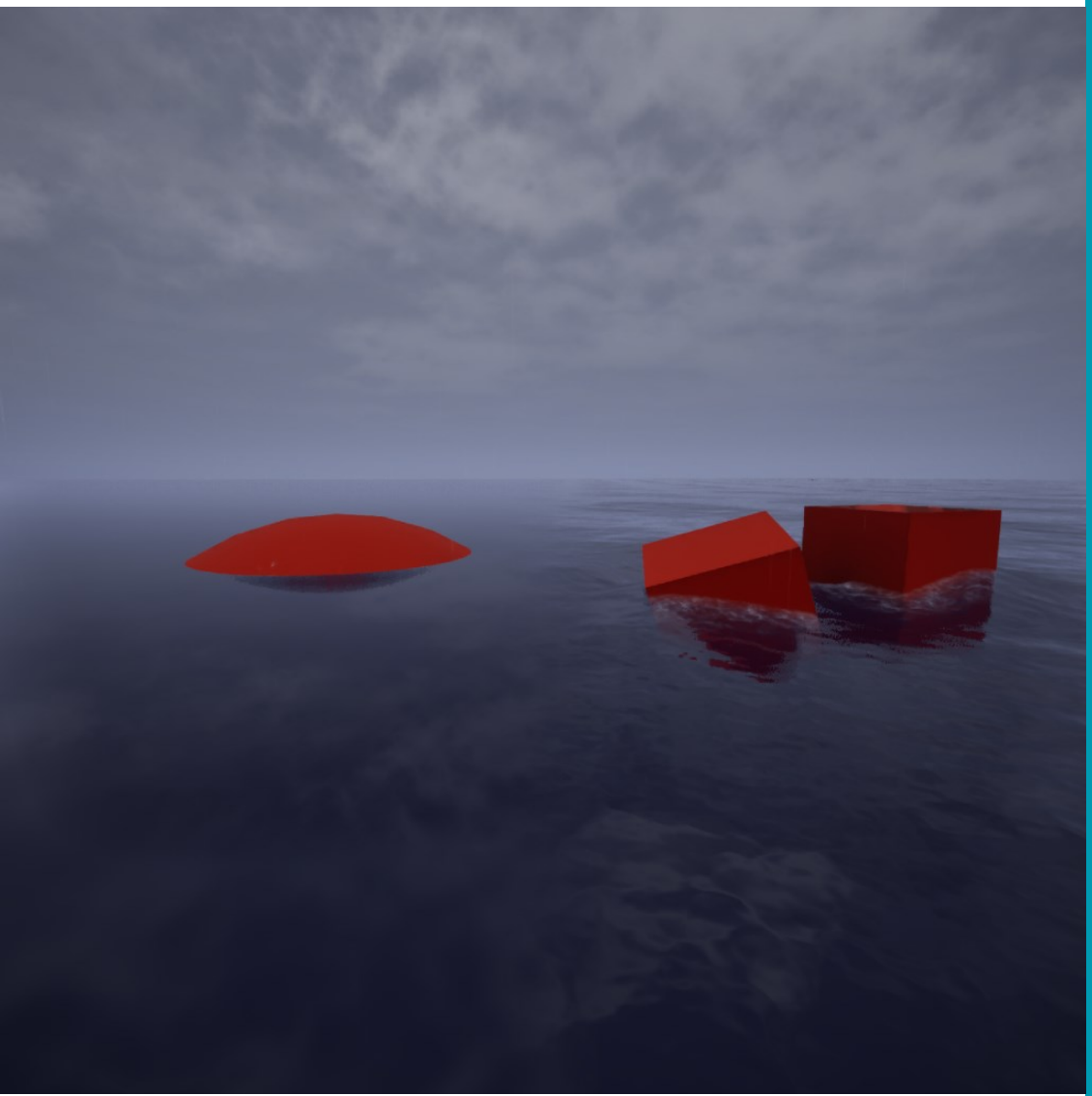
Tommi Järvinen

Interaktiivisen vesimateriaalin luominen Unreal Engine 5 -pelimoottorilla



Tradenomi

Tietojenkäsittely

Syksy 2024

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Järvinen Tommi

Työn nimi: Interaktiivisen vesimateriaalin luominen Unreal Engine 5 -pelimoottorilla

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: Unreal Engine, materiaali, interaktio, pelinkehitys

Opinnäytetyön tavoitteena oli tutkia ja kasvattaa ymmärrystä vuorovaikutuksen luomisesta Unreal Engine 5 -pelimoottoria ja sen sisältämää materiaalieditoria käyttäen. Materiaalieditorin kautta luotavien varjostinimien lisäksi käytettiin myös C++-ohjelmointikieltä. Tarkoituksena ei ollut luoda valmista vesimateriaalia, vaan sen sijaan käydä läpi, mitä tekniikoita käyttämällä pystytään luomaan illuusiota veden kuulumisesta pelimaailmaan. Tutkittaessa erilaisia tekniikoita päädyttiin hakemaan tietoa niin kirjallisuudesta kuin erimuotoisista sähköisistä lähteistäkin. Sähköiset lähteet koostuivat pääsääntöisesti pelimoottorin omasta dokumentaatiosta sekä alan ihmisten julkisesti jakamasta tiedosta. Nämä yhdistettynä koulun aikana opittuihin taitoihin loivat opinnäytetyön pohjan.

Erilaiset interaktioon vaikuttavat tekniikat jaettiin niin staattisiin kuin dynaamisiinkin osioihin. Staattisen osion puolella käsiteltiin tekstuurien manipuloimista objektin pinnalla tekstuurikoordinaatistoa hyödyntämällä. Tämän lisäksi osoitettiin, miten interaktion illuusiota pystytään kasvattamaan lisäten pelin aikana tapahtuvaa suunnan muutosta C++-ohjelmointikieltä hyödyntämällä. Näiden lisäksi käsiteltiin virtauskarttojen merkitystä ja eroavaisuutta tekstuurin tavanomaiseen liikuttamiseen verrattuna.

Dynaamisiin osioihin sisällytettiin syvyyshäivytyssolmun tuomat käyttömahdollisuudet pelimoottorin ennalta laskemaa objektien välistä etäisyyttä hyödyntämällä. Tämän lisäksi käsiteltiin etäisyyskenttien tuomat mahdollisuudet pelin aikana tapahtuvien muutosten huomioonottamisessa, mikä luo dynaamisempaa kokonaisuutta. Viimeiseksi käytiin läpi, kuinka hahmontamiskohteita voidaan hyödyntää varjostinta luodessa.

Lopputuloksena päädyttiin toteamaan, vaikka erilaisia tekniikoita voidaankin hyödyntää pelimoottorin sisällä, eivät kaikki sovellu halutun lopputuloksen luomiseen. Siinä missä objektin pinnalla liikutettavien tekstuurien sekä esilaskettujen etäisyyksien hyödyntäminen mahdollistaa interaktion illuusion luomisen niin staattisessa kuin dynaamisessakin ympäristössä, hahmontamiskohteiden vahvuudet ovat muualla. Tutkitut tekniikat eivät ole rajattuja vain vesimateriaalin interaktion luomiseen, sillä ne ovat sovellettavissa myös monenlaisiin muihin pelin aikana näkyviin maailmassa tapahtuviin muutoksiin.

KAMK • University of Applied Sciences

Abstract

Author(s): Järvinen Tommi

Title of the Publication: Creating interactive water shader in Unreal Engine 5 game engine

Degree Title: Bachelor of Business Administration, Business Information Technology

Keywords: Unreal Engine, shader, interaction, game development

The goal of this thesis was to study and gain understanding about how to create interaction using the Unreal Engine 5 game engine and harvesting the possibilities of creating shaders with its built-in material editor accompanied by C++ programming language. The goal was not to create a complete water shader but instead to go through some possible techniques that can be used to create an illusion of water elements belonging to the game world. While studying different techniques it was decided to use various sources including literature as well as different forms of digital sources. Those sources were mainly collected from the game engine's own documentation as well as from the public information made available by people in the industry. This combined with the author's already-learned skills during the studies acted as a basis for this thesis.

The author categorized different interaction techniques to static and dynamic sections. During the static section it was addressed how texture manipulation could be handled using texture coordinates to alter the texture's appearance on the object's surface. Taking the coordinate manipulation further, the author addressed how to further enhance the illusion of generating interaction at runtime using C++ programming language. On top of these the usage of flow maps was addressed and how those differ from the more traditional way of moving textures.

Within the dynamic section the author included the usage of the depth fade node which takes advantage of the game engine's pre-calculated data to access the information about distance between objects. On top of that it was addressed how to use distance fields to alter the material and to create an illusion about a more dynamic looking interaction. Finally, the author went through how to make use of render targets when creating shaders.

As a result, it was concluded that although the game engine allows utilizing different techniques, not all of them are suitable for creating the desired result. Whereas the use of manipulated texture coordinates and taking advantage of pre-calculated distance calculations enables creating interaction in both static and dynamic environments, the strengths of render targets lie elsewhere. The techniques studied are not limited to being used only with water interaction as those can be adapted to various changes happening within the game world.

Sisällys

1 Johdanto

Tämän opinnäytetyön tarkoituksena on luoda tekniikoita avaten ymmärrystä interaktiivisen veden luomisesta Unreal Engine 5 -pelimoottorin materiaalieditoria ja C++-ohjelmointikieltä hyödyntäen. Ohjelmointi on näistä kuitenkin pienemmässä osassa, sillä materiaalieditorin laajuus mahdollistaa pääsyn suurimpaan osaan tarvittua dataa. Tarkoituksena ei ollut luoda valmista vesimateriaalia, mutta sen sijaan luoda kuvaa siitä, mitä jo olemassa olevan materiaalin päälle voisi rakentaa. Ajatus tämän opinnäytetyön aiheesta on rakentunut koulussa olon aikana, sillä mielenkiintoni on kasvanut materiaaleja ja näiden avulla luotavia varjostimia (engl. shaders) sekä niiden tuomia mahdollisuuksia kohtaan pelimaailman dynaamisuuden lisäämiseksi.

Koska materiaalieditorin tarkoituksena on luoda varjostimia, voidaan niillä määrittää, miltä objekti tulee näyttämään pelimaailmassa käyttäjän ruudulla. Varjostimet sisältävät erilaisia pinnan ominaisuuksien määritelmiä, kuten minkä värisiä ruudulle piirretyn objektin tulisi olla, mitä osia niistä tulisi näkyä sekä millä tavalla niiden tulisi reagoida valaistukseen. Toinen merkittävä varjostimista saatu hyöty on niiden mahdollistama laskentatehon siirtäminen tietokoneen prosessorilta näytönohjaimen puolelle, jolloin saadaan tarvittaessa vapautettua prosessorin laskentatehoa muihin sitä vaativiin asioihin.

Varjostimet eivät ole uusi ilmiö, sillä ne ovat olleet käytössä jo pitkään pelejä tehtäessä ja ovat keskeisessä osassa pelien graafisen ulkoasun luomisessa. Vaikka osa käsitellyistä menetelmistä on matemaattisesti pelimoottorista riippumattomia, käsittelin aiheeni kuitenkin mainitun pelimoottorin näkökulmasta. Tämän kokonaisuuden ymmärtämisen edesauttaminen oli yksi syy tämän opinnäytetyön aiheen valinnassa, sillä koin tämän edesauttavan omaa teknistä osaamistani materiaalieditoria hyödyntäen.

Aloitin raporttini käsittelemällä teoreettista viitekehystä, jonka tarkoituksena on luoda ymmärrystä keskeisiä varjostimien luomisessa käytettäviä osa-alueita kohtaan. Ennen toteuttavaa osiota esittelin perustelut kehitysalustan ja vaadittavien kriteerien täyttyminen tekniikoita määriteltäessä. Toteutusosan aikana kävin läpi erilaisia valittuja interaktion illuusion luomisen kannalta keskeisiä tekniikoita ja miten nämä soveltuvat kyseiseen menetelmään. Lopuksi kävin yhteenvetona läpi opinnäytetyön sisällön ja miten tämän tuottamisessa oli onnistuttu sekä mahdollisia jatkokehitysmahdollisuuksia.

2 Unreal Engine ja materiaalit

Moniin muihinkin aloihin verrattaessa pelien kehitykseen vaaditaan omat työkalunsa. Sen sijaan, että tehtäisiin kaikki alusta loppuun itse, on myös mahdollista käyttää muiden tuotoksia lopputuloksen saamiseksi. Päädyin valitsemaan omaksi työkalukseni Unreal Engine 5 -pelimoottorin tämän vakiintuneen aseman sekä oma mielenkiintoni kyseistä pelikehityksen työkalua kohtaan vuoksi.

2.1 Unreal Engine pelimoottorina

Unreal Engine on amerikkalaisen Tim Sweeneyn vuonna 1991 perustetun Epic Games -yrityksen tarjoama 3D-moottori, jota käytetään niin pelien kuin elokuvienkin ja muiden digitaalista grafiikkaa sisältävien tuotteiden kehitykseen. Kyseinen pelimoottori toimii kehitysalustana useille julkaistuille peleille, joista esimerkkinä yhtenä isoimpana pelinä tunnettu Fortnite (kuva 1). [1.]

Kuva 1. Esimerkkikuva Fortnite-pelistä [2]

Moottorin työkalut mahdollistavat materiaalien ja varjostimien luomisen materiaalieditorin solmupohjaisella käyttöliittymällä. Materiaalieditoria hyväksikäyttämällä voidaan määritellä ruudulla näkyviä ominaisuuksia Unreal Engine -työkalun muilla osa-alueilla, kuten erikoistehosteiden

luonnissa. Pelimoottori käyttää metrijärjestelmää ja yksi Unreal-yksikkö vastaa yhden senttimetrin pituista etäisyyttä [3]. Unreal Engine tukee niin C++-ohjelmointikielen käyttöä kuin Blueprint-pohjaista, solmuja hyväksikäyttävää visuaalista ohjelmointitapaa. [4, 5.]

