KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Eindverslag

Team:
Bronze

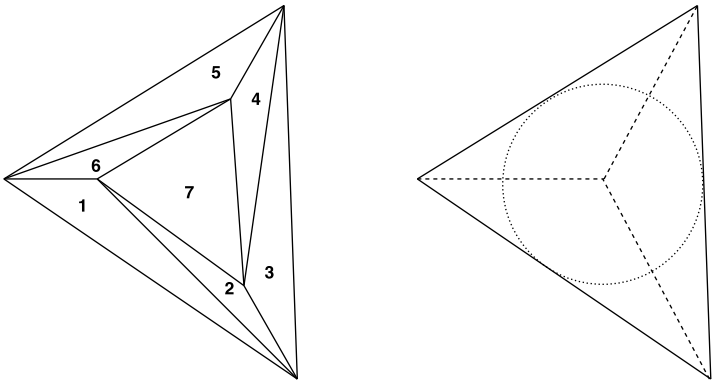
Yana Dimova
(Coördinator)
Thor Galle
(Secretaris)
Valentin Claeys
Ruben Coppens
Maarten Craeynest
Antoon De Cleen

Academiejaar 2016-2017

Samenvatting

In dit verslag staat een oplossing beschreven voor het ontwerp en de implementatie van een virtuele drone met bijbehorende 3D-wereld en artificiële intelligentie autopilot. Het verslag licht het ontwerp, de ontwerpkeuzes en denkprocessen toe. Ook beschrijft het het functioneren en de ervaringen van het team.

Inhoudsopgave



Figuur 1: Links: De 7 deel-driehoeken die samen de originele driehoek vormt. Rechts: Middelpunt van ingeschreven cirkel

1 Inleiding

Dit verslag gaat over de uitbreidingen en aanpassingen van het drone project voor P&O Computerwetenschappen. In elke sectie is bondig uitgelegd wat de grootste veranderingen zijn. Enkel volledig nieuwe delen worden uitgebreid uitgelegd. Om de omvang te beperken zijn delen die reeds in vorige verslagen stonden weggelaten. Het verslag gaat ervan uit dat de lezer deze ook heeft gelezen.

2 Visualisatie

Hier bespreken we de aanpassingen die gemaakt zijn aan de render engine om aan de nieuwe specificaties te voldoen.

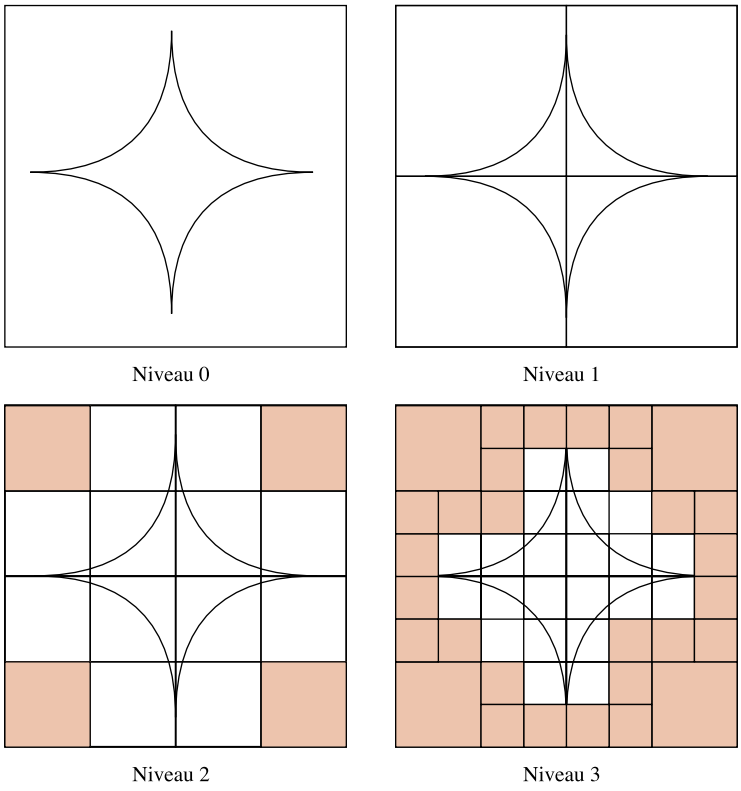
2.1 Saturatie en driehoekranden

Om elke driehoek van een rand te voorzien wordt er een tussenstap gemaakt voor de opengl draw call. Hier wordt elke driehoek omgezet naar 7 anderen. Het gevormde patroon is te zien op figuur 1a. Om de positie van de binnenste driehoek te bepalen leggen we de twee middelpunten van de ingeschreven driehoeken over elkaar (één voorbeeld te zien op figuur 1b). Dit zorgt ervoor dat alle randen even breed zijn. De kleuren van de outline van driehoeken worden bepaald bij het inladen (of genereren) van objecten. Globaal wordt er bijgehouden welke kleuren reeds gebruikt zijn, door het RGB-spectrum af te gaan in een bepaalde volgorde en bij te houden waar de simulatie zit in het spectrum. De binnenste driehoek kan elke mogelijke rgb-kleur hebben (maar het testbed zorgt er wel voor dat target- en obstacle-objecten steeds de juiste HSV-waarde krijgen). Ook kan men een texture inladen voor de binnenste driehoek maar door een verlies in performantie door het gebruiken van textures wordt dit zeer zelden gebruikt.

2.2 Octrees

Octrees worden gebruikt om een compacte, eenvoudig bewerkbare voorstelling van een volume te verkrijgen. Dit wordt gedaan met behulp van assen-uitgelijnde omvattende kubussen (axis-aligned bounding boxes of AABB). Het volume wordt in een eerste stap volledig omgeven door een kubus (box). Vervolgens wordt deze box opgedeeld in 8 nieuwe kubussen. Afhankelijk van de applicatie kan een booleaanse waarde volstaan om voor te stellen dat de kubus wel of niet volledig in het volume zit. Iedere extra laag splitst de vorige kubussen opnieuw op. Een voorbeeld in 2 dimensies kan gevonden worden op figuur 2.

