ER: Creating Fake UI Elements Using SFX

Tutorial Author: Seagoingnote1

Introduction

This guide will cover using sfx as fake UI elements, including adding/positioning the sfx. This guide will use CCCode's JavaScript fxr library for ease of editing, and any js files used will be included at the end.

Tools used:

WitchyBND - unpacking and repacking sfxbnd_commoneffects

CCCode's fxr library - sfx editing and creating

FXR spreadsheet - fxr ids and descriptions

UXM - only if you don't have your files unpacked/don't have access to your sfx files

JS UI Tutorial File – tutorial file, used with CCCode's libary

Have all of these setup before continuing in this tutorial, all of them have setup instructions

Step 1 - Preparation and basic explanation, can be skipped if you've modded sfx before

First navigate to the sfx folder, make a copy of sfxbnd_commoneffect, this one loads on all maps so we'll be using it exclusively for this tutorial. Unpack the sfxbnd_commoneffect file with WitchyBND, then navigate into the unpacked folder. You should see a selection of folders including model, effect, resourcelist, and tex or texture depending on your WitchyBND version.

Step 2 - Picking a model or texture

Any model or texture can be used to make an sfx, although the exact setup of the Javascript file will change depending on what you pick. The model folder contains flver files that have a 3D model with a texture assigned to them, for example s88300.flver is the model for the carian glintblade, and links to the textures s88300_a and s88300_em. You can view these models directly with a tool such as Pear's flver editor or look at the fxr spreadsheet to see which models are used in which fxr. The Texture folder contains both textures that are used in conjunction with models such as the glintblade textures (88300) and textures that are used by themselves without a model such as the carian sigil (60140). Pick what you want to use and write down it's file name

Step 3 - Writing javascript to make an sfx

I've given you a javascript file labeled "UI Element Maker - Tutorial". I'll be using that as a basis for this tutorial, download it and open it in any text editor you like. Please note that I won't be explaining every bit of this file, only the nodes of the fxr itself and a bit about states. There are comments in the file itself if you're curious and want to know more. Also see CCCode's github for the library as it has several general sfx examples

NodeTransform

This moves and rotates the sfx relative to the dummypoly it is attached to, the fields offsetZ, offsetX, and offsetY control position while rotationX, rotationY, and rotationZ control rotation.

Important: positive offsetZ places the sfx in front of the camera, so always have it at a slight positive offset, I use 1, the closer it is to the camera the less clipping you get with walls. OffsetX controls horizontal placement, and offsetY controls vertical placement.

This is the thing that typically takes the longest to tweak, and I recommend a way to reload sfx midgame. The numbers already in the file place the sfx just above the ash of war name on the bottom left.

Example

```
NodeTransform({
      offsetZ: 1,
      offsetX: -0.635,
      offsetY:  -0.09,
      rotationY: 90,
      rotationZ: 99,
    }),
```

NodeAttachToCamera

This connects the sfx to the camera, causing it to move with the rotation and motion of the camera; it has no fields or attributes to edit. Just make sure it's present

Example

```
new NodeAttachToCamera,
```

RichModel, BillboardEX, and Model

Richmodel, BillboardEX, and Model are used to create particles using textures or models. In this tutorial these are what will give the sfx its appearance. All three share several similar fields, either in name or function. Those fields will be discussed here while a description of each action can be found below.

Shared Fields:

model/texture - model is used in richmodel and model, texture is used in BillboardEx. Both require an integer value, Model/Richmodel takes a flver file id from the model folder while BillboardEX uses a

texture file id from the tex folder. Keep in mind both only use the number component so the carian sigil as a texture is texture: 60140 rather than texture: s60140_a.

SizeX/Width - SizeX and Width control the same thing, width of a given particle. sizeX is used for model/Richmodel and width is used for BillboardEx, UniformScale (shared between model/Richmodel and BillboardEx) allows sizeX and Width to control the overall scale of the particle.

Color1, color2, and color3 - these 3 fields control the color of an sfx and can be used on model, richmodel, and BillboardEx.

Important: A lot of particles that use model or Richmodel will not have their color controlled by these fields while BillboardEx typically will.

Richmodel/Model

Richmodel/Model is used to add a 3D model to a particle, use this anytime you want to use a flver file as the particle, try model first then richmodel if model doesn't work properly

Example

```
new RichModel({ //use for 3d particles when model doesn't work
      model: 88300,
      uniformScale: true,
      color1: colors[5], //a number of models/rich models don't use the
color attribute, so changing this will do nothing
      color2: colors[5], //for a decent number of sfx.
      color2: colors[5],
      blendMode: BlendMode.Add,
      sizeX: 0.2, //used for scale of the sfx
    }),
```

BillboardEx

BillboardEx is used to add 2D effects such as sigils for spells or pseudo 3D effects such as the spear the cleanrot knights use. Use this when you want a texture file as the particle such as any sigil.

Example:

```
new BillboardEx({ //used for 2D textures, such as school of magic sigils
      texture: 60270,
      uniformScale: true,
      color1: colors[5], //these can almost always if not always be recolored
      color2: colors[5],
      color2: colors[5],
      blendMode: BlendMode.Add,
      width: 0.2, //used for scale of the sfx
    })
```

States

States can be used to control the behavior of a particle based on external factors, such as having fire sizzle in the rain. For these sfx we've added a very common state that simply makes the sfx go away when the thing making it appear goes away. So when an SpEffect adding an sfx expires the sfx will expire as well.

Example:

```
fxr.states = [
  State.from('ext(0) < 1') //makes the sfx go away when the SpEffect it's attached to is removed
]
```

Final Notes

This is a very simple explanation that only details the very basics of adding elements as UI, and can largely be treated as a basic explanation of the nuances of positioning UI sfx and an overview of creating them. There are literally hundreds of other things you can add to this example to improve it as an sfx. So please don't take this as the only way to do this, it can and should be improved. If you have any questions, please ping me (@seagoingnote1) on ?ServerName? and I'll be happy to answer.