LAB University of Applied Sciences

Fluidisimulaatioiden ja visuaalisten tehosteiden käyttö peleissä

Case: Simulaatio Pixtell Oy

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
2022
Konsta Rinta-Runsala

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Rinta-Runsala, Konsta	Opinnäytetyö, AMK	2022
	Sivumäärä 35	

Työn nimi

Fluidisimulaatioiden ja visuaalisten tehosteiden käyttö peleissä

Tutkinto

Insinööri (AMK)

Toimeksiantajan nimi, titteli ja organisaatio

Henri Koskinen, CEO, Pixtell Oy

Tiivistelmä

Opinnäytetyössä tutkittiin erikoistehosteissa käytettävien fluidisimulaatioiden luontiin liittyviä tekniikoita ja menetelmiä. Näiden metodien avulla yritettiin selvittää fluidien käytettävyyttä videopeleissä Pixtell Oy:n Last Drop -Legend of Seppo -peliprojektia varten. Peliprojektiin haluttiin simulaatioassetteja myöhemmin tehtäviä elokuvamaisia välikohtauksia varten.

Projektin simulaatioiden visuaalinen ulkonäkö suunniteltiin ja toteutettiin SideFX:n Houdini -ohjelmaa käyttäen. Houdini toimii alan standardina visuaalisten tehosteiden teossa. Simulaatioiden on tarkoitus toimia Unreal Engine -pelimoottorissa, sillä peliä suunnitellaan kyseisellä ohjelmalla. Työssä selvitetään simuloidun geometrian ja partikkeliefektien vientiä Houdinista Unreal Engineen. Tämä suoritettiin FBX- ja Alembic-tiedostoilla sekä Niagara-lisäosaa käyttäen.

Lopputuloksista huomattiin pelimoottorin rajallisuudet ja vapaudet verrattuna elokuvateollisuuden menetelmiin tehdä fyysisiä simulaatioita. Pelimoottorin sisällä luotavat efektit ovat huomattavasti kevyempiä kuin siihen siirrettävät valmiiksi laskelmoidut tiedostot. Reaaliaikainen renderöinti luo niin peleihin kuin elokuviinkin paljon mahdollisuuksia, mutta myös rajoituksia, jotka teknologian ja tekniikoiden kehittyessä vähenevät.

Asiasanat

simulaatio, pelimoottori, renderöinti, fluidi, visuaalinen tehoste

Abstract

Author(s)	Type of Publication	Published
Rinta-Runsala, Konsta	Thesis, UAS	2022
	Number of Pages	
	35	

Title of Publication

The use of fluid simulations and VFX in games

Name of Degree

Engineer (UAS)

Name, title and organization of the client

Henri Koskinen, CEO, Pixtell Oy

Abstract

The thesis studies the techniques and methods related to the creation of fluid simulations used in special effects. These methods were used to determine the usability of fluids in video games for Pixtell Oy's Last Drop -Legend of Seppo game project. Fluid simulation assets were wanted for the game project’s cinematic scenes, which are finalized later.

The visual appearance of the project simulations was designed and implemented using SideFX's Houdini program. Houdini is the industry standard for creating visual effects. The simulations are meant to work in Unreal Engine, as the game is designed with it. The thesis investigates the export methods of simulated geometry and particle effects from Houdini to Unreal Engine. This was done with FBX and Alembic files and using the Niagara plugin.

The end results showed the differences between game engines and movie industry when it comes to fluid simulations. The effects created inside the game engine are much lighter than the pre-calculated files transferred to it. Real-time rendering creates a lot of opportunities for both games and movies, and the number of limitations decreases as technology and techniques evolve.

Keywords

simulation, game engine, rendering, fluid, vfx

Sisällys

Keskeiset käsitteet

Alembic: Tiedostomuoto, joka tallentaa monimutkaisia animoituja kohtauksia valmiiksi laskettuun ohjelmasta riippumattomaan muotoon.

Asset: 3D-teknologialla tuotettu valmis tuote, esim mallinnus tai simulaatio.

Frame: Merkkaa siirtymän animaatiossa alusta loppuun.

FBX: Tiedostomuoto, jota käytetään siirtämään esimerkiksi 3D-geometriaa tai animaatiodataa.

Fluidi: Epäkiinteä, virtaava ja muotoa muuttava aine, kuten neste, kaasu tai plasma.

Node: Datapiste verkostossa, joka hajaantuu ja yhdistyy muihin datapisteisiin.

Rasterointi: Muuntaa vektoreista muodostuvan kuvadatan pikseleiksi, näytöltä katselua varten.

Renderöinti (engl. Render): Muuntaa digitaalisen tiedon näytölle sopivaan muotoon, esimerkiksi 3D-näkymän objektit kuvaksi.

Shader: Menetelmä, joka laskelmoi 3D-näkymässä kappaleisiin kohdistuvan valon, varjon ja värin määrän.

Solver (sets of equations): Laskee objektin käyttäytymistä kunkin aikayksikön kohdalla.

VDB (Volume/Voxel Data Base): Tiedosto, joka sisältää tilavuutta simuloivien efektien dataa, kuten pilviä, savua tai veden pintaa.

Vokseli (engl. Voxel): Kuvastaa yksittäistä arvoa kolmiulotteisessa koordinaatistossa.

VOP (Vector Operator): Houdinin node-verkko, jolla voidaan muuttaa mm. geometriaa, pikseleitä ja tiheyttä.

1 Johdanto

Digitaalisen maailman ja sen efektien luomisessa simulaatio- ja dynamiikkatekniikat ovat ydinosassa. Niillä pyritään imitoimaan todellisen maailman fyysisiä ominaisuuksia, jotta katsojalle saadaan mahdollisimman todentuntuinen kokemus. 1990-luvun alusta lähtien erikoistehosteet ovat siirtyneet fyysisestä digitaaliseen muotoon. Nykyään on tavallista käyttää digitaalisia simulaatioita monimutkaisten objektien, fluidien aiheuttaman vuorovaikutuksen kuvantamiseen. Fluideja ovat epäkiinteät, virtaavat ja muotoa muuttavat aineet, kuten nesteet, kaasut sekä plasmat. Fluidisimulaatioissa käytettävät tekniikat ovat hyvin samanlaisia elokuvien erikoistehosteissa ja videopeleissä. Suurimpina ja vaikuttavimpina eroina ovat reaaliaikainen renderöinti ja se, että kamerahenkilönä toimii useimmissa tapauksissa itse pelaaja.

Opinnäytetyö käsittelee fluidien simulaatioissa käytettäviä tekniikoita, niiden visuaaliseen tyylin sovellettavia käytäntöjä sekä niiden renderöinnissä hyödynnettäviä keinoja. Tutkimus käsittelee myös fluidisimulaatioiden tuontia ja mukauttamista peleihin. Tämä tehdään Pixtell-videotuotantoyrityksen Last Drop -Legend of Seppo -peliprojektin kanssa, jonka toimeksiantajana toimii Henri Koskinen. Peliin toteutetaan välikohtauksessa käytettävä sortuva vesitorni sekä pelatessa tapahtuva kerrostalohuoneiston räjähdys. Projektiin kuuluu rakennusten aidonnäköinen dynaaminen hajoaminen sekä simulaatiot vedelle ja savulle. Näiden toteutuksessa hyödynnetään SideFX:n Houdini-ohjelmaa sekä Unreal Engine -pelimoottoria.

Houdini on yksi alan standardiohjelmista, jolla voidaan luoda täsmälleen sellaisia efektejä ja simulaatioita kuin halutaan monipuolisten kustomointimahdollisuuksien vuoksi. Se sopii myös pelimoottorin kanssa käytettäväksi, kun halutaan tietynlainen tulos, jota ei välttämättä olisi helppoa toteuttaa pelkillä pelimoottorin työkaluilla. Tätä edesauttaa partikkelisimulaatioihin erikoistuva Niagara-lisäosa, assettien siirtoa helpottava Houdini Engine, sekä FBX- ja Alembic-tiedostot. Unreal Engine on yksi moderneimmista pelimoottoreista ja se toimii Houdinin kanssa suhteellisen vaivattomasti.

Opinnäytetyössä tutkitaan simulaatiopohjaisten visuaalisten tehosteiden luontia, toimintaa sekä tuontia 3D-ohjelmasta pelimoottoriin. Tämä projekti auttaa myös ymmärtämään keskeisiä menetelmiä fotorealististen sekä peleissä esiintyvien simulaatioiden saavuttamisessa. Työssä selitetään, miten simulaatioiden ulkonäköä muokataan ja mikä työprosessissa saa ne näyttämään realistiselta. Reaaliaikaisesta renderöinnistä selvitetään sen tuomat haasteet suorituskyvyn kannalta. Tutkimuksessa käsitellään myös pelialan eroja ja yhtäläisyyksiä elokuvateollisuuteen.

