Introduction

Hi everybody,

We have seen some nice basic tips to get optimized and readable code. This tutorial will give even more optimization tips but they may be more difficult than the tips in the first tutorial. Hope they will be useful

Inline functions

Although inlining is one the easiest optimization possible in C++, it is discussed in this tutorial. Inlining can improve the execution speed dramatically but it also produced bigger files. Bigger file sizes also reduce speed sometimes, so always monitor if the inline function increase the speed or reduces the speed.

Small methods should be inlined and larger methods not. Like getting and setting function can be inlined because they generate not so much extra code and would speed up the execution time. Too much use of the inlining can cause a heavy increase of the file size which increases the execution time because of lower cache hit rates. Singleton methods can be inlined to get better speeds and will not produce any bigger code.

Example of an inline function:

```
inline int add(int x, int y) {
   return x+y;
}
```

Using the inline naming is not always necessary, some compilers decide for themselves if a function can be inlined and then do so.

Looping loops

Using loop functions is very easy and create nice, clean and readable code. There is however a bit of an overhead. When you unroll loops in function it will increase the speed. The code however gets a little bit larger and less readable. Speed increases of factor 2 or 3 can be possible.

Temporary objects

We have seen in the previous tutorial that initializing can increase speed over first defining and then initializing. This part will go into that even further. When returning an object you can do it as follows (pseudo code):

```
x getAlteredX(int y) {
```

```
    x tmp = someVariable + y;
    return x;
}
```

This results in a temporary object, we can do this better (pseudo code):

```
x getAlteredX(int y) {
    return (x(someVariable + y));
}
```

Now we have only one line and it is faster. This code enables the compiler to omit creating a temporary object, which saves us some function calls. This optimization has not much (almost nothing) effect on simple variable types but can increase the speed with the more complex objects. Debugging gets slightly more difficult though on intermediate results.

Added note: Passing the object by reference in the function enables to store directly to that variable, also avoiding the temporary creation.

Outroduction

These were some optimization tips, which are a bit more difficult and the next tutorial will be even more difficult and probably out of the scope of most projects but the fourth tutorial, on using the VFPU will be very interesting.