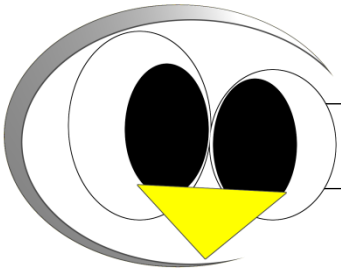
Frank van Smeden

Afstudeerverslag



3D renderer implementatie met OpenGL

Academie voor ICT & Media

Bredewater 24 Zoetermeer

September 2010

Referaat

Omschrijving
Dit verslag bevat alle ontwikkelingen rond een afstudeerproject op 3D engine development in combinatie met het gebruik van OpenGL.

Descriptoren
Crossplatform game development, OpenGL, renderer, 3D Engine development

Organisatie
Academie voor ICT & Media te Zoetermeer
Bredewater 24
2715 CA Zoetermeer

Afstudeerperiode

begin:	26 april 2010
afronding:	24 september 2010

Examinator (begeleider)
A. Nederend

Examinator (expert)
T. Cocx

Afstudeerbegeleider
V. Broeren

Auteur
Frank van Smeden

Voorwoord

De game-industrie is in grote de laatste twintig jaar enorm toegenomen. En waarmee men in de tachtiger jaren nog genoegen nam met een simpel 2 dimensionale game, zijn nu de 3 dimensionale speelwerelden niet meer uit de game-industrie weg te denken. Binnen de game-development wereld werd hier uiteraard op ingespeeld en de kwam de term serious gaming tot leven, wat weergeeft dat het ontwikkelen van games big business is.

Games zijn inmiddels allang geen spelletje meer maar zijn uitgegroeid tot een serieuze tak in het bedrijfsleven waar grote hoeveelheden geld in omgaan. De game-studio's waar programmeurs werken aan een nieuwe game om op de markt te zetten schoten de afgelopen 10 dan ook uit de grond. Ook het onderwijs besteedt steeds meer aandacht aan 3D gerelateerde onderwerpen. De Academie voor ICT & Media loopt hierin in Nederland voorop met onderwijs in het ontwikkelen van 3D game-engines. Deze Academie is namelijk de enige in Nederland die studenten leert een 3D engine te ontwikkelen.

Mijn project waarin ik in dit document verslag doe, zal zich richten op deze tak van de Academie voor ICT & Media.

Verder wil ik graag een aantal leraren bedanken die me gedurende mijn studie en mijn afstuderen hebben geholpen. Allereerst wil ik graag Dhr. V. Broeren en Dhr. J. de Knegt bedanken die me tijdens mijn afstuderen hebben ondersteunt en me kennis hebben bijgebracht over 3D development.

Daarnaast wil ik graag Dhr. A. Nederend bedanken voor de vier jaar lange begeleiding tijdens mijn studie en bij begeleiding van mijn afstuderen, waar ik tevens Dhr. T. Cocx voor wil bedanken.

Als laatste wil ik graag Dhr. A. Biegstraaten bedanken voor diens aandeel binnen mijn studie en mijn afstuderen.

Inhoudsopgave

1. Inleiding

Dit verslag heb ik (Frank van Smeden) geschreven in opdracht van de Academie voor ICT & Media. Als vierde jaars informatica student was ik op zoek naar een afstudeerplek waarin ik iets met crossplatform programmeren zou kunnen doen. Na rondgezocht te hebben kwam ik uiteindelijk uit op de sector game-development uit van de Academie, waar ik wel een eigen project kon krijgen. Na wat projectmatige details te hebben doorgesproken kon ik beginnen met mijn project: het implementeren van een nieuwe renderer implementatie binnen een bestaande 3D engine.

Een 3D engine is software die gebruikt kan worden voor allerlei grafische applicaties. Een voorbeeld van een dergelijke grafische applicatie is een game. Een 3D engine zorgt niet alleen voor het creëren van 3D werelden maar heeft ook de verantwoordelijkheid over het afvangen van input (muis, keyboard et cetera), en nog meer onderdelen waar een game mee te maken heeft.

Het doel van dit verslag is inzicht geven aan mijn begeleiders van dit project over wat ik tijdens mijn afstudeerperiode heb geleerd en hoe ik te werk ben gegaan om het project tot een goed einde te brengen. Daarnaast zal dit verslag interessant zijn voor mijn bedrijfsmentor om de werkwijze en project voortgang te kunnen reflecteren.

De structuur van dit verslag is als volgt opgebouwd. Ik begin met de beschrijving van het bedrijf waar ik dit project heb uitgevoerd (de Academie voor ICT & Media). Daarna vertel ik meer over mijn opdracht en wat ik precies heb uitgevoerd. In hoofdstuk 4 en 5 ga ik meer vertellen over hoe ik me heb voorbereid op het uitvoeren van het project. Hoofdstuk 6 tot 10 beschrijven al mijn uitvoeringen met uitleg hierover. De evaluatie is beschreven in de hoofdstukken 11 ,12 en 13.

2. Organisatie beschrijving

Dit project is uitgevoerd op de Academie voor ICT & Media.

De Academie voor ICT & Media (AICTM) is een onderdeel van de Haagse Hogeschool. Binnen deze academie vallen de volgende opleidingen:

- Informatica
- Bedrijfskundige Informatica
- Information Security Management
- Communicatie en multimedia design
- Informatiedienstverlening en –management
- Technische informatica

Op dit moment (naar september 2010) geven er, totaal op de hele AICTM, 164 docenten les en zijn er 2260 studenten ingeschreven voor een studie. De Haagse Hogeschool heeft drie vestigingen waarop de AICTM actief is. Deze vestigingen bevinden zich in Den Haag, Delft en Zoetermeer. Mijn project zal uitgevoerd worden in opdracht van de AICTM in Zoetermeer.

Op deze vestiging worden sinds 2005 minoren aangeboden waarin studenten kunnen leren omgaan met 3D software. Tijdens deze minoren worden de studenten geacht een 3D engine op te zetten of deze aan te passen volgens de wensen van Vincent Broeren. Vincent is de begeleider van het onderdeel 3D engine development op de AICTM. Een 3D engine is software die gebruikt kan worden door allerlei grafische applicaties, bijvoorbeeld bij computer spelletjes.

Sinds 2008 wordt er door studenten die minoren in 3D software hebben gevolgd, een 3D engine onderhouden. Deze engine, genaamd Polemos, is een engine die geheel onderhouden en uitgebreid wordt door studenten. Inmiddels is deze engine uitgegroeid tot een applicatie met ongeveer 40.000 regels broncode die voor het grootste deel geschreven zijn in de programmeertaal C++. Eigenlijk bestaat Polemos uit een aantal stukken software(libraries) die bij elkaar de gehele 3D engine vormen. Elke library heeft zijn eigen verantwoordelijkheden en wisselt informatie uit met andere libraries binnen de engine en daarbuiten. Enkele voorbeelden van de Polemos libraries zijn hieronder weergegeven:

Library naam	Omschrijving
Windows	Library die verantwoordelijk is voor het aanmaken van een window op het beelscherm
Input	Zorgt voor het opvragen van de invoer van keyboard, muis en eventueel een controller
ResourceManagement	Houd alle externe bestanden bij die worden ingeladen in de engine.

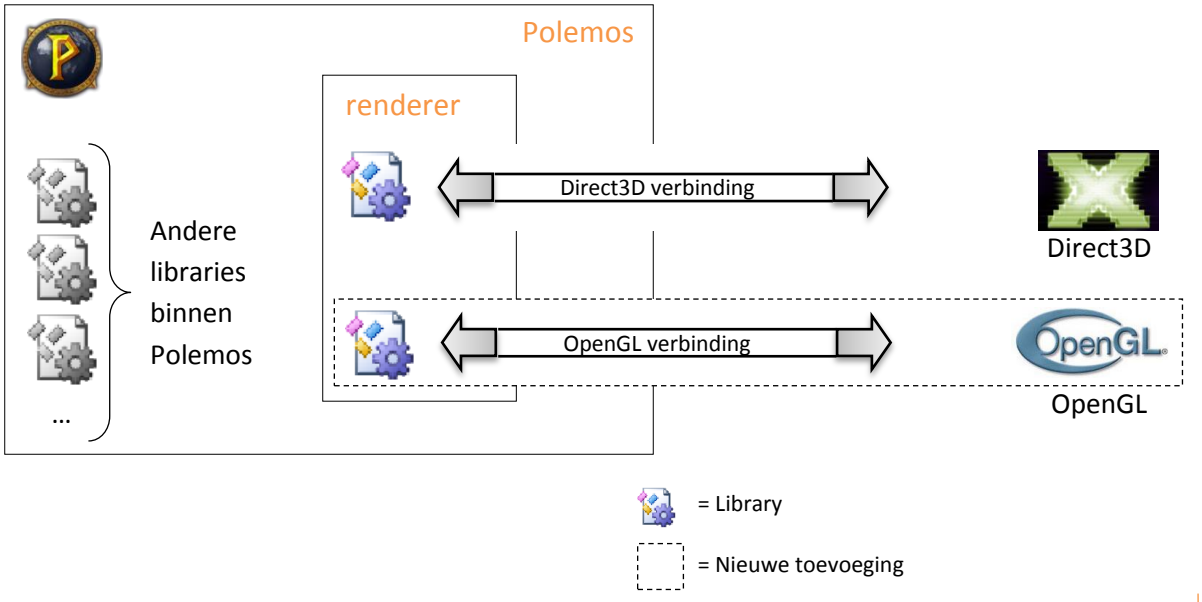
Mijn afstudeerproject heeft zich gespitst op deze engine. Ik heb Polemos uitgebreid door het programmeren van een nieuwe renderer implementatie. Een renderer is onderdeel van een 3D engine. Dit deel zorgt voor de grafische output van de engine en bouwt driedimensionale werelden binnen een grafische kaart zo om, dat er op het beeldscherm een 2D beeld wordt getoond.

Omdat een nieuwe renderer implementatie een langdurig proces is, en niet binnen het tijdsbestek van een afstudeer periode valt, zal ik alleen basisfunctionaliteit van een nieuwe renderer implementatie implementeren.

Tijdens dit proces ben ik begeleid door Dhr. Vincent Broeren, expert op het gebied van 3D game-development op de AICTM.

3. Opdrachtomschrijving

Zoals beschreven in de organisatie beschrijving heb ik verder ontwikkeld aan de Polemos engine. Ik heb aan deze engine een deel van een nieuwe implementatie van een renderer aan toegevoegd. Voordat ik met dit project begon was er al een implementatie van een renderer. Deze renderer implementatie maakte gebruik van een externe library genaamd Direct3D. De nieuwe implementatie maakt gebruik van een andere externe library genaamd OpenGL. De volgende afbeelding laat zien hoe de structuur op het gebied van de Polemos renderer eruit ziet en wat dit project heeft toegevoegd.



Afbeelding 1: Polemos renderer OpenGL toevoeging

Direct3D en OpenGL zijn libraries die de programmeur in staat stellen te communiceren met de grafische kaart in een computer systeem. Als er dus veel rekenkracht nodig is voor het berekenen van een grafische instructie binnen een computer, is deze het snelste berekend als deze instructie naar de videokaart (GPU) wordt gestuurd.

3.1 Probleemstelling

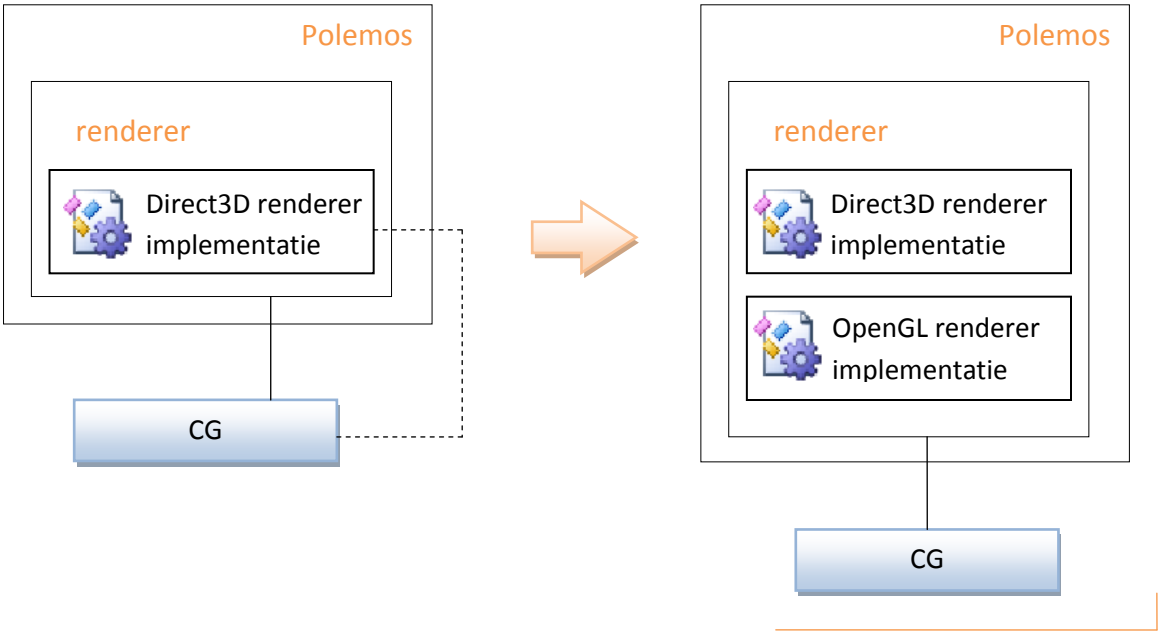
Polemos is op dit moment niet crossplatform. Crossplatform betekent dat de software werkt op meerdere besturingsystemen (bijvoorbeeld op Windows®, MAC OS® en Linux). Polemos is geschreven voor het Windows® besturingssysteem gebruikt meerdere externe libraries die alleen werken op Windows®. De Direct3D renderer is een van de libraries die Polemos Windows® afhankelijk maken. OpenGL is niet platform afhankelijk en brengt Polemos dus een stap dichterbij een crossplatform implementatie.

De reden waarom de AICTM graag een crossplatform renderer implementatie toegevoegd wil hebben aan Polemos, is ten behoeve van het onderwijs en om misschien in de toekomst geheel over te gaan op crossplatform libraries om zo Polemos crossplatform te maken. Het ontwikkelen van een OpenGL implementatie is dus tegelijkertijd ten behoeve van een onderzoek naar of het effectief is om Polemos crossplatform te maken. Ik zal ook aan het einde van het project een adviesrapport opleveren waarin ik deze vraag beantwoord.

Een bijkomstige moeilijkheid in dit project is dat niemand op de AICTM met OpenGL heeft gewerkt. Vincent Broeren heeft alleen ervaring met het gebruik van een Direct3D renderer. Mede hierdoor krijgt de opdracht een onderzoekend karakter.

3.2 Doelstelling

Het doel van dit project zal zijn om de Polemos engine zo aan te passen dat de renderer, naast Direct3D, ook OpenGL kan gaan gebruiken. De huidige Direct3D implementatie zal dus niet verdwijnen maar de engine zal worden uitgebreid.



Afbeelding 2: Globale doelstelling

Door de aanpassingen van de renderer binnen Polemos, heb ik een aantal veranderingen binnen de engine aangebracht. Zo heb ik de engine volledig abstract gemaakt. Dit houdt in dat er binnen de 3D engine geen verbindingen meer zijn naar Direct3D of OpenGL behalve in de renderer van de engine. Voordat ik dit project begon was niet bekend of deze abstractie al was ingebouwd. Daarnaast was deze abstractie ook nog nooit getest omdat dit alleen mogelijk is als de engine meerdere renderers bevat.

Zoals te zien is in afbeelding 2 (“Globale doelstelling”), bestond er in Polemos een verbinding tussen Direct3D en Cg. Cg is een programmeertaal. Deze taal stelt een 3D engine in staat om bepaalde effecten te kunnen berekenen (bijvoorbeeld licht en schaduw). Deze effecten worden shaders genoemd. Deze verbinding is niet ideaal. In mijn project heb ik gekeken of deze verbinding niet weg kan worden gehaald (zie hoofdstuk: sprint 2 uitvoeringen).

De hoeveelheid tijd die ik bezig ben geweest met de renderer is bepalend geweest voor het verloop van het project. Bij aanvang van het project was het lastig te bepalen hoelang de implementatie van de OpenGL renderer aan tijd zou kosten. Daarom heb ik in overleg met mijn bedrijfsmentor requirements opgesteld die aangeven wanneer het project voltooid is. Dit zou zijn zodra er basisfunctionaliteit van een OpenGL renderer implementatie zou worden opgeleverd (zie hoofdstuk: initialisatie-fase: opstellen requirements).

Naast de functionaliteit van een OpenGL renderer implementatie heb ik een adviesrapport geschreven over de efficiëntie van een crossplatform Polemos. Hierin adviseer ik of het effectief is om Polemos crossplatform te gaan maken.

Om de functionaliteit van deDirect3D renderer implementatie te kunnen waarborgen heb ik tests uitgevoerd. Deze tests zijn gedurende het hele project uitgevoerd om er zeker van te zijn dat de functionaliteit van de Direct3D renderer geen schade ondervond van de gemaakte aanpassingen. De tests zijn uitgevoerd door middel van een softwarepakket genaamd CXXtest. Deze software kan unittests uitvoeren op C++ broncode.

3.3 Resultaat

Voor het stagebedrijf is het voornaamste product in dit project, de nieuwe OpenGL renderer implementatie en de documentatie hiervan. Verder is het resultaat zijn dat de engine flexibeler is geworden en dat, nu er een begin is gemaakt, er misschien door andere studenten later nog verder gebouwd gaat worden om Polemos geheel crossplatform te maken.

Om Polemos crossplatform te maken zullen er nog meer libraries van Polemos moeten worden aangepast die, net als Direct3D, alleen op het Windows® besturingsysteem werken. Pas zodra iedere library op meerdere platformen functioneel is, is Polemos crossplatform.

Voor de Academie als opleidingsinstituut is mijn afstudeerverslag het voornaamste product geweest. Hierin staat informatie waaraan getoetst kan worden of de student genoeg competenties heeft gehaald om te mogen afstuderen.

Voordat ik het project begon had ik succesfactoren opgesteld waaraan in dit project moest worden voldaan om het een geslaagd project te laten zijn. Deze factoren staat opgesteld in het plan van aanpak (Zie hoofdstuk 3: Plan van aanpak).

4. Plan van aanpak

In dit hoofdstuk staat mijn Plan van Aanpak beschreven. Hierin komen onder andere de mijlpalen van dit project, de methodiek en de globale planning naar voren. Ook zal ik de implementatie van Scrum in dit project beschrijven en toelichten.

4.1 Mijlpalen

Het project zal zijn geslaagd zodra er aan de volgende criteria zal zijn voldaan:

- Er is een OpenGL implementatie aanwezig
  - OpenGL kan worden geïnitialiseerd door Polemos en er kan basis functionaliteit worden gebruikt via de OpenGL renderer. Deze basisfunctionaliteit bevat de volgende eisen:

    Polemos…
    - Is in staat om OpenGL te gebruiken om een window op het scherm te tonen
    - Kan de kleur van de content van het window kan worden aangepast.
    - Heeft een abstracte renderer.

- Er zijn unittests van de twee renderers aanwezig om de functionaliteit hiervan te waarborgen.
  - In mijn project zal ik unittests maken die de beide renderer implementaties zullen testen. Hierbij zal ik beginnen met de Direct3D unittest zodat ik ervan verzekerd kan zijn dat mijn werkzaamheden aan de OpenGL renderer implementatie de Direct3D renderer implementatie niet zullen verstoren.
- Er is documentatie aanwezig van de aangepaste delen van Polemos
- Er is een onderzoeksrapport over de verschillen tussen Direct3D en OpenGL
- Er is Scrum documentatie aanwezig (o.a. product-backlogs en sprint-backlogs)
  - Scrum documentatie is een bijproduct en zal dus niet apart worden opgenomen als requirement in een sprint-backlogs.
- Er zal een adviesrapport worden geschreven over een volledige crossplatform implementatie van Polemos

Deze mijlpalen heb ik samen met de product-owner opgesteld voor de aanvang van het project. Het was toen nog niet zeker of ik dit project zou doen en we hebben besloten samen naar de producten te kijken om zo tot een besluit komen of dit project als mijn afstudeerproject wilde.

4.2 Methodiek

Tijdens dit project heb ik gebruik gemaakt van Scrum. Scrum is een raamwerk van allerlei technieken die gericht zijn op de ontwikkeling van software. De product-owner van dit project had van tevoren vastgesteld dat ik Scrum moest gaan gebruiken. Om te kunnen beoordelen of ik Scrum wel een geschikte techniek vond voor dit project heb ik me hierin eerst verdiept. Ik had namelijk nog nooit met Scrum gewerkt. Ik heb me ingelezen in het boek *Agile software development. Principles, Patterns , and practices* (zie hoofdstuk: gebruikte literatuur) en op internet literatuur geraadpleegd. Ook heb ik de

vragen die ik had, gesteld aan mijn bedrijfsmentor. Om een evenwichtig oordeel te kunnen geven over het gebruik van deze methode heb ik specifiek gelet op een aantal factoren:

- Hoe werkt de planning bij deze methodiek?
  - ***Probleem***
    Het inschatten van de tijdsduur van de producten in dit project is ingewikkeld. Dit komt grotendeels doordat er niet veel documentatie beschikbaar is over de Polemos engine en omdat niemand van de AICTM mij kan ondersteunen in het gebruik van OpenGL.

    Antwoord
    Door dit onderzoekend karakter wordt Scrum erg interessant om te gebruiken. Scrum plant pas iteraties in, zodra de vorige iteratie is afgerond. Hierdoor kan er dus snel gereageerd worden op eventuele veranderingen die de vorige iteratie met zich meebracht.

- Hoe reageert de methodiek op onvoorziene veranderingen?
  - ***Probleem***
    Omdat ik niet precies weet hoe de software qua structuur geïmplementeerd is, binnen de omgeving waarin ik ga werken, is de kans op onvoorziene veranderingen groter dan als dit wel het geval zou zijn.

    Antwoord
    Binnen Scrum wordt de eigenaar van het project nauw bij het project betrokken. Hierdoor kunnen onvoorziene problemen snel worden afgehandeld naar de wens van de eigenaar (product-owner). Ook door het gebruik van vaak kleine iteraties kan er gemakkelijker worden ingespeeld op onvoorziene factoren.

- Heb ik ervaring met deze methodiek?
  - ***Probleem***
    Een nieuwe methodiek eigen maken kost tijd. Het is daarom, om tijdsredenen, praktischer om een methodiek te kiezen waarmee ik ervaring heb.

    Antwoord
    Met Scrum had ik geen ervaring. Het zou daarom een meer voor de hand liggende keus zijn geweest om RUP te gaan gebruiken waar ik jaren ervaring mee heb.

- Is er al documentatie aanwezig van de huidige situatie, welke methodiek word daar gebruikt?
  - ***Probleem***
    Als er reeds documentatie aanwezig is van het project, kan dit een reden zijn hierop verder te gaan. Dit om de overzichtelijkheid van het project te waarborgen.