2.2 Materiaalin, materiaali-instanssin ja dynaamisen materiaali-instanssin erot

Materiaalin tarkoituksena on kertoa pelimoottorille, miltä objektin tulisi näyttää pelimaailmassa. Materiaalieditoria hyötykäyttämällä voidaan määrittää objektikohtainen vaikutus pelimaailman valaistuksen kanssa tekstuureja ja matemaattisia laskelmia hyödyntäen. Materiaalit ovat vastuussa siitä, millaisia ominaisuuksia objektilla tulisi olla esimerkiksi värin, pinnan tasaisuuden sekä heijastavuuden osalta. [6.]

Materiaali-instanssilla tarkoitetaan päämateriaalista johdettua materiaaliversiota. Uuden materiaalin luominen jokaiselle objektille on ajallisesti tehotonta, sillä monet materiaalit käyttävät samanlaisia ratkaisuja visuaalisen ilmeen luomisessa. Instanssin avulla voidaan muuttaa materiaalissa olevia julkisia parametreja, jolloin muunnelmia luodessa ei tarvitse täysin uutta materiaalia, vaan voidaan hyödyntää jo olemassa olevaa tuotosta. [7.]

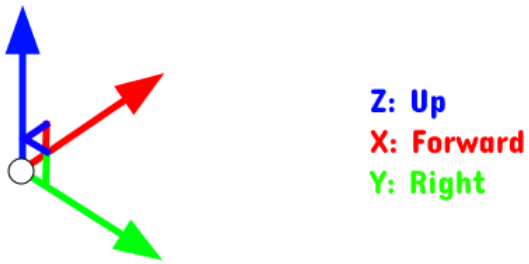
Materiaali-instanssia voidaan muokata ennen pelin ajoa muuttamatta päämateriaalia, mutta muutosta ei voida enää tehdä pelin alkaessa. Mikäli materiaalia halutaan muuttaa pelin aikana, tulee luoda dynaaminen materiaali-instanssi. Siinä missä materiaali ja materiaali-instanssi luodaan pelimoottorin sisäisellä käyttöliittymällä, dynaaminen materiaali-instanssi luodaan koodissa näiden pohjalta, minkä jälkeen päästään käsiksi instanssissa oleviin julkisiin parametreihin. [8.]

2.3 Materiaaliparametrikokoelma

Materiaaliparametrikokoelma on Unreal Enginen tapa määritellä ulkoisia muuttujia, joita voidaan käyttää materiaalien sisällä. Jokainen materiaali voi käyttää hyväkseen kyseisessä kokoelmassa määriteltyjä muuttujia, joita muuttamalla kaikki materiaalit päivittyvät ilman, että niitä tarvitsee yksitellen käsitellä erikseen. Parametrikokoelmien rajoitteena on, että ne voivat sisältää vain skalaari- ja vektoriarvoja. [9.]

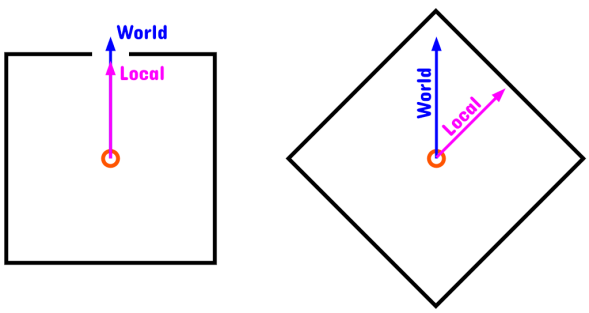
2.4 Pelimoottorin koordinaatistojärjestelmien merkitys materiaalia rakentaessa

Objektia tarkastellessa ja varjostinta luodessa on tärkeää ymmärtää erilaisten tilojen erot. Unreal Engine käyttää niin sanottua vasemman käden koordinaatistoa. Tällä tarkoitetaan muistisääntöä, jolla peukalon, etusormen ja keskisormen asentoon asettamisella voidaan havainnollistaa ohjelmiston koordinaatistoakselien suunnat. Z-akseli ylös, X-akseli eteen ja Y-akseli sivulle (kuva 2). [10.]



Kuva 2. Pelimoottorin koordinaatistoakselin havainnollistaminen.

Maailman tilalla tarkoitetaan paikan omaa koordinaatistoa. Objektia kiertäessä koordinaatisto pysyy paikallaan, eikä sen suuntaan pystytä vaikuttamaan objektitasolla. Paikallisesta tilasta puhuttaessa voidaan ajatella tilannetta, jolloin objektin kierto tapahtuu suhteessa sen napapisteeseen (engl. pivot point). Toisin kuin maailman tilassa, ylöspäin osoittava paikallinen koordinaatistoakseli kiertää objektia tämän kiertoa mukaillen (kuva 3). [11, s. 47.]



Kuva 3. Maailman ja paikallisen koordinaatistoakselin havainnollistaminen.

2.5 Näytönohjaimen merkitys varjostinta luodessa

Varjostin yhdistää matematiikan virtuaalisen taiteen luomisessa. Tuottamalla koodia voidaan määrittää, minkä värisinä pikselien tulisi näkyä ruudulla. Varjostimet myös määrittelevät materiaalin ominaisuuksia, joiden mukaan pelimoottori osaa laskea oikeanlaisen lopputuloksen annetun valaistuksen kannalta. [12, 1:47.]

Kärkivarjostimen (engl. vertex shader) osio käydään ohjelmoitavan kärkiprosessorin puolella, mikä ottaa sisäänsä kärjen datan ja muuntaa tämän sopivaan muotoon ruudulla nähtäväksi. Näin ollen sitä voidaan myös käyttää kärkiin liitettäviin toimintoihin, mikäli nämä tapahtuvat kärkidatan mahdollistavissa puitteissa. Kärkidata sisältää kärkien uv-koordinaattitiedot, kärjen väridatan, normaalit ja kärkien pisteiden paikan objektin paikallisessa tilassa. [13, 1:20.]

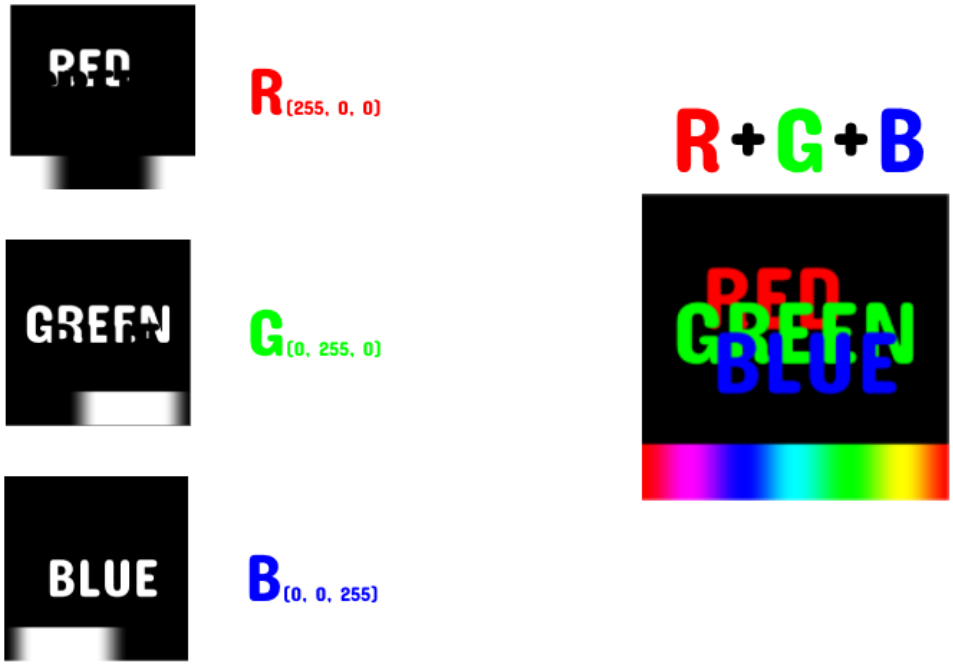
Toinen varjostimen luonnin kannalta tärkeä osio on ohjelmoitava pikseliprosessori. Tämä sisältää kärkivarjostimen antaman tuloksen ja ylimääräisen suorittimelta saadun tiedon, joista saatujen tietojen pohjalta voidaan tarkentaa haluttu pikselin väri, joka piirretään ruudulle ennen pelimaailmassa olevien valojen laskemista. [13, 4:19.]

3 Tekstuurit

Tekstuurit ovat keskeinen osa varjostimien luonnissa, sillä ne mahdollistavat monenlaisia ennalta määrättyjä ominaisuuksia. Tekstuureita on monenlaisia ja jokaisella on oma tehtävänsä lopputulosta rakentaessa, minkä takia on tärkeää ymmärtää, mitä tekstuurit ovat ja mitkä ovat niiden keskinäisiä eroavaisuuksia.

3.1 Tekstuurit pelimoottorissa

Tekstuuri on materiaaleissa käytettävä kuvatiedosto, jota voidaan hyödyntää määrittäessä erilaisia ominaisuuksia varjostimia luodessa. Tekstuurit sisältävät useamman tekstuurikanavan (R, G, B, A), joita käyttämällä voidaan tekstuuria käyttää erilaisiin tarkoituksiin. Jokainen kanava sisältää data-arvon nollan ja yhden välillä. Tätä voidaan visualisoida mustavalkoisella kuvalla, jossa täysin musta kuvaa arvoa nolla ja täysin valkoinen kuvaa arvoa yksi. Toinen mahdollinen rajaus on nollan ja 255:n välillä (kuva 4). Esimerkiksi väridataa halutessaan voidaan käyttää tekstuurin RBG-kanavia, jotka yhdistämällä saadaan siirrettyä väri-informaatio käyttäjän ruudulle. [14.]



Kuva 4. Tekstuurin yksittäisten kanavien havainnollistaminen.

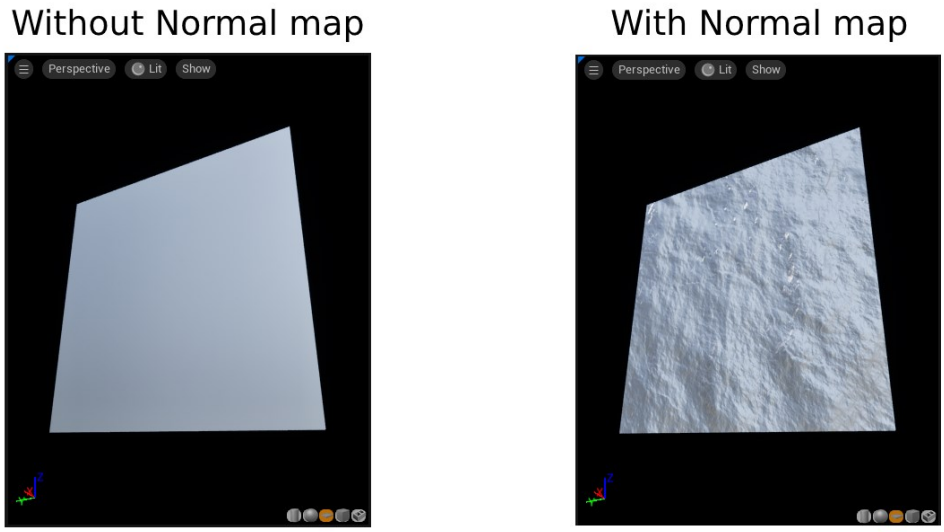
3.2 Tekstuurikoordinaatisto

Tekstuurikoordinaatilla tarkoitetaan viittausta tekstuurin pikseliin, jota käytetään apuna kaksiulotteista tekstuuria kartoittaessa kolmiulotteisin objektin pinnalle. Jotta voidaan erottaa ruudulla näkyvät pikselit tekstuurin omista pikseleistä, käytetään tekstuurien pikseleistä yleisemmin termiä texel. Jokaisen kolmiulotteisen objektin kärjen data sisältää paikallisen koordinaatiston (x, y, z) lisäksi koordinaatiston (s, t), joiden mukaan näytönohjain osaa piirtää tekstuurin objektin pinnalle. [11, s. 106.]

Asian yksinkertaistamiseksi oletetaan tekstuurin olevan suorakulmion muotoinen, jossa kaikki reunat ovat yhtä pitkiä sekä tekstuurikoordinaatiston arvojen sijoittuvan nollan ja yhden välille. Selkeyden vuoksi kaksiulotteisesta koordinaatistosta (s, t) käytetään tekstuurikoordinaatistosta puhuttaessa muotoa (u, v) [13, 0:25]. Unreal Engine kartoittaa (u, v) -arvon nollapisteen (0, 0) vasempaan yläkulmaan ja pisteen (1, 1) oikeaan alakulmaan. Tämä voidaan todistaa manipuloimalla koordinaatistoa lisäämällä tähän arvoja, joka siirtää tekstuuria objektin pinnalla halutun määrän. [15, 7:43.]

3.3 Normaalitekstuurit

Normaalitekstuurit voidaan käsittää tapana kertoa pelimoottorille, kuinka valon tulisi heijastaa objektin pinnalta ilman, että objektissa tarvitsee olla ylimääräistä geometriaa (kuva 5). Tämä antaa illuusion, että objektissa olisi enemmän yksityiskohtaisia pinnan muutoksia, kuten esimerkiksi tiileissä tai golfpalloissa esiintyvät pintojen kumpuisuudet. Normaalien päätehtävä on siis vaikuttaa valojen laskentaan kuvaa ruudulle piirrettäessä ilman sen vaikuttamista objektin siluettiin. [16, s. 236–237.]



Kuva 5. Normaalikartan havainnollistaminen pelimoottorin mukana tulevalla tekstuurilla.

Normaalikartta käännetään pelimoottorin puolesta tekstuuridatasta vektoridataksi, jotta sitä voitaisiin käyttää normaalien laskemiseen. Sen sijaan, että data tallennetaan tekstuurikoordinaatiston (0–1) välille, laajentaa pelimoottori normaalin (-1–1) välille. Kyseiset vektorit ovat saatavilla tekstuurin R-, G- ja B-kanavilta. [17, 8:35.]

3.4 Virtuaaliset tekstuurit

Runtime Virtual Texture on pelimoottorrissa käytettävä tekstuurityyppi, joka toimii muuten kuin normaalitekstuuri, mutta se luo texel-datan näytönohjaimelle pelin aikana. Saatu tieto voidaan laittaa välimuistiin, millä voidaan auttaa tarvittavan laskentatehon vaatimuksissa. Jotta tämän tyylistä tekstuuria voitaisiin käyttää, tulee virtuaaliset tekstuurit olla aktivoituna pelimoottorin projektiasetuksista. Maailmassa tulee myös olla asetettuna Runtime Virtual Texture Volume, joka mahdollistaa tämän sisällä olevien asioiden olevan luettavia virtuaalisille tekstuureille. [18.]

