Advanced 3D Texture Filtering extension

by MantaGames

Introduction

The advanced 3D texture filtering extension can be used to improve the visual appearance of any 3D scene by using hardware-accelerated texture filtering methods to reduce visual artefacts (or artifacts), texture-banding and noise in the final image of a scene.

Texture filtering is considered one of the most important processes in improving the graphical fidelity of the scene.

Features:

- Enable fast hardware-accelerated texture filtering with one line of code.
- Reduces visuals artefacts in the rendered 3D scene
- Filtering modes including:
    - **Linear filtering:** Fast and approximate
    - **Nearest point filtering:** Fastest method, good for pixel-art styled 3D games.
    - **Bilinear filtering:** Improvement upon regular linear filtering
    - **Anisotropic filtering:** Best looking form of filtering (Up to 16 samples)
- Almost no performance impact even on most demanding filtering method.
- Functions can be called at any time during the games execution.
- Can be used in conjunction with shaders
- **Very easy to integrate into existing games**

Usage:

In order to integrate this into your project, you must call tf_init(); at the very start of your game. This function loads the extension allowing it to be used later on.

tf_init()

This function must be called to initialise the Extension.

texture_set_filtering(Filtering Type)

This function enables filtering on the base sampler. The following values can be used:

```
TextureFiltering {
    NONE = 0,
    LINEAR = 1,
    NEAREST_POINT = 2,
    BILINEAR = 3,
    ANISOTROPIC = 4
}
```

For readability, you can use the enumeration: TextureFiltering.
Use example:

```
texture_set_filtering(  TextureFiltering.ANISOTROPIC  );
```

texture_set_filtering_ext(Filtering Type, sampler index)

This allows you to perform the same operation as with texture_set_filtering, however you can control which texture sampler the filtering is enabled for. This is designed for use with GameMaker's shader functions.

Filtering Mode – same as for texture_set_filtering

Sampler index – Sampler index from 0 to 7, as returned from shader_get_sampler_index(..)

Use example:

```
shader_set( sh_Depth );
texture_set_filtering_ext(  TextureFiltering.ANISOTROPIC,
                            shader_get_sampler_index( sh_Depth, "depthTex"  );
```

texture_set_filtering_ext_f(Filtering Type, sampler index, filter Type)

This function extends the functionality of texture_set_filtering_ext to allow you to decide which filter you are modifying. A different filter is applied depending on the situation, and you can control which type of texture filtering is used with each filter.

FilterType {

- **MinFilter** =0: the filter applied to textures which are squashed. (Minified)
- **MagFilter** =1: the filter applied to textures which are stretched or magnified.
- **MipFitler** =2: Used when the texture is mipmapped. (Mipmapping is the process of splitting a texture up into smaller sized intervals to reduce grain when a texture is viewed at distance.)

}

texture_set_anisotropy(Anisotropy level)

This function must be used in conjunction with texture_set_filtering(..) when the filtering type is set to anisotropic. Anisotropic filtering is a more advanced form of filtering in which the anisotropy (I.E Elongation of a pixel) is calculated and factored into the filtering process.

Sometimes it can be beneficial to combine anisotropic filtering with other methods of filtering depending on the desired visual appearance of the game.

This function allows control over how many times the filtering is sampled. A higher value will invoke a larger performance cost; however the final image result will look better.

Anisotropy level – Value between 0 and 16. (Higher = better looking, but slower)

Use example:

```
texture_set_filtering (  TextureFiltering.ANISOTROPIC  );
texture_set_anisotropy( 8 );
```