MEETRAPPORT ACCES TIMES DIFFERENT RGB STRUCTS

NAMEN EN DATUM

Jos Bijlenga en Robert Bezem
21-4-15

DOEL

Uit vinden of een struct met exact 32 bits sneller is dan een struct met 24 bits i.v.m. caching.

HYPOTHESE

De struct met 32 bit heeft snellere acces times.

WERKWIJZE

50x uitgevoerd:
1 miljard keer het rode element in de rgb struct veranderen.
Hiervan wordt het gemiddelde berekend voor 1 miljard keer het rode element aanpassen
Ook wordt het in zowel debug als release mode getest

Pseudo code:

```
100 keer:
    Start timer
    1000000000 keer:
        Schrijf een waarde naar een element in de array
    Stop timer
    Bereken verschil en stop in een array

    Het zelfde voor de andere struct
Bereken de gemiddelde waarde in de array's
```

RESULTATEN

DEBUG:

Struct	**Average Time**
R,G,B	4230
R,G,B,a	8391

RELEASE:

Struct	**Average Time**
R,G,B	2440
R,G,B,a	5019

CONCLUSIE

Een struct met alleen de RGB waardes (24 bit) is sneller dan eentje met ook een alpha channel (32 bit) ondanks onze hypohese. In de caching wordt dit dus al geoptimaliseerd;

EVALUATIE

We zijn er dus achter gekomen dat het sneller is om maar 3 elementen in de struct te hebben. Wij zullen dus ook alleen de rgb waardes gaan opslaan.

CODE

```
#include <Windows.h>
#include <iostream>
#include "basetimer.h"
#include <vector>

#define timesPerArray 1000*1000*1000
#define timesTotal 50
#define sizeOfArray 10000

struct argbColor{
	argbColor(
	unsigned char a = 0, unsigned char r = 0,
	unsigned char g = 0, unsigned char b = 0)
	:a(a), r(r), g(g), b(b){}

	unsigned char a;
	unsigned char r;
	unsigned char g;
	unsigned char b;
};

struct nRGB {
	nRGB(
	unsigned char r = 0, unsigned char g = 0,
	unsigned char b = 0)
	:r(r), g(g), b(b){
	}

	unsigned char r;
	unsigned char g;
	unsigned char b;
};
```

```
int main()
{
	argbColor *a = new argbColor[sizeOfArray];
	nRGB *b = new nRGB[sizeOfArray];

	BaseTimer bt;

	std::vector<unsigned long long> ta, tb;

	for (int j = 0; j < timesTotal; j++){
		bt.start();

		int i = 0;
		for (; i < timesPerArray; i++){
			a[i%sizeOfArray].r = i % 256;
		}
		bt.stop();
		unsigned long long testa = bt.elapsedMilliSeconds();
		bt.reset();

		bt.start();
		i = 0;

		for (; i < timesPerArray; i++){
		b[i%sizeOfArray].r = i % 256;
		}
		bt.stop();
		unsigned long long testb = bt.elapsedMilliSeconds();

		ta.push_back(testa);
		tb.push_back(testb);
		std::cout << '.';
	}
	std::cout << "\n";
	unsigned long long sumA = 0;
	for each (unsigned long long var in ta)
	{
		sumA += var;
	}

	unsigned long long sumB = 0;
	for each (unsigned long long var in tb)
	{
		sumB += var;
	}

	std::cout << "Result a: " << sumA / ta.size() << std::endl;

	std::cout << "Result b: " << sumB / tb.size() << std::endl;

	(void)getchar();
	delete a;
	delete b;
	return 0;
}
```