4 Kehitystyökalujen valitseminen

Ennen tekniikkojen esittämistä halusin tuoda perusteluja, miksi ja miten lähdin toteuttamaan tekniikoita. Miksi juuri tämä pelimoottori ja miten hyödyin tämän tarjoamista työkaluista sekä mitä kriteereitä tekniikoiden tulisi täyttää näitä läpi käytäessä?

4.1 Kehitysalustan valitseminen ja perustelut

Unreal Engine käyttää Microsoftin DirectX 3D -viitekehyspohjaista HLSL-ohjelmointikieltä (High Level Shading Language) varjostimia luodessaan. Visuaalista materiaalieditoria käytettäessä pelimoottori luo vaadittavan koodin taustalla sen sijaan, että käyttäjän tarvitsee osata kirjoittaa koodia perinteisin menetelmin. [11, s. 8, 12, 2:46.]

Käytettävyyden kannalta materiaalieditorin visuaalisuus ja laajuus toimivat pääsyinä Unreal Engine 5:n valitsemiseen kehitysalustaksi. Työkalun versioksi valitsin 5.4, millä varmistan, että pelimoottorin tarjoamat ominaisuudet vastaavat kirjoitushetkellä viimeisimpiä saatavissa olevia. Vaikka Blueprint-ohjelmointi on nopeampaa käyttää, C++-ohjelmointikielen käyttö takaa pääsyn kaikkiin pelimoottorin tarjoamiin ominaisuuksiin sekä hieman parempaa suorituskykyä erityisesti matemaattisia toimintoja suoritettaessa [19].

4.2 Interaktion toteutusmenetelmien vaatimusten määrittäminen

Veden tulee luoda illuusio siitä, että se kuuluu osaksi pelimaailmaa. Sen tulee mukautua niin pelaajan aiheuttamaan kuin pelimaailmassakin tapahtuviin muutoksiin materiaalin ja sen varjostimen mahdollistamin rajoittein.

Pelimaailmassa tapahtuvilla muutoksilla tarkoitetaan tuulen voimakkuuden vaihtumista, virtauksen kulkemista jo luodun staattisen pelialueen ympärillä sekä mahdollisten ulkoisten esineiden vaikutusta veden pintaan. Pelaajan aiheuttamalla muutoksella tarkoitetaan pelaajan liikkumista vesialueen sisällä ja tämän veden pintaan aiheuttamia muutoksia.

4.3 Käytettävien menetelmien määrittäminen

Staattisessa ympäristössä voidaan hyödyntää niin UV-koordinaatistoa hyväksikäyttäen objektin pinnalla liikutettavia tekstuurikarttoja kuin virtauskarttojakin (engl. flow map). Jotta nesteiden liikkumista ei tarvitse simuloida fysiikan lakien mukaisesti, voidaan esimerkiksi tuulen aiheuttaman turbulenssin vaikutuksesta johtuvaa illuusiota toteuttaa sekoittamalla eri suuntiin liikutettavia normaalikarttoja ja vahvistaa näiden Z-akselia osoittavan vektorin arvon voimakkuutta. Tämän lisäksi voidaan käyttää virtauskarttoja, jolloin voidaan normaalikarttapohjaisella vektori-informaatiolla määrittää haluttu tekstuurin kulkusuunta objektin pintaa myöten.

Dynaamisessa ympäristössä tutkin, kuinka erilaisia toimintatapoja voidaan hyödyntää, kuten etäisyyskentät (engl. distance fields), hahmontamiskohteet (engl. render target) sekä materiaalieditorin oma syvyyshäivytyssolmu (engl. depth fade node). Unreal Engine antaa mahdollisuuden päästä käyttämään pelimoottorin jo laskemaan dataan eri objektien välisistä etäisyyksistä pelimaailmassa [20]. Tätä tietoa hyödyntämällä voidaan materiaalieditorissa luoda dynaamisia, parametrisoituja muutoksia objektien pinnalla näiden ollessa lähellä toisiaan.

5 UV-koordinaatiston manipulointi

UV-koordinaatisto on keskeinen asia ymmärtää varjostimia luodessa, sillä sen hyödyntäminen mahdollistaa monenlaisia pinnalla tapahtuvia muutoksia. Koska koordinaatisto on ainut tapa päästä käsiksi haluttuihin kohtiin objektien pinnalla, voidaan tätä manipuloimalla ja tekstuureja hyväksikäyttämällä luoda erilaisia elementtejä ylläpitäen vuorovaikutuksen illuusiota.

5.1 Mikro- ja makroyksityiskohdat UV-koordinaatistoa liikuttamalla

Aloitin materiaalin luonnin prosessin keskittymällä niihin asioihin, mitä ensimmäisenä tuli mieleen vettä ajatellessa. Koska veden tulee mukautua niin staattisiin kuin dynaamisiinkin muutoksiin, niin vedestä yleisesti puhuttaessa tuli ensimmäisenä mieleeni sana dynaamisuus. Vaikka oikeassa elämässä vesi voi olla silmämääräisesti melkein paikallaan, haettaessa dynaamisuutta halusin tuoda veden pinnalle siinä tapahtuvaa liikettä.

Dynaamisuuden lisäksi halusin rikkoa pinnan liikkumattomuuden luomaa kuvaa yhteneväisyydestä tuoden mielenkiintoa tasaisuuteen. Koska vesi on luonteeltaan kaoottista, pitäisi olevilla keinoilla saada tämä tuotua myös materiaaliin. Nämä asiat määriteltyäni, päädyin tutkimaan mahdollisia ratkaisuja asian toteuttamiseksi.

Suurin kysymykseni oli siinä, miten voidaan määritellä pinnalla tapahtuvia muutoksia siten, että se kiinnittäisi oleellisesti pelaajan huomion. Tämä ei saisi kuitenkaan rikkoa illuusiota siitä, miltä veden tulisi näyttää. Veden pinta heijastaa hyvin valoa ja näin ollen epätasaisuudet näkyvät katsojalle peilimäisen luonteensa takia [21]. Päädyin käyttämään tätä hyväkseni, sillä ominaisuuksiensa vuoksi normaalikartat sopivat siihen erityisen hyvin.

Koska normaalikartat ovat tehty kertomaan grafiikkamoottorille siitä, miten valojen tulisi taittua kyseisellä pinnalla, en tarvinnut monimutkaista geometriaa sisältävää objektia, vaan yksinkertainen kahdesta kolmiosta rakentuva levymäinen objekti riitti monimutkaisten yksityiskohtien illuusion ylläpitämiseen. Kyseinen objekti toimii hyvin myös tekniikoiden havainnollistamiseen, koska ylimääräiset muodot eivät häiritse katsojan keskittymistä olennaiseen.

Aloitin luomalla uuden materiaalin ja lisäämällä sinne TextureSample-solmun, jolla päästään käsiksi tekstuurin dataan. Määritin moottorin mukana tulleen normaalikartan solmun tekstuuriksi

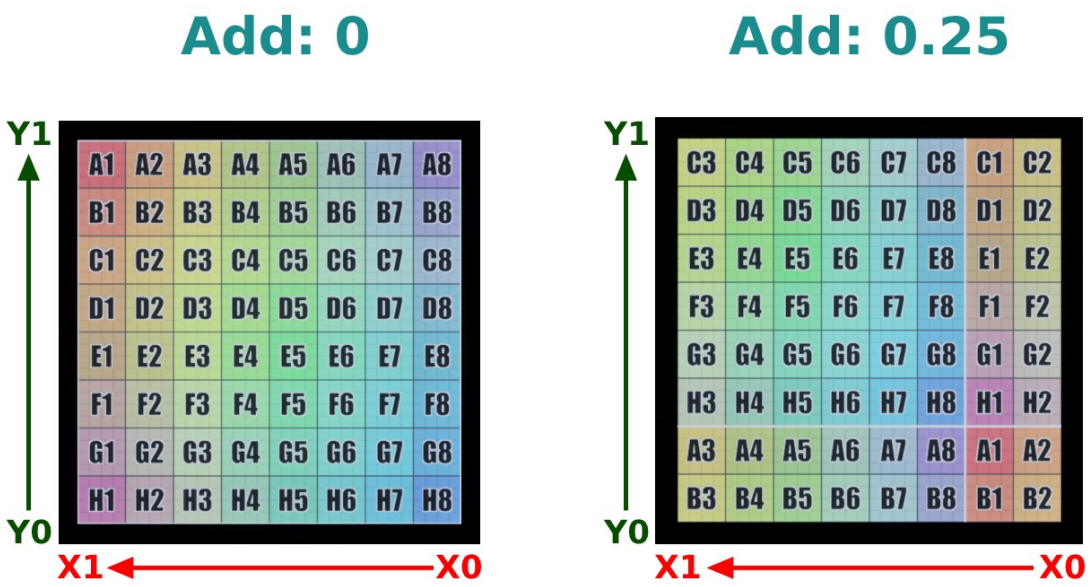
ja kiinnitin tämän materiaalin lopputuloksen määrittelemään pääsolmuun, jotta kyseinen materiaali osaa käsitellä sille annetun datan. Vaikka pinta ei enää ollutkaan tasainen, niin tässä kohtaa lopputuloksena on liikkumaton epätasaiselta näyttävä pinta. Jotta se saataisiin dynaamisemmaksi, päädyin käyttämään hyväkseni UV-koordinaatistoa ja sen liikuttamista.

Jotta tekstuurin saa liikkumaan objektin pinnalla, voidaan TextureSample-solmun UVs -kiinnityskohtaan lisätä pelimoottorin tarjoama tekstuurikoordinaatistoon käsiksi päästävä TextureCoordinate-solmu, johon arvoja lisäämällä saadaan hetkellistä liikettä aikaiseksi. Esimerkkinä tästä, että mikäli koordinaatistoon lisätään skalaariarvo 0.5, niin saadaan tekstuuria liikutettua puolet koordinaatiston etäisyyttä vastaavaa matka viistosti vasemmalle ja ylös. Tämä perustuu tekstuurikoordinaatiston ollessa välillä (0, 1), joten mikäli koordinaatistoon lisättäisiin skalaariarvo yksi, niin huomattavissa olevaa eroa ei näy käyttäjälle, koska näkyvä arvoväli muuttuu välille (1, 2).

Pelkkä arvon lisääminen tekstuurikoordinaatiston ei ole riittävä ratkaisu, sillä pelissä pinta näyttää hieman erilaiselta mutta silti staattiselta. Tämän takia tavoitteenani oli saada tekstuuri liikkumaan yhtäjaksoisesti. Jotta saadaan haluttu lopputulos, tulee lisätä pelimoottorin tarjoama Time-solmu, joka pitää kirjaa siitä, miten paljon aikaa on kulunut. Koska kyseinen solmun arvo muuttuu jatkuvasti kasvaen, se sopii täydellisesti elementtien animoimiseen. Päädyin luomaan yksittäisen skalaariarvon, mutta arvon olisi voinut myös parametrisoida, jotta sen muuttaminen olisi mahdollista materiaali-instanssin puolella. Mikäli annettu arvo kerrotaan aikaa vastaavalla solmulla, saadaan lopputulokseksi yhteneväisesti objektin pinnalla liikkuva tekstuuri.

Valitettavasti tekstuurin liikkuminen yhteen suuntaan ei ole erityisen käytännöllistä, joten seuraava tavoitteeni oli saada liikkuminen tapahtumaan haluttuun suuntaan. Tämän tavoitteen saavuttamiseksi pystyin hyödyntämään tekstuurikoordinaatiston tarjoamia X- ja Y-komponentteja. Mikäli koordinaatisto kerrotaan yhdellä skalaariluvulla, se kertoo molemmat komponentit samanaikaisesti, joten tarvitsin tavan käsitellä kumpaakin komponenttia erikseen.

Sen sijaan, että käytin yksittäistä skalaariarvoa ajan kertoimena, päädyin käyttämään kahta komponenttia sisältävää vektoria. Tämä antoi mahdollisuuden syöttää kertoimeen kaksi erillistä lukua, jotka vastaavat tekstuurikoordinaatiston X- ja Y-komponenttia ja näin mahdollistaen molempien akselien erillisen käsittelyn (kuva 6).

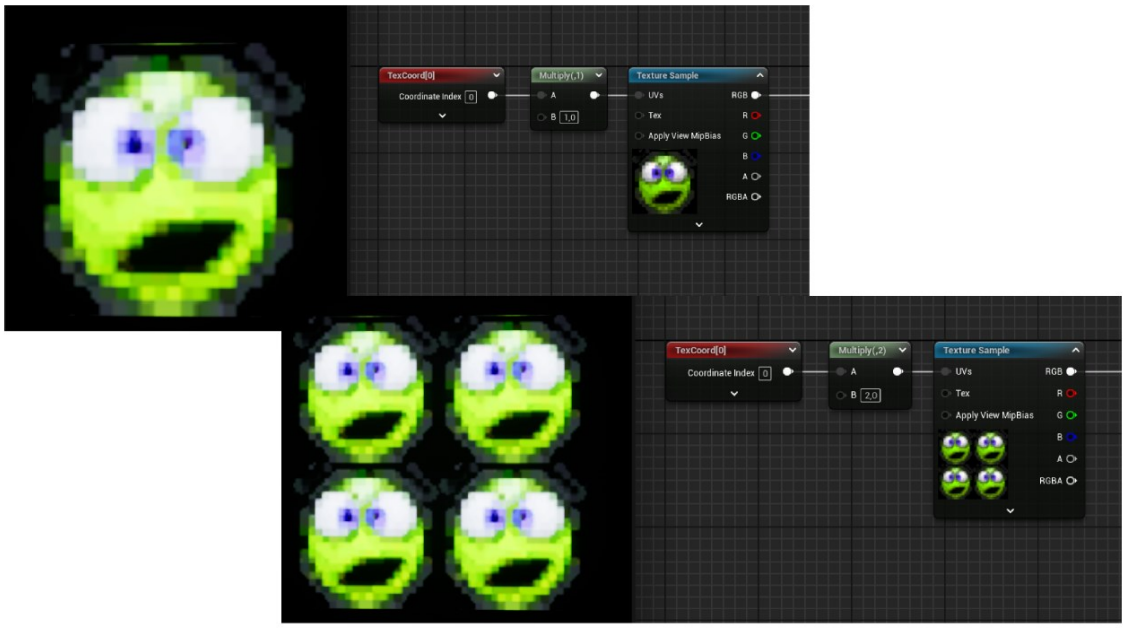


Kuva 6. Esimerkkikuva UV-koordinaatiston arvoihin lisäämisen vaikutuksesta tekstuurin liikkumissuuntaan.

