VText
Virtence GmbH
Altenburger Str. 13
04275 Leipzig

VText 2.0
Unity Package

info@virtence.com

Februar 15, 2017

1 Quickstart

VText is Unity package to generate dynamic 3D–Text geometry directly from truetype fonts (`.ttf` or `.otf`).

The main features are

- Different Materials for
  - Frontface
  - Bevel
  - Sides
- Kerning
- Texture layout suitable for Lightmapping
- Available build targets
  - Android (Arm 32–bit)
  - Linux (32/64–bit)
  - Windows (32/64–bit)
  - OSX (32/64–bit)
  - iOS (Arm 32/64–bit)
- Layout along AnimationCurve
- Circular bending with animatable radius
- Automatic generation of Colliders and or RigidBodies.
- Experimental: Adding arbitrary components to each letter (like AudioSources, Animations, Scripts, etc)

1.1 Install

After importing the VText package, select the menu ***Windows/Virtence/Setup VText...***

This will show up a dialog like in figure 1.

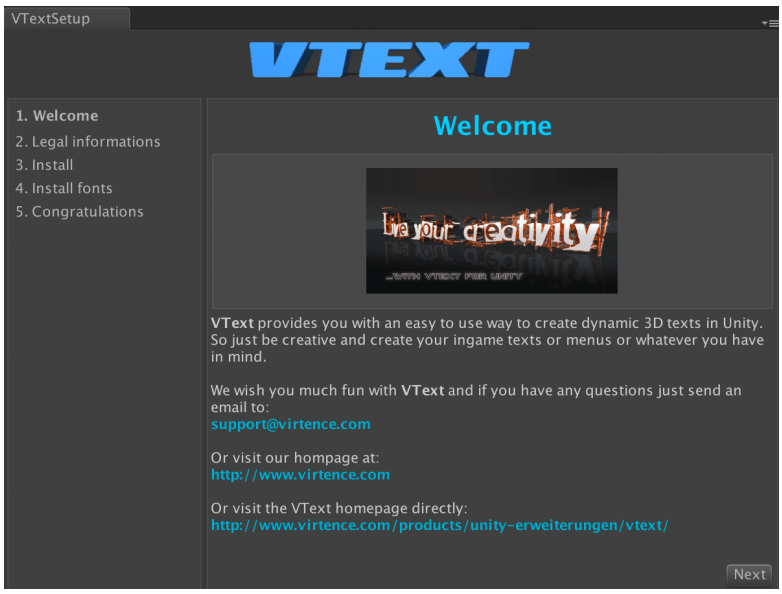

Figure 1 – Setup

Just follow all installation steps. You can go to the next step by clicking the "Next" button in the lower right.

1.2 Demo scenes

Important!
Before opening the test scenes, ensure you setup VText like in section 1.1. To make the demo scenes work you must install the demo fonts we provide. You can do that in ***Step 4 (Install fonts)***.

You will find the VText demo scenes in the `Virtence/VText`-folder. VText comes with the following demo scenes:

1.2.1 StartScene

StartScene is a simple animated VText scene with a graphical user interface. In play mode Unity will show something like figure 2. Its intention is to show some common settings for the 3D and layout parameters.

Figure 2- StartScene

1.2.2 TextureLayoutScene

Figure 3 shows the Unity views for TextureLayoutScene. This shows the used UV-Layout which has light baking, etc. in mind. See *section 3.2* for detailed information about texture layout.

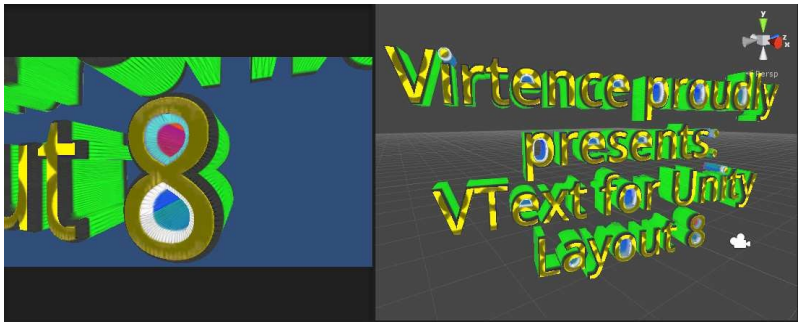

Figure 3 - TextureLayoutScene

1.2.3 AdditionalMonobehaviours

Figure 4 shows a small scene with self-written scripts attached to each letter of the text.

