Font3D

Version 1.60

User's Guide

Table of Contents

Introduction

Overview

Font3D is a utility program for the generation of 3-dimensional text objects. It can write these object descriptions in any of the following file formats: POV-Ray 3.0, POV-Ray 2.x, Radiance 2.x, Vivid 2.0, AutoCad DXF, RenderMan RIB, and RAW Triangles. There is a ready-to-use executable already built for MS-DOS users, and the source code is also available for those of you lucky enough to have a UNIX workstation at your fingertips.

Font3D lets you do quite a few things with just a TrueType font, and a string of text. Here are some of the features you'll find in this release:

* Extrude a string of text to create a 3-D object that can be used in your favorite rendering program
* Use almost any TrueType font; both the Macintosh and Windows varieties are supported
* Bevel your 3-D text with either flattened, or rounded edges
* Use interpolated triangle patches to build a 'smooth' text object (only for some output formats)
* Write object descriptions using the new POV-Ray 3.0 'mesh' primitive
* Configuration files can store frequently used options, and allow easy experimenting with different effects
* Store length, width, and height information about an object in the output file
* Assign a symbolic name to a text object, for easier inclusion in a larger scene file
* Assign a separate texture identifier to each part of an object (faces, bevels, and sides)
* Control the visibility of each part of an object

Availability

I have decided with this release of *Font3D* to keep separate archives of the individual binaries, and of the source code. This will likely be welcome news to all the UNIX people out there who cringe at the thought of downloading a 300K, MS-DOS *.EXE file. Each archive has its own complete copy of the documentation--both ASCII text, and PostScript. Unfortunately, I no longer have access to an OS/2 machine and compiler, so unless someone offers to build a native version for that platform, the 32-bit DOS executable will have to do.

One of the drawbacks of keeping the source code in a separate archive is that it's more difficult to control distribution. If you're looking for a specific binary, or if you have a binary but want the source, the best place to begin your search is probably the Official POV-Ray FTP Site (ftp.povray.org). It is here that I will first upload any new versions of *Font3D*. Specifically, try the following two URL's:

ftp://ftp.povray.org/pub/povray/incoming/utilities
ftp://ftp.povray.org/pub/povray/utilities

The files will be named with the following convention:

`f3src160.zip` *Font3D* Version 1.60 Source Code & Documentation
`f3dos160.zip` *Font3D* Version 1.60 32-bit DOS Executable & Documentation

In general, filenames will be of the form f3xxxnnn.zip, where 'xxx' is a three character description of the archive's contents, and 'nnn' is the version number.

Copyright

Comments

If you have any questions, suggestions, comments, I would love to hear from you. If you think you have found a bug, be sure to let me know. Remember, the more detailed a description of the problem you give, the easier it will be for me to track it down (and, believe me, I need all the help I can get).

The preferred means of communication is E-Mail. Currently, I can be reached at either of the following electronic addresses:

squid@cis.ksu.edu
squid@ksu.ksu.edu

If neither of these work, then chances are that I have finally graduated, and that you have a somewhat outdated version of this program. If all else fails, you can try to get ahold of me the old fashioned way:

Todd A. Prater
2303 Candlewood Drive Apt. 9
Manhattan, Kansas 66503
(913) 537-7724

Acknowledgements

First of all, many thanks go out to those of you who have already become registered users of *Font3D*. I very much appreciate your support! Thanks also to everyone for all the great suggestions, and for some great detective work. The following people have either added to the functionality of this program, or have provided me with valuable information, or both:

Scott Adkins (sadkins@voyager.cns.ohiou.com)	Added the font-path, and output-path options
Dr. Thomas Hiller (hiller@tu-harburg.d400.de)	Added Radiance 2.x support
Larry Gritz (gritz@seas.gwu.edu)	Information on RenderMan and RIB files

Using the Program

DOS Installation

The only file you really need is f3dos160.zip. That archive contains both an executable, and the user's manual in both ASCII text, and PostScript formats. Please take a moment to look at the readme.txt file, which may contain late-breaking news (eg. new features, changes to the documentation, or bug reports). The following can be used as a rough guide of what you need to do to install *Font3D* (the exact steps will depend on which directory you choose to keep the program in):

1.) Copy the archive (F3DOS160.ZIP) to a separate directory (eg. C:\POV\UTIL\FONT3D).
2.) Decompress the archive using your favorite unzipper.
3.) Edit the default configuration file to reflect your preferences. In particular, change the font-path setting to point to the directory where you keep your TrueType fonts.
4.) Add the new *Font3D* directory to the PATH statement in your AUTOEXEC.BAT file.
5.) Add the following line (or something like it) to your AUTOEXEC.BAT file:

    SET FONT3D_DEFAULT_CONFIG=C:\POV\UTIL\FONT3D\FONT3D.DEF

    This will allow *Font3D* to find your default configuration file, regardless of which directory you run the program from.

Hardware requirements include at least an 80386 processor, a coprocessor, and two megabytes of RAM.