    Antwoord
    Deze documentatie was al aanwezig in de vorm van Scrum.

Na deze factoren te hebben afgewogen kwam ik tot de conclusie dat Scrum een passende methodiek is die goed aansluit bij dit project. Doorslaggevend was de factor waarin de methodiek omgaat met de planning.

Bij het kiezen van Scrum als methodiek kwamen echter ook wat risico's mee. Ik had persoonlijk nog nooit met Scrum gewerkt. Ik had dus geen basiskennis en wist dus ook niet waarop ik moest letten bij het gebruik van Scrum. Dit had mijn project negatief kunnen beïnvloeden en me veel tijd kunnen kosten als ik Scrum verkeerd in mijn project zou toepassen. Dit is waarom ik regelmatig mijn bedrijfsmentor heb gevraagd om advies en veel zelfstudie gedaan heb naar de gebruiken binnen Scrum. Uiteraard waren er ook alternatieven. Zelf had ik al wel veel ervaring met de methodiek RUP (Rational Unified Process). De keus voor RUP te gaan heb ik dan ook zeker overwogen. Maar na de zojuist opgesomde aspecten goed te hebben bekeken bleek dit project toch beter bij Scrum te passen. Verder vond ik het ook voor mijn persoonlijk leerproces beter om verder te kijken dan RUP en iets te leren wat ook in de huidige markt iets is wat vaak wordt gebruikt.

4.2.1 Producten

Scrum werkt met een aantal producten. De producten van scrum die ik binnen mijn project heb gehanteerd zijn de volgende:

- Product-backlog
  - De product-backlog bestaat uit de volgende items:
    - Het nummer van het item. Ieder item heeft een vaststaand nummer om onduidelijkheden over de items te voorkomen en om naar te kunnen refereren.
    - Ingeschatte tijd. Hier wordt geregistreerd hoeveel tijd er gedacht wordt nodig te hebben voor een bepaald item uit de product-backlog. Deze schatting wordt gemaakt door mij en gekeurd door de product-owner. Deze tijd wordt aangegeven in units waarbij 1 unit gelijk staat aan 10 minuten.
    - De omschrijving van het item dat moet worden uitgevoerd.
    - Prioriteit. De urgentie van het item, hieronder een overzicht van de mogelijke prioriteiten:

Prioriteit	
1	Laag
2	Gemiddeld
3	Hoog
4	Heel hoog
9001	Extreem

De prioriteiten dienen voor het indelen van een sprint. Items met een hoge prioriteit worden eerder op een sprint-backlog gezet dan een lagere prioriteit. Daarnaast is "9001" een prioriteit waarmee ik de product-owner kan verplichten deze op een sprint-backlog te plaatsen

Alle documentatie van Polemos maakt reeds gebruik van deze richtlijnen.

- Sprint-backlog
  - Voorafgaande aan een sprint wordt er een sprint-backlog opgesteld. Hierin staan de items uit de product-backlog waaraan wordt gewerkt in de aankomende sprint.
- Burndown chart
  - Een burndown chart is een grafiek die de voortgang van de sprint laat zien. Hierin staan de units vermeld die nog niet voltooid zijn tegenover het aantal dagen van de sprint.
- Product-backlog advies
  - Voordat een sprint-meeting begint, zal ik een product-backlog advies document opstellen. In dit document worden uitvoerbare taken (items) gedefinieerd die op de product-backlog kunnen komen. Of deze ook daadwerkelijk op deze backlog komen is de keus van de product-owner.

Het bijhouden van deze producten van Scrum heeft een aantal doelen. Ten eerste heeft het de functie om de voortgang te kunnen bewaken. De burndown charts werken hier onder andere aan mee door de voortgang in een grafiek weer te geven. Ten tweede is het van belang voor product-owner om terug te kunnen kijken in het project naar wat er gedaan is op welk moment.

Alle opgeleverde producten heb ik per sprint gebackupt.

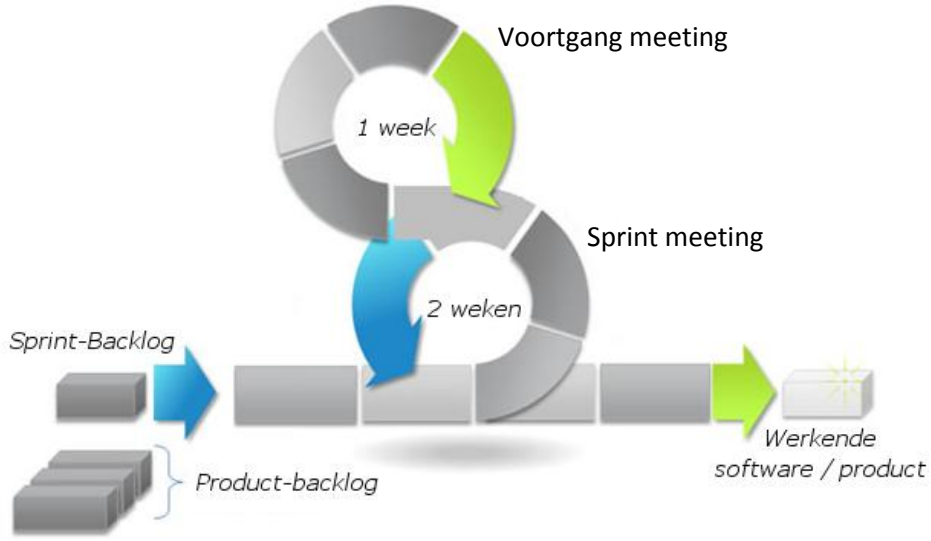
4.2.2 Meetings

Tijdens dit project heb ik een tweetal meetings gehanteerd, namelijk de sprint-meeting en een meeting halverwege de sprint genaamd de voortgang-meeting.

De sprint-meetings zijn gehouden in het bijzijn product-owner en vonden plaats aan het einde van een sprint.

Één sprint heeft een periode beslaan van twee weken. Dit omdat twee weken tijd de mogelijkheid biedt om werkende software en producten op te kunnen leveren. Minder dan twee weken zou niet toereikend zijn om een product te kunnen opleveren wat, voor de product-owner, interessant zou zijn. Bij een periode van meer dan twee weken kan de product-owner het project onvoldoende bijsturen mocht het project problemen tegenkomen.

Daarnaast heb ik in iedere voortgang-meeting een demonstratie gegeven waarin alles wat er sinds die sprint gedaan is werd getoond. Dit is voornamelijk een moment voor de product-owner geweest om de voortgang te kunnen bewaken.



Afbeelding 3: Scrum iteratie

4.3 Globale planning

Zoals beschreven, heb ik tijdens mijn project gebruik gemaakt van Scrum. Bij dit raamwerk van technieken behoort geen tot in detail uitgewerkte planning. De activiteiten worden immers per sprint ingedeeld. Toch heb ik ervoor gekozen een globale planning te maken. Dit om de hoeveelheid sprints in te kunnen schatten die ingepland kunnen worden in het gehele project. Op deze manier kan de product-owner ook een betere afweging maken bij het opstellen van de items voor een sprint-backlog.

Voordat ik kon gaan beginnen met sprints heb ik eerst een fase van initialisatie ondernomen waarin ik kennis heb opgedaan over de onderwerpen van mijn producten (o.a. Scrum, OpenGL en Direct3D). Deze fase duurde 4 weken leverde verschillende producten op. Deze producten hebben mijn kennis op het gebied van 3D programmeren verbreed. Deze kennis kon ik dan weer gebruiken binnen de sprints voor het werken aan Polemos.

De reden waarom deze producten niet zijn opgenomen in een sprint maar in een fase vooraf, is omdat deze niet direct te maken hebben met het project Polemos, maar met mijn persoonlijke kennis en vaardigheden.

Na deze Initialisatie- fase zijn er sprints gepland. Dit gebeurde in week 5. De sprint indeling had ik aan het begin van het project als volgt opgesteld:

- Sprint 1: week 5
- Sprint 2: week 7
- Sprint 3: week 9
- Sprint 4: week 11
- Sprint 5: week 13

Tijdens het project bleek deze indeling iets anders te verlopen en is sprint 2 een week uitgesteld. Meer hierover in hoofdstuk 6.2.

Na deze sprints zullen er nog twee weken zijn voor de uitloop. Hierin kunnen eventueel de opgeleverde producten worden verbeterd of aangepast. Mocht er genoeg tijd zijn kan er uiteraard ook worden verder gebouwd aan de renderer of eventuele andere producten.

Afstudeerweek	Activiteiten
1 (26 april)	Afstemmen opdrachten met product-owner Inrichting werkomgeving Installatie Polemos engine Start Plan van aanpak
2 (3 mei)	Onderzoek Direct3D API Oplevering Plan van Aanpak
3 (10 mei)	Onderzoek verschil Direct3D & OpenGL Opstellen requirements (o.a. voor product backlog)
4 (17 mei)	Onderzoek verschil Direct3D & OpenGL (+ oplevering) Verdieping Scrum / Agile Tutorials OpenGL Oplevering requirements Opstellen XP-dev account
5 (24 mei)	Begin Scrum-meetings Testen abstractie renderer Begin implementatie renderer
6 (31 mei)	implementatie renderer
7 (7 juni)	implementatie renderer
8 (14 juni)	implementatie renderer
9 (21 juni)	implementatie renderer
10 (28 juni)	implementatie renderer (implementatie eventuele andere libraries)
11 (5 juli)	implementatie renderer (implementatie eventuele andere libraries)
12 (12 juli)	implementatie renderer (implementatie eventuele andere libraries)
13 (23 aug)	Testen OpenGL renderer
14 (30 aug)	Schrijven Afstudeerverslag
15 (6 sept)	Schrijven Afstudeerverslag Oplevering documentatie renderer Oplevering adviesrapport poort naar Linux Oplevering tests abstractie niveau Polemos Oplevering product-backlog
16 (13 sept)	Uitloop
17 (20 sept)	Uitloop

(Deze planning is gemaakt bij het begin van het project en komt niet geheel overeen met de werkelijke invulling)

4.3.1 Timeboxing

Een eigenschap van Scrum is ook dat, naast alle sprints, alle meetings 'timeboxed' zijn. Dit betekent dat er maar een bepaalde tijd voor beschikbaar is. Deze tijd is van tevoren vastgesteld met de product-owner en hangt onder andere af van de grootte van het project, het aantal projectleden en de beschikbare tijd van de product-owner.

Activiteit	Regelmaat	Timeboxed
Sprint meeting	1 / 2 weken	30 minuten
Voortgang-meeting	1 / 2 weken	30 minuten

Aangezien ik de enige ben die aan dit project heeft gewerkt, had ik gedacht ongeveer een half uur per meeting nodig te hebben. Op deze manier had ik tijdens een voortgang-meeting genoeg tijd om mijn werk laten zien en vragen stellen. Bij een sprint-meeting was een half uur ook genoeg om, naast een demonstratie te geven, te vergaderen en om een nieuwe sprint in te delen.

In dit verslag zal er per sprint ruimte zijn voor een hoofdstuk "what's on my mind wall". Dit zijn foto's/schetsen van mijn werkomgeving. Het is onderdeel van Scrum om grafieken, code, diagrammen of andere relevante informatie op te hangen zodat deze informatie gedeeld en gemakkelijk toegangbaar is. Dit vond ik ook voor mijzelf erg handig om een goed overzicht te houden over mijn werkzaamheden en om snel diagrammen en code te kunnen raadplegen. Ik heb deze foto's/schetsen in dit verslag verwerkt om u als lezer een idee te geven hoe ik invulling geef dit onderdeel van Scrum.

5. Initialisatie fase

De initialisatie-fase was de eerste fase van het project. Hierin komen onder anderen het plan van aanpak, Het onderzoek naar de verschillen tussen OpenGL en Direct3D en mijn verdieping op Scrum naar voren.

Sprint duur : 4 weken

Toen ik aan de initialisatie-fase begon, heb ik allereerst een plan van aanpak geschreven. Hierin heb ik het afstudeerplan uitgewerkt om met mijn product-owner alles te kunnen kortsluiten over mijn werkwijze. Tegelijkertijd heeft dit plan van aanpak gediend om de scope van mijn opdracht ten opzichte van de product-owner te waarborgen. In dit document staat beschreven welke methodiek ik zou gaan gebruiken en welke factoren een risico vormde voor het slagen van mijn werkzaamheden. Om dit plan van aanpak te kunnen schrijven moest ik me eerst inlezen in de methodiek Scrum, en een globaal beeld krijgen hoe de aangeleverde software in elkaar zat. Hoe ik dit gedaan heb is in dit hoofdstuk te lezen.

5.1 Beschrijving aanvang situatie

Omdat ik te maken had met een stuk software waar al jaren aan was gewerkt, ben ik allereerst gaan inventariseren wat er aanwezig was. Van de product-owner kreeg ik de broncode van Polemos, wat digitale documentatie, de product- en sprint-backlogs en een aantal burndown charts van vorige sprints. Omdat Scrum al eerder was gebruikt bij het ontwikkelen van Polemos, heb ik overlegd met de product-owner over de aansluiting van mijn project hierop. Een van de aanpassingen die ik daarom heb doorgevoerd was de nummering van de sprint. Deze beginnen daarom niet bij 1 maar bij nummer 6. Er waren namelijk al 5 scrum iteraties voor mijn project geweest.

Hierna heb ik de broncode geïmporteerd in mijn programmeeromgeving. De programmeeromgeving die ik gebruikt heb voor dit project was Visual Studio 2005. Ik had de keus hiervoor gemaakt omdat in vorige sprints ook gebruik werd gemaakt van Visual Studio 2005. Daarnaast wist ik uit ervaring door eerdere blokken die ik gevolgd heb over 3D engines, dat het overzetten van projecten van Visual Studio 2005 naar een nieuwere versie van Visual Studio een tijdrovend proces is. Daarnaast werd dit me ook sterk afgeraden door mijn bedrijfsmentor wat mijn keuze bevestigde.

Na verdere documentatie gelezen te hebben, ontdekte ik dat ik Cg geïnstalleerd moest hebben om de engine te kunnen draaien. Zoals in de opdrachtomschrijving staat beschreven, wordt Cg gebruikt voor shaders. Deze heb ik dan ook in mijn programmeeromgeving toegevoegd om zo kreeg ik Polemos op mijn computer systeem draaien.

5.2 Verdiepen Scrum

Aangezien ik nog geen ervaring had met Scrum moest ik me hier in gaan verdiepen. Eerst ben ik op internet gaan kijken naar wat Scrum precies is hoe dit kan worden gebruikt. Ook heb ik het boek *Agile software development. Principles, Patterns , and practices* gelezen. Dit boek gaat over een verzameling van best-practices (Agile) waaronder Scrum valt.

5.2.1 XP-dev

XP-dev is een tool die ik gebruikt heb om scrum documentatie mee te beheren. Deze tool kan, onder andere uit items op een sprint-backlog, burndown charts genereren. Via deze tool houd ik ook de sprint-backlogs bij en kon de product-owner mijn voorgang bekijken.

Ook bevat XP-dev online webspace waarin ik alle broncode van Polemos heb ondergebracht. Hierdoor is er één centrale plek voor alle code. Hierdoor wordt het project gemakkelijker onderhoudbaar, kunnen andere gebruikers van deze tool mijn voortgang bekijken en kan de broncode worden gedownload

5.3 Onderzoek Direct3D library

Kennis van de Direct3D library had ik al voordat ik aan dit project was begonnen. Ik had al eerder een blok op de academie voor ICT & Media gevolgd over het bouwen van een 3D engine met behulp van Direct3D. Het doel van dit onderzoek was om meer kennis op te doen van de Direct3D library en hoe Polemos Direct3D gebruikt. Dit zou vooral later in het project van toepassing zijn. Om te begrijpen hoe Polemos omging met Direct3D heb ik broncode van de engine doorgelezen en externe bronnen geraadpleegd om de structuur hierachter te begrijpen. Hieronder staan een aantal facetten van Polemos waar ik achter kwam:

Polemos…

- Heeft ±40.000 regels aan broncode
- Bevat 15 verdeelde stukken software waarin verschillende externe libraries worden gebruikt (onder andere Direct3D en Cg)
- Ondersteund het gebruik van shaders

Afbeelding 4: Het Polemos logo

Doordat ik al meer ervaring had met Direct3D heb ik meer tijd besteed aan het OpenGL onderzoek dan Direct3D.

5.4 Onderzoek OpenGL library

Om te beginnen met OpenGL heb ik een aantal boeken geraadpleegd. Waar ik het meeste informatie uit heb gehaald is *Beginning OpenGL Game programming*. Voor dit boek is geen voorkennis van het programmeren op een grafische kaart vereist en het begint bij de basisprincipes van OpenGL. Daarnaast heb ik veel informatie uit het boek *OpenGL Programming guide (Red book)* kunnen halen (zie hoofdstuk: gebruikte literatuur).

Om meer praktische kennis op te doen over de OpenGL library, ben ik op zoek gegaan naar tutorials. De website van OpenGL zelf, leek me de beste optie om te beginnen met het zoeken hiernaar. Omdat ik

geen voorkennis had van OpenGL wilde ik het liefst een tutorial die begint bij de basisprincipes om zo de structuur van OpenGL gemakkelijker te kunnen onderzoeken.

Na de website van OpenGL te hebben doorgekeken heb ik een tutorial gedownload die een driehoek in een scherm liet zien met verschillende kleuren. De broncode heb ik hierna aangepast om te bekijken hoe OpenGL hierop reageert en zo stapsgewijs de structuur van OpenGL te onderzoeken.

Hieronder staat een voorbeeld van een stukje broncode dat een vierkant met verschillende kleuren kan tekenen.

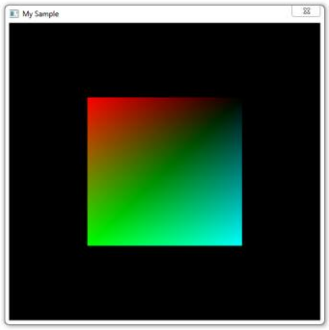
Code	Resultaat

```
glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );
glClear( GL_COLOR_BUFFER_BIT );

glPushMatrix();
glRotatef( theta, 0.0f, 1.0f, 0.0f );
glBegin( GL_QUADS );

// Top Left
glVertex3f(-0.5f, 0.5f, 0.0f);glColor3f(0.0f,0.0f,0.0f);
// Top Right
glVertex3f( 0.5f, 0.5f, 0.0f);glColor3f(0.0f,1.0f,1.0f);
// Bottom Right
glVertex3f( 0.5f,-0.5f, 0.0f);glColor3f(0.0f,1.0f,0.0f);
// Bottom Left
glVertex3f(-0.5f,-0.5f, 0.0f);glColor3f(1.0f,0.0f,0.0f);
glColor3f(0.0f,0.0f,0.0f);

glEnd();
glPopMatrix();
```




```
glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );
glClear( GL_COLOR_BUFFER_BIT );

glPushMatrix();
glRotatef( theta, 0.0f, 1.0f, 0.0f );
glBegin( GL_TRIANGLES );

// Top
glVertex3f( 0.5f, 0.5f, 0.0f);glColor3f(0.0f,1.0f,0.0f);
// Bottom Light
glVertex3f( 0.5f,-0.5f, 0.0f);glColor3f(1.0f,1.0f,1.0f);
// Bottom Left
glVertex3f(-0.5f,-0.5f, 0.0f);glColor3f(1.0f,0.0f,0.0f);

glColor3f(0.0f,0.0f,0.0f);

glEnd();
glPopMatrix();
```

5.5 Onderzoek Verschillende Direct3D en OpenGL

Nadat ik onderzoek had gedaan naar de twee libraries afzonderlijk, ben ik begonnen met onderzoek te verrichten naar waarin OpenGL en Direct3D van elkaar verschillen. Allereerst ben ik me gaan verdiepen in OpenGL. Ik heb hiervoor gebruik gemaakt van de *OpenGL Programming guide* (zie gebruikte literatuur), het internet en mijn ervaring met het gebruik van OpenGL in de tutorials. Tijdens het schrijven van dit document was ik ook nog bezig met deze tutorials. Dit leek me een goed idee zodat ik alle verschillen tussen OpenGL en Direct3D tegenkwam gelijk in dit document zou kunnen verwerken. Uiteraard zou dit document gedurende het verdere verloop van het project worden aangevuld met verschillen die ik tegen zou gaan komen (zie hoofdstuk: sprint 5). Deze verschillen onderscheiden zich in:

- Architecturale verschillen
  - Onder deze verschillen vallen die kenmerkend zijn voor de gekozen architectuur van de library. Voorbeelden hiervan zijn de keuzes in de OpenSource richting of backward compatibility.
- Technische verschillen
  - Hieronder vallen de verschillen waar de programmeur mee te maken krijgt. Hier worden vragen beantwoord die Direct3D programmeurs kunnen hebben bij het gebruik van OpenGL.

Tijdens deze fase was ik nog niet begonnen met de implementatie van de OpenGL renderer en wist ik dus niet erg veel technische verschillen. Later in het project werd dit document dus voornamelijk aangevuld met meer technische verschillen.

Voordat ik dit document begon te schrijven heb ik eerst opgemerkt voor welke doelgroep dit document geschikt moest zijn. Na overleg met de product-owner hierover kwam ik tot de volgende criteria:

De lezer van het document…

- Studeert Informatica of technische informatica
- Zit minstens in het tweede jaar van zijn/haar opleiding
- Heeft het eerste blok van de minors over 3D engines succesvol afgerond.

Omdat ik de doelgroep gespecificeerd had, kon ik mijn schrijfwijze hierop aanpassen. Zo kon ik dankzij de voorkennis die de lezer in ieder geval heeft, dieper op de stof ingaan.

5.6 Opstellen requirements

Nu ik in mijn plan van aanpak had beschreven wat er aan producten moest worden opgeleverd kon ik deze opdelen in requirements. Vervolgens heb ik deze requirements weer als items voor op de product-backlog opgenomen. Deze ‘vertaalslag’ heb ik gemaakt om de producten, indien mogelijk, op te delen in kleinere (tussen)producten zodat deze in een sprint kunnen worden ingedeeld.

Om niet te gedetailleerd te plannen heb ik ervoor gekozen om niet alle requirements op te stellen. Producten die pas later in het project aan bod komen volgens de globale planning, heb ik binnen de

initialisatie-fase nog buiten beschouwing gelaten. Dit omdat ik anders het project te veel in zou plannen en niet meer flexibel de sprints in zou kunnen delen.

De mijlpalen die bij van de planning van sprints (deze beginnen in week 5) naar voren komen zijn:

- (1) Er zijn unittests van de renderers aanwezig om de functionaliteit hiervan te waarborgen.
- (2) Er is een OpenGL implementatie aanwezig

Om de unittests (1) te verdelen in requirements heb ik gekeken naar de structuur van de Polemos renderer. Deze bevatte een kleine 40 functies (broncode met aparte functionaliteit). Ik heb er toen voor gekozen losse unittests te gaan schrijven die per test, één functie testen. Ik had er ook voor kunnen kiezen alle functies direct in één functie te testen. Dit heb ik niet gedaan omdat dan de code erg on gestructureerd in de unittest zou komen te staan. Het schrijven van één test per functie is ook veel leesbaarder. Mochten na mijn project er nog studenten zijn die verder gaan met de unittest uit Polemos, dan zullen deze veel meer aan deze optie hebben.