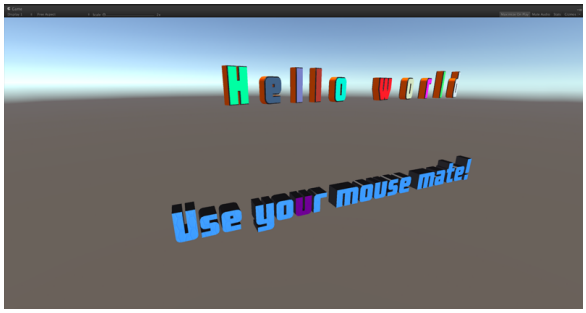

Figure 4 - Additional Monobehaviours

In this scene, you can see how to rotate the single letters around their own center and change their materials. Furthermore, you can highlight any letter of the bottom text with the mouse. This demonstrates the usage of adding own scripts automatically to each letter.

1.2.4 Physics

This scene shows you how to use Colliders and Rigidbodies for you VText objects. It looks like figure 5.

Figure 5 - Physics

Here you can throw small balls to the letters of the VText. This demonstrate the automatic generation of Colliders and Rigidbodies to the letters.

1.2.5 Equalizer

Here we show how you can modify the bending curves of a VText object at runtime. The scene will look like in figure 6.

Figure 6 – Equalizer

The text in this scene will wiggle to the sound by changing its bending curves to get an equalizer effect.

1.3 Adding VText objects to your scene

Clicking on the `GameObject/Virtence/VText` menu item will add an empty child to your hierarchy, which contains the VTextInterface component.

The Inspector editor pane shown in Figure 7 will allow you to adjust the settings.

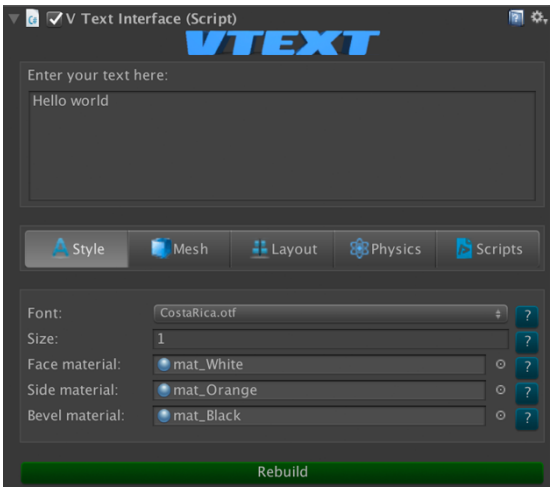

Figure 7 - Inspector

We describe the User Interface and the parameters in the next chapters.

To work properly you should have at least one font installed in the ***Assets/StreamingAssets/Fonts*** folder because for new texts we choose the first font we find there and if there isn't any we can't generate the text.

Note: All Fonts in that folder will be distributed when building for mobile devices! So, keep only fonts which are required in your project to reduce the memory footprint. Also be aware of copyrights!

2 User Interface

In this chapter, we will describe the different panes of the User Interface.

2.1 Common

There are three components which are visible all the time (except the VText-Logo of cause ☺).

The first component is the **text input field** which looks like in figure 8.

