Contents

1 Introduction

1.1 What's it?

"selftree.inc" is an include-file-generator that is written in the POVRay-scripting-language. The generated includes contain one or more objects, which may looks like trees, if the parameters are set neatly.

Shape, looking and complexity of the objects ineracting with a big bunch of parameters.

It is not needed to read following guide completely. For usage only it's enough to read the sections about usage and the one about the variables, leafs and textures. The rest is idle talk.

Ah, and sorry for my poor english...

1.2 Why?

The idea to write a tree-include is from a student of mathematics (hello, Holger!). But his includes contained no recursion. If you want ten nesting, you have to take ten includes... After long discussions, how to solve this problem he gave me the definitively proof, that there is no way to make recursive includes in POVRay. Nevertheless I decided to find a way. "selftree.inc" is the way.

2 Usage

2.1 General

To use the object generated by "selftree.inc", we need a little trick. The include writes the source of the generated object to the #debug-stream. To use the objects in your scenen-file, the #debug-stream must be written to another file. This can be done with the POVRay commandline-parameter -GD_FILENAME_ (or -GD_FILENAME_, if you want to see the output from "selftree.inc". You don't need to see it, it's not very interesting. If you want some infos, some statistics are written to the #statitics-stream after parsing the objects.). This commandline-parameter sends the #debug-stream to the file _FILENAME_. This file must included **after** "selftree.inc" in your scene-file. Before all this you should set a bunch of variables to influence the shape and the complexity of the trees (although the defaults generate usable trees).

So the following steps are needed to generate a tree:

- Set a big bunch of variables (or be satisfied with the default-tree).
- Include "selftree.inc" in your scene-file.
- Eventually repeat the first two steps to generate multiple trees.
- Include _FILENAME_ in your scene-file, where _FILENAME_ is the name of the by -GD_FILENAME_ generated file.
- Insert an object-statement with thegenerated tree-object:
  object { TREE texture {...} rotate {...} translate {...}}
- Run POVRay like this:
  povray +Imyscene.pov -GD_FILENAME_... more POVRay-arguments

2.2 Variables

"selftree.inc" can be modified by a big bunch of variables. There are three types of variables:

- Simple numbers: There are only two of them: DEPTH and NR-LEAFINCS.
- Strings: They must be set in parentheses and should contain a simple string, preferably without spaces or special characters. Example:
  #declare OBJECTNAME = "mytree"

- Functions: Most of the variables are functions, which are calculated in dependance of the current recursion-step. Functions can have constant values (**#declare NRBRANCHES = "3"**) oder they can be defined as functions in dependance of X (capital X): (**#declare NRBRANCHES = "int(sqrt(X)+(X/2))"**). The advantage of this is the possibility to influence the tree more accurately. For example with a simple function the tree can branch out more at the top than at the bottom of the recursion.

Aside from DEPTH, NRLEAFINCS and the texture-modifiers all variables should be defined in parentheses.

"selftree.inc" does not check the variables. It is presumed (this is the major bug in this include...) that nobody ever makes a mistake. However, if someone makes a mistake, it is presumed, that he he will notice...

The values of many of the variables are inluenced by random-functions when the trees are generated (Setting only the seed-value of a random-generator to another value can change the tree enormous). A little luck and experience help to find good values.

The following is a description of all variables, their defaults and some hints for good values (for a short reference look at the include-file itself).

2.2.1 OBJECTNAME

1. Description: To distinguish between different trees, they must have names. With this variable you set the name of the next tree.

2. Default: "TREE" (String)

3. Note: If you generate multiple trees without setting OBJECTNAME every time to a new name, the previous trees are overwritten.

2.2.2 USESTDTRUNK

1. Description: The parts of the trees are made of cones and spheres. To make the trees more realistic, you can define your own shape for the trunk. This variable is a bool, that says, if the default-trunk should be used or not.

2. Default: "YES"

2.2.3 TRUNKLEN

1. Description: How long should the trunk be? This value is for both the default-trunk and the user-defined one.

2. Default: “5”

2.2.4 TRUNKBASER

1. Description: What radius should the trunk have at his base?

2. Default: “0.7”

2.2.5 TRUNKCAPR

1. Description: What radius should the trunk have at his top?

2. Default: “0.5”

2.2.6 DEPTH

1. Description: This is the recursion-depth for the trees, which means how often the tree branches from the trunk to the branchtops.

2. Default: 6

3. Note: This one must not set in parentheses.

2.2.7 BLEN

1. Description: How long the branches should be at the first recursion-step (the branches the trunk).

2. Default: “5”

3. Note: We only have to define this length, because the branches in the deeper recursion-steps are scaled and a little bit varied versions of these.