Vaikka tekstuuri liikkuu objektin pinnalla haluttuun suuntaan, sen tarjoama toiminnallisuus ei riitä illuusion luomiseksi, sillä pelaaja huomaa nopeasti yksittäisen kuvan liikkuvan alustalla. Tämän ratkaisemiseksi lisäsin toisen tekstuurin, jota liikutin hieman eri arvoilla saaden ne liikkumaan eri suuntiin. Normaalikarttojen yhdistämiseen voidaan käyttää pelimoottorin tarjoamaa BlendAngleCorrectedNormals-solmua. Sen sijaan, että yhdistäisin normaalit kyseistä materiaalifunktiota käyttäen, päädyin kuitenkin käyttämään yksinkertaisempaa ratkaisua. Yhdistin normaalikartan R- ja G-kanavat keskenään käyttäen Append-solmua. Tämän jälkeen lisäsin molempien karttojen saadut X- ja Y-komponentit keskenään Add-solmua hyödyntäen. Materiaalin pääsolmu vaatii kolmen komponentin vektorin, joten päädyin lisäämään luvun yksi jälleen Append-solmua käyttäen. Lopputuloksena sain normaalikartta-arvon, joka sisältää kolme komponenttia (X, Y, Z) eikä lopputuloksessa näy suurempaa havaittavaa eroa. Yhdistämällä liikkuvat normaalikartat keskenään pystyin luomaan version, millä voidaan auttaa kiinnittämään pelaajan huomiota valojen luomiin kirkkauseroihin.

Yksi tapa parantaa yksityiskohtia samaa tekstuuria hyväksikäyttämällä, on tehdä niistä erikokoisia. Pelimoottorissa olevat tekstuurit ovet oletusasetuksina itseään toistavia. Tämä voidaan todentaa kertomalla tekstuurikoordinaatisto skalaariluvulla. Tekstuurikoordinaatistoon lisääminen mahdollistaa tekstuurin liikuttamisen, mutta koordinaatiston kertominen suurentaa tai pienentää kuvan kokoa koordinaatiston sisällä. Tämä voidaan todentaa kertomalla koordinaatisto luvulla

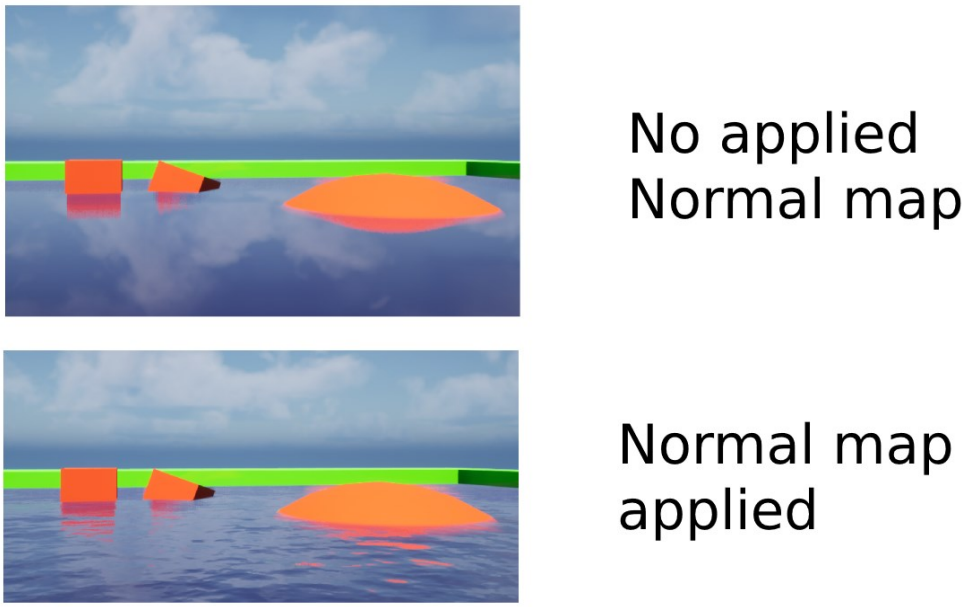
kaksi, mikä näyttää kyseisen tekstuurin materiaalin esikatselussa neljään kertaan samalla alueella. Tämä perustuu siihen, että kyseisellä luvulla kertominen muuttaa koordinaatiston väliltä (0, 1) välille (0, 2) (kuva 7).



Kuva 7. Esimerkki tekstuurin toistosta pelimoottorin mukana tulevalla tekstuurilla.

Yhdistämällä liikkumisen ja tekstuurien eroavat koot toistossa saadaan illuusioita lisättyä, koska pelaajan on vaikeampaa hahmottaa kyseessä olevan sama tekstuuri. Tätä voidaan myös jatkaa lisäämällä vielä kolmas normaalikartta, joka voidaan toistaa kertomalla koordinaatisto pienemmällä luvulla, mikä tekee tekstuurista isomman näköisen ruudulla. Tekstuurin liikuttamista ja toistoa muuttamalla voidaan luoda staattista dynaamisuutta pelin ajaksi ja sitä voidaan hyödyntää niin makro- kuin mikroyksityiskohtienkin luomisessa.

Tuodessani liikettä veden pinnalle, pystyin luomaan illuusiota siitä, miten maailmassa tapahtuu muutoksia pelaajan siihen vaikuttamatta. Tämä itsessään auttoi vahvistamaan mielikuvaa pelimaailman elossa olemisesta, eikä se ollut rajattu vain pelaajan omiin toimintoihin (kuva 8).



Kuva 8. Esimerkki normaalikartan vaikutuksesta veden pintaan.

Vaikka kyseisellä tavalla tehdyt yksityiskohdat toimivatkin hyvin luomaan pintojen syvyyden eroja valoja taittamalla, ei se kuitenkaan sovellu jokaiseen tilanteeseen. Yksi tämän tyylisen ratkaisun suurimmista heikkouksista on se, että sitä ei voida katsoa viistosta tason pintaa myötäillen, sillä silloin pelaajalle näkyy selvästi, kuinka kyseisiä muotoja ei ole oikeasti pinnalla. Tämä ei ole ongelma, mikäli voidaan taata pelaajan kykenemättömyys katsoa tasoa tietyistä kulmista. Muissa tilanteissa voidaan pitää hyvänä nyrkkisääntönä sitä, että normaalikarttoja käytettäessä on niitä hyvä hyödyntää tilanteissa, joissa haluttu yksityiskohdan lisäys ei vaikuta objektin siluettiin. Kyseinen metodi toimi tämän tyylisessä ratkaisussa hyvin, koska halusin saada pienempää yksityiskohtaa veden pinnalle vaikuttamatta sen siluettiin.

Tekstuurien siirtäminen ja animoiminen objektin pinnalla ennalta määrättyjen arvojen perusteella toimii hyvin, mikäli halutaan, että lopputulos pysyy muuttumattomana koko pelitilanteen ajan. Kyseinen metodi tuollaisenaan ei toimi, mikäli halutaan minkäänlaisia muutoksia kesken pelin. Siihen tarkoitukseen tarvitsin kokonaan toisenlaisen ratkaisun, minkä vuoksi päädyin käyttämään materiaaliparametrikokoelmaa.

Vaikka materiaalien muuttujiin pääseekin pelin aikana käsiksi dynaamisen materiaali-instanssin kautta, niin kyseinen parametrikokoelma antaa mahdollisuuden hoitaa pelimaailman laajuisia muutoksia keskitetysti. Tämä kuitenkin vaatii kyseisen logiikan kirjoittamista, jonka toteutin C++ -ohjelmointikieltä hyödyntäen.

5.2 Logiikan lisääminen C++ -ohjelmointikielellä

Dynaamisemmasta esimerkistä päädyin käsittelemään, kuinka tuulen vaikutusta voitaisiin käyttää hyödyksi veden interaktion kannalta. Tämäkään ei ole suoraan pelaajan toimista johtuva muutos veden pinnalla, mutta mahdollisuus luoda pelimaailmaan loogisesti kuuluvia sekä visuaalisesti näkyviä muutoksia kesken pelin auttaen entisestään vahvistamaan pelimaailman dynaamisuutta. Tähän esimerkkiin päädyin luomaan kaksi arvoa materiaaliparametrikokoelmaan, mistä skalaariarvo kuvaa tuulen voimakkuutta ja vektoriarvo tuulen suuntaa.

Koska tuuli voidaan käsittää globaalina asiana ja ainoat siinä muuttuvat asiat, jotka materiaalien tulisi ottaa huomioon ovat voimakkuus ja suunta, päädyin luomaan yhden alijärjestelmän (engl. Subsystem). Alijärjestelmä on Unreal Enginen automaattisesti luotu luokkainstanssi, johon päästään helposti käsiksi ja jonka elinkaari on määritelty ennalta pelimoottorin puolesta [22]. Päädyin valitsemaan maailman alijärjestelmästä (engl. World Subsystem) johdetun luokan, koska sen luonti tapahtuu tason auetessa ja loppuu tason suljettaessa. Näin ollen se on aina käytettävissä tason ollessa auki, eikä sen saatavuudesta tarvinnut huolehtia pelin ollessa käynnissä, mutta mikäli olisin tarvinnut arvon, joka säilyy tasojen välissä, niin kyseinen alijärjestelmä ei olisi siihen soveltunut. Määrittelin luotuun alijärjestelmään funktion, jolla voidaan päivittää kyseisen parametrikokoelman haluttuja arvoja, minkä johdostoa kaikki sitä käyttävät päivittyvät samanaikaisesti (kuva 9).

Fetch Material Parameter Collection

```
if (!WeatherMaterialCollectionInstance && IsValid(WeatherMaterialCollection))
{
    WeatherMaterialCollectionInstance = GetWorld()->GetParameterCollectionInstance(WeatherMaterialCollection);
}
```

Fetch Subsystem

```
WeatherSubsystem = GetWorld()->GetSubsystem<UWeatherSubsystem>();
if (WeatherSubsystem)
{
    WeatherSubsystem->OnWindUpdateRequestedDelegate.AddDynamic(this, &ASkySystem::OnWindUpdateRequested);
    WeatherSubsystem->OnTimeOfDayUpdateRequestedDelegate.AddDynamic(this, &ASkySystem::OnTimeOfDayUpdateRequested);
}
```

Function to call when requesting wind to update from Subsystem

```
/**
 * Update wind properties
 * @param InWindIntensity = Intensity of the wind
 * @param InWindDirection = Direction of the wind
 * @return void
 */
UFUNCTION(BlueprintCallable, Category = "DynamicEnvironmentModule")
void RequestUpdateWind(const float InWindIntensity, const FVector& InWindDirection) const;
```

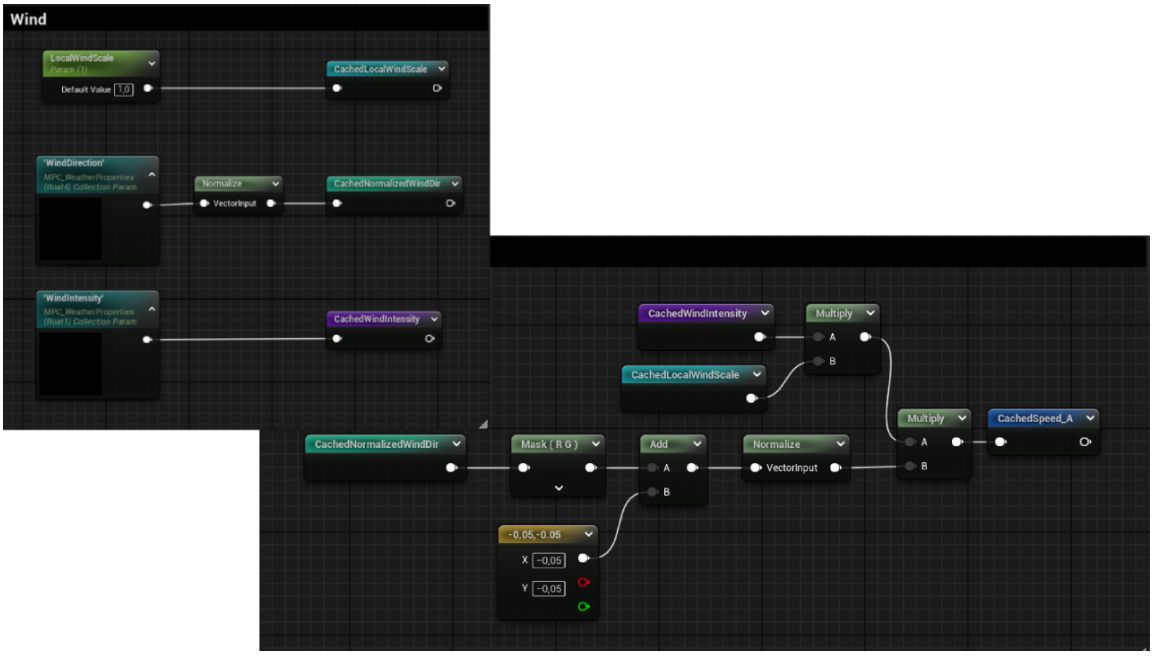
Callback function for when the RequestUpdateWind is called

```
void ASkySystem::OnWindUpdateRequested(const float InWindIntensity, const FVector& InWindDirection)
{
    if (IsValid(WeatherMaterialCollectionInstance))
    {
        WeatherMaterialCollectionInstance->SetScalarParameterValue(SkySystemParameterNames::WindIntensityParameterName, InWindIntensity);
        WeatherMaterialCollectionInstance->SetVectorParameterValue(SkySystemParameterNames::WindDirectionParameterName, InWindDirection);
    }
}
```

Kuva 9. Esimerkkikuva alijärjestelmän ja parametrikokoelman hakemisesta

Pienellä muutoksella aiempaan materiaaliin sain lisättyä parametrikokoelman muuttujat kuvaamaan tuulta. lisäämällä tuulen suuntaa kuvaavan vektorin X- ja Y-komponentit aiemmin määriteltyyn vektoriin ja sen uudelleen normalisoimalla sain määriteltyä uuden tuulen suuntaa vastaavan suuntavektorin, joka kunnioittaa myös alkuperäisiä arvoja. Tämän jälkeen pystyin kertomaan normalisoidun suuntavektorin kokoelmasta saadulla tuulen voimakkuudella vaikuttaen sen kokonaisnopeuteen.