2 Dynamiikan ja simulaatioiden perusteet tietokonegrafiikassa

2.1 Simulaatioiden luontiprosessi

Dynamiikassa ja fluidisimulaatioissa käytetyillä tekniikoilla halutaan kopioida oikean maailman luonnonmukaisia ja realistisia piirteitä digitaaliseen ympäristöön. Realististen fysikaalisten ilmiöiden käyttäytymisen saa helpoimmin aikaan tietokonelaskennalla kuin käsin mallintaen ja animoiden. Simulaatioita käytetään usein katastrofeihin ja tuhoamiseen esimerkiksi murtuvien objektien, räjähdysten, tulen, savun ja nesteiden kuvantamiseen. Simulaatioita voidaan kuitenkin myös käyttää esimerkiksi vaatteisiin, hiuksiin ja lihaksiin. (Crow 2021, 568–569.)

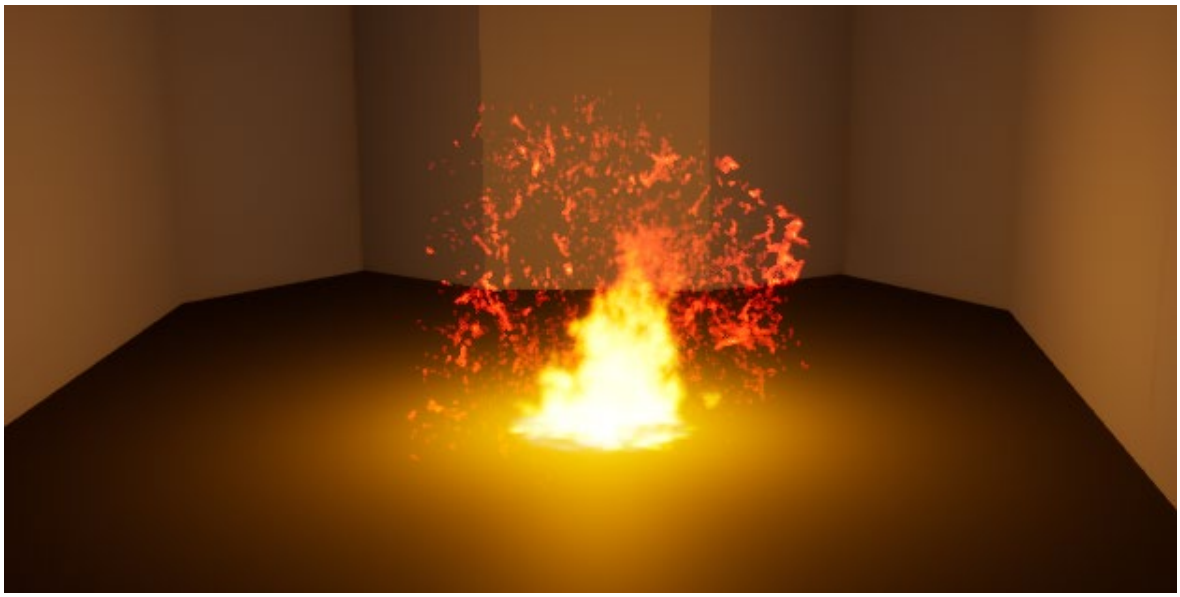
Simulaatioiden luonti alkaa objektilla, jolle asetetaan fysikaalisia ominaisuuksia kuten massa, tiheys tai elastisuus. Painovoima ja tuuli ovat esimerkkejä voimista, joita erilaiset solverit (sets of equations) käyttävät laskemaan objektin käyttäytymistä kunkin framen aikana. Cloth ja rigid-body -solvereita voidaan käyttää samaan aikaan laskemaan vaatekappaleen muotoutumista ja sen vuorovaikusta muihin objekteihin. Simulaation työstäminen on usein alkutilan asetusten muuntamista ja simulaation laskemista niin kauan, että saadaan haluttu lopputulos. Simulaatiot vaativat paljon aikaa, sekä tietokoneelta RAM:ia ja levytilaa. Tämän vuoksi projektin alkuvaiheessa täytyy olla varma, tarvitaanko simulaatiota vai onnistuuko se kevyemmällä keyframe-animaatiolla. Hidasta työprosessia voidaan nopeuttaa pienemmällä resoluutiolla ja epätarkemmalla simulaatiolla esikatseluvaiheessa. Yksityiskohtia voi alkaa lisäämään, kun tyyli on valmis. Kiireisessä työaikataulussa prosessointitehon ja sen puutteiden hallitseminen on suoraan verrannollinen luovuuteen käytettävään aikaan. (Crow 2021, 568–569.)

Simulaatioita on hyvä hajottaa osiin, jolloin muokkaamisesta tulee selkeämpää ja nopeampaa, kun useampaa elementtiä voidaan muokata samanaikaisesti. Tämä voi tarkoittaa esimerkiksi hyökyaallon roiskeiden, vaahdon ja aallon muodon simuloimista erikseen. Mitä enemmän halutaan sujuvuutta itse simulaatioiden esikatselua varten, sitä enemmän vaaditaan myös datalle tilaa. Simulaatioiden kanssa ollaan lähes aina tekemisissä myös muiden 3D-assettien kanssa, joita voidaan käyttää simulaatioiden ajamisessa. Tällöin myös datan siirto muista ohjelmista on tärkeää. Simulaatioiden luonnista tekee helpompaa oikeiden mittasuhteiden käyttö sekä oikeiden arvojen kuten painovoiman ja kitkan käyttö. On hyvä ymmärtää myös jälkikäsittelyn painoarvo simulaatioiden suunnittelussa, että päästään vähimmillä resursseilla lopputulokseen. Jälkikäsittelyssä voidaan esimerkiksi lisätä yksityiskohtia tekstuureilla renderöintivaiheessa. (Crow 2021, 570.)

2.2 Partikkelit ja dynamiikka

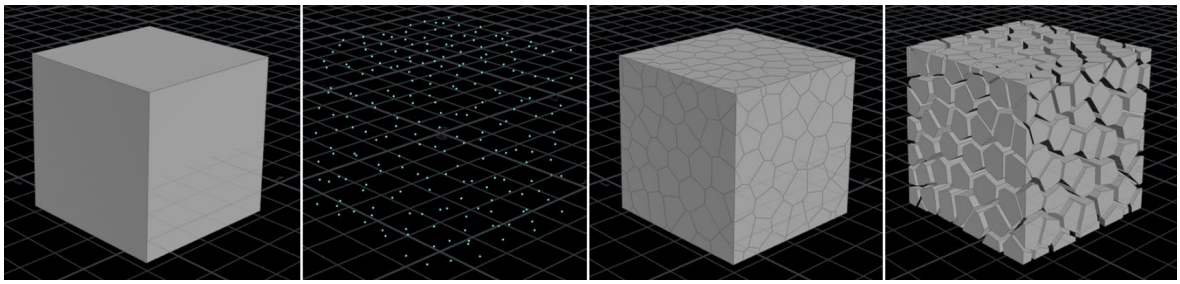
Partikkelit ovat lähes kaikkien tietokonesimulaatioiden perusta. Ne ovat pisteitä kolmiulotteisessa koordinaatistossa, joille voidaan antaa erilaisia ominaisuuksia. Tällaisia attribuutteja voivat olla esimerkiksi väri, massa, nopeus, suunta tai elinaika. Partikkeleita voidaan käyttää värikkäinä pisteinä, mutta ne pitävät paikkaa pääasiassa erilaiselle geometrialle, esimerkiksi kuville tai 3D-malleille. Tämä mahdollistaa esimerkiksi eläinparvien arvaamattoman liikehdinnän luonnin. (Zerouni 2021a, 571.)

Partikkelien pääominaisuus on niiden liike. Liikkuminen määräytyy partikkelien sisälle asetetuista arvoista, kuten valmiiksi asetetusta nopeudesta ja suunnasta sekä ulkopuolisista voimista, joita voidaan simuloida matemaattisesti. Ulkopuolisia voimia ovat esimerkiksi painovoima, tuuli ja turbulenssi. Partikkelien animointi tapahtuu siis joka framen aikana lasketusta suunnasta, nopeudesta ja ulkopuolisista vaikuttajista. Tämän lisäksi partikkeleihin vaikuttavat myös niiden keskeiset hylkivät tai puoleensavetävät voimat. Partikkeleita voidaan synnyttää tasaisena virtana tai vain lyhyesti reaktion aiheuttamana. Esimerkiksi tasainen partikkelivirta voi kuvastaa nuotiosta lähteviä kipinöitä, ja luodin osuminen seinään voi aiheuttaa pienien partikkelien irtoamisen seinästä. Syntyessään sadoille partikkeleille voidaan antaa satunnaistetut lähtöarvot, mutta tämän jälkeen ne simuloidaan, sillä niihin alkaa vaikuttaa lähtöarvoista riippumattomat ympäristön voimat. (Zerouni 2021a, 572–573.) Partikkelien liikkeen ja muodon tehokkaalla yhteiskäytöllä simulaation voi saada näyttämään tulelta, savulta tai vaikka nesteeltä (kuva 1).