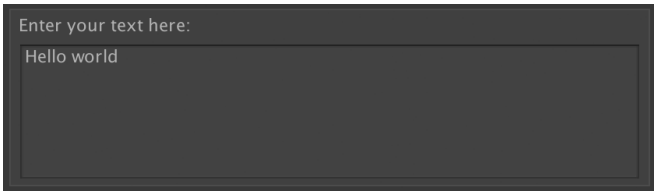

Figure 8 - Text input field

This allows you to enter the text which will be generated. Keep in mind that this input field has **no word wrapping** on purpose because you can create multiline text with VText. To do so you need to press the return key explicitely to start a new line.

The second component which is always visible is the **toolbar** (see figure 9).

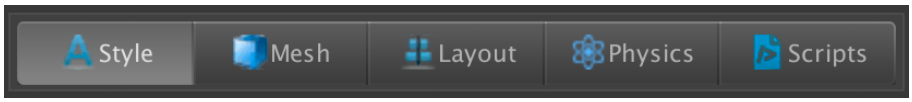

Figure 9 – Toolbar

This allows you to switch the main categories of a VText object.

We distinguish between the following categories for now:

- **Style** (fonts and materials)
- **Mesh** (everything which affects the 3D geometries like the depth of the letters, the bevel, shadow-casting or –receiving, tessellation-quality, etc.)
- **Layout** (here we handle layout parameters like alignment, bending, size and spaces)
- **Physics** (add colliders or rigidbodies to your letters)
- **Scripts (experimental!)** (add any Unity components to your letters, like scripts, animations, sound, etc.)

We explain the sections in the following chapter in detail.

The third component which is always visible is the Rebuild-button like shown in figure 10.

Figure 10 – Rebuild

You can see this button as a fallback. This will rebuild all aspects mentioned above of your VText object.

2.2 Style

Here we define the style parameters of the text. The inspector view for the style parameters looks like figure 11.

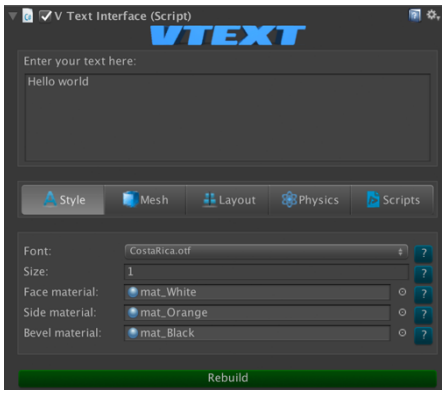

Figure 11 - Style

2.2.1 Font

Here you can define the font which is used for the generated geometry. Keep in mind that only fonts are usable which are located at the ***Assets/StreamingAssets/Fonts*** folder. That's because we want to make sure that your fonts are distributed correctly if you build the application to an executable for any provided system.

2.2.2 Size

This will define the size of the generated geometry in meters.

2.2.3 Materials

To understand the materials, it's a good start to see how VText-letters are organized. Figure 12 shows you a single letter with its components.

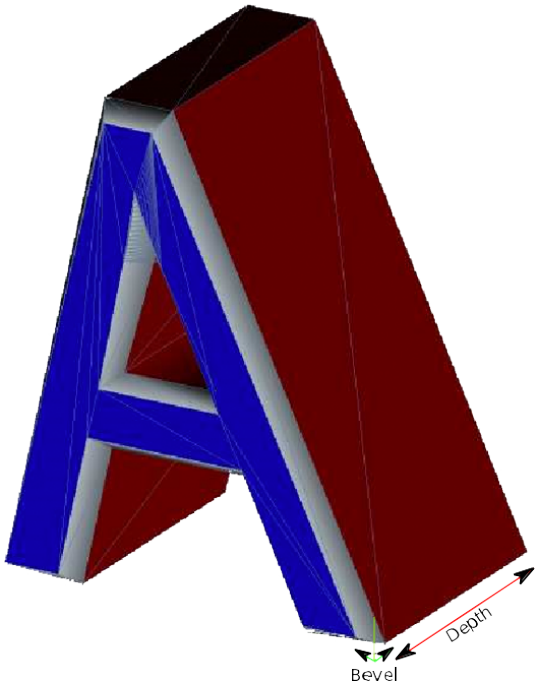

Figure 12 - Organization of a letter

The blue part in figure 12 shows the so called **face** part. The gray part is called **bevel** and is a smoothly curved surface between the blue and the red part … which is called **side**. And that's exactly what the several materials in the **style** section describe. You can define a material for each of those parts.