Tämä yksinään ei riitä, sillä on mahdollista, että useammat asiat käyttävät samaa voimakkuusparametria yhtäaikaisesti, vaikka niiden ei tulisi reagoida pelimaailmassa asiaan samalla voimakkuudella. Tämän vuoksi määrittelin vielä yhden materiaalin oman parametrin, joka määrittää kokonaisvaltaisen voimakkuuden vaikutuksen paikallisesti (kuva 10). Se voidaan asettaa instanssikohtaisesti, joten mikäli halutaan, että kyseinen pinta reagoi vahvemmin tai heikommin, kuin parametrikokoelmasta on määritelty, voidaan kerrointa nostamalla tai laskemalla vaikuttaa lopputulokseen. Tässä tilanteessa kerroin tuulen voimakkuuden parametrin paikallisella muuttujalla, antaen enemmän valtaa lopputuloksen näkymiseen pelaajalle.



Kuva 10. Esimerkkikuva paikallisen LocalWindScale-parametrin käyttämisestä.

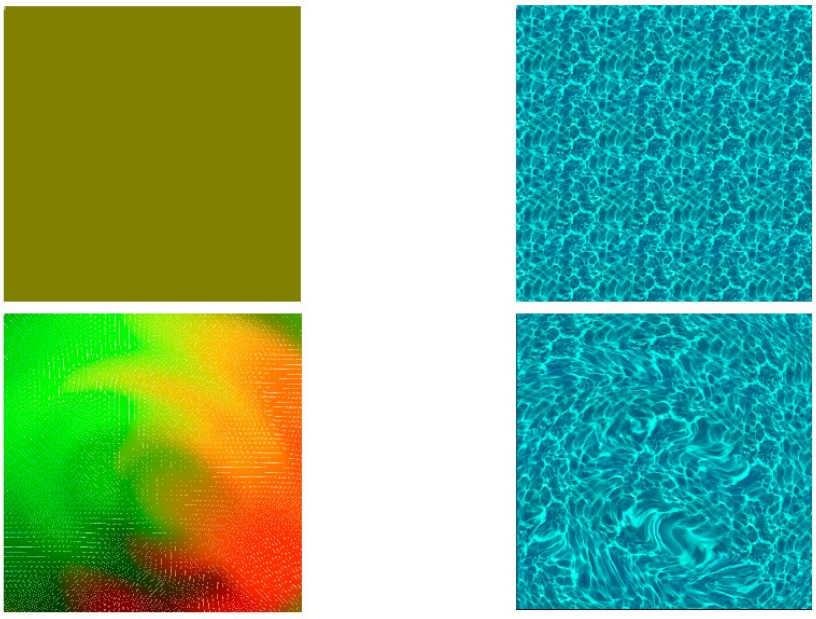
Jatkojalostamisen kannalta voitaisiin myös määritellä normaalin voimakkuutta riippuen tuulen voimakkuudesta. Pelimoottorissakin käytettyä tapaa, jossa kerrotaan normaalikartan Z-komponentti skalaariarvolla, voidaan hyödyntää illuusiota tuulen voimakkuuden vaikutuksesta. Tämä antaa mahdollisuuden siihen, että nopeuden ei tarvitse muuttua mutta normaalikartan vaikutukset eroavat paremmin pelaajalle luoden voimakkaamman lopputuloksen visuaalisuuden puolella.

5.3 Virtauskartat

Vaikka haluttuun suuntaan dynaamisesti liikutettavat tekstuurit luovat parempaa illuusiota veden interaktiosta maailman kanssa, ei sitä voida käyttää tilanteisiin, joissa veden pitäisi liikkua paikallisesti moneen suuntaan yhdellä alustalla. Mitä jos veden tulisi liikkua nopeammin keskellä kuin reunoilla tai esimerkiksi väistää maailmassa olevia objekteja? Tätä ongelmaa ratkaistakseni päädyin tutkimaan virtauskarttojen toteuttamista.

Pelimaailman liikkumattomat osat vesialueella sekä vesialueen ennalta määrätty muoto mahdollistavat hallitumman vettä kuvaavan objektin käsittelyn. Virtauskarttapohjaista varjostinta luodessa käytetään hyväksi objektin UV-koordinaatistoa, joka suunnan saamiseksi kerrotaan aiemmin tehdyllä vektoripohjaisella tekstuurikartalla. Tämän kartan luontiin voidaan käyttää apuna ulkoisia ohjelmistoja. Tässä kohtaa päädyin ilmaiseen vuonna 2012 päivitettyyn FlowMapPainter-ohjelmistoon sen yksinkertaisuuden vuoksi [23]. Vaihtoehtoisesti virtauskarttoja voidaan tehdä muun muassa myös maksullisella Adobe Substance Painter -ohjelmistolla tai ilmaisella Krita-kuvankäsittelyohjelmalla [24, 25].

FlowMapPainter-ohjelmistolla pystyin piirtämään suuntaa ja voimakkuutta osoittavat vektorit suoraan tekstuurin päälle mahdollistaen visuaalisen työympäristön, jossa näin tuloksen piirtyvän eteeni reaaliajassa. Tekohetkellä vektoreita voidaan myös tarkastella väreinä, jotka vastaavat tulevan normaalikartan X-, Y-, ja Z-komponentteja (kuva 11).



Kuva 11. FlowMapPainter-ohjelmistossa luodun virtauskartan vaikutus tekstuuriin.

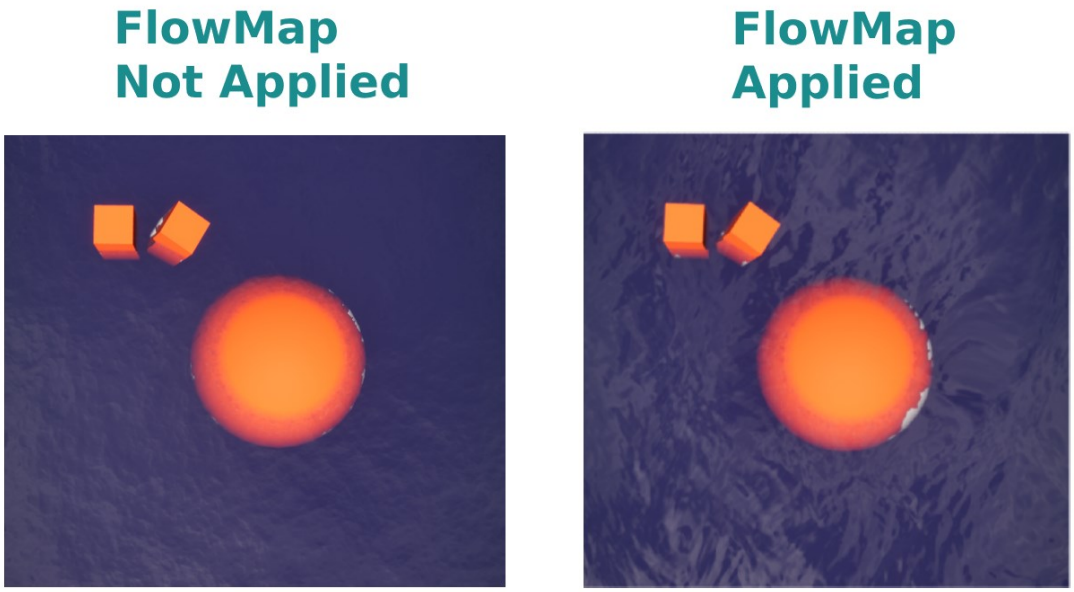
Kun olin saanut piirrettyä esimerkkitekstuurin, tuli tallennettavan tekstuurikartan punainen kanava kääntää päinvastaiseksi, jotta Unreal Engine näyttää virtauskartan oikein. Mikäli kanavaa ei ole käännetty, näyttää virtauskartta kulkevan väärään suuntaan. Tämä voidaan myös korjata materiaalin puolella kääntämällä normaalikartan punainen kanava käänteiseksi, minkä jälkeen saadaan sama haluttu lopputulos. Tekstuurin sisältävän vektoridatan takia on myös tärkeää muistaa muuttaa tuotava virtauskarttatekstuuri normaalikarttamuotoon.

UV-Koordinaatiston animoimiseen käytetään aikasolmua, joka mahdollistaa tekstuurien yhtäjaksoisen vierittämisen. Ongelmaksi muodostuu ajan kasvaminen liian suureksi, jolloin UV-koordinaatisto ei enää pysty tarkasti piirtämään haluttua lopputulosta vääristäen kuvaa liian paljon. Tämän ratkaisemiseksi voidaan rajoittaa ajan kulkeminen yhden sekunnin sisälle käyttämällä materiaalieditorin Frac-solmua, joka palauttaa desimaaliosan siihen syötetystä luvusta [26].

Solmun luonteen takia ajan päästyä arvoon 0,99, joka palautuu takaisin nollaksi, jolloin ajan arvo ei koskaan saavuta yhtä. Tämä tulos voidaan lisätä tekstuurikoordinaatistoon, minkä jälkeen saatu lopputulos voidaan liittää veden pintaa määrittelevän normaalikartan UV-koordinaatistoon. Tämän tuloksena tulee tekstuuri liikkumaan yhtäjaksoisesti arvon ollessa alle yhden, minkä jälkeen se palautuu alkuperäiseen pisteeseensä luoden liikkeeseen nykivän lopputuloksen. Sen korjaamiseksi tulee interpoloida kahden erijaksoisen UV-koordinaatiston välillä, mikä voidaan toteuttaa lisäämällä alkuperäisarvoon 0,5 ja lisäämällä kyseinen versio toiseen normaalikartan tekstuurikoordinaatistoon saaden sulavasti sekoittuva lopputulos. Tämä perustuu lukujen olevan aina yhtä kaukana toisistaan, millä saadaan luotua illuusio yhtäjaksoisesta liikkeestä.

Frac-solmun jälkeen saatu luku voidaan kertoa virtauskartan X- ja Y- komponenteilla, jolloin saadaan määriteltyä pinnan liikkuminen haluttuun suuntaan. Liikkumisen tapahtuminen UV-koordinaatistossa tekee kartan Z-komponentin tarpeettomaksi. Halusin pystyä vaikuttamaan liikkumisen voimaan, joten annoin vielä parametrisoidun skalaariarvon mahdollistaen liikkumisnopeuden hienosäätämisen materiaali-instanssin puolella.

Lopputuloksena saadaan normaalikartan vektoreita hyödyntämällä haluttuihin suuntiin liikkuva tekstuurikartta UV-koordinaatiston (0, 1) sisällä (kuva 12). Sen sijaan, että tekstuurit liikkuisivat ennalta määrätysti eri suuntiin mahdollistaa virtauskartta tekstuurin mukautumisen tekselintarkasti.



Kuva 12. Esimerkki virtauskartan mahdollistamasta ulkoasusta normaaliin verrattuna.

Kyseisen menetelmän käyttämistä yksinään ei sellaisenaan mahdollista dynaamisen ympäristön muuttamista, mutta auttaa rakentamaan kuvaa veden reagoimisesta ympäristön kanssa esilasketuilla tiedoilla. Rajoitteena voidaan myös pitää sitä, että virtauskartat eivät toimi hyvin yhteen normaalin tekstuurin liikuttamisen kanssa, sillä tuodut esilasketut muutokset sekä eri suuntiin liikkuvat tekstuurikartat eivät tule näyttämään luonnollisilta keskenään. Koska virtauskartat vaativat ennalta luodun tekstuurin tekemisen määritetylle alueelle, ei se yksin sovellu maailmaan, missä tulee vielä tapahtumaan muutoksia.

6 Syvyystiedon hyödyntäminen

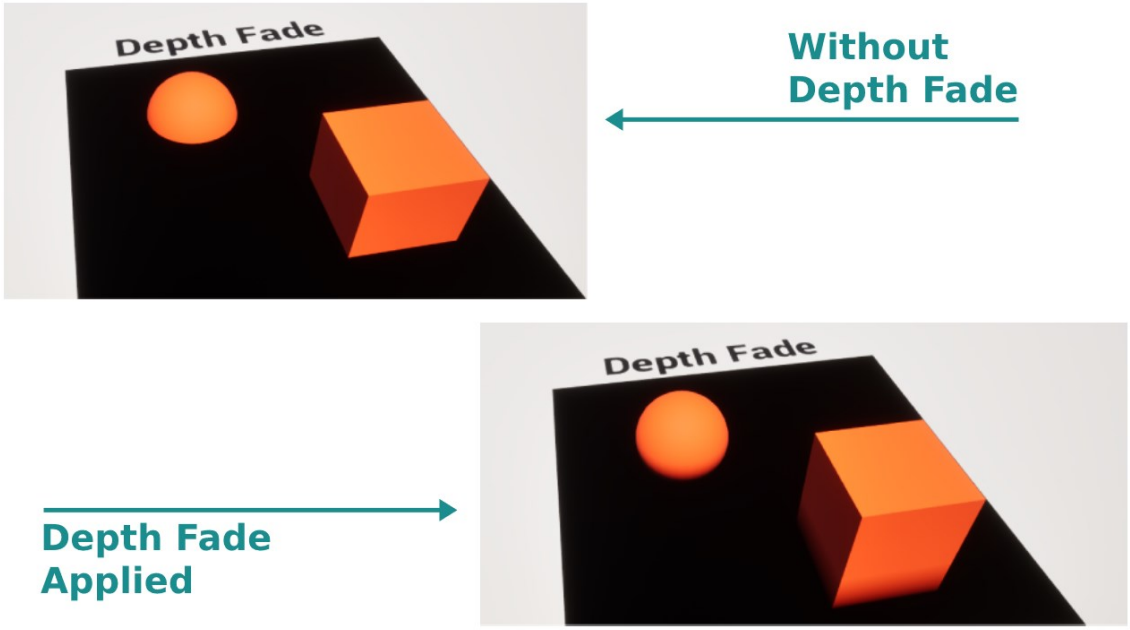
Joitain asioita ei pystytä määrittämään pelkästään objektin antamilla tiedoilla. Tämän takia päädyin käyttämään pelimoottorista saatua syvyystietoa niin syvyyshäivytyssolmun kuin etäisyyskenttienkin kautta. Molemmat tekniikat mahdollistivat monipuolisemman lopputuloksen luomista, mutta kumpikaan ei ollut käytettäessä ongelmaton.

