Scenery3d - Walkable 3D Models in Stellarium

Georg Zotti*

February 7, 2015

1 Introduction

Have you ever wished to be able to walk through Stonehenge or other ancient building structures described as being constructed with astronomical orientation in mind, and experience such orientation in 3D?

The Stellarium Scenery3d plugin allows you to see architectural 3D models embedded in a landscape combined with the excellent representation of a sky simulation provided by Stellarium. You can walk around, check for (or demonstrate) possible astronomical alignments of ancient architecture, see sundials and other shadow casters in action, etc.

2 Usage

You activate the plugin with the *circular enclosure* button at screen bottom or by pressing [Ctrl+3]. The other button with circular enclosure and tool icon (or [Ctrl+Shift+3]) opens the settings dialog. Once loaded and displaying, you can walk around pressing [Ctrl] plus cursor keys. Change eye height with [Ctrl]+[PgUp]/[PgDn] keys. Adding [Shift] key increases speed by 10, [Alt] by 5 (pressing both keys multiplies by 50!). If you release [Ctrl] before the cursor key, animation will continue. (Press [Ctrl]+any cursor key to stop moving.)[1]

Further key bindings exist which can be configured using the Stellarium default key-binding interface. Some options are also available in the Scenery3d dialog. For example coordinate display can be enabled. If you have georeferenced models in a true geographical coordinate grid, e.g. UTM or Gauss-Krueger, you will especially like this, and this makes the plugin usable for scientific purposes. Display shows grid name, Easting, Northing, Altitude of ground, and eye height above ground.

3 Hardware Requirements

In order to work with the advanced projection models in Stellarium, this plugin uses a trick to create the foreground renderings: it renders the scene into the six planes of a cube map which are then drawn into the foreground. Your graphics card must be able to do this, i.e., it must support the OpenGL extension called `EXT_framebuffer_object`. Typical modern 3D cards (by NVidia or ATI/AMD) support this extension. In case your graphics hardware does not suppport it, it still works, but you are limited to perspective projection, and the program will switch to it as soon as you switch on, and switch back once you switch off the Scenery3d plugin.

You can influence rendering quality, but also speed, using the plugin's GUI, which provides some options such as enabling the use of shadows, bumpmapping or configuring the sizes of the textures used for the cubemap or shadowmaps. Larger values there improve the quality, but require

*`Georg.Zotti@univie.ac.at`, `http://astrosim.univie.ac.at`

[1] Bug or feature? I (GZ) had to change keyboard handling in the main program, somewhat breaking the plugin concept. TBD: Discuss with the core team.

faster hardware for smooth results. For further information on the different options, consult the tooltips in the interface.

4 Model Configuration

The model format supported in Scenery3d is Wavefront .OBJ, which is pretty common for 3D models. You can use several modeling programs to build your models. Software such as Blender, Maya, 3D Studio Max etc. can export OBJ.

A simple and cost-free modeling program is Google Sketchup, commonly used to create the 3D buildings seen in Google Earth. It can be used to create georeferenced models. OBJ is not a native export format for the standard version of Google Sketchup. If you are not willing to afford Sketchup Pro, you have to find another way to export a textured OBJ model.

One good exporter is available in the Kerkythea renderer project available at `http://www.kerkythea.net/joomla/`. You need `SU2KT 3.17` or better, and `KT2OBJ 1.1.0` or better. Deselect any selection, then export your model to the Kerkythea XML format with settings shown in figure 1.

Geometry	Yes
Lights	Yes
Clay	No
Photomatched	Yes
DefaultUVs	No
Instanced	No

Figure 1: Kerkythea Export Settings

You do not have to launch Kerkythea unless you want to create nice renderings of your model. Then, use the `KT2OBJ` converter to create an OBJ. You can delete the XML after the conversion. Note that some texture coordinates may not be exported correctly. The setting Photomatched:Yes seems now to have corrected this issue, esp. with distorted/manually shifted textures.

