Dynamic Text for Unity

Dynamically updated alternative for built-in Text Mesh, from Jetro Lauha / Strobotnik Ltd.

Table of Contents

Introduction

This package contains the Dynamic Text component. It is meant for displaying pixel-perfect camera-facing text, with size & position defined in world units. If compared to Unity's built-in text components, Dynamic Text is sharp like built-in GUIText, but part of the scene like TextMesh.

HINT – Also check out this component's official web page: www.strobotnik.com/unity/dynamictext/

How to add Dynamic Text to scene?

You can either create a new game object, or add this component to an existing game object.

To create a new game object with Dynamic Text, choose from menu:

GameObject → Create Other → Dynamic Text

To add Dynamic Text to an existing game object, choose from menu:

Component → Mesh → Dynamic Text

NOTE – If the object has an existing Text Mesh, it'll be converted to Dynamic Text.

Note that the Dynamic Text needs to have a reference to the camera where it is meant to be visible in. It will automatically grab the `Camera.main` (first camera with the "`MainCamera`" tag) if there is one.

If you don't have a main camera, or if you want the text to be visible in a different camera, you have to manually drag the correct camera from your Hierarchy view to the `Cam` field of Dynamic Text component.

If you are using custom layer for text stuff, make sure to also set the Layer manually for your Game Object (top of Inspector), and verify the Culling Mask of your Cameras.

Fields of the Dynamic Text component explained

The table below explains the fields which are visible in Inspector of the Unity Editor.

Cam	Reference to the camera where this text is meant to be visible. Initialized to `Camera.main` by default (first camera with the "`MainCamera`" tag).
Text	The text that is displayed, that is, to be generated as a mesh to this object's mesh filter. You can also paste in text and it can contain newline and tab characters (”`\n`” and ”`\t`” when setting from code; see Line Spacing and Tab Size). ***NOTE:*** *For script-updated and dynamically built strings you should use textSB field which is a String Builder. If you update text content using textSB, this Text string won't be kept up-to-date except when running in the editor.*
Offset Z	Additional Z offset – how far should the text be offset from the `transform.position.z`. Usually you'll just want to keep this as 0.
Size	Approximate text size (height), measured in world units. Actual height varies by font and by character, e.g. between upper- and lowercase characters.
Line Spacing	Baseline-to-baseline distance between two rows of text.
Anchor	Which point of the text shares the position of the Transform.
Alignment	How lines of text are aligned (Left, Center, Right).
Tab Size	Size of a tab stop, measured in world units. Tab char in text moves to the next tab stop. Note that this is defined differently when comparing to Text Mesh, which defines tab size just as a multiple of space char.
Font Style	The font style to use (Normal, Bold, Italic, BoldAndItalic), for the fonts that support it. Depending on font, styles may look different by platform.
Font	The Font to use – e.g. Unity's own default Arial. ***NOTE:*** *If you use a custom truetype (.ttf) font, in import settings you must define the Character field to be “Dynamic”!*
Color	The color used to render the text (used as vertex colors for the mesh).
Baseline Ref Char	The baseline font metric is measured from a reference character, which is usually the lower-case character “x”. Usually you don't need to change this, but you may need to set a different character if you have a font which has only certain characters like numbers or upper-case letters, for example.
Metrics Ref Chars	The ascent and descent font metrics are measured from the characters given by this list of characters. Again, if your font has only a subset of characters, you should change this field to contain a list of typical characters you're using with that particular font.

Pixel Snap Transform When enabled, Dynamic Text will continuously snap the `transform.position` to closest screen pixel boundary (as viewed from the camera set in the `cam` field).

Min Font Px Size Minimum font size measured in pixels (not world units). It's recommended to keep this as a small value like 6 or 10.

Max Font Px Size Maximum font size measured in pixels (not world units). If your text can occasionally be very very big or very close to camera, you can limit how large it can be rendered in the font atlas. Usually a good value for this field is something like 100 or 150.

The fields above can naturally be changed from code as well. Public field names, with their types and default values, are respectively as follows:

```
Camera cam;
string text = "Text";
System.Text.StringBuilder textSB;
float offsetZ = 0;
float size = 0;
float lineSpacing = 1.25f;
DynamicTextAnchor anchor = DynamicTextAnchor.BaselineLeft;
TextAlignment alignment = TextAlignment.Left;
float tabSize = 1.0f;
FontStyle fontStyle = FontStyle.Normal;
Font font;
Color color = UnityEngine.Color.white;
string baselineRefChar = "x";
string metricsRefChars = "xgjpflXOQ09";
bool pixelSnapTransformPos = true;
int minFontPxSize = 6;
int maxFontPxSize = 150;
```

In addition, there's following public properties:

```
Bounds bounds;  // Equals to bounds from the Mesh Filter
float baseline; // Distance to baseline from arbitrary top reference Y
float ascent;   // Distance from baseline to top of the glyphs
float descent;  // Distance from baseline to bottom of glyphs
```

What features does Dynamic Text have?