Welke functies ik in ieder moest gaan testen, heb ik gehaald uit de mijlpalen waar de volgende functionaliteitseisen van de renderer staan:

Polemos…

- Is in staat om OpenGL te gebruiken om een window op het scherm te tonen.
- Kan de kleur van de content van het window kan worden aangepast.
- Heeft een volledig abstracte renderer.

Dankzij gesprekken met mijn bedrijfsmentor en door mijn opgedane kennis van Direct3D kon ik deze eisen omzetten in functies die moeten worden aangeroepen binnen Polemos. Dit zijn de functies: *initialize*, *clear*, *beginscene*, *endscene* en *present*.

Deze functies heb ik daarom ook opgenomen in de advies product-backlog document:

#	Prio	Afh	Units	Requirement
1	3	-	20	Testen renderer (unittests) – GraphicsDirect3D9::Initialize
2	3	-	25	Testen renderer (unittests) - GraphicsDirect3D9::Clear
3	3	-	10	Testen renderer (unittests) – GraphicsDirect3D9::Beginscene
4	3	-	10	Testen renderer (unittests) – GraphicsDirect3D9::Endscene
5	3	-	10	Testen renderer (unittests) – GraphicsDirect3D9::Present
6	3	-	30	Testen renderer (unittests) – GraphicsDirect3D9::createRenderTarget
7	3	-	30	Testen renderer (unittests) – GraphicsDirect3D9::SetActiveRenderTarget

Prio = Prioriteit van het item
afh = Afhankelijkheid van ander item
Units = Het aantal units dat dit item zal gaan kosten

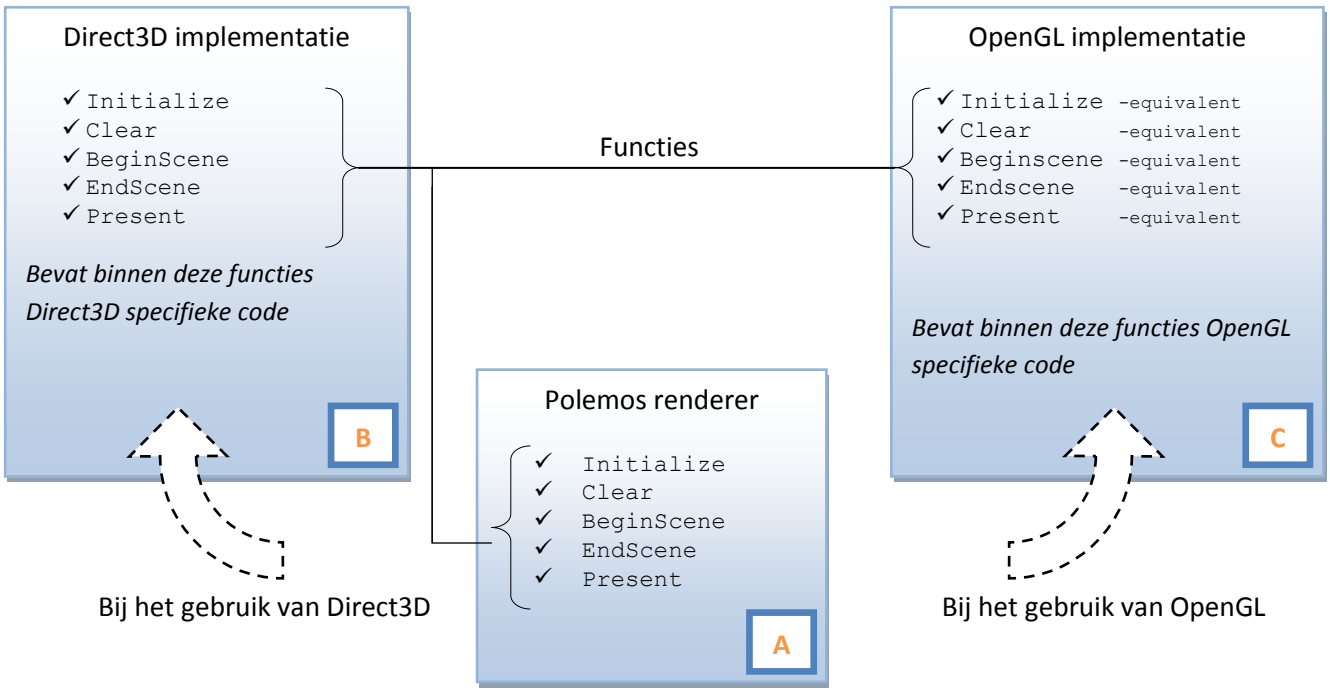
De *createRenderTarget* en de *SetActiveRenderTarget* zijn functies die de bijkomend belang waren in het project. Deze onderdelen van de renderer waren ook alvast opgenomen in de renderer voor het geval de implementatie van de andere functies sneller zouden verlopen dan gedacht. Later in het project zou blijken dat deze functies wel van fitaal belang waren om de basisfunctionaliteit te laten werken (zie sprint 3: uitvoeringen).

De units van deze requirements kwamen voort uit mijn persoonlijke kennis van unittests. Om deze schatting nog preciezer te kunnen maken had ik me ingelezen in het gebruik van CXXtest en mensen vragen gesteld die eerder met dit softwarepakket hadden gewerkt over hun ervaring. Daarnaast beoordeelde de product-owner altijd de inschatting van deze units voordat deze op de product-backlog kwamen zodat een zo goed mogelijke schatting tot stand komt.

De prioriteit heb ik hoog ingedeeld omdat het testen van de renderer van groot belang is voor mijn project. Daarbij is het niet de allerhoogste prioriteit omdat het testen van de renderer niet het doel is van mijn project, dit is echter de OpenGL renderer implementatie.

De requirements van de implementatie van de renderer heb ik op bijna dezelfde manier uitgevoerd als die van de unittests. Ik heb hier alleen niet gebruik gemaakt van de namen van de functies van de Polemos renderer, maar de Direct3D functies die aangeroepen worden binnen de engine. Dit omdat sommige functies binnen Polemos meerdere Direct3D functies bevatten waardoor de implementatie van één Polemos-functie daardoor zo ingewikkeld kan worden dat (door de hoeveelheid units) deze niet meer in één enkele sprint ingedeeld kan worden.

#	Prio	Afh	Units	Requirement
17	4	#1 -7	200	Implementatie OpenGL – OpenGL initialisatie (GL_initgraphics)
18	4	#1 -7	150	Implementatie OpenGL – OpenGL Clear (glClear)
19	4	#1 -7	50	Implementatie OpenGL – OpenGL beginscene (...)
20	4	#1 -7	50	Implementatie OpenGL – OpenGL endscene (...)
21	4	#1 -7	150	Implementatie OpenGL – OpenGL present (popmatrix / swapBuffers / bindbuffer)
22	4	#1 -7	200	Implementatie OpenGL – OpenGL setStreamSource (glVertexPointer/glAttribPointer)



Afbeelding 5: Schematische weergave Polemos renderer functies en implementaties

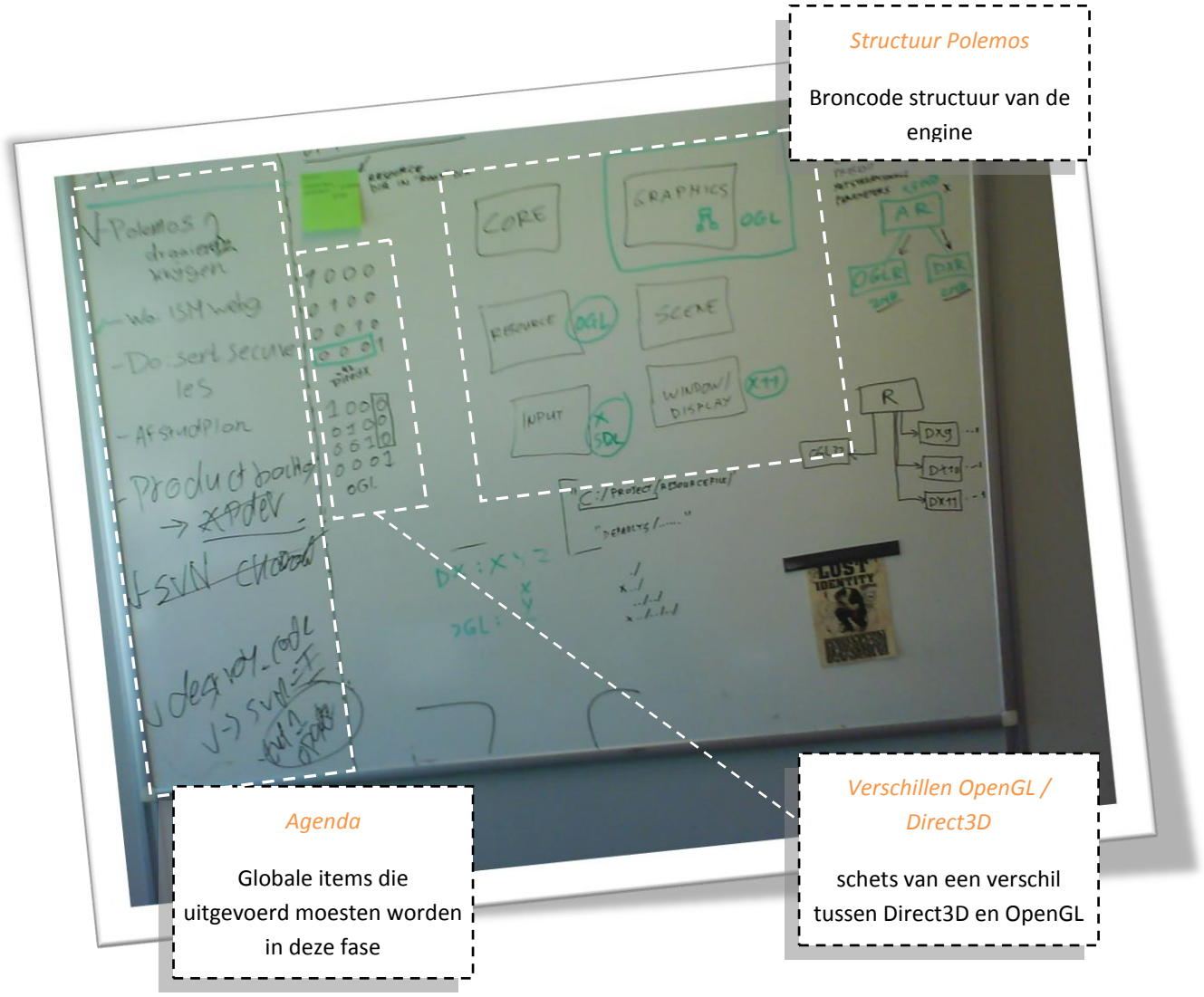
Zoals ook op afbeelding 5 te zien is, heeft de Polemos renderer (A) dezelfde namen voor de functies als de Direct3D implementatie (B). Tijdens het implementeren van de Direct3D renderer (B), is blijkbaar geen rekening gehouden met een eventuele uitbreiding naar een OpenGL renderer implementatie (C).

Als mijlpaal is ook opgenomen in het plan van aanpak “Er is documentatie aanwezig van de aangepast delen van Polemos”. De vorm van deze documentatie, zoals de diagrammen, had ik hierbij nog niet gespecificeerd. Dit zodat ik deze tijdens het project het beste zou kunnen aansluiten bij de te beschrijven implementatie.

#	Prio	Afh	Units	Requirement
9	2	#11	10	Diagram huidige implementatie
10	2	-	20	Diagram structuur OpenGL implementatie
11	3	-	160	Documentatie OpenGL renderer

5.7 What's on my mind wall

In de volgende afbeelding is te zien wat voor relevante informatie ik heb verzameld tijdens de initialisatie-fase. Deze informatie heb ik opgehangen of opgeschreven op een whitebord zodat ik deze gegevens snel kon raadplegen. Daarnaast zal deze informatie ook beschikbaar zijn voor eventuele studenten die in de toekomst aan Polemos zullen gaan werken.



6. Sprint 1

In dit hoofdstuk beschrijf ik mijn werkzaamheden gedurende de eerste sprint. De sprint zal vooral in het teken staan van het werken met CXXtest en het opzetten van tests voor de huidige implementatie van de renderer. Deze sprint duurde van 26 mei tot en met 4 juni.

Sprint duur : 2 weken

Allereerst zal de sprint-backlog van sprint 1 worden toegelicht , daarna licht ik de verdere werkzaamheden binnen de sprint toe en aan het einde van dit hoofdstuk staat de burndown chart.

Tijdens de eerste sprint-meeting van mijn project samen met de product-owner, had ik deze mijn requirements voorgelegd die ik in de initialisatie-fase had opgesteld. Tijdens deze meeting moesten er keuzes worden gemaakt over mijn werkzaamheden waarvoor ik acht dagen de tijd zou hebben. Hiervoor hebben we gekeken naar de globale planning van mijn project en kwamen we tot de conclusie dat de komende sprint in het teken moest komen te staan van het testen van de huidige situatie. De reden waarom ik de tests aan het begin wilde uitvoeren, was om er zeker van te zijn dat mijn werkzaamheden aan de OpenGL renderer geen negatief effect zouden kunnen hebben op de Direct3D renderer. Een manier om te kunnen nagaan of mijn werk fouten zou opleveren in de huidige implementatie, zou zijn met een unittest op de huidige Direct3D implementatie van de renderer.

Prio	(#)	Requirements	Units
3	330	Testen renderer (unittests) – GraphicsDirect3D9::initialize	20
3	329	Testen renderer (unittests) – GraphicsDirect3D9::clear	25
3	328	Testen renderer (unittests) – GraphicsDirect3D9::beginscene	10
3	327	Testen renderer (unittests) – GraphicsDirect3D9::endscene	10
3	326	Testen renderer (unittests) – GraphicsDirect3D9::present	10
3	325	Testen renderer (unittests) –GraphicsDirect3D9::createRenderTarget	30
3	324	Testen renderer (unittests) – GraphicsDirect3D9::setActiveRenderTarget	30
2	322	Diagram huidige implementatie	10
2	321	Diagram structuur OpenGL implementatie	20
4	307	Inlezen / runnen / oefenen testframework	40

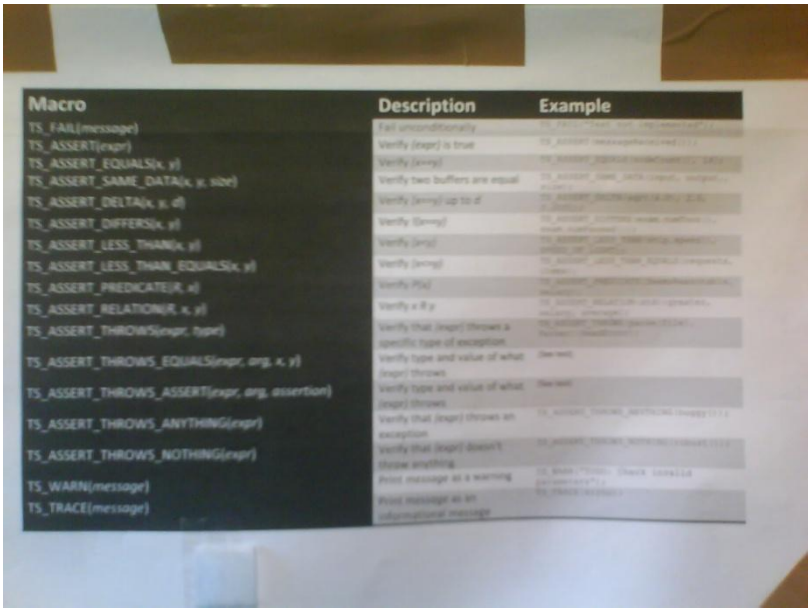
Het nummer, de prioriteit en het aantal units kunnen verschillen met die ik had opgesteld in de initialisatie fase. Dit komt doordat deze requirements beoordeeld zijn door de product-owner. Zodra deze de requirements heeft beoordeeld en eventueel aangepast, worden deze opgenomen in de product-backlog. Om duplicaties in nummers (#) te voorkomen, krijgen de requirements een ander nummer zodra deze op de product-backlog komen (zie bijlage: “product-backlog”).

6.1 Uitvoeringen

Voordat ik deze sprint in ging, wist ik al dat ik het framework CXXtest zou gaan gebruiken voor het testen van de engine. De keus voor CXXtest lag niet voor de hand. Persoonlijk had ik weinig ervaring met blackbox-testing, en geen ervaring met het opzetten van een testframework binnen Visual Studio. CXXtest kende ik dus ook niet. Ik heb me dan ook laten adviseren bij leraren, oud studenten en via online fora hoe ik het beste C++ code zou kunnen testen. Hierdoor kwam ik uit bij CXXtest. CXXtest is eigenlijk een stuk software om C++ code te kunnen testen en is geschreven in de programmeertaal Python.

Ook had ik kunnen kiezen voor whitebox-tests. Dit is een andere manier van testen. Het verschil tussen black- en whitebox testing is, dat bij whitebox testing de interne structuur van een object of stuk broncode wordt getest. Bij blackbox testing wordt de functie of broncode als geheel getest en wordt er niet gekeken naar de interne structuur van code die wordt doorlopen. Dit is ook de reden waarom ik in dit project gekozen heb om niet whitebox te testen. De tests zouden immers dienen voor het controleren van de huidige renderer, en met blackbox-testing wordt dit doel bereikt. Whitebox-testing heeft hierin geen toegevoegde waarde, mede omdat ik de renderer alleen wil testen op functionaliteit. Daarnaast duurt de uitvoering van whitebox tests veel langer dan die van blackbox-tests.

Nadat ik CXXtest had geïnstalleerd kon ik beginnen met het schrijven van de tests. CXXtest heeft zijn eigen commando's om te testen en daarom heb ik de mogelijke commando's op een rijtje gezet om zo op een zo snel mogelijke manier de juist commando's toe te kunnen passen.

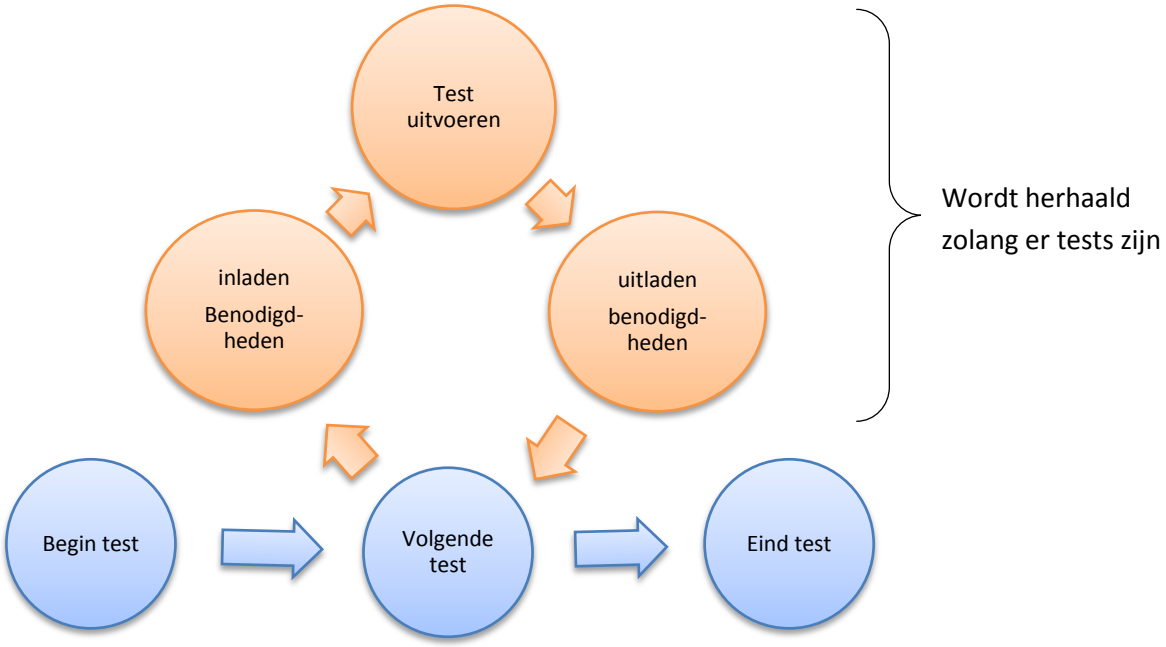


Afbeelding 6: CXXtest commando's

De meeste units (40, ongeveer één dag) in deze sprint stonden ingedeeld om kennis op te doen met dit framework (sprint-backlog item 307). Dit aantal units bleek zeker niet overdreven. Omdat dit

testframework erg uitgebreid was en omdat ik weinig documentatie tot mijn beschikking had, heb ik veel moeten uitzoeken en diep in de broncode moeten duiken om aanpassingen aan het framework te maken.

Een van de keuzes die ik moest maken bij het creëren van de tests kwam naar voren bij de eerste test (*GraphicsDirect3D9::Initialize*). Voorafgaande aan deze functie, moeten er een aantal andere functies worden aangeroepen. Hierna kan de functie *initialize* worden getest. Zodra de functie *initalize* is getest heeft kan de functie *beginscene* pas worden getest. Om de functie *endscene* aan te roepen moet eerst *beginscene* zijn aangeroepen Et cetera. Iedere functie hangt dus af van de voorgaande functie. Om ervoor te zorgen dat toch alle tests afzonderlijk konden worden getest heb ik het volgende schema opgesteld waarin te zien is hoe de tests verlopen:



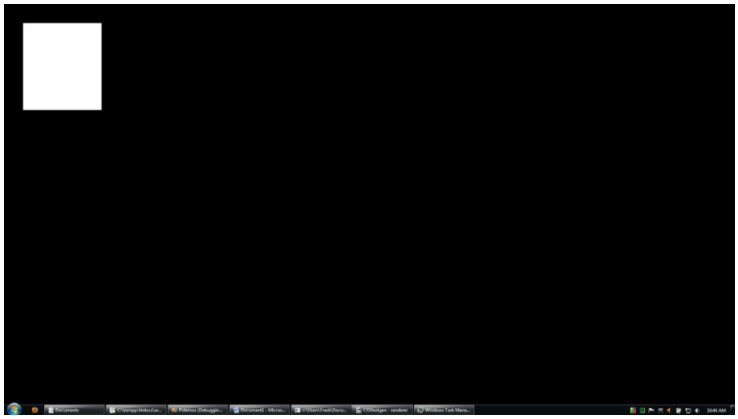
Afbeelding 7: Schematische weergave implementatie unittests

Deze methode kost een computer veel rekenkracht, maar biedt het voordeel dat iedere methode afzonderlijk wordt getest. Mocht er dus één functie van de renderer falen, dan zal er ook maar één functie worden aangegeven in plaats van alle functies die afhangen van de falende functie.

