# Projectopgave: 3D Modelling and Image Based Rendering

3 maart 2025

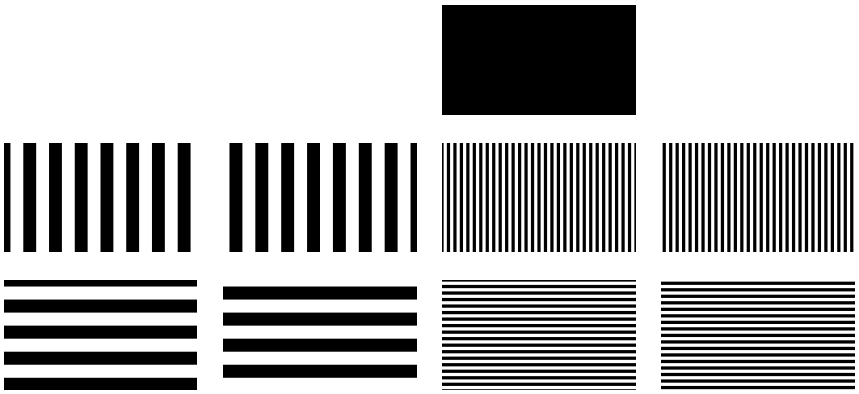
Het doel van deze oefening is het maken van een digitale versie van een fysiek object op basis van camerabeelden en projecties. Hiervoor zal je onder andere gebruik maken van een dense reconstructie van een scène, met behulp van *structured light* scanning. Structured light scanning is een actieve techniek (zie les 4: Actieve 3D reconstructie), d.w.z. dat er actief informatie aan de scène wordt toegevoegd om meer correspondenties te bekomen. Dit in tegenstelling tot passieve technieken, zoals bvb. stereo (zie les 3: passieve dichte stereo beeldinterpolatie en diepte reconstructie). Het principe van structured light scanning is eenvoudig. Er worden een aantal patronen (Gray Codes[1], sinuspatronen, ...) geprojecteerd op een scène. Na de projectie heeft ieder belicht deel van het tafereel een unieke code, waarmee correspondenties gevonden kunnen worden tussen camerabeelden. In deze opgave zal je structured light implementeren m.b.v. graycodes om een 3D puntenwolk te maken.

Verder zal je ook een passieve methode voor het genereren van virtuele camerastandpunten implementeren, namelijk het plane-sweeping algoritme.

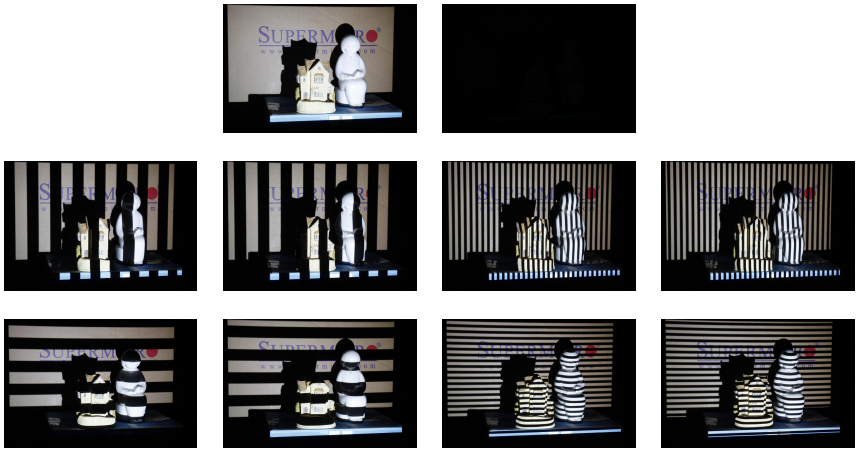
Tot slot mag je deze technieken ook gebruiken om de gescande scène op een light field display te tonen.