Kuva 1. Partikkelisimulaatiolla tehty nuotio

Rigid-body dynamics (RBD) tarkoittaa fyysisiä simulaatioita, joissa simuloitava kohde ei muuta muotoaan. Näitä hyödynnetään objektien luonnollisen käyttäytymisen saavuttamiseksi esimerkiksi massan ja painovoiman avulla. RBD-ominaisuuksia voidaan hyödyntää myös edellä mainituissa partikkeleissa. Rigid-body -dynamiikkaa käytetään suurilta osin tuhoutumisen simuloimiseen. Tämä vaatii geometrian hajottamisen palasiin etukäteen ja sen liimaamisen takaisin yhteen. Objektin luontaisesti tapahtuva hajoaminen ei ole vielä helposti saavutettavissa nykyohjelmilla. (Zerouni 2021b, 576.) Valmiiksi suunniteltava kappaleen murtuminen mahdollistaa kuitenkin artistisen panostuksen tuhottavan kappaleen teossa. Itse hajoavat palaset ohjataan usein myös partikkelien avulla (kuva 2). Enemmän muokkaavuutta tuo ulkopuolisten voimien lisääminen simulaatioon, kuten esimerkiksi räjähdys.



Kuva 2. Partikkeleilla ohjattu kuution hajoaminen

Kankaan tai hyytelön simuloiminen hyödyntää hyvin samoja menetelmiä kuin rigid-body -simulaatiot, mutta siinä animoidaan myös kappaleen muodonmuutosta. Cloth-simulaatiossa ulkoisia vaikuttavia voimia ovat esimerkiksi painovoima ja pakotettu venyminen. Kappaleeseen vaikuttavia sisäisiä ominaisuuksia ovat elastisuus, vetojännitys, repeytyvyys, taittuvuus sekä ympäristön törmäyksistä aiheutuvat reaktiot kappaleessa. (Hughes ym. 2007.) Venyvä kappale saadaan aikaiseksi simuloituvien partikkelien avulla, jotka on yhdistetty geometriaan (kuva 3).

Kuva 3. Cloth-simulaatiolla tehty lippu.

3 Fluidisimulaatiotekniikat

3.1 Fluidit tietokonegrafiikassa

Nesteanimaatiolla tarkoitetaan tietokonegrafiikalla toteutettua realistisen näköistä fluidia. Nesteanimaatioissa käytetään usein hyödyksi virtausdynamiikan yhtälöitä, mutta ne ovat pääasiassa visuaalisia tehosteita varten yksinkertaistettuja. Numeerisen virtausdynamiikan (Computational Fluid Dynamics, CFD) menetelmät kuvaavat tarkasti tosielämän kaasujen ja nesteiden käyttäytymistä ja ovat siksi usein liian raskaita, komplekseja ja huonosti skaalautuvia käytettäväksi tietokonegrafiikassa. (Seymore 2011.)

> *Emme yritä keksiä tai parantaa sitä (CFD), parantaa tiedettä, vaan yritämme hyödyntää sitä. Joten kun meillä on taiteellisia ongelmia, emme välttämättä ole huolissamme luonnon absoluuttisen todellisuuden simuloimisesta. Huomaamme usein, että absoluuttinen todellisuus ei ole tärkeä, ja syy siihen on se, että todellisuus, jota todella tavoittelemme, on se, joka on ohjaajan mielessä. (Tessendorf 2010, kirjoittajan suomennos)*

Navierin–Strokesin yhtälöt kuvaavat erilaisten fluidien käyttäytymistä. Navierin–Strokesin yhtälöistä ei saada ratkaisuksi tiettyä positiota, vaan nopeusvektori, jota kutsutaan nopeuskentäksi (velocity field). Kyseessä on advektioyhtälö, joka osoittaa miten partikkelit liikkuvat nopeuskentän sisällä. Yhtälö ottaa huomioon fluidin osien liikemäärästä johtuvan paineen sekä viskositeetin eli fluidin rakenneosien kitkan muutokset (kaava 1).

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u}$$

$$\nabla \cdot \vec{u} = 0$$

Kaava 1. Oletetaan, että neste on puristautumaton ja viskositeetti on 0: nesteen osien kiihtyvyys = painovoima – paineen muutos / tiheyden muutos (Barley 2009)

3.2 Simulointimetodit

Seuraavat teknologiat ovat kehittyneet vuosien varrella ja muotoutuneet omia käyttökohteita varten. Vaikka ne käyttävät suurelta osin samoja kaavoja, ne hyödyntävät niitä eri tavoin eri tarkoituksiin. Menetelmät käyttävät esimerkiksi partikkeleita, ristikoita tai molempia.

Smooth Particle Hydrodynamics (SPH)

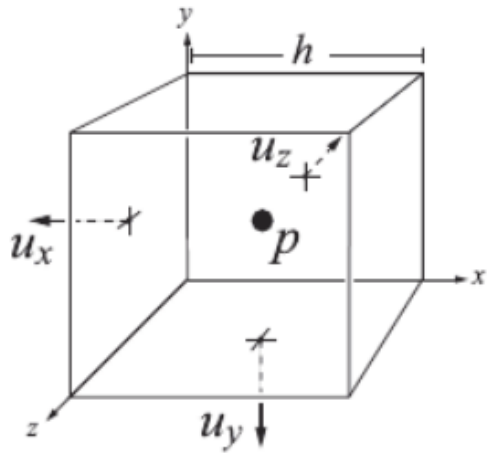
SPH:ssa käytetään Navierin-Strokesin yhtälöitä liikuttamaan partikkeleita, jotka myöhemmin muutetaan monikulmioiseksi tasoksi. Monikulmiot ovat tällöin simulaation renderöitävä osa. SPH sopii hyvin esimerkiksi kohtauksiin, joissa kaadetaan vettä lasiin, mutta voi olla tietokoneelle liian raskas esimerkiksi meren kuvantamiseen. (Seymore 2011.)

Volume Grid

Volume Grid -menetelmä ei käytä ollenkaan partikkeleita, vaan ristikkoa, jossa vokseleille eli kolmiulotteisen koordinaatiston sijaintiarvoille, määräytyy tietty korkeus ja taajuus. Näin vokselit muodostavat mallin tilavuudella eikä esimerkiksi monikulmaisilla tasoilla. Tätä menetelmää käytetään useimmin suurien vesimassojen, kuten meren, kaukaa kuvantamiseen. Yksityiskohdat ovat epätarkkoja, mutta tarpeeksi realistisia ihmissilmälle. (Seymore 2011.)

Marker and Cel method (MAC)

MAC-metodi on yksi suosituimmista tavoista simuloida nesteen virtausta. Sillä kuvannetaan alue kuutiomaisilla soluilla, joilla on leveys, paine keskellä sekä nopeus kuution kolmella pinnalla (Kuva 4). Tämä tekee simulaatiosta vakaamman kuin se, että nopeus olisi kuution keskipisteessä. Simulaatio toimii nopeuskentän määräaikaisella päivittämisellä. Markerhiukkasia liikutetaan nopeuskentässä, että saadaan selville fluidin sijainti tietyllä alueella, jota taas käytetään seuraavan framen laskemiseen. (Cline ym. 2015.)



Kuva 4. Cel partikkeli (Cline ym. 2015)

Particles in Cels (PIC)

PIC:ssä partikkelit lasketaan ryhmänä. Partikkelit vaikuttavat ristikkoon ja siirtävät vektoridatan siihen. Tämän jälkeen data lasketaan ja ohjelma liikuttaa partikkeleita sen mukaisesti. Tämän jälkeen prosessi toistetaan. Simulaatiodataa pystytään vähentämään, kun partikkeleita käsitellään ryhmänä. PIC-metodilla saadaan aikaan sulavia simulaatioita, koska data saadaan partikkelien keskinäisistä vuorovaikutuksista eikä keskiarvoista. (Seymore 2011.)

