Verslag Augmented Reality

Dylan Rakiman (1709808)

# Introductie

Dit is mijn verslag over mijn project voor Augmented Reality. In dit verslag leg ik uit hoe mijn applicatie eruit zou moeten zien, hoe ik te werk ben gegaan, hoe de laatste versie van mijn code werkt, wat ik heb gedaan om tot dit punt te komen, wat de problemen zijn met mijn huidige implementatie en wat er nog gedaan zou kunnen worden om die problemen tegen te gaan.

# Bedoelde werking van applicatie

Allereerst leg ik kort uit hoe mijn applicatie had moeten werken. Mijn applicatie zou camerabeelden van een webcam inlezen, hieruit bepalen of er een rechthoek in het camerabeeld te detecteren is, de posities van de hoeken van de rechthoek bepalen om daar vervolgens een projectiematrix mee te berekenen. Dit projectiematrix wordt vervolgens op een 3D-model toegepast om de illusie te creëren dat dat 3D-model zich ‘op’ het voorheen genoemde rechthoek bevindt.

# Gebruikte programmatuur

Ik heb gekozen om Unity 2018.3.10f1 te gebruiken voor het bewerkstelligen van deze applicatie. Het idee was dat ik met Unity razendsnel van library zou kunnen wisselen, mocht ik dat nodig achten. Verder gebruikte ik git om een repository bij te houden (<https://github.com/Dylrak/ARQR>) en gebruikte ik Trello om bij te houden welke taken ik moest voltooien en wat mijn voortgang met deze taken was (<https://trello.com/b/7ZMuaaZl/augmented-reality>). Tevens gebruik ik Visual Studio 2017 en Visual Studio Code om C#-scripts aan te passen.

# Werking van laatste versie

De laatste versie van mijn code (<https://github.com/Dylrak/ARQR/releases/tag/v1.3>) werkt als volgt: ik heb drie scripts aangemaakt die op verschillende plekken in het project gedraaid worden, namelijk RecognizeCorners, CalculateProjectionMatrix en Transform3DModel.

Transform3DModel werkt zoals het klinkt; het is een script dat op een 3D model toegepast kan worden om deze te verplaatsen, roteren of te schalen aan de hand van een 4x4 matrix. Om in één regel een 4x4 matrix op een 3D model toe te passen heb ik de klasse TransformExtensions gebruikt (<https://forum.unity.com/threads/how-to-assign-matrix4x4-to-transform.121966>). Deze extraheert de schaal, rotatie en positie uit het 4x4 matrix en past deze toe op het 3D model waar dit script aan vastgebonden is.

CalculateProjectionMatrix is een script dat een 4x4 matrix berekent aan de hand van vier pixelposities die de hoeken van een vierkant of rechthoek moeten voorstellen. Het script is gebonden aan het EventSystem. Deze klasse slaat een Tuple op met een quarternion en een 4x4 matrix. Het 4x4 matrix wordt elke update-call uitgerekend aan de hand van het quarternion. Aangezien dit een reeds opgelost probleem is, is dit met wiskundige formules op te lossen. Deze wiskundige formules (en een scriptje dat aantoont dat dit werkt) zijn hier te vinden: <https://gamedev.stackexchange.com/questions/138016/detect-transform-from-detection-of-qr-code>

RecognizeCorners is een script dat aan twee RawImages gebonden kan worden (in dit geval, WebCamImage en ProcessedImage) om daar de rauwe en verwerkte camerabeelden op te tonen en om berekeningen te doen over die camerabeelden. WebCamImage toont direct de WebCamTexture erwijl voor ProcessedImage de WebCamTexture geconverteerd wordt naar een OpenCV Mat, hier bewerkingen mee gedaan worden (zoals canny image filtering) en dit bewerkte beeld vervolgens terug naar een Unity Texture geconverteerd wordt voor debugging (omdat dit zwart-witbeeld normaliter niet getoond hoeft te worden). De huidige implementatie van RecognizeCorners herkent geen hoeken; waarom dit zo is, wordt nader uitgelegd.

# Werkproces en resultaten gedurende project

In de eerste paar weken heb ik onderzoek gedaan naar de werking van Unity en de libraries die ik kon gebruiken. Nadat ik uitgevogeld had hoe ik webcambeelden in Unity kon tonen, heb ik een null-versie gemaakt van mijn drie scripts zodat ik hier later op verder kon bouwen. Deze null-versie van mijn code is hier te vinden: <https://github.com/Dylrak/ARQR/releases/tag/v1.0>

Hierna heb ik Transform3DModel en CalculateProjectionMatrix geïmplementeerd aan de hand van de links die ik hierboven geplaatst heb. Deze twee klassen zijn naast menig refactoring grotendeels hetzelfde gebleven. Nadat ik deze twee scripts gemaakt had, heb ik geprobeerd om de library ZXing, normaliter bedoeld om QR-codes uit te lezen, bruikbaar te maken binnen mijn project (<https://github.com/Dylrak/ARQR/releases/tag/v1.1>). Ik had de library werkbaar gekregen en kon eindelijk code schrijven ervoor toen ik op een gegeven moment stuitte op het volgende probleem: ZXing geeft alleen de posities van de drie vierkanten in een QR-code die de oriëntatie bepalen. Deze library geeft dus niet vier hoeken en heeft geen manier om dit te doen zonder aanpassingen aan de library zelf te doen. Zonder de positie van de vierde hoek kan ik ook geen projectiematrix uitrekenen en dus moest ik het gebruik van ZXing loslaten.