2.3 Mesh

The Mesh section allows you to adjust parameters which affects the generated 3D geometry. It will look like figure 13.

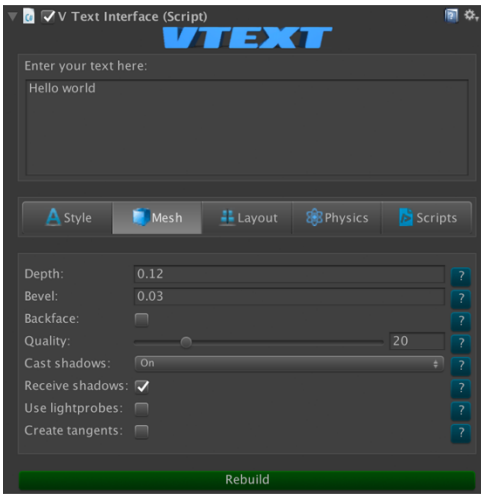

Figure 13 - Mesh

Here you can adjust the following parameters:

- **Depth:** this describes the depth of each letter. This makes your text three dimensional. You can set the material for these faces by changing the *Side material* in the *Style section*. Setting the depth to 0 creates a 2D text which can be placed in the 3D world and gives us a couple of performance rele
- **Bevel:** this is the smoothly rounded part between the front- or back-faces and the side of each letter. That's why **you can set the bevel size only if you have defined a depth greater than 0.** You can change the *bevel material* in the *style section*.
- **Backface:** often you will not see the backfaces of the text. That's why we do not generate them per default. But if you bend or rotate your text you maybe want to create the backfaces too. This parameter allows you to switch them on or off. If you enable the backfaces we create the geometry for the back parts of the letters and the bevel on the backside as well.
- **Quality:** The quality parameter allows you to adjust the number of triangles generated for the 3D text. Of cause, less triangles means better performance but less smoothness. When creating the geometry, we mainly adjust the curved parts of the letters.

- **Cast shadows:** This allows you to specify if (and how) the generated text should cast shadows. The parameter values are the same as the *cast shadow* parameters of *Unity's MeshRenderer component*.
- **Receive shadows:** here you can specify if your texts should receive shadows or not
- **Use lightprobes:** if your scene contains lightprobes then you can define here if your text should be affected by them.
- **Create tangents:** some shader (esp. normal or bumpmapping shader) require tangents to work correctly. If you use such shaders for your text you should enable this. Keep it off if you don't need them.

2.4 Layout

Within this section you can adjust all parameters which change the layout or bending of your texts. The layout section will look like figure 14.

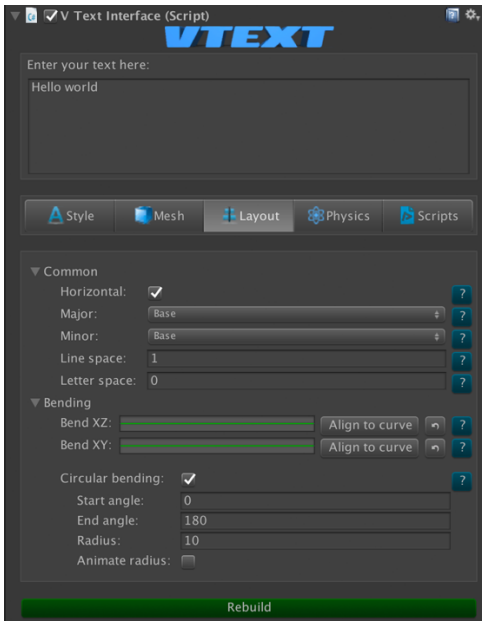

Figure 14 - Layout

2.4.1 Common

The following parameters will help you to adjust the layout of your text to your needs:

- **Horizontal:** here you can set the main layout of your text … if you want it is layouted horizontally or vertically. This will also influence the parameters *Major* and *Minor*.

