Architectuur en Algoritmen van Computer Games

Hovercraft Universe

http://uhasseltaacgua.googlecode.com/

Olivier Berghmans, Nick De Frangh, Dirk Delahaye,
Kristof Overdulve, Pieter-Jan Pintens, Tobias Van Bladel

1 juni 2010

Inhoudsopgave

1 Introductie

Hovercraft Universe is een race game waarin spelers tussen planeten in de ruimte kunnen rondvliegen met hovercrafts. Het nadeel aan traditionele race games is dat de camera en game play nogal statisch en weinig beweeglijk is. Dit spel is revolutionair in het opzicht dat wanneer je van planeet naar planeet springt de speler in een andere atmosfeer komt met een ander niveau van zwaartekracht en andere eigenschappen.

2 Gebruikte libraries

Physics De fysische gamewereld wordt gesimuleerd door Havok Physics[1]. Havok draait enkel op de server en zorgt voor alle berekeningen die nodig zijn voor fysische interactie.

Rendering Voor het renderen gebruiken we de Ogre Open Source Graphics Engine[2]. Ogre draait enkel op de clients en zorgt voor de visuele representatie van de game-entiteiten. Andere, uitsluitend visuele, effecten en overlays worden ook door Ogre afgehandeld: particle effects, tekst overlays boven de entiteiten, ... SkyX[3] en OgreMax[4].

Netwerk ZoidCom[5]

Input Object Oriented Input System (OIS)[6]

Geluid FMOD Interactive Audio Middelware[7]

Scripting Lua[8] en LuaBind[9] bieden de mogelijkheid om op een dynamische manier eigenschappen van het spel te veranderen.

GUI Adobe Flash[10] en Hikari[11].

3 Programmastructuur en -organisatie

3.1 Algemeen

De basisstructuur van de engine achter Hovercraft Universe is gemaakt naar analogie met de architectuur van *Shellshock: Nam '67* [**?**]. Elk object in de spelwereld waarmee geïnterageerd kan worden, wordt voorgesteld door een `Entity`.

[1] http://www.havok.com/index.php?page=havok-physics
[2] http://www.ogre3d.org/
[3] http://www.ogre3d.org/forums/viewtopic.php?t=48414&f=11
[4] http://www.ogremax.com/
[5] http://www.zoidcom.com/
[6] http://sourceforge.net/projects/wgois/
[7] http://www.fmod.org/
[8] http://www.lua.org/
[9] http://www.rasterbar.com/products/luabind.html
[10] http://www.adobe.com/products/flash/
[11] http://code.google.com/p/hikari-library/

Deze entiteiten worden gegroepeerd in de `EntityManager`, die de entiteiten tijdens het spel kan updaten. In onze implementatie is een `Entity` tevens een uitbreiding van `NetworkEntity`, die replicatie over het netwerk mogelijk maakt.

Entiteiten kunnen bestuurd worden door `Controllers`. Een controller genereert *events*, die door de entiteiten gepolld worden. Deze interface laat toe om entiteiten te besturen op verschillende manieren: toetsenbord/muis, AI, ...

Voor de visuele representatie op de clients heeft elke relevante entiteit een `EntityRepresentation`. Deze verbindt de conceptuele entiteit met een entiteit in Ogre en is gegroepeerd in een `RepresentationManager`. Om de wereld te kunnen tekenen vanuit het standpunt van één speler worden deze representatieobjecten, samen met de Ogre camera, bijgehouden in een `GameView`. Deze `GameView` is tevens verantwoordelijk voor het tekenen van de 2D HUD per speler. In tegenstelling tot het originele design wordt hier niet gebruik gemaakt van fixed update frequencies. De visualisatie van objecten gebeurt zo snel mogelijk en wordt dus aangevat in de `RepresentationManager` wanneer de vorige rendering cyclus voorbij is. Updates voor entities worden daarentegen geüpdatete als de `Server` wijzigingen aankondigt.

De bovenstaande klassen zitten vervat binnen het `CoreEngine`-project, dat in principe voor eender welk soort spel gebruikt kan worden. Voor Hovercraft Universe is een specifieke implementatie voorzien: de `HovercraftController`, waarvan de `HovercraftAIController` en `HovercraftPlayerController` afgeleid worden.