Octrees en AABB worden gebruikt omdat ze door hun eenvoudige, hiërarchische structuur toestaan om enkele bewerkingen efficiënt uit te voeren:



Figuur 2: 2D Quadtree van een stervorm. Meer niveaus verbeteren de approximatie

Ray casting of picking Zoeken naar de eerste (of alle) intersectie(s) van een rechte lijn en een volume. Picking is de uitwerking hiervan waar de muis een object in de 3D wereld selecteert.

Collision detection Botsen van objecten bestaande uit niet reguliere vormen. Voor eenvoudige geometrische volumes zoals de cirkel zijn zijn specifieke algoritmes vaak efficiënter.

Frustum culling Voor het tekenen kan er reeds een eerste schifting gemaakt worden van de te tekenen objecten. Dit is vaak een ruwe, snelle schatting, die schat welke objecten zich buiten het gezichtsveld van de camera bevinden. Omdat één object vaak uit vele driehoeken bestaat is dit veel efficiënter dan dit later driehoek per driehoek te doen in de GPU.

Verder gebruiken we octrees nog om voor te stellen wat de drone nog niet gezien heeft. Dit kan omdat het wegsnijden van het geziene volume het perfect tegengestelde is van frustrum culling.

2.3 Mogelijke uitbreidingen

- De volledige implementatie voor textures, inclusief van grootte veranderen tot de inwendige driehoek.
- Octrees toepassen op de systemen van het testbed.
- Frustum culling implementeren

3 Collision Detection

Dit hoofdstuk bespreekt de opsporing en behandeling van botsingen in de simulatie van het testbed, respectievelijk collision detection en collision handling genoemd. Dit hoofdstuk bespreekt eerst de algemene werking van collision detection. Het duikt daarna in de details en behandelt ook de

geïmplementeerde collision handling. Wel wordt er verwacht dat de lezer het vorige verslag gelezen heeft en zo kennis heeft van wat een WorldObject is.

3.1 Collision detection

Elk object in de simulatie dat moet kunnen botsen heeft een bounding geometry. Dit is een simpele geometrische voorstelling van het werkelijke model dat op elk moment alle hoekpunten van het object omvat. Het testbed gebruikt deze omsluitende geometrieën om de reken-intensieve collision checks tussen objecten wiens omsluitende geometrieën niet snijden over te slaan. Elk object in de simulatie heeft ook een kenmerk "Moved" dat op 'true' wordt ingesteld als het object zich verplaatst of roteert en op 'false' wanneer het testbed heeft gecontroleerd dat deze niet botst met de andere objecten in de simulatie.

De collision detection en handling wordt verzorgd door een CollisionDetector die toegang heeft tot de lijst van alle WorldObjects in de simulatie. Elke keer dat het testbed een simulatie lus uitvoert, wordt het algoritme dat hieronder beschreven is uitgevoerd.

Algorithm 1: Het Collision check algoritme

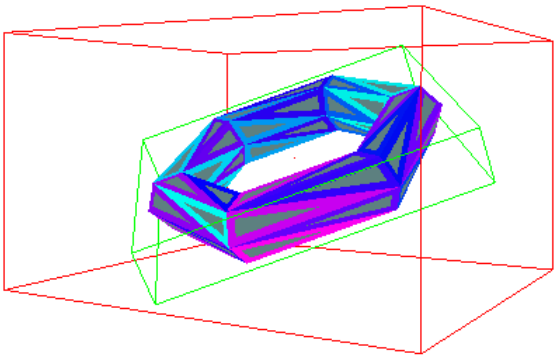
```
collidableObjects = de WorldObjects die gecheckt moeten worden op collision;
aantalIteraties = 0;
collisionInLaatsteIteratie = true;
while collisionInLaatsteIteratie ∧ aantalIteraties <maxIteraties do
    collisionInLaatsteIteratie = false;
    for i = 0; i <collidableObjects.size(); i++ do
        current = collidableObjects.get(i);
        for j = i + 1; j <collidableObjects.size(); j++ do
            other = collidableObjects.get(j);
            if current.moved == false ∧ other.moved() == false then
                continue
            end
            if boundingGeometriesCollide(current,other) then
                if actuallyCollide(current,other) then
                    collisionInLaatsteIteratie = true;
                    handleCollision();
                end
            end
        end
    end
    aantalIteraties++;
end
```

3.2 Bounding geometry

De simulatie ondersteunt twee modellen: een balk en een bol, respectievelijk AABoundingCuboid en BoundingSphere genoemd.

3.2.1 AABoundingCuboid

Alle WorldObjects in de simulatie die moeten kunnen botsen, buiten de drones, hebben een AABoundingCuboid (Axis-Aligned Bounding-Box of AABC). Een AABC is een balk waarvan al zijn zijden parallel liggen t.o.v. de x-, y- of z-hoofdas van de simulatiewereld. Dit zorgt ervoor dat het eenvoudig is om te detecteren of een AABC met een andere AABC of een bol snijdt. Wanneer een WorldObject in de wereld wordt geladen en het geen drone is, krijgt het een balk toegewezen. Deze balk is de kleinst mogelijke balk die uitgelijnd is t.o.v. de hoofdassen en alle hoekpunten van het WorldObject bevat. Deze balk is echter niet de AABC van dat object omdat deze balk niet



Figuur 3: De balk (in het groen) en AABC (in het rood) van een taurusachtig veelvak

alleen mee zal transleren met het object maar ook roteren, zodat de balk steeds alle hoekpunten van het object bevat. Dit schendt de definitie van een AABC omdat het niet gegarandeerd is dat de balk uitgelijnd is t.o.v. de hoofdassen. Om dit op te lossen zal de AABC van een object zich aanpassen wanneer het object (en de balk dus ook) transleert en roteert. De herberekening van de AABC gebruikt de hoekpunten van de balk van het object, zoals geïllustreerd op figuur 3. Dit is efficiënter dan een herberekening van de AABC op basis van de vertices van het WorldObject.