De oefening gebruikt beelden van twee camerastandpunten als invoer. Voor elke standpunt zijn er een reeks afbeeldingen waarbij in ieder beeld een ander patroon geprojecteerd werd op het tafereel. In totaal zijn er per standpunt 42 afbeeldingen. Elke set kan opgedeeld worden in 21 groepen van 2 afbeeldingen, waarbij in de 2e afbeelding binnen een groep het inverse patroon van de eerste afbeelding geprojecteerd werd. De eerste groep (afbeeldingen "00.jpg" en "01.jpg") bevat opnames waarbij het tafereel volledig belicht werd door de projector (een volledig wit patroon, afbeelding "00.jpg") en volledig onbelicht (een volledig zwart patroon, afbeelding "01.jpg") was. De 10 daaropvolgende groepen (afbeeldingen "02.jpg" t.e.m. "21.jpg") bevatten afbeeldingen waarin de graycodes voor een horizontale identificatie geprojecteerd werden (de horizontale pixelpositie in het geprojecteerde patroon). Tot slot bevatten de laatste 10 groepen (afbeeldingen "22.jpg" t.e.m. "41.jpg") de graycodes voor de verticale identificatie (de verticale pixelpositie in het geprojecteerde patroon). Je zal zien dat er voor elke afbeelding een inverse is. Een subset van de patronen vind je in Figuur 1. Figuur 2 toont de overeenkomstige beelden voor 1 camerastandpunt. Naast deze afbeeldingen zijn er ook afbeeldingen van een schaakbordpatroon. Deze kan je gebruiken om de intrinsieke parameters van de camera te berekenen.

[1] `https://en.wikipedia.org/wiki/Gray_code`



Figuur 1: Voorbeeld van de geprojecteerde patronen.



Figuur 2: Voorbeeld invoerafbeeldingen van twee cameras gebruik makend van graycodes. De geprojecteerde patronen komen overeen met deze in Figuur 1.

# 1 Opgave

Het doel van de opgave is om de invoerafbeeldingen van Figuur 2 te gebruiken om correspondenties te bekomen tussen de twee cameras. De correspondenties worden dan gebruikt om de extrinsieke calibratie-informatie te berekenen (zie Les 2: Camera Geometrie). Dit omvat de cameraoriëntatie en het projectiecentrum. Vervolgens kan de extrinsieke calibratie gebruikt worden om de 2D-correspondenties te herprojecteren in 3D. Daarna implementeer je het planesweep algoritme. Tot slot toon je het gescande object op een lichtveldscherm.

In dit project maak je gebruik van OpenCV en Python om de verschillende stappen te implementeren. Verder zal je Open3D gebruiken om bepaalde resultaten te visualiseren (bvb. puntenwolken en camerastandpunten).

## Stap 1: Camerakalibratie

In de eerste stap van dit project moet je de intrinsieke parameters en de distortieparameters van de camera berekenen. Hiervoor kan je gebruik maken van de afbeeldingen in de map *chess*. De afbeeldingen bevatten checkerboard patronen die als referentieobject dienen. Binnen dit project zijn enkel het aantal hoekpunten binnen het patroon belangrijk ($7 \times 9$). De breedte van ieder veld is 35mm.

1. Detecteer de patronen in de map *chess*. Tip: bij afbeeldingen met een hoge resolutie kan de detectie lang duren. Je kan dan de afbeeldingen verkleinen en daarop de detectie doen. Vergeet niet de uitvoer terug te schalen naar het oorspronkelijke formaat van de afbeelding.

2. Gebruik de gedetecteerde patronen om de intrinsieke calibratie matrix $K$ en de lensdistortie van de camera te berekenen. Let op, de sequenties van beide standpunten zijn berekend met dezelfde camera, dus er moet maar één matrix $K$ berekend worden. OpenCV heeft een aantal tutorials over camerakalibratie die als startpunt kunnen dienen.