- **Major:** This defines the alignment of your text depending on its *main layout (horizontal or vertical)*. If it is set to *horizontal*, then the Major alignment works in x-direction of you text. If it is set to *vertical*, then the Major alignment works in y-direction. The following values can be set:
    - **Base:** This is only used if your text is setup horizontally and only used for the Major alignment. Then the pivot point of your text is set to the baseline of the font. In all other cases, it works the same way like **Start** (see the next value).
    - **Start:** The pivot point of your text is set to the *top-left* of the entire text.
    - **Center:** The pivot point is set to the center of the entire text.
    - **End:** The pivot point is set to the bottom-right of the entire text.
    - **Block:** This aligns the text like you are using the **Start** setting but it will set the space between words in a way that all lines have equal length. *(**Note:** This works only if you set it in **Major** mode (horizontal and vertical) and only if you have multiple lines. Actually we do **NOT** trim whitespaces from the beginning or end of the line and they are taken into account for the calculation … just because maybe you intend this. Maybe we put in a parameter for this later or switch over to trim whitespaces automatically. So if your results looks wrong please double check if you have whitespaces at the beginning or end of line.)*
- **Minor:** This describes the alignment in exactly the opposite direction of the *Major* alignment. So, if your text is layouted horizontally then this describes the alignment in y-direction. If your text is layouted vertically this describes the alignment in x-direction. The values are the same like for the *Major* alignment (see *Major*).
- **Line space:** This parameter defines the distance of the **base line** of one line to the **base line** tex of the next line. This means that a line space of 0 will put all lines on top of each other. In this way, you can create overlapping texts too. A value of 1 means the "*normal*" line space as they are defined by the font. A value of 2 for instance doubles the distance between two lines.
- **Letter space:** Using this parameter you can adjust the distance between two letters. A value of 0 means that there is no extra space between the letters. They are rendered "*normally*".

2.4.2 Bending

In the bending section of the layout you can bend your text in a couple of ways. All methods described here have the following in common. To bend your text we use **Unity Animation Curves** which also allows it to animate these curves and therefor animate the bending of the text. Furthermore, each bending allows you to align the single letters to your defined curve (the Circular Bending will always align your letters along the circle … but we will come to it later).

- **Bend XZ:** This allows you to bend your text in *z direction*. This means that your text will stay on the ground but you can change its direction along z. The result will be something like shown in figure 15.

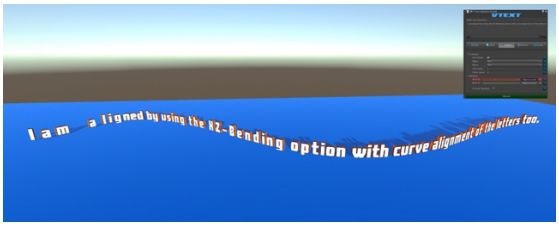

Figure 15 - Bend XZ

- **Bend XY:** This allows you to bend your text in y *direction*. This means that your text will move up and down. The result will be something like shown in figure 16.

Figure 16 - Bend XY

- **Circular bending:** The circular bending allows you to bend your text around a (*imaginary*) circle or a part of it. This is often used to rotate a text around a sphere or cylinder. We bet you know this text rotating around the earth logo like in figure 17?