FLIP

FLIP solver on hybridimetodi, joka perustuu partikkeleihin ja tilavuuteen. Tässä metodissa kaikki data on partikkelien sisällä, jolloin fluidien sekoittumista ja katoamista ei voi tapahtua. FLIP solverin hyvänä puolena on se, ettei sen tarvitse laskelmoida partikkelien liikettä useampaan kertaan framen aikana. Joskus FLIP solverillakin voidaan lisätä laskentakertoja yhdellä aikavälillä (substeps), mutta tällöin se ei voita ajassa esimerkiksi SPH-menetelmää. (SideFX 2022b; Seymore 2011.)

Nesteen ja kaasun ero

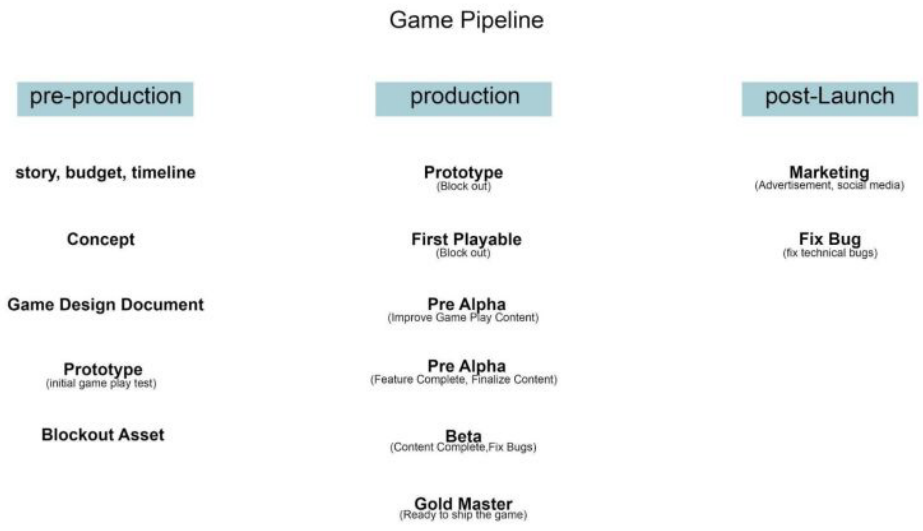
Kaasusimulaatiot käyttävät paljolti samoja metodeja kuin nesteetkin, koska molemmat ovat fluideja. Kaasuissa täytyy kuitenkin ottaa huomioon vapaampi liikkuvuus sekä ilman liikkeeseen ja koostumukseen vaikuttavat voimat. Näitä ovat esimerkiksi lämpötila ja lämmön lähde. Lämpötilan ero savun ja ympäröivän ilman välillä saa aikaiseksi esimerkiksi nostovoiman. Lämmön lähteellä saadaan taas aikaiseksi mielenkiintoisempaa liikettä fluidin nopeutta muuttaessa. Koska savulla ei ole samanlaista rajapintaa kuin nesteellä, tietokonegrafiikassa käytetään ulkoisia fysikaalisia voimia kuten nostetta ja painovoimaa. Näiden lisäksi savun simuloinnissa käytetään Vorticity confinement -metodia, jolla voidaan kontrolloida savun pyörivää ja turbulenttista luonnetta. (Zhanpeng ym. 2014, 7573–7577.)

4 Simulaatiot peli- ja elokuvateollisuudessa

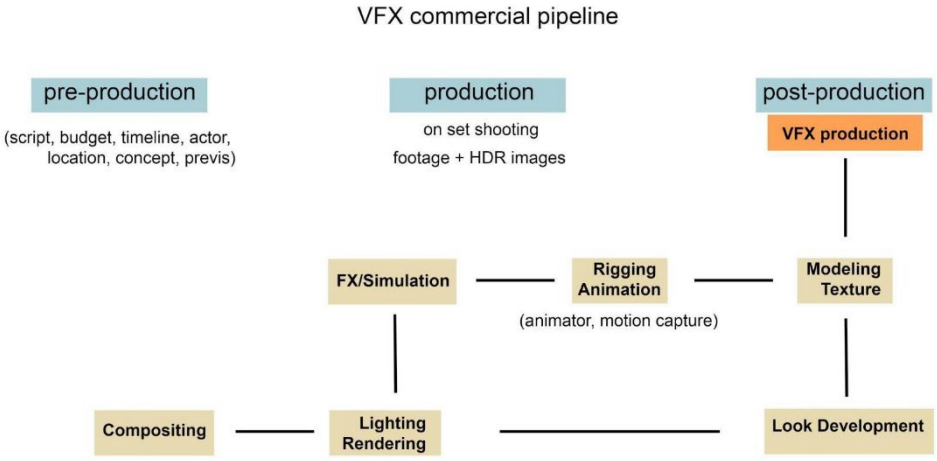
4.1 Ammatit ja työtehtävät

David Johnsonin (2021a, 641–642) mukaan pelistudio jaetaan usein kolmeen suurempaan osaan: Design, Engineering ja Art Department. Itse pelattavuudesta, muun muassa tavoitteista ja vuorovaikutuksista, on vastuussa Game Design -osasto. Tämän osaston johtajaa voidaan verrata elokuvateollisuuden ohjaajaan. Engineering osasto työskentelee koodaten laajentaakseen pelimoottorin kyvykkyyttä esimerkiksi yhteyksien, grafiikan tai äänien osalta.

Art Department sisältää erikoistehosteita koskevat roolit, kuten mallinnus, valaistus, animaatio sekä tehosteet. Työtehtävät ja niiden painoarvo voivat kuitenkin muuttua eri studioiden välillä. Usein kaikki Art Departmentin alla työskentelevät vastaavat Art Directorille. Peleissä ei ole erillistä compositor-roolia. Efektiartistit hoitavat kyseisen elokuvateollisuudessa tutun roolin istuttamalla erilaisia elementtejä pelinäkymään, kuten tulta ja savua. Tech Artist yhdistää koodaamisen ja taiteen. Heidän työnkuvaansa kuuluu ratkaista suorituskykyyn liittyvät ongelmat pelin visuaalisuudessa. Joissain tapauksissa he ovat vastuussa myös simulaatioista, partikkelitehosteista tai efektiartistien auttamisesta. (Johnson 2021a, 642–643; Johnson 2021a, 649–651.) Kuvista 5 ja 6 näkyy miten simulaatioita ja tehosteita tehdään suurilta osin videopeleissä jo tuotantovaiheessa. Mainos- ja elokuva-alalla huomattavampi osuus erikoistehosteista tehdään vasta jälkikäsittelyvaiheessa (Zhao 2020).



Kuva 5. Peleissä käytettävä työnkulun prosessi (Zhao 2020)



Kuva 6. Mainoksissa käytettävä työnkulun prosessi (Zhao 2020)

Peleihin tarvitaan monenlaisia efektejä. Visuaalisiin tehosteisiin erikoistuvat keskittyvät usein pelimoottorin partikkelieditoriin. Näiden lisäksi osasto kehittää omia tekstuureja tekstuurikirjastojen lisäksi. Tekstuureilla muodostettavat animaatiotkin ovat usein käytettyjä niiden kevyen koon vuoksi. Tekstuureissa voidaan käyttää kuvattua, simuloitua tai käsin animoitua materiaalia esimerkiksi savua tai räjähdystä varten. Järjestelmän efektit ovat tehosteita, jotka tapahtuvat pelaajan toimesta. Esimerkiksi, jos pelaajan halutaan käyttävän supervoimia, voidaan näitä tehostaa äänellä ja visuaalisilla parannuksilla. Mallien, partikkelien ja materiaalien yhteiskäytöllä saadaan kehitettyä edellä mainittuja visuaaleja kuin myös aseiden ja kulkuneuvojen tehosteita. Ympäristöön kuuluvat erilaiset simuloidut efektit ovat esimerkiksi pienet soihdut, kärpäset tai usva, jotka taas luovat realismia pelialueeseen. Monissa peleissä myös rikkoutuva ympäristö on yleistä, kuten seinät tai vaasit. Yksinkertainen fysiikka simulaatioissa pienissäkin asioissa saa aikaan paljon immersiota. (Johnson 2021a, 649–651.)