Iets over de application / server application structuur

3.2 Netwerkstructuur

Server, client, netwerk, zoidcom, chatfunctionaliteit, replicatie, interpolatie, ...

3.3 Modellen en user-data

Olivier!

De **modellen** zijn gedefinieerd in het (niet open) bestandsformaat van 3DS Max. Deze worden geëxporteerd naar het DotScene XML-formaat met behulp van de OgreMax Scene Exporter[12]. DotScene[13] is een gestandaardiseerd XML-formaat dat informatie bevat over het soort entiteit, de grootte van het model, de gebruikte materialen en meshfiles, etc. Tevens bevat dit DotScene bestand informatie specifiek voor gebruik in HovercraftUniverse (snelheid, gewicht, ...). Een voorbeeld van deze user-data is gegeven in Listing 1.

Listing 1: UserData in DotScene

```
<Hovercraft>
    <Name>USA Hovercraft</Name>
    <Description>Fast and responsive, rather light.
```

[12] http://www.ogremax.com/
[13] http://www.ogre3d.org/wiki/index.php/DotScene

```
        </Description>
    <OgreEntity>USAHovercraft</OgreEntity>
    <ProcessInterval>0.016</ProcessInterval>
    <Speed>100</Speed>
    <Mass>50</Mass>
    <Acceleration>10</Acceleration>
    <Steering>100</Steering>
</Hovercraft>
```

Voor de **mappen** (tracks) wordt er één groot DotScene wereld gegenereerd waarin alle relevante nodes zijn gedefinieerd: planeten, checkpoints, statische objecten, startposities, etc. Een voorbeeld van een dergelijke node is te zien in Listing 2

Listing 2: Eén van de nodes in de DotScene van een wereld.

```
<node name="Jump01">
    <position x="-148.891" y="21.0134" z="-20.245" />
    <scale x="1" y="1" z="1" />
    <rotation qx="0.13454" qy="0.990908"
        qz="-5.34715e-008" qw="-7.48115e-008" />
    <entity name="Jump01" castShadows="true"
            receiveShadows="true" meshFile="Jump01.mesh">
        <userData>
            <![CDATA[<OgreEntity><GameEntity>Jump01_ent
                </GameEntity></OgreEntity>]]>
        </userData>
        <subentities>
            <subentity index="0" materialName="Jump02" />
        </subentities>
    </entity>
</node>
```

Door middel van scene loaders kunnen objecten gepositioneerd en getransformeerd worden om volledige scènes te creëren. Het spel laat toe volledig uitgebreid en aangepast te worden door data files aan te passen zonder nood aan hercompilatie. Dit data-driven model beperkt zich niet alleen tot het toevoegen van werelden, maar ook tot het invoegen van nieuwe hovercrafts en characters. Tevens kan de kwaliteit van de werelden dynamisch worden aangepast door schaduws, lichten en materials aan en uit te zetten om zo ook oudere systemen te ondersteunen.

3.4 Visualisatie en Ogre

Ook worden hier een aantal visuele effecten aan toegevoegd. Ogre Particle Systems worden gebruikt om rook, vonken, etc. weer te geven. Er is tevens een tekst-overlay voor de nicknames voorzien zodat de spelers elkaar kunnen herkennen tijdens het spel. Aangezien deze effecten enkel client-side gebeuren bij het tekenen van de entiteiten, heeft dit geen invloed op de performantie van de netwerkcomponenten.

Schaduws en lichteffecten kunnen zoals voordien dynamisch aangepast worden. SkyX wordt gebruikt om dag- en nacht cycli te renderen en om wolken en sterrenhemels te tonen. Omdat het spel zich in de ruimte afspeelt wordt er enkel gerenderd tussen 22.00 en 5.00 uur. Beperkend aan SkyX is echter dat in de huidige vorm enkel hemels worden gerenderd en dus geen hele universe vol sterren. OgreMax[14] wordt gebruikt om modellen te exporteren vanuit 3DS Max en te importeren in Ogre.

Hovercraft Universe ondersteunt vier verschillende camera's: een third-person view waarbij de hovercraft zelf zichtbaar en waarbij een groot deel van de wereld wordt getoond. First-person view geeft een beter snelheidsgevoel en verhoogt het inlevingsvermogen van de speler. Vervolgens is er een rear-view camera en een free-roaming camera om de omgeving te verkennen.