Gedurende het testen van de *initialize* -functie van de renderer kwam ik een bijzonderheid tegen. Terwijl deze functie door CXXtest werd doorlopen en zo dus getest, begon het beeld erg te flikkeren. Dit is op zich niet vreemd. Aangezien de renderer verantwoordelijk is voor het grafische gedeelte van een engine, zal deze veel communiceren met het beeldscherm om grafische output te geven. Tijdens het initialiseren van een renderer kan er dus het een output worden gegeven en dat resulteert in een flikkerend scherm. Een probleem hiervan echter was, dat het beeld zwart bleef na de test. Dit probleem deed zich echter alleen voor tijdens het aanmaken van een window (scherm) dat even groot is als het

beeldscherm zelf. Daarom heb ik dit als een item opgenomen voor op de sprint-backlog en omschreven als *Fullscreen window kunnen deleten* (item 338). Dit item zou later in sprint 2 gelijk op de sprint-backlog komen om te worden verholpen.

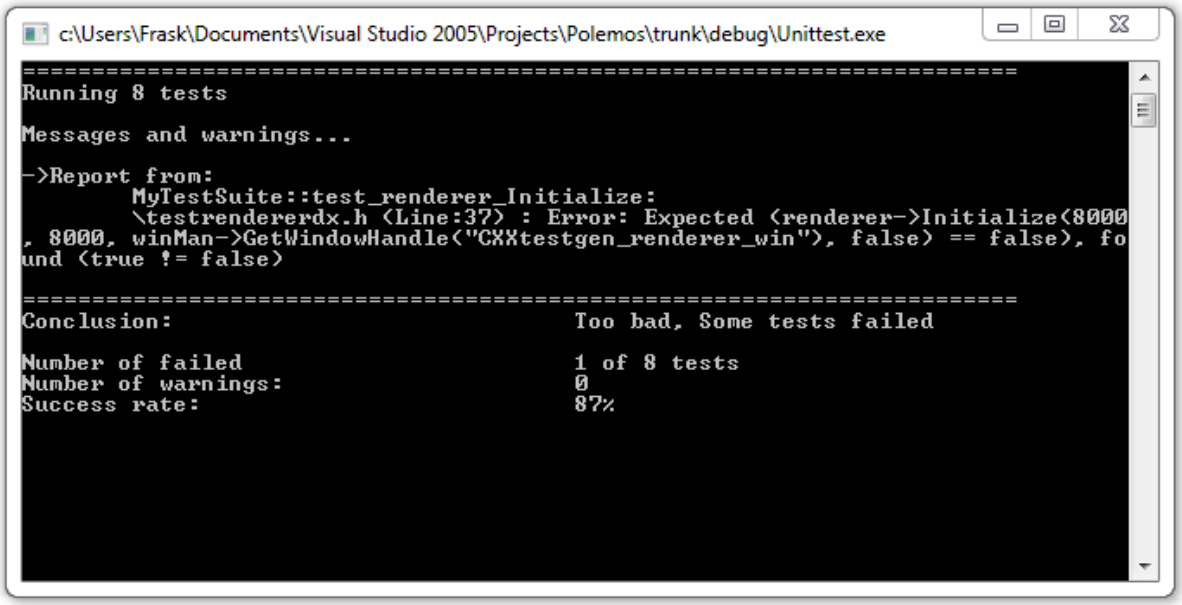
Dit probleem bleef zich voordoen en heb ik dan ook opgenomen in het product-backlog advies voor sprint 2.



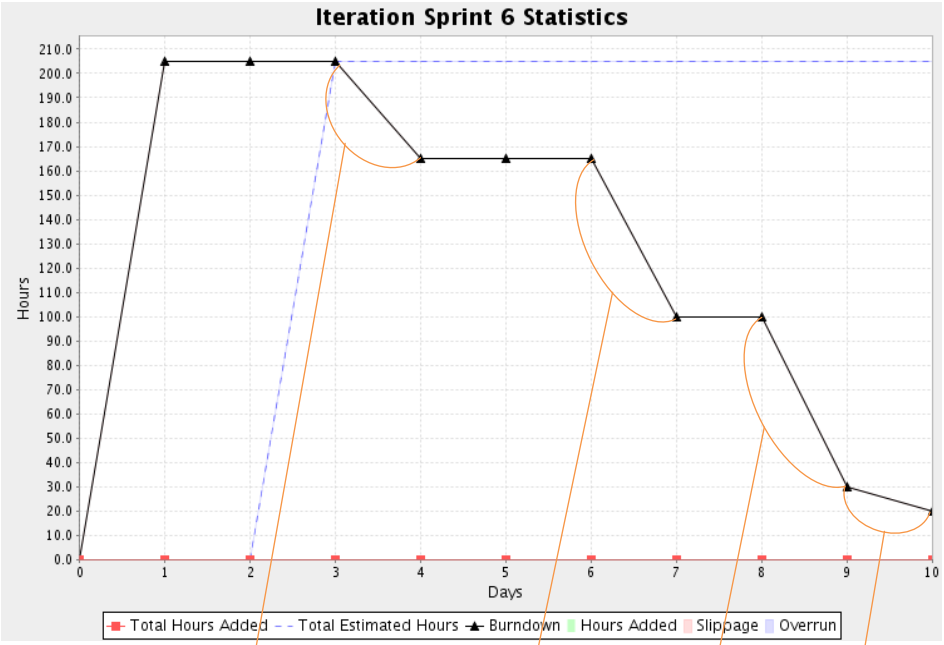
Afbeelding 8: Een unittest van de Polemos renderer

Halverwege de sprint vertelde de product-owner dat hij de uitkomsten (output) van de tests graag in een scherm getoond wilde hebben met alle tests en een indicatie of deze geslaagd was of niet. Deze eis stond niet op de sprint-backlog voor deze sprint, maar omdat dit maar een kleine aanpassing was kon ik dit wel uitvoeren. Ik heb dus een aantal aanpassingen aan het CXXtest-framework gemaakt. Als eerste heb ik de output op een andere manier in het scherm laten komen. Daarna heb ik de python broncode zo aangepast dat dit scherm op het beeld getoond werd en ook bleef staan. Zonder deze aanpassing is het scherm maar enkele seconden zichtbaar en sluit deze automatisch af.

Hieronder staat een voorbeeld van een test, uitgevoerd op de Polemos renderer. Er wordt onder andere vermeld op welke functie faalt, op welke regel dit plaatsvindt en er wordt een overzicht opgemaakt van alle tests die zijn uitgevoerd.



Afbeelding 9: Aangepaste output van CXXtest



- Inlezen / runnen / oefenen testframework

- (Test) GraphicsDirect3D9::Initialize
- (Test) GraphicsDirect3D9::Clear
- (Test) GraphicsDirect3D9::BeginScene
- (Test) GraphicsDirect3D9::EndScene

- (Test) GraphicsDirect3D9::present
- (Test) GraphicsDirect3D9::createRenderTarget
- (Test) GraphicsDirect3D9::SetActiveRenderTarget

- Diagram huidige implementatie

Toelichting

Hiernaast staat de burndown chart van Sprint 1 weergegeven. Hierin is te zien dat de sprint 10 dagen bevat met daarin 4 burndowns (dag 3-4 / 6-7 / 8-9 / 9-10). Deze vier punten zijn momenten in het project waarop ik items van de sprint-backlog de status “voltooid”. Hierna wordt er niet meer aan gewerkt en wordt het item op de product-backlog aangeduid als een uitgevoerde taak.

Wat opvalt is dat de burndown chart niet eindigt op de horizontale as. Dit betekent dat, aan het einde van de sprint, niet alle items van de sprint-backlog zijn afgerond. Dit komt doordat de product-owner item 322 (“Diagram huidige implementatie”) toch niet van toepassing vond voor de sprint. Een overzicht van de huidige Direct3D implementatie voegde weinig toe in zijn ogen en is daarom van de sprint-backlog geschrapt

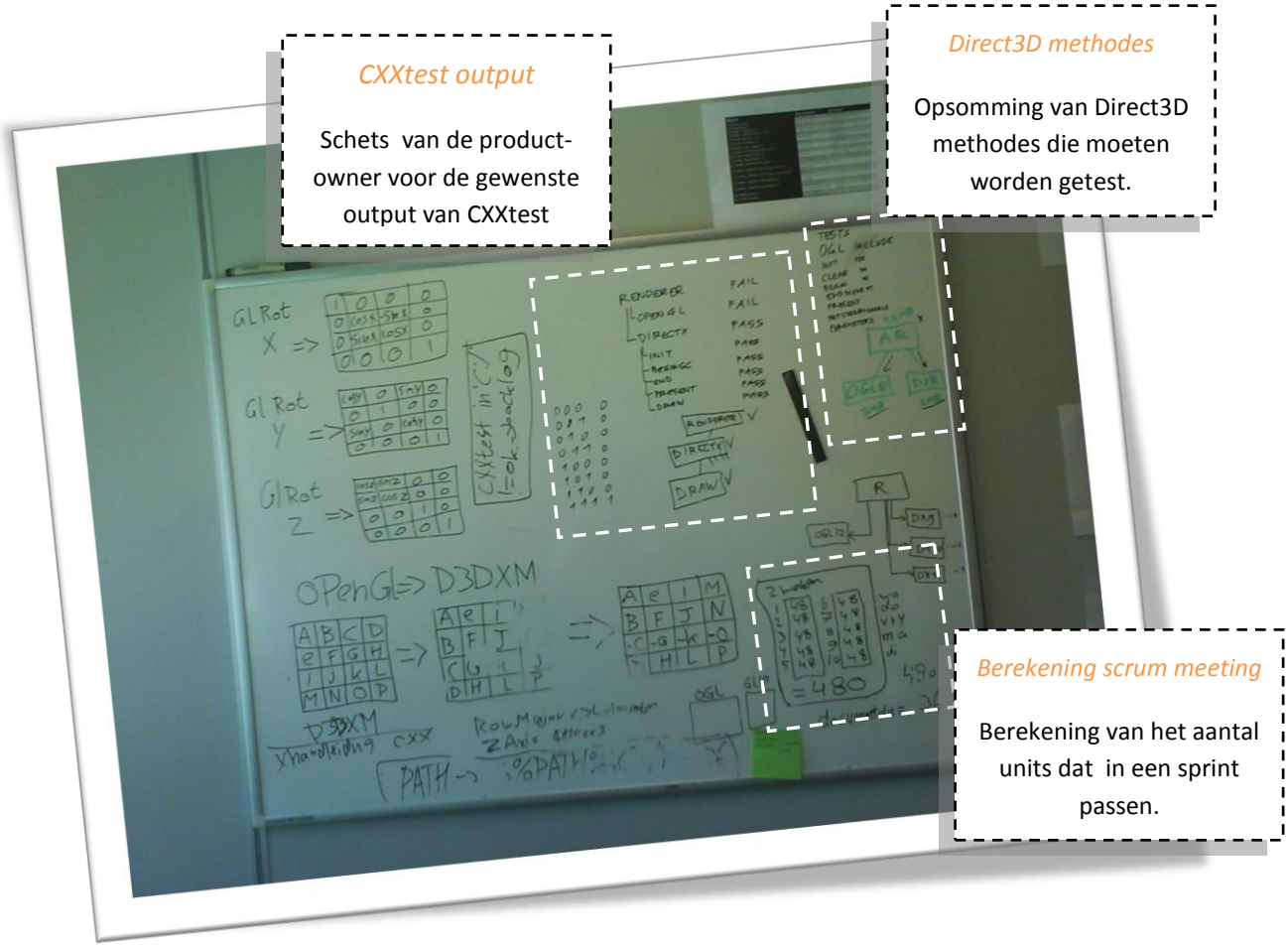
Wat niet te zien is aan de grafiek, wat tevens geldt voor alle burndown charts, is dat het weekend is opgenomen in de grafiek. Bij Scrum is dit gebruikelijk omdat er de mogelijkheid bestaat dat mensen thuis werken.

Zoals beschreven in het plan van aanpak moet, na afloop van iedere sprint, er een back-up worden gemaakt van alle producten en toevoegingen van de laatste afgeronde sprint. Tijdens het maken van deze back-up kwam ik erachter dat er in Polemos een hoop dubbele bestanden werden gebruikt. Dit werd mede veroorzaakt omdat ik, voor het gebruik van de unittests, alle externe bestanden die Polemos gebruikt (plaatjes, modellen en andere externe bestanden) moest kopiëren. Na wat dieper te hebben gezocht in de structuur, bleek dat Polemos al vaker dezelfde problemen was tegengekomen aangezien er over het hele project externe bestanden waren te vinden.

De oplossing hiervoor was ingewikkeld en viel buiten de scope. Toch hebben de product-owner en ik in overleg besloten dat dit probleem moest worden opgelost. De hoeveelheid tijd die dit probleem me verder in het project zou gaan opleveren zou vele malen groter zijn dan de tijd die het zou duren dit op te lossen. Daarom hebben de product-owner en ik besloten dit probleem direct op te lossen. Dit is niet erg gebruikelijk binnen Scrum aangezien 'normaal' gesproken sprints elkaar opvolgen, maar omdat sprint 2 al was ingepland was het een beter idee om sprint 2 in zijn geheel uit te stellen. Op deze manier hoefde sprint 2 niet opnieuw te worden ingepland en kon direct aan dit probleem worden gewerkt. Het project liep op deze manier een week uit op de oorspronkelijke globale planning.

Om dit probleem verder in kaart te brengen ben ik begonnen om een diagram op te stellen waarin is verwerkt, waar alle externe bronnen (resources) zich begeven. Op deze manier kon ik Polemos zo gaan aanpassen dat de resources op één centrale plek aanwezig waren en dat elke resource door de engine gevonden kon worden. Na mijn aanpassingen te hebben doorgevoerd heb ik deze ook laten keuren door de product-owner om er zeker van te zijn dat deze er tevreden mee was.

6.3 What's on my mind wall



7. Sprint 2

Nu sprint 1 was afgerond en de verdere problemen waren opgelost kon er begonnen worden aan sprint 2. Deze sprint duurde van 14 juni tot en met 25 juni.

Sprint duur : 2 weken

Omdat er in sprint 1 tests waren opgesteld kon er nu begonnen worden aan de implementatie van de OpenGL renderer. Door deze tests kon ik ervan verzekerd zijn dat mijn werkzaamheden aan een nieuwe renderer implementatie niet de huidige renderer zou laten crashen zonder hier een melding van te geven.

Na de sprint-meeting kwamen de volgende requirements op de sprint-backlog van sprint 2:

Prio	(#)	Requirements	Units
3	338	Fullscreen window kunnen deleten	30
4	337	Aanmaken OpenGL renderer project files	80
3	336	Documentatie aanpassingen CXXtest framework	30
4	315	Implementatie OpenGL – OpenGL include	20
4	314	Implementatie OpenGL – OpenGL initialisatie (GL_initgraphics)	200

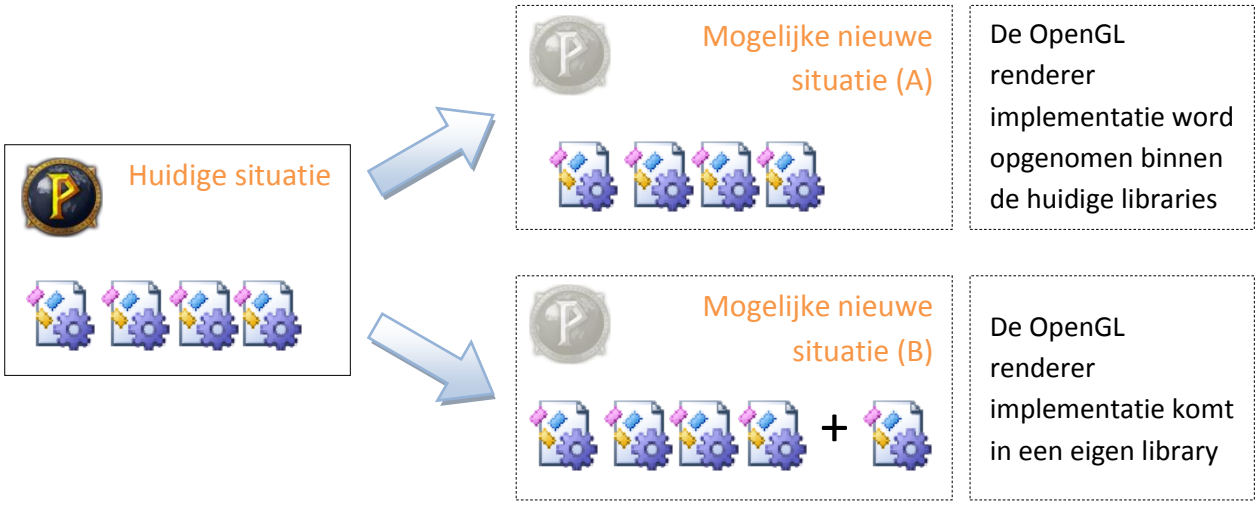
Een *Fullscreen window kunnen deleten* kwam voort uit de vorige sprint waar ik tijdens het testen achter dit probleem met het aanmaken van windows (schermen) was gekomen. Deze fout in de huidige renderer zou er eerst uit moeten zodat ik tijdens het implementeren van OpenGL niet tegen ditzelfde probleem aan zou lopen. Het *Aanmaken OpenGL renderer project files* bevat de broncode structuur en hoe ik het beste de nieuwe renderer een plek heb kunnen geven in Polemos. De *Documentatie aanpassingen CXXtest framework* bevat de aanpassingen die ik heb gemaakt op het CXXtest framework. Tijdens het inplannen van de requirements bleek dat de product-owner graag een document had waarin alle aanpassingen uitgewerkt staan die zijn gedaan op CXXtest. Item #315 en #314 waren al opgesteld in de initialisatie-fase en zijn ook in sprint 2 ingedeeld.

7.1 Uitvoeringen

Allereerst ben ik begonnen met het plannen van de taken binnen deze sprint. Omdat ik niet lang hiervoor gewerkt had aan unittests en dit dus nog helder voor ogen had, leek het me handig om te beginnen met item #338 (*Fullscreen window kunnen deleten*). Deze fout in de software kwam ik tegen tijdens het aanmaken van mijn tests in sprint 1. Zodra de tests werden gedraaid, werd het scherm zwart. Na het nalopen van de testcode kwam ik het probleem tegen. Het zwarte scherm dat verscheen, deed zich alleen voor in een bepaalde samenloop van drie variabelen. Echter, tijdens het afzonderlijk testen van deze drie variabelen, verscheen er geen zwart scherm. Na dit probleem op te hebben gezocht in de Direct3D documentatie, en de Direct3D interface te hebben getest met deze variabelen kwam ik tot de conclusie dat, met de samenloop van deze drie variabelen, het niet mogelijk is de renderer te initialiseren. Na zelf de broncode te hebben nagekeken kwam de product-owner tot dezelfde conclusie en heb ik de test die het probleem veroorzaakte gedocumenteerd in de broncode en weggehaald uit tests.

In de vorige sprint heb ik beschreven dat ik heb gewerkt met CXXtest. Binnen dit framework heb ik een aantal onderdelen veranderd, onder andere hoe de output werd weergegeven. De product-owner was hier tevreden over en wilde elk van deze aanpassingen op codeniveau gedocumenteerd hebben. De reden hiervoor was dat indien er een nieuwe versie van CXXtest uit zou komen, deze aanpassen wederom uitgevoerd zouden kunnen worden. Dit leek mij persoonlijk ook een goed idee, zo zouden de huidige unittests een stuk beter onderhouden kunnen worden over een langere tijdsperiode. Als ik de aanpassingen niet gedocumenteerd zou hebben zouden deze bij een update naar een nieuwe versie van CXXtest immers verloren gaan.

Hierna ben ik gaan beginnen met OpenGL te gaan programmeren binnen Polemos (items #314,#315 en #337). De eerste beslissingen die ik moest nemen was waar ik de OpenGL renderer binnen de engine zou plaatsen. Polemos is namelijk opgebouwd uit een aantal libraries (zie hoofdstuk: organisatie beschrijving) die in verbinding staan met elkaar. Ik had de keus om mijn huidige code binnen die bestaande libraries onder te brengen (situatie A), of om een nieuwe library aan te maken (situatie B).



Afbeelding 10: Mogelijkheden OpenGL renderer implementatie

Om hier een afweging in te kunnen maken heb ik de voor- en nadelen van beide mogelijkheden opgesteld.

Pro 'A' contra 'B'

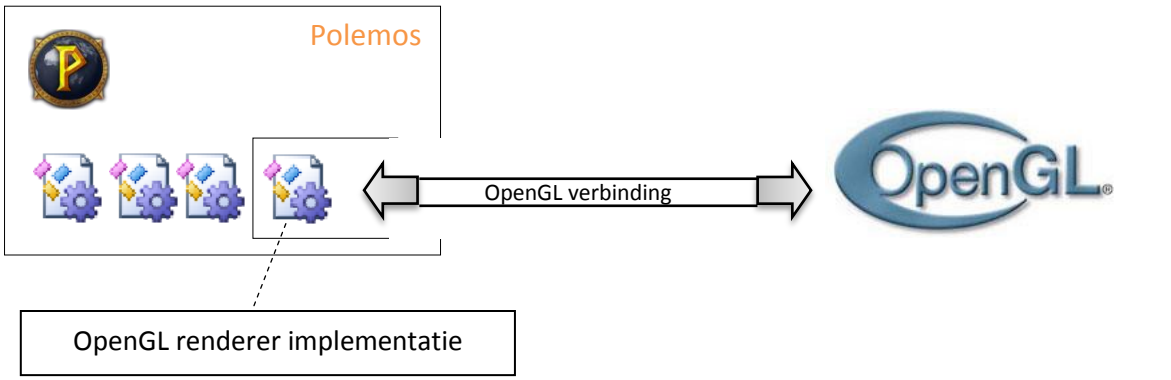
- Kost minder tijd
- Gemakkelijker te programmeren

Pro 'B' contra 'A'

- Zorgt voor betere onderhoudbaarheid doordat alle code opgedeeld is in kleinere 'brokken'
- Bewerkstelligd crossplatform functionaliteit.

Na deze te hebben opgesteld en ik erachter kwam dat via situatie 'A' de renderer niet crossplatform zou maken wist ik dat deze situatie niet de voorkeur zou hebben van de product-owner. Het was immers onderdeel van het project om een stap richting een crossplatform engine te zetten (zie hoofdstuk: Opdrachtomschrijving). Deze techniek van het opdelen van stukken software en zo crossplatform functionaliteit te bewerkstelligen wordt static-linking genoemd. Deze kennis zou ook van waarde zijn bij de eventuele implementatie in het adviesrapport van sprint 5.

Na deze architecturale beslissing, kon ik gaan programmeren in OpenGL. Aangezien er in Polemos niet werd gewerkt met OpenGL, moest er eerst een koppeling geschreven worden om OpenGL te kunnen gebruiken. Deze koppeling bestaat uit een aantal regels broncode die zich in het Polemos project bevinden.