4.2 Pelimoottorit ja reaaliaikainen renderöinti

3D-mallinnus- ja -animointiohjelmia käytetään paljon hyödyksi erilaisissa pelimoottoreissa. Pelimoottorit koostuvat työkaluista, joita käytetään esimerkiksi mallien, materiaalien, tekstuurien, animaatioiden ja muiden assettien tuontiin ja muokkaamiseen. Suosituimpia pelimoottoreita ovat esimerkiksi Unreal Engine ja Unity. On myös olemassa peliyritysten itsekehittämiä pelimoottoreita, mutta suuri osa niistä ei ole saatavilla julkiseen käyttöön. Yleisemmin pelimoottorista löytyy pelimaailman luomiseen tarkoitettu käyttöliittymä. Tässä

asetetaan pelikenttään esimerkiksi 3D-mallit, valot ja hahmot. Koodilla lisätään muun muassa pelimaailman interaktiivisuutta, kuten vihollisten ilmestyminen tiettyyn alueeseen tultaessa tai aseen ampuminen kontrollerin napin painalluksella. Unityssä käytetään C# -koodauskieltä, kun taas Unreal Engine käyttää node-pohjaista visuaalista koodausta (Blueprints). (Johnson 2021c, 638; Johnson 2021d, 638.)

Editorien lisäksi pelimoottorit sisältävät pelin aikana suoritettavan prosessin, joka on vastuussa kaikkien edellä mainittujen assettien esittämisestä reaaliajassa. Tämän suorittaa konsoli tai mikä tahansa pelaamiseen kelpaava tietokone, joka luo saaduista määritelmistä pelaajalle kuvasta ja äänestä koostuvan elämyksen. Prosessiin kuuluu toistuva päivittäminen, joka laskelmoi esimerkiksi pelin fysiikkaan liittyvät kappaleiden positiot tai efekteihin kuuluvien partikkelisimulaatioiden vaiheet. Suurimman osan pelin päivittymisestä hoitaa prosessori. (Johnson 2021c, 639–640.) Yksi tärkeimmistä eroista verrattuna perinteiseen videon renderöintiin on, että 3D-ohjelman renderöintimoottori keskittyy vain siihen mitä määrätystä kamerasta näkyy. Suoritustehoa rajaa suurimmilta osin valotus ja valon säteiden heijastumiskerrat (samples) ympäristöstä. Heijastuskerrat lisäävät renderöintimoottorin laskenta-aikaa huomattavasti, mutta tekee valaistuksesta aidomman näköistä. 3D-ohjelmilla renderöitäessä monet valoa koskevat asetukset voidaan tehdä renderöintiasetuksista. Pelien valotuksessa joutuu ottamaan huomioon muun muassa dynaamisten valojen vaikutukset ympäristön muodostamiin varjoihin sekä päällekkäin asettuvat valojen ja varjojen lähteet. (Zhao 2020.) Myös shaderit täytyy ottaa peleissä eri tavalla huomioon. Shaderit ovat menetelmiä, jotka laskelmoivat 3D-näkymässä kappaleisiin kohdistuvan valon, varjon ja värin määrän (Upstill 1990, 209–211). Lopputulokseen vaikuttavat huomattavasti valojen ja kameran paikat, jolloin pelien teossa korostuu valaistuksen vaikutukset useammasta kuin yhdestä suunnasta.

Reaaliaikainen renderöinti vaatii suuren määrän muistia pystyäkseen tuottamaan pikseleistä koostuvan valmiin kuvan moniosaisesta datavirrasta. Tähän dataan kuuluu: mallien sijainnit, näkymään määritetyt valotukset sekä jälkikäsittelyä koskevat asetukset. Se, miten tämä osa renderöintiä tapahtuu, riippuu itse pelimoottorista ja renderöintitavasta. Eri tekniikoihin kuuluu esimerkiksi jokaisen objektin ominaisuuden rasterointi, kuten valotuksen, tekstuurin ja shaderin renderöinti pikseli kerrallaan. Rasterointi muuntaa vektoridatan pikseleiksi, näytöltä katselua varten. Toinen renderöintitapa on käsitellä jokaisen objektin frameen sisällytettävä data erikseen, kuten objekteista heijastuvan valon osuus tai pintojen muotoon liittyvät syvyyserot. (Johnson 2021c, 639–640.)

Huomattavin ero peleissä tietokonegrafiikkaan verrattuna on se, että kaiken täytyy olla reaaliajassa renderöityä. Tällöin geometrian, materiaalien ja simulaatioiden on oltava

kevyempiä ja vakaampia. Realistisuus täytyy toteuttaa eri menetelmin kuin esimerkiksi animaatiossa. Nesteet voidaan toteuttaa esimerkiksi liikkuvilla tekstuureilla, valmiiksi animoidulla geometrialla tai pelimoottoriin sisällytettävällä partikkelipohjaisella simulaatiolla. Sea of Thieves -pelissä (Rare, 2018) näyttää kuin meri olisi tehty animoidulla tasolla, joka on shadereilla saatu näyttämään realistiselta aallokolta (kuva 7). Myrskyinen aallokko tai siihen osuvat objektit kuten laiva synnyttää roiskeen näköisiä partikkeleita ja tekstuuria.

Kuva 7. Sea of Thieves -pelin simulaatio (Rare, 2018)

Kaasut ja tuli toteutetaan usein partikkelisimulaatiolla, jonka partikkeleihin lisätään spritetekstuuri, joka useimmassa tapauksessa käy kuvasarjan läpi partikkelin iän kuluessa. Tällä menetelmällä saadaan suhteellisen realistista savua, mikä on myös kevyt prosessoida tietokoneella. (Johnson 2021a, 649.) Esimerkkinä hyvin saavutetusta savusimulaatiosta pelissä on It Takes Two -pelin kohtaus, jossa savu elää ja liukuu lattiaa pitkin (kuva 8). Savuun käytettävästä partikkelisimulaatiosta saadaan realistisempi, kun sen käyttäytymistä ohjataan savusimulaatiolla esimerkiksi Houdini-ohjelmaa käyttäen. Johnsonin (2021a, 651) mukaan pelimoottoreita ei ole tehty esimerkiksi spektaakkelimaisten tuhoutumisten luomiseen. Siksi Alembic-tekniikkaa on alettu käyttää useammin pelien teossa. Alembic-tiedostoilla voidaan tallentaa monimutkaisia muotoa muuttavia 3D-malleja sovelluksesta riippumattomaksi valmiiksi lasketuksi animoiduksi geometriaksi. Tämän kanssa tulee omat haasteensa, kuten

suurien data määrien lukeminen ja topologian muuttaminen varsinkin, jos sitä pyörittää ainoastaan pelimoottori.

Kuva 8. It Takes Two -pelin kohtaus savusimulaatiosta (Electronic Arts, 2021)

4.3 Suorituskyvyn vaatimukset ja tulevaisuus

Pelien tehosteista pyritään saamaan näyttäviä, mutta myös pysymään optimaalisen suoritustehon asettamien rajojen sisällä. Esimerkiksi pelillä, jonka virkistystaajuus on 60 framea sekunnissa, on aikaa suorittaa yhden framen muodostukseen käytävä prosessi 16,6 millisekunnin aikana. Tämän aikana näytönohjaimen ja prosessorin täytyy suorittaa tarvittavat prosessit. Prosessori voi kuitenkin jakaa kyseisen ajan sen ydinten määrällä. (Johnson 2021d, 655) Pelimoottorien renderöinnissä voidaan käyttää culling-tekniikkaa. Tämä tarkoittaa sellaisten objektien, kolmioiden tai pikselien poistamista, jotka eivät vaikuta lopulliseen renderöityyn kuvaan. Näitä ovat esimerkiksi kauas jäävät objektit, joiden yksityiskohtia ei erota lopullisessa kuvassa. Culling tehdään varhaisessa vaiheessa, ettei se itse vie paljoa suoritustehoa. (Cryengine, 2022.)

Jos jokin kohtaus on liian monimutkainen pelatessa suoritettavaksi, voidaan tehdä myös valmiiksi renderöityjä kohtauksia. Kerronnallisissa tarinakeskeisissä peleissä voidaan säilyttää pelaajan kontrolli juonen kulun kannalta esimerkiksi päätöstilanteissa: pelaajan tehdessä päätöksen, sen mukainen valmiiksi renderöity video lähtee pyörimään. Valmiiksi

renderöityjä kohtauksia luodaan usein pelimoottorilla, että saadaan aikaan sama tyyli kuin peliä pelatessa. Joillakin yrityksillä on omat animaatioita työstävät osastot. Joskus elokuvamaisia kohtauksia voidaan ulkoistaa elokuviin erikoistuneisiin studioihin mainontaakin varten (Johnson 2021e, 654). Esimerkki tällaisesta on The Call -musiikkivideo (Riot Games 2022) kuvassa 9. Pelimoottoreita käytetään myös runsaasti tehosteita käyttävän elokuvakohtausten esivisualisointiin.