3. Verwijder distortie van de invoerbeelden op een efficiënte manier

4. Buiten de matrix $K$ en de lensdistortie geeft OpenCV's *calibrateCamera* ook rotatie- en translatievectoren terug die de extrinsieke parameters (de pose) van de camera ten opzichte van het patroon voorstellen. Bereken voor iedere afbeelding in *chess* - waarin je een patroon detecteert - de pose. Sla deze poses, samen met $K$ op in een tekstbestand en visualiseer de camerastandpunten met behulp van Open3D[2]. Open3D's *LineSet* klasse heeft functionaliteit om een visualisatie van een camera te maken (een wireframe piramide). Doe deze visualisatie in een ander project dan je OpenCV project: afhankelijk van de manier waarop de GUI module van OpenCV gebuilt is conflicteert deze met de GUI van Open3D.

## Stap 2: Structured Light

1. Genereer, gebruik makend van de meegeleverde "GrayCodeEncoder" klasse, een reeks van graycode patronen die geprojecteerd kunnen worden op de scène om de invoerafbeeldingen te genereren. Bij een resolutie van $1920 \times 1080$ en een bitdiepte van 10 zijn 20 patronen nodig voor zowel de horizontale als de verticale richting. Voor elk patroon met een even index, is het daaropvolgende patroon de inverse van dit patroon. De projectie van een patroon en zijn inverse geven ons later een manier om eenvoudig te bepalen welke pixels in het camerabeeld belicht zijn en welke niet. Elk patroon, samen met diens inverse, laat ons toe om 1 bit te bepalen voor de identificatie van een punt in de scène. Omdat we zowel voor de horizontale en verticale richting 10 paar patronen hebben, kunnen we in totaal aan $2^{10} \times 2^{10}$ punten in de ruimte een unieke code toekennen. Bestudeer de generatie van de graycode patronen goed, zodat je straks weet hoe je ze kan decoden. Het maken van een eigen dataset is een taak voor een latere stap in dit project. Waarom zou je graycodes gebruiken in plaats van een gewone binaire voorstelling?

[2] `http://www.open3d.org/docs/release/python_example/visualization/index.html`

2. Door de gegenereerde patronen te projecteren zorgen we ervoor dat ieder punt in de ruimte, dat belicht kan worden door de projector, een unieke identifier krijgt. Na het capteren van de beelden kun je ieder paar van eenzelfde patroon (de normale en de inverse) gebruiken om 1 bit van de identifier te bepalen. Hiervoor kan je de verschilafbeelding nemen van de twee en met behulp van een threshold bepalen of de bit voor een pixel 0, 1 of onbepaald is (een te hoge onzekerheid heeft, typisch aan de overgang van een patroon). In het geval dat een bit onbepaald is kan je stellen dat de gehele identifier van die pixel ongeldig is. Je kan gebruik maken van de full-white en full-black projecties om te bepalen welke pixels in het projectiegebied liggen en welke niet. Voer deze decodeerstap uit voor beide camerastandpunten. Kies zelf of je de verwerking doet op de voor lensdistortie gecorrigeerde beelden. Afhankelijk van je keuze zal je in latere functies andere parameters moeten meegeven.

3. Normaal heb je nu twee lijsten (of andere datastructuur) van geïdentificeerde pixels: één voor ieder camerastandpunt. Zoek op basis van deze identificaties correspondenties tussen beide standpunten. Gebruik OpenCV's *drawMatches* om te controleren of de correspondenties plausibel zijn.
(Tip 1: er kunnen enkele foutieve matches zijn, die filteren we er in de volgende stap uit.
Tip 2: teken slechts een subset van de matches uit zodat je de lijnen nog kan onderscheiden.)

