Writing and drawing on textures

Inhaltsverzeichnis

1 Overview

Text textures and Script textures can be used for both scenery objects and vehicles. They can be used to show static text on an object (Bus/tram stop sign, streetsign or vehicle number) and to show dynamic text (Destination display, driver information).

Text- and Script textures are configured in the Content-Tool. Every object can have several Text textures. For every Text texture the font and the connected variable are configured. Script textures are also configured there, but don't need font information.

1.1 Text textures vs. Script textures

- Text textures are considerably less flexible than Script textures. They can just work with a single texture and single color. It is placed at a fixed position and offers no further draw functions.
- Script textures are way more flexible: Any number of colors can be used, individual pixels or rectangles can be drawn and any number of texts with individual fonts can be places at a position. For this however, the object needs to have a script that uses the draw functions, otherwise the texture will remain empty.

2 Setup of a Text texture

Configuration is done on the left in the section "General settings" using "Object Properties". There you can find the section **Drawable Textures**. Use the "Add" button to add a new Text texture. There you can find all configuration options.

2.1 Variables

To set up a Text texture TWO variables have to be set up:

- a String variable, this controls what text is displayed on the Text texture. When the variable is updated

in the script is not important - when the variable is updated also the Text texture is updated. A line end can be achieved in the script as follows: `stringvar := 'first line'+#13#10+'second line';`.
- an integer variable called Texture-Index variable. This is on the one hand used in the configuration of the Text texture and on the other hand as Texture Variable in the Material properties. This variable is only written by LOTUS and **must not be changed** in the script!

Both variables must be created in the script beforehand.

2.2 Further Options

- Width / Height: The size of the final Text texture in pixels. The larger the texture the smaller the writing (for the same font).
- Text Texture: This box is normally *checked*. When it is not set then a Script texture is used as opposed to a Text texture.
- Font: The font to be used. You can choose all fonts that come with LOTUS, that are installed by Addons and all fonts you created yourself.
- Specify color: If this box is *not* set then the original textures of the font are used. If it *is* set then the font is colored completely in one color, only the alpha channel of the font is used.
- Color: If "Specify color" is set then the color to be used can be specified here. "A" stand for alpha channel and is ignored here.
- Orientation: How will the text be aligned? Center, left and right are assumed to be known. "Centered in integer" means that the writing centered in such a way that in the exact center there is always a whitespace. The idea for this is that it is possible to make a stop display with 2 segments, where the dark border between segments always overlaps with a whitespace in the text. Otherwise the center letter would be dragged into two.
- Grid: This also has a background in displays. A "display pixel" is usually larger than a "font pixel" (e.g. because the pixels are not just rectangular but are circular or look completely different). Because it looks better, when the orientation is "Center" the displayed text would continually slip towards its actual grid. On the other hand, if the size of a "display pixel" including the distance to the next pixel is entered in "Grid", then LOTUS always aligns the text to this grid - the "Display pixels" are therefore always located correctly in the grid.

2.3 Remark:

All areas not covered by a font character are color black with alpha channel "0" (black). Therefore, the usual procedure is that first a single-colored background is modeled and then for the text itself an additional polygon is placed, which then receives an alpha-controlled transparency in the material properties (see material properties).

3 Using a text-texture

A Text texture can only be used if it is in a separate material. This material should (as we just discussed) have an Alpha transparency. Now for Standard texture in the field on the right of texture name the Texture-Index variable of the Text texture we just created (and that we want to use) is set.

As a helper for mapping the text texture, you can use the button `Exp. Text Texture` in the *Test Environment* tab: First, you have to configure your text texture, then you have to open the script test dialog and to enter an example text into the linked script variable. Finally you click on the button *Exp. Text Texture*. The tool will ask you which text texture you would like to export (the index) and the filename for the bitmap. Two bitmaps will be exported: One with the color channels and one (with _alpha postfix) with the alpha channel.

These textures will help you now mapping the object properly.

4 Creating a new font

Instead of using an existing font you can also create a new font in the Content-Tool. In the main menu choose "**Bitmap Font**".