Kuva 9. Kohtaus League of Legends -pelistä kertovasta musiikkivideosta (Riot Games 2022)

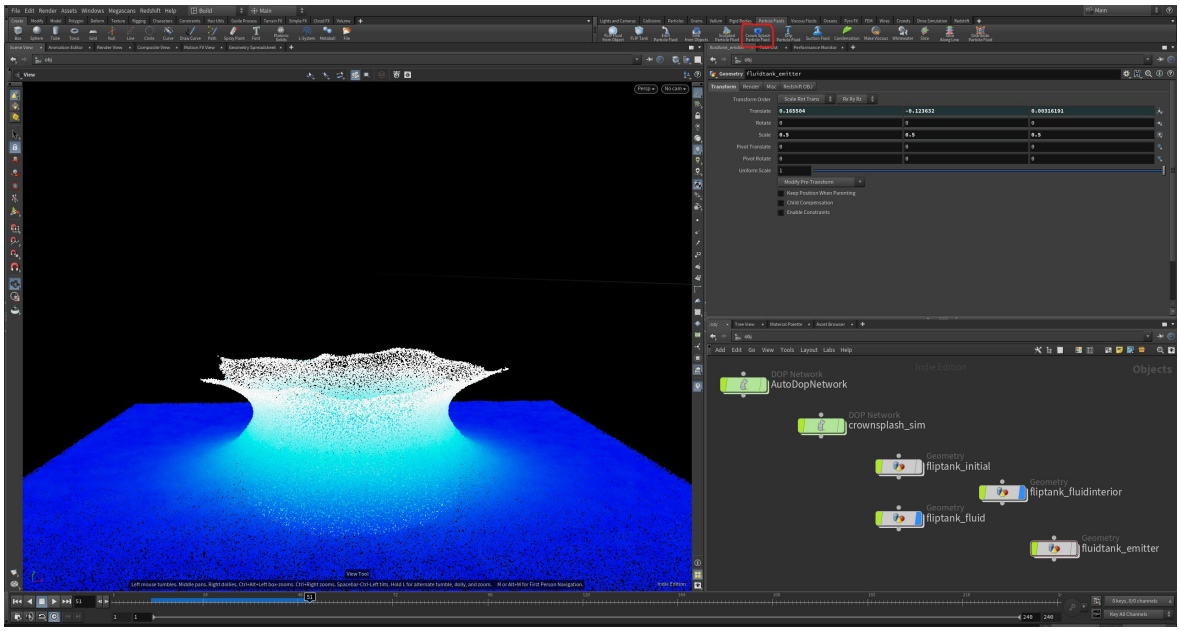
Teknologian kehittyessä pelisuunnittelussa saadaan jatkuvasti parempia tuloksia. Pelejä kehittäviä tekijöitä ovat nopeutuvat internetyhteydet, pilviprosessointi sekä helposti saatavilla olevat renderöintimahdollisuudet. Myös valon fyysinen simulointi on nykyään mahdollista reaaliajassa. Vaikka monet asiat reaaliaikaisessa renderöinnissä toimivat käytännössä samalla tavalla kuin valmiiksi renderöity materiaali, visuaalisissa tehosteissa on vielä paljon varaa kehittyä. (Johnson 2021b, 660.)

5 Houdini

5.1 Käyttökohteet ja käyttöliittymä

Houdini on elokuvateollisuuden standardiohjelma, jota käytetään pääasiassa visuaalisiin tehosteisiin. Houdini on mallinnus-, animaatio-, efekti-, simulaatio-, renderöinti- ja kompositointityökalu. Nesteen simulointiin voidaan käyttää useitakin ohjelmia. Suosituimpia näistä ovat Blender, RealFlow ja Bifrost. Houdinin vahvin ominaisuus on sen node-pohjainen proseduraalinen työnkulku, jonka takia sen vahvuudet ovat proseduraalinen mallinnus, ympäristön luonti, väkijoukkojen simulointi, hahmoefektit, partikkelit sekä dynamiikka (Kidd 2021, 630). Houdinin fluidisimulaatiomenetelmiin kuuluu FLIP solver, Volume fluid ja SPH fluid (SideFX 2022b). Houdini mahdollistaa perusteellisen kustomoinnin ja työstettävyyden, mikä tekee sen oppimiskynnyksestä moneen ohjelmaan verrattuna suurimman. SideFX:llä on myös lisäosa nimeltä Houdini Engine. Tämä mahdollistaa monen tiedoston hyödyntämisen ja siirtämisen eri 3D-ohjelmiin ja pelimoottoreihin.

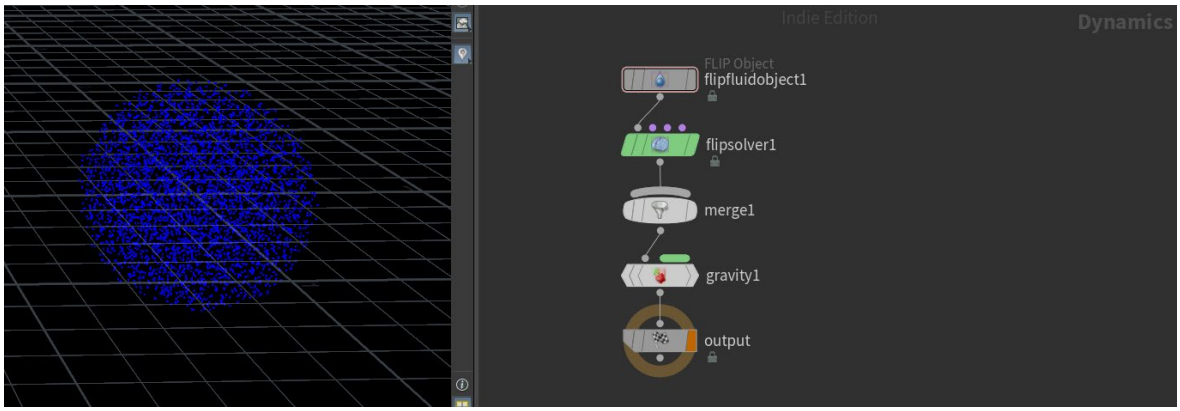
Houdinin käyttöliittymän perusnäkymä sisältää monia samoja elementtejä, joita muistakin 3D-ohjelmissa löytyy, kuten primitiivit ja erilaiset mallinnustyökalut. Houdinin omalaatuisen työkulun näyttävät parhaiten node-verkkonäkymä sekä jokaisen noden sisältämät tiedot. Houdinin yläpalkista löytyy myös Shelf Toolit, jotka tarjoavat valmiita nodeja tai node-kokonaisuuksia erilaisia tehosteita varten (kuva 10). (SideFX 2022f.) Näitä ovat esimerkiksi aallokot ja tuli. Houdini on täysin proseduraalinen, joten jokainen vaihe näkyy työnkulussa, jopa pienet muutokset geometriassa.



Kuva 10. Houdinin Shelf Tool kruunumaisen roiskeen tekoon

5.2 Simulaatioiden kustomointi

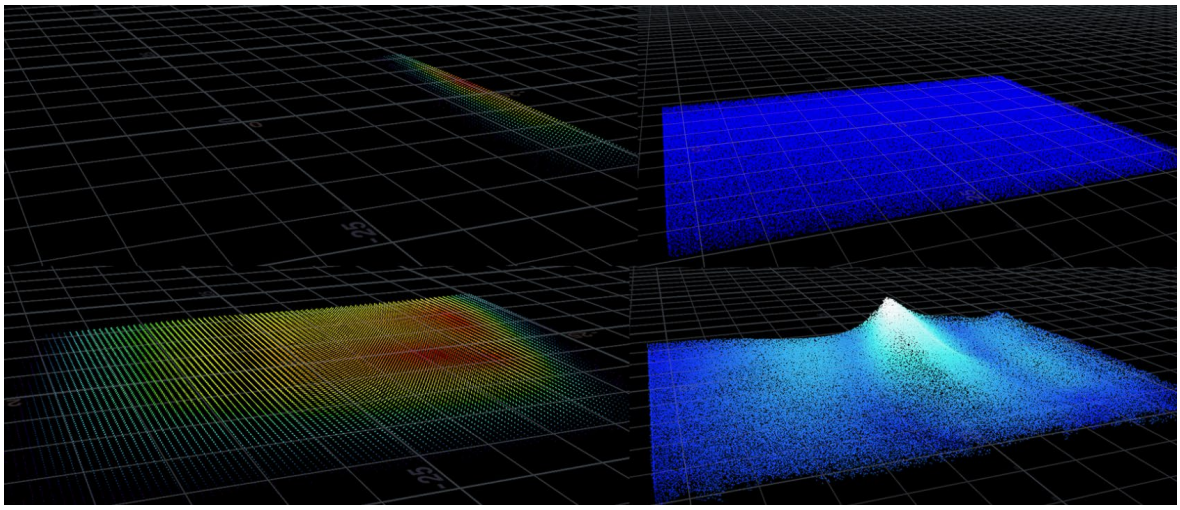
Fluidisimulaatiot tehdään Houdinissa käyttäen DOP Network -nodea (Dynamic OPerators). DOP käyttää hyväkseen valmiiksi konstruktoitua tai muokattua geometriaa, jota taas kutsutaan SOP:ksi (Surface OPerators) (SideFX 2022g). Esimerkiksi pallo voidaan muuttaa FLIP-objektiksi, jolloin se käyttäytyy kuin vesi (kuva 11).



