

# Texturen beschreiben und bemalen

## Inhaltsverzeichnis

- [1 Überblick](#)
  - [1.1 Text-Texturen vs. Script-Texturen](#)
- [2 Einrichten einer Text-Textur](#)
  - [2.1 Variablen](#)
  - [2.2 Anmerkung:](#)
- [3 Verwenden der Text-Textur](#)
- [4 Erstellen eines neuen Fonts](#)
- [5 Unicode](#)
  - [5.1 Zeichen müssen im Font enthalten sein](#)
  - [5.2 Das Script selbst ist nicht Unicode-kompatibel](#)
- [6 Script-Texturen](#)
  - [6.1 Performance-Tipp:](#)
  - [6.2 Script-Befehle](#)

## 1 Überblick

[Text-Texturen](#) und Script-Texturen können sowohl für Szenerieobjekte als auch für Fahrzeuge eingerichtet werden und können verwendet werden, um ein Objekt statisch (z.B. Haltestelle, Straßenschild oder Wagennummer) oder dynamisch (Zielanzeige, Fahrerdisplays) zu beschriften.

Im ContentTool erfolgt die Konfiguration der Text- und Script-Texturen, wobei für jedes Objekt beliebig viele eingerichtet werden können. Für jede Text-Textur werden der Font und die verknüpfte Variable sowie weitere Einstellungen festgelegt. Script-Texturen werden ebenfalls dort konfiguriert, aber benötigen keine Font-Information.

### 1.1 [Text-Texturen](#) vs. Script-Texturen

- [Text-Texturen](#) sind wesentlich "starrer" als Script-Texturen. Sie können nur einen Font mit nur einer Farbe verarbeiten, welcher an einer festen Position platziert wird und haben auch keine weiteren Zeichenbefehle. Dafür benötigen sie aber auch keine zusätzlichen Script-Befehle.
- Script-Texturen sind sehr viel flexibler: Es kann mit beliebig vielen Farben gearbeitet werden, es können einzelne Pixel oder Rechtecke gezeichnet werden und es können beliebig viele Texte mit unterschiedlichen Fonts an beliebigen Stellen platziert werden. Dafür benötigt das Objekt aber ein Script mit den entsprechenden Zeichen-Befehlen, andernfalls bleibt die Textur "leer".

## 2 Einrichten einer Text-Textur

Die Einrichtung erfolgt links im Abschnitt "Allgemeine Einstellungen" über die "Objekteinstellungen". Dort befindet sich der zugehörige Abschnitt **Drawable Textures**, bei dem mittels "Hinzufügen" eine neue Text-Textur angelegt wird. Dann können dort alle Konfigurationen vorgenommen werden.

### 2.1 Variablen

Um eine Text-Textur einzurichten, müssen ZWEI Variablen eingerichtet werden:

- eine String-Variable, die steuert, was auf der Text-Textur ausgegeben werden soll. Wann sie vom Script geschrieben wird, ist unerheblich - sobald sie verändert wird, wird auch die Text-Textur aktualisiert. Ein Zeilenumbruch wird im Script über folgende Formulierung erreicht: `stringvar := 'erste Zeile'+#13#10+'zweite Zeile';`
- eine Integer-Variable, die sog. Textur-ID-Variable. Diese wird einerseits in der Konfiguration der Text-Textur verwendet und andererseits als Textur-Variable in den Material-Einstellungen. Diese Variable wird von LOTUS geschrieben und darf (im Script) **nicht geändert werden!**

Beide Variablen müssen zuvor im Script angelegt werden.