6.1 Syvyyshäivytyssolmu

Syvyyshäivytyssolmu on pelimoottorissa sisäänrakennettuna ja mahdollistaa moottorin tuottaman syvyyskartan lukemisen materiaalin sisällä. Solmua käyttämällä voidaan havainnollistaa taustan häivyttämistä tietyllä etäisyydellä materiaalin omaavalta pinnalta. Materiaalin varjostusmallin tulee olla läpinäkyvyyden mahdollistavassa Translucent-tilassa, jotta kyseistä solmua voidaan käyttää, sillä muuten pelimoottori antaa käytöstä virheilmoituksen.

Se sisältää kaksi kiinnityskohtaa, joita käyttämällä voidaan määritellä haluttuja muuttujia efektin luomiseksi. Opacity-sisääntulo vaikuttaa siihen, kuinka läpinäkyvä materiaalin omaava objekti on ennen syvyyden laskemista, kun taas FadeDistance-sisääntulo antaa mahdollisuuden säätää kuinka pitkältä matkalta häivytysefekti tulisi laskea. Yksi etäisyyden yksikkö vastaa yhtä unreal-yksikköä, joka taas vastaa yhtä senttimetriä. [27.]

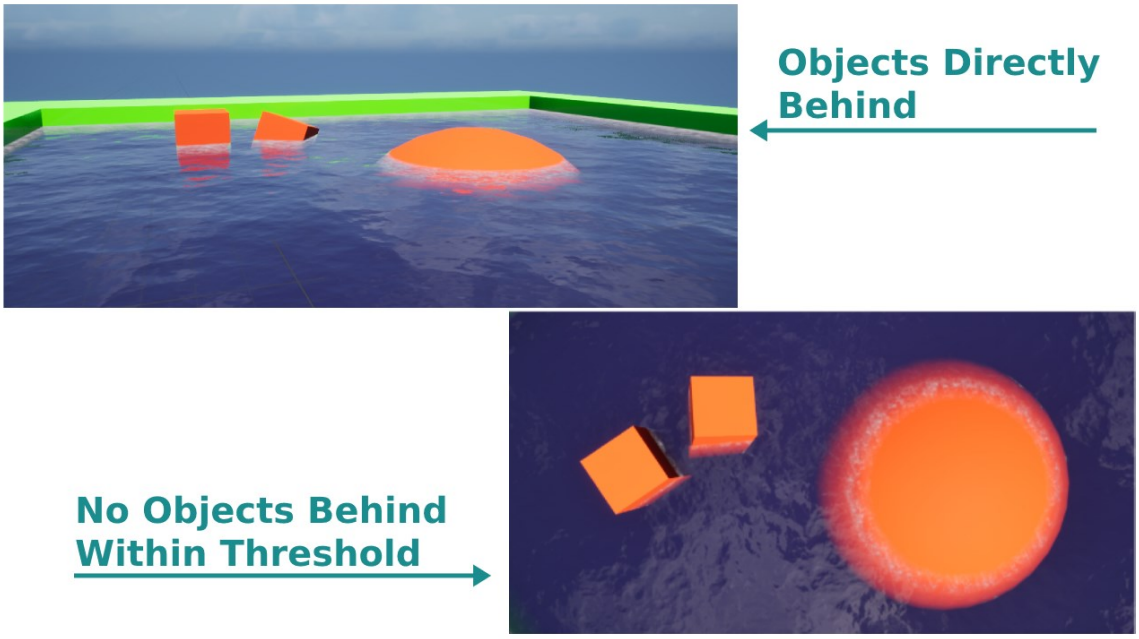
Koska kyseinen solmu antaa mahdollisuuden hyödyntää syvyyden laskemista, voidaan sitä käyttää muuhunkin kuin vain pinnalla tapahtuviin muutoksiin. Yksi mahdollinen käyttö on häivyttää objektin reunoja objektien leikkauspisteissä vähentäen kovan rajan muodostumista näiden välille. Tätä voidaan myös hyödyntää vesimateriaalin luonnissa, jolloin saadaan luotua illuusiota veden sulautumisesta paremmin ympäristöönsä ilman, että veden ja sen pinnalla olevan objektin välinen sauma näkyisi selkeänä pelaajalle (kuva 13).



Kuva 13. Esimerkkikuva syvyyshäivytyssolmun vaikutuksesta leikkauskohtien pehmeyteen.

Sen lisäksi, että solmua hyödyntämällä saadaan reunan välistä rajaa pehmennettyä, halusin myös luoda muunlaista näkyvää muutosta, jolla tuodaan lisää interaktiivisuutta pelimaailmaan. Tähän päädyin hyödyntämään aiemmin hyväksi toteamaani tapaa liikutella tekstuureja eri suuntiin, jotta sain muusta pinnalla tapahtuvasta muutoksesta johtuvaa liikettä.

Päädyin kertomaan saadun syvyyskartasta luodun maskin kolmisuuntaisella tekstuurinvierityksellä, jonka arvon rajasin nollan ja yhden välille. Lopulta liitin pääsolmun läpinäkyvyyttä kuvaavaan kiinnitykseen. Päädyin myös interpoloimaan saadun liikkuvan rajamaskin arvoilla kahta eri väriarvoa, joista toinen oli puhdas valkoinen ja toinen sinertävän värinen, joiden avulla pystyin määrittelemään reuna-alueen maskin valkoiseksi ja muun sinertäväksi, mikä loi helpommin erotettavissa olevan liikkeen ja enemmän veden vaahtoamisen kaltaista interaktiota ympäristön kanssa. Tässä kohtaa kävi myös ilmi syvyyshäivytyssolmun mielestäni suurin ongelma sitä käytettäessä: Sen luoma illuusio on riippuvainen katselukulmasta ja mikäli käyttäjän määrittelemän etäisyyden sisällä ei ole objektia takana, kyseistä vaikutusta ei ole selvästi nähtävissä (kuva 14).



Kuva 14. Lopputuloksen esimerkkikuva havainnollistamaan syvyyshäivytyssolmun rajoitettua katselukulmaa.

Syvyyshäivytyssolmu toimii parhaiten objektien leikkauskohtien pehmentämiseen sekä tietyin varauksin myös reunalla tapahtuvien muutoksien määrittämiseen. Pelimoottorin puolelta rajauksia määrittää materiaalin varjostusmallin oltavan Translucent-tilassa tai solmua ei voida hyödyntää. Katselukulman määrittelemät rajoitteet tekevät käytöstä hankalan, mikäli pelaajana käytössä on vapaasti liikutettava kamera, sillä tekniikan luoma illuusio katoaa nopeasti, jos reunan lähietäisyydellä ei ole pintaa, mistä saada etäisyystietoa. Solmu ei samasta syystä myöskään toimi sellaisenaan veden syvyyden laskemisessa.

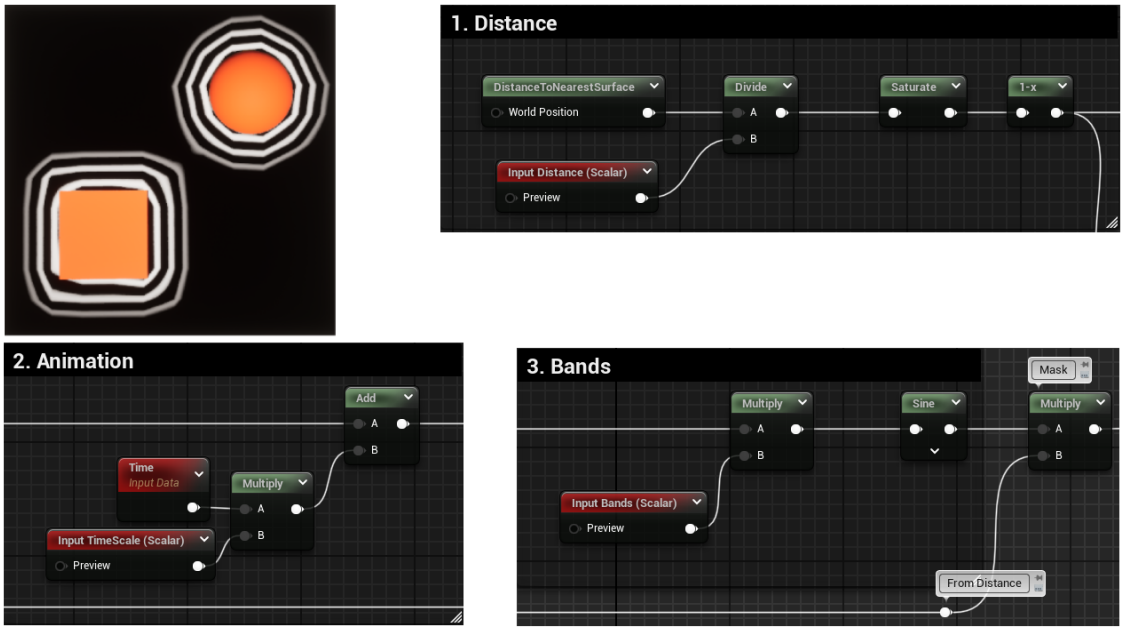
6.2 Etäisyykentät

Jotta etäisyyskenttää voidaan käyttää, pitää ensin varmistaa tämän olevan projektissa käytössä. Asetus löytyy Projektin asetuksiin menemällä ja aktivoimalla Generate Mesh Distance Field -valinta. Tämä mahdollistaa etäisyyskenttien laskemisen ja tallentamisen maailmalle tuotuihin objekteihin, mikäli näiden Affect Distance Field Lighting -valinta on käytössä. Mikäli valintaa ei ole määritetty kyseiselle objektille, ei tämä tule näkymään etäisyyskenttien seurannassa ja tätä ominaisuutta pystyin hyödyntämään interaktiota ajatellessa. Etäisyyskenttien laskeminen tapahtuu ennalta, mikä rajaa käyttöä, sillä niitä ei voida luoda pelin aikana. Mikäli halutaan käyttää materiaalin tarjoamaa pääsolmussa olevaa World Position Offset -sisääntuloa, jolla voidaan määrittää

objektin kärkipisteen paikkaa maailman tilassa, tulee ottaa huomioon, kuinka tämä saattaa aiheuttaa piirtämisen epätarkkuuksia, sillä etäisyyskentät eivät tiedä materiaalista johtuvia muodonmuutoksia. [20, 28.]

Pelimoottorin etäisyyskentän laskemaan dataan päästään käsiksi käyttämällä materiaalieditorin DistanceToNearestSurface-solmua, joka palauttaa tarkistetun kohdan pisteen maailman koordinaatistossa [27]. Tämän tiedon avulla voidaan luoda maski tietyn etäisyyden päähän jakamalla saatu tulos halutulla etäisyydellä. Tämä antaa mustavalkoisen maskin, jossa aivan objektin leikkauspisteen tuntumassa oleva osio ja siitä aiemmin määritetty etäisyys piirtyvät mustana ja ulkoiset arvot valkoisena. Jotta maskia voidaan käyttää objektin ympärillä olevan kohdan tarkistamiseen, maskin arvot tulee kääntää päinvastaiseksi OneMinus-solmua käyttämällä.

Mikäli tähän tulokseen lisätään aikasolmu, saadaan tulokseksi liikkuva maski halutun etäisyyden rajoissa. Saatu lopputulos tulee vielä rajata nollan ja yhden välille, jotta pysytään mahdollisten käsiteltävän data-arvon sisällä. Väreilevän lopputuloksen saamiseksi voidaan käyttää hyödyksi pelimoottorin materiaalieditorin tarjoamaa sine-solmua, jolla mahdollistetaan oletusarvoilla arvon jatkuvan siirtymisen (-1, 1) välillä [26]. Viimeiseksi kerrotaan saatu animoitu arvo aiemmin saadun käänteisen etäisyysmaskin kanssa, minkä seurauksena materiaalin omistavan objektin pinnalle piirtyy maskin rajaama alue kahden objektin leikkauspisteeseen (kuva 15).



Kuva 15. Parametrisoitu etäisyyskenttää hyödyntävä materiaalin osa ja sen objektin ympärille luoma maski pelimaailmassa.

