.SetMesh(myMesh, 0);
```

NOTE: If you don’t want to allocate a new tree, then you can use the `P3D_Tree.TempInstance` static property.

You can now search the tree using various methods, for example:

```
var result = myTree.FindNearest(myLocalPosition, 10.0f);
```

```
myPainter.Paint(myBrush, result);
```

or

```
var results = myTree.FindBetweenAll(myLocalStartPosition, myLocalEndPosition);
```

```
myPainter.Paint(myBrush, results);
```

NOTE: All tree coordinates are in local-space to the mesh. If you want to use world-space coordinates then inverse transform your coordinates using the transform your mesh belongs to.

NOTE: The P3D_Painter and P3D_Tree classes are only designed to work with one mesh and texture at a time. If you want to paint everything in the scene then look at the P3D_Paintable class, which wraps all of this functionality into a single component, and adds additional functionality.