
Subject: Re: BufferPainter::Clear() optimization
Posted by [Tom1](#) on Tue, 28 Apr 2020 09:17:16 GMT
[View Forum Message](#) <> [Reply to Message](#)

Hi,

Yes, CPU is likely the major player here. I took the liberty to modify TimingInspector to get finer granularity for timing using usecs(). The modified testcase can be found below. I get the following results on my Core i7 + Windows 10 professional x64. Now we can focus on the best round 'min:' to better avoid other tasks' effect. As you can see memsetd on MSBT19x64 is quite amazing performer.

MSBT19x64, Intel Core i7:

```
TIMING memsetd      : 1.45 s - 1.45 ms ( 1.45 s / 1000 ), min: 1.15 ms, max: 5.25 ms,  
nesting: 0 - 1000  
TIMING Fill        : 3.73 s - 3.73 ms ( 3.73 s / 1000 ), min: 3.25 ms, max: 9.92 ms, nesting: 0 -  
1000
```

CLANGx64, Intel Core i7:

```
TIMING memsetd      : 3.85 s - 3.85 ms ( 3.85 s / 1000 ), min: 3.35 ms, max: 10.36 ms,  
nesting: 0 - 1000  
TIMING Fill        : 3.87 s - 3.87 ms ( 3.87 s / 1000 ), min: 3.38 ms, max: 11.33 ms, nesting: 0 -  
1000
```

I guess that in my larger program the optimizations did not work this well as the Fill would have performed at around 5 ms level for this size of a buffer.

Anyway here's the modified benchmark.

```
#include <CtrlLib/CtrlLib.h>  
  
using namespace Upp;  
  
class UTimingInspector {  
protected:  
    static bool active;  
  
    const char *name;  
    int     call_count;  
    int64   total_time;  
    int64   min_time;  
    int64   max_time;  
    int     max_nesting;  
    int     all_count;  
    StaticMutex mutex;
```

```

public:
UTimingInspector(const char *name = NULL); // Not String !!!
~UTimingInspector();

void Add(dword time, int nesting);

String Dump();

class Routine {
public:
Routine(UTimingInspector& stat, int& nesting)
: nesting(nesting), stat(stat) {
start_time = usecs();
nesting++;
}

~Routine() {
nesting--;
stat.Add(start_time, nesting);
}

protected:
int64 start_time;
int& nesting;
UTimingInspector& stat;
};

static void Activate(bool b) { active = b; }

bool UTimingInspector::active = true;

static UTimingInspector s_zero; // time of Start / End without actual body to measure

UTimingInspector::UTimingInspector(const char *_name) {
name = _name ? _name : "";
all_count = call_count = max_nesting = min_time = max_time = total_time = 0;
static bool init;
if(!init) {
#if defined(PLATFORM_WIN32) && !defined(PLATFORM_WINCE)
timeBeginPeriod(1);
#endif
init = true;
}
}

UTimingInspector::~UTimingInspector() {
if(this == &s_zero) return;

```

```

Mutex::Lock __(mutex);
StdLog() << Dump() << "\r\n";
}

void UTimingInspector::Add(dword time, int nesting)
{
    time = usecs() - time;
    Mutex::Lock __(mutex);
    if(!active) return;
    all_count++;
    if(nesting > max_nesting)
        max_nesting = nesting;
    if(nesting == 0) {
        total_time += time;
        if(call_count++ == 0)
            min_time = max_time = time;
        else {
            if(time < min_time)
                min_time = time;
            if(time > max_time)
                max_time = time;
        }
    }
}

String UTimingInspector::Dump() {
    Mutex::Lock __(mutex);
    String s = Sprintf("TIMING %-15s: ", name);
    if(call_count == 0)
        return s + "No active hit";
    ONCELOCK {
        int w = GetTickCount();
        while(GetTickCount() - w < 200) { // measure profiling overhead
            thread_local int nesting = 0;
            UTimingInspector::Routine __(s_zero, nesting);
        }
    }
    double tm = max(0.0, double(total_time) / call_count / 1000000 -
                    double(s_zero.total_time) / s_zero.call_count / 1000000);
    return s
        + timeFormat(tm * call_count)
        + " - " + timeFormat(tm)
        + " (" + timeFormat((double)total_time / 1000000) + " / "
        + Sprintf("%d ", call_count)
        + ", min: " + timeFormat((double)min_time / 1000000)
        + ", max: " + timeFormat((double)max_time / 1000000)
        + Sprintf(", nesting: %d - %d", max_nesting, all_count);
}

```

```

#define RUTIMING(x) \
static UTimingInspector COMBINE(sTmStat, __LINE__)(x); \
static thread_local int COMBINE(sTmStatNesting, __LINE__); \
UTimingInspector::Routine COMBINE(sTmStatR, __LINE__)(COMBINE(sTmStat, __LINE__), \
COMBINE(sTmStatNesting, __LINE__))

```

```

GUI_APP_MAIN
{
Color c = Red();

int len = 4000 * 2000;

Buffer<RGBA> b(len);

for(int i = 0; i < 1000; i++) {
{
RUTIMING("Fill");
Fill(b, c, len);
}
{
RUTIMING("memsetd");
memsetd(b, *(dword*)&(c), len);
}
}
}

```

Best regards,

Tom