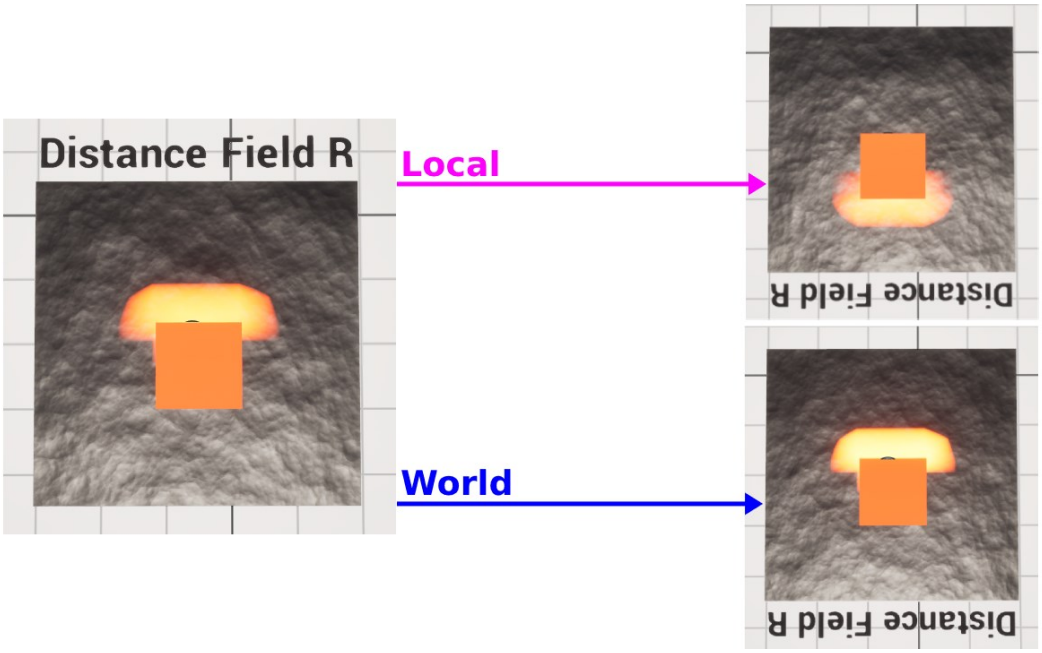
Tämä antaa mahdollisuuden luoda esimerkiksi pinnan aaltoilua, mikäli objekti pysyy paikallaan. Jos objekti liikkuu veden pinnalla, pysyy luotu maski aina yhtä kaukana haitaten dynaamisen interaktion illuusiota. DistanceToNearestSurface-solmu toimiikin parhaiten paikallaan olevien objektien kanssa, esimerkiksi rantaviivan tai vedessä paikallaan olevan asian – kuten esimerkiksi kiven – ympärillä tapahtuvaa pinnan aaltoilun muutosta. Tämä metodi ei anna yhtä paljon mahdollisuuksia virtauksen suunnan määrittämiseen verrattuna virtauskarttoihin, mutta se tapahtuu ilman, että tarvitsee luoda uutta tekstuurikarttaa vastaamaan haluttua muutosta.

DistanceToNearestSurface-solmu mahdollistaa vain ympärillä tapahtuvan asian huomioimista eikä solmusta itsestään saada laskettua kuin sijainti ja etäisyys leikkauspisteestä. Mikäli halutaan sijainnin ja etäisyyden lisäksi myös suuntatieto, voidaan aikaisemman lisäksi hyödyntää DistanceFieldGradient-solmua, joka normalisoituna antaa ympäröivän sijainnin lisäksi myös halutun suunnan [27].

Koska normalisoidun DistanceFieldGradient-solmun avulla voidaan saada suuntavektori, pystyin tämän avulla määrittelemään maskin objektin ympärille ja saman aikaisesti saamaan halutun suunnan. Sain ComponentMask-solmua hyväksikäyttämällä määriteltyä halutun suunnan, koska kyseisellä solmulla voidaan eristää halutun komponentin sisältämä arvo. Valitsin R-kanavan, joka vastaa X-komponenttia ja näin ollen vastaa pelimoottorin maailmassa eteenpäin osoittavaa vektoria. Tässä kohtaa maski osoittaa keilamaisesti eteenpäin.

Tästä saatua maskia voidaan rajata hyödyntämällä DistanceToNearestSurface-solmua, jolla sain kyseisen solmun näkymään vain 50 yksikön päähän objektista. Saadun maskin rajausta pystytään myös rajoittamaan vähentämällä normalisoidusta DistanceFieldGradient-solmusta saatua arvoa. Tämä vähennettävä arvo toimii siirtoarvona lopputulokseen pienentämällä tai suurentamalla saadun keilamaisen maskin leveyttä.

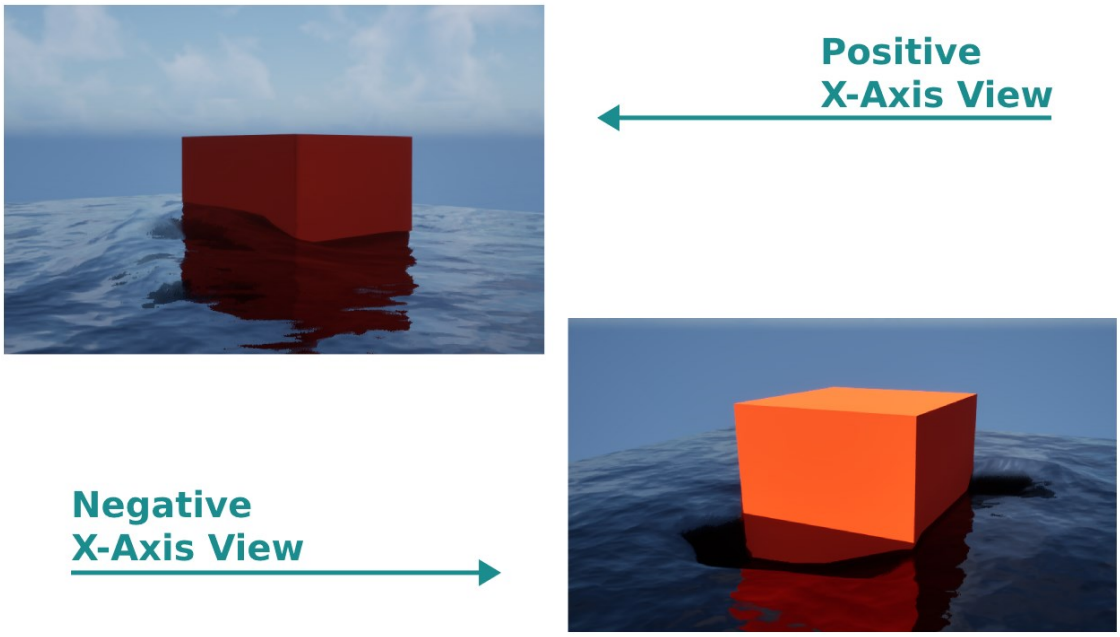
Mikäli ei haluta, että rajoitettu komponentti on maailman tilassa, voidaan DistanceToGradient-solmu vetää TransformVector-solmun läpi, jolloin voidaan solmun asetukset asettamalla määritellä vektorin laskenta maailman tilasta paikalliseen tilaan. Näin saadaan arvo, joka muuttuu materiaalin omaavan objektin eteenpäin osoittavan vektorin suuntaisesti. Mikäli objektia käännetään, kääntyy maski tämän mukana (kuva 16).



Kuva 16. DistanceToGradient-solmun maskin sekä maailman ja paikallisen tilan havainnollistaminen.

DistanceToGradient-solmun perustuessa etäisyyskenttiin tulee sen kanssa samanlaisia ongelmia kuin DistanceToNearestSurface-solmua käytettäessä, jolloin maski pysyy näennäisesti samanlaisena objektin liikkuessa. Tämän solmun vahvuus on kuitenkin siitä saadussa suuntavektorissa, jota voidaan käyttää hyväksi kärkipisteiden paikkojen liikuttamiseen maailmalla. Vaikka tämä rajoittaakin käyttöä, sillä se vaatii enemmän geometriaa materiaalin omistavalta objektilta, voidaan maskin avulla nostaa ja laskea vesitason pintaa kärkivarjostinta ja maskista saatavia arvoja hyväksikäyttämällä. Saadut arvot pääsolmun World Position Offset -sisääntuloon lisäämällä voidaan määrittää objektin kärkipisteen paikkaa maailman tilassa luoden dynaamisempaa muutosta.

Tämän rajoituksena on kuitenkin aiemmin komponentin eristämällä saatu suunta, jota tulee kunnioittaa materiaalin luonnin aikana. Esimerkkinä tästä, jos aiemmasta esimerkistä mallia ottaen eristetään maailman suunnan X-komponentti, niin maskia hyödyntämällä tulee ottaa huomioon se, miten vain tuolla akselilla tulee tapahtumaan muutoksia. Pystyin hyödyntämään tätä tietoa laittamalla aiemmasta esimerkistä DistanceToGradient- ja DistanceToNearestSurface-solmujen yhteen kerrotun arvon World Position Offset -sisääntuloon, joka nostattaa pintaa X-akselin positiiviselta puolelta ja laskee sitä X-akselin negatiiviselta puolelta tietyn etäisyyden päästä objektista (kuva 17). Tätä tekniikkaa hyödyntäen pystyin luomaan dynaamisemman illuusion virtauskarttoihin verrattuna, missä vesi näyttää fyysisesti väistävän tiellä olevaa objektia.



Kuva 17. Kärjistetty esimerkki kuvaamaan kärkipisteiden liikuttamista DistanceToGradient-solmusta saatua maskia hyödyntäen.

Etäisyyskarttoja voidaan käyttää erinomaisesti datan saamiseksi objektien pintojen leikkauskohdista hyödyntämällä pelimoottorin tarjoamia etäisyystietoja. Vaikka etäisyyskarttojen luominen olisikin asetettuna projektiasetuksista, lopputulos on riippuvainen niin objektin tarjoamasta resoluutiosta kuin katseluetäisyydestäkin. Molempia voidaan nostaa niin objektin omista asetuksista (resoluutio) kuin katseluetäisyyden vaikuttamaa resoluution laskemista projektiasetuksistakin. Molemmat vaativat enemmän resursseja koneelta ja mikäli etäisyyskenttiä ei muuten tarvita sen hetkisessä peliprojektissa, saatetaan ne laittaa kokonaan pois päältä, mikä tekee tästä materiaalin ominaisuudesta täysin turhan. Mikäli mahdollisuus löytyy etäisyyskarttojen käyttöön, ovat ne hyviä työkaluja interaktiivisen veden illuusion luomiseen.

7 Hahmontamiskohteet

Aloitin luomalla tarvittavat asiat pelimaailmaan, jotta pystyin testaamaan hahmontamiskohteiden toimivuutta. Lisäsin levyn maailmaan, minkä päälle sijoitin pelimoottorin tarjoaman VirtualTextureVolume-objektin niin, että sen reunat peittivät koko levyn. Tämä mahdollisti muutoksen tapahtumisen todentamisen halutulla alueella. Kyseinen virtuaalinen tila tarvitsi viittauksen virtuaaliseen tekstuurin, joten loin tyhjän version pelimoottorin sisällä. Tämä tapahtui tyhjästä tilasta hiiren oikealla painikkeella painamalla ja hakupalkkiin kirjoittamalla ”Runtime Virtual Texture” ja ainoan tuloksen valitsemalla. Asetuksista en muuttanut mitään ja laitoin tämän vaadittuun viittaukseen.

Saatuani virtuaalisen tekstuurin laitettua, tarvitsin seuraavaksi tavan muuttaa kyseistä tekstuuria pelin aikana, mihin soveltuu hahmontamiskohteen käyttö. Tähän käytin pohjana löytämääni artikkelia siitä, kuinka materiaalin lopputulos voidaan kuvantaa tekstuurin pinnalle. Hahmontamiskohde tarvitsee materiaalin, joka piirretään objektin pinnalle ja josta saatu lopputulos tallennetaan luodun hahmontamiskohteen pinnalle. [29.]

Luotuani hahmontamiskohteen ja tilan virtuaaliselle tekstuurille, piti nämä saada vielä yhdistettyä. Tähän tarvitsin kaksi ylimääräistä materiaalia, joista toinen antaa tietoa virtuaalitekstuurille ja toinen lukee kyseistä tietoa. Näiden käyttötarkoitusta selventääkseni voidaan niitä ajatella siveltimenä ja kanvaasina.

Siveltimen materiaalina käytin pelimoottorin tarjoamaa RadialGradientExpontential-funktiota, joka antaa ympyrän mallisen lopputuloksen. Sen sijaan, että se kiinnitettäisiin materiaalin pääsolmuun, tulee se kiinnittää RuntimeVirtualTextureOutput-solmuun, joka mahdollistaa virtuaaliselle tekstuurille annettavan tiedon kulkemisen.

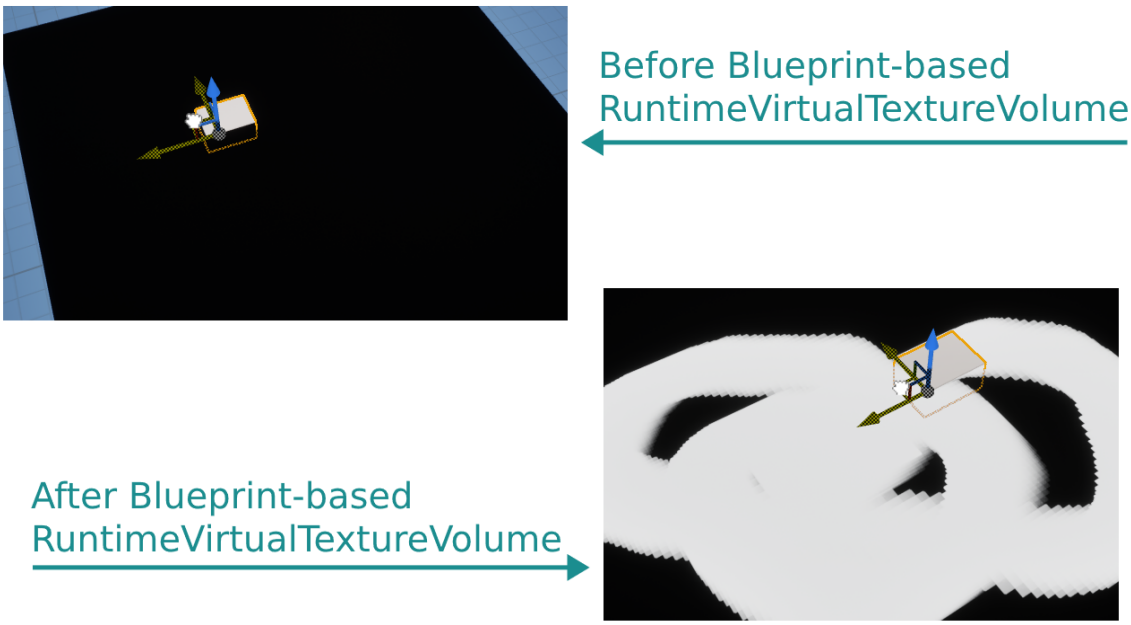
Kanvaasin puolella pystyin käyttämään RuntimeVirtualTextureSample-solmua, joka ottaa sisäänsä viittauksen virtuaaliseen tekstuuriin, jonka määritin aiemmin tämän osion toisessa kappaleessa mainitussa luomisvaiheessa. Tässä piti ottaa huomioon se, että materiaali tuli asettaa Translucent-tilaan, ja liittää RuntimeVirtualTextureSample-solmun BaseColor-kiinnityksestä pääsolmun Opacity-kiinnitykseen. Tämän lisäksi laitoin valkoista vastaavan skalaariarvon yksi pääsolmun EmissiveColor-kiinnitykseen. Tämä mahdollisti tulevan piirtämisen tapahtuvan ainoastaan siveltimen määrittelemällä alueella.