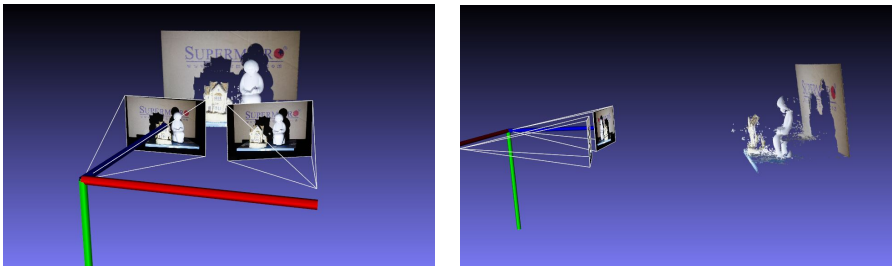
## Stap 3: Reconstructie Puntenwolk

1. Gebruik de bekomen correspondenties om de essentiële matrix te berekenen.

2. Bereken de rotatie- en translatierichting aan de hand van de essentiële matrix. Je hebt nu een gok voor de pose van de tweede camera. De eerste camera staat in de oorsprong. (Tip: *recoverPose*). Visualizeer beide poses met Open3D.

3. De kalibratieinformatie, samen met de 2D-puntcorrespondenties, kunnen nu gebruikt worden om de 3D-coördinaten te bekomen. Doe dit m.b.v. de ingebouwde triangularisatiefunctie *triangulatePoints*. Los nu ook manueel het triangularisatieprobleem op door voor ieder punt een stelsel in de vorm van $Ax = B$ op te lossen. Het stelsel kan opgebouwd worden met behulp van de projectiematrices ($K \cdot [R|t]$) van camera 1 en 2 en de 2D-correspondenties. Los het stelsel op met SVD (Tip: *cv2.SVD)*. Meer info kan je lezen in *uitlegTriangulation.pdf* en *mvg_triangulation.pdf*.

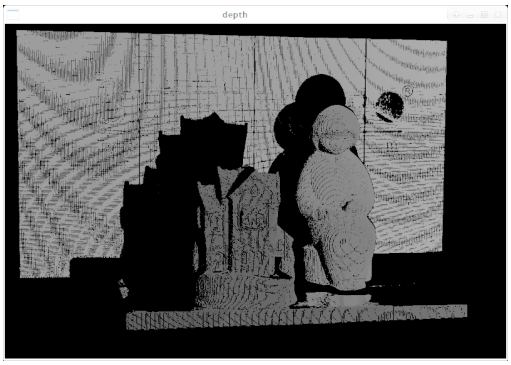
4. Visualiseer de puntenwolk en de berekende poses met Open3D. Doe dit in een apart programma/script om mogelijke conflicten tussen de Open3D en OpenCV GUI's te vermijden. Een voorbeeld is weergegeven in Figuur 3. Merk op: de conventie van het coördinatensysteem is vaak anders tussen de verschillende toolkits (zie slides Les 2). Hier wordt een typisch computervisie assenstelsel gebruikt: de kijkrichting komt overeen met de positieve Z-as. X en Y geven de beeldcoördinaten naar rechts en naar onder. Open3D gebruikt echter dezelfde conventie als OpenCV.



Figuur 3: Resultaten van een gereconstrueerde puntenwolk en de camera poses.

## Stap 4: Plane-Sweep

1. Maak en visualiseer een dieptemap. Itereer hiervoor over de pixels in het beeld en bereken de afstand tot het getriangulariseerde 3D-punt. Let op, om de dieptemap te kunnen visualiseren zullen de dieptewaarden moeten gemapped worden tussen 0 en 255. Een voorbeeld is weergegeven in Figuur 4. Zorg dat je kan switchen naar een nieuw standpunt d.m.v. het toetsenbord. Om de tussenliggende camerastandpunten te bepalen kan je de rotatiematrix omzetten naar een quaternion om vervolgens tussen de rotaties te interpoleren[3]. Bij een juiste implementatie kan je tijdens dit switchen de parallax in de scène waarnemen.




Figuur 4: Dieptemap.

2. Implementeer plane-sweeping (zie Les 3: consensus based rendering). Het berekenen van de tussenliggende camerastandpunten is dezelfde als bij de depth maps en kan dus herbruikt worden.

   (a) De dieptelagen moeten gecentreerd worden rond de scène. Bereken het centrum van de puntenwolk en gebruik de projectiematrix van de virtuele camera om de diepte van het centrum te achterhalen. Die diepte kan gebruikt worden om het aantal dieptelagen te beperken.