UNIX Installation

UNIX users will need to get the file f3src160.zip. This archive contains the complete source code for *Font3D*, a GNUish Makefile, and the user's manual in both ASCII text, and PostScript formats. Please take a moment to look at the readme.txt file, which may contain late-breaking news (e.g. new features, changes to the documentation, or bug reports). The following can be used as a rough guide of what you need to do to install *Font3D* on your system (the exact steps will depend on which directory you choose to keep the program in):

1.) Copy the archive (f3src160.zip) to a separate directory (e.g. ~/pov/util/font3d).
2.) Decompress the archive using your favorite unzipper.
3.) Type 'make' to build an executable. The makefile assumes you have the GNU C/C++ Tools installed on your system--if this is not the case, you may have to make a few changes.
4.) Add the new *Font3D* directory to your 'path' variable, or alternatively, you can copy the executable and default configuration file (font3d.def) to a directory already in your path.
5.) Edit the default configuration file to reflect your preferences. In particular, change the font-path setting to point to the directory where you keep your TrueType fonts.
6.) Add the following line (or something like it) to the end of your shell initialization file (.cshrc, .tcshrc, etc.):

    setenv FONT3D_DEFAULT_CONFIG ~/pov/util/font3d/font3d.def

    Note that this is not just the directory, but also the filename. This will allow *Font3D* to find your default configuration file, regardless of which directory you run the program from.

Program Options

Fortunately, although there are quite a few different program options, most of them fall into a few general categories. Please note that all program options must be given to *Font3D* in the form of

option-name=*value*

where **option-name** is the name of a command to use (like **font**, **depth**, or **face-cut**), and *value* is the parameter required by that command. All commands must have a value, and everything is case-insensitive (except string values).

The various *Font3D* commands can be usefully categorized as follows:

Fonts	**font**, **font-path**, **map**
Visibility	**faces**, **sides**, **bevels**, **front-face**, **back-face**, **front-bevel, back-bevel**
Texturing	**texture**, **face-texture**, **side-texture**, **bevel-texture, front-face-texture**, **back-face-texture**, **front-bevel-texture**, **back-bevel-texture**
Beveling	**bevel-type**, **cut**, **face-cut**, **side-cut**, **front-face-cut**, **front-side-cut**, **back-face-cut**, **back-side-cut**
Object	**char**, **code**, **depth, resolution, string**, **triangle-type,**
Output	**coordinate-system, constants, format**, **name, output, output-path, precision**
Positioning	**xpos**, **ypos**, **zpos**
Miscellaneous	**config, verbose**

Configuration Files

One of *Font3D*'s most useful features is the ability to store frequently used commands in a configuration file. Upon invocation, *Font3D* looks for a file called "font3d.def" in the current directory (unless you have set the FONT3D_DEFAULT_CONFIG environment variable to point to something else). Here you can put any command line options you find yourself repeatedly typing over and over (like **font-path**, **output-path**, etc...). Take a look at the font3d.def file that is included in the distribution to see what it looks like. The second way to have *Font3D* read options from a text file is the **config** option. You give it the name of a file, and it will be processed as if it were a list of options on the command line. For example if you create a text file named "test.def" with the following lines:

```
font-path=C:\WINDOWS\SYSTEM;C:\FONTS
output-path=C:\POV\INCLUDE
cut=0.003 depth=0.25
```

and then call the program like this:

```
C:>font3d font=arial.ttf config=test.def
```

it will create exactly the same output file as if you used a much longer command line:

```
C>font3d font-path=C:\WINDOWS\SYSTEM;C:\FONTS output-path=C:\POV\INCLUDE
cut=0.003 depth=0.25 font=arial.ttf
```

Anything enclosed in a pair of quotes (either single or double) is treated as a single word. This makes it possible to specify a string of text with embedded spaces (e.g. "This is a test"). If you need to use a double quote character in a string of text , just enclose the string in single quotes (e.g. '"Good Night!"'), and vice-versa. Here are a couple of examples of how you could assign a string of text to the **string** program option:

```
string="This is a test..."
string='This is a test...'
string="That's all folks"
string='That'"'"'s all folks'
```

You can also use comments in your configuration files. Anything following a '#' on a particular line will be ignored (unless it comes in the middle of a quoted string); the behavior is similar to that of C++'s '//'-style comment. If you need to use this pound-sign character, in a string of text for example, just put a pair of quotes around it.

Objects

The objects that *Font3D* creates are actually composed of (quite a few) tiny little triangular patches. These triangles can be thought of in groups: the front face, the front bevel, the sides, the back bevel, and the back face. Any of these individual components can be made invisible, so that for example only the triangles that make up the front face or the back face are generated--see **Visiblilty** in this section for more information. The text is always extruded along the z-axis, with its height in the y direction, and width in the x direction.

By default, a right-handed coordinate system is used, which may, or may not be what you're used to (by default POV-Ray uses a left-handed coordinate system). You can use the **coordinate-system** option to change this behavior.

Fonts

Both Windows and Macintosh TrueType fonts can be used by *Font3D*. By default, a Windows encoding table is assumed, so if you see an error message like:

```
ERROR: A required part of this font is missing.
```

or,

ERROR: Unable to read font file.

try to use the **map=MAC** option to force *Font3D* to look for a Macintosh encoding table.

New with version 1.50 is kerning support. This is entirely transparent to the user; if kerning information is present in a particular font, then it will be used, otherwise you won't miss it. Most public-domain fonts don't have a kern pair table, but there are a few that do. On the other hand, many commercial fonts do have this information, and it can make a big improvement in some situations.