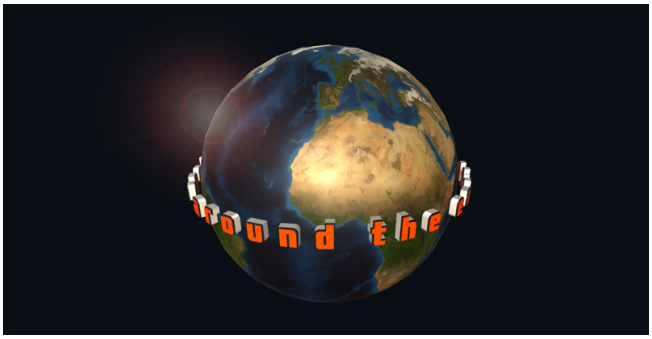

Figure 17 - Circular bending example

To setup a circular bending you can change the following parameters:

- **Start angle:** this will set the start position of your text. It is defined in degrees around the circle. Values lower than 0 degree and higher than 360 degree will result in a multiple bending around the circle.
- **End angle:** this defines the end position of your text on the circle. It is defined in degree too. Values lower than 0 degree and higher than 360 degree will result in a multiple bending around the circle.
- **Radius:** This defines the radius of the circle. Changing this parameter moves your text farther away from the center of your VText object.

If you enable the **Animate radius** button then you have the chance to change the radius along your text. This is not an animation per se (we will come to it later) but it uses Unity's animation curves again to modify the radius along your text. The values of the curve should be defined in x-direction in a 0-1 range where 0 is the beginning of your text and 1 is the end. The y-values are multiplied with the radius you defined above.
This allows you in a simple way to bend your text like a spiral for instance (see figure 18)

Figure 18 - Change radius along the text

Tip:
There is no need to use just one of the parameters mentioned above. Be creative and combine them.
This way you can combine the **Bend XY** curve to be a slope and additionally add a **Circular binding** like described before to get a tornado-bending like in figure 19.

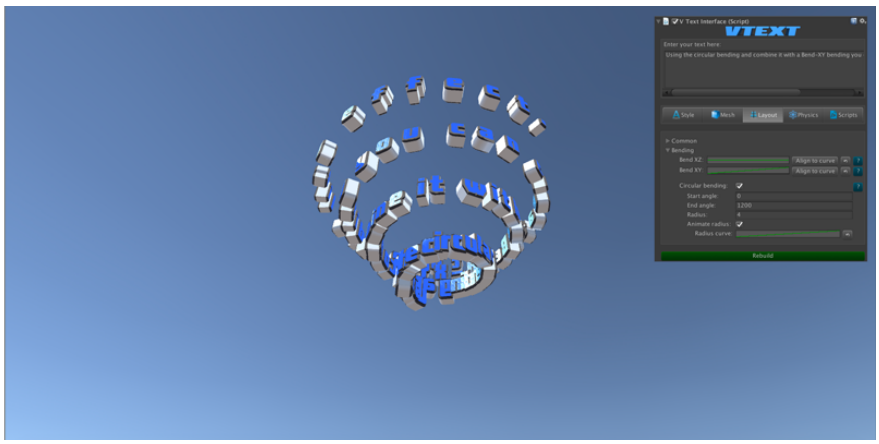

Figure 19 - Tornado example

2.4 Physics

In the **physics section,** you can automatically add **RigidBodies** and **Collider** to each of the generated letters. This allows you to interact with them later or add physic behaviours to it.
The physics section will look like in figure 20.

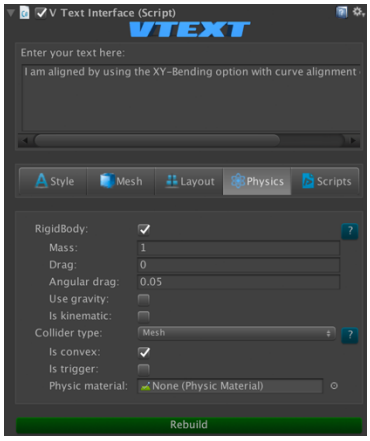

Figure 20 - Physics

Actually, we allow the generation of **box- and mesh colliders** with the common parameters like they are defined by Unity.
Also for rigidbodies we provide you with the common parameters known from Unity.
See the **Physics example scene** for a quick start.

2.4 Scripts

Be careful, this section is experimental and can lead to Unity crashes!
Here we want to give you the possibility to add arbitrary components to each of the generated letters automatically. This would allow you to add components like Animators, AudioSources, your own scripts, etc to each letter and therefore giving you much more flexibility for later usage of the text.
The scripts section looks like figure 21.