2.2.8 LDIFF

1. Description: The range for random variations of the branch-legth.

2. Default: “0”

3. Note: This value is added to BLEN and is scaled in deeper recursion-steps also.

2.2.9 RADIUS

1. Description: The radius of the branches at the first recursion-step.
2. Default: "0.5"
3. Note: To have a smooth transit you should set this to the same value as TRUNKCAPR.

2.2.10 RADSCALE

1. Description: The scale for the branch-radius.
2. Default: "0.5"

2.2.11 RDIFF

1. Description: The range for random variations of RADIUS.
2. Default: "0"
3. Note: Would be subtracted from RADIUS and is scaled in deeper recursion-steps also.

2.2.12 RECSCALE

1. Description: The general scale-value for the recursion.
2. Default: "0.5"
3. Note: This is valid for all values, which have no own scale-value.

2.2.13 SLOPEANGLE

1. Description: The slope-angle for the branches.
2. Default: "40"

2.2.14 SANGLEDIFF

1. Description: The range for random variations of SLOPEANGLE.
2. Default: "0"
3. Note: Would be added to SLOPEANGLE.

2.2.15 ANGLESCALE

1. Description: The scale for SLOPEANGLE.
2. Default: “1.1”

2.2.16 NRBRANCHES

1. Description: The number of child-branches a branch or the trunk can have.
2. Default: “4”
3. Note: Here you can write some little funktions which make very good trees. (attention: you should keep in mind that the values of the function do not grow to big, the number of objects which build the tree can grow very fast to very gigantic values.)

2.2.17 NRBDIFF

1. Description: The range for random variations of NRBRANCHES.
2. Default: “0”
3. Note: Would be added to NRBRANCHES.

2.2.18 NRBSCALEADD

1. Description: One of two scales for NRBRANCHES.
2. Default: “0”
3. Note: This variable is a relict from old versions which did not understand functions. These values can better simpulated by functions in NRBRANCHES. But if you don't want to write functions, these two values are good alternatives.

   The value would be added to NRBRANCHES in every recursion-step.

2.2.19 NRBSCALEMUL

1. Description: The second scale for NRBRANCHES.
2. Default: “1”in every recursion-step.
3. Note: NRBRANCHES would be multiplied with NRBSCALEMUL in every recursion-step.

2.2.20 BRANCHTOP

1. Description: A threshold for the probability that a branch is put to the top of his predecessor.
2. Default: “0.6”
3. Note: If a random-value between 0 and 1 is bigger than BRANCHTOP, then the branch is put to a random place at his predecessor, otherwise it is put at the top.

2.2.21 LEAFINC

1. Description: Shoult the tree have leafs?
2. Default: “NO”

2.2.22 NRLEAFINCS

1. Description: How many different leaf-shapes are defined?
2. Default: 0
3. Note: This one must not set in parentheses.

   For a description how to define leafs see chapter 3 about leafs.

2.2.23 NRLEAFS

1. Description: How many leafs should a branch have?
2. Default: “0”
3. Note: Only the topmost branches have leafs.

2.2.24 NRLDIFF

1. Description: The range for random variations of NRLEAFS
2. Default: “0”
3. Note: Would be added to NRLEAFS.

2.2.25 ROTRANGE

1. Description: The range of rotation for the branches around their predecessors.
2. Default: "360"
3. Note: With a little experience you can simulate phenomena like wind with shrinking this value.

2.2.26 RANDINIT_ROT

1. Description: Seed for the random-generator for ROTRANGE.
2. Default: "1.1234"
3. Note: The changing of one of these values can change the shape of a tree dramatically. So if a tree has good structure but a bad shape, you can improve (or worsen) the shape with changing one of the seed-values.