Kuva 11. Flip-objektiksi muutettu geometria

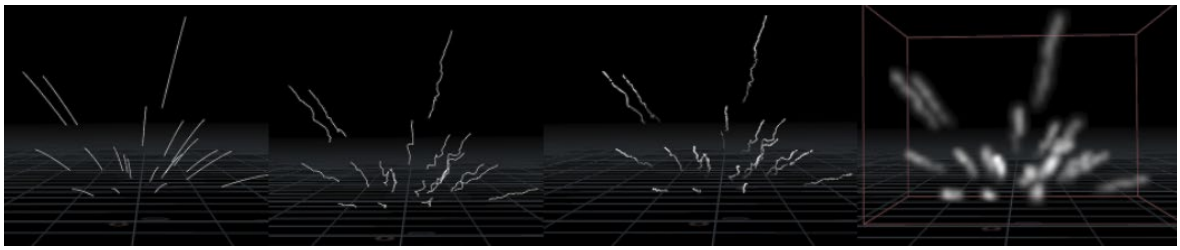
Nestesimulaatioita voidaan Houdinissa visualisoida useilla eri tavoilla. Yleisin näistä tavoista on partikkelit, joiden väri muuttuu niiden nopeuden mukaan. Partikkelien määrää lisäämällä saadaan simulaatiosta aidompi ja yksityiskohtaisempi. Tällöin laskentateho saattaa kasvaa, mikä tekee simulaation tarkastelusta hidasta ja vaivalloista. Simulaatiot eivät ole näytönohjaintuettuja, joten simulaatioiden nopeutta saadaan lisättyä tietokoneen prosessointitehon ja ytimien lisäämisellä. Simulaatiosta voidaan nähdä myös esimerkiksi karkea esikatselu nesteen pinnasta, nopeusvektorit tai paineen muutokset. (SideFX 2022b.)

Flip object sekä Flip solver -nodella voidaan muuttaa veden ominaisuuksia halutulla tavalla. Flip solver -noden asetuksiin kuuluu mm. räiskyvyys, viskoosisuus ja törmäysobjektien kanssa käyttäytyminen. Näiden ja monien muiden parametrien avulla voidaan saada aikaiseksi mikä tahansa neste. Flip Solver sisältää kaksi metodia, joilla määritetään nesteen rakenteen nopeus. Ensimmäinen on FLIP (Splashy), joka saa aikaan roiskuvan ja energisen simulaation. Toinen on APIC (Swirly), joka toimii paremmin pyörteiseen ja viskoosisempaan materiaaliin. Flip Solver tarvitsee aina Flip Fluid -objektin, joka toimii nesteen lähteenä. Tämän noden ominaisuuksiin kuuluu mm. edellä mainittu partikkelien tiheys (Particle Separation), joka periytetään arvona usein geometrian muodostusvaiheeseen, että partikkelin visualisointi ja lopputulos näyttäisivät samalta. Aiemmin mainituilla substepeillä eli laskentakerroilla saadaan simulaatio käyttäytymään aidommin, jos sen liikkeeseen vaikuttaa useampi tekijä eikä simulaatio pysy toistonopeuden perässä. (SideFX 2022b.) Fluidien muotoa ja käyttäytymistä ohjaillaan usein VOP(Vector Operator)-nodella. Sillä saadaan muokattua geometriaa ja shadereita vektorien avulla (SideFX 2022h). Suuren aallon saa esimerkiksi aikaiseksi animoidulla nopeuskentällä, joka sijoitetaan fluid-tankin pintaan. Esimerkki kuvassa 12 on tehty ottaen mallia Carlos Parmentierin työstä (2020).



Kuva 12. Aallon synnyttäminen VOP-noden nopeuskentällä

Kaasu vaatii Smoke object- ja Pyro solver -noden. Näiden avulla voidaan säätää esimerkiksi tulen lämpötilaa, emittoituvaa savua, muotoa ja palamisnopeutta. Renderöitäviä attribuutteja ovat esimerkiksi density, heat, fuel ja temperature. (SideFX 2022c.) Tulen ja savun kanssa pätee sama periaate VOP-verkkojen käytössä kuin nesteissä; erona on, että muutokset tehdään savun lähdegeometriaan. Esimerkiksi räjähdyksen suuntaa, paineaaltoa ja siitä lähteviä juovia voidaan ohjata yksinkertaisilla VOP-verkolla muokatulla partikkelisimulaatioilla. Partikkeleille tai juoville annetaan fuel-node, kun ne on saatu käyttäytymään halutulla tavalla (kuva 13). Tämän jälkeen suositeltavin metodi on rasteroida savun lähdepisteet, että saadaan selkeämpi käsitys lähteen voimakkuudesta ja visuaalisesta ulkomuodosta. (SideFX 2022a).



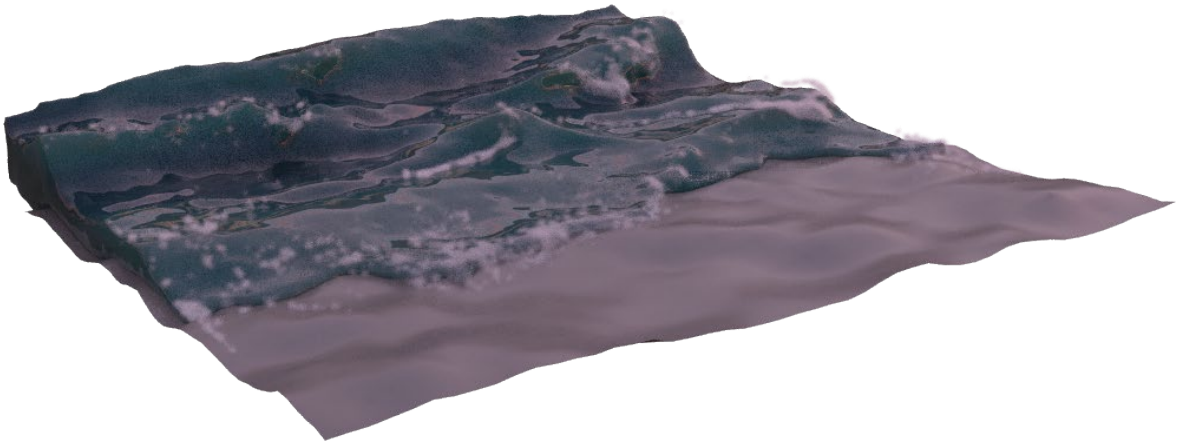
Kuva 13. Savujuovien geometrian luonnista simulaation polttoaineen rasterointiin

5.3 Renderöinti

Ennen renderöintiä nesteisiin muodostetaan pinta uloimmista simulaation partikkeleista Particle Fluid Surface -nodella. Sillä voidaan määrittää nestemassan resoluutio ja pisaroiden skaala (SideFX 2022d). Pinnanmuodostuksen jälkeenkin simulaation parametrit pysyvät nesteessä, kuten tiheys, jota voi käyttää hyödyksi materiaaleissa. Renderöinnin voi suoriuttaa Houdinin omilla renderöintimoottoreilla, kuten Mantralla tai Karmalla. Simulaatiot ovat kuitenkin raskaita renderöidä, joten nopein tapa saada tuloksia on usein kolmannen osapuolen renderöintimoottoreilla, esimerkiksi Redshiftillä, joka voi tukea useampaa näytönohjainta samanaikaisesti.

Realistisen nesteen tavoittamisessa, esimerkiksi meriveden rendauksessa otetaan huomioon sen simulaation käyttämiä parametreja, kuten kiihtyvyyttä, kaarevuutta sekä pyörteisyyttä. Näillä arvoilla saadaan synnytettyä partikkelisimulaatio vaahdolle ja kuohuvalle vedelle. (SideFX 2022b.) Meren materiaalia säädellään lähinnä aallokon korkeus- ja paksuuseroilla. Näillä ominaisuuksilla vesisimulaatioista saadaan hyvinkin realistisia (kuva 14). Kaikkien näiden parametrien lukeminen valmiiksi tallennetulta cache-tiedostolta on jo

hidasta, joten esimerkiksi pelimoottorilla ei ole mahdollista käsitellä dataa samanlaisella tarkkuudella reaaliajassa.