3.2.2 BoundingSphere

Omdat drones in het testbed voorgesteld worden als veelvlakkige representaties van bollen, krijgen drones bollen als bounding geometry. De straal van een BoundingSphere is gelijk aan de afstand van het centrum van de drone tot zijn verste hoekpunt. Deze BoundingSphere moet niet roteren maar transleert wel altijd mee met de drone.

3.2.3 Collision tussen bounding geometries

1. AABC vs AABC: Deze snijden als hun drie intervallen (projecties op de hoofdassen) snijden.
2. BoundingSphere vs BoundingSphere: Deze snijden als de afstand tussen hun centra kleiner is dan de som van hun stralen.
3. AABC vs BoundingSphere: Om te berekenen of deze twee snijden neemt men het punt op de AABC dat het dichtste is bij het centrum van de BoundingSphere. Dit punt kan men berekenen door de dichts bijzijnde waarde in het x-,y- en z- interval van de AABC t.o.v. respectievelijk de x-,y- en z- coördinaat van het centrum van de BoundingSphere te nemen. Als de afstand tussen dit punt en het centrum van de BoundingSphere kleiner is dan de straal van de BoundingSphere, dan snijden deze bounding geometries.

3.3 Feitelijke collision check

Eenmaal dat de CollisionDetecter een collision detecteert tussen de bounding geometries van twee WorldObjects, moet het checken of de modellen van de objecten echt snijden. Ook hier is een onderscheid tussen de drone en de andere WorldObjects.

1. Drone vs Drone: Omdat drones voorgesteld worden als veelvlakkige representaties van bollen, kunnen we hun veelvlakkig model verwaarlozen en berekenen of de afstand tussen hun centra kleiner is dan de som van hun stralen. Als dit waar is, snijden ze.
2. Niet-Drone vs (Niet-Drone of Drone): De modellen van deze twee objecten zijn veelvlakken. Deze kan men ook voorstellen als een maas van driehoeken. Om te zien of de twee modellen snijden moet men dan zien of er een driehoek is van het ene object dat een driehoek van

het andere object snijdt. Dit doet de CollisionDetector door de "Fast Triangle-Triangle Intersection Test" van Tomas Moller[3].

3.4 Collision handling

Elk WorldObject heeft een attribuut genaamd "CollidableType" dat een van de volgende waarden kan hebben: Fixed, Bouncy of Weak. Dit wordt gebruikt om te determineren wat er moet gebeuren wanneer het object botst met een ander object.

1. Fixed: Objecten die Fixed zijn mogen door de collision handling verplaatst worden als ze met een ander Fixed object botsen. In een normale simulatiewereld zijn enkel de obstakels 'fixed'.
2. Bouncy: Deze objecten worden verplaatst door de collision handling als ze botsen behalve als ze tegen een Weak object botsen. De enige objecten die in een normale simulatiewereld 'bouncy' zijn, zijn de drones.
3. Weak: Als een Weak object met een ander niet-Weak object botst zal dit object verwijderd worden uit de simulatielijst. Als twee Weak objecten botsen zal de collision handling maar een van de twee verwijderen. In een normale simulatiewereld zijn enkel de doelobjecten 'weak'.

Dit is geïmplementeerd volgens algoritme 2.

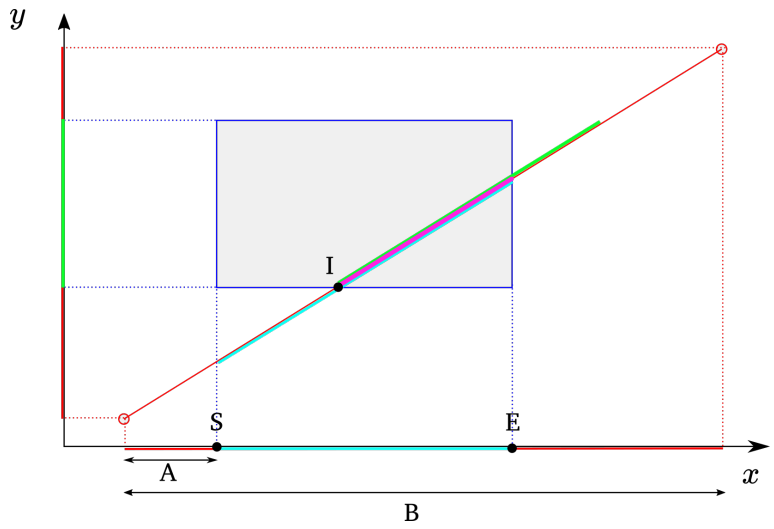
Algorithm 2: Collision handling

```
one = eerste WorldObject dat het algoritme checkt;
two = tweede WorldObject dat het algoritme checkt;
if one is WEAK then
    haal one uit de simulatielijst;
    return;
end
if two is WEAK then
    haal two uit de simulatielijst;
    return;
end
if one is FIXED then
    fixed = one;
    bouncy = two;
else
    fixed = two;
    bouncy = one;
end
repeat
    Verschuif fixed weg van bouncy volgens de rechte die door hun centra gaat.
until actuallyCollide(fixed,bouncy) == false;
```

3.5 Raycast Collisions

Het systeem biedt een functionaliteit waarbij de gebruiker in het beeld kan klikken op een WorldObject om het te selecteren (picking). Zo kan hij de eigenschappen van het object zien en aanpassen in de zijbalk van de GUI. Om dit te bereiken wordt er na een klik in het beeld intern een lijnstuk (ray) gecreëerd startend van de positie van de camera van de kijker en eindigend op de maximaal zichtbare afstand, in de richting waarin de gebruiker klikte.

Het dichtstbijzijnde WorldObject dat snijdt met het lijnstuk wordt geselecteerd. Om dit te bepalen worden collision tests uitgevoerd tussen enerzijds het lijnstuk en anderzijds de bounding geometries van WorldObjects, genoemd raycast collision tests. Deze tests verlopen op verschillende manieren naargelang het type bounding geometry.