Defaults

The only program option that doesn't have a default value is **font**. You always need to specify, either on the command line, or in a configuration file, which TrueType font to use. If you type the following at a command line prompt (and if the font arial.ttf is in the current directory):

```
C:>font3d font=arial.ttf
```

Font3D will generate the word 'Font3D', without bevels, and then write a **POV** formatted output file named "font3d.inc" in the current directory. The object will be #declared in that output file as FONT3D_OUTPUT. All you need to do to place this object in a **POV** scene is first #include it, and then reference it with a statement like:

```
object
{
    FONT3D_OBJECT
    texture { YOUR_FAVORITE_TEXTURE }
}
```

In the rendered scene, an extruded string of text will be centered at the origin. See the **Command Reference** later in this guide for a list of all commands and their default values.

Smooth Triangles

Many rendering programs recognize a special kind of triangular patch, a 'smooth triangle', that can have different surface normals at each vertex. The renderer then averages those individual normals to calculate a normal at any point inside the triangular patch. If the three vertex normals are calculated correctly, this can eliminate the faceted look that many objects built from simple flat triangles have. Believe me, this can make a huge difference in the quality of an object, which is why smooth triangles are the default. Keep in mind that neither the **RAW** or the **DXF** file formats support this type of primitive, so they simply ignore it. If you want to turn this feature off, to speed up rendering for example, simply use the **triangle-type** command:

```
C:>font3d font=arial.ttf string=TestString triangle-type=FLAT
```

Rounded edges are physically identical to a flat bevel. The one difference is that the vertex normals are altered so that the bevels *appear* to be rounded. This is why the **RAW** and **DXF** file formats are incapable of the effect, and that even if **triangle-type** is set to **SMOOTH**, any bevels will still appear to be flat.

Output Files

The default output file is "font3d.inc", and it is placed in the current working directory. There are two settings that allow you to change this behavior: **output**, and **output-path**. **output** specifies the name of the file to which the object information will be written. **output-path** allows you to specify the directory of the output file. This option is especially useful in a configuration file (see below).

Here is an example of how you can use both of these options at the same time:

```
C:>font3d font=times.ttf output-path=C:\POV\INCLUDE output=text.inc
```

And, here is an example of how you can use just the **output** option to do the same thing:

```
C:>font3d font=times.ttf output=C:\POV\INCLUDE\text.inc
```

Once again, the reason the **output-path** option was provided, was so that you could place an output path into one of your configuration files, and forget about it.

POV-Ray-compatible output files have a set of useful constants (XMin, XMax, Height, etc.) declared at the very beginning. When designing especially complex scenes, some users may find themselves running up against the 1,000 #declared constant limit imposed by POV. The **constants** option causes these #declared values in the output file to be commented out.

Also included in the (commented-out) header section of the output file is a list of all program options and their settings, when a particular object was generated.

Object Naming

POV users can assign a name to the object that *Font3D* generates. This is a very useful feature if you're using more than one object in a particular scene. Here's an example. Let's say you want to have four different words in your scene; call the program four times like this:

```
C:>font3d font=arial.ttf output-file=this.inc string=This name=THIS_STR
C:>font3d font=times.ttf output-file=is.inc string=is name=IS_STR
C:>font3d font=times.ttf output-file=a.inc string=a name=A_STR
C:>font3d font=arial.ttf output-file=test.inc string=Test name=TEST_STR
```

A POV scene description that uses these objects might look like this:

```
#include "colors.inc"
#include "textures.inc"

#include "this.inc"
#include "is.inc"
#include "a.inc"
#include "test.inc"
```

```
// Declare a light and a camera here...

object {
    THIS_STR
    texture { Jade }
    translate <0,2,0>
}
object {
    IS_STR
    texture { White_Marble }
 translate <0,1,0>
}
object {
    A_STR
    texture { Red_Marble }
}
object {
    TEST_STR
    texture { Blood_Marble }
    translate <0,-1,0>
}
```

Output Formats

Currently, seven different output formats are supported:

- POV-Ray 3.0
- POV-Ray 2.x
- Radiance 2.x
- Vivid 2.0
- AutoCad DXF
- AutoCad DXF (single entity)
- RenderMan RIB
- RAW Triangles

Each format has it's own set of capabilities, so remember that some *Font3D* features may not be available with all output file types.

POV-Ray 3.0. *Font3D* can now generate objects using POV-Ray 3.0's new 'mesh' primitive. This primitive allows for much more efficient memory usage. In every other way--including usage--the output is identical to that of the older 2.x style file. All program options are supported with this file format. Use the program option **format=POV3**.

POV-Ray 2.x. This format also supports all program options. *Font3D* generates an object #declared as a union of either smooth or flat triangles. If you use any of the **texture-*** options, be sure to define these before you #include the output file in your scene. Here is an example of generating a POV-Ray 2.x output file, and then using it as an include file in a POV scene. Type the following at a command line:

```
>font3d font=arial.ttf string="Font3D Rules" texture=F3D_TEXTURE
```

Then, create a POV-Ray 2.x scene file like the following:

```
// Begin POV-Ray 2.x Scene File

#include "colors.inc"
#include "textures.inc"

#define F3D_TEXTURE =
texture
{
    DMF_WOOD
}

camera
{
    location  <6,10, 5 >
    right <-1,0,0>
    direction <0, 0,4>
    sky <0,0,1>
    look_at   <0,0,0>
}

light_source { <5,2,70> color Gray70 }
light_source { <-18,-0.6,60> color Gray70 }
light_source { <16,0,60> color Gray80 }
light_source { <0, 2, 40> color Gray70}

#include "font3d.inc"

object { FONT3D_OBJECT }

// End POV-Ray 2.x Scene File
```

That's all there is to it! Just render this scene file, and enjoy the beautiful, rendered, 3-D text! This is the default output file type, so you don't need to specify anything with the **format** option (unless you've specified another type in your configuration file).

