Implementatieplan voor de efficiëntste container

Florian Humblot - 1720570
Vera Schoonderwoerd - 1721202

Datum: 11 februari 2019

Inhoudsopgave

Doel

Het doel van de implementatie is om een container te bouwen voor RGB en Grayscale images met een zo snel mogelijke access time.

Methoden

Je geeft hier aan welke methoden er zijn, wat de verschillende tussen de methodes zijn.

- Arrays
    - Pros
        - Access complexiteit is O(1) waardoor snelheid voor getPixel vrij hoog is
        - De array mag fixed-size zijn, aangezien de width en height van het plaatje ofwel bij de constructor bekend wordt gemaakt of bij de set functie. Daardoor kunnen we een array of de heap maken en in een keer de goeie size hebben.
    - Cons
        - Fixed size – uitbreiding is costly
        - Insertion en deletion time is relatief traag (O(n))
- Vectors
    - Pros
        - Access complexity is O(1)
        - Mocht de size veranderen hoeft niet de hele vector opnieuw gegenereerd te worden.
    - Cons
        - Overhead
- Linked lists
    - Pros
        - Neemt weinig ruimte in
        - Uitbreidbaar zonder de hele list de herbouwen
    - Cons
        - Access time is hoog
- 1-D array/vector
    - Pros
        - Snellere array access doordat er minder handelingen zijn
    - Cons
        - Moeilijker om over na te denken/te implementeren dan een 2 dimensionale array of vector
        - Handmatige memory management voor de 1-D array.

Keuze

Je geeft een onderbouwing over waarom een bepaalde methode is gekozen, en/of waarom bepaalde settings zijn gebruikt.

Wij hebben uiteindelijk gekozen voor een 1 dimensionale array, dit omdat wij zo min mogelijk overhead willen hebben voor het accessen van array elementen. Door middel van raw-pointers te gebruiken kunnen wij direct het geheugen aanspreken en op die manier de latency te verminderen.

Implementatie

Je geeft aan hoe deze keuze wordt geimplementeerd in de code

Wij gaan de RGBImageStudent classe implementeren met een 1-D array van pointers naar RGB pixels. In de array worden alle pixels sequentieel opgeslagen, alsof alle rijen van een matrix van pixels aan elkaar worden geplakt.

De IntensityImageStudent wordt op dezelfde wijze geimplementeerd alleen dan met een Intensity pixel pointer als onderliggend datatype in plaats van een RGB pixel pointer.

Dit is handig omdat wij op deze manier makkelijk de pixel kunnen bepalen die wij op moeten halen/bewerken en het is een eenvoudige manier om zowel x/y coördinaten als een index te gebruiken.

Om de goeie pixel positie te bepalen kan "I" direct gebruikt worden na een check om te controleren of het opgevraagde element wel daadwerkelijk in de bounds past. Voor de x/y access is het een kwestie om de "x" te vermenigvuldigen met de breedte van het plaatje en daar de "y" bij op te tellen om zo de index te krijgen.

Evaluatie

Je geeft aan welke experimenten er gedaan zullen worden om de implementatie te testen en te 'bewijzen' dat de implementatie daadwerkelijk correct werkt. Dit geeft direct informatie over de meetrapporten die er zullen worden gemaakt.

Om de implementatie van de containers te controleren gaan wij een test uitvoeren waarbij wordt gekeken of de rest van het programma hetzelfde resultaat levert op basis van de RGBImageStudent en de IntensityImageStudent klasse als met de default klasse.

Verder willen wij ook de snelheid van de applicatie testen door vanaf de DLL een plaatje uit de testset 1000x te processen en de snelheid en geheugenverbruik ervan te monitoren en vervolgens dezelfde test uitvoeren maar dan met onze implementatie.