## Weitere Optionen

- Breite / Höhe: Die Größe der finalen Text-Textur in Pixeln. Je größer die Textur, desto kleiner die Schrift (bei selbem Font)
- Text-Textur: Dieser Haken wird standardmäßig *gesetzt*. Wird er nicht gesetzt, dann handelt es sich nicht mehr um eine Text- sondern um eine Script-Textur.
- Font: Der zu verwendende Font. Zur Verfügung stehen alle bei LOTUS mitgelieferten, alle durch Addons installierten und die selbst erstellten Fonts.
- Fullcolor: Ist dieser Haken *nicht* gesetzt, dann werden die Original-Farben des Fonts verwendet. *Ist* er hingegen gesetzt, dann wird der Font einfarbig gezeichnet; nur der Alphakanal des Fonts wird genutzt
- Farbe: Sofern der Haken "Fullcolor" gesetzt wird, kann hier die zu verwendende Farbe eingegeben wird. "A" bedeutet Alphakanal und wird hier ignoriert.
- Orientierung: Wie wird der Text ausgerichtet? Links-, Rechtsbündig und zentriert dürften wohl bekannt sein. "Ganzzahlig zentriert" bedeutet, dass der Schriftzug zwar eingemittet wird, aber dafür gesorgt wird, dass sich in der exakten Mitte stets ein Zwischenraum befindet. Hintergrund ist hierbei die Möglichkeit, halbierte Haltestellenanzeige zu realisieren, deren dunkler Mittelsteg immer mit einem Zwischenraum zusammenfallen soll, da andernfalls der mittige Buchstabe auseinander gerissen wird.
- Gitter: Auch hier soll auf eine Eigenheit einer Anzeige zurückgegriffen werden: Da der besseren Optik wegen üblicherweise ein "Anzeigen-Pixel" größer als ein "Font-Pixel" ist (z.B. weil die Pixel nicht einfach nur rechteckig sind, sondern kreisförmig oder ganz anders aussehen) würde im Modus "zentrieren" der Anzeigentext andauernd - je nach Text - gegenüber ihrem eigentlichen Raster verrutschen. Wird bei "Gitter" dagegen die Größe eines "Anzeigen-Pixels" samt Abstand zum nächsten Pixel eingegeben, dann richtet LOTUS den Text immer an diesem Raster aus - die "Anzeigen-Pixel" befinden sich daher immer korrekt im Raster.

## 2.2 Anmerkung:

Alle Bereiche, die nicht von einem Font-Zeichen beschrieben werden, erhalten die Farbe Schwarz und den Alpha-Kanal "0" (ebenfalls "schwarz"). Daher ist die übliche Vorgehensweise die, dass zunächst ein einfarbiger Untergrund modelliert wird und dann darauf für den Text selbst ein zusätzliches Polygon gelegt wird, welches dann in den Materialeigenschaften eine Alpha-gesteuerte Transparenz erhält (siehe Artikel Materialeigenschaften).

## 3 Verwenden der Text-Textur

Die Text-Textur kann nun verwendet werden, indem zunächst ein separates Material angelegt wird. Dieses sollte - wie soeben erläutert - zunächst eine Alpha-Transparenz erhalten. Nun wird unten bei der Standard-Textur im Feld rechts vom Texturnamen die Textur-ID-Variable derjenigen Text-Textur ausgewählt, die angewendet werden soll.

Als Hilfestellung für das Mapping gibt es die Schaltfläche `Text-Textur exp.` im Abschnitt *Test-Umgebung*:



Symbolen. Dies gilt auch für die Fonts – es sind allerdings zwei Dinge zu beachten:

