Estas optimizaciones son fácilmente aplicables a código existente y en algunos casos aceleran notablemente la velocidad con que nuestros programas se ejecutan. Recuerden sin embargo lo siguiente: el código más rápido es aquel que no es llamado.

Usa listas de inicialización

Siempre usa listas de inicialización en constructores, por ejemplo usa:

```
TMyClass :: TMyClass (const
{
}
```

en lugar de

```
TMyClass :: TMyClass (const
{
    m_Data = data ;
}
```

Sin listas de inicialización, el contructor por defecto es llamado detrás de la escena en lugar del contructor de la clase.

Optimización para ciclos

Cuando sea posible, usa ciclos que vayan contando hacia abajo, por ejemplo usa:

```
for ( i = n
```

en lugar de:

```
for ( i = 0
```

Es más rápido cuando se compara contra cero que contra cualquier otro valor. Nota también que:

```
++ i
```

es más rápido que:

```
i ++
```

Usa 'int'

Siempre usa el tipo de dato int en lugar de char o short cuando sea posible. int es siempre el tipo de dato nativo de la máquina

Haz las funciones locales estáticas (static)

Siempre declara las funciones locales como estáticas, usa static, por ejemplo:

```
static void foo ()
```

Esto quiere decir que la función no será visible a funciones fuera del archivo .cpp donde se usa, y algunos compiladores C++ pueden tomar ventaja de esta optimización.

Optimiza las sentencias if

Factor del salto. Por ejemplo usa:

```
bar ();
if ( condition )
{
    undoBar ();
    foo ();
```

```
}
```

en lugar de:

```
if ( condition )
{
    foo ();
}
else
{
    bar ();
}
```

Usa tu buen juicio para ver si la operación undo es más rápida!

Optimiza las sentencias switch

Pon los casos más comunes al inicio

Evita las operaciones excesivas

La suma es más rápida que la multiplicación y la multiplicación es más rápida que la división.

Inicializar en la declaración

Cuando sea posible inicializa las variables cuando las declaras, por ejemplo:

```
TMyClass myClass = data ;
```

es más rápido que:

```
TMyClass myClass ;
myClass = data ;
```

Declarar y luego inicializar invoca el contructor por defecto y luego hacer una signación son 2 operaciones. Inicializar en la declaración invoca sólo el contructor copia.

Paso por referencia

Siempre trata de pasar clases por referencia en lugar de hacerlo por valor, por ejemplo usa:

```
void foo ( TMyClass &
```

en lugar de:

```
void foo ( TMyClass myClass )
```

Uso de operadores

Usa:

```
myClass += value ;
```

en lugar de:

```
myClass = myClass +
```

La primera versión es mejor que la segunda porque se evita la necesidad de crear un objeto temporal.

Has las funciones pequeñas de tipo inline

Las funciones pequeñas deberían ser declaradas inline para mejorar el rendimiento, por ejemplo:

```
inline void foo ()
```

Cuando se trata de funciones largas, no se debe declararlas "inline"

Usa clases sin nombre

Cuando sea posible usa clases sin nombre, por ejemplo:

```
foo ( TMyClass (
```

es más rápido que:

```
TMyClass myClass ( "abc" );
foo ( myClass );
```

porque en el primer caso, el parámetro y la clase comparten memoria