Radiance 2.x. This format supports just about everything the POV-Ray formats do, with the execption of object naming. Objects are output as a series of polygons. Use the program option **format=RAD**.

Vivid 2.0. The Vivid format, currently, does not support either object naming, or texturing. Objects are output as a series of 'patch' primitives. Use the program option **format=VIVID**.

AutoCad DXF. *Font3D* uses 3DFACE primitives to describe an object with the DXF format. Each component of an object is output as a separate entity. Naming, texturing, and smooth traingles (and hence, rounded edges), are not available with this format. Use the program option **format=DXF.**

AutoCad DXF (single entity). Some programs, including Imagine, only look at the first entity in a DXF file. If you use the program option **format=DXF2**, *Font3D* will generate a series of 3DFACEs, specified as a single entity. As with the **DXF** type above, naming, texturing, and smooth triangles (and hence, rounded edges), are not available with this format.

RenderMan RIB. Using the program option **format=RIB** causes *Font3D* to generate a series of RIB polygons. Texturing, and object naming are not supported with this output file type.

RAW Triangles. Using the program option **format=RAW** causes *Font3D* to generate an output file containing raw triangle vertex date in the RAW format. This is a common denominator for many different programs, but the format is very limited. Only flat triangles are available.

Textures

Font3D allows a texture identifier to be assigned to each individual component of a text object (faces, bevels, sides). Currently, only the **POV** and **POV3** file formats uses this information when writing an output file. An object can have up to five different parts: a front face, a front bevel, a side, a back bevel, and a back face. There are eight different program settings that can be used to give a texture to one of these object parts: **front-face-texture**, **back-face-texture**, **side-texture**, **front-bevel-texture**, **back-bevel-texture**, **face-texture**, **bevel-texture**, and **texture**. The first five of these only set the texture of one individual component, while the last three assign the same texture to a group of components.

Here's an example. Let's say you've already declared two textures in a POV 2.x scene description file:

```
// BEGIN POV SCENE DESCRIPTION FILE

#include "colors.inc"
#include "textures.inc"

#declare FaceTexture =
texture
{
   pigment { color IndianRed }
}

#declare SideTexture =
texture
{
   DMF_Wood1
   rotate <3,5,13>
}
```

To make *Font3D* assign the first texture to both faces of an object, and the second texture to its sides, invoke the program like this:

```
C:>font3d font=arial.ttf face-texture=FaceTexture side-texture=SideTexture string=Font3D
```

The **face-texture** option assigns 'FaceTexture' to both the object's front and the back faces at the same time, while **side-texture** assigns 'SideTexture' to the its sides. By default, the object *Font3D* generates is given the name "FONT3D_OBJECT", so the rest of our scene description file would look something like this:

```
#include "font3d.inc"

object { FONT3D_OBJECT }
```

// END OF POV SCENE DESCRIPTION FILE

Note that the output file is included *after* #declaring the textures. It is also important to remember that you have to do either none of your texturing with *Font3D* (ie. do it in your scene description file), or all of your texturing with *Font3D*. Any texture applied to an object in a scene description file will override the textures assigned to it by this program.

Bevels

The way in which *Font3D* bevels the edges of an object is not unlike how a router might cut into a piece of wood. There are two parameters that define the size and appearance of a bevel: **face-cut**, and **side-cut**. **face-cut** specifies the size of the cut that a bevel will make into the face of an object. Likewise, **side-cut** specifies the size of cut that will be made into the side of an object (see Figure 2.1). These two settings are actually generalization of the following, more precise versions: **front-face-cut**, **front-side-cut**, **back-face-cut**, and **back-side-cut**. To cause an object's edges to be beveled, all you need to do is specify a cut value. By default, all the cut settings are zero, ie. beveling is turned off.

```
                 |<----------- Depth ------------->|
                 |                                 |
                 |  Front  |                |  Back  |
                 |  Side   |     Object     |  Side  |
                 |<-Cut -> |      Side      |<- Cut->|
                 |         |       |        |        |
       _______             ________v_________          ______
        Front          /                       \       Back
         Face        /                           \     Face
          Cut      /                               \   Cut
       _______   /                                   \ ______
                 |             Object                |
                 |          Cross-Section            |
                 |                                   |
    Object       |                                   |   Object
    Front        |                                   |     Back
    Face --->    |                                   |<--- Face
                 |                                   |
                 |                                   |
```

Figure 2.1: Cross section of a beveled object generated by *Font3D*.

If you've played around with the beveling features at all, you probably already know that making too large a cut into the face of an object just won't work. In fact, the beveling and rounding alogorithm that *Font3D* uses is only designed to create subtle effects, not large, easy-to-see-from-a-distance bevels. With proper lighting though, a small bevel (or rounded edge) on an object can make a scene much more realistic. For many fonts, usable **face-cut** values will be less than about 0.006, but you may be surprised at how much of an effect this can have on an object's perceived realism.