Hierna had ik ervoor gekozen om AForge werkende te krijgen met Unity, omdat deze een op het eerste oog vrij simpel te implementeren functie had genaamd QuadrilateralFinder (<https://github.com/Dylrak/ARQR/releases/tag/v1.2>). Om Aforge werkende te krijgen met Unity moest ik echter System.Drawing aanzetten, zodat ik een bitmap naar AForge kon sturen. System.Drawing, zoals geïmplementeerd in C#, heeft echter talloze overlappende definities in combinatie met Unity en niemand heeft een concrete manier om System.Drawing in Unity werkende te krijgen; dit zou ik dus handmatig voor elke overlappende definitie moeten aanpassen.

Aangezien dit onbegonnen werk is heb ik geprobeerd om AForge for Unity te gebruiken. Dit is een versie van AForge die compatibel is met Unity en dus geen gezeur met System.Drawing creëert. Het probleem met deze library was echter dat deze niet de klasse AForge.Imaging bevatte, waar de QuadrilateralFinder in zit. Omdat deze library de Quadrilateralfinder niet ondersteunde, heb ik ervoor gekozen om het gebruik van AForge in zijn geheel in de steek te laten.

Nadat ik een tijdje gekeken had naar de mogelijkheden met Vuforia in Unity, kwam ik tot de conclusie dat het voor mij het handigste zou zijn om OpenCV in combinatie met Unity te gebruiken (<https://github.com/Dylrak/ARQR/releases/tag/v1.3>). De beste gratis versie van OpenCV voor Unity is OpenCV plus Unity (<https://assetstore.unity.com/packages/tools/integration/opencv-plus-unity-85928>). Dit is een conversie van OpenCVSharp, wat op zijn beurt weer een conversie van OpenCV is voor C#. Deze library wordt echter niet meer ondersteund en bevat ook lang niet alle features die OpenCV bevat. Deze problemen werden alsmaar duidelijker naarmate de deadline naderde. Ik heb tevergeefs geprobeerd om de functie FindContours werkende te krijgen met mijn project. Nadat ik hier geen succes in had gevonden, ben ik gaan kijken naar een implementatie van HoughLinesP; OpenCV plus Unity ondersteunt dit echter niet en maakt alleen gebruik van GeneralizedHough. Aangezien ik met tijdsnood zat en dus niet genoeg tijd had om onderzoek te doen naar het maken van een lijn-specifiek Hough-algoritme met behulp van GeneralisedHough, heb ik besloten om mijn project werkend te maken tot het punt waarop een GeneralisedHough-implementatie toegevoegd zou kunnen worden.

# Problemen huidige implementatie

De huidige implementatie herkent geen hoeken en om deze reden heb ik de code van CalculateProjectionMatrix ook eruit gecomment. Omdat er geen matrix berekend kan worden, wordt het 3D-model ook niet getransformeerd. RecognizeCorners had in de eerste instantie een aparte functie die door CalculateProjectionMatrix aangeroepen werd om een set van vier hoeken terug te geven; deze is eruit gesloopt omdat ik dit op een andere manier wilde implementeren maar aangezien er geen hoeken berekend kunnen worden bestaat die implementatie ook nog niet.

# Oplossingen voor in de toekomst

Een manier om dit huidige project werkende te krijgen is om GeneralisedHough in te stellen zodat het lijnen probeert te herkennen, zoals OpenCV’s HoughLinesP dat doet. Dit vereist echter een grote hoeveelheid onderzoek in de werking van gegeneraliseerde Hough-transformaties.

Een andere manier om deze applicatie tot stand te brengen is om het project over te zetten naar de originele OpenCV voor C++. Dit zou problemen met missende implementaties door geconverteerde libraries voorkomen. OpenCV voor C++ maakt het mogelijk om camerabeelden in te lezen, deze te tonen op het beeld, een 3D-model daar overheen te plakken en om de hoeken van een vierkant of rechthoek in de camerabeelden te herkennen en daar berekeningen mee te doen. Ook zou het gebruik van HoughLinesP de implementatie van dit project ten zeerste vergemakkelijken. Als ik dit project opnieuw zou doen met de kennis die ik nu heb, zou ik alles van de grond af aan in OpenCV voor C++ willen bouwen.