The basis for a font is two bitmaps following a distinct convention. There is a Color bitmap and an Alpha bitmap. We have seen before that the characters are "cut out" using the Alpha channel. This means that the Color bitmap can simply be one single color and the actual design of the characters is done in the Alpha bitmap.

As example this is the Alpha bitmap for the **IBIS2_7x5** bitmap:

It is very important to notice that the characters are placed in a very specific way. Above a character there are exactly 2 rows of pixels, under these rows the letters start, no further space. That second row is only meant as a visual separation and is completely ignored. The first row in the Alpha channel (and only in the Alpha channel) is used as coding where the characters start and end (that is, how wide they are). Between (or before or after) characters there can be any distance. Only the columns that have the coding in the first row are used, the rest is ignored.

The combination between the bitmaps and the actual characters is made by entering the correct ordering of the letters (as text) in the "Letter Sequence" field.

Further configuration is specifying the vertical height and horizontal distance. The vertical height should already include the distance to the next line. Also note that the height of the bitmap has to be at least this height value plus two pixels (the coding pixels). The horizontal distance is the distance between two characters in a horizontal direction.

5 Unicode

LOTUS is completely Unicode compatible and therefore supports in principle the sheer amount of normal, special, foreign and exotic writing systems as well as special characters and symbols. This also extends to fonts, there are however two things to notice:

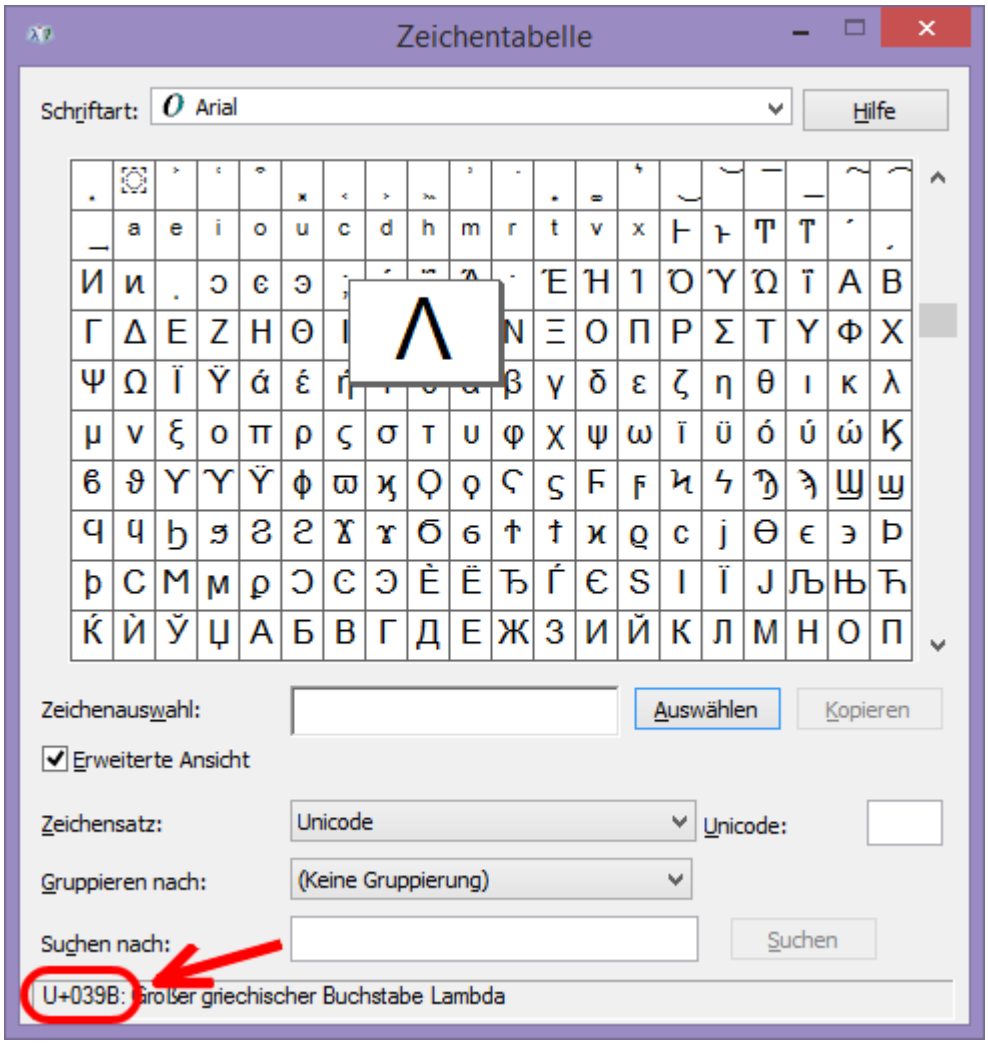
5.1 Characters must be included in the font

