This is the second special FX tutorial

Introduction

Hi everybody,

This tutorial is about screenfading. For this and all tutorials I assume that you have some C++ experience and that the basics are clear to you. This tutorial will show a 2D image on screen and when you press the X button the screen will fade to black. This tutorial continues on the creating 2D on 3D tutorial.

The graphics object

As with a lot of other functions throughout these tutorials, the screenfading will be done using one function in the graphics object. We will create this effect using a 2D sprite which has alpha values. Alpha values tell how transparent that color is. For example an alpha value of 255 is fully visible but a value of 0 is full transparent. Lets alter our graphics object so that we can use alpha channels:

GraphicsObject.cpp

```
sceGuEnable(GU_BLEND);
sceGuBlendFunc(GU_ADD, GU_SRC_ALPHA, GU_ONE_MINUS_SRC_ALPHA, 0, 0);
```

We turn on blending and then we set our blend function. These lines have to be in the Init function of the graphics object. The first parameter sets the blending operation. There are six options:

- GU_ADD - (Source color*Blend function for source fragment) + (Destination color*Blend function for destination fragment)
- GU_SUBTRACT - (Source color*Blend function for source fragment) - (Destination color*Blend function for destination fragment)
- GU_REVERSE_SUBTRACT - (Destination color*Blend function for destination fragment) - (Source color*Blend function for source fragment)
- GU_MIN - Source color < Destination color ? Source color : Destination color
- GU_MAX - Source color < Destination color ? Destination color : Source color
- GU_ABS - |Source color-Destination color|

The calculations behind the items tell what exactly will be done with that option during the blend. The second and the third parameter set the blendfunctions, which the options above use, of the source and destination. You can choose one of the following items:

- GU_SRC_COLOR
- GU_ONE_MINUS_SRC_COLOR
- GU_SRC_ALPHA
- GU_ONE_MINUS_SRC_ALPHA
- GU_DST_ALPHA
- GU_ONE_MINUS_DST_ALPHA
- GU_DST_COLOR
- GU_ONE_MINUS_DST_COLOR
- GU_FIX

The last two parameters are for when you use GU_FIX but we do not use that so we set them 0. I will explain this usage in another tutorial. So now we have enabled blending we can create our function:

```
int GraphicsObject::RenderScreenFade(const unsigned int r, const unsigned int g,const unsigned
intb,const unsigned int alpha) {
```

This is our screenfade function. We pass the RGB colors and the alpha values.

```
   sceGuDisable(GU_DEPTH_TEST);
   sceGuDisable(GU_TEXTURE_2D);
```

Here we disable the depth test because it will be a 2D sprite and the texture2d because we will only use color and no texture.

```
   vertexc2d* vertices = (vertexc2d*) sceGuGetMemory(2 * sizeof(vertexc2d));

   vertices[0].color = GU_RGBA(r, g, b, alpha);
   vertices[0].x = 0;
   vertices[0].y = 0;
   vertices[0].z = 0;
   vertices[1].color = GU_RGBA(r, g, b, alpha);
   vertices[1].x = 480;
   vertices[1].y = 272;
   vertices[1].z = 0;
```

This code should also look familiar. We use the passed RGB and alpha values to color our vertices and thus our 2D rectangle. A sprite only needs two points, one of the left top corner and the second of the right bottom corner. In our case we use (0,0) to (480,272) so that it fills our whole screen. The GU_RGBA function lets us create a color int using the RGB and alpha values. We need this because we alter the alpha channels a lot. The z coordinate as always is 0 because we render it in 2D.

```
   sceGuDrawArray(GU_SPRITES, GU_COLOR_8888 | GU_VERTEX_32BITF | GU_TRANSFORM_2D, 2, 0, vertices);
```

Here we draw our sprite with the vertices.

```
   sceGuEnable(GU_TEXTURE_2D);
   sceGuEnable(GU_DEPTH_TEST);

return 0;
};
```

This is the function to fade in and fade out the screen. We also have to make some changes to the header file:

GraphicsObject.h

```
typedef struct { unsigned int color;
            float x, y, z; } vertexc2d;
```

This struct is the structure of the vertices for our screenfade. Since we have used our 2d on 3d tutorial source, there will already be a vertex2d structure which holds coordinates and UV's. We only want coordinates and color so we had to create this new structure.

```
int RenderScreenFade(const unsigned int r, const unsigned int g,const unsigned int b,const unsigned
int alpha);
```

We also have to declare the function off course. Now we are done with altering the Graphics Object. Lets alter the GameApp to get some nice working screenfader in our example.

GameApp

We need to create some variables and code to fade our screen. First we will alter the GameApp.h file:

GameApp.h

```
bool fadeout;
int fade;
```

We use the fade int to set the alpha channel and the bool to check whether the user has pressed the X button so that we can begin with the fade.

GameApp.cpp

```
// setting fadeout to false
fadeout = false;
fade = 0;
```

These lines need to be in the load function. Here we set the variables to default values.

```
if (pad.Buttons != 0){
   if (pad.Buttons & PSP_CTRL_CROSS){
      fadeout = true;
   }
}
```

We adjust the cross button press to set the fadeout to true. So when we press the cross button the screen starts to fade.

```
// fade the screen
if(fadeout) {
   fade++;
   if(fade>254)
      fadeout=false;
}
gfx->RenderScreenFade(0,0,0,fade);
```

These lines need to be after the function call to render the 2D image on screen. What happens? We check whether the screenfade is started with the boolean. If it is started we increment the fade value. If it goes above 254 we stop with the screen fade because the screen is all dark at that moment. The last line calls the function we already have discussed. We pass three times 0 which is together the color black in RGB and we also pass the fade variable. This variable is incremented everytime the render function is called. So this means that the rectangle has less transparency every time the render function has been called.

So now we are done with our screenfade. Compile and what happens ?! First you see the 2D texture on screen but when you press the X button the screen fades to black. (Do not forget to include the texture.png to the EBOOT.PBP)

MakeFile

The makefile for tutorial FX2

```
TARGET = out
```

```
OBJS = $(wildcard *.cpp) $(wildcard *.c)

INCDIR =
CFLAGS = -O2 -G0 -Wall -g
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti -g
ASFLAGS = $(CFLAGS)

LIBDIR =
LDFLAGS =
LIBS = -lc -g -lpng -lz -lpspgum -lpspgu -lstdc++ -lm

PSPSDK=$(shell psp-config --pspsdk-path)
include $(PSPSDK)/lib/build.mak
```

Source files

Download the source files.