Most times, you will probably want to keep things simple. Here is an example of how you might use the **cut** option to bevel a text object:

C:>font3d font=arial.ttf string=TestString cut=0.003

Rounded Edges

Rounded edges are very similar in most respects to beveled edges. You use the **bevel-type=ROUND** switch, along with a non-zero **cut**, to generate them. *Font3D* only approximates how a rounded edge might look by creating a flat bevel, and then manipulating the vertex normals of those triangles. For this reason (and for the reasons mentioned above) you will want to keep your **face-cut** values relatively small. Here's an example of how to make *Font3D* round the edges of an object:

C:>font3d font=arial.ttf string=TestString cut=0.003 bevel-type=ROUND

Visibility

The visibility of each individual part of an object can be turned on or off by using one or more of the following switches: **front-face**, **front-bevel**, **sides**, **back-bevel**, **back-face**, **faces**, **bevels**. The first five of these control the visibility of one distinct part, while the last two control groups of components. These options are useful if for any particular scene only part of an object will ever be visible. For example, if you were using *Font3D* to generate raised lettering on a planar surface, you could make invisible the sides, the back bevel, and the back face, so that only the triangles making up the front face, and the front bevel are written to the output file. A sample invocation of the program could look like this:

C:>font3d font=arial.ttf sides=OFF back-bevel=OFF back-face=OFF

Remember that the visibility settings only control whether triangles are written to a file, not whether they are actually a part of the object. If you have specified a **face-cut** value for bevels, then simply using **bevels=OFF** will not keep the object from being beveled. It *will* however, cause there to be a small gap between the object's faces and the sides, where the bevels *would* have been. By default, all of the components of an object will be generated.

Command Reference

This section contains a complete reference of all *Font3D* options. Each one of the following commands can either be used on the command line, or in a configuration file.

back-bevel

Syntax: back-bevel=**ON|OFF**

Description: This option determines the visibility of an object's back bevel (or rounded edge). If it is set to **OFF**, then triangles that make up the object's back bevel will not be generated. Note that this only keeps the back bevel triangles from being written to an output file; if either *back-face-cut* or *back-side-cut* are nonzero, then there will be a small gap between the back face and sides of the generated object (where the bevel would have been).

Default: **ON**

Example: C:>font3d font=arial.ttf back-bevel=OFF

See Also: **bevels, cut**

back-bevel-texture

Syntax: back-bevel-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV**, **POV3** or **RAD** (with the *format* switch). It specifies the name of a texture that will be associated with the back bevel of an object. See **Using the Program** for more information on assigning textures to different object components.

Default: None.

Example: C:>font3d font=arial.ttf back-bevel-texture=BBTexture

See Also: **back-face-texture, bevel-texture, face-texture, front-bevel-texture, front-face-texture, side-texture, texture**

back-face

Syntax: back-face=**ON|OFF**

Description: This option determines the visibility of an object's back face. If the switch is set to **OFF**, then triangles that make up the back face will not be written to the output file.

Default: **ON**

Example: C:>font3d font=arial.ttf back-face=OFF

See Also: **faces, front-face**

back-face-cut

Syntax: back-face-cut=*float*

Description: This option specifies the size of cut that will be made into the back face of an object by it's back bevel (or rounded edge). The default is zero, which causes the back bevel not to cut into the back face of the object at all. If both *back-side-cut* and *back-face-cut* are zero, then *Font3D* will not bevel (or round) the back edges of an object. Large cut sizes can cause some problems, so keep this number relatively small.

Default: 0.0

Example: C:>font3d arial.ttf back-face-cut=0.005

See Also: **back-side-cut, cut, face-cut, front-face-cut, front-side-cut, side-cut**

back-face-texture

Syntax: back-face-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV, POV3** or **RAD** (with the *format* switch). It specifies the name of a texture that will be associated with the back face of an object. See **Textures** in **Using The Program** for more information on assigning textures to different object components.

Default: None.

Example: C:>font3d font=arial.ttf back-face-texture=BFTexture

See Also: **back-face-texture, bevel-texture, face-texture, front-bevel-texture, front-face-texture, texture**

back-side-cut

Syntax: back-side-cut=*float*

Description: This option specifies the size of cut that will be made into the side of an object by it's back bevel (or rounded edge). The default is zero, which causes the back bevel not to cut into the side of an object at all. If both *back-side-cut* and *back-face-cut* are zero, then *Font3D* will not bevel (or round) the back edges of an object.

Default: 0.0

Example: C:>font3d font=arial.ttf back-side-cut=0.005

See Also: **back-face-cut, cut, face-cut, front-face-cut, front-side-cut, side-cut**

bevel-texture

Syntax: bevel-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV**, **POV3** or **RAD** (with the format switch). It specifies the name of a texture that will be associated with both the front and back bevels of an object. See **Textures** in **Using The Program** for more information on assigning textures to different object components. This is the same as setting both *front-bevel-texture* and *back-bevel-texture* to the same value.

Default: None.

