# Implementatieplan - RGBImage

## Namen en datum

Auteurs: Stephan Vivie en Mathijs van Bremen

Klas: V2B

Datum: 15-04-2016

## Doel

Het doel van deze implementatie is om een storage container te maken waarin de pixels van afbeeldingen worden opgeslagen. We proberen een container gebruiken die de beste performance behaalt.

## Methoden

Wij hebben bekeken welke containers in C++ veel worden gebruikt en welke snel zijn. Omdat afbeeldingen een vaste grootte hebben is er geen dynamisch schaalbare container nodig. Voor elke methode wordt dus tijdens het aanmaken van de container het geheugen gereserveerd. De containers die veel worden gebruik zijn C style arrays en std::vector. Wij hebben deze containers op twee verschillende manieren met elkaar vergeleken, namelijk een 1-dimensionale container en een 2-dimensionale container. In een 1-dimensionale container is er maar 1 rij van pixels, een 2-dimensionale container heeft in een container een extra containers met daarin de RGB waardes, hierdoor kun je eenvoudig de pixels m.b.v. X en Y coordinaten benaderen.

#### RGBImageArray

De eerste methode is een pre-allocated C style array.

#### RGBImageVector

De tweede methode is een pre-allocated std::vector.

#### RGBImage2DArray

De derde methode is een 2-dimensionale pre-allocated C style array.

#### RGBImage2DVector

De vierde methode is een 2-dimensionale pre-allocated std::vector.

## Keuze

De keuze is gemaakt voor de tweede methode, de RGBImageVector. We hebben de methodes en standaard implementatie getest en met d.m.v. benchmarks met elkaar vergeleken. Deze benchmarks zijn in bijlage 1 te vinden. Uit de benchmarks is gebleken dat de methodes qua performance weinig verschillen. Het blijkt dat de RGBImageVector het snelst een afbeelding kan laden en weinig verschilt qua performance verschilt met de orginele implementatie. Ook is RGBImageVector de methode die het minst geheugen verbruikt. Daarnaast heeft een std::vector al standaard veel methodes die gebruikt kunnen worden eenvoudig extra functionaliteiten toe te voegen.

## Implementatie

## Evaluatie

## Bijlage: Benchmarks

### Benchmark 1: Afbeelding laden

Gemeten: aantal microseconden dat het inladen van dezelfde afbeelding duurt (loadImage).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **ImageArray** | **ImageVector** | **Image2DArray** | **Image2DVector** | **Image** |
| 1 | 16497 | 14489 | 17435 | 21059 | 15824 |
| 2 | 15555 | 13925 | 14204 | 21488 | 15830 |
| 3 | 15920 | 13892 | 14172 | 20641 | 15610 |
| 4 | 15693 | 14106 | 14505 | 20692 | 16363 |
| 5 | 15470 | 15134 | 14133 | 20747 | 15391 |
| **totaal** | 79135 | 71546 | 74449 | 104627 | 79018 |
| **gemiddeld** | 15827 | 14309,2 | 14889,8 | 20925,4 | 15803,6 |

### Benchmark 2: Geheugen gebruik

Gemeten: aantal kilobyte private geheugen het process verbruikt.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **ImageArray** | **ImageVector** | **Image2DArray** | **Image2DVector** | **Image** |
|  | 1540 | 1448 | 3684 | 1792 | 1536 |

### Benchmark 3: Ophalen willekeurige pixels

Gemeten: aantal microseconden het gemiddeld duurt om een pixel op te halen (100.000 operaties om zo een betrouwbaar resultaat te krijgen).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| methode | **ImageArray** | **ImageVector** | **Image2DArray** | **Image2DVector** | **Image** |
| getPixel(int) | 0.04267 | 0.07128 | 0.09726 | 0.14216 | 0.04352 |
| getPixel(x,y) | 0.08237 | 0.11169 | 0.06029 | 0.11137 | 0.08165 |

### Benchmark 4: Aanpassen willekeurige pixels

Gemeten: aantal microseconden het gemiddeld duurt om een pixel op te slaan (100.000 operaties om zo een betrouwbaar resultaat te krijgen).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| methode | **ImageArray** | **ImageVector** | **Image2DArray** | **Image2DVector** | **Image** |
| setPixel(int) | 0.11394 | 0.20611 | 0.16736 | 0.21291 | 0.1118 |
| setPixel(x,y) | 0.15524 | 0.17844 | 0.13244 | 0.17893 | 0.15497 |