Material mapping wordt gedetailleerd uitgevoerd door middel van UVW-unwrapping. Niet alleen is dit erg flexibel om gecontroleerd materials te geven aan modellen, ook laat dit single-pass GPU pixel shading toe en vergemakkelijkt dit het proces voor LoD.

Het gebruik van pixel shaders wordt extensief aangewend om alpha splatting te verwezenlijken en dus de overgang tussen texture tiles vloeiend te laten overkomen.

3.5 Physics

Alle acties op de hovercrafts en interacties met de omgeving gebeuren door middel van fysische krachten. Dit gebeurt volledig op de server, die een Havok-thread draait. De input van de clients, zowel menselijke spelers als AI, wordt vertaald in een kracht op de hovercraft. Havok berekent vervolgens alle totale krachten en daaruit volgende versnellingen en snelheden. Hieruit zal de nieuwe positie van alle entiteiten bepaald worden. Deze posities worden dan doorgestuurd naar alle clients zodat zij de wijzigingen kunnen zien. Bovendien zorgt Havok ook voor de collision detection zodat verschillende entiteiten niet tegelijk op dezelfde plaats kunnen bestaan.

De zwaartekracht op de verschillende planeten is afgeleid van het `PlanetGravity` voorbeeld van Havok. Concreet wordt er rond elke planeet een *phantom* geplaatst. Wanneer een object binnen deze phantom komt, wordt er een kracht uitgeoefend waardoor de objecten aangetrokken worden door de planeet. Deze kracht kan per planeet verschillen zodat elke planeet een eigen zwaartekrachtsterkte heeft.

Op elke hovercraft-entiteit wordt nog een extra actie gedefiniëerd om het object te laten zweven. Dit is een kracht die in tegengestelde richting ten opzichte van de zwaartekracht werkt. Afhankelijk van de hoogte verandert de grootte van de kracht. Dicht bij het planeetoppervlak zal er een sterke tegengestelde kracht gebruikt worden. Hoe verder van de planeet, hoe zwakker de kracht wordt. Vanaf een bepaalde hoogte valt de kracht weg, zodat de hovercraft terug naar beneden begint te vallen. Op deze manier krijgen we een lichte oscillatie van de hoveringhoogte.

[14] http://www.ogremax.com/

3.6 Grafische user interface

De onderdelen van de GUI zijn geïmplementeerd in ActionScript[15], en worden met behulp van Hikari gevisualiseerd in Ogre. Hikari is een library die Flash (SWF) bestanden kan inlezen, en voor de binding tussen Flash en C++ zorgt. Hikari is aangepast en uitgebreid, oa. met de functionaliteit om overlays uit te schakelen.

De `BasicOverlay` klasse bevat alle basisfunctionaliteit en stelt een enkel Flash-bestand voor. Deze kan een bepaalde positie en dimensies innemen.

De HUD van het spel is geconfigureerd aan de hand van een XML-bestand. Gebruikers kunnen hun eigen HUD zo aanpassen, instellen en eventueel andere HUD-elementen gebruiken. Listing 3 toont een voorbeeld van de configuratie van een HUD-element.

Listing 3: HUD configuratie van de LapTimer

```
<element type="timer" file="lapTimer.swf">
    <size>
        <min width="174" height="40" />
            <max width="520" height="120" />
    </size>
    <position>
        <relative val="TopRight" />
    </position>
</element>
```

3.7 Geluid

De FMOD library en tools zijn gebruikt voor het geluid en de muziek. De muziek is onderverdeeld Deze queues worden georganiseerd in een stack. De muziek en geluidseffecten worden door de FMOD Designer gecompileerd in één enkel .fsb bestand.

Om toe te laten dat entiteiten in de wereld geluid produceren is `Moveable-3DEmitter` gemaakt. Deze is een wrapper voor de FMOD functionaliteit, en bepaalt het geluid voor een entiteit in de wereld aan de hand van de positie, snelheid, orientatie. In onze architectuur zorgen de `EntityRepresentation`s voor de specifieke implementatie van het 3D geluid.