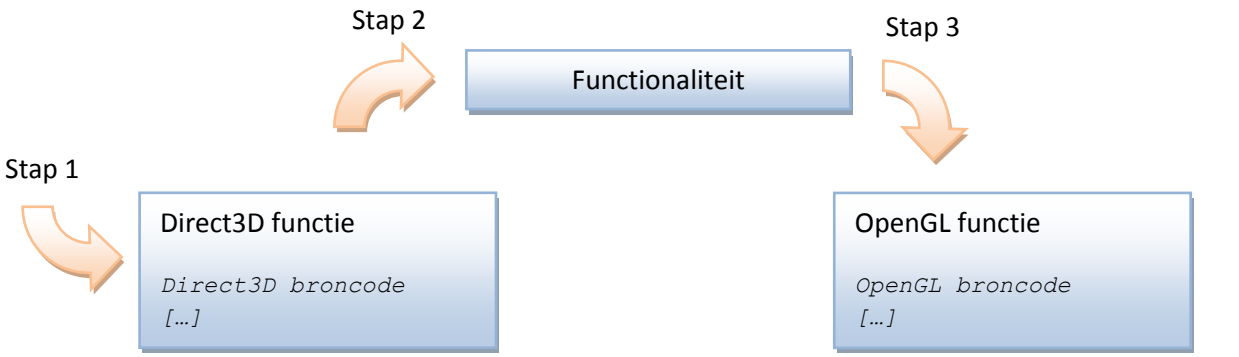
Afbeelding 11: Koppeling Polemos en OpenGL

Zodat te zien is in deze afbeelding, wordt de OpenGL verbinding maar in één stuk van de engine gemaakt. Dit zorgt voor de crossplatform functionaliteit zoals eerder beschreven in dit hoofdstuk. Tijdens het ontwikkelen van deze koppeling heb ik extra informatie opgezocht over de OpenGL versie die gebruikt wordt op mijn computer. Dit om deze informatie te kunnen documenteren en om latere problemen eventueel te kunnen adresseren. De OpenGL versie die gebruikt wordt op een computersysteem bepaald de functionaliteit die de programmeur kan gebruiken. Zodra ik dus gebruik zou maken van functionaliteit die niet beschikbaar zou zijn onder de OpenGL versie die op mijn

computersysteem beschikbaar is, zou de software niet werken terwijl deze wel zou werken onder een andere OpenGL versie.

Om achter deze informatie te komen heb ik het programma “OpenGL Extension Viewer” gebruikt.

Als laatste onderdeel van de sprint ben ik begonnen met de functie *Initialize* van de OpenGL renderer. In Polemos is dit de eerste functie die uitgevoerd moet worden om de renderer te kunnen gebruiken. Om duidelijk te krijgen wat er precies in deze functie worden uitgevoerd aan broncode, heb ik de Direct3D renderer als referentiekader gebruikt. Ik heb de functie *Initialize* uit de Direct3D renderer onderzocht en hierbij gekeken naar alle functionaliteit die de code bewerkstelligd. Door mijn kennis van Direct3D en OpenGL kon ik dezelfde functionaliteit die binnen de Direct3D renderer wordt toegepast vertalen naar dezelfde functionaliteit voor de OpenGL renderer. Hieronder staat het stappenplan dat ik heb toegepast voor de functie *initialize* en daarna ook voor alle andere functies.



Afbeelding 12: Globaal Stappenplan functioneel OpenGL implementatie

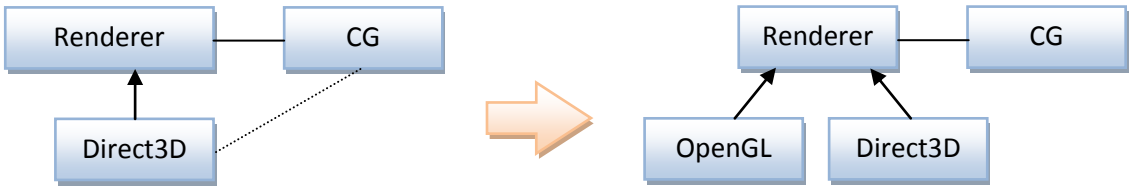
- Stap 1: Onderzoek naar functie
  - Ik begin met de broncode te onderzoeken van de Direct3D renderer. In deze functie probeer ik helder te krijgen welke variabelen worden gemanipuleerd en aangemaakt
- Stap 2: bepalen functionaliteit
  - Hierna probeer ik de structuur hiervan te begrijpen en deze te categoriseren tot functionaliteit. Een voorbeeld hiervan bij de functie *initialize* is:

    - Het aanmaken van een window (scherm)

    Voor deze functionaliteit zijn meerdere regels broncode nodig en het is één van de functionaliteiten van de functie *initialize*.
- Stap 3: vertaalslag maken van functionaliteit naar OpenGL implementatie

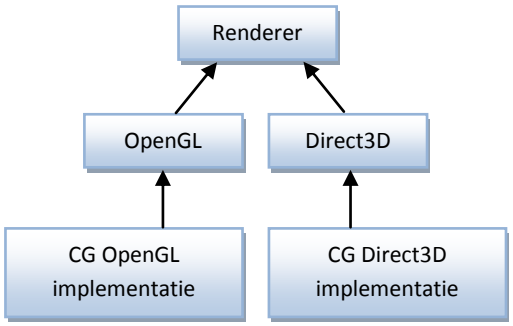
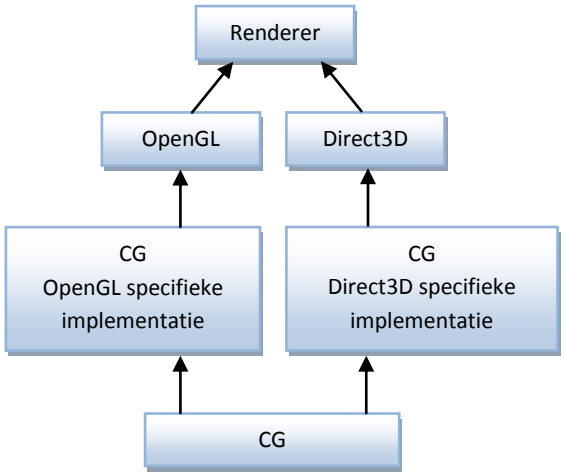
- Zodra ik de functionaliteit van een functie helder heb (dit kunnen er uiteraard meer zijn dan één) programmeer ik dezelfde functionaliteit in OpenGL.

Een van de problemen die ik tijdens het bepalen van de functionaliteit tegenkwam was de functionaliteit voor het gebruik van Cg. Cg is een aparte programmeertaal die in Polemos wordt gebruikt voor het berekenen van shaders (zie hoofdstuk: Opdrachtomschrijving). De structuur die in de doelstelling van dit project al naar voren kwam over het loskoppelen van Cg van de renderer kwam dus nu ter sprake. Hieronder nogmaals de doelstelling voor het gebruik van Cg:

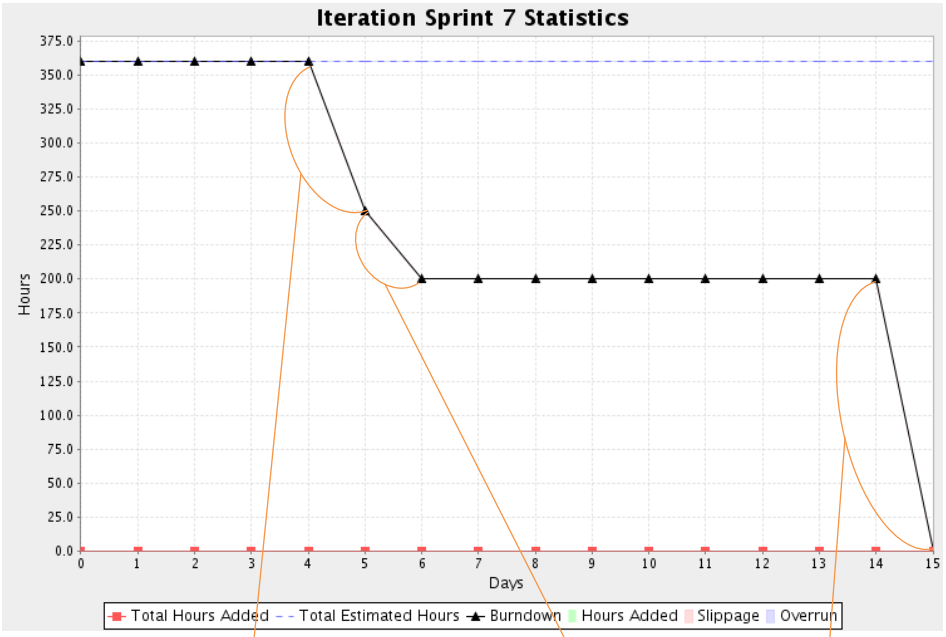


Afbeelding 13: Globale doelstelling

Om te onderzoeken of de nieuwe situatie mogelijk is, heb ik dit eerst onderzocht. Hiervoor heb ik de functies die Cg gebruikt bekeken en onderzocht in hoeverre deze afhankelijk waren van de renderer. Na de documentatie van Cg te hebben bekeken en overleg te hebben gehad met de product-owner bleek de nieuwe situatie niet te kunnen. Cg is renderer-API specifiek, wat betekent dat iedere renderer-API op zijn eigen manier Cg benaderd in de broncode. Omdat deze loskoppeling hierdoor onmogelijk wordt gemaakt, ben ik op zoek gegaan naar alternatieve oplossingen. Door mijn ervaring met C++ code programmeren kwam ik op de volgende twee manieren: (zie volgende pagina)

Situatie A	Situatie B
	
Voordelen: ○ Snel te programmeren (tijdwinst)	Voordelen: ○ Minder duplicatie van code ○ Op lange termijn goed te onderhouden (weinig om aan te passen)

Het verschil tussen situatie ‘A’ en situatie ‘B’ is dat bij ‘A’ er een gehele aparte implementatie is van Cg Voor Direct3D en een aparte voor OpenGL. Bij Situatie ‘B’ worden alle functies die niet afhankelijk zijn van de renderer, apart verzameld zodat beide renderers er gebruik van kunnen maken. Dit zorgt ervoor dat erin verhouding met situatie ‘A’ weinig duplicatie van broncode is. De voordelen van situatie ‘A’ wegen niet op tegen de voordelen van situatie ‘B’ omdat er in de toekomst erg veel zou moeten worden aangepast om situatie ‘A’ te laten functioneren.



- Fullscreen window kunnen deleten
- Aanmaken OpenGL renderer project files

- Implementatie OpenGL – OpenGL include
- Documentatie aanpassingen CXXtest

- Implementatie OpenGL – OpenGL initialisatie (GL_initgraphics)

Toelichting

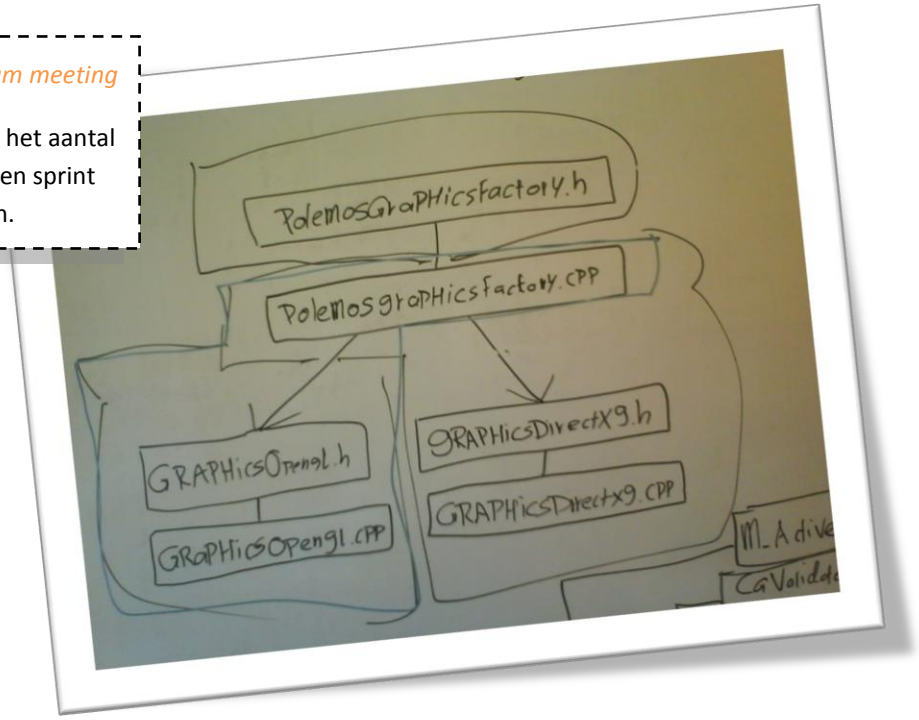
Hiernaast is de burndown chart weergegeven van sprint 2. De sprint telt hierin 15 dagen omdat ik de sprint al had ingedeeld op de vrijdag van sprint 1. Hierdoor komen er op de burndown chart 3 dagen bij (dag 1,2 en 3). Verder heeft de grafiek een uitzonderlijke vorm. Dit komt doordat ik, in verhouding, lang bezig ben geweest met sprint-backlog item 314 (“Implementatie OpenGL – OpenGL initialisatie”). De andere items waren binnen veel korte tijd voltooid.

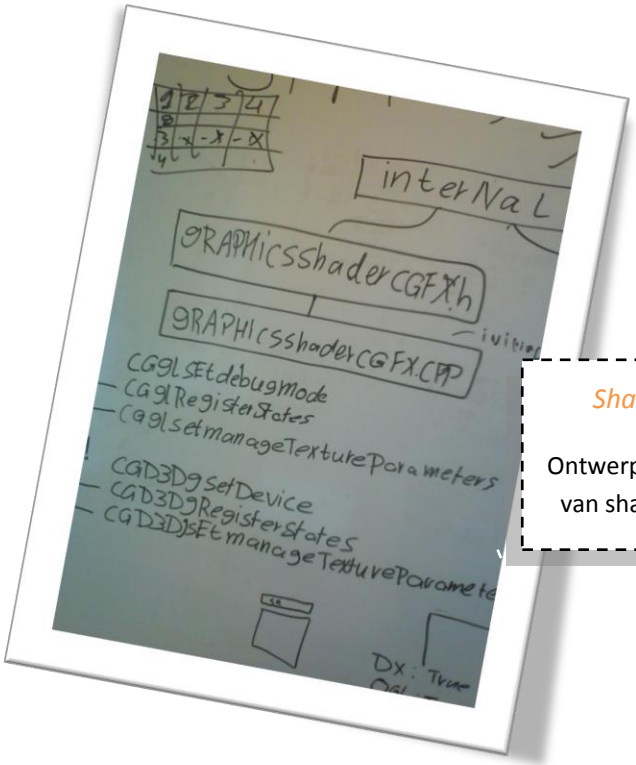
De burndown chart eindigt op de horizontale as wat betekend dat alle items van de sprint zijn uitgevoerd en voltooid.

7.3 What's on my mind wall

Berekening scrum meeting

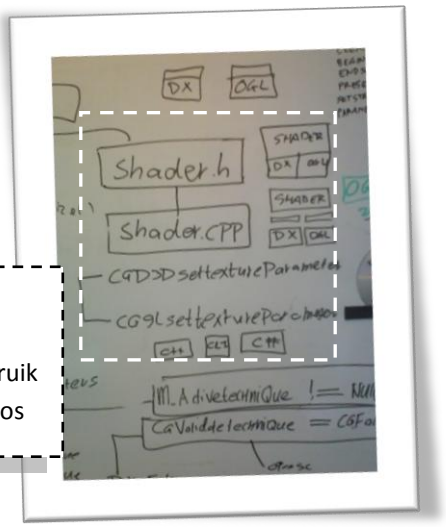
Berekening van het aantal units dat in een sprint passen.





Shader ontwerp

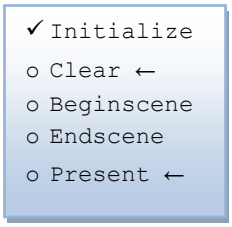
Ontwerp voor het gebruik van shaders in Polemos



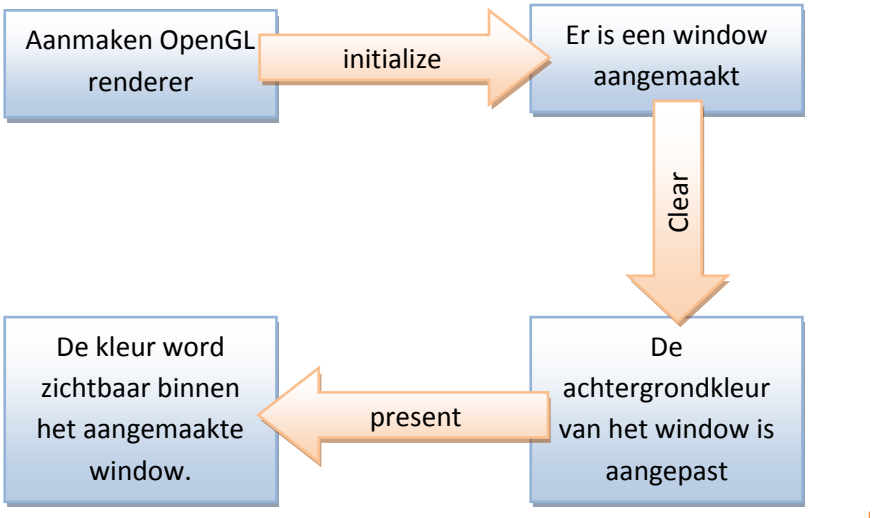
8. Sprint 3

Nu er een begin was gemaakt met OpenGL in de 3D engine Polemos, kon ik hierin verder gaan. Deze sprint zal dan ook in het teken staan van het verder programmeren aan de OpenGL renderer. Deze sprint duurde van 28 juni tot en met 9 juli.

Sprint duur : 2 weken



Van de vijf functies die verantwoordelijk zijn voor de basisfunctionaliteit voor OpenGL binnen Polemos was nu de eerste afgerond. De implementatie van de OpenGL renderer had in sprint 2 een *initialize* functie gekregen waardoor een scherm (window) aangemaakt kon worden. Nu kon er verder worden gegaan met het implementeren van de rest van de basisfuncties (*clear*, *beginscene*, *endscene* en *present*). Omdat al deze functies niet in een sprint passen moest ik samen met m de product-owner een keus maken over welke functies we konden inplannen voor sprint 3. We hebben daarbij gekozen voor de functies *clear* en *present*. Dit kwam doordat deze twee functies het snelste zichtbaar resultaat op zouden leveren. *Initialize, clear* en *present* samen bevatten namelijk de functionaliteit om een scherm (windows) te openen en hierop een kleur te kunnen laten zien. Hierdoor zou ik ook gelijk de functie *initialize* kunnen testen op functionaliteit. Deze *initialize* -functie moet immers eerst worden doorlopen om een window te creëren die vervolgens door de functie *clear* kan worden veranderd van kleur. Daarna kan door middel van de *present* -functie ook daadwerkelijk de kleur zichtbaar worden die het window heeft gekregen. Voordat de *present* -functie wordt aangeroepen heeft het window wel een andere kleur maar wordt deze nog niet op het scherm getoond.



Afbeelding 14: Initialize, Clear en present functies renderer

Het lag dus voor de hand deze twee functies in de sprint-backlog op te nemen.

Aangezien ik deze sprint veel units aan de OpenGL renderer implementatie zou gaan besteden heb ik in overleg met de product-owner het besluit genomen om al mijn bevindingen, synchroon met de ontwikkeling, te documenteren. De rendererdocumentatie zou een document worden wat andere studenten die aan Polemos zullen gaan werken als doelgroep had. Het grootste voordeel van het synchroon ontwikkelen en documenteren van de OpenGL renderer implementatie was dat ik de aspecten die ik tegenkwam preciezer kon beschrijven dan als ik de documentatie achteraf zou maken. Aangezien Polemos en OpenGL grote stukken software zijn, is het lastig om alles in detail te omschrijven na een aantal dagen tijd.

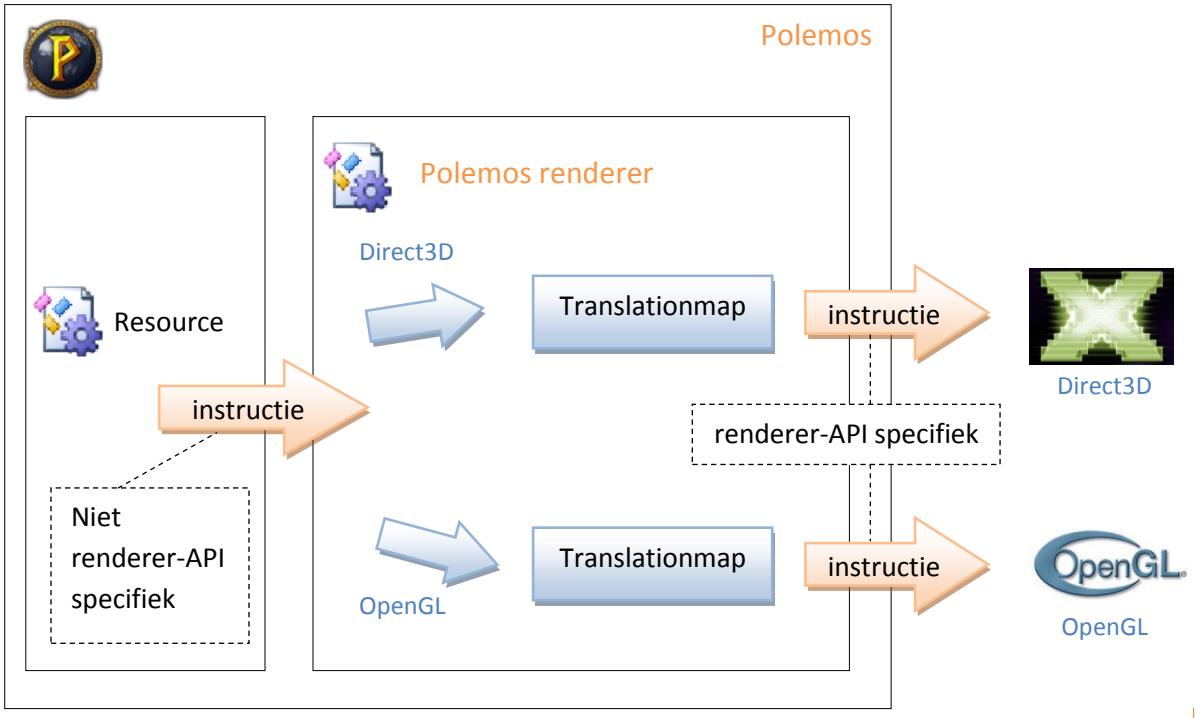
Wat in de vorige sprints niet aan bod was gekomen was de sprint-buffer. Deze buffer maakt officieel onderdeel uit van Scrum en is eigenlijk een tweede sprint-backlog waarin items worden opgenomen waaraan gewerkt kan worden indien alle items van een sprint-backlog zijn voltooid. Uiteraard geld dit alleen waarneer de tijd die staat voor een sprint nog niet voorbij is. Deze sprint-buffer kan vol worden geplant met een maximaal aantal units dat gelijk staat de units die zijn ingeplant voor de sprint-backlog. De reden waarom we deze Scrum-buffer zijn gaan gebruiken is omdat alle sprints tot dusver precies uitkwamen qua units. De kans was dus groot dat er een sprint zich zou voordoen waarbij ik een aantal dagen te vroeg klaar zou kunnen zijn. Om ervoor te zorgen dat ik op zo'n moment ik wel productief door zou kunnen werken aan het project, is deze sprint-buffer in het project toepast.