Näiden jälkeen piti saada määritettyä, mikä on maailmassa hahmontamiskohteeseen piirtävä asia sekä millä saan tiedon halutun tason pinnalle. Aiemmasta artikkelista lainaten tein vielä yhden materiaalin, johon lisäsin tekstuurin lukuun tarkoitetun TextureSample-solmun. Pystyin asettamaan kyseiseen solmuun luodun hahmontamiskohteen, jonka RGB-kiinnityksen yhdistin pääsolmun BaseColor-kiinnitykseen havainnollistaakseni piirrettävää osiota itselleni.

Koska hahmontamiskohteeseen piirrettäessä tulee kutsua vain yhtä pelimoottorin tarjoamaa DrawMaterialToRenderTarget-funktiota, päädyin yksinkertaisuuden vuoksi tekemään tämän Blueprinttien puolella C++-ohjelmointikielen sijaan. Loin Actor-luokasta johdetun Blueprintin, jotta sain sen laitettua pelimaailmaan ja päästäkseni käsiksi logiikan kirjoittamiseen. Liitin DrawMaterialToRenderTarget-funktion Tick-solmuun, jota normaalisti kutsutaan ruudun jokaisella päivityskerralla [30]. Kyseinen funktio tarvitsi kaksi viittausta – hahmontamiskohteen sekä materiaalin, joka lukee virtuaaliselle tekstuurille annettua tietoa, tässä tilanteessa tarkoittaen aiemmin määritellyn kanvaasin. Koska en ollut vielä tässä kohtaa määrittänyt sitä, mitä kautta data annetaan virtuaalitekstuurille, päädyin määrittämään tekniikan esittelymielessä suorakulmion mallisen objektin luodun Blueprintin sisälle, mille määritin aiemmin luomani siveltimenä toimivan materiaalin.

Tämän avulla pitäisi lopputulokseksi saada luotua objektia liikuttamalla mustavalkoista maskia määritellyn levyn pinnalle. Näin ei kuitenkaan käynyt, sillä pelimoottorin tarjoama RuntimeVirtualTextureVolume-objekti ei tue sellaisenaan pelin aikana tapahtuvia muutoksia. Tämä voidaan kuitenkin kiertää ja todentaa luomalla oma Blueprint-luokka, joka on johdettu RuntimeVirtualTextureVolume-luokasta, mikä antaa mahdollisuuden pelin aikaisten muutoksien käsittelemiseen.

Tässä kohtaa törmäsin pelimoottorissa piilevään ongelmaan, jossa pelimoottori kaatui jokaisella yrityksellä avata luotua Blueprint-luokkaa. Kyseisen ongelman pystyi kiertämään olematta avaamatta kyseistä luokkaa ja asetettu virtuaalitekstuuriviittaus voidaan asettaa suoraan pelimaailmassa sivupalkin kautta. Näiden asioiden jälkeen objektia pelissä liikuttaessani sain piirrettyä asetetun levyn pintaan valkoisen maskin objektin liikkumilla alueilla (kuva 18).



Kuva 18. Esimerkkikuva havainnollistamaan Blueprint-pohjaisen RuntimeVirtualTextureVolume-luokan luomisesta aiheutuvaa lopputuloksen muutosta.

Mustavalkoisen maskin piirtyessä huomasin yhden tämän tekniikan isoimmista heikkouksista, sillä saa piirrettyä suoraan objektin pinnalle, mutta luodun maskin poistaminen tapahtuu joko piirtämällä vanhan päälle eri väriarvolla tai suoraan luotu hahmontamiskohde tyhjentämällä ja aloittamalla alusta. Kumpikaan ei mahdollista luonnollisen näköisen lopputuloksen luomista järkevällä aikavälillä.

Tässä kohtaa ymmärsin, että virtuaalitekstuurin vahvuudet ovat muualla kuin pelin aikana tapahtuvassa reaaliaikaisen muutoksen illuusion luomisessa vesimateriaalista puhuttaessa. Näin ollen päätin, ettei ollut järkevää jatkaa veden reaaliaikaista, dynaamista interaktion luomista tällä menetelmällä. Turhaa työtä en tehnyt, sillä tämä antaa hyvän työkalun isojen vesialueiden alueiden määrittelyyn mustavalkoisen tekstuurin avulla. Kyseisen alueen maskin luomiseen ei tarvitse käyttää esiteltyä objektin liikuttamista maailmassa, vaan voidaan käyttää esimerkiksi hiiren paikkaa ruudulla, millä voidaan luoda luonnollisempi tapa piirtää suoraan pelimaailmaan.

Tämän tyylinen ratkaisu toimii kuitenkin paremmin esimerkiksi pinnan nostamiseen ja laskemiseen pelaajan kohdalla luoden interaktiota muissa tilanteissa kuten esimerkiksi lumen tai mudan painuminen objektien alla, mutta veden kanssa se ei näytä luonnolliselta. Vaikka hahmontamiskohteet täyttävätkin ennalta määräämäni kriteerit, niin se minkälaista interaktiota tällä menetelmällä voisi luoda, on tehtävissä helpommin toisilla tavoilla.

8 Päätäntö

Tutkiessani mahdollisia tapoja törmäsin nopeasti ongelmaan, jossa englanninkieliset termit eivät kääntyneet helposti suomen kielelle. Olisin voinut jättää termit englanniksi, mutta mielestäni oli tärkeää, että pyrin luomaan opinnäytetyön, joka olisi sisäistettävissä myös alan ulkopuolisten henkilöiden sitä lukiessa. Päädyin näin ollen käyttämään aikaa sanaston hakemiseen ja lopullisien sanavalintojen sisäistämiseen niin sanakirjoista kuin yleisiä kääntäjiäkin hyödyksi käyttäen.

Kokonaisuudessaan keskeisimmiksi osa-alueiksi muodostuivat tekstuurikoordinaatiston ymmärtämisen merkitys sekä pelimoottorin sisäänrakennetut tiedot pelimaailmasta syvyystiedon muodossa. Yhdessä ne mahdollistavat monenlaisten tekniikoiden luomisen, millä voidaan rakentaa interaktion illuusiota pelaajan ympärille niin staattisessa kuin dynaamisemmassakin ympäristössä.

Itselleni suurimman ongelman tuotti hahmontamiskohteiden käyttäminen ja määrittäminen. Vaikka näiden perusteista ei ollut vaikea löytää tietoa pelimoottorin dokumentaation ja keskustelupalstojen tarjoamien tietojen ansiosta, lopputulos ei kuitenkaan vastannut alkuperäistä olettamustani. Sain kuitenkin oppimieni tietojen valossa korjattua ennakko-olettamukseni toisenlaiseksi, minkä pyrin tuomaan esille kyseistä aihetta käsittelevässä kappaleessa.

Kokonaisuudessaan veden interaktion luominen on laajaosainen prosessi eikä opinnäytetyön pituuden ja laskennallisesti käytettävän ajan puitteissa ollut mahdollista käsitellä kaikkia tekniikoita. Päädyin kuitenkin valitsemaan mielestäni tärkeimpiä asioita kokonaisuuden ymmärtämisen kannalta pyrkien tuomaan tekniikoita aloittaen yksinkertaisemmasta ja rakentaen kohti monimutkaisempaa lopputulosta. Käsitellyt tekniikat ovat yleispäteviä muissakin tilanteissa, joissa materiaaleja hyväksikäyttämällä halutaan muokata, miltä objekti näyttää pelimaailmassa.

Objektiin liitettävä materiaali ei myöskään ole ainut tapa luoda interaktiota, sillä tämän päälle voidaan rakentaa muitakin tekniikoita, esimerkiksi erikoistehosteiden ja jälkikäsittelyjen lisäämiseksi. Näistä esimerkkinä erikoistehosteiden partikkelien hyödyntäminen vaahtopäiden luomisessa veden pinnalla sekä jälkikäsittelyn (engl. post process) määrittelemään kuvan lopputulosta veden alla oltaessa. On kuitenkin hyvä pitää mielessä, että ongelmien ratkaisemiseksi ei ole vain yhtä oikeaa ratkaisua, mutta tämän opinnäytetyön kautta halusin tuoda mahdollisia tapoja vuorovaikutuksen luomisesta muodostuvien ongelmien ratkaisemiseksi.

Kokonaisuudessaan tämän aihepiirin valitseminen ja toteuttaminen antoi hyvän kuvan siitä, miten erilaisia tapoja hyödyntämällä voidaan ratkaista hyvinkin erilaisia ongelmia. Vaikka kaikki tekniikat eivät lopulta soveltuneetkaan suoraan käsittelemäni aihepiirin lopputuloksen luomiseen, jatkoa ajatellen onnistuin saamaan uusia työkaluja auttamaan tulevien visuaalisen puolen ongelmatilanteiden ratkaisemista.

Lähteet

1. About Epic Games. Epic Games. [Internet]. [Viitattu 8.2.2024]. Saatavilla: https://www.epicgames.com/site/en-US/about

2. Fortnite Press Kit. IGDB. [Internet]. [Viitattu 18.11.2024]. Saatavilla: https://www.igdb.com/games/fortnite/presskit

3. Scale and measurement inside Unreal Engine. TechArtHub [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://www.techarthub.com/scale-and-measurement-inside-unreal-engine/

4. Material Editor User Guide. Epic Games. [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/unreal-engine-material-editor-user-guide/

5. Blueprints Visual Scripting. Epic Games [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://docs.unrealengine.com/5.2/en-US/blueprints-visual-scripting-in-unreal-engine/

6. Materials. Epic Games. [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/unreal-engine-materials/

7. Instanced Materials. Epic Games. [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/instanced-materials-in-unreal-engine/

8. Essential Material Concepts. Epic Games. [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/essential-unreal-engine-material-concepts/

9. https://dev.epicgames.com/documentation/en-us/unreal-engine/using-material-parameter-collections-in-unreal-engine

10. A Practical Guide to Unreal Engine's Coordinate System. TechArtHub [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://www.techarthub.com/a-practical-guide-to-unreal-engines-coordinate-system/

11. Gordon V, Clevenger L. 2. päivitetty painos. Computer Graphics Programming in OpenGL with C++. Mercury Learning & Information; 2021.

12. Cloward B. What Is A Shader? UE4 Materials 101 – Episode 1 [Video]. Ben Cloward. Youtube. 21.11.2019 [Viitattu 28.10.2023]. Saatavilla: https://www.youtube.com/watch?v=uQG0SWv5Ibw

13. Cloward B. The Graphics Pipeline – Shader Graph Basics – Episode 2 [Video]. Ben Cloward. Youtube. 17.6.2021 [Viitattu 28.10.2023]. Saatavilla: https://www.youtube.com/watch?v=ZEXVQgbWxQY

14. Textures. Epic Games. [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/textures-in-unreal-engine/

15. Cloward B. Texture Coordinates – Shader Graph Basics – Episode 8 [Video]. Ben Cloward. Youtube. 29.7.2021 [Viitattu 28.10.2023]. Saatavilla: https://www.youtube.com/watch?app=desktop&v=reAlVCXBtjs

16. Williams M. WebGL HotShot. Packt Publishing, Limited; 2014.

17. Cloward B. Advanced Channel Packing – Shader Graph Basics – Episode 23 [Video]. Ben Cloward. Youtube. 18.11.2021 [Viitattu 28.10.2023]. Saatavilla: https://www.youtube.com/watch?v=m5bP-xc6Sgs&t

18. Runtime Virtual Texturing. Epic Games [Internet]. [Viitattu 2.11.2024]. Saatavilla: https://dev.epicgames.com/documentation/en-us/unreal-engine/runtime-virtual-texturing-in-unreal-engine

19. Balancing Blueprint and C++. Epic Games [Internet]. Viitattu [31.8.2024]. Saatavilla: https://docs.unrealengine.com/4.27/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/

20. Mesh Distance Fields. Epic Games [Internet]. [Viitattu 31.8.2024]. Saatavilla: https://docs.unrealengine.com/5.2/en-US/mesh-distance-fields-in-unreal-engine/

21. Reflection of light. Science Learning Hub [Internet]. Viitattu [23.10.2024]. Saatavilla: https://www.sciencelearn.org.nz/resources/48-reflection-of-light

22. Programming Subsystems. Epic Games [Internet]. [Viitattu 26.10.2024]. Saatavilla: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-subsystems-in-unreal-engine

23. FlowMap Painter. TECKARTIST [Internet]. [Viitattu 18.11.2024]. Saatavilla: https://teckartist.com/?page_id=107

24. Flow Map Painting |Substance 3D Painter. Adobe [Internet]. [Viitattu 26.10.2024]. Saatavilla: https://helpx.adobe.com/substance-3d-painter/painting/advanced-channel-painting/flow-map-painting.html

25. Tangent Normal Brush Engine. Krita [Internet]. [Viitattu 23.10.2024]. Saatavilla: https://docs.krita.org/en/reference_manual/brushes/brush_engines/tangen_normal_brush_engine.html

26. Math Expressions. Epic Games [Internet]. [Viitattu 27.10.2024]. Saatavilla: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/ExpressionReference/Math/

27. Utility Expressions. Epic Games [Internet]. [Viitattu 27.10.2024]. Saatavilla: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/ExpressionReference/Utility/

28. 1.10 - World Position Offset. Epic Games [Internet]. [Viitattu 6.11.2024]. Saatavilla: https://dev.epicgames.com/documentation/en-us/unreal-engine/1.10---world-position-offset?application_version=4.27

29. Creating Textures Using Blueprints and Render Targets. Epic Games [Internet]. [Viitattu 3.11.2024]. Saatavilla: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-textures-using-blueprints-and-render-targets?application_version=4.27

30. Actor Ticking. EpicGames [Internet]. [Viitattu 3.11.2024]. Saatavilla: https://dev.epicgames.com/documentation/en-us/unreal-engine/actor-ticking-in-unreal-engine