Recently, another free OBJ exporter has been made available by TIG: `OBJexporter.rb`[2]. This is the only OBJ exporter capable of handling large TIN landscapes (> 450.000 triangles). As of version 2.6 it seems to be the best OBJ exporter available for Sketchup.

This exporter swaps Y/Z coordinates, but you can add a key to the config file to correct swapped axes, see below. Other exporters may also provide coordinates in any order of X, Y, Z – all those can be properly configured.

Another (almost) working alternative: `ObjExporter.rb` by author Honing. Here, export with settings 0xxx00. This will not create a `TX...` folder but dump all textures in the same directory as the OBJ and MTL files. Unfortunately, this time some material assignments seem to be bad.

Yet another exporter, `su2objmtl`, does also not provide good texture coordinates and cannot be recommended at this time.

4.1 Notes on OBJ file format limitations

The OBJ format supported is only a subset of the full OBJ format: Only (optionally textured) triangle meshes are supported, i.e., only lines containing statements: `mtllib`, `usemtl`, `v`, `vn`, `vt`, `f` (with three elements only!), `g`. Negative vertex numbers (i.e., a specification of relative positions) are not supported.

A further recommendation for correct illumination is that all vertices should have vertex normals. Sketchup models exported with the Kerkythea or TIG plugins should have correct normals. If your model does not provide them, default normals will be reconstructed from the triangle edges, resulting in a faceted look.

[2] Available from `http://forums.sketchucation.com/viewtopic.php?f=323&t=33448`

Parameter	Default	Range	Meaning
Ka	set to Kd values	$0 \dots 1$ each	R/G/B Ambient color
Kd	0.8 0.8 0.8	$0 \dots 1$ each	R/G/B Diffuse color
Ke	0.0 0.0 0.0	$0 \dots 1$ each	R/G/B Emissive color
Ks	0.0 0.0 0.0	$0 \dots 1$ each	R/G/B Specular color
Ns	8.0	$0 \dots \infty$	shinyness
d or Tr	1.0	$0 \dots 1$	opacity
bAlphatest	0	0 or 1	perform alpha test
bBackface	0	0 or 1	render backface
map_Kd	(none)	filename	texture map to be mixed with Ka, Kd
map_Ke	(none)	filename	texture map to be mixed with Ke
map_bump	(none)	filename	normal map for surface roughness

Table 1: MTL parameters evaluated

Every `usemtl` statement must come after a `g` (group) statement and sets the material for the respective face group. If necessary, you must edit your OBJ file.

The MTL file contains the material parameters. The minimum that should be specified is either `map_Kd` or a `Kd` line specifying color values used for the respective faces. But there are other options in MTL files, and the supported parameters and defaults are listed in Table 1.

If no ambient color is specified, the diffuse color values are taken for the ambient color. An optional emissive term `Ke` can be added, which is modulated to only be visible during nighttime. It can optionally also be modulated by the emissive texture `map_Ke`.

If a value for `Ks` is specified, specularity is evaluated using the Phong reflection model with `Ns` as the exponential shininess constant. Larger shininess means smaller specular highlights (more metal-like appearance). Specularity is not modulated by the texture maps.

If a value for `d` or `Tr` exists, alpha blending is enabled for this material. This simulates transparency effects. Transparency can be further controlled using the alpha channel of the `map_Kd` texture.

A simpler and usually more performant way to achieve simple "cutout" transparency effects is alpha-testing, by setting `bAlphatest` to 1. This simply discards all pixels of the model where the alpha value of the `map_Kd` is below the `transparency_threshold` value from `scenery3d.ini`, making "holes" in the model. This also produces better shadows for such objects. If required, alpha testing can be combined with "real" blending-basded transparency.

Sometimes, some exported objects only have a single side, and are only visible from one side when looked at in Scenery3d. This is caused by an optimization called back-face culling, which skips drawing the back sides of objects because they are usually not visible anyway. If possible, avoid such "thin" geometry, this will also produce better shadows on the object. As a workaround, you can also set `bBackface` to 1 to disable back-face culling for this material.