Prio	(#)	Requirements	Units
2	332	OpenGL::CreateRendertarget	50
4	313	OpenGL::clear	50
4	310	OpenGL::present	100
2	331	OpenGL::setActiveRendertarget	50
3	344	Rendererdocumentatie	20
4	345	Translationmaps	30

Prio	(#)	Requirements (buffer)	Units
4	312	OpenGL::Beginscene	50
4	311	OpenGL::Endscene	50
2	334	OpenGL::DrawPrimitive	200

8.1 Uitvoeringen

Een van de items van de sprint-backlog waarmee ik ben begonnen zijn de translationmaps. Translationmaps zijn in de Polemos engine een oplossing voor het omvormen van niet renderer-API specifieke functies naar renderer API specifieke functies. Hieronder is een voorbeeld gegeven over hoe onderdelen van Polemos (in dit geval een “Resource”) gebruik kan maken van de renderer zonder hierbij renderer-API specifieke instructies te hoeven geven.

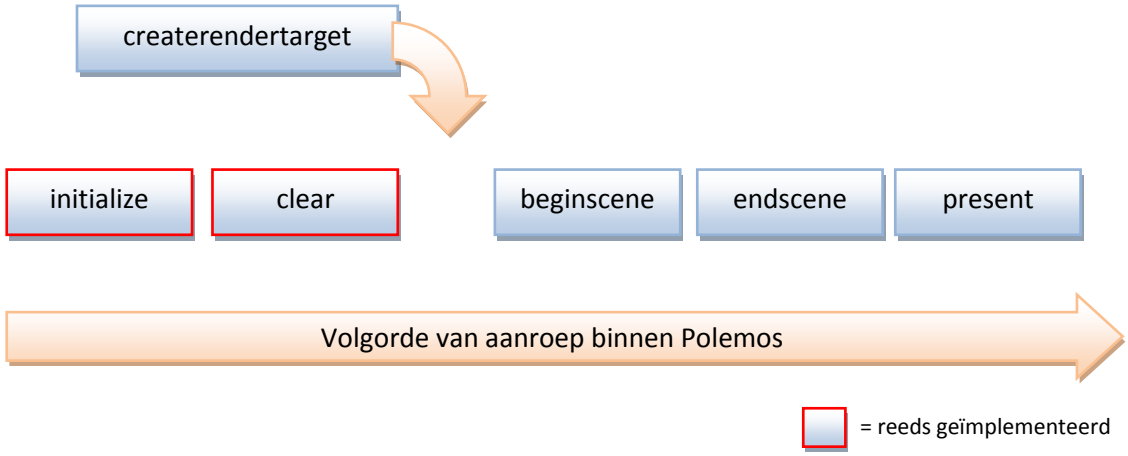


Afbeelding 15: Gebruik translationmaps

Zoals in deze afbeelding te zien is, kan de resource een instructie doorgeven aan de Polemos renderer. Afhankelijk van welke renderer er wordt gebruik, wordt er een translatiemap aangeroepen. Deze translatiemap zal de renderer-API specifieke selecteren op basis van de doorgegeven instructie. De renderer-API specifieke instructie zal vervolgens doorgegeven worden aan OpenGL of Direct3D. Ik ben met deze translatiemaps begonnen omdat deze functie wordt aangeroepen in de *initialize* –functie. Maar omdat deze functie zoveel broncode bevatte en gebruikt word in vele delen van de renderer heb ik deze als apart requirements opgenomen

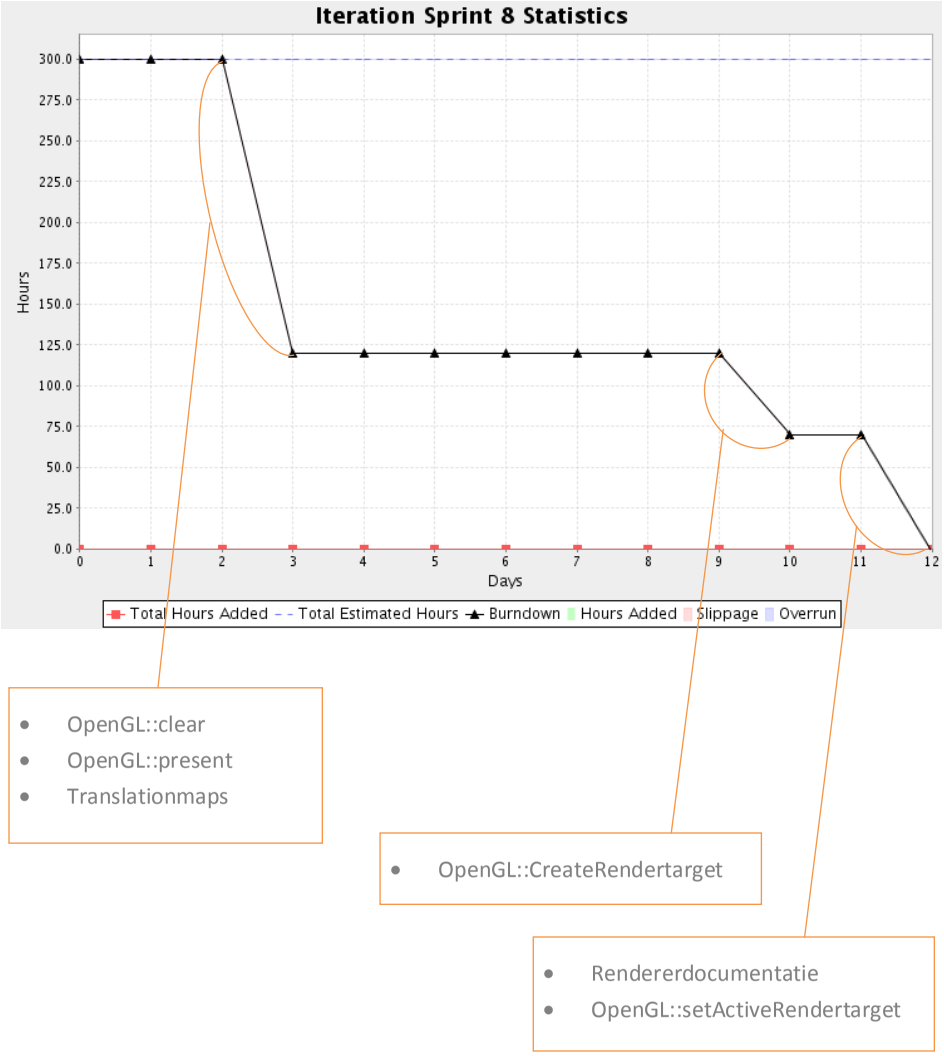
Het meeste aantal units in deze sprint stond voor de functionaliteit *present*. Deze functie zorgt ervoor dat de grafische kaart binnen een computer, zijn scherm als het ware update naar het een scherm op het beeldscherm. Op deze manier wordt er dus iets zichtbaar binnen een gemaakt scherm. Deze functie *present* kent OpenGL niet, maar gelukkig was er wel een functie die dezelfde functionaliteit had. Zo koste deze implementatie me uiteindelijk toch niet zoveel units en had ik meer tijd voor de andere items van de sprint-backlog.

Waar ik in deze sprint in totaal 100 units aan heb gewerkt waren de functies *createrendertarget* en *setactiverendertarget*. Deze twee functies horen in het project niet bij de 5 functies die de basisfunctionaliteit van de OpenGL renderer implementatie beschrijven. Echter, gedurende het programmeren aan Polemos kwam ik erachter dat de Direct3D renderer implementatie deze functie wel degelijk nodig heeft om Polemos te laten functioneren. Ik kwam hierachter door de functie niet te laten uitvoeren en de status van Polemos te bekijken na het niet uitvoeren van de functie. Polemos functioneerde daardoor in zin geheel niet meer en daardoor kon ik concluderen dat de functie *createrendertarget* van fitaal belang was voor de Direct3D renderer implementatie. Omdat deze functie binnen de engine eerder wordt aangeroepen dan *beginscene* en *endscene*, heb ik deze in overleg met de opdrachtgever ook eerder op de sprint-backlog laten terugkomen. Op deze manier zou de volgorde van mijn implementatie van de basisfunctionaliteit grotendeels gelijk staan aan de volgorde waarin deze functionaliteit wordt gebruikt binnen Polemos (zie afbeelding)



Afbeelding 16: Polemos basisfuntionaliteit volgorde aanroep

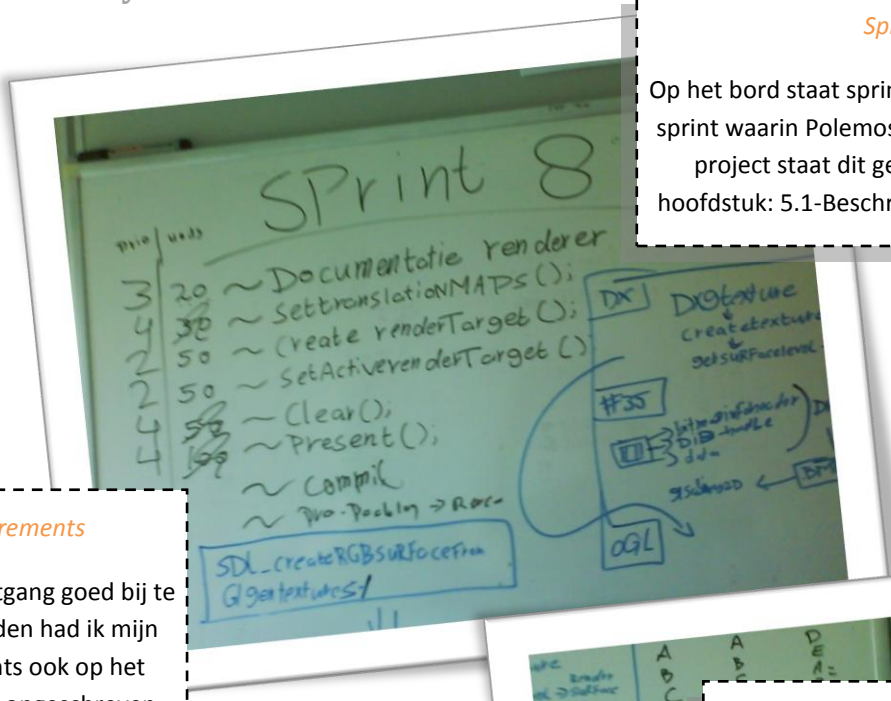
Waar deze volgorde niet mee overeenkomt, is dat ik de functie *present* eerder heb geïmplementeerd. De reden waarom de functie present ook in deze sprint-backlog is opgenomen is omdat deze functie binnen het ontwikkelproces een hogere waarde heeft dan *beginscene* en *endscene*. Present zorgt namelijk voor grafische output. De implementatie van de functie *clear*, had ik nooit kunnen testen zonder hier een *present* -functie op te laten volgen.



Toelichting

In de afbeelding hiernaast is goed te zien dat het begin van de sprint erg snel ging. Al na drie dagen lukte het me om drie functionaliteiten van de sprint-backlog af te maken waar veel units voor stonden. Op dat moment had ik ook verwacht de sprint-buffer uit deze sprint nodig te gaan hebben om later verder te kunnen werken. Echter bleek de implementatie van "createRendertarget" een stuk ingewikkelder dan geplant en was ik hier erg lang in verhouding tot de rest mee bezig. De documentatie van renderer was een item waarmee ik de hele sprint bezig ben geweest en uiteindelijk in de laatste dagen een einde aan heb gemaakt. Zo kon ik al mijn laatste bevindingen hierin ook opnemen.

8.3 What's on my mind wall



Sprints

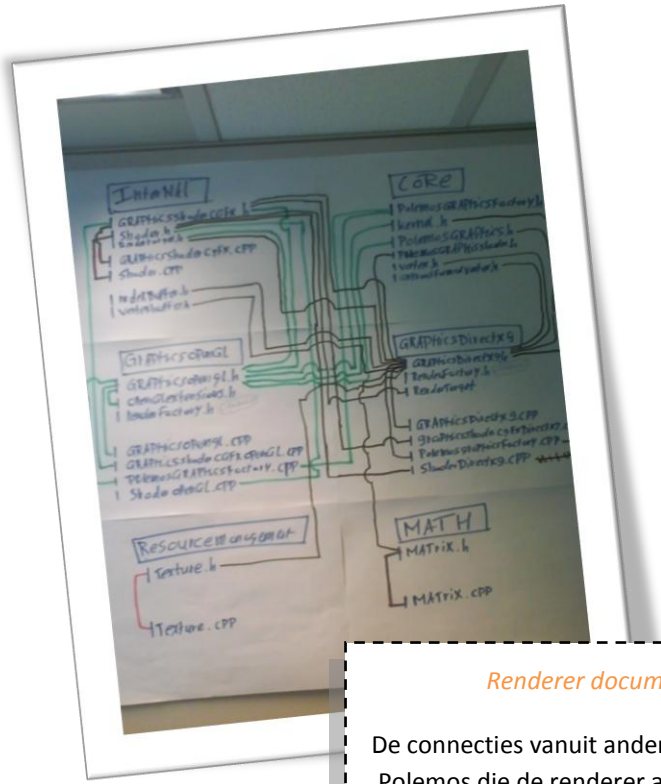
Op het bord staat sprint 8. Dit verwijst naar de sprint waarin Polemos zich bevind. Voor mijn project staat dit gelijk aan sprint 3 (zie hoofdstuk: 5.1-Beschrijving aanvang situatie)

Requirements

Om mijn voortgang goed bij te kunnen houden had ik mijn requirements ook op het white-board opgeschreven

createrendertarget

Hier heb ik een tekening gemaakt over wat de problemen waren bij de requirement *createrendertarget*



Renderer documentatie

De connecties vanuit andere onderdelen van Polemos die de renderer aanspreken heb ik eerst op papier in kaart gebracht. Meer hierover is te lezen in de rendererdocumentatie (zie bijlage)

9. Sprint 4

Nu de functies *initialize*, *clear*, *createrendertarget* en Present waren geïmplementeerd, hoefde ik om de basisfunctionaliteit te voltooien, nog de functies *beginscene* en *endscene* implementeren. Naast die twee implementaties komen in deze sprint nog een aantal implementaties aan bod. Onder andere het gebruik van matrices. Deze sprint duurde van 12 juli t/m 27 augustus.

Sprint duur : 7 weken

Deze sprint kent in totaal 35 werkdagen. Dit aantal komt niet overeen met mijn globale planning die ik in het begin van het project heb opgesteld. Dit komt omdat hier een vakantie tussen zit. De Scrum documentatie zegt echter dat ook deze dagen (net als weekenden) moeten worden meegerekend als werkdagen. Deze dagen mogen echter niet mee worden geteld bij het indelen van het aantal units. De uitkomst van deze regel zal uiteraard ook te zien zijn in de burndown chart als een rechte lijn.

Bij het inplannen van de sprint (die dus eigenlijk minus de vakantie gewoon twee weken duurde) zijn de product-owner en ik gaan kijken naar wat er nog moest gebeuren aan de basisfunctionaliteit. Dit waren de *beginscene* en *endscene implementatie.* Deze twee functionaliteiten zou dus zeker op de sprint-backlog komen. De product-owner en ik dachten dat ik hier ongeveer twee en een halve dag mee bezig zou zijn. Ik had dus nog anderhalve week over om verder te programmeren dan alleen de basisfunctionaliteit.

Na met de product-owner hierna gekeken te hebben kwamen we tot het besluit de functies *drawprimitive* en *setstreamsource* erbij te voegen. De functie *drawprimitive* heeft onder Direct3D de eigenschap een lijn te kunnen tekenen binnen een aangemaakt window (scherm). Deze functie wordt binnen Polemos vaak aangeroepen en het zou ook veel meervisuele aspecten van Polemos laten zien. Het zien van visuele output is een groot voordeel omdat men vaak de oorzaak van problemen binnen de renderer kan vinden aan de hand van grafische output die wordt getoond. Het is immers gemakkelijk om grafische output te hebben dan alle variabelen binnen de renderer en de externe library (OpenGL of Direct3D) stuk voor stuk te moeten onderzoeken. Dat is een veel meer tijdrovend proces.

De functie *setstreamsource* moet binnen eenDirect3D renderer implementatie altijd eerder worden aangeroepen dan *drawprimitive*. Bij *setstreamsource* wordt namelijk de videokaart klaargezet om data te gaan omzetten en data door te geven die weer kan worden gebruikt bij de functie *drawprimitive.* Hoe OpenGL precis omgaat met het tekenen van lijnen en of daar een equivalent van *setstreamsource* voor nodig was, wisten mijn product-owner en ik op dit moment nog niet. Helaas ben ik door tijdsbestek redenen nooit toegekomen aan de oplossing hiervan.

Prio	#	Requirements	Units
4	334	OpenGL::Drawprimitive	80
4	312	OpenGL::Beginscene	50
4	311	OpenGL::Endscene	50
4	309	OpenGL::SetStreamSource	200

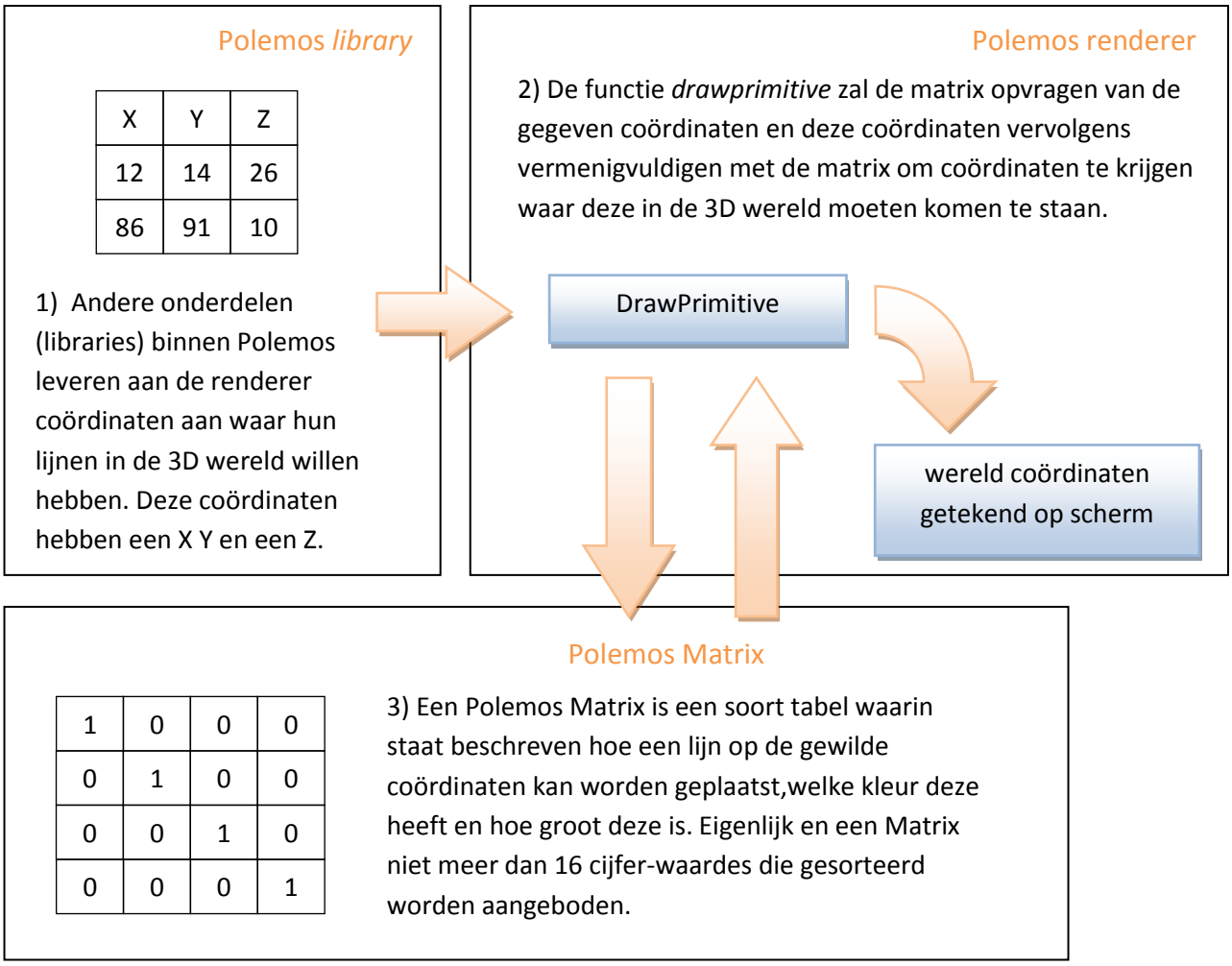
De invulling van de buffer waren een aantal requirements die buiten de scope van dit project lagen. Uiteindelijk zou ik hier nooit aan toe zijn gekomen. De requirements bevatten opdrachten die hoog op de ranglijst liggen van de extra functionaliteiten van de OpenGL renderer implementatie. Onder andere het *OpenGL::setTexture* item is iets wat de product-owner graag in de renderer zou zien. Dit houdt in dat de renderer om kan gaan met het inladen en tekenen van plaatjes. Helaas ben ik nooit aan deze buffer requirements toegekomen.