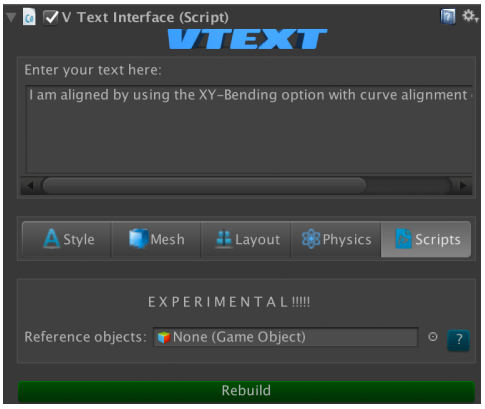

Figure 21 - Scripts

The way we handle it is that you create a reference gameobject (simply an empty gameobject) and add all components you want to it. You can disable this game object in your scene so it does not affect your scene.
On this reference gameobject you can adjust all settings which you want to have on each letter.
Now drag and drop your **reference gameobject** into **the Reference object field in the script section**. If new letters are generated we take all components and their settings from the reference object and copy them to the new generated letter.

Some components (and their subtypes) will not be copied because we generate them in VText or Unity does not allow it at all. **The following components will not be copied:**

- Transform
- Renderer
- MeshFilter
- Rigidbody
- Collider

We also know that adding **scripts which have a VTextInterface as a public variable** (which is assigned in the Inspector) **can lead to deadlocks** which will result in Unity crashes. There is no problem if you get access components in your scripts by using the **GetComponent** methods.

3 Programming Interface

All mentioned sections and parameters can be accessed by using the programming interface too. This will allow you to **change or even animate each parameter** at runtime.
With the exception of the choosen font and the size of the generated text, all parameters can be found in the VTextInterface programming section corresponding to the sections in the VText User Interface. **The choosen font can be found in the parameter section** and **the size of the generated text can be found in the layout section**. We must keep it this way for backward compatibility reasons.

To access the VText interface you can use the GetComponent method from Unity like:

```
VTextInterface vi = GetComponent<VTextInterface>();
```

Or you can create a public variable of the type VTextInterface and drag and drop the VText interface of your text into it in the Inspector.

3.1 Sections

Corresponding to the VText User Interface we splitted up the Programming Interface into the same sections.

3.1.1 VTextInterface parameter

The following aspects can be changed by using the VTextInterface class:

Property	Type	Description
RenderText	string	The Text to generate.
parameter	VTextParameter	Mesh construction parameter.
layout	VTextLayout	Text layout parameter.
Physics	VTextPhysics	The physics paramter
AdditionalComponents	VTextAdditionalComponents	Additional components

Table 1: VTextInterface

3.1.2 Mesh parameter

The following aspects can be changed by using the mesh parameters section of the Programming Interface:

Property	Type	Description
Depth	float	The relative depth of extruded glyph
Bevel	float	Outline width and depth–shift
Backface	bool	Will generate back side if `true`
GenerateTangents	bool	Will generate tangents on each glyph if `true`
Fontname	string	The font file name

Table 2: Mesh parameter

You can access them by using something like the following:

```
VTextInterface vi = GetComponent<VTextInterface>();
vi.RenderText = “My text”;

// this must be in Assets/StreamingAssets/Fonts
vi.parameter.FontName = “MyFont.ttf”;
vi.parameter.Depth = 1.0f;
vi.parameter.Bevel = 0.03f;
vi.parameter.Backface = true;
```

3.1.2 Layout parameter

To access the layout specific parameters you can change the following parts of the VTextInterface. You can access it by:

```
VTextInterface vi = GetComponent<VTextInterface>();
vi.layout.Major = VTextLayout.align.Center;
```

for instance.

