NormalMap Plugin

White Paper

Using Normal Maps In GTA San Andreas

2015

Table of Contents

1. Introduction 3
2. Requirements 4
3. Installing Normal Map plugin 5
4. Adapting your models to Normal Map Plugin 6
    Importing normal map textures into TXD archive 6
    Attaching normal maps to DFF files with nm_matbuild tool 6
    Attaching normal maps to DFF files with MatFx tool 8
    Normal maps for player model 10
6. Normal Map Plugin settings 11
    General 11
    Ambient 11
    Directional 11

1. Introduction

NormalMap is an .ASI plugin for GTA San Andreas (named as “normalmap_byDK.asi”) which gives an ability for modmakers to use normal maps with their models.

2. Requirements

Normal Map Plugin requires GTA San Andreas with 1.0 US executable.

Also, for loading .ASI plugins you need to have an ASI Loader installed for your game. It is recommended to use Silent's ASI Loader.

3. Installing Normal Map plugin

An installation process is pretty simple. Just copy following items to your GTA SA folder:

- normalmap_byDK.asi – a plugin itself
- normalmap_byDK.ini – configuration file for plugin
- normalmap_player – folder for player model normal maps (empty by default)

Notice the plugin can be launched even without additiona files. If the .ini file is not presented, a plugin will load default settings. And if a folder ("normalmap_players") is not exist, it just means the plugin won't load any normal maps for player model.

You can also install a plugin into "GTA SA\scripts" folder.

4. Adapting your models to Normal Map Plugin

The process of adapting your models can be divided into two processes:

- importing normal map textures into txd archive
- attaching normal maps to materials in dff model

And also, there are two ways to attach normal maps to dff mode:

- using nm_matbuild tool
- using MatFx tool

You must also know that Normal Map Plugin gives you an option to attach reflection map to material.

Attaching Normal Map to material means that material will lose such settings as specular and environment map (used for vehicle models in San Andreas). Also, it's impossible to combine Normal Map with MatFX effects.

If a geometry uses 2 UV sets (or more) and you attach normal map to its material, you could see wrong results in a game. Thus nm_matbuild tool deletes additional UV sets for geometry and leaves only one (first) set.

Adapted models will work normally on a game without Normal Map Plugin installed.

Importing normal map textures into TXD archive

It is recommended to use DDS2TXD plugin to generate TXD archives.

In a usual way, use DXT1/no alpha for textures without an alpha channel and DXT5/interpolated alpha for textures with an alpha channel.

There are no any restrictions about normal map texture names inside TXD. But in practing of game development such suffixes as "_n" or "_norm" are often used for normal map names (for example, "tree_n"). Also, remember texture names inside TXD archives are limited to 31 character in length.

Attaching normal maps to DFF files with nm_matbuild tool

Nm_matbuild tool attachs normal maps to materials inside DFF files by material's texture name. The tool can be set up within "nm.txt" file.

This file has following syntax:

```
#
# A DFF file path
# B Texture name
# C Normal map texture name
# D Normal map texture alpha mask name     --use "-" if texture has no alpha
# E Normal map texture filtering
# F Normal map texture u addressing
# G Normal map texture v addressing
# H Normal map texture has mipmaps
# I Reflection map texture name            --optional. If NM doesn't use
reflection, skip these parameters
# J Reflection map texture alpha mask name --use "-" if texture has no alpha
# K Reflection map texture filtering
# L Reflection map texture u addressing
# M Reflection map texture v addressing
# N Reflection map texture has mipmaps
# O Reflection intensity
# P Modulate reflection map with normal map
#
# A         B     C       D E F G H I       J K L M N O   P
willard.dff wheel wheel_n - 6 1 1 1 wheel_r - 6 1 1 1 2.0 1
```

Lines marked with “#” at the beginning are ignored by tool. These lines are commentaries. They explain parameters which are used in this line:

```
willard.dff wheel wheel_n - 6 1 1 1 wheel_r - 6 1 1 1 2.0 1
```

Notice parameters since ‘I’ parameter are optional. You don’t need them if you don’t plan to add reflection map with normal map to material. So the line will look so in this case:

```
willard.dff wheel wheel_n - 6 1 1 1
```

In most of cases, you won’t need to touch parameters other than texture names and reflection intensity. So use them only if you know what do they mean.