2.2.27 RANDINIT_LDIFF

1. Description: Seed for the random-generator for LDIFF.
2. Default: "0.1234"

2.2.28 RANDINIT_RDIFF

1. Description: Seed for the random-generator for RDIFF.
2. Default: "0.1234"

2.2.29 RANDINIT_SDIFF

1. Description: Seed for the random-generator for SANGLEDIFF.
2. Default: "0.1234"

2.2.30 RANDINIT_NRBDIFF

1. Description: Seed for the random-generator for NRBDIFF.
2. Default: "0.1234"

2.2.31 RANDINIT_NRLDIFF

1. Description: Seed for the random-generator for NRLDIFF.
2. Default: "0.1234"

2.2.32 RANDINIT_LEAFSELECT

1. Description: Seed for the random-generator for the random-choosing of the leafs.
2. Default: "0.1234"

2.2.33 RANDINIT_LEAFTRANS

1. Description: Seed for the random-generator for the placing of the leafs at the branches.
2. Default: "0.1234"

2.2.34 RANDINIT_LEAFROT

1. Description: Seed for the random-generator for rotation of the leafs.
2. Default: "0.1234"

2.2.35 RANDINIT_BT

1. Description: Seed for the random-generator for the treshold for BRANCHTOP.
2. Default: "0.1234"

2.2.36 MOD_TR

1. Description: Texture for the trunk.
2. Default: the #default-Textur is used.
3. Note: You should define a full texture, not only a pigment-statement. Example:

```
texture {
   pigment {
      color DarkBrown
   }
   normal {
      bozo 0.2
      scale <0.05, 0.2, 0.08>
   }
}
```

2.2.37 MOD_X

1. Description: The textures for the branches in the different recursion-steps.

2. Default: the #default-Textur is used.

3. Note: The X in the name is a place-holder. X is a number between 0 and DEPTH-1 (0 means the branches at the trunk, DEPTH-1 means the topmost branches). This makes it possible to define a texture the branches of every recursion-step individually (for example it makes sense to define a smother texture for the small topmost branches as for the thick branches at the trunk).

2.2.38 #default

1. Description: A default-texture for the tree.

2. Default: { pigment { color rgb $< 0.35, 0.16, 0.14 >$ } }

3. Note: If you have defined your own default-texture, you should save it and restore it afer using "selftree.inc".

3 Leafs

To generate not only wintery trees, "selftree.inc" can add leafs to the trees, of course. To do thie, you must define leafs. "selftree.inc" can have different leaf-shapes at one tree. How many different shapes a tree has, is defined in NRLEAFINCS.

The names of the leaf-objects must be set as follows: LEAFINC_X where X is a number between 0 and NRLEAFINCS-1. Is the value of NRLEAFINCS 11, then "selftree.inc" looks for leafs with the names LEAFINC_0, LEAFINC_1, ..., LEAFINC_10.

It is presumed, that the shaft of the leaf starts at $< 0, 0, 0 >$ and the direction of the leaf is Y.

This is an example for a leaf-definition:

```
#declare LEAFINC_0 =
sphere
{
  <0, 0, 0>, 0.3
  scale <1, 0.1, 0.5>
  translate <0.3, 0, 0>
  pigment { color Green }
}
```

3.1 leafshapes.inc

This include defines a little number of leafs. If you render the file leafs.pov, you can see all the different leafs.

4 How does it work?

To get real recursion in POVRay, we have to take a little trick: POVRay string-functions are powerful enough to build POVRay-statements. So we build the needed statements and write them to the #debug-stream (or another stream, which is not used by POVRay itself). Then we send this stream to a file (with -GD_FILENAME_, if you use the #debug-stream) and include this file in our scene-file.

To get recursion, you only have to generate source-code with nested while-statements. That's all...

If you want to have a variable recursion-depth, you can use two while-statements: one for opening all recursion-steps, the other for closing them.

5 General stuff

5.1 History

1. Version 1.0: This is the firs finished release of the include. Previous versions though are released, but had no good docs and horribly formatted source-code. There may be incompatibilities to the older versions.

5.2 Todo

1. User-defined objects for the branch-shapes, for more realistic and interesting branches.
2. A "translation-tracker", to align branches and leafs to defined directions (for simulating gravity, wind etc.)
3. Nicer output with correct indention.
4. "Joker-variables" for choosing different classes of trees (for beginners, which don't want to set all the variables): small and many branches, slim and high or oak-like, pine-like — but that's very utopian.
5. Better and enhanced leafshapes.inc
6. A number of bark-textures in bark.inc
7. Generating other plants by setting a plant-class (at this time you can make a cactus with setting the variables to very strange values...)
8. Checking the variable-values.
9. Restore the default-texture.

5.3 Bugs

Surely there are some, but I haven't them yet. If you find some or have questions and suggestions, mail me:
stk@informatik.uni-bremen.de
(http://www.informatik.uni-bremen.de/˜stk)

5.4 Thanksgiving

Thanks for motivation and idea to the members of my little POVRay-playgroup Holger Bettag (http://www.informatik.uni-bremen.de/˜ hobold) and Harald Wagener (http://www.informatik.uni-bremen.de/˜hollow).

5.5 Copyright

Although POVRay come not out of the GNU-corner, this file is under GNU-copyright. Read the file COPYRIGHT for this.