Prio	#	Requirements (buffer)	Units
2	354	CXXtest koppellen aan OpenGLrenderer door middel van static-linking	5
4	353	renderer documentatie 'testen' en verbeteren	20
3	350	Textureloading onderzoek (sdl / OpenIl /...)	115
2	335	OpenGL::"setTexture"	50

9.1 Uitvoeringen

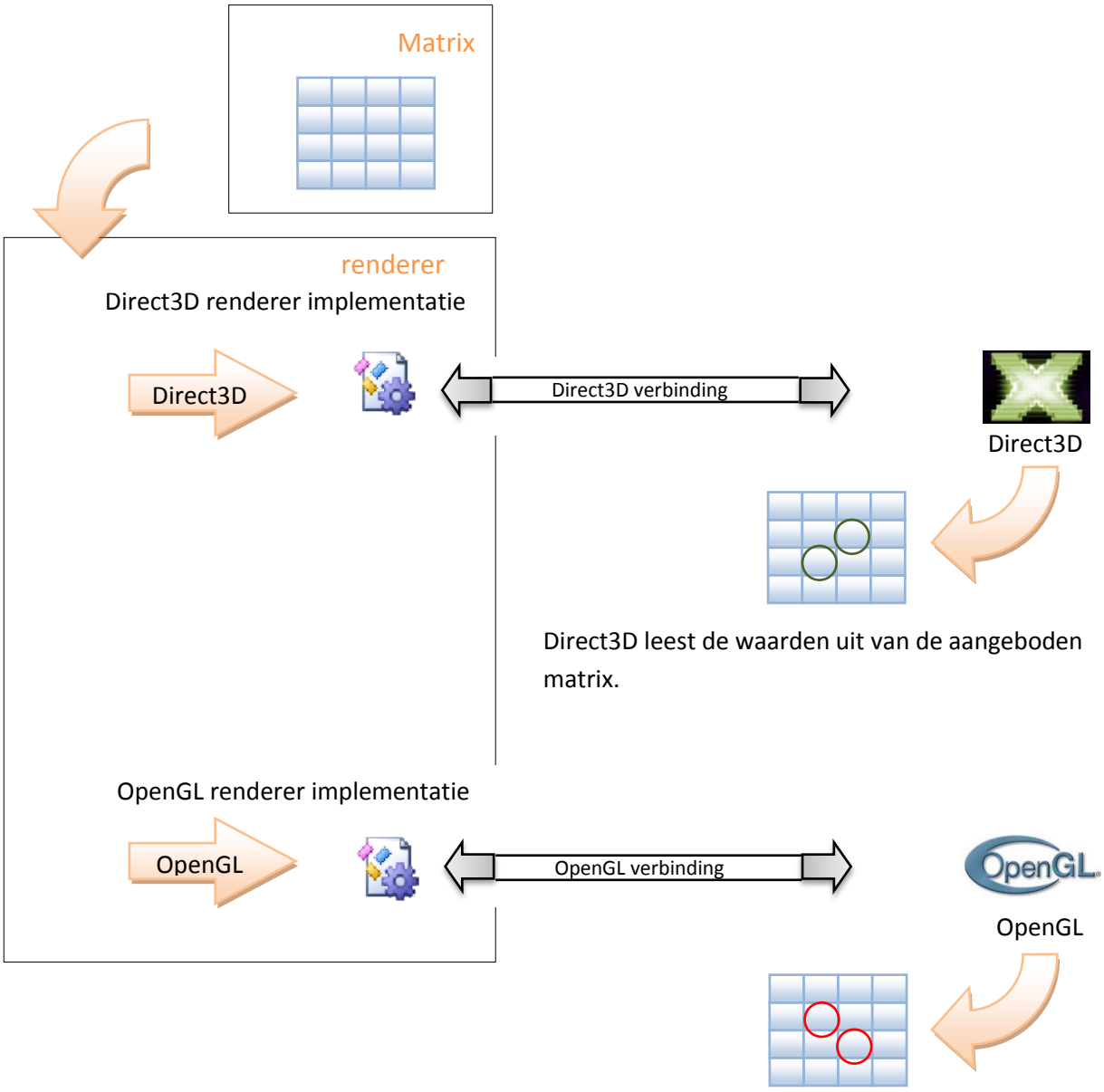
Omdat de basisfunctionaliteiten functies (*beginscene* en *endscene*) de functies waren die van noodzaak waren voor het slagen van het project (zie hoofdstuk: plan van aanpak) ben ik hiermee begonnen. De implementatie van *beginscene* had ik al vrij snel voor elkaar. Gaandeweg in het project werd het echter steeds duidelijker dat mijn implementatie van *beginscene* niet klopte. Omdat deze twee functies me erg veel tijd koste heb ik ze toen even laten liggen en ben verder gegaan met de functie *drawprimitive*. Op deze manier wilde ik voorkomen dat ik me zou blindstaren op dit probleem en zo veel tijd kwijt zou zijn. Ik begon dus aan de functie *drawprimitive* in de hoop later het probleem op te kunnen lossen.

Het tekenen van lijnen in een engine is een proces wat erg vaak gebruikt wordt. In het engine worden per frame van een gemiddeld spel duizenden lijnen getekend om uiteindelijk het gewenste beeld te krijgen. Hieronder is schematisch weergegeven hoe Polemos lijnen op het beeldscherm weergeeft.



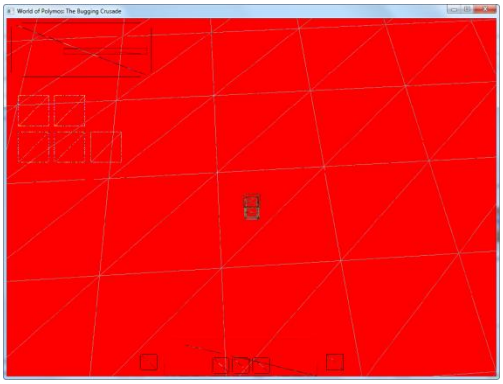
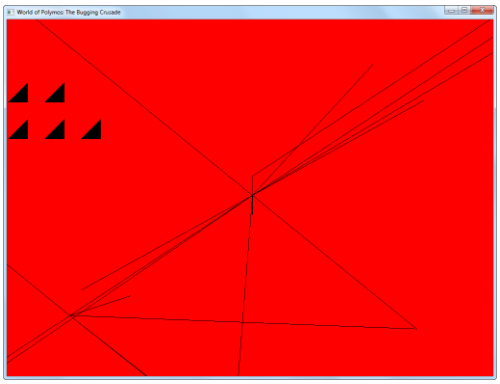
Afbeelding 17: DrawPrimitive / matrices gebruik binnen Polemos

Wat ik al verwacht had ik bij het implementeren van deze functie, was het verschil in de matrices. Dit verschil had ik al onderzocht in de initialisatie-fase. Direct3D en OpenGL willen een matrix op een andere manier gesorteerd aangeboden krijgen. Uit de aangeboden matrix worden waarden uitgelezen door Direct3D of OpenGL. Polemos biedt de matrix aan op de manier hoe Direct3D deze nodig heeft. Dit is logisch omdat Direct3D eerder geïmplementeerd was. OpenGL leest echter verkeerde waarden uit omdat OpenGL naar waarden zoekt op een andere locatie binnen de matrix dan Direct3D.



Afbeelding 18: Polemos matrix probleem

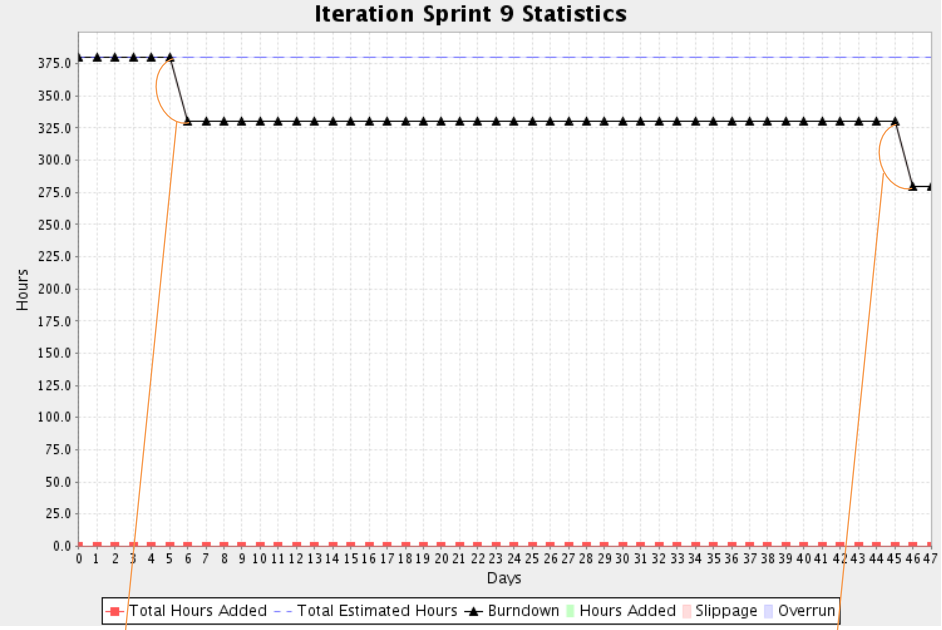
Door het verkeerd uitlezen van de coördinaten worden er dus lijnen getekend die niet op de goede plek staan. Het volgende beeld laat goed de verschillen zien tussen de coördinaten van Polemos met de Diret3D renderer implementatie en de OpenGL renderer implementatie.

Direct3D	OpenGL
	

Afbeelding 19: Polemos met verkeerd uitgelezen OpenGL matrix

Om dit probleem op te lossen zou nog veel tijd in beslag nemen. En omdat het al tegen het einde van de sprint liep, besloot ik me toch weer te focussen op het *beginscene*/*endscene* probleem waar ik de sprint mee begon. Na het boek *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics* er op nageslagen te hebben en veel met de product-owner te hebben overlegt, kwam ik tot de conclusie dat de implementatie van *beginscene* en *endscene* onder OpenGL geen waarde hebben. Direct3D maakt gebruik van deze functies om de grafische kaart te 'laten weten' dat de engine klaar is om lijnen te willen tekenen. Bij een OpenGL renderer implementatie hoeft dit niet aan de grafische kaart te worden verteld. OpenGL kan altijd de grafische kaart aanspreken om lijnen te tekenen.

Na deze conclusie was de laatste sprint van mijn project afgerond. Ondanks het feit dat er weinig was geïmplementeerd binnen deze sprint was de product-owner verheugd dat er toch veel voorruitgang was geboekt op het gebied van kennis wat weer toegespeeld kan worden op andere studenten.



- OpenGL::beginScene

- OpenGL::endScene

Toelichting

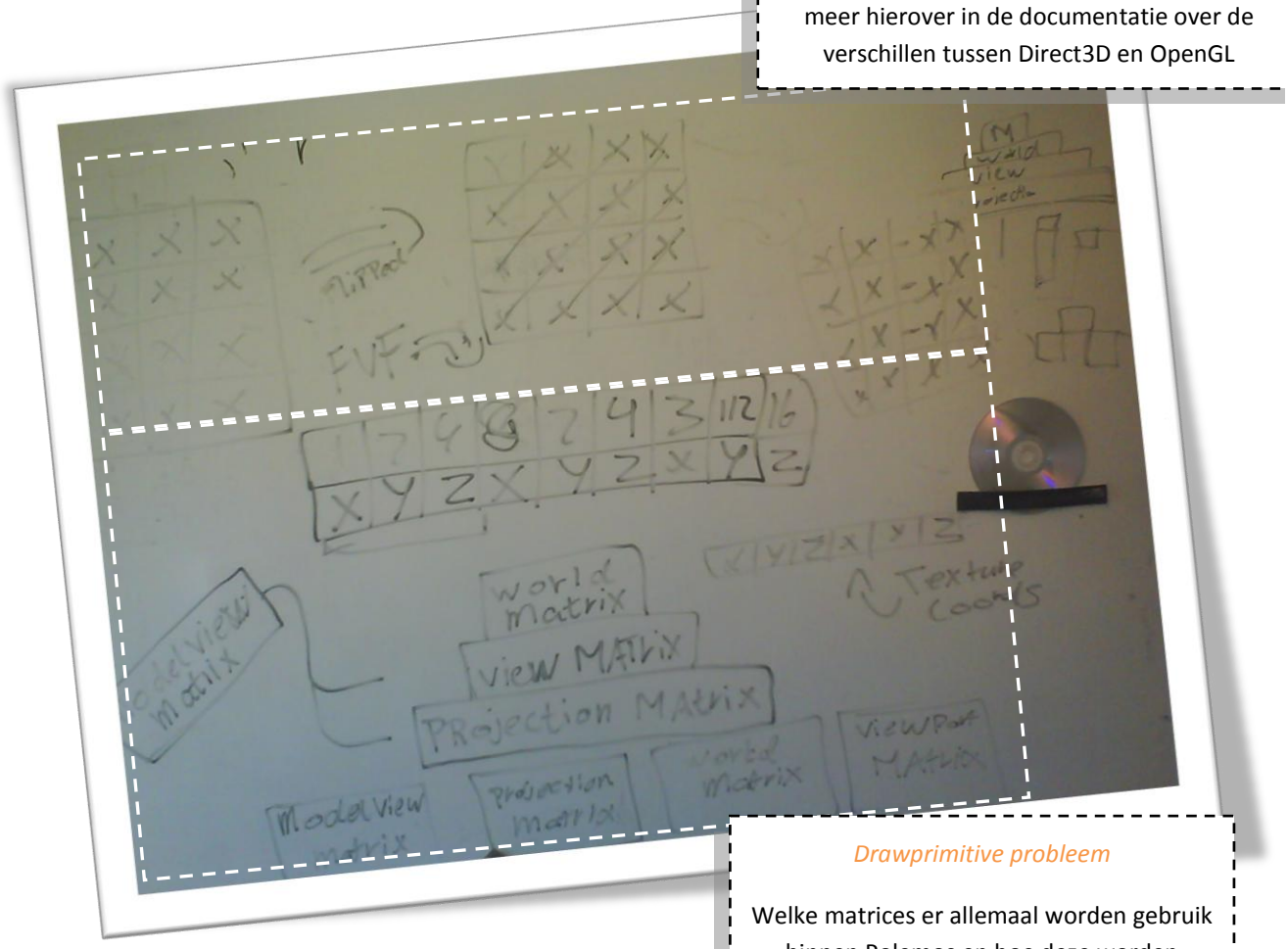
In deze sprint zijn er twee van de vier items uiteindelijk afgevinkt op de burndown chart. Dit zijn beginscene en endscene. Wat niet op de burndownchart te zien is zijn mijn werkzaamheden aan de functionaliteit van drawprimitive. Hier ben ik wel degelijk een groot deel van de tijd mee bezig geweest (±250 units). Helaas is deze nooit helemaal afgekomen. En een item mag pas worden afgevinkt zodra deze helemaal af is.

Wat ook opvalt is dat de burndown chart erg lang is. Dit komt doordat er een vakantie tussen zit (day 7 tot en met day -40). De sprint werk dus als het ware door tweeën gesplitst door een vakantie.

9.3 What's on my mind wall

Matrix translatie

Het verschil in matrices wat ik ben tegengekomen in beschreven heb in deze sprint meer hierover in de documentatie over de verschillen tussen Direct3D en OpenGL



Drawprimitive probleem

Welke matrices er allemaal worden gebruik binnen Polemos en hoe deze worden aangeboden heb ik hier samen met de product-owner geprobeerd vast te stellen

10. Sprint 5

Nu alle basisfunctionaliteit in Polemos was geïmplementeerd, en ik hiermee gedeeltelijk had onderzocht hoeveel werk het was om een stap richting een crossplatform Polemos te zetten was, kon ik beginnen om een advies te schrijven. Een groot deel van deze sprint zal dan ook gaan over dit adviesrapport en over het aanvullen van documentatie van eerder gemaakte documentatie. Deze sprint duurde van 30 augustus t/m 10 september.

Sprint duur : 2 weken

Omdat dit de laatste sprint was van het project, kwam er een extra requirement bij op de sprint-backlog. Dit was het “schoonmaken” van de broncode. Omdat ik tijdens het programmeren soms wat code neerzet om ‘iets te testen’, wordt de engine rommelig. Na afloop van een project is het dan ook handig om deze code op te ruimen voordat de engine wordt opgeleverd.

In tegenstelling tot de vorige sprint, stond deze sprint meer in het teken van documentatie. Het adviesrapport kwam aan bod, wat in de initialisatie-fase al was beschreven, en het aanvullen van de documentatie over de renderer en de verschillen tussen OpenGL en Direct3D. De reden waarom ik in deze sprint juist de documentatie wilde aanpassen, was omdat in de vorige sprints het programmeren aan bod was gekomen, nu kon ik deze opgedane kennis documenteren en zo overbrengen aan andere studenten die eventueel aan Polemos zullen gaan werken.

Zoals in de sprint-backlog hieronder te zien is, zijn er in deze sprint 220 units toebedeelt. Dit komt doordat ik in deze week extra tijd nodig had voor het creëren van dit afstudeerverslag. De rest van de units is dus toebedeeld aan dit document.

Prio	#	Requirements	Units
4	355	Adviesrapport Polemos crossplatform	100
2	351	Aanvullen verschillen OpenGL & Direct3D (toevoeging: matrices)	80
4	357	OpenGL renderer unittesten	3
4	353	Aanvullen OpenGL renderer documentatie	31
4	356	Code clean (hacks/tests eruit)	6

De sprint-buffer van deze sprint bevat maar één item waarin het probleem van de vorige sprint verder wordt opgelost. Als ik er in deze sprint aan toe zou komen zou requirement 358 een leuke toevoeging zijn op de OpenGL renderer implementatie omdat deze veel grafische output geeft en daardoor veel voldoening geeft aan mij persoonlijk van dit project. Helaas ben ik aan deze requirement nooit toegekomen.

Prio	#	Requirements (buffer)	Units
2	358	Projection/view/World matrices transleren/transposen	40

10.1 Uitvoeringen

Het testen van de OpenGL renderer was de eerste requirement waaraan ik ben begonnen. Dit bleek me verstandig omdat deze weinig units had (namelijk maar 3) en deze requirement onderdeel was van de mijlpaal om unittests te schrijven. Om de OpenGL renderer implementatie te testen hoefde ik geen nieuwe unittest te schrijven. Omdat unittests de functionaliteit van broncode testen en niet de code zelf, kon ik dezelfde code gebruiken als bij de Direct3D unittest. De functionaliteit van beide renderers is immers hetzelfde. Nadat ik deze code had hergebruikt gaf de test met de OpenGL renderer implementatie dezelfde uitkomst als die van Direct3D. Hier kwam ik dus helemaal geen problemen tegen. Deze unittest was het laatste binnen dit project wat enig programmeerwerk vereiste. Hierna kon ik dus mijn code wat gaan opschonen (requirement: 356).

Het document met de verschillen tussen Direct3D en OpenGL heb ik aangevuld met technische verschillen die ik ben tegengekomen na het programmeren aan de OpenGL renderer implementatie (sprint 2, 3 en 4). Om ervoor te zorgen dat deze documentatie te begrijpen is en om studenten hier wat aan hebben, heb ik dit document laten valideren door studenten zelf. Namelijk toen sprint 5 van mijn project begon, zijn er studenten verder gegaan met de ontwikkeling van Polemos. Deze studenten hebben gewerkt aan een deel van Polemos wat gebruik maakt van de renderer. Aan deze studenten heb ik mijn documentatie voorgelegd. Daarna heb ik samen met deze studenten de documentatie doorlopen en gevraagd of ze de beschreven onderwerpen begrepen. Op een aantal verbeteringspunten na, vonden ze de documenten begrijpelijk en hadden ze er zeker wat aan tijdens hun project.

Als laatste mijlpaal van dit project was er het adviesrapport. In dit rapport leg ik allereerst uit wat voor voordelen een crossplatform engine heeft en of dit de kwaliteit van Polemos verbeterd. Dit doe ik aan de hand van de ISO standaard. Deze standaard geeft richtlijnen voor de kwaliteit van software. Daarnaast geef ik weer wat er aangepast zal moeten worden om tot een crossplatform engine te komen. Ik heb deze ISO standaard in het advies verwerkt omdat dit een begrijpelijke standaard is waar ook de studenten in hun opleiding mee te maken krijgen. Op deze manier is het adviesrapport dus ook leesbaar voor studenten. De reden waarom ik voor de ISO standaard heb gekozen en niet voor een andere richtlijn voor kwaliteit van software, is omdat ik al eerder met deze standaard heb gewerkt en hier ervaring mee heb.

Na de kwaliteit van Polemos aan de hand van ISO te hebben beschreven heb ik nagedacht over wat de product-owner nog meer aan informatie nodig heeft. Samen met de product-owner kwam ik tot de conclusie dat de volgende aspecten nodig zijn om de product-owner een evenwichtig oordeel te laten vellen.

- De product-owner moet weten wat de mogelijkheden zijn om crossplatform te gaan
  Als Polemos crossplatform gaat, moeten er een aantal libraries worden aangepast. Er kunnen verschillende nieuwe libraries worden geïmplementeerd die wel crossplatform

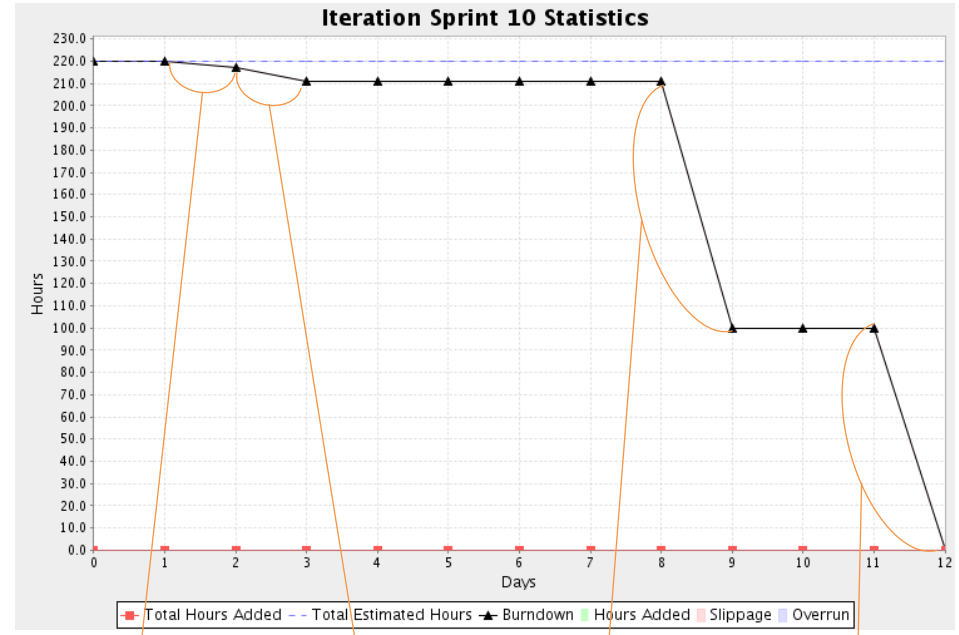
zijn. Dat is de reden waarom ik drie verschillende mogelijkheden met nieuwe libraries heb opgesteld.

- De aanpassingen aan Polemos moeten duidelijk zijn voor de product-owner
  Mocht er over worden gegaan op een crossplatform implementatie, dan is het van belang dat de product-owner een overzicht heeft van de libraries van Polemos die moeten worden aangepast.

- Persoonlijk advies
  Omdat ik een lange tijd met Polemos heb gewerkt heb ik ook een persoonlijk advies gegeven over de toekomst van Polemos. Dit is puur een advies en hoeft de product-owner uiteraard niet op te volgen.

Door de beschrijving van deze drie aspecten zou de product-owner een oordeel kunnen vellen over de toekomst van Polemos. De doelgroep van dit verslag is uiteraard de product-owner en daardoor kon ik diep op de informatie in gaan en hoefde ik weinig achtergrond informatie in het document op te nemen.

In de conclusie van het adviesrapport heb ik mijn eigen persoonlijk advies geschreven waarin ik aanraad om Polemos een crossplatform te maken. Meer hierin in de bijlage onder ‘adviesrapport’.