3.8 Besturing

De centrale klasse voor de besturing is de `KeyManager`. In deze klasse kunnen alle nodige acties geregistreerd worden door middel van een uniek ID, bijvoorbeeld in een enumeratie. Aan elk van deze acties kunnen één of meerdere knoppen gebonden worden. Telkens wanneer de `HovercraftPlayerController` een key event binnenkrijgt, zal er bij de key manager nagegaan worden welke actie er

[15] http://www.adobe.com/devnet/actionscript/

overeenkomt met deze toets. Op deze manier is het mogelijk om de spelbesturing gemakkelijk te wijzigen zonder dat de controller aangepast moet worden. Bovendien kunnen er meerdere toetsen voor dezelfde actie gebruikt worden.

In de `KeyManager` staat ook nog een mapping tussen alle toetsen die OIS ondersteunt en hun bijhorende string-representatie. Zo kunnen de controls ingelezen of weggeschreven worden naar een configuratiebestand. Standaard is dit bestand te vinden in de data\controls map, maar wanneer dit bestand ontbreekt zal er automatisch een nieuw aangemaakt worden met de default controls. In dit configuratiebestand staan alle geregistreerde acties per categorie. Voor elke actie kunnen één of meerdere toetsen opgegeven worden, gescheiden door een komma. Wanneer een actie ontbreekt, zal de standaardtoets voor deze actie gebruikt worden.

3.9 Scripting en AI

Voor de bots is een AI voorzien, die "menselijk" stuurgedrag nabootst. De algoritmen van deze AI zijn gebaseerd op de autonome stuursimulaties van Craig Reynolds [**?**]. De huidige AI is in staat om een voorgedefinieerd pad te volgen binnen een bepaalde marge ("Path Following"). Een pad wordt voorgesteld als een *multiline*: een lijst van punten in de wereld, telkens geassocieerd met een straal (`radius`). De straal op punt x_i stelt de *breedte* van het pad voor tussen punt x_i e, x_{i+1}. Deze paden worden opgeslagen in aparte bestanden. Dit betekent dat de level designers per map een ideaal pad (of meerdere paden) voor de AI kunnen aanmaken.

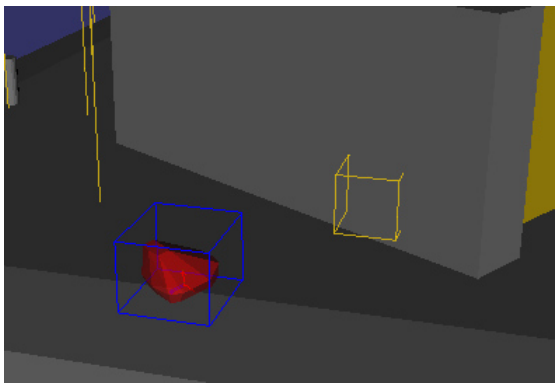
De AI is geprogrammeerd in Lua [**?**], om aanpassing van de AI-algoritmen mogelijk te maken zonder dat het spel hiervoor gehercompileerd moet worden. Voor de binding tussen Lua en C++ wordt Luabind gebruikt. De `ScriptWrapper` klasse abstraheert deze scripts en laat de programmeurs toe om met de scripts te werken, zonder Lua-specifieke constructies te moeten gebruiken.

Collision avoidance voor de AI is ook geïmplementeerd met behulp van Havok physics. Er zijn twee implementaties beschikbaar voor de `EntityCollision` interface. Het eenvoudige model (zie Figuur 1) plaatst een kubus op een bepaalde afstand van het voertuig om objecten te detecteren. Dit model gebruikt zeer weinig resources, maar is ook niet zo krachtig. Het geavanceerde model gebruikt een 3D-lichaam dat zich uitstrekt van het voertuig tot de gewenste afstand voor de figuur (zie Figuur 2). Deze manier van werken heeft een kleine meerkost in resources in vergelijking met het eenvoudige model.

