1. Implementatieplan titel

1.1. Namen en datum

Edwin Koek, Jacob Visser 1-6-2015

1.2. Doel

Het doel is om snelle en efficiente edge detection te implementeren.

1.3. Methoden

Sobel operator,prewitt operator en Robert's cross operarator
Deze 3 lijken enorm veel op elkaar aangezien ze alle 3 2 losse kernels gebruiken. Sobel en prewitt zijn gemaakt om de verticale en horizontale edges te vinden. Robert's cross operator zoekt naar de schuine edges. Deze methodes zijn echter inaccuraat en gevoelig voor ruis.

Laplacian of gaussian
Bij deze methode wordt de image eerst gesmoothed en vervolgens wordt een laplacian kernel losgelaten over de image. Deze kernels zorgen ervoor dat er plekken ontstaan waar het intensiteits verschil groot is. Op deze plekken bevinden zich de edges. De gaussian en laplacian kernels kunnen samengevoegd worden tot 1 kernel. Het resultaat hiervan is de volgende kernel :
0 1 0
1 -4 1
0 1 0
Het gebruik van deze kernel zorgt voor een sneller resultaat aangezien de image maar 1x doorlopen hoeft te worden. Deze methode is bij rondingen helaas niet altijd even accuraat.

Canny
Canny is een edge detection methode die die zo nauwkeurig mogelijk probeert te zijn. Deze methode kost echter wel veel tijd om toe te passen. Bij canny edge detection wordt een image eerst gesmoothed met een gaussian filter. Vervolgens wordt doormiddel van een sobel operator de "kracht" van de edges bepaald. Na deze stap worden de richtingen van de edges bepaald. Vervolgens worden de edges van het resultaat duidelijker gemaakt met behulp van thresholding.

1.4. Keuze

Voor edge detection hebben we voor de laplacian of gaussian gekozen. De sobel operator zou het snelste resultaat geven en is ook het makkelijkst te implementeren, maar deze is erg inaccuraat. De canny methode is juist weer te traag en lastig om te implementeren. Canny geeft goede resultaten, maar is de moeite niet waard.
De laplacian of gaussian is een vrij eenvoudige methode die bruikbare resultaten geeft. Met deze reden hebben we voor laplacian of gaussian als implementatie gekozen. Als kernel hebben we gekozen voor de samengevoegde gaussian/laplacian kernel die beschreven staat in het vorige paragraaf.

1.5. Implementatie

Nieuwe image en kernel initialiseren

```
IntensityImage * StudentPreProcessing::stepEdgeDetection(const IntensityImage &image) const {
    IntensityImageStudent* IM = new IntensityImageStudent(image.getWidth(), image.getHeight());
    // Initialise base kernel
    std::array<int, 3 * 3> kernel{ {
            0, 1, 0,
            1,-4, 1,
            0, 1, 0,
        } };
    int imageWidth = image.getWidth();
    int sum = 0;
    int temp = 0;
    //Width of the base 3x3 kernel
    int kernelWidth = 3;
    //Radius of the kernel when taking blockWidth in account
    int kernelRadius = 4;
    //Width of the blocks, all elements in the base kernel represent a single block.
    //For example if blockWidth is 3 the kernel wil look like this:
    //      0, 0, 0, 1, 1, 1, 0, 0, 0,
    //      0, 0, 0, 1, 1, 1, 0, 0, 0,
    //      0, 0, 0, 1, 1, 1, 0, 0, 0,
    //      1, 1, 1, -4, -4, -4, 1, 1, 1,
    //      1, 1, 1, -4, -4, -4, 1, 1, 1,
    //      1, 1, 1, -4, -4, -4, 1, 1, 1,
    //      0, 0, 0, 1, 1, 1, 0, 0, 0,
    //      0, 0, 0, 1, 1, 1, 0, 0, 0,
    //      0, 0, 0, 1, 1, 1, 0, 0, 0
    int blockWidth = 3;
```

Door de image heen lopen

```
    int maxX = image.getWidth() - kernelRadius;
    int maxY = image.getHeight() - kernelRadius;
    // Loop through all pixels except the outer rows where the kernel doesnt fit.
    for (int y = kernelRadius; y < maxY; ++y){
        for (int x = kernelRadius; x < maxX; ++x){
            sum = 0;
            // Loop thought the kernel
            for (int ky = 0; ky < kernelWidth; ++ky){
                for (int kx = 0; kx < kernelWidth; ++kx){
                    temp = 0;
                    // Skip a block if the element in the kernel has the value 0
                    if (kernel[ky * kernelWidth + kx] == 0) {
                        continue;
                    }
                    // Loop through block of pixels that are to be evaluated
                    for (int by = 0; by < blockWidth; ++by){
                        for (int bx = 0; bx < blockWidth; ++bx){
                            // Add an evaluated pixels value to the sum of the block
                            temp += image.getPixel((x - kernelRadius + bx + (kx * blockWidth)) +
                                (imageWidth * (y - kernelRadius + by + (ky * blockWidth))));
                        }
                    }
                    // add the blocks weighed value to the total sum, no normalisation is required for laplacian
                    sum += temp * kernel[ky * kernelWidth + kx];
                }
            }
            // making sure the intensity values are within limits
            if (sum > 255){
                sum = 255;
            }
            if (sum < 0){
                sum = 0;
            }
            // Set the pixel according to the sum
            IM->setPixel(x + imageWidth * y, sum);
        }
    }
    return IM;
}
```

Threshholding toepassen

```
IntensityImage * StudentPreProcessing::stepThresholding(const IntensityImage &image) const {
    IntensityImageStudent* IM = new IntensityImageStudent(image.getWidth(), image.getHeight());
    int imageSize = image.getWidth() * image.getHeight();
    for (int i = 0; i < imageSize; ++i){
        Intensity p = image.getPixel(i);
        if (p > 220){
            IM->setPixel(i, 0);
        }
        else{
            IM->setPixel(i, 255);
        }
    }
    return IM;
}
```

1.6. Evaluatie

Met experimenten zal getest worden welke kernel grote de meest duidelijk edges geeft. Daarnaast wordt ook de thresholding getest. Bij deze tests zal worden gekeken welk thresholding niveau het minste ruis en de beste edges geeft.