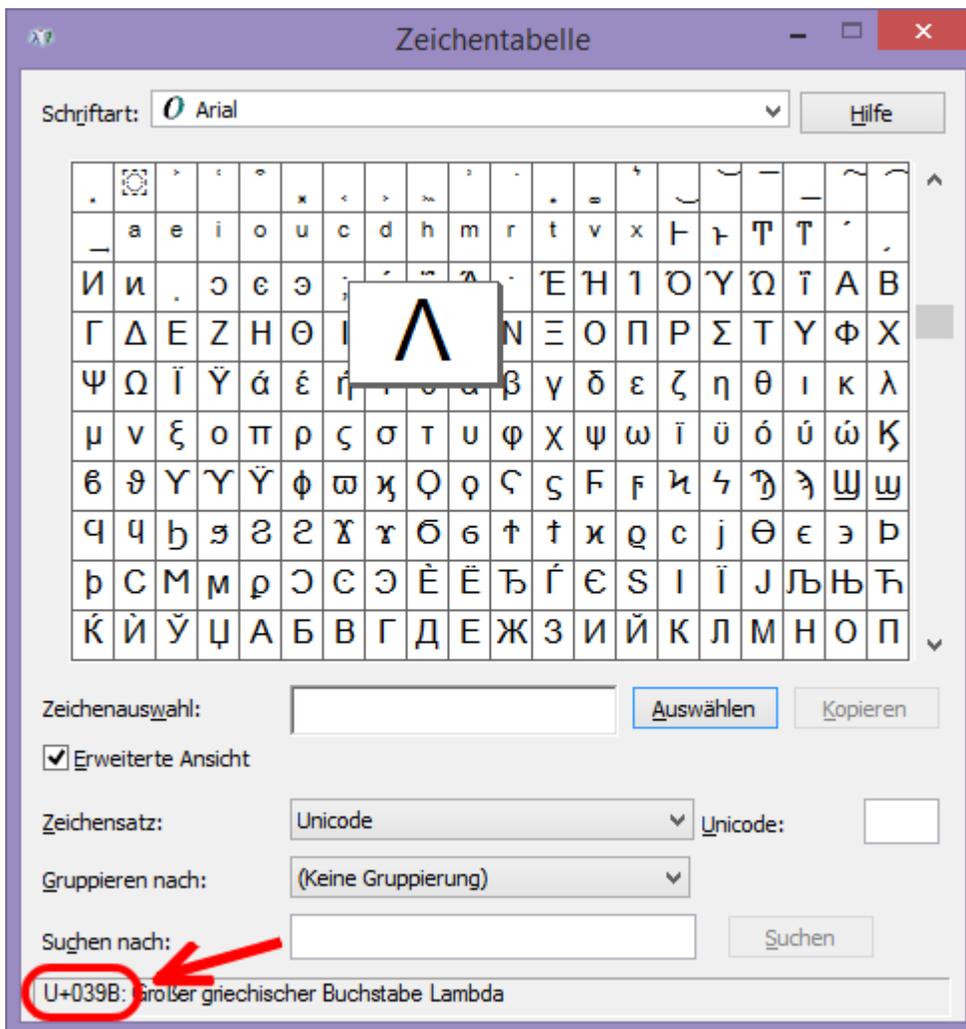
### 5.1 Zeichen müssen im Font enthalten sein

Sehr einleuchtend dürfte die Forderung sein, dass natürlich nur jene Zeichen dargestellt werden können, die auch Teil der Font-Bitmap sind. Wenn man mal von der speichermäßigen Größe der Bitmap absieht, können nach Belieben lateinische, kyrillische, griechische, thailändische, arabische... Schriftzeichen in den Font gepackt werden. Tut man das nicht, dann können sie aber natürlich nicht angezeigt werden.

### 5.2 Das Script selbst ist nicht Unicode-kompatibel

Weniger offensichtlich ist die Tatsache, dass die Scriptdateien selbst nicht im Unicode-Format vorliegen (und damit eine große Besonderheit darstellen)! Soll also ein String mit Unicode-Zeichen direkt aus dem Script heraus in eine Variable geschrieben werden, dann muss ein etwas sonderbar anmutender Umweg gegangen werden.

Zunächst müssen die Unicodes der Schriftzeichen ermittelt werden. Das gängigste Tool hierfür ist die Zeichentabelle, bei der man nach Auswählen des gewünschten Zeichens den Unicode direkt ablesen kann:



Sind die Codes ermittelt, können sie wie folgt ins Script eingebaut werden:

Code

1. stringvar := 'Der griechische Buchstabe Lambda: ' + UChar(\$039B) + '. Der Rest sind lateinische Buchstaben.';
2. // wenn der ganze Text besondere Buchstaben benutzt:
3. stringvar :=
4. UChar(\$0393) + // Gamma
5. UChar(\$0394) + // Delta
6. UChar(\$0398) + // Theta
7. UChar(\$039B) + // Lambda
8. UChar(\$039E); // Xi

Das Resultat ist dann der folgende String: "?????".