As usual, for texture filtering use “2” if texture has no mipmaps and “6” if it has them.

```
1 FILTER_NEAREST
2 FILTER_LINEAR
3 FILTER_MIP_NEAREST
4 FILTER_MIP_LINEAR
5 FILTER_LINEAR_MIP_NEAREST
6 FILTER_LINEAR_MIP_LINEAR
```

For addresing mode, use these values:

```
 1 WRAP_WRAP
 2 WRAP_MIRROR
 3 WRAP_CLAMP
```

For “has mipmaps” parameter use “1” if texture has mipmaps and “0” if it hasn’t.

For texture alpha mask name (if texture uses alpha) use the same texture name but with “a” suffix (for example, “reflection” and “reflectiona”).

For reflection intensity, it is normal to use “1.0” value. But you can experiment with higher and lower values too.

Notice you can’t use spaces (“ “) within dff path or texture names.

Number of lines inside nm.txt is not limited anyhow. You can use few lines to set up different materials within one dff model or materials in different dff models.

If there were any problems while attaching normal maps, you’ll see an error message in tool window:

- “File “nm.txt” not found” – A tool can’t find this file. Check if this file is placed in the same folder with nm_matbuild.exe.
- “File “filename.dff” not found” – A tool can’t find this dff model. Check if a path is correct.
- “Can’t find material with texture “texture”” – A tool can’t find material with such texture name in dff model.
- “Error when reading dff model” – There was an error when reading dff model. Check if everything fine with this model.

Attaching normal maps to DFF files with MatFx tool

There’s also an other way (which is much harder if you don’t know how to use RwAnalyze) to attach normal map to material in DFF file. It requires such tools as MatFx Section Builder and RwAnalyze.

MatFX tool builds Normal Map data file (.mfx) which could be attached to material inside dff model then. Read here more about MatFx Section Builder.

Normal Map data has following format:

DffVersion: 0x1803FFFF

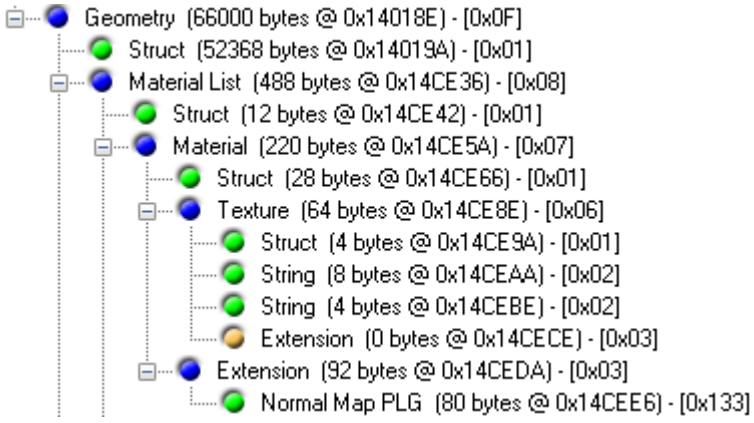
```
MatFxType   NORMALMAP
### parameters ###
###        Enable Filtering U-addressing V-addressing HasMipMaps Name
Alpha
NormalMap   1     6        1           1           1         Muddy_N -
### Env Map
###        Enable Filtering U-addressing V-addressing HasMipMaps Name
Alpha
EnvMap      1     6        1           1           1         Muddy_R -
EnvMapCoefficient 1.0
EnvMapModulate 1
```

As you see, it has same parameters with nm_matbuild tool. The only difference is that if you don't want to include reflections, set "Enable" parameter to "0" for EnvMap texture.

```
EnvMap    0    6    1    1    1    -  -
```

After you built a .mfx file, you will need to import it to DFF model with

RwAnalyze. Create a new section inside Material's Extension (Clump > Geometry List > Geometry > Material List > Material > Extension) with type "Normal Map PLG" (which has an ID 0x133). After this, import .mfx file as a data into this section.

