Introduction

Hi everybody,

With all those optimization techniques from the previous tutorials, you are probably wondering is there any more? Can there be more? Well that answer is quite easy; yes and even a lot more than all these tutorial will cover. Anyway, on to the VFPU!

The VFPU

The VFPU is a vector floating point unit, which can be used for fast calculations. This tutorial will not explain how to use assembly and how to code the most amazing stuff with it, it will however show us how we can use the VFPU and this tutorial will show some examples. Using the VFPU will increase speed extremely, for example, I have reduced the speed of one frame in a 3D application down to 25% of its original time (increasing the fps factor 4!!).

Enable the VFPU

Your game or application has to enable the VFPU before it can be used.

```
PSP_MAIN_THREAD_ATTR(PSP_THREAD_ATTR_USER | THREAD_ATTR_VFPU);
```

This function is usually set in the main.c or main.cpp file, I added the THREAD_ATTR_VFPU flag. This tells the PSP that the VFPU is used. Whenever the PSP uses the VFPU for another thread other than yours, it restores the state when it uses your thread again. At this point we have enabled the VFPU so now we can use it!

A sideway: Quick speed improvements

If you already have a large project with a lot of 3D stuff in it you can easily increase the execution speed by changing some library information in the makefile. The libraries -lpspgum -lpsp should be changed to -lpspgum_vfpu -lpspvfpu in order to use the VFPU versions, which are faster. This will be noticeable when using a lot of 3D transformation math in the project.

VFPU the beginnings

So now we will use some VFPU functions instead of the normal functions we often use. Lets take a look at an example:

```
float ghtMath::x_sinf(float x) {
   float result;
   __asm__ volatile (
      "mtv %1, S000\n"
      "vcst.s S001, VFPU_2_PI\n"
      "vmul.s S000, S000, S001\n"
      "vsin.s S000, S000\n"
      "mfv %0, S000\n"
      : "=r"(result)
      : "r"(x)
   );
   return result;
}
```

Wow!! whats this? Well this is the sine function using the VFPU. I do not know the exact difference with the sinf C version but the VFPU version is a lot faster. In an early build of Raze (a game in development) it got 130 frames per second, which is not bad, but after introducing the new sin, cos and sqrt VFPU version the speed increased to 350 frames per second. That made me create more functions with the VFPU, which in the end got the fps to 495! This is the power of the VFPU!

Anyway, you all are probably wondering what that code means. I will try to explain. The mtv command sets the float x in the register S000, the second line loads a constant (which is know by the VFPU) into register S001. The third line multiplies register S000(float x) with S001(constant 2 divided by pi) and stores it into S000. The fourth line then stores the sin of S001 into S001. The mfv command moves the S001 into the float result.

I have created a utility class to do all sorts of math stuff. In this class I have added a lot of functions that speed up the execution. I recommend that you also create such a class, it will be very helpful.

This is all you need to know on the VFPU. I will give some other examples in the next section but if you want to create stuff yourself and are in need of a good tutorial on ASM and VFPU, read the tutorial of mrmrice. On to the examples.

More examples

Cosine function:

```
float ghtMath::x_cosf(float x) {
   float result;
   __asm__ volatile (
      "mtv %1, S000\n"
      "vcst.s S001, VFPU_2_PI\n"
      "vmul.s S000, S000, S001\n"
      "vcos.s S000, S000\n"
      "mfv %0, S000\n"
      : "=r"(result)
      : "r"(x)
   );
   return result;
}
```

Identity matrix:

```
void ghtMath::x_identity_matrix(ScePspFMatrix4 *m) {
   __asm__ volatile (
      "vmidt.q M000\n"
```

```
        "sv.q C000, 0 + %0\n"
        "sv.q C010, 16 + %0\n"
        "sv.q C020, 32 + %0\n"
        "sv.q C030, 48 + %0\n"
    :"=m"(*m));
}
```

The worldwide web is full of other useful functions using the VFPU, search them and use them.