Figuur 4: Intersectie test van een lijnstuk met een rechthoek in het 2D vlak.

3.5.1 AABoundingCuboid raycast collision

Het algoritme om te bepalen of een lijnstuk snijdt met een balk is gebaseerd op projecties van het lijnstuk en de balk naar de hoofdassen. Het algoritme zoekt het startpunt en eindpunt van de overlappende delen van de projecties van het lijnstuk en de balk op een enkele as. In het 2D voorbeelddiagram (Figuur 4) zijn deze voor de x-as aangeduid als S en E.

De verhouding tussen enerzijds het startpunt van het lijnstuk tot het startpunt van de overlapping (A) en anderzijds de lengte van het lijnstuk (B) in een projectie bepaalt welk deel van het lijnstuk op die as met de balk overlapt. Het algoritme bepaalt dit deel voor elke as (in het voorbeeld: lichtgroen en lichtblauw voor respectievelijk de x- en y-as). Wanneer een deel van het lijnstuk overlapt op elke as (paars), dan is dat deel van het lijnstuk in het inwendige van de balk en geeft het algoritme het beginpunt ervan terug (het intersectie punt, I).

3.5.2 BoundingSphere raycast collision

Het algoritme om op een intersectie met een BoundingSphere te testen is eenvoudiger. Het beschouwt twee vectoren: de vector die het lijnstuk voorstelt en de vector die van de oorsprong van het lijnstuk naar het centrum van de bol gaat. Het past hierop een bewerking met het vectorproduct toe om de loodrechte afstand te vinden van het centrum van de bol naar het dichtstbijzijnde punt op het lijnstuk. Wanneer die afstand kleiner is dan de straal van de bol, dan is er een intersectie.

3.6 Mogelijke uitbreidingen

- Een systeem gebruiken dat de snijdende delen van de bounding geometries kan berekenen zodat CollisionDetector niet elke face van de twee objectmodellen moet controleren op intersectie.
- Een algoritme zoeken dat gemakkelijk de afstand van een punt tot een face kan berekenen om de uiteindelijke intersectie tussen een drone en een niet-drone object snel te kunnen berekenen

4 AI

De AI, verder autopiloot genoemd, moet zelfstandig in staat zijn om objecten te herkennen, inspecteren en reconstrueren. Uit de reconstructie moet hij ook de drone kunnen besturen zonder ongewenste botsingen.

4.1 Werking

De autopilot is een statemachine. De routeplanner bepaalt aan de hand van de gereconstrueerde wereld waar de drone kan vliegen. Meer uitleg over de dronestates is te vinden in sectie 4.2 , meer over de routeplanner in 4.4. Omdat het mogelijk is dat er willekeurige translaties en rotaties zijn, veroorzaakt door de wind, is er een PID-controller (zie sectie 4.3) ingebouwd. Deze zorgt ervoor dat de drone zichzelf bij elke iteratie bijstuurt en op deze manier steeds de gewilde snelheidsvector heeft.

4.2 Besturing

De besturing is afhankelijk van in welke state de drone zich bevindt. Voor elke state vraagt de drone aan de routeplanner naar waar hij moet vliegen en in elke iteratie wordt deze locatie omgezet in een bepaalde rotatie en thrust om de gewilde snelheidsvector te bekomen. Dit gebeurt aan de hand van een PID-controller om te compenseren voor afwijkingen. De besturing is accurater dankzij de implementatie van de exacte positie van de drone maar is nog steeds gebaseerd op de besturing zoals beschreven in het eindverslag van het eerste semester.

4.2.1 AIStabilizeState

Deze state zorgt ervoor dat de drone zo stabiel mogelijk op dezelfde locatie blijft. Bij initialisatie van de state slaat de drone zijn positie op. Zolang hij zich in deze state bevindt zal hij telkens terugvliegen naar deze locatie.

4.2.2 AIInspectState

Deze state zorgt ervoor dat de drone een object inspecteert. Dit gebeurt door rond het object te vliegen en telkens nieuwe beelden te analyseren om zo een reconstructie te kunnen maken.

4.2.3 AIFlyToState

Deze state zorgt ervoor dat de drone naar een gewenste locatie vliegt. Dit gebeurt aan de hand van de routeplanner, de werking hiervan staat in meer detail uitgewerkt in 4.4.

4.3 PIDController

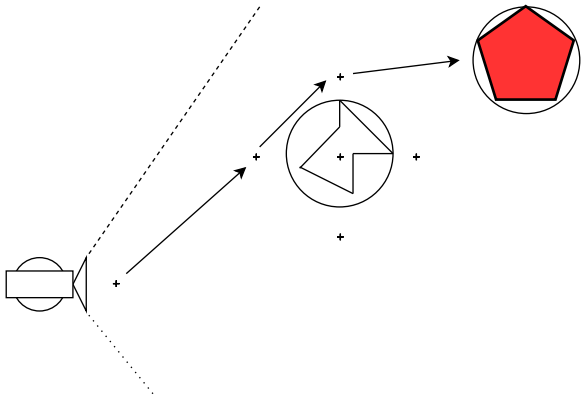
Voor de autopilot een rotatie- of thrustverandering toepast, gaat de informatie over de gewilde snelheidsvector en de effectieve snelheidsvector langs een PIDController. De controller geeft de verandering die nodig is om de drone de juiste snelheidsvector te geven. Gebruik hiervan zorgt ervoor dat de drone ongewenste afwijkingen door wind en onverwachte routeveranderingen door nieuwe informatie met vloeiende bewegingen oplost.