Example: C:>font3d font=arial.ttf bevel-texture=BEVEL_TEXTURE

See Also: **back-bevel-texture, back-face-texture, face-texture, front-bevel-texture, front-face-texture, side-texture, texture**

bevel-type

Syntax: bevel-type=**ROUND|FLAT**

Description: This option specifies the kind of beveled edges to generate. **FLAT** bevels can be used with any triangle type, and with any output file format. **ROUND** bevels are only *Font3D's* approximation of what a real rounded edge might look like. They are generated by altering the triangle vertex normals, and thus depend on a particular renderer's ability to recognize 'smoothed' triangles as a primitive object. The rounded effect will only be apparent if *triangle-type* is set to **SMOOTH**.

Default: **FLAT**

Example: C:>font3d font=arial.ttf bevel-type=ROUND

See Also: **triangle-type**

bevels

Syntax: bevels=**ON|OFF**

Description: This option determines the visibility of an object's front and back bevels (or rounded edges). Note that this only keeps the triangles from being written to the output file. If either *front-face-cut* or *front-side-cut* are nonzero, then there will be a small gap between the front face and sides of a generated object. Likewise, if either *back-face-cut,* or *back-side-cut* are nonzero, then there will be a small gap between the back face and sides of the generated object where the bevel would have been. Using this option is the same as using both *front-bevel* and *back-bevel* with the same value

Default: **ON**

Example: C:>font3d font=arial.ttf bevels=OFF

See Also: **back-bevel, front-bevel**

char

Syntax: char=*character*

Description: This option specifies a single character to generate.

Default: None. (A string is generated as a default)

Example: C:>font3d font=arial.ttf char=T

See Also: **code, string**

code

Syntax: code=*integer*

Description: This option specifies the character code of a single glyph to generate.

Default: None. (A string is generated by default)

Example: C:>font3d font=arial.ttf code=81

See Also: **char, string**

config

Syntax: config=*filename*

Description: This switch specifies a text file that will be processed as if it were a list of command line options. The options must look exactly as they would on the command line, except that they may be separated by linefeeds. Configuration files can also have comments: anything following a '#' character on any given line will be ignored.

Default: None.

Example: C:>font3d font=arial.ttf config=bevelcnf.def

constants

Syntax: constants=**ON|OFF**

Description: This option allows the #declared constants at the beginning of a **POV** or **POV3** formatted output file to be commented out. All other output formats ignore this setting.

Default: **ON**

Example: C:>font3d font=arial.ttf constants=OFF

coordinate-system

Syntax: coordinate-system=**RIGHT|LEFT**

Description: This option allows you to specify which coordinate-system is used when generating the object. In a right handed coordinate-eyetem, the text objects are extruded along the z-axis, with the front of the object being more positive than the back. With a left handed coordinate-system essentially the opposite situation holds: the text object is still extruded along the z-axis, but the front is more negative than the back.

Default: **RIGHT**

Example: C:>font3d font=arial.ttf coordinate-system=LEFT

cut

Syntax: cut=*float*

Description: This option specifies the amount that a bevel (or rounded edge) will cut into an object. The default is 0.0, which causes *Font3D* not to bevel (or round) the edges of an object. Using this option is the same as using both *face-cut* and *side-cut* with the same value.

Default: 0.0

Example: C:>font3d font=arial.ttf cut=0.005

See Also: **back-face-cut, back-side-cut, face-cut, front-face-cut, front-side-cut, side-cut**

depth

Syntax: depth=*float*

Description: This switch controls an object's thickness.

Default: 0.2

Example: C:>font3d font=arial.ttf depth=1.275

face-cut

Syntax: face-cut=*float*

Description: This option specifies the amount that a bevel (or rounded edge) will cut into one of the faces of an object. Using this option is the same as using *front-face-cut* and *front-side-cut* with the same value.

Default: 0.0

Example: C:>font3d font=arial.ttf face-cut=0.005

See Also: **back-face-cut, back-side-cut, cut, front-face-cut, front-side-cut, side-cut**

face-texture

Syntax: face-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV**, **POV3,** or **RAD** (with the *format* switch). It specifies the name of a texture that will be associated with both the front and back faces of an object. See **Textures** in **Using The Program** for more information on assigning textures to different object components. Using this option is the same as using both *front-face-texture* and *back-face-texture* with the same values.

Default: None.

Example: C:>font3d font=arial.ttf face-texture=FACE_TEXTURE

See Also: **back-bevel-texture, back-face-texture, bevel-texture, front-bevel-texture, front-face-texture, texture**

faces

Syntax: faces=**ON|OFF**

Description: This option determines the visibility of an object's front and back faces. If the switch is set to **OFF**, then triangles that make up the front and back faces of the object will not be written to the output file. Using this option is the same as using both *front-face* and *back-face* with the same value.

Default: **ON**

Example: C:>font3d font=arial.ttf faces=OFF

See Also: **back-face, back-bevel, bevels, front-bevel, front-face, sides**

font

Syntax: font=*filename*

Description: This option determines the typeface in which the object will be generated. *filename* must be a Windows or Macintosh TrueType font file (usually of the form *.TTF). If the font is not in the current working directory, then you should specify both the path and filename here. This is the only option that is required but not associated with a default value; you must either provide it on the command line, or in a configuration file.

Default: None.

Example: C:>font3d font=arial.ttf

See Also: **map**

font-path

Syntax: font-path=*pathstring*

Description: This option takes a semicolon or colon separated list of directories that make up the path of where to locate the truetype file. If the font is not found in any of those directories, then the current directory is used before finally giving up.