Entstammt der String allerdings einer Texteingabe im Map-Editor (z.B. Straßenschild) oder einer Datei (z.B. Ziel- oder Haltestellenanzeige), dann werden keinerlei derartige Umwege benötigt.

Eine Besonderheit stellen die Unicode-Zeichen dar, die (hexadezimal) fünf- bis achtstellig sind, das sind insbesondere die Emoticons und Nahverkehrssymbole. Aus Gründen, die hier zu weit führen würden, müssen diese Zeichen mit zwei UChar-Aufrufen kodiert werden (Stichwort UTF-16BE).

Die von unseren Schriftarten unterstützten, fünfstelligen Zeichen beginnen alle mit \$1F6##. Diese Zeichen müssen wie folgt kodiert werden: UChar(\$D83D) + UChar(\$DE##). Das Symbol "Straßenbahn von der Seite", das offiziell den Code \$1F683 hat, wird somit in LOTUS kodiert mit UChar(\$D83D) + UChar(\$DE83).

## 6 Script-Texturen

Mit Script-Texturen sind jene Texturen gemeint, die nicht von LOTUS automatisch mit Text beschrieben werden, sondern die über Script-Befehle "bemalt" und beschrieben werden können.

Eine Script-Textur wird genauso wie eine Text-Textur eingerichtet – bis auf die Tatsache, dass der Haken "Text-Textur" deaktiviert wird und alle darunter befindlichen [Optionen](#) ignoriert werden. Das Beschreiben der Script-Textur erfolgt mit einer Reihe von speziellen Prozeduren/Funktionen aus dem Script heraus.

### 6.1 Performance-Tipp:

Wird die Textur in einem SimStep von den vorgestellten Prozeduren verändert, erfolgt automatisch die Übertragung der veränderten Textur in den Grafikspeicher nach Abschluss des Scripts. Da es sich hierbei um einen in Bezug auf die Leistung nicht trivialen Prozess handelt, sollten unnötige Änderungen der Textur vermieden werden. Es wäre also z.B. sehr ungünstig, wenn in jedem Frame dasselbe Rechteck auf die Textur gemalt wird, obwohl sich die Textur hierzu nicht verändert. Besser wäre es, durch entsprechend geeignete Prüfungen derartiger unnötige Zeichenbefehle abzufangen.

## 6.2 Script-Befehle

Die vorhandenen Zeichenbefehle fürs Script werden hier in der Reihenfolge der üblichen Nutzung vorgestellt:

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

Der erste Schritt in der SimStep-Prozedur ist üblicherweise das Auswählen der zu bemalenden/beschreibenden Textur. Der erste Parameter ist – wie bei allen folgenden Prozeduren und Funktionen auch – auf "Self" zu setzen. Der Index entspricht dem Index der Text-Textur in den Objekteinstellungen.

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

Zunächst habe ich hier eine weitere Funktion versteckt: `function Color(r,g,b,a: byte): cardinal` wandelt die vier Farb-Komponenten (Rot, Grün, Blau, Alphakanal) in eine einzige 32bit-Ganzzahl um, mit der intern weiter gearbeitet wird und die von "TexSetColor" verwendet wird. In diesem Fall ist das reines Blau.

Mit TexSetColor wird nun die Farbe gesetzt, die für die folgenden weiteren Schritte verwendet werden soll.

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

Mit dieser Prozedur kann (zur Zeit nur für "TexWriteLn" - s.u.) eingestellt werden, wie der Font auf die Textur gezeichnet werden soll:

- 0: Die Buchstaben-Rechtecke werden komplett auf die Textur geschrieben. Der Hintergrund verschwindet dort komplett.
- 1: "Harte Alpha-Transparenz": Nur die Buchstaben selbst werden – unter Verwendung ihres Alphakanals – auf den bestehenden Untergrund gezeichnet. Es entstehen keine umreißende Rechtecke wie bei "0". Es wird jedoch nur unterschieden zwischen "zeichnen" und "nicht-zeichnen". Halbtransparenzen sind nicht möglich.
- 2: "Weiche Alpha-Transparenz": Wie bei "1", aber zusätzlich wird die Deckkraft abhängig vom Alphakanal variabel gewählt. Damit ist auch Anti-Alias möglich. Allerdings sollte dieser Modus nur verwendet werden, wenn er auch wirklich notwendig ist, da er langsamer als "1" ist.

