Introduction

Hi everybody,

This series of tutorials will be on the subject optimization. The main goal of this series is to optimize your code so that you can create as much detail and content as you can add. These tutorials are for noobs, newbies and maybe some of you more advanced coders may learn a few more tricks. In this tutorial I will just give some general optimization and I will probably forget loads of other optimization stuff so please mail me if you know more so that it can be shared with all of us. I hope you will enjoy this series of tutorials.

Using The Correct Syntax

I know for a fact that there are quite some beginning game coders in the PSP homebrew scene who haven't code much in C or C++ and never did game programming. They however just dive in and see how far the come. These people would benefit from some of the pointers given here so if you are such a person, read on.

When coding you often do not want to use integer because you may need more precision.

```
someVariable = 1.1
```

In this line of code a double is assigned to someVariable. If the someVariable is of the type float, it gets assigned but there is something not going entirely as we want. The PSP is optimized for floating point operations. Doubles are emulated by software so using them can be quite slow. So how should be assign it?

```
someVariable = 1.1f
```

When using float variables, always use the 'f' behind the value; it will be a lot faster if you do. As a side note, using math functions like sin, cos and sqrt should also add the 'f' like: sinf, cosf and sqrtf. We will show in the 4th tutorial that we can create our own very quick math functions but for now this will suffice. A lot of you guys now probably are thinking: "This is just too easy, will the whole tutorial be this way?" Well to be honest, this series of tutorials will be going more deeply in coding with every section. So just read through them, maybe you will learn something.

By Reference VS Pointers

Always try to use by reference instead of pointers. A little example:

```
int lifepoints_;
void addLifepointsPtr(const int* lifepoints) { lifepoints_ += *lifepoints; }
void addLifepointsRef(const int& lifepoints) { lifepoints_ += lifepoints; }
```

These functions are basically the same, however the reference function looks cleaner, there is no need to check if it is NULL and compilers sometimes have difficulty with pointers and less with references which CAN result in faster machine code.

Passing by reference

Instead of passing classes or other structures by value, one should pass them by reference. When you pass a class by value it is copied (copy constructor) and you do not have that with referencing. Although there is a small penalty because you have to dereference the object in the actual code of the function you passed it to, passing by reference can increase the speed by factor 25 to 10000, depending on the complexity of the class or structure. You should however not use it on the integers and smaller objects because the dereferencing in the functions costs more.

Variable declaration

It is good practise to always declare the variables at the moment you need them. In C you often declare it at the beginning of a function. C++ does not require this so we can declare it only when we need. Some related tips:

- If you use a variable only in an IF-statement, declare it inside the IF-statement. This way it is only declared when needed.
- The same counts for the switch function. If you do not use the variable for all cases, declare it only inside the case you will use it with.
- If you use a variable in a loop and it stays constant, declare it outside the loop, if it changes each loop declare it inside the loop.
- The same counts for a while loop, declare it outside if constant otherwise declare it inside.

If we only declare it when we want to use it we will reduce the declaration cost when it is done for nothing. This will speed up your code although it may not always be noticeable.

Initialize VS Assignment

If you initialize a variable instead of defining it and then initializing it, you only use the copy constructor instead of the constructor and assignment operator. An example:

```
// Init
classExample example = something; // only the copy constructor

// Defining and initializing
classExample example;
example = something;
```

As you can see the upper code is more readable, it is less code and it is faster! What else do you want? The speed increase however is small.

Prefix VS Postfix

Often you want to increment or decrement an object, take for example the integer. You can increment it with the operator ++ and decrement it with the -- operator. These operators can be used both prefix as postfix. Why two different ways? Well the functionality is not quite the same of both ways. The postfix operator adds one up but returns the old value. When you create such an operator you often use the prefix operator in that function anyway so stick with the prefix, it increments by one and returns the new object. It saves a copy call. Although this tip has more effect on complex objects, it also has (very little) effect when using integers.

+= or +

Using integers and the like, using the operator will be faster but as soon as you use more complex classes with costly constructors, operator= (+= for example) will be faster. Speed increases with complex objects of 25% are not uncommon comparing these two approaches.

Outroduction

These optimization tips are useful, they often also create smaller code and more readable code. Using these tips will benefit your project. The next tutorials will give some more complex or bizar tips and we will also cover PSP specific optimizations.