Default: None.

Examples: C:>font3d font-path=C:\fonts;C:\MISC\FONTS font=arial.ttf
C:>font3d font-path=/usr/local/fonts:/home/fonts font=arial.ttf

See Also: **output-path**

format

Syntax: format=**POV|POV3|RAW|RIB|RAD|DXF|DXF2|VIVID**

Description: This switch specifies which format the output file will be written in. The following options are available:

POV Causes the output file to be written in POV's (version 2.x) scene description language. All of *Font3D*'s texturing options are available with this file format, as are all triangle and bevel types.

POV3 Causes the output file to be written in POV's (version 3.0) scene description language. This output file type is exactly the same as the POV type, except that the object is written as one of POV-Ray 3.0's new 'mesh' primitives.

RAD Causes the output file to be Radiance 2.x compatible.

RAW Causes the output file to be written as raw triangle data. Texturing options are ignored with this file format, as are requests for **SMOOTH** triangles and **ROUND**ed edges.

RIB Causes the output file to be written as a RenderMan Interface ByteStream file.

DXF Causes the output file to be written as an AutoCad Drawing Interchange File.

DXF2 Causes the output file to be written as an AutoCad Drawing Interchange File. This output format type writes the output as a *single* DXF entity. Some programs won't work correctly, otherwise.

VIVID Causes the output file to be written in Vivid 2.0 scene description language, as a list of triangle patches.

See **Output Formats** in **Using the Program** for more information on the specifics of each file format.

Default: **POV**

Example: C:>font3d font=arial.ttf format=RIB

See Also: **bevel-type, triangle-type**

front-bevel

Syntax: front-bevel=**ON|OFF**

Description: This option determines the visibility of an object's front bevel (or rounded edge). If it is set to **OFF**, then triangles that make up the object's front bevel will not be generated. Note that this only keeps the front bevel triangles from being written to an output file; if either *front-face-cut* or *front-side-cut* are nonzero, then there will be a small gap between the front face and sides of the generated object (where the bevel would have been).

Default: **ON**

Example: C:>font3d font=arial.ttf front-bevel=OFF

See Also: **bevels, back-bevel, back-face, faces, front-face, sides**

front-bevel-texture

Syntax: front-bevel-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV, POV3,** or **RAD** (with the *format* switch). It specifies the name of a texture that will be ssociated with the back bevel of an object. See **Using The Program** for more information on assigning textures to different object components.

Default: None.

Example: C:>font3d font=arial.ttf front-bevel-texture=FBEVEL_TEXTURE

See Also: **back-bevel-texture, back-face-texture, bevel-texture, face-texture, front-face-texture, side-texture, texture**

front-face

Syntax: front-face=**ON|OFF**

Description: This option determines the visibility of an object's front face. If the switch is set to **OFF**, then triangles that make up the front face will not be written to the output file.

Default: **ON**

Example: C:>font3d font=arial.ttf front-face=OFF

See Also: **back-face, faces**

front-face-cut

Syntax: front-face-cut=*float*

Description: This option specifies the size of cut that will be made into the front face of an object by it's front bevel (or rounded edge). The default is zero, which causes the front bevel not to cut into the front face of the object at all. If both *front-side-cut* and *front-face-cut* are zero, then *Font3D* will not bevel (or round) the front edges of an object. Large cut sizes can cause some problems, so keep this number relatively small.

Default: 0.0

Example: C:>font3d font=arial.ttf front-face-cut=0.003

See Also: **cut, back-face-cut, back-side-cut, face-cut, front-side-cut, side-cut**

front-face-texture

Syntax: front-face-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV**, **POV3**, or **RAD** (with the *format* switch). It specifies the name of a texture that will be associated with the front face of an object. See **Textures** in **Using The Program** for more information on assigning textures to different object components.

Default: None.

Example: C:>font3d font=arial.ttf front-face-texture=FFACE_TEXTURE

See Also: **back-bevel-texture, back-face-texture, bevel-texture, face-texture, front-bevel-texture, side-texture, texture**

front-side-cut

Syntax: front-side-cut=*float*

Description: This option specifies the size of cut that will be made into the side of an object by it's front bevel (or rounded edge). The default is zero, which causes the front bevel not to cut into the side of an object at all. If both *front-side-cut* and *front-face-cut* are zero, then *Font3D* will not bevel (or round) the front edges of an object.

Default: 0.0

Example: C:>font3d font=arial.ttf front-side-cut=0.007

See Also: **back-face-cut, back-side-cut, cut, face-cut, front-face-cut, side-cut**

map

Syntax: map=**MAC|MS**

Description: This switch specifies which character encoding method is used in a particular font. **MAC** instructs *Font3D* to look for a Macintosh type character encoding table, while **MS** requests that it look for a Microsoft Windows type character encoding table.

Default: **MS**

Example: C:>font3d font=arial.ttf map=MAC

See Also: **font**

name

Syntax: name=*string*

Description: If the output format supports it, *Font3D* can assign an identifier to the object. The way in which this is done will vary depending on which output format is selected. See **Output Formats** in **Using the Program** for more information.

Default: FONT3D_OBJECT

Example: C:>font3d font=arial.ttf string=Test name=TEST_STRING

output

Syntax: output=*filename*

Description: This switch specifies the name of the file to which the object information will be written.