It is obvious that for a character to be displayed it must be part of the Font bitmap. If one disregards the memory size of the bitmap, Latin, Cyrillic, Greek, Thai, Arabic ... characters can be added to the font. If you do not include them, then of course you will not be able to use them.

5.2 The script itself is not Unicode compatible

Less obvious is the fact that the script files themselves are not in Unicode format! So if a string with Unicode characters is written directly from the script into a variable, then a somewhat strange detour must be used.

First we need to find out the Unicode value of a character. The easiest tool for this is the Character Map. Select your character and you can find the Unicode value below:



After you have found the values you can use them as follows:

Code

```
1. stringvar := 'The greek capital Lambda: ' + UChar($039B) + '. The rest is in latin letters.';
2. //If the text uses all non-latin characters
3. stringvar :=
4. UChar($0393) + // Gamma
5. UChar($0394) + // Delta
6. UChar($0398) + // Theta
7. UChar($039B) + // Lambda
8. UChar($039E); // Xi
```

This results in the following string: "?????".

If the string is entered using a text field in the Map-Editor (e.g. streetsign) or a data file (e.g. destination or stop display) then you don't need this detour.

A special case are the Unicode characters, which are (hexadecimal) five to eight digits long, these are in particular the emoticons and local traffic symbols. For reasons that would go too far here, these characters must be encoded with two UChar commands (? UTF-16BE).

The symbols supported by our fonts start with $1F6##. These characters must be encoded as follows: UChar($D83D) + UChar($DE##). The "tram from the side" symbol, which officially has the code $1F683, has thus to be coded in LOTUS with UChar($D83D) + UChar($DE83).

6 Script textures

Script textures are textures where LOTUS does not place the text on the texture. They are drawn on using script functions.

A Script texture is set up in the same way as a Text texture, except for the "Text texture" checkbox which is unchecked. All options underneath are ignored. The writing on the Script texture is done using a sequence of special procedures/functions in the script.

6.1 Performance Tip:

If a texture is updated using these functions during the SimStep, the transfer of the updated texture into the graphics memory is done automatically at the end of the script. Because this process is not easy performance wise, you should try to avoid unnecessary changes in the texture. Drawing the same rectangle on the texture every frame is heavy on performance and also unnecessary as the texture doesn't change. It would be better to intercept such unnecessary drawing commands by suitable tests.

6.2 Script Commands

The provided commands for scripts are presented here in order of use:

`procedure TexSelTex(Self; id: integer);` Example: `TexSelTex(Self, 0);`

The first step in the SimStep procedure is obviously selecting the texture to be written/drawn on. The first parameter, like all future procedures and functions, is "Self". The index corresponds to the index of the Text texture in the object properties.

`procedure TexSetColor(self: integer; col: cardinal);` Example: `TexSetColor(Self, Color(0,0,255,255));`

Another function is hidden in here: `function Color(r,g,b,a: byte): cardinal` translates the four color components (red, green, blue and alpha channel) into a single 32 bit integer, which can then be used internally and also by "TexSetColor". In the example the color is bright blue.

TexSetColor is used to set the color which will be used for the next stepts.