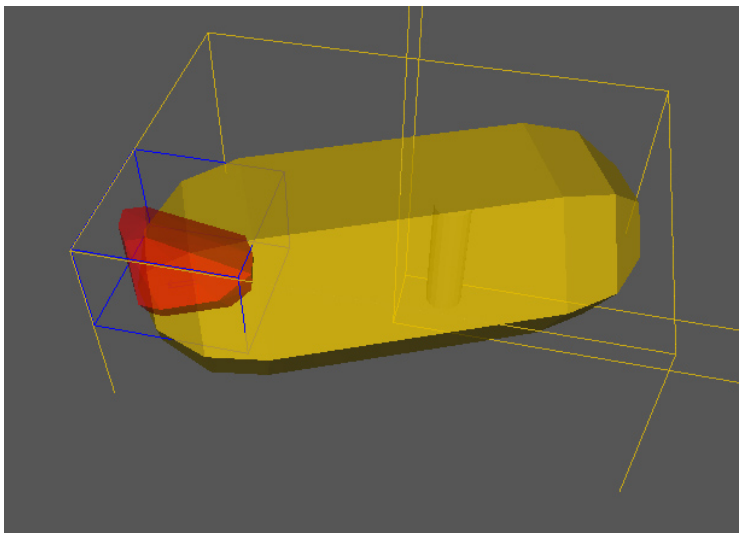
Aangezien deze methodes afhankelijk zijn van de physics, draaien ze op de server. Als een object wordt gedetecteerd, wordt dit door middel van een event doorgegeven aan een callback functie in de entiteit. Dit wordt vertaald naar een boolean in de entiteit, die door het netwerk wordt gerepliceerd op de client. Client-side maakt de AI dan een beslissing om het obstakel te ontwijken.

3.10 Game Logic

Lobby, admins? of bij netwerken



Figuur 1: Het eenvoudige collision avoidance model



Figuur 2: Het geavanceerde collision avoidance model

Raceplayer, Racestate, positiebepaling

3.11 Level of Detail

3.11.1 Netwerk

De snelheid waarin updates worden gebroadcast naar andere avatars wordt dynamisch aangepast door middel van prioriteiten. Zoidcom bevat functionaliteit om bepaalde objecten relevanter of minder relevant te maken zodat deze sneller of minder snel worden geupdatete en worden doorgestuurd naar de clients. Deze relevantie wordt in Hovercraft Universe enerzijds bepaald door de afstand in wereldcoördinaten, maar ook door de planeten waar we op zitten. Updates van andere objecten worden sneller ontvangen als deze zich op dezelfde planeet vinden dan wanneer ze zich op een andere planeet bevinden.

Om de hoeveelheid updates te verminderen wordt er gebruik gemaakt van dead reckoning. Concreet betekent dit dat er een positie en een richting wordt doorgestuurd naar de clients en dat deze zelf berekenen waar de entiteiten zich de volgende frames zullen bevinden tot er een nieuwe update binnenkomt van de server en de correcte nieuwe positie en richting wordt ingesteld. Het berekende pad wordt dan geïnterpoleerd naar de geüpdatete posities. Door dit toe te passen kan de benodigde frequentie van de updates sterk verlaagd worden. Een eventuele discrepantie met betrekking tot bijvoorbeeld collision detection wordt opgelost doordat de hele collision engine server-based gebeurt en deze inherent een consistente staat van de wereld bevat. Bij bijvoorbeeld botsingen worden dan events gegenereerd die onmiddelijk de staat op de client aanpassen zodat kritische acties meteen bij alle relevante clients zichtbaar worden.

3.11.2 Rendering

Ogre bevat reeds een standaard aantal algoritmes voor LoD. Dit wordt zoals reeds vermeld in de hand gewerkt door een slim gebruik van textures. De mate waarin LoD wordt toegepast kan dynamisch worden aangepast in configuratiebestanden.

3.12 Misc.

3.12.1 Configuratiebestanden

Configuratiebestanden zijn beschikbaar voor de server en client, in INI formaat (zie Listing 4). Dit laat gebruikers toe om hun nickname, gekozen voertuig, enz. te kiezen en op te slaan. Een aantal server-specifieke instellingen worden ook ondersteund (aantal spelers, AI-script, sterkte van de zwaartekracht, ...). Het spel kan deze bestanden ook opslaan met de huidige instellingen.

Listing 4: Het INI bestandsformaat

```
[section name]
keyname=value
```

```
#comment
keyname = value , value , value #comment
```

4 Rolverdeling en verantwoordelijkheden

Kristof Overdulve Algemene client rendering architectuur, Ogre visualisatie en animatie, 2D artist, 3D modellering

Pieter-Jan Pintens Havok Physics, 3D modellering, Ogre visualisatie en animatie

Dirk Delahaye Scripting, AI, visuele effecten, configuratiebestanden

Tobias Van Bladel Input, controls, physics

Nick De Frangh Grafische user interface, Geluid

Olivier Berghmans Algemene server architectuur, netwerk

Referenties