Implementatieplan Practicum Week 1

Chiel Douwes

March 6, 2020

1 Doel

De implementatie van de Image-klasse moet de volgende dingen bieden:

- Een dynamische opslag van het type van de pixels die van groote veranderd kan worden
- Methodes om te indexen in de pixel array en om deze te lezen en schrijven
- Een interface om deze functionaliteiten aan te bieden aan de meegeleverde RGBImageStudent en IntensityImageStudent classes

Er hoeft in de implementatie geen garanties geleverd te worden over onder meer:

- De run-time complexiteit
- De access time constraints
- Plaatsing van de opslag van het object
- De memory safety
- De memory contiguity
- De waarde of het runtime gedrag van uninitialized values/pixels
- De staat van het object na een out of memory condition tijdens een set operatie
- De hoeveelheid geheuhengebruik in het object of tijdens het aanroepen van een van de functies

- De exception-safety van de functies in het object, behalve onder bepaalde precondities

De eisen die wel gesteld worden zijn onder meer:

- Get-operaties moeten geen size effects hebben in de pixel data
- De pixel data dat gezet in na het opzetten van het object moet in redelijke mate onveranderlijk zijn tijdens de lifetime van het object

2 Methoden

Er zijn enkele methoden die gebruikt kunnen worden:

- Fixed-size array: De simpelste implementatie, met een vaste hoogte en breedte
- Sparse array: Een array die stukken kan bevatten die niet gealloceerd zijn
- Dynamic array: Een array die naar vraag groter of kleiner kan worden gemaakt

Voor de fixed-size array kan de storage op verschillende plekken geplaatst worden afhankelijk van de grootte van het array en de geheugen opties in de target architectuur; zo kan het bijvoorbeeld op de stack worden geplaatst in het geval van grootes van minder dan een aantal kilobytes, of op de heap voor grotere images. Dit geld ook voor de sparse array en dynamic array, maar deze zijn iets flexibeler in het plaatsen van het geheugen en in de allocatie van geheugen tijdens de set en get methodes.

3 Keuze

De keuze wordt gemaakt om een fixed-size array te implementeren, mits dit de snelste array access oplevert en ook een correcte implementatie gemakkelijk te verifiëren is. Naast deze voordelen is er in het fixed-size array ook geen mogelijkheid tot momory errors of allocation failures tijdens de set-methodes sinds deze nooit extra geheugen zal hoeven te alloceren.

4 Implementatie

Het fixed-size array is geïmplementeerd door een type te maken met vaste hoogte en breedte parameters, zodat de access instructies in compile time geoptimalizeerd kunnen worden doormiddel van SIMD. Omdat de implementatie ook functionaliteit moet leveren om de grootte van de image te veranderen is er ook de optie om de klasse een dynamische grootte te geven door de parameters naar 0 te zetten. Het type heeft als een van de variabelen de data van de image. Er zal een functie gegeven worden die kan indexen in de array, en deze methode zal gebruikt worden om de get implementatie te maken van de RGB/IntensityImageStudent, maar sinds de methode een reference geeft kan het ook gebruikt worden om de set methode te implementeren.

5 Evaluatie

De experimenten die uitgevoerd kunnen worden zijn onder anderen tests om de correctness van de implementatie te testen, zoals out of bounds checks en andere edge cases, en er kunnen tests gemaakt worden om de performance van de pixel update methodes te testen. Voor de evaluatie van de performance van de image klasse moeten er meerdere patronen getest worden sinds compiler optimalisatie en cacheline grootte een impact kunnen hebben in hoe snel de test wordt uitgevoerd. Om deze reden is het ook een goed idee om meerdere grootes images te testen in combinatie met verschillende access patronen. Om de correctness van de image klasse aan te tonen is het ook mogelijk om een integration test te doen met het image processing programma dat mee geleverd is voor de cursus Vision.