Implementatieplan Image Shell

Nico van Bentum, Gianluca Piccardo 27-02-2019

Doel

Het maken van een klasse die pixel-informatie van een afbeelding op kan slaan.

Methoden

We hebben verschillende methodes bekeken met betrekking tot de container. Het is de vraag of we een fixed-size container gebruiken of dynamisch. Bij een dynamische container kan je denken aan een std::vector. Deze allokeert elementen standaard op de heap, maar de gebruiksfunctie's hebben redelijk veel performance overhead in vergelijking met andere containers. Een c-style (of std) array is fixed size, dus hiervoor moeten extra variabelen bijgehouden worden over de grootte. Dit is lastig aangezien afbeeldingen ingeladen worden in run-time, en er dus geen templates of andere compile-time opties gebruikt kunnen worden. Een andere manier is simpelweg een pointer bijhouden naar een stuk geallokeerd geheugen waar we de pixels bijhouden. Zo is het makkelijker om de size dynamisch te houden en is er weinig tot geen overhead qua snelheid en memory usage.

Keuze

Op basis van online onderzoek ("*C++ Data Structures and Algorithms Cheat Sheet*", z.d.) , is onze keuze gevallen op de pointer data. Om zo min mogelijk overhead te creeëren is er niet gekozen voor een smart pointer maar allokeren en deallokeren we de pixel data handmatig. Ook omdat er functies aanwezig zijn die de data aanspreken in row-major volgorde is het handig om een één dimensionale opslag te gebruiken.

Implementatie

Aangezien de student klasses IntensityImage en RGBImage vrijwel identiek zijn wordt alleen de RGBImage beschreven. In de klasse houden we drie extra variabelen bij voor de RGB data en de grootte van de afbeelding. De andere code snippet is een deel code van de copy constructor, de rest van de implementatie gebruikt ongeveer dezelfde code. De h en w variabelen houden de grootte van de afbeelding bij. We gebruiken de delete[] operator om de data te wissen en new[] om nieuwe opslag te initialiseren. Pixel data overkopieëren is simpelweg assignment.

```
class RGBImageStudent : public RGBImage {
public:
        RGB * data = nullptr;
        unsigned int w = 0;
        unsigned int h = 0;
```

```
RGBImageStudent::RGBImageStudent(const RGBImageStudent &other)
        : RGBImage(other.getWidth(), other.getHeight())
{
        h = other.h;
        w = other.w;
        delete[] data;
        data = new RGB[w * h];
        data = other.data;
```

Evaluatie

Om onze implementatie te testen doen we twee metingen, één voor snelheid en één voor geheugengebruik. In een paar test runs kwamen we erachter dat in de standaard implementatie een memory leak en/of bug zit waardoor het niet mogelijk is om de test vaak te runnen, anders crasht de applicatie of gooit het een exception.

Bronnen:

C++ Data Structures and Algorithms Cheat Sheet. (z.d.). Geraadpleegd op 28 februari 2019, van https://github.com/gibsjose/cpp-cheat-sheet/blob/master/Data%20Structures%20and%20Algorithms.md