[3] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.transform.Rotation.html`

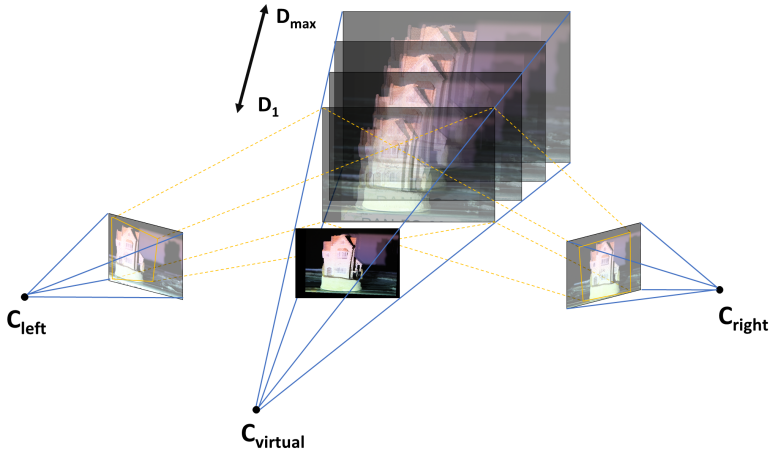
(b) Gegeven een virtuele camera en een diepte, bereken het 3D dieptevlak $D_i$ waarop geherprojecteerd moet worden. Deprojecteer de 2D hoekpunten van het image plane van de virtuele camera naar 3D om de 3D hoekpunten van het vlak te bekomen.

(c) Herprojecteer de verschillende invoercameras op het dieptevlak $D_i$. Bepaal de homografie tussen het image plane van de afbeelding en het 3D vlak (tip: *cv2.findHomography()*). Om dit te berekenen projecteer eerst de 3D hoekpunten van het vlak op het respectievelijke camerabeeld. Gebruik de vier hoekpunten van de afbeelding en de geprojecteerde 2D punten van het 3D vlak (aangegeven in geel in Figuur 5) om de homografie op te bouwen. Pas de homografie toe op de afbeelding (tip: *cv2.warpPerspective()*).

(d) Itereer over de verschillende dieptelagen en bereken de error tussen de verschillende herprojecties. Bereken voor iedere pixel de euclidische norm van het verschil tussen de herprojectie van de linkse camera en de herprojectie van de rechtse camera (tip: *cv2.absdiff()*, *cv2.compare()*). De pixels waarvoor de error functie minimaal is, worden gebruikt als finaal resultaat in de geïnterpoleerde afbeelding. Figuur 6 toont een aantal resultaten.

3. Breidt nu deze implementatie uit:

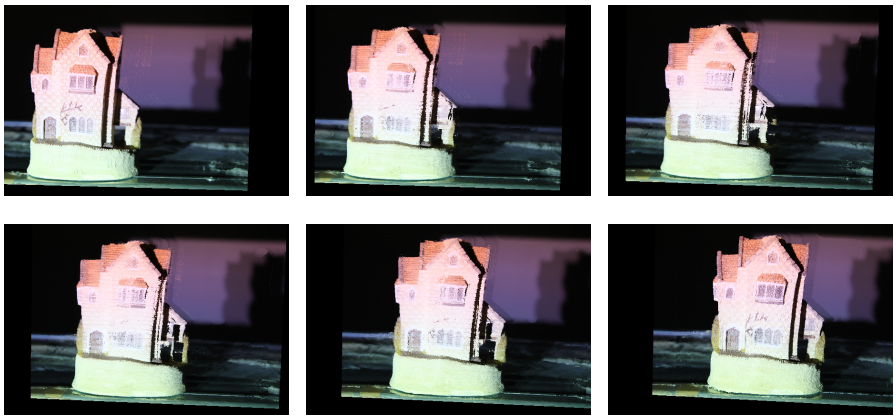
   Gebruik in de error metriek een neighborhood rond iedere pixel (tip: *cv2.filter2D()*). Probeer ook andere kostfuncties dan de Euclidische norm.Laat ook een niet-uniforme verdeling van de dieptelagen toe. Gebruik de puntenwolk uit de *structured light* reconstructie om te bepalen waar dieptelagen een nuttige bijdrage kunnen leveren.