The optional `map_bump` enables the use of a tangent-space normal maps, which provides a dramatic improvement in surface detail under illumination.

On reasonably good hardware (tested on a notebook PC with NVidia M9800 GTS), models up to 100.000 triangles are fluent, up to 250.000 are still "interactive". If display is too slow, switch to perspective projection: all other projections require almost sixfold effort! You can also try if the new "lazy" cubemap mode works for you, where the scene is only rendered in specific timesteps or when movement happens. See in the GUI for details.

4.2 Configuring OBJ for Scenery3d

The walkaround in your scene can use a ground level (piece of terrain) on which the observer can walk. The observer eye will always stay "eye height" above ground. Currently, there is no collision detection with walls implemented, so you can easily walk through walls, or jump on high towers, if their platform or roof is exported in the ground layer. If your model has no explicit ground layer, walk will be on the highest surface of the scenery layer. If you use the special name NULL as ground layer, walk will be above a zero-height level.

Technically, if your model has cavities or doors, you should export your model twice. Once, just the ground plane, i.e. where you will walk. Of course, for a temple or other building, this includes its socket above soil, and any steps. This plane is required to compute eye position above ground. Note that it is not possible to walk in several floors of a building, or in a multi-plane staircase. You may have to export several "ground" planes and configure several scenery directories for those rare cases.

The second export includes all visible model parts, and will be used for rendering. Of course, this requires the ground plane again, but also all building elements, walls, roofs, etc.

If you have not done so by yourself, it is recommended to separate ground and buildings into Sketchup layers in order to easily switch the model to the right state prior to exporting.

Filename recommendations:

```
<Temple>.skp        Name of a Sketchup Model file.
                    (The "<>" brackets signal "use your own name here!")
                    The SKP file is not used by Scenery3d.
<Temple>.obj        Model in OBJ format.
<Temple>_ground.obj Ground layer, if different from Model file.
```

OBJ export may also create folders TX_<Temple> and TX_<Temple>_ground. You can delete the TX_<Temple>_ground folder, <Temple>_ground.obj is just used to compute vertical height.

Stellarium uses a directory to store additional data per-user. On Windows, this defaults to C:\Documents and Settings\<username>\Application Data\Stellarium, but you can use another directory by using the command-line argument -user-dir <USERDATA>. We will refer to this directory. Put the OBJ, MTL and TX directories into a directory, <USERDATA>/Stellarium/modules/scenery3d/<Temple>, and add a text file called scenery3d.ini (This name is fixed!) with content described as follows.

```
[model]
name=<Temple>                 Unique ID within all models in scenery3d directory.
                              Recommendation: use directory name.
landscape=<landscapename>     Name of an available Stellarium landscape.
```

This is required if the landscape file includes geographical coordinates and your model does not: First, the location coordinates of the Landscape file are used, then location coordinates given here. The landscape also provides the background image of your scenery. - If you want a zero-height (mathematical) horizon, use the provided landscape called Zero.

```
scenery=<Temple>.obj          The complete model, including visible ground.
ground=<Temple>_ground.obj    Optional: separate ground plane. (NULL for zero altitude.)
description=<Description>     A basic scene description (including HTML)
```

The scenery3d.ini may contain a simple scene description, but it is recommended to use the new *localizable* description format: in the scene's directory (where scenery3d.ini lies) create files in the format description.<lang_code>.utf8 which can contain arbitrary UTF-8-encoded HTML content. lang_code stands for the ISO 639 language code.

```
author=<Your Name yourname@yourplace.com>
copyright=<Copyright Info>

obj_order=XYZ       | Use this if you have used an exporter which swaps Y/Z coordinates.
                    | Defaults to XYZ, other options: XZY, YZX, YXZ, ZXY, ZYX
camNearZ=0.3        This defines the distance of the camera near plane, default 0.3.
                    Everything closer than this value to the camera can not be
```

```
                    displayed. Must be larger than zero. It may seem tempting
                    to set this very small, but this will lead to accuracy issues.
                    Recommendation is not to go under 0.1
camFarZ=10000       Defines the maximal viewing distance, default 10000.
shadowDistance=<val>  The maximal distance shadows are displayed. If left out, the
                      value from camFarZ is used here. If this is set to a smaller
                      value, this may increase the quality of the shadows that are
                      still visible.
shadowSplitWeight=0..1 Decimal value for further shadow tweaking. If you require
                       better shadows up close, try setting this to higher values.
                       The default is calculated using a heuristic that incorporates
                       scene size.
```

```
[general]
```