Most of the listed features are demonstrated in the `DynamicTextExamples` scene (folder in the package: `DynamicText/Examples/`).

Text size is proportional to world units, and not dependent on screen size in pixels or dpi resolution.

- For example, if you use an orthogonal camera in Unity with Size=5, it means the total height of window is 10 units. Dynamic Text with unit size of 2.0 will mean one row takes vertically 20% of screen so that there will be space for 5 rows of text.

For maximum sharpness, pixel size for text font rendering is dynamically selected to be the nearest matching size for text world unit size.

- For example, suppose your window height is 512 pixels and you have a camera like in the previous example. If you make a text with size 1.0, the row height will be 1 unit, corresponding to 51.2 pixels. The font will be rendered sharp with pixel size of 51.

Dynamic Text component also works with both orthogonal and perspective cameras.

- For perspective cameras, it measures distance from text object to the camera plane. With the power of trigonometry it figures out what is the correct pixel size for the text (with its current position and size, measured in world units).
- This is done separately for each different piece of text with varying distances.

You can choose if text is pixel snapped to keep it sharp when it moves on screen.

- With pixel snapping enabled, sub-pixel position of `transform.position` will be snapped to nearest pixel (of currently used screen resolution).

The Dynamic Text component uses Unity's internal font renderer and atlasing.

- So you can use same fonts what you can use with the built-in Text Mesh or GUI Text — use custom truetype fonts.
- Regular Font Styles are supported: normal, **bold**, *italic*, ***bold and italic***.
- Change color dynamically.
- Standard built-in font material & shader is used. No need for custom heavy-weight shaders.
- Note: There's no support for Rich Text with html-like syntax (for now, at least).

Text can be aligned and anchored, similar to how Unity's built-in Text Mesh (3D Text) works.

- Text can be left-, center- or right-aligned.
- Horizontally text can be left-, center- or right-anchored.
- Vertically text can be upper-, middle-, lower- or baseline-anchored. Dynamic Text defaults to **baseline anchoring** for easier working with text (baseline anchor is not featured by built-in Text Mesh).

You can ask for font metrics and use tabs for alignment.

- The component has baseline, ascent and descent properties.
- You can also get size of the whole text (bounds of the generated mesh).
- Tab size is defined in world units and tab char in text moves to the next tab stop.

The code contains many small tweaks to make it better.

- Mesh is regenerated/reallocated only when necessary. E.g. when there's longer text than previous frame, or if text properties change (size, style, anchor, alignment, ...). Previous mesh arrays are reused when possible.
- Dynamic Text is compatible with dynamic batching. It can even combine with built-in text meshes as same font material and texture can be used with both.
- Aware of different texel sampling between graphics APIs, and adjusts as needed to keep your text sharp.
- Automatically converts & replaces existing Text Mesh objects if you add a new Dynamic Text component to them.
- You can update text dynamically, even every frame. See the last screen (10/10) of example scene (called “interactive demo” in the web site). It has a "text mode" effect consisting of over 2000 characters updated every frame.
    - **NOTE:** When updating text with dynamic content, you should change it using the **textSB** string builder field instead of the **text** string. And after you have finished constructing the new dynamic text string, call **FinishedTextSB()**.
      C# example:

```
int playerScore; // assume you have an int like this
DynamicText dt = GetComponent<DynamicText>();
dt.textSB.Length = 0; // clear previous text content
dt.textSB.Append("Score: ");
dt.textSB.Append(playerScore);
dt.FinishedTextSB();  // signals that your new text is ready!
```

No native code & small footprint.

- Works with Desktop, Mobile and Webplayer platforms.
  (probably with consoles as well, but we haven't tried)
- Supplied as only a small DLL with the component and with bit of C# source for editor extension.
- No native code — works with the Free version of Unity!

Dynamic Text works also with the new Unity 2D.

- The default font material & shader seems to render using the Default Sorting Layer with Order 0. So, to make sprites render behind text, you can make a new Sorting Layer and drag it to be before Default. Or you can use Default Layer for your sprite but set the Order in Layer to negative value. And, of course, in similar way to force sprites render front of text.

What is it not good for?

As usual, there are some **trade-offs and caveats**, and this is not an exception.

Exact font metrics and glyph sizes vary a little bit when the font's actual pixel size is adjusted to the best match.

- So, if text size varies slowly and smoothly over time, the text "pops" a bit.
- This problem can be alleviated a little bit by using a custom font imported using Smooth Rending Mode (not Hinted Smooth).
- However, changing text size all the time leads to constant updating or re-generation of font atlas, so in any case this kind of use is not optimal for this component.

This is not a good choice when combined with a camera moving and rotating freely in 3D.

- For example, if your camera can get very close to text, it can dynamically resize to be very large on screen. This can lead to constant update of the font texture, which causes a drop in frame rate.
- 3D-rotating text semi-far from camera can still trigger constant font atlas update, since the actual size on screen may change all the time. This is basically the same limitation than with camera-facing text changing size all the time, as mentioned above.

This is not a solution for all your UI needs.