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

Hiermit wird die gesamte Textur mit der zuvor mit `TexSetColor` ausgewählte Farbe gefüllt.

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

Mit diesem Befehl wird ein einzelner Pixel gezeichnet, und zwar in der mit `TexSetColor` ausgewählten Farbe. In diesem Beispiel der Pixel an der Stelle  $x = 10$  und  $y = 20$ .

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

Dementsprechend zeichnet dieser Befehl ein (ausgefülltes) Rechteck mit der mit `TexSetColor` ausgewählten Farbe. In diesem Beispiel befindet sich die linke Kante bei  $x = 10$ , die rechte bei  $x = 29$ , die obere bei  $y = 20$  und die untere bei  $y = 39$ . Achtung: Aus Gründen der Logik/Mathematik wird die jeweils letzte Koordinate ausgelassen. Auf diese Weise entspricht nämlich dann die Breite dieses Rechtecks genau  $30 - 10 = 20$  Pixel, andernfalls wäre die Breite 21 Pixel.

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

Mit dieser Funktion kann die Farbe eines bestimmten Pixels ausgelesen werden. Im Beispiel wird die Farbe am Pixel  $x = 10$  und  $y = 20$  in die Variable "col" gespeichert.

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

Diese Funktion muss nur und sollte daher auch nur im Initialisierungs-Teil verwendet werden. Er ermittelt den (internen) Index eines Fonts, der in den entsprechenden Script-Befehle diesen Font identifiziert. In diesem Beispiel wird der Font mit der ContentID 1000:210 ausgewählt.

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

Dieser Befehl schreibt einen Text an eine beliebige Stelle auf der Textur. In diesem Fall wird der Text "testABC" mit dem im vorherigen Beispiel ermitteltem Font 1000:210 an die Stelle 5/5 (linke obere Ecke) geschrieben, und zwar einfarbig mit der zuvor mit "TexSetColor" ausgewählten Farbe und mit 3 Pixeln "Sperrung", also mit zusätzlichen 3 Pixeln Abstand zwischen den Zeichen.

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

Mit dieser Funktion kann die Länge eines Strings in Pixeln ermittelt werden, ohne dass dieser in die Textur geschrieben wird. Eine entscheidende Rolle spielt sie daher beim Programmieren einer Vollmatrix-Anzeige, die je nach Textlänge zwischen verschiedenen Fonts wechseln soll. Im Beispiel wird die Länge des Strings "Texttext" im Font "fontID" berechnet und in die Variable "lenpixel" geschrieben.

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

Diese Funktion färbt abhängig von der Farbe der Pixel in einem bestimmten Bereich einer Textur einen bestimmten Bereich einer zweiten Textur mit einer Wunschfarbe ein, ggf. mit einem ganzzahligen Faktor. Die beiden Texturen werden mit sourceId und targetId festgelegt (vgl. TexSelTex), der zu kopierende Bereich erstreckt sich von (x1/y1) bis (x2-1/y2-1) (vgl. auch TexDrawRect). Die linke obere Ecke (also x1/y1) wird auf der Zieltextur an der Position (targetX/targetY) ausgerichtet. Der Skalierungsfaktor muss größer gleich 1 sein. sourceColor legt die Farbe fest, die die zu prüfenden Pixel der Quelltextur haben müssen, damit in die Zielextextur mit der Farbe targetColor gezeichnet wird. Hat die Quelltextur auf einem bestimmten Pixel eine von sourceColor abweichende Farbe, dann wird dieser Pixel nicht übertragen.