`procedure TexSetBlendMode(self: integer; mode: byte);` Example: `TexSetBlendMode(Self, 1);`

Using this procedure you can select (for now only for "TexWriteLn:) how the font will be drawn on the texture:

- 0: The character rectangles are printed completely onto the texture. The background completely disappears.
- 1: "Hard alpha transparency": Only the characters themselves (using their alpha channel) are printed on top of the existing background. There are no full rectangles like with option "0". There is however only a binary option: "draw" or "not draw", Semi-transparency is not possible.
- 2: "Soft alpha transparency": Like with option "1" the alpha channel is used to determine what to draw. In this case opacity of the characters also varies depending on the alpha channel. This makes Anti-Alias possible. Nevertheless this option should only be used when it is absolutely nesscecary as is it slower than option "1".

`procedure TexClear(self: integer);` Example: `TexClear(Self);`

This covers the entire texture in the color selected using TexSetColor.

`procedure TexDrawPixel(self: integer; x, y: word);` Example: `TexDrawPixel(Self, 10, 20);`

This command draws one single pixel in the color selected with TexSetColor. In this example the pixel colored is at position x = 10 and y = 20.

`procedure TexDrawRect(self: integer; x1, y1, x2, y2: word);` Example: `TexDrawRect(Self, 10, 20, 30, 40);`

This command draws a (filled) rectangle in the color selected by TexSetColor. In this example the left side is at x = 10, the right side is at x = 29, the top side is at y = 20 and the bottom side is at y = 39. Attention: For reasons of logic / mathematics the last coordinate is omitted. In this way, the width of this rectangle is exactly 30-10 = 20 pixels, otherwise the width would be 21 pixels.

`function TexReadPixelColor(self: integer; x, y: word): cardinal;` Example: `col := TexReadPixelColor(Self, 10, 20);`

This function is used to read the color of a selected pixel. In the example the color at location x = 10 and y = 20 is stored in the variable "col".

`function TexGetFontIndex(self: integer; fontUserID, fontSubID: integer): integer;` Example: `fontstd := TexGetFontIndex(Self, 1000, 210);`

This function should **only** be used in the Initialisation. It sets the (internal) index of the font which is then used by the relevant script commands to determine to font to use. In the example the font with ContentID 1000:210 is chosen.

`procedure TexWriteLn(self: integer; text: string; x, y: integer; fontindex: integer; fullcolor: boolean; addSpaces: byte);` Example: `TexWriteLn(Self, 'testABC', 5, 5, fontstd, true, 3);`

This command writes a text at any position on the texture. In the example the text "testABC" is printed in the Font 1000:210 selected before at position 5/5 (left upper corner). It is printed printed in a single color (selected using "TexSetColor") with a spacing of 3 pixels. That is, there are a further 3 pixels between each character.

`function TexGetTextLenPixel(self: integer; text: string; fontindex: integer): integer;` Example: `lenpixel = TexGetTextLenPixel(Self, 'Testtext', fontID);`

This function calculates the length of a string in pixels, without actually writing them into the texture. This function is very useful in the programming of a "full matrix display" which should change fonts based on the

length of the text. In the example the length of the string "Testtext" in the font "fontID" is calculated and stored in the variable "lenpixel"

```
procedure TexCopy(self: integer; sourceId, targetId: integer; x1,y1,x2,y2,targetX,targetY: integer; scalingfactor: integer; sourceColor, targetColor: cardinal);
```

Depending on the color of the pixels in a certain area of a texture, this function colors a certain area of a second texture with a desired color, possibly with an integer factor. The two textures are defined with sourceId and targetId (see TexSelTex), the area to be copied extends from (x1/y1) to (x2-1/y2-1). The upper left corner (i.e. x1/y1) is aligned to the target texture at the position (targetX/targetY). The scaling factor must be greater than or equal to 1. sourceColor defines the color that the pixels of the source texture to be checked must have in order to draw into the target texture with the color targetColor. If the source texture on a certain pixel has a color different from sourceColor, then this pixel is not transferred.