4.4 Routeplanner

Om het pad te vinden dat de drone moet afleggen, maakt hij gebruik van een graaf doorzoek algoritme. De drone geeft zijn huidige en gewenste positie door en de routeplanner zegt naar waar hij moet vliegen. De drone houdt een lijst bij van alle gekende objecten

Elk object dat de drone als obstakel ziet krijgt zes knooppunten rond zich die buiten het oppervlak van de bol liggen. Hierna wordt voor elke twee knooppunten gezocht of ze kunnen verbonden worden zonder een bol te raken. Dit bepaalt of er een boog tussen de knooppunten staat. Eenmaal dat dit voor alle paren gebeurd is, kan er eenvoudigweg een graaf doorzoek algoritme gebruikt worden om het kortste pad naar het doel te vinden. Een visuele voorstelling hiervan kan gevonden worden op figuur 5.

Het gebruikte graaf algoritme is Dijkstra's single source shortest path algoritme[1].



Figuur 5: Voorbeeld van pathfinding. Merk op dat 1 node onder het linker object niet zichtbaar is.

4.5 Driehoek detectie

Om een veelvlak te genereren uit de camerabeelden van de drone, worden de hoekpunten van driehoeken gedetecteerd en doorgegeven per kleur. Dit gaat gemakkelijk omdat de randkleur van elke driehoek uniek is. Het herkenningsalgoritme bestaat uit 3 delen. Dat gebeurt allemaal in de klasse ImageProcessor.

4.5.1 Het herkennen van de driehoeken

Dit is de hoofdfunctie van het algoritme. Hier overlopen we alle pixels en indien ze deel zijn van de buitenrand van een driehoek, passen we een functie toe om 2 van de 3 hoekpunten te vinden. Vervolgens zoeken we het derde punt door de pixel te zoeken die het verste ligt van die twee punten en dezelfde kleur heeft. Deze functie geeft een array van driehoeken terug. Voor pseudocode zie Algoritme 3.

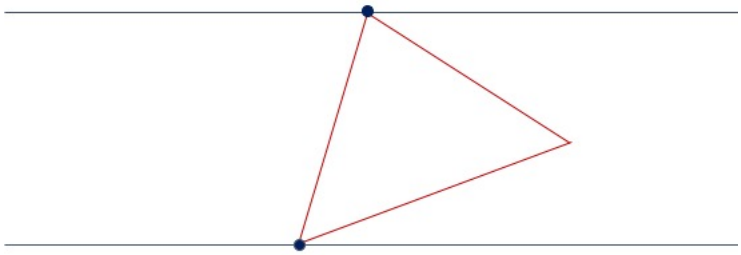
Algorithm 3: Zoekt driehoeken in een afbeelding

```
for i = 1..imageWidth do
    for j = 1..imageHeight do
        if kleur van pixel(i,j) is randkleur then
            Voeg kleur toe aan map
        end
    end
end
for Kleur in kleurenmap do
    Zoek verste punt
    Maak driehoeken per kleur
end
return kleurenmap
```

4.5.2 Een kleur toevoegen aan de kleuren map

De functie controleert of de kleur al in de map zit. Indien niet, wordt de kleur toegevoegd en de oorspronkelijke x- en y-coördinaat worden bijgehouden als eerste voorkomen van de kleur. Indien de kleur al in de map zat, wordt de x- en y-coördinaat geüpdatet indien de y-coördinaat van de huidige pixel groter is dan de vorige pixel van deze kleur die is voorgekomen.

Na de uitvoering van deze functie op alle pixels bevat de kleuren map voor elke kleur het eerste en laatste voorkomen van een pixel in functie van zijn y-coördinaat. Omdat we ook de x-coördinaten bijhouden, hebben we zeker al 2 van de 3 hoekpunten gevonden van elke driehoek.



Figuur 6: 2 punten zijn gekend, de zone tussen die twee punten worden overlopen voor het verste punt

4.5.3 Het vinden van het verste punt

Voor elke kleur zoeken we het punt dat het verste ligt van de twee gevonden punten en die dezelfde kleur heeft. Voor de pseudocode zie algoritme 4.

Algorithm 4: Zoeken van het verste punt

```
minY = de y-coördinaat waarop de kleur voor het eerst voorkomt;
maxY = de y-coördinaat waarop de kleur voor het laatst voorkomt;
kleur = kleur waarvoor het verste punt gezocht wordt;
for i=0..imageWidth do
    for j=minY..maxY do
        if kleur = kleur van pixel(i,j) ∧ afstand tot pixel(i,j) is groter dan de huidige afstand
         then
            update punt met grootste afstand
        end
    end
end
return pixel met grootste afstand tot de twee punten
```

4.5.4 Fouten detectie

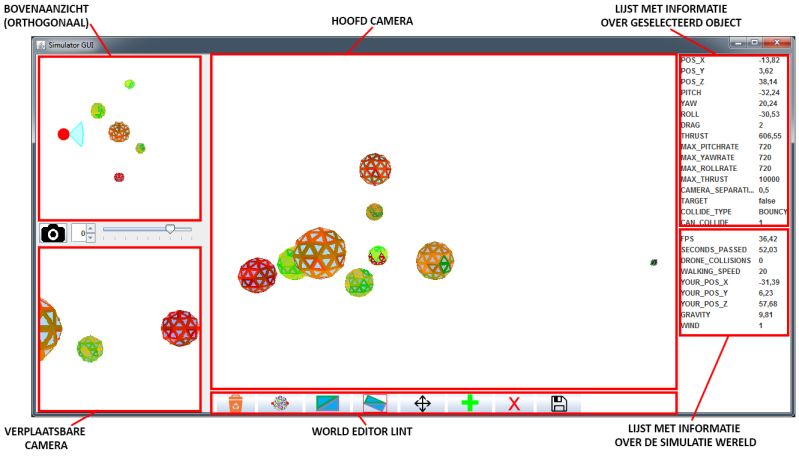
Om de uitschieters te filteren wordt de vorm van de driehoek nagekeken wordt op onregelmatigheden. Indien een van de zijden van de driehoek groter is dan een bepaalde waarde, verwerpen we de driehoek en slaan we die niet op in de wereld. Deze waarde hebben we empirisch bepaald en die is gelijk aan 3 eenheden(volgens het stelsel van ons TestBed). Ook wanneer twee van de drie zijden van een driehoek meer dan drie keer groter zijn dan de derde zijde, verwerpen we de driehoek en beschouwen we die als onregelmatig.