Default: font3d.inc

Example: C:>font3d font=arial.ttf char=T output=t.inc

See Also: **format**

output-path

Syntax: output-path=*pathname*

Description: This option takes a single directory which will be prepended to the output filename. A trailing slash is not needed.

Default: font3d.inc

Example: C:>font3d output-path=C:\tmp font=arial.ttf char=T output=t.inc

See Also: **font-path**

precision

Syntax: precision=*integer*

Description: This switch specifies the floating point precision of the output file.

Default: 6

Example: C:>font3d font=arial.ttf precision=4

resolution

Syntax: resolution=*integer*

Description: This switch specifies the number of line segments that are used to approximate a curved section of a font outline (it is this outline that will be eventually extruded).

Default: 5

Example: C:>font3d font=arial.ttf char=8 resolution=10

side-cut

Syntax: side-cut=*float*

Description: This option specifies the amount that a bevel (or rounded edge) will cut into the sides of an object. This is the same as giving *front-side-cut* and *back-side-cut* the same value.

Default: 0.0

Example: C:>font3d font=arial.ttf side-cut=0.003

See Also: **back-face-cut, back-side-cut, cut, face-cut, front-face-cut, front-side-cut**

side-texture

Syntax: side-texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV, POV3**, or **RAD** (with the *format* switch). It specifies the name of a texture that will be associated with sides of an object. See **Textures** in **Using The Program** for more information on assigning textures to different object components.

Default: None.

Example: C:>font3d font=arial.ttf side-texture=SIDE_TEXTURE

See Also: **back-bevel-texture, back-face-texture, bevel-texture, face-texture, front-bevel-texture, front-face-texture, texture**

sides

Syntax: sides=**ON|OFF**

Description: This option determines the visibility of an object's sides. If the switch is set to **OFF**, then triangles that make up the sides of the object will not be written to the output file.

Default: **ON**

Example: C:>font3d font=arial.ttf sides=OFF

See Also: **back-bevel, back-face, bevels, faces, front-bevel, front-face**

string

Syntax: string=*string*

Description: This option specifies a string of text to extrude into an object. *string* overrides both *char* and *code* regardless of the order in which they are given.

Default: Font3D

Example: C:>font3d font=arial.ttf string=TestString

See Also: **char, code**

texture

Syntax: texture=*string*

Description: Currently, this option is only useful if the output file format type is set to **POV**, **POV3**, or **RAD** (with the *format* switch). It specifies the name of a texture that will be associated with the object. See **Textures** in **Using The Program** for more information on assigning textures to different object components. Using this option is the same as using *face-texture*, *side-texture*, and *bevel-texture* with the same values.

Default: None.

Example: C:>font3d font=arial.ttf texture=Blue_Agate

See Also: **back-bevel-texture, back-face-texture, bevel-texture, face-texture, front-bevel-texture, front-face-texture, side-texture**

triangle-type

Syntax: triangle-type=**FLAT|SMOOTH**

Description: This switch tells *Font3D* which type of triangle primitive to use when building an object. **FLAT** triangles are compatible with all output file formats, and usually require less memory by a renderer, but the **ROUND** bevel-type is not supported. **SMOOTH** triangles, on the other hand, are not available with some output file formats, but in many cases are capable of producing a much better-looking object. Rounded edges are only generated if the triangle-type is **SMOOTH**.

Default: **SMOOTH**

Example: C:>font3d font=arial.ttf triangle-type=SMOOTH

See Also: **bevel-type, format**

verbose

Syntax: verbose=**ON|OFF**

Description: Setting this option to OFF causes all informational messages (except errors) to be suppressed. This can be useful when using the program in a batch file, or a shell script.

Default: **ON**

Example: C:>font3d font=arial.ttf verbose=OFF

xpos

Syntax: xpos=**LEFT|CENTER|RIGHT**

Description: This switch controls an object's positioning along the *x*-axis. The following options are available:

LEFT	Places the object so that it's rightmost point is flush with the $x=0$ plane.
CENTER	Centers the object with respect to the $x=0$ plane.
RIGHT	Places the object so that it's leftmost point is flush with the $x=0$ plane.

Default: **CENTER**

Example: C:>font3d font=arial.ttf xpos=LEFT

See Also: **ypos, zpos**

ypos

Syntax: ypos=**TOP|CENTER|BOTTOM|BASELINE**

Description: This switch controls an object's positioning along the *y*-axis. The following options are available:

TOP	Places the object so that it's topmost point is flush with the *y=0* plane.
CENTER	Centers the object with respect to the *y=0* plane.
BOTTOM	Places the object so that it's bottommost point is flush with the *y=0* plane.
BASELINE	Places the object so that its baseline is flush with the *y=0* plane.

Default: **CENTER**

Example: C:>font3d font=arial.ttf ypos=BASELINE

See Also: **xpos, zpos**

zpos

Syntax: zpos=**FRONT|CENTER|BACK**

Description: This option controls an object's positioning along the x-axis.

FRONT	Places the object so that it's front face is flush with the *z=0* plane.
CENTER	Centers the object with respect to the *z=0* plane.
BACK	Places the object so that it's back face is flush with the *z=0* plane.

Default: **CENTER**

Example: C:>font3d font=arial.ttf zpos=FRONT

See Also: **xpos, ypos**