The general section defines some further import/rendering options.

```
transparency_threshold=0.5   Defines the alpha threshold for alpha-testing,
                             as described in section 4.1. Default 0.5
scenery_generate_normals=0   Boolean, if true normals are recalculated by the
                             plugin, instead of imported. Default false
ground_generate_normals=0   Boolean, same as above, for ground model. Default false.
```

```
[location]
```

Optional section to specify geographic longitude λ, latitude φ, and altitude. Required if `coord/convergence_angle==from_grid`, else location is inherited from landscape.

```
planet = Earth
latitude = +48d31'30.4"       ; Required if coord/convergence_angle==from_grid
longitude = +16d12'25.5"      ; "--"
altitude =from_model|<int>    ;
```

altitude (for astronomical computations) can be computed from the model: if `from_model`, it is computed as $(z_{min} + z_{max})/2 +$ `orig_H`, i.e. from the model bounding box centre height.

```
display_fog = 0
atmospheric_extinction_coefficient = 0.2
atmospheric_temperature = 10.0
atmospheric_pressure = 1013.0
light_pollution = 1
```

```
[coord]
```

Entries in the `[coord]` section are again optional, default to zero when not specified, but are required if you want to display meaningful eye coordinates in your survey (world) coordinate system, like UTM or Gauss-Krueger.

```
grid_name=<string>
```

Name of grid coordinates, e.g. “`UTM 33 U (WGS 84)`”, “`Gauss-Krï¿½ger M34`” or `‘‘Relative to <Center>’’` This name is only displayed, there is no evaluation of its contents.

```
orig_E=<double> | (Easting)  East-West-distance to zone central meridian
orig_N=<double> | (Northing) North distance from Equator
orig_H=<double> | (Height)   Altitude above Mean Sea Level of model origin
```

These entries describe the offset, in metres, of the model coordinates relative to coordinates in a geographic grid, like Gauss-Krï¿½ger. If you have your model vertices specified in grid coordinates, do not specify `orig_...` data, but please add `start_...` data, below.

```
convergence_angle=from_grid|<double>
grid_meridian=<double>|+<int>d<int>'<float>"
```

Typically, digital elevation models and building structures built on those are survey-grid aligned, so true geographical north will not coincide with grid north, the difference is known as meridian convergence.

$$\gamma(\lambda, \varphi) = \arctan(\tan(\lambda - \lambda_0) \sin \varphi) \tag{1}$$

This amount can be given in `convergence_angle` (degrees), so that your model will be aligned with True North[3]. Central meridian λ_0 of grid zone, e.g. for UTM or Gauss-Krï¿½ger. `grid_meridian` is only required to compute convergence angle if `convergence_angle="from_grid"`

```
zero_ground_height=<double>
```

height of terrain outside `ground.OBJ`, or if `ground=NULL`. Allows smooth approach from outside. This value is relative to the model origin, or typically close to zero, i.e., use a Z value in model coordinates, not world coordinates! (If you want the terrain height surrounding your model to be `orig_H`, use 0, not the correct mean height above sea level!) Defaults to minimum of height of ground level (or model, resp.) bounding box.

```
start_E=<double>
start_N=<double>
start_H=<double> /* only meaningful if ground==NULL, else H is derived from ground */
start_Eye=<double> /* default: 1.65m */
start_az_alt_fov=<az_deg>,<alt_deg>,<fov_deg> /* initial view direction and field of view.*/
```

`start...` coordinates to be set after loading the scenery. Default to center of model boundingbox.