4.6 Omzetten van driehoeken naar 3D objecten

Eenmaal we de 2D-driehoeken hebben berekend van elk oog van de drone, berekenen we de exacte positie van de 3D-driehoeken en die slaan we op als TriangleFace. Samen met de 3-coördinaten slaan we ook de kleur op van elke TriangleFace. Voor het algoritme om 3D-objecten te maken, verwijzen we naar ons verslag van vorig semester.

4.7 Veelvlak generatie

De drie dimensionale driehoeken bekomen in de vorige sectie worden door de AI samengenomen tot veelvlakken. Hiervoor wordt eerst op alle hoekpunten punten van alle driehoeken een clusteringsalgoritme toegepast. Dit algoritme neemt alle punten die binnen de empirisch bepaalde afstand van



Figuur 7: De finale vorm van de SimulatorGUI

0,5 van elkaar liggen samen en herleidt deze tot hun rekenkundig gemiddelde. Door toepassing van dit algoritme sluiten de gedetecteerde driehoeken mooi op elkaar aan een wordt de detectiefout uit gemiddeld.
Driehoeken die op elkaar aansluiten worden in een tweede stap samengenomen tot veelvlakken. Voor de eenvoud stelt de routeplanner alle veelvlakken voor als bollen. Deze bollen hebben als middelpunt de gemiddelde waarde van de hoekpunten van alle driehoeken geassocieerd met dit veelvlak. De straal is gelijk aan aan de maximale afstand van het middelpunt tot een hoekpunt van een driehoek. Een beperking van deze voorstelling is dat het niet correct werkt met asymmetrische vormen of vormen met een opening in zoals een torus. Afhankelijk van de waarde van de kleur van de geassocieerde driehoeken wordt ook een boolean aan het veelvlak toegekend dit aangeeft of het een doel of obstakel is.
Omdat detectie van op een afstand vaak slechter is, wordt voor de berekenende positie van de bollen gebruik gemaakt van een exponentiële afvlakking waarbij de nieuw berekende waarde meetelt voor 10 procent [2].

5 GUI

Dit hoofdstuk zal de aanpassingen aan de Graphical User Interfaces sinds het vorige verslag bespreken. Het gaat er dus van uit dat de lezer al weet heeft van hoe de vorige GUI's eruit zagen en werkten.

5.1 SimulatorGUI

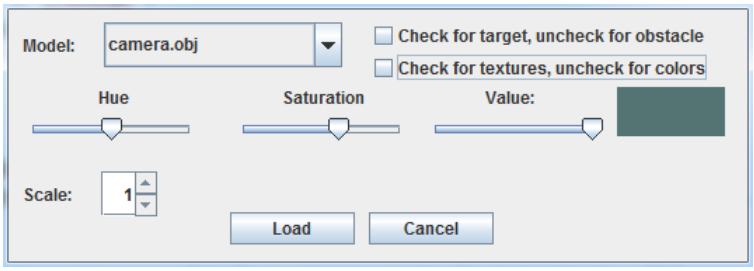
De SimulatorGUI heeft, zoals Figuur 7 toont, voornamelijk veranderingen in de knoppen van het world editor lint:

5.1.1 Verwijderde functies

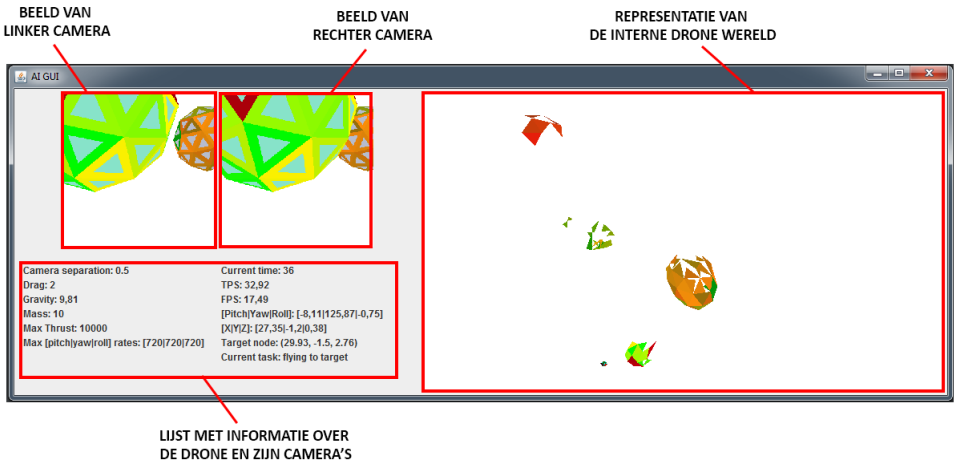
1. Het toevoegen van een nieuwe target bal met een voorgedefinieerde kleur is niet meer mogelijk.
2. De gebruiker kan een nieuwe obstakel bal niet meer toevoegen.

5.1.2 Toegevoegde functies

1. Het toevoegen van een nieuwe doel- of obstakel-object, gedefinieerd als wavefront .obj [4], in de wereld met een te kiezen kleur dankzij een nieuwe GUI, zichtbaar op figuur 8
2. De gebruiker kan, door op een knop te drukken, kiezen of hij de bounding geometries van de zichtbare objecten wil zien zoals in Figuur 3



Figuur 8: De GUI om een object in het testbed in te laden uit een wavefront .obj file



Figuur 9: De finale vorm van de AI GUI

3. Een nieuwe knop laat de gebruiker toe een geselecteerd object te kopiëren en dan te plaatsen waar hij wil.

5.2 AI GUI

Ook bij de AI zijn er aanpassingen gemaakt zoals Figuur 9 toont. De grootste verandering is dat de AI nu ook weergeeft hoe hij de wereld ziet. Dit is een gereduceerde versie van de functionaliteit die reeds bestond in het testbed GUI. De gebruiker kan ook kiezen om de modellen, die de drone van de opgeslagen driehoeken maakt, te tonen door op spatie te drukken. Verder kan met de p-toets het geplande pad van de drone getoond worden.