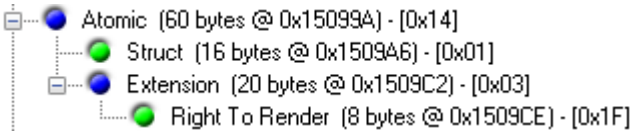


Notice you need to delete all other Material Extension (if they are presented) if you attach Normal Map plugin data (such as Specular, Reflection, and Material Effects).

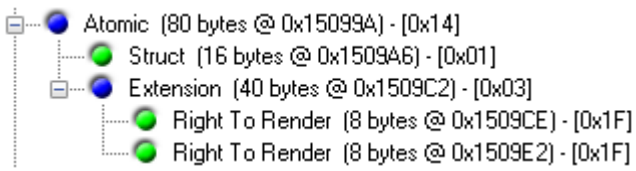
Also, you need to import some specific data to Atomic which use this material. Create a new section in Atomic's Extension (Clump > Atomic > Extension) with type "Right To Render" (which has an ID 0x1F). After this, import:

- normalmap_static.rights file if model is not skinned
- normalmap_skin.rights file if model is skinned

For non-skinned atomic, it will look like this:



For skinned atomic, the result will be as following:



Normal maps for player model

Since player model has different structure than other models (it is constructed dynamicly from different parts), there is a special way to setup normal maps for player model. However, it also has some limitations.

- The first and the main difference is that you don't need to touch dff model files anyhow.
- Normal map textures are stored inside .txd files inside normalmap_player folder.
- The Normal Map Plugin detects that player use normal mapping if there are any .txd files placed in normalmap_player folder.
- TXD archives inside normalmap_player are linked with TXD archives inside player.img. So, for example, if we have player_torso archive, we need to create payer_torso archive (with normal map textures) in normalmap_player folder.
- Normal map textures are called in same way as diffuse textures, just with "_n" at the end.

There is a file "normalmap_env_settings.dat" for setting up environment (reflection) maps. These textures must be placed in .txd with normal map texture.

The line has following format:

```
# bodypart # texture name # env coeff # env modulate
   0          tex_reflection    1.0          1
```

If reflection is not used, "texture name" parameter must be set to "-".

"Normal Maps For Player" thing is mainly oriented for global player modification, such as full player model/textures replacement. It supposes that all player textures have their normal maps. So if some part hasn't, you could see some strange result in a game.

6. Normal Map Plugin settings

There is a settings file (normalmap_byDK.ini) comes with Normal Map Plugin. You can setup some lighting parameters there.

General

UpdateSettingsEachFrame – if true (1), settings will be updated each frame. Gives you an ability to change values in a file and do a quick check in a game without closing it. A value in ranges [0;1]

MoonIntensity – intensity of moon light at night. A value in ranges [0.0;1.0]

Ambient

AmbientMultR, AmbientMultG, AmbientMultB – ambient light intensity. It is recommended to use values in ranges [0.0;1.0], but you can also try higher values.

Directional

UseSunColors – use sun colors (those defined in timecyc.dat file) for directional light. A value in ranges [0;1]

SunColorsContribution – intensity of sun colors (if UseSunColors parameter is enabled). A value in ranges [0.0;1.0]

DirectionalMultR, DirectionalMultG, DirectionalMultB – directional light intensity. It is recommended to use values in ranges [0.0;1.0], but you can also try higher values.

Due to RenderWare limitations, models with normal mapping can be affected only by one directional light. Thus, you could see how lights switching between one to another at night time.