Property	Type	Description
Horizontal	bool	Major layout direction. Vertical layout if `false`.
Major	align	Major layout mode.
Minor	align	Minor layout mode.
Size	float	Local scaling factor for each glyph.
Spacing	float	Local scaling factor for minor adjustment.
CurveXZ	AnimationCurve	Z–Position AnimationCurve.
OrientationXZ	bool	Text faces tangent in Z if `true`.
CurveXY	AnimationCurve	Y–Position AnimationCurve.
OrientationXY	bool	Text faces tangent in Y if `true`.
CastShadows	bool	Generated glyphs will cast shadow.
ReceiveShadows	bool	Generated glyphs will receive shadow.
UseLightProbes	bool	Text will use Light Probes if available.

Table 3: Layout parameter

3.1.3 Physics

Here you can access the physic parameters of your text. To change them you can do something like:

```
VTextInterface vi = GetComponent<VTextInterface>();
vi.Physics.Collider=VTextPhysics.ColliderType.Mesh;
```

3.1.4 Scripts

In the same way you can access the Scripts section of the VText interface:

```
VTextInterface vi = GetComponent<VTextInterface>();
vi.AdditionalComponents.AdditionalComponentsObject = myRefObj;
```

3.2 Textures

The UV–Coordinates of each glyph are constructed in a light mapping conformal way.

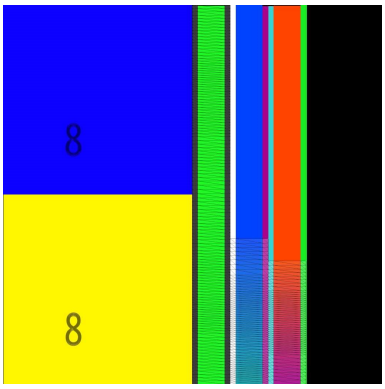

Figure 22 - Texture layout

As shown in Figure 22, the front face consumes the yellow UV–range $(0,0)\ldots(0.5,0.5)$. The blue UV–range $(0,0.5)\ldots(0.5,1.0)$ covers the backface area. Each contour side covers an U–range of $(0.5,0.5+n*0.1)$ where n is the index of active contour. Bevel range is dependant on its amount, and whether backface rendering is enabled. The V–range is normalized from the length of the longest contour. Figure 23 shows the resulting unlit material in Unity.

Figure 23 - Textured VText

You may wonder why front- and back–face layout covers such a small area in the UV–range. It's because the maximum glyph size of `DroidSans.ttf` is that much larger than the common used glyphs. Other fonts may show different ranges.

3.3 Performance

As long as you are using only 2D–Text[1] all VText objects referencing the same font use only one dictionary for glyph lookup. So Unity is able to batch most of your text, even if some use different sizes.

3D–Text requires a separate glyph dictionary for each VText object. So batching is only possible in VText itself.

It is generally no good idea to change parameters at runtime, which require a complete rebuild of the glyphs[2]! Especially text with a lot of different glyphs will need some time to rebuild. A change of layout parameter is not that costly, because no rebuild of glyphs is required.

Depending on the settings of **Need Tangents**, extra attributes are generated. This is only required if one of your materials contains a bumpmapping shader. For best performance turn it off if not required.

If the parameter **Backface** is turned on, additional faces are generated for each glyph. If you'll never see the backfaces in your scene, turn it off.

3.4 Build

If you change or set the name of the font via script, you should be aware of the **case sensitivity** of filenames on non Windows based operating systems! If your text will not show up in your build, first check the right spelling(including case) of the fontname.

[1] depth and bevel zero

[2] such as changing Depth or Bevel

4 Legal

4.1 3rd party

This package uses the following 3rd party products:

freetype2

```
Portions of this software are copyright ©2014 The
FreeType Project (www.freetype.org). All rights
reserved.
```

Fonts

This package contains the following fonts, which are licensed under the SIL Open Font License:

- CostaRica.otf
- Dotrice-Bold-Condensed.otf
- Lack.otf
- Lobster.otf
- Segment14.otf
- Xolonium-Bold.otf
- RacingSansOne-Regular.ttf