5.3 LoaderGUI

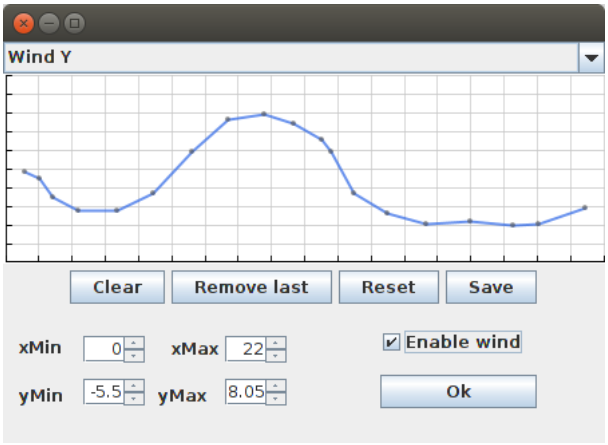
De Loader GUI is aangepast om ook de nieuwe world description (v2) te ondersteunen. Werelden kunnen geëxporteerd en geïmporteerd worden in het binaire .wdf formaat.

5.4 WindGUI

In deze programmaversie is er een nieuwe GUI toegevoegd die een gebruiker toelaat om de lineaire interpolators voor de wind en de rotationele wind grafisch te bekijken en aan te passen. Dit is nuttig aangezien er buiten de externe binaire .wdf files of het direct aanpassen van programmacode geen manier is om zelf wind in het systeem te brengen.

Figuur 10 toont het uitzicht van deze GUI. De GUI wordt opgeroepen door op de eigenschap 'Wind' van in de SimlatorGUI te dubbel kikken. Een bespreking van de functies:

- De bovenste ComboBox laat een gebruiker de gewenste interpolator kiezen. Door in de grafiek eronder te klikken kunnen nieuwe punten toegevoegd worden.



Figuur 10: De GUI om de wind interpolators te beheren

- De knop 'Remove last' verwijdert het laatst toegevoegde interpolatiepunt.
- 'Clear' wist alle punten van de interpolator. 'Reset' zet de interpolator terug naar de versie die het laatst opgeslagen was.
- 'Save' schrijft de huidige (bewerkte) interpolator weg naar het testbed systeem.
- De bewerkbare tekstveldjes linksonder tonen de minimale en maximale x- en y waarden van de grafiek. Wanneer deze veranderd worden past de grafiek zich dynamisch aan. Deze kunnen gebruikt worden om de interpolator over een arbitrair lange tijdspanne en y-reikwijdte te laten lopen.
- De checkbox 'Enable Wind' is een schakelaar voor zowel lineaire als rotationele wind in het testbed.
- 'Ok' sluit de GUI.

6 Wireprotocol

Het wireprotocol is een manier om een autopilot met een testbed te verbinden via een netwerk. Dit was aanvankelijk bedoeld om het gegeven testbed te verbinden met de gecreëerde autopilot maar het is later ook uitgebreid naar het gecreëerde testbed. Het gecreëerde testbed ondersteunt drones die via de originele methode communiceren en drones die via het wireprotocol communiceren.

6.1 Drone

De AI heeft drie nieuwe klassen om het wireprotocol te ondersteunen: WireAI, WireCamera en WireDrone. WireDrone is een implementatie de gegeven Drone interface. Dit is nodig omdat de originele implementatie deze interface, genaamd DroneObject, enkel toegankelijk was via het testbed en de autopilot deze interface gebruikt. WireCamera is een implementatie van de gegeven Camera interface en is voor dezelfde reden aangemaakt. WireAI staat in voor zowel het initialiseren van een drone als de communicatie met het testbed. Indien de drone merkt dat de verbinding verbroken is sluit deze zichzelf af.

6.2 Testbed

Het testbed heeft een nieuwe klasse: WireSocket. WireSocket dient als server en accepteert alle inkomende drones als cliënt. Voor elke inkomende verbinding maakt het testbed een nieuwe drone aan zonder autopilot. Het testbed kan zowel drones die geïnitialiseerd zijn met autopilot als drones die via het wireprotocol communiceren tegelijk simuleren.

7 Besluit

Als team kunnen we zeggen dat we een geslaagd project afgeleverd hebben. We construeerden een volledig werkende simulatie met een drone die bestuurd is door een autopilot. De autopilot slaagt erin on zich in de wereld te voortbewegen met wind variaties. Verder kunnen willekeurige objecten gereconstrueerd worden als deze uit driehoeken met unieke randen bestaat. Gedurende het project hebben we nieuwe technieken aangeleerd met betrekking tot projectorganisatie, code onderhoud en testen, 3D-vision, collision en wereldreconstructie.

Referenties

[1] E. W. Dijkstra, *A note on two problems in connexion with graphs.* http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf.

[2] A. D. Little, *Exponential smoothing for predicting demand.* https://www.industrydocumentslibrary.ucsf.edu/tobacco/docs/#id=jzlc0130.

[3] T. Moller, *A fast triangle-triangle intersection test.* http://web.stanford.edu/class/cs277/resources/papers/Moller1997b.pdf.

[4] W. Technologies, *Wavefront .obj file.* https://en.wikipedia.org/wiki/Wavefront_.obj_file.