It is advisable to use the grid coordinates of the location of the panoramic photo ("landscape") as `start_..` coordinates, or the correct coordinates and some carefully selected `start_az_alt_fov` in case of certain view corridors (temple axes, ...).

4.3 Working with non-georeferenced OBJ files

There exists modeling software which produces nice models, but without concept of georeference. One spectacular example is AutoDesk PhotoFly, a cloud application which delivers 3D models from a bunch of photos uploaded via its program interface. This "technological preview" is in version 2 and free of cost as of mid-2011.

The problem with these models is that you cannot assign surveyed coordinates to points in the model, so either you can georeference the models in other applications, or you must find the correct transformation matrix. Importing the OBJ in Sketchup may take a long time for detailed photo-generated models, and the texturing may suffer, so you can cut the model down to the minimum necessary e.g. in Meshlab, and import just a stub required to georeference the model in Sketchup.

Now, how would you find the proper orientation? The easiest chance would be with a structure visible in the photo layer of Google Earth. So, start a new model and immediately "add location" from the Google Earth interface. Then you can import the OBJ with TIG's importer plugin. If the imported model looks perfect, you may just place the model into the Sketchup landscape and export a complete landscape just like above. If not, or if you had to cut/simplify the OBJ to be able to import it, you can rotate/scale the OBJ (it must be grouped!). If you see a shadow in the photos, you may want to set the date/time of the photos in the scene and verify that the

[3] `http://en.wikipedia.org/wiki/Transverse_Mercator_projection`

shadows created by Sketchup illuminating the model match those in the model's photo texture. When you are satisfied with placement/orientation, you create a `scenery3d.ini` like above with the command `Plugins->ASTROSIM/Stellarium scenery3d helpers->Create scenery3d.ini`.

Then, you select the OBJ group, open `Windows->Ruby Console` and call `Plugins->ASTROSIM/Stellarium scenery3d helpers->Export transformation of selected group (e.g., from PhotoFly import)`.

On the Ruby console, you will find a line of numbers (the 4×4 transformation matrix) which you copy/paste (all in one line!) into the `[model]` section in `scenery3d.ini`.

```
obj2grid_trafo=<a11>,<a12>,<a13>,<a14>,<a21>,<a22>,<a23>,<a24>,
               <a31>,<a32>,<a33>,<a34>,<a41>,<a42>,<a43>,<a44>
```

You edit the `scenery3d.ini` to use your full (unmodified) PhotoFly model and, if you don't have a panorama, take `Zero` landscape as (no-)background. It depends on the model if you want to be able to step on it, or to declare `ground=NULL` for a constant-height ground. Run Stellarum once and adjust the `start_N`, `start_E` and `zero_ground_height`.

4.3.1 Rotating OBJs with recognized survey points

If you have survey points measured in a survey grid plus a photomodel with those points visible, you can use Meshlab to find the model vertex coordinates in the photo model, and some other program like CoordTrans in the JavaGraticule3D suite to find either the matrix values to enter in `scenery3d.ini` or even rotate the OBJ points. However, this involves more math than can be described here; if you came that far, you likely know the required steps. Here it really helps if you know how to operate automatic text processors like AWK.

Authors and Acknowledgements

Scenery3d was conceived by Georg Zotti for the Astrosim project. It was implemented originally in 2010/2011 by Simon Parzer and Peter Neubauer as student work supervised by Michael Wimmer (TU Wien). Improvements in integration, user interaction, .ini option handling, OBJ/MTL loader bugfixes and georeference testing by Georg Zotti. Andrei Borza (again supervised by Michael Wimmer) in 2011/12 further improved rendering quality (shadow mapping, normal mapping) and speed.

This work has been created during the ASTROSIM project supported 2008-2012 by the Austrian Science Fund (FWF) under grant number P 21208-G19.