- Dynamic Text is just a component to show only text in your scene, nothing more.
- There's no support for rich text with html-like syntax (for now).

No support for special effects like gradients, drop shadows or outline. (*For now.*)

- But you can do basic versions of some effects yourself, by using some old tricks. For example, to make a simple drop shadow: Duplicate your text, setting color to black. Offset Y down a little bit and Z behind the original text.
- Some support for special effects may be added in a future version.

Questions and Answers

But there's GUI Text, and it's not blurry or jaggy like in the examples. Why isn't it enough?

Yes the GUIText is there. But it forces you to work in screen space using pixels. Same text can look way too small or big if you move it to a screen with different resolution or dpi than what you used initially. Check out again the two "What is the problem?"-examples on the Dynamic Text info web page. By the way, Dynamic Text looks just as sharp as GUI Text looks.

Why does my text look garbled?

Verify that the Mesh Renderer has material correctly set up. Normally it should be correct automatically, but sometimes one can set it wrong by mistake. (Expand the font in Assets view and drag Font Material to Element 0 in list of Materials of the Mesh Renderer.)

I can't see text, what's wrong?

1. Check that you have a font specified (i.e. Dynamic Text component's Font field doesn't display "None"). Drag a font there, or click the small dotted circle and select Arial from the list.
2. Try right-clicking Dynamic Text component in Inspector and choose "Regenerate Mesh".
3. Make sure the material is correctly set up (see previous question's answer above).
4. As always, make sure rendering of some other mesh doesn't obscure your text.

Why is the text sometimes blurry in the editor Scene view?

Text in the scene view shows with the size it is currently needed to be displayed sharply in the camera (game) view, which is the point of this Dynamic Text component. So, the text is blurry if it is small on the final view (either with small specified size when using orthogonal camera, or it is far when using a perspective camera). Also, check if your text is not visible in the camera view and instead of that it's behind the camera. In this case the text is also clamped to minimum font size in pixels ("Min Font Px Size").

Why is this particular text blurry or jaggy in the game view?

Make sure your text has a reference to the currently active camera. Or, if you have multiple cameras and text is on a custom layer rendered by one of the cameras, make sure the text refers to the correct camera which renders that layer. Text size on the game view is determined with the camera it refers to.

I have some of my own custom UI movement/alignment code. Pixel snapping "fights" with it, moving each frame. How can I fix this?

Move the DynamicText down in scene hierarchy so that it's a child of the object having your code (which has custom movement).

Is there garbage collector (GC) problems?

No, in the extent what can be reasonably expected. Dynamic Text allocates memory only when it needs to enlarge mesh arrays (e.g. if you switch to display a longer text than what you had previously). Also, if you want to update to dynamically built text from script code, you should change the text content using the supplied textSB field (String Builder) and call FinishedTextSB() when you're done. This way there's no leaks of strings of the previous content.

What if I have performance problems?

First, try these steps:

- Lower your “Max Font Px Size” and maybe don't use that big text in your design. Big letters take lot of texture space and there's a limit how much you can have. It seems on some mobile platforms the max atlas may be limited to smaller than you can get on a desktop machine.

- Use less characters for the “Metrics Ref Chars”. These characters (and the baseline ref char) are always requested to be in the font atlas texture, so if you are using a large text size, Unity has to try to fit all of these letters in that large size to the atlas. So, if for some text you only use numbers, put only “0123456789” here. Or, with normal text, it may be enough to have only 1-2 lowercase low-hanging characters like 'j' or 'g' and 1-2 full-height capital characters like O or X.

If that doesn't help, please read the following about other possible situations:
As mentioned and shown in example scene (interactive demo), it is not a good idea to have a screen with a camera rotating in 3D (see screen 8/10). Also it's not a good idea to constantly adjust size of a text in 2D either (see screen 5/10). That is, if the final text size on screen changes constantly due to 3D rotation or changing distance to camera. This will force Unity to flush and re-create the font atlas often. You can try to alleviate the problem by lowering the Max Font Px Size of Dynamic Text objects. However, in this case the text will appear blurry if it gets close to camera. This is one of the cases where you still might want to use the standard Text Mesh instead. If you need text in varying 3D angles which is also sharp when close to camera, then you should consider using one of the font packages which use distance field based font rendering. Note that those typically don't support dynamic fonts but use bitmapped pre-processed fonts instead, and they will require use of a custom heavy-weight shader, so what they're good at comes with their own set of limitations. If you know you need such a thing, here's links to some assets you might want to check out: VFont, Text Box. (note: we haven't tried these, so we don't know how good they are in practice)

How to fix this error message: No camera - can't generate mesh?

The Dynamic Text component needs to have a reference to the camera where it is meant to be visible in. It will automatically grab the Camera.main (first camera with the "MainCamera" tag) if there is one.

This is just like basic text should work!

That's not a question... but, thanks! That was the goal.

Feedback, Feature Suggestions and Bug Reports

Send by email: contact@strobotnik.com. Write “Dynamic Text” in the subject.