De week van ...	Yana Dimova	Thor Galle	Valentin Claeys	Ruben Coppens	Maarten Craeynest	Antoon De Cleen	Weektotaal
14/2: Sessie	05:00	04:25	06:00	05:00	05:00	05:00	30:25
21/2: Sessie	05:00	08:40	05:00	05:00	07:00	05:00	35:40
28/2: Sessie	05:00	06:05	07:30	06:00	07:00	08:00	39:35
7/3: Sessie	08:00	06:12	11:00	06:00	09:00	08:00	48:12
14/3: Sessie	07:00	06:08	05:00	07:00	09:00	06:00	40:08
21/3: Sessie	10:00	09:10	05:00	10:00	10:00	10:00	54:10
28/3: Sessie	08:00	05:30	06:30	07:00	05:00	05:00	37:00
18/4: Sessie	06:00	05:00	07:30	06:00	07:00	06:00	37:30
25/4: Sessie	06:00	05:42	08:00	07:00	06:00	07:00	39:42
2/5: Sessie	05:00	05:58	07:30	06:00	09:00	05:00	38:28
9/5: Sessie	06:00	06:53	06:30	05:00	06:00	05:00	35:23
16/5: Sessie	06:00	**05:00**	**08:00**	06:00	**08:00**	07:00	
Individueel totaal	**77:00**	**74:43**	**83:30**	**76:00**	**88:00**	**77:00**	**436:13**

Figuur 11: Wekelijkse tijdbesteding van de teamleden aan het project.

A Beschrijving van de werkverdeling

Het P&O team brons bestaat uit zes leden: Valentin Claeys, Ruben Coppens, Maarten Craeynest, Antoon De Cleen, Yana Dimova en Thor Galle.
Hier onder wordt besproken waar elk teamlid zich mee heeft bezig gehouden. De wekelijkse tijdbesteding van de teamleden aan het project staat in Figuur 11.

- **Valentin Claeys** heeft gewerkt aan aan beide GUIs. Waarbij de focus vooral lag op Simulator GUI. Hij heeft ook meegewerkt aan het wireprotocol.
- **Ruben Coppens** was verantwoordelijk voor de driehoek detectie en hoe de AI deze informatie moest verwerken.
- **Maarten Craeynest** heeft voornamelijk geholpen bij de driehoekdetectie en het testbed.
- **Antoon De Cleen** heeft meegewerkt aan de besturing, de routeplanner en het wireprotocol.
- **Yana Dimova** is de CEO van het team. Zij heeft gewerkt aan de beeldherkenning van de AI samen met Ruben.
- **Thor Galle** is de CAO van het team. Hij heeft gewerkt aan het testbed. Eerst in het algemeen en nadien vooral collision detection. Hij was ook verantwoordelijk voor het lezen en schrijven van werelden. Algemeen was hij verantwoordelijk voorde CAO taken zoals de planning en latex files in GitHub steken.

B Beschrijving van het proces

In deze sectie wordt kort besproken hoe we het project hebben aangepakt als een team, waar onze moeilijkheden zaten en welke lessen we hier uit hebben getrokken.

B.1 Moeilijkheden

Tijdens het project traden er toch regelmatig moeilijkheden op. De meeste hiervan konden snel opgelost worden. Hieronder worden kort de belangrijkste problemen vermeld:

- **Wire Protocol** In het begin hebben we ons gefocused op het herkennen en weergeven van polyhedra. Dat bleek meer werk te zijn dan verwacht en daarom hebben we de milestone voor de Wire Protocol niet gehaald op de tussentijdse deadline. Pas een paar weken na de deadline waren de Wire Drone en Testbed volledig af.
- **Beeldherkenning** Geen van ons was vertrouwd met beeldherkenning. Hierdoor hebben we veel verschillende algoritmes proberen te implementeren zonder succes.

- **Crossplatform** We hadden zowel Linux, OSX als Windows systemen in ons team waardoor er sommige libraries nog niet uit waren voor iedereen. Daardoor hebben we oudere versies van openGL moeten gebruiken.

B.2 Lessen getrokken uit het project

Door te zoeken naar oplossingen voor de moeilijkheden beschreven in de vorige subsectie hebben we als team enkele lessen geleerd:

- We hebben leren werken met Git en Github, wat de efficiëntie van het samenwerken verbeterde. In het begin verliepen zaken zoals merges moeizaam. In het tweede semester ging het beter en sneller omdat we meer vertrouwd waren met Git.
- Werken met openGL was nieuw voor ons.
- Het implementeren van collision detection was leerrijk.

B.3 Werken in teamverband

Om voor communicatie te zorgen hebben we elke P&O sessie twee team meetings gehouden. Eentje in het begin van de sessie, waarbij we bespreken wat er tijdens de sessie moet gebeuren en eentje op het einde van de sessie, waarbij we bespreken hoe ver we zitten en wat we nog gaan afmaken tegen de week erna. Daarnaast hebben we gebruik gemaakt van een facebook groep voor de communicatie buiten de zittingen. Ook was er een google drive waarop administratieve zaken werden bijgehouden (zoals hoeveel tijd een teamlid wekelijks in het project steekt).

C Kritische analyse

In deze sectie wordt kort besproken wat de sterke en zwakke punten van ons project zijn.

Enkele zwakke punten:

- De beeldherkenning door de drone gebeurt iets te traag waardoor objecten niet altijd correct worden weergegeven in de wereld.
- We zijn er niet in geslaagd om door complexe objecten te vliegen. Dit komt grotendeels door het feit dat de beeldherkenning niet optimaal is waardoor er moeilijke complexe objecten herkend kunnen worden.
- Door de manier waarop de drone objecten die hij reeds gezien heeft opslaat, is het mogelijk dat sommige target ballen nooit bereikt kunnen worden.

Enkele sterke punten:

- De Wind GUI laat toe dat de gebruiker volledig zelf controle heeft over de wind veranderingen.
- Het testbed is uitgebreid en gebruiksvriendelijk en laat het instellen van waarden en werelden toe zonder te moeten afsluiten.
- De AI GUI geeft een duidelijk beeld van wat de drone ziet en weet over de wereld.
- De routeplanner en vliegprestatie van de drone zijn beide goed. Dit wil zeggen dat hij heel accuraat kan vliegen zonder ongewenste botsingen.