Figuur 5: Overzicht algoritme plane sweeping.

## Stap 5: Eigen Dataset(s) Capteren

Maak zelf eens een dataset met een interessant object naar keuze. Denk wel goed na over de materiaalkeuze (sommige materialen zijn niet geschikt voor



Figuur 6: Geïnterpoleerde virtuele standpunten na plane sweep.

Structured Light). Wij stellen een projector, camera en kalibratiepatronen ter beschikking. Maak eerst een afspraak per mail[4] zodat we de opstelling kunnen klaarzetten. De gebruikte camera zal anders zijn dan de camera waarmee de invoerdata is opgenomen: je moet de camera dus ook zelf kalibreren.

**Lees ook eerst goed de mogelijke extra's en volgende stappen van het project: sommige extra's vereisen meer invoerdata dan de door ons aangeleverde data.**

## Stap 6: Reconstructie Camera-Camera-Projector

(a) Breidt de reconstructie uit zodat deze werkt voor 3 of meer camerastandpunten: je zal de reconstructie, matching en pose estimation moeten aanpassen om met dit groter aantal beelden overweg te kunnen. Zorg ervoor dat punten die slechts in twee standpunten zichtbaar zijn ook gereconstrueerd worden. Tip: probeer de puntenwolk niet in 1x op te bouwen, maar doe dit in stappen. Een functie zoals OpenCV's *SolvePnP*, of diens afgeleiden, zal ook nuttig blijken.

2. De projector werd tot nu toe enkel gebruikt om een dicht rooster van correspondenties tussen de twee camerabeelden te genereren. Je kan er echter evenveel informatie uit halen als een camera zelf. Inderdaad: veel structured light systemen gebruiken slechts 1 camera en een projector en gebruiken de correspondenties tussen die twee. Pas je oplossing aan zodat de projector gebruikt wordt als een camera. Je zal dus ook de projector moeten kalibreren.

[4] lode.jorissen@uhasselt.be

## Stap 7: Light Field Displays (Extra - verplicht voor groepen van 3)

Nu je een virtuele versie van een echt object kan maken kan je deze data gebruiken om de scène op een lichtveldscherm te tonen. Maak, op basis van de bekomen resultaten, een dataset die geschikt is voor het Looking Glass Portret lichtveldscherm[5]. Het scherm laat verschillende vormen van invoerdata toe: je kan een dieptemap combineren met een RGB afbeelding, je kan een grote (50-100) hoeveelheid afbeeldingen - gegenereerd door een camera die over een lijn beweegt - meegeven, of je kan Unity/Unreal/Blender/... gebruiken in combinatie met een plugin om rechtstreeks een 3D model of puntenwolk te renderen naar het scherm, etc.

Probeer deze drie verschillende methodes uit. Kies zelf of je de puntenwolk of het plane-sweep algoritme gebruikt voor het renderen bij de tweede methode.

**Dien bij het verslag een filmpje van het scherm in voor iedere methode.**

## Extra's

Naast de extra in Stap 7 kan je ook de volgende extra's implementeren.

1. Implementeer structured light ook met sinuspatronen i.p.v. graycodes (zie [ZHHZ06]). Maak een vergelijking tussen beide methodes. Welke sterktes/zwaktes zie je in beide?

2. Gebruik Iterative Closest Point om twee puntenwolken met elkaar uit te lijnen. Maak hiervoor twee datasets van eenzelfde scène zodat je beelden van 4 verschillende camerastandpunten hebt en dat de projector op twee verschillende plaatsen heeft gestaan. Vervolgens kan je met elke dataset een puntenwolk maken die je uitlijnt met ICP. Opgelet: zorg ervoor dat er voldoende overlap is tussen beide datasets. Maak bijvoorbeeld twee datasets van de voorkant van het object, maar zorg toch voor een verplaatsing van de camera en projector.