- OpenGL renderer unittesten
- Code clean (hacks/tests eruit)
- Aanvullen verschillen OpenGL & Direct3D (matrices)
- Aanvullen OpenGL renderer documentatie
- Adviesrapport Polemos crossplatform

Toelichting

In deze laatste burndown chart van mijn project is te zien dat alle 220 units uiteindelijk afgerond zijn. De overige units die ik voor mijn afstudeerverslag heb gebruikt (zie inleiding sprint 5) zijn dus niet in de burndown chart te zien. Dit komt doordat het afstudeerverslag niet in directe relatie staat met Polemos en dus ook niet op de product-backlog mag worden opgenomen.

Verder is ook goed te zien dat ik pas de documentatie aan het einde van sprint heb afgerond en begonnen ben met het deel waarin ik nog moest programmeren (OpenGL renderer unittesten / Code clean).

10.3 What's on my mind wall

Sprints

Sprint 10 staat gelijk aan sprint 5. Voor meer informatie zie hoofdstuk 5.1:aanvang situatie

Matrices

Om de verschillen van matrices die ik de vorige sprint tegen was gekomen goed te documenteren, had ik deze eerst weer uitgeschreven voor mezelf om dit goed helder te krijgen

Burndown charts

Aan het einde van het project heb ik de burndown charts opgehangen en zo kon ik de voortgang zien van geheel mijn project

11. Evaluatie procesgang

Zoals beschreven staat in dit verslag heb ik in dit project gebruik gemaakt van Scrum. Ik heb dit als positief ervaren. Persoonlijk was dit mijn eerste keer dat ik gebruik maakte van dit raamwerk van technieken. Ik heb het me dan ook eerst eigen moeten maken om er vervolgens mee te kunnen werken. Dit eigen maken Scrum (initialisatie-fase) ging me best goed en gemakkelijk af. Ik had snel door wat Scrum voor documentatie producten heeft en hoe ik deze kon gebruiken in mijn project.

Waar ik vooral erg positief over was, was de manier hoe Scrum omgaat met de planning. Doordat er in dit project gemakkelijk onvoorziene problemen zich konden voordoen (wat ook is gebeurd, zie einde sprint 1)was is huiverig om een planning te maken. Scrum houdt hier rekening mee om per sprints een planning op te stellen. Natuurlijk heb ik wel een planning aan het begin van het project gemaakt met globaal tijdsbestek van dit project. Dit omdat het project een maximale tijd van 17 weken mocht beslaan.

Wat lastig bleek aan Scrum was het inplannen van requirements voor een sprint-backlog. Omdat ik nog nooit met de programmeertaal OpenGL had gewerkt, en mijn begeleider ook niet, kon ik lastig inschatten hoelang bepaalde functies me zouden kosten. Uiteindelijk bleek dat de planningen toch goed waren aangezien ik in vier van de vijf sprints alle requirements hebben kunnen voltooien.

De omgang met de product-owner ging in dit project ook erg goed. Deze was altijd op tijd aanwezig bij de afgesproken sprint- en voortgangs-meetings en was altijd goed voorbereid. Daarnaast kon ik goed met de product-owner overweg en kon ik altijd binnenstappen voor een vraag. Dit heeft me veel geholpen in het project omdat het project op deze manier geen vertraging op liep als ik tegen een probleem aan liep.

Verder was ik achteraf erg blij met initialisatie-fase. De vier weken die ik hiervoor had staan in de globale planning bleken te kloppen en had ik ook hard nodig om kennis op te doen van OpenGL, Direct3D, Polemos en Scrum. Dit is ook iets wat ik de volgende keer beter zou plannen. Door de initialisatie-fase te verlengen had ik misschien iets meer tijd gehad me te verdiepen in OpenGL en de basistechnieken hiervan zodat ik deze later gemakkelijker had kunnen implementeren. In het vervolg zal ik dus meer letten op wat ik mezelf moet aanleren en preciezer onderzoek doen naar hoeveel tijd ik hiermee denk bezig te zijn.

Een punt van verbetering voor mijzelf vind ik het risicomanagement. Omdat Scrum per sprint indeelt is dit punt van minder belang dan als er bijvoorbeeld RUP wordt gebruikt, maar wat er precies moet gebeuren als er mijlpalen niet kunnen worden opgeleverd, had ik beter kunnen uitwerken. Dit is bij Scrum minder van belang omdat de flexibiliteit van het project velen malen groter is dan die van RUP. Scrum kan dus gemakkelijk ingrijpen in het project zonder al te groten gevolgen. Echter, ook bij het gebruik van Scrum kunnen zulke problemen zich voordoen en deze vorm van risicomanagement had ik beter kunnen beschrijven.

Al met al ben ik dus erg tevreden met Scrum en de procesgang van mijn project. De eerste keer dat ik Scrum heb toegepast binnen een project is me dus goed bevallen en is denk ik ook een goede toevoeging aan mijn persoonlijke kennis op het gebied van projectmethodieken.

Sterke punten

- Zelfstandigheid

  Ondanks dat ik geen ervaring had met OpenGL en Scrum, heb ik een grote 3D software applicatie weten om te bouwen en hier een OpenGL renderer implementatie ingbouwd (basisfunctionaliteit). Hierbij kwam een hoop zelfstandigheid kijken en heb ik ook veel aan zelfstudie gedaan.

- Aanpassing aan technieken

  Tijdens het project heb ik veel moeten leren en me veel technieken en tools eigen moeten maken (CXXtest, Scrum, OpenGL, XP-dev). Ondanks de hoeveelheid aan nieuwe technieken en tools, is het me gelukt deze allemaal eigen te maken en hierin een project tot een goed einde te brengen

Verbeter punten

- Scrum Buffer

  De Scrum-buffer was iets waar ik pas later in het project mee ben begonnen. Deze toevoeging kwam ik pas later mee in aanraking, maar kan van groot nut zijn binnen een project. Hierin had ik me kunnen verbeteren door deze direct toe te passen binnen mijn project

- Inplannen sprints

  Het inplannen van de units uit een sprintis iets wat ik de volgende keer . anders zal gaan aanpakken. In dit project kwamen de units die ik had ingeplant soms niet overeen met hoeveel tijd ik er echt voor kwijt was. Dit kwam o.a. doordat ik nog nooit met OpenGL had gewerkt. De volgende keer ga ik proberen de units proberen beter in te plannen door hier meer aandacht aan te geven in mijn project en onderzoek te doen naar de gemiddelde duur van een requirement.

12. Verantwoording competenties

Voor dit afstudeerverslag heb ik een aantal competenties (beroepstaken) opgesteld die ik in dit afstudeerproject heb uitgevoerd. Deze competenties geven criteria weer, waaraan moet worden voldaan binnen een project.

- Selecteren methoden, technieken en tools

  Vooral in de beginfase van het project heb ik met deze competentie te maken gekregen. Het selecteren van mijn programmeeromgeving en het kiezen van een geschikte methodiek voor mijn project zullen een groot deel van deze competentie vormen en invullen. Een aantal methoden en tools die ik in dit project heb gebruikt zijn:

  - Visual Studio 2005
  - Scrum
  - XP-dev
  - Cxxtest

  Ik heb deze competentie invulling gegeven door me te laten adviseren door mede studenten en docenten om bepaalde tools te gaan gebruiken. Daarnaast heb ik online documentatie opgezocht over methoden, technieken en tools of deze toereikend zouden zijn.

- Voorbereiden en opstarten software-ontwikkeltraject

  Ik heb deze competentie invulling gegeven door middel van een aantal beslissingen in de structuur van mijn project te nemen. Om goed het software-ontwikkeltraject in te aan (wat begint bij uitvoeren van de sprints heb ik een initialisatie-fase opgenomen in de (globale)planning. In deze fase heb ik voorbereidingen getroffen die ik nodig had om te beginnen aan het softwaretraject (sprint 1 tot en met 5). Tijdens mijn software-ontwikkeltraject heb ik de kennis die ik opgedaan had tijdens de initialisatie-fase tijdens iedere sprint wel gebruikt. Dit bewijst dat mijn initialisatie-fase van belang is geweest.

- Uitvoeren analyse door definitie van requirements

  Het opstellen van requirements is een onderdeel wat wederom wordt uitgevoerd in de initialisatie-fase. Het behalen van deze competentie richt zich op het correct analyseren van de opdracht. Aan de hand van deze analyse vervolgens een probleemstelling vormen en hieruit requirements filteren is dan ook wat ik heb gedaan. Dit is gevalideerd door de product-owner, was immers tevreden met het resultaat van het project en de uitgevoerde requirements.

- Ontwerpen software architectuur

  Voordat ik de nieuwe architectuur van de OpenGL renderer implementatie heb geprogrammeerd, heb ik hier eerst diagrammen over gemaakt. Deze diagrammen geven de structuur van de nieuwe OpenGL renderer implementatie weer

- Ontwerpen systeemdeel

  Deze competentie staat eigenlijk in het verlengde van “Ontwerpen software architectuur”. Deze competentie gaat alleen verder en vereist een ontwerp wat direct te implementeren is. Deze ontwerpen heb ik gedurende het project gemaakt voordat ik de OpenGL renderer implementatie heb geschreven

- Bouwen applicatie

  Bij deze competentie ligt de nadruk vooral op het bouwen van de nieuwe software. In dit project vallen onder deze competentie:

  - Het programmeren van OpenGL renderer implementatie.
  - Het opzetten van het CXXtest framework.

  Beide aspecten zijn in dit project uitgevoerd en succesvol binnen Polemos geïmplementeerd.

Beroepstaken naar “Beroepstaken Informatica HHS” van Juni 2009 (documentnummer: INF2009-001)

13. Evaluatie opgeleverde producten

Hieronder staan alle producten die ik in dit project heb opgeleverd. Daarnaast heb ik per product een sterk punt en een verbeter punt opgenomen en beschreven.

- OpenGL renderer implementatie

  Over de OpenGL renderer implementatie ben ik tevreden. In Polemos is basisfunctionaliteit voor OpenGL toegevoegd en uiteindelijk zelfs iets meer dan dat (onder andere de implementatie van *drawprimitive* functionaliteit zie sprint 4). Ook vind ik dat Polemos er beter op is geworden met deze implementatie ongedacht of de product-owner misschien besluit er niet mee verder te gaan.

  Een punt waarin de OpenGL renderer implementatie nog zou kunnen verbeteren is de implementatie van *drawprimitive* -functie verder af te maken. Deze functie is (zoals te lezen in sprint 4) niet geheel geïmplementeerd door gebrek aan tijd. Het zou een stap voorwaarts zijn in de grafische output die de OpenGL renderer implementatie op dit moment levert.

  Sterke punt
  - Meer dan basis geïmpl.

  verbeterpunt
  - Implementatie *drawprimitive*

- Unittests van de twee renderer implementaties

  De unittests die gemaakt zijn in sprint 1 vind ik persoonlijk ook naar behoren uitgevoerd. Ze geven op een duidelijke manier weer of er iets mis gaat met de renderer en waar deze eventuele fout zich voordoet. Ook ben ik tevreden over het feit dat ik het CXXtest framework heb kunnen aanpassen zonder veel ervaring te hebben met de programmeertaal Python, waar de framework interface is in geschreven.

  Wat ik persoonlijk nog een verbeterpunt zou vinden, is de mogelijkheid om meerdere tests tegelijkertijd uit te voeren. Het CXXtest framework laat dit standaard niet toe. Helaas heb ik deze functionaliteit niet kunnen toevoegen door gebrek aan tijd.

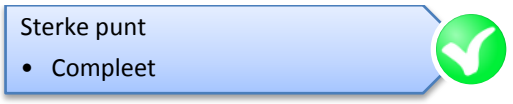
  Sterke punt
  - Duidelijke weergave tests

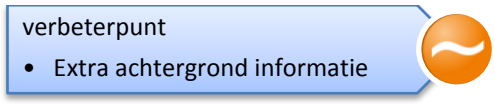
  verbeterpunt
  - Meerdere tests tegelijk

- Renderer documentatie

  Een positief punt van de documentatie van de renderer vond ik dat deze compleet was. Er stond informatie in die van belang is voor studenten die verder zullen gaan met het ontwikkelen aan Polemos. Dit bleek ook uit de validatie van de documentatie die ik in sprint 5 heb uitgevoerd bij studenten.

Als verbeterpunt had ik de diagrammen iets meer kunnen toelichten. Veel toegevoegde waarde zal dit echter niet hebben aangezien de doelgroep (studenten met 3D programeer ervaring) deze diagrammen zou moeten begrijpen zonder veel achtergrond informatie erbij te verschaffen.





- verschillen Direct3D en OpenGL

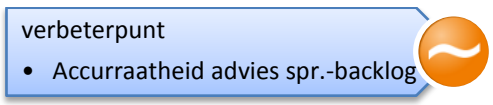
De documentatie die zich richt op de verschillen tussen Direct3D en OpenGL vind ik een mooi document geworden. Er staat veel in over waarin de twee libraries met elkaar verschillen. Verder was ik ook erg tevreden met het feit dat de studenten die na mij aan Polemos hebben gewerkt, de documentatie goed vonden en hier wellicht veel gebruik van zullen maken.

Een verbeterpunt vind ik echter wel dat de technische verschillen wel uitgebreid beschreven zijn, maar het zijn er in aantal niet erg veel. Met meer tijd had ik hier eventueel nog wat meer van kunnen maken. De beschreven technische verschillen zijn echter wel die in de huidige renderer zijn verwerkt.

- Scrum documentatie

De Scrum documentatie (burndown charts, product-backlog, advies sprint-backlog sprint-backlog) vind ik goed bijgehouden. Voordat iedere meeting begon had ik mijn advies mooi opgeleverd zodat de product-owner een goed overzicht had van de stand van zaken. Ik had hiervoor een document opgesteld dat inmiddels ook door andere studenten, die aan Polemos werken, wordt gebruikt.

Wat ik beter had kunnen doen was de precisie van het advies sprint-backlog verhogen. Soms had ik mijn ingeplande units die ik hierin opstelde niet goed berekend en dit stond slordig. Hierin had ik dus wat preciezer moeten zijn. Verder vond de product-owner dit niet erg en had dit ook geen verdere gevolgen. Wel was dit een leermoment om de volgende keer op te letten.



Adviesrapport

Het adviesrapport, dat in de laatste sprint aan bod kwam, vind ik een duidelijk document geworden. Er staat een duidelijk overzicht van Polemos en waarin deze wel of niet crossplatform is. Ook wordt er een duidelijk overzicht gegeven van de mogelijk opties die Polemos heeft in het wel of niet opstappen naar het worden van een crossplatform engine. Hierdoor kan de product-owner een wel overwogen keuze maken.

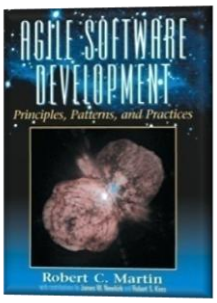
Wel had ik wat dieper op de problemen in willen gaan als het gaat om het bewerkstelligen van crossplatform code. Een soort handleiding voor de programmeur waar deze op moet letten bij het schrijven van broncode. Maar omdat de tijd dit niet toeliet, ben ik hier niet verder op ingegaan.

Sterke punt
- Duidelijke uiteenzetting

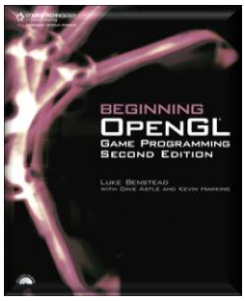
verbeterpunt
- handleiding schrijven broncode

14. Geraadpleegde literatuur

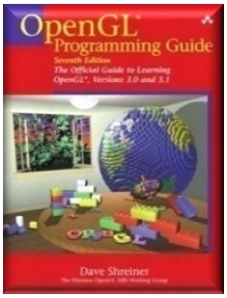
Boek:	**Agile software development. Principles, Patterns , and practices**
Auteur:	Robert C. Martin
Uitgever:	Pearson education
ISBN	0-13-597444-5
Overzicht:	Dit boek gaat over het gebruik van Agile als ontwikkelmethode. Het is geschreven voor zowel managers als ontwikkelaars en behandeld onder andere onderwerpen zoals Extreme programming (XP) en test-driven development.



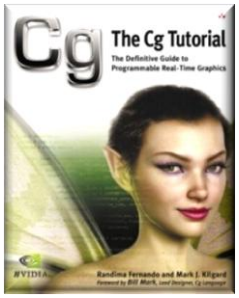
Boek:	**Beginning OpenGL Game programming (second edition)**
Auteur:	Luke Benstead
Uitgever:	Cengage Learning
ISBN	9781598635287
Overzicht:	Dit boek heb ik gebruikt om de basis van OpenGL te leren begrijpen. Dit boek begint bij de basis en legt op een begrijpbare manier uit hoe verschillende OpenGL functies werken binnen de API en hoe deze het beste gebruikt kunnen worden



Boek:	**OpenGL Programming guide (Red book)**
Auteur:	Dave Shreiner & The Khronos OpenGL ARB Working Group
Uitgever:	Pearson education
ISBN	0321552628
Overzicht:	Dit is de officiële programmers-guide om te leren programmeren in OpenGL. Het is geschreven voor programmeurs en er zitten voorbeelden van broncode in. Ook wordt er in dit boek aandacht geschonken aan GLUT en GLSL.



Boek:	**The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics**
Auteur:	Randima Fernando, Mark J. Kilgard
Uitgever:	Addison-Wesley 2003
ISBN	9780321194961
Overzicht:	Een boek over het gebruik van Cg (high-level shading language). Ik heb dit boek veel gebruikt voor het nakijken van vertex en fragment profiling.



Website:	**Officiële OpenGL website**
Adres	http://www.opengl.org/
Overzicht:	Deze website heb ik gebruikt om informatie van te krijgen. Onder andere over hoe de OpenGL API werkt, maar ook om tutorials van te downloaden en stukken code uit te proberen.

Website:	**OpenGL tutorial (Cprogramming)**
Adres	http://www.cprogramming.com/tutorial/opengl_first_opengl_program.html
Overzicht:	Deze website heb ik gebruikt bij het inlezen over OpenGL. Op deze website staat uitleg over verschillende functies binnen OpenGL en wat deze precies inhouden

Website:	**OpenGL tutorial (gamedev)**
Adres	http://www.gamedev.net/reference/articles/article947.asp
Overzicht:	Deze website gaat wat dieper in op het gebruik van OpenGL technieken. Ook kunnen er van deze website stukken code gedownload worden

Website:	**Wikipedia website**
Adres	http://nl.wikipedia.org/
Overzicht:	Wikipedia heb ik gebruikt om globaal informatie te verzamelen over onderwerpen zoals Direct3D en OpenGL.

Woordenlijst

- 3D engine

  Een 3D engine is software die gebruikt kan worden voor allerlei grafische applicaties. Vaak wordt er als er een game geproduceerd word, gebruik gemaakt van een 3D engine om vervolgens een game mee te maken. Een 3D engine is dus eigenlijk de fundering voor een 3D applicatie.
  Een 3D engine is niet alleen verantwoordelijk voor het creëren van grafische output, maar voor alles wat een 3D applicatie nodig kan hebben. Het opvangen van Input (keyboard en muis), het inladen van externe bestanden (resourceManagement) vallen ook onder de verantwoordelijkheden van een 3D engine.

- Agile

  Agile is een verzameling best-practices waaronder Scrum valt.

- API (Application programming interface)

  Een API is een stuk software die het mogelijk maakt om met andere soft- of hardware te kunnen communiceren.

- backward compatibility

  Als een API backward compatible is, betekend dit dat oudere sofware die gebruik maakt van een oudere versie van deze API ook zal werken met de nieuwe versie van deze API.

  Voorbeeld:
  Software die gebruik maakt van OpenGL 1.1, zal ook zonder problemen draaien onder OpenGL 2.0 of 3.2. Dit komt doordat OpenGL backward compatible is.

- Blackbox-testing

  Bij een blackbox-test wordt een functie getest op functionaliteit. In tegenstelling tot een whitebox-test waar alle individuele broncode wordt getest.

- Cg

  Library die gebruikt wordt om shaders toe te kunnen passen. Vaak wordt Cg gebruikt binnen een 3D engine maar dit is niet vereist.

- CGFX

  Programmeertaal waarin shaders worden geschreven.

- Compileren

  Het omzetten van broncode naar een taal die rechtstreeks door een computer kan worden uitgevoerd.

- Crossplatform

  Zodra software crossplatform is, betekend dit dat deze zal werken onder verschillende besturingssystemen. Voorbeelden van besturingsystemen zijn: Windows® MAC OS® en Linux.

- Direct3D

  Direct3D is een library die de programmeur in staat stelt te communiceren met de grafische kaart binnen een computer systeem.

- Library

  Software die het programmeren voor de programmeur gemakkelijk kan maken of die de programmeur in staat stelt bepaalde functionaliteit te gebruiken. Polemos bestaat uit 15 libraries die de programmeur in staat stellen bij bepaalde te komen.

- OpenGL

  OpenGL staat voor Open Graphics Library (OGL). Het Is een stuk software die programmeurs in staat stelt om te kunnen programmeren op de grafische kaart (GPU).

- OpenSource

  Als een applicatie OpenSource is, betekend dit dat de broncode hiervan beschikbaar is. Op deze manier kan de software worden gekopieerd of aangepast zonder hier extra voor te hoeven betalen.

- Polemos

  Polemos is de naam van een 3D engine die al jaren wordt onderhouden door studenten van de Academie voor ICT & Media in Zoetermeer.

- Scrum

  Scrum is een raamwerk van allerlei technieken die gericht zijn op de ontwikkeling van software.

- Shader

  Instructies voor een 3D engine om effecten kunnen berekenen. Wordt gebruikt voor Cg/CGFX.

- Static-linking

  Static-linking is het proces wat er gebeurd zodra een library aan een andere library wordt gelinkt door de compiler. Dit kan voorkomen als de library gelinkt wordt aan een statische interface.

- Tutorial

  Praktische oefening met doel iets te leren of onder de knie te krijgen.

- Unittests

  Bij unittests word een bepaald stuk broncode van een programmeertaal (vaak een gehele functie) getest op functionaliteit.

- Visual Studio

  De programmeeromgeving waarin ik dit project heb geprogrammeerd. Visual Studio heeft onder andere een compiler in zich en biedt ondersteuning voor C++ broncode.

- Whitebox-testing
  - Bij een whitebox-test wordt de broncode van een applicatie getest. Deze test ligt in het verlengde van een blackbox-test.

Afbeelding index

Bijlagen

De bijlagen zijn extern opgenomen in het document: 3D renderer implementatie met OpenGL – Bijlagen.

- Bijlage 1 – Afstudeerplan
- Bijlage 2 – Plan van Aanpak
- Bijlage 3 – Onderzoek Verschillen Direct3D en OpenGL
- Bijlage 4 – Rendererdocumentatie
- Bijlage 5 – CXXtest framework aanpassingen
- Bijlage 6 – Adviesrapport
- Bijlage 7 – Sprint-backlogs
- Bijlage 8 – Product-backlog
- Bijlage 9 – Burndown charts