3. Implementeer een eigen versie van één of enkele OpenCV functies. Mogelijke opties zijn findEssentialMatrix, recoverPose, ...

4. Implementeer voor de projector een eigen kalibratiefunctie. Je kan gebruik maken van de puntenwolk, bekomen door twee cameras, om een volledige kalibratie te doen waarbij de intrinsieke en extrinsieke matrix samen geschat worden. Je mag er hierbij vanuit gaan dat de lensvervorming van de projector beperkt is.

5. Maak een mesh van je puntenwolk.

6. Optimaliseer puntenwolk en poses met behulp van bundle adjustment. Een goed startpunt hiervoor is de Ceres solver.

7. ...

[5] https://lookingglassfactory.com/looking-glass-portrait

# 2 Modaliteiten

- Het practicum wordt gemaakt in groepjes van twee. Je bent vrij in de partnerkeuze. Geef via Blackboard jullie groepsindeling door en doe dit voor 11/3/2025.
- Indien een opdeling in groepen van uitsluitend 2 personen niet mogelijk is, kan er ook een groep van 3 personen gevormd worden. Deze groep dient dan ook stap 7 verplicht te implementeren, alsook een extra uit de sectie "Extra's" in overleg met het onderwijsteam.
- De opgave telt mee voor een derde van de examenpunten voor het vak. Zonder tegenindicaties, worden partners in een groep gelijk gekwoteerd.
- Het practicum wordt onafhankelijk van andere groepjes gemaakt. Code van anderen gebruiken, of je code door anderen laten gebruiken, mag enkel indien dat op voorhand en door beide partijen gerapporteerd wordt aan de begeleider, en als dat gecompenseerd wordt door ander werk (in overleg met de begeleider). Je kan best op voorhand niet met andere groepjes discussiëren hoe je de problemen in dit practicum zal oplossen. Onregelmatigheden worden op dezelfde manier gesanctioneerd als spieken op een examen.
- Rapportering: Je programmacode wordt met beknopt verslag, code en resultaten ingediend via de Blackboard opdracht. Tussenliggende resultaten en resultaten van de reconstructie moet ook toegevoegd worden aan het ingezonden resultaat. Het verslag beschrijft in het kort (richtlijn: 3 pagina's), wat er in het zip bestand staat, voldoende tussenresultaten voor alle stapjes, en bevat alle andere informatie die relevant kan zijn voor de kwotering van het practicum, zoals bijvoorbeeld: onderdelen van de opgave die je niet geïmplementeerd hebt, eventuele samenwerking met anderen (en hoe die gecompenseerd werd), extra mogelijkheden, etc...
- De deadline van het project valt in het weekend voor het examen: 8/6/2025 om 23:59. Begin er dus zeker op tijd aan zodat je er niet tijdens de examenperiode aan moet werken.
- Demo: op de dag van het examen (13/6/2025) zal je je practicum demonstreren. Tijdens de demonstratie kan er gevraagd worden je programma te laten lopen op afbeeldingen van een door ons gegeven voorbeeldscène. Er kunnen vragen gesteld worden over hoe je bepaalde deeltaken hebt opgelost, er kan gepeild worden naar je inzicht in de materie, en je krijgt de kans om eventuele extra's te laten zien. Je kan uiteraard ook toelichten waar je eventuele problemen mee had. De demonstratie zal ca. 20 minuten per groep duren, en is bepalend voor de kwotering.
- Kwotering: een minimale, werkende implementatie van de volledige opgave levert een basisscore van 7 op 10 op. Deze score wordt verminderd of vermeerderd naargelang de kwaliteit van je werk.

# Referenties

[ZHHZ06] Song Zhang, Peisen S. Huang, Peisen S. Huanga, and Song Zhangb. A fast three-step phase-shifting algorithm. 2006.