Kuva 14. Renderöity aallokko ja siitä syntyvä vaahto Mantra-renderöintimoottorilla

Tulen ja kaasujen renderöinnissä käytetään niille ominaisia attribuutteja: density, temperature, fuel ja heat. Näiden data voidaan siirtää myös muihin 3D-ohjelmiin käyttäen VDB-tiedostomuotoa (SideFX 2022e). Shaderit käyttävät attribuutteja esimerkiksi emission tai värin muokkaamiseen, millä voidaan saada esimerkiksi savupilvi näyttämään räjähdykseltä. Kuva 15 näyttää simulaation käyttäytymistä ohjaavien attribuuttien vaikuttavat alueet räjähdyksessä.

Kuva 15 Pääräjähdyksen emissioarvoon vaikuttava attribuutti vasemmalta oikealle Redshift-renderöintimoottorilla: density, temperature, fuel ja heat

6 CASE: Simulaation toteutus peleihin

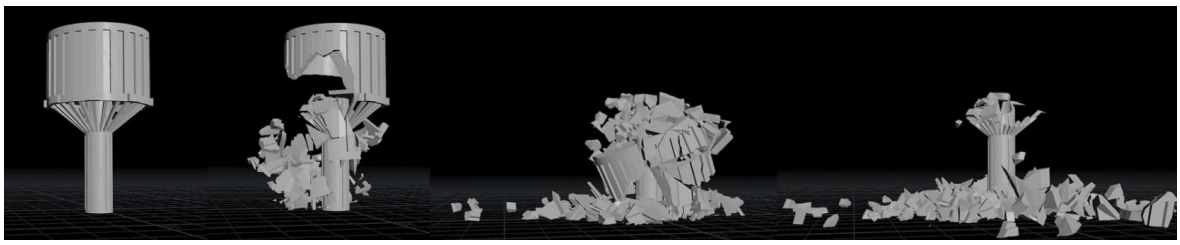
6.1 Tavoitteet

Työn tarkoitus on luoda Jyväskyläläisen Pixtell Oy:n Last Drop -Legend of Seppo -peliin kaksi välikohtausta. Ensimmäisessä vesitorni sortuu ja aiheuttaa tulvan metsään. Kohtaus perustuu Kangaslammella 2012 tapahtuneeseen vesitornin sortumiseen. Työssä otetaan huomioon ympäristö, mureneva vesitorni sekä itse vesi, joka valuu vesitornista metsään. Mallia on otettu tapahtuman ennen ja jälkeen kuvista, sekä Google Earthin ympäristön muodoista. Toteutuksessa käydään läpi simulaation luontia, sekä siihen vaikuttavia törmäysobjekteja, kuten ympäristöä ja vesitornia, Houdinin käyttöliittymän avulla. Työssä käydään myös läpi simulaation siirtämistä pelimoottoriin sekä siihen liittyviä haasteita.

Toinen simulaatio liittyy savuun. Sen tarkoituksena on luoda räjähdys kerrostalon huoneeseen, jonka seurauksena syntyy savua ja seiniä lentää irti. Työhön kuuluu valmiiksi mallinnetun talon tuhoaminen ja savusimulaation kehittäminen tuotavaksi pelimoottoriin. Simulaation on tarkoitus tapahtua pelatessa, joten suorituskyky ja tila on otettava huomioon. Talon tuhoutuminen on suunniteltu tapahtuvan siten, että yhdessä huoneessa tapahtuu räjähdys. Työssä otetaan huomioon ympäristö ja siihen lentävät talon seinämät sekä savun reagointi talon kanssa. Savu ohjataan pelimoottorin partikkelisimulaatiolla. Hiukkasien käyttäytymistä ohjataan Houdini-savusimulaation avulla.

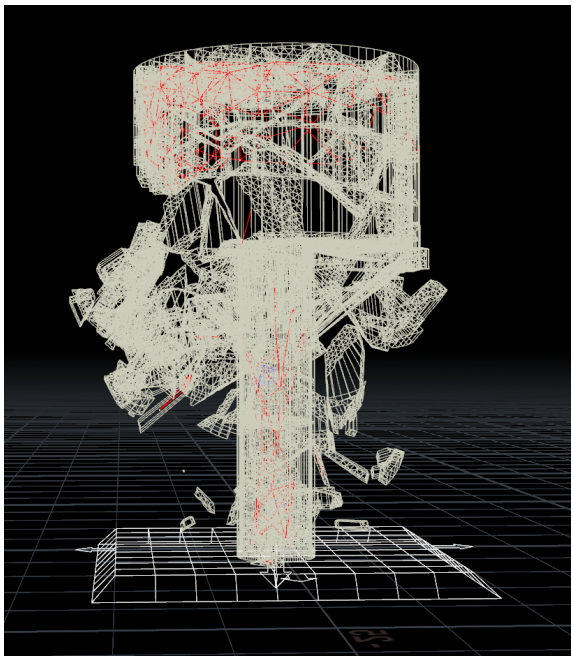
6.2 Vesisimulaatio

Vesitorni mallinnetaan Blenderillä referenssikuvia käyttäen ja tuodaan Houdiniin hajotettavaksi. Houdinin RBD Material Fracture -nodella vesitorni saadaan murenemaan kuin se olisi betonia. Kyseinen node antaa palasille myös sopivia liimasidoksia, ettei torni kaadu kaikkialta samaan aikaan. Hitaaksi suunniteltu murenemisen tyyli näkyy kuvassa 16.



Kuva 16. Vesitornin sortuminen

Mureneminen tehdään simulointiin tarkoitetussa DOP (Dynamic Operators) Network -nodessa BulletSolveria käyttäen, joka saa objektit käyttäytymään määritetyn fysiikan mukaan. Tässä liimasidoksia heikennetään kahdella massan omaavalla pallolla, jotka osuvat vesitorniin sen sisällä. Pallot poistetaan näkyvistä lopullisessa animaatiossa blast-nodea käyttäen. Tällä voidaan poistaa node-verkosta mitä vain. Kuva 17 näyttää punaisella liimasidokset, jotka pitävät murenevia palasia vielä yhdessä, mutta jotka heikentyvät vesitornin murentuessa. Vesitornin liimasidoksia voidaan muuttaa tarkemmin Glue constraints -noden asetuksilla tai koodilla. Mureneva vesitorni lasketaan valmiiksi simulaatioksi erilliseen cache-tiedostoon paremman prosessointinopeuden saavuttamiseksi, jonka jälkeen vesitornia käytetään törmäysobjektina tornista pursuavalle vesimassalle.

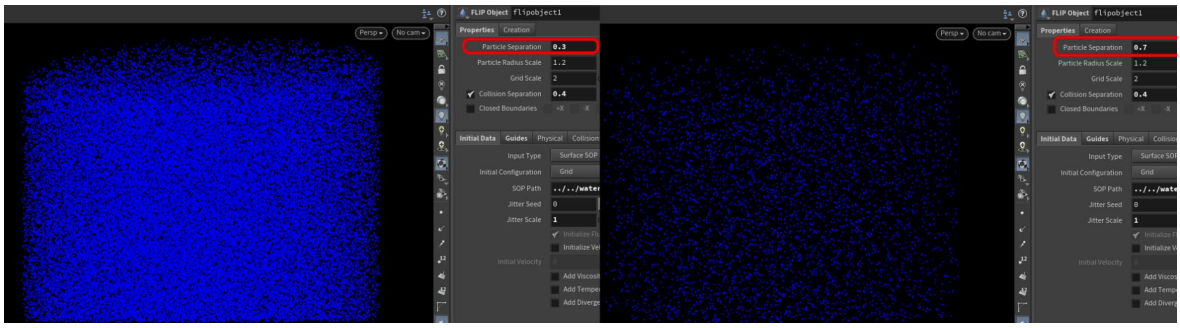


Kuva 17. Vesitornin liimasidokset

Nesteen muodostus

Nestesimulaatiossa määritetään, millä metodilla sortuva vesitornigeometria törmää nesteen kanssa. Se liitetään node-verkkoon Static object -nodella, jossa todetaan parhaaksi Ray Intersect mode. Tällä geometriasta saadaan mahdollisimman tarkka tiheysmalli. Tarkempi vaihtoehto olisi ollut muuttaa vesitorni VDB (Voxel Data Base) muotoon, mutta tässä tapauksessa nestepartikkelit katoavat niiden joutuessa geometrian sisälle. Vesitornin kanssa tämä tapahtuu lähes varmasti, koska murenevat palaset litistävät vesipartikkeleja toistensa väliin, jonka seurauksena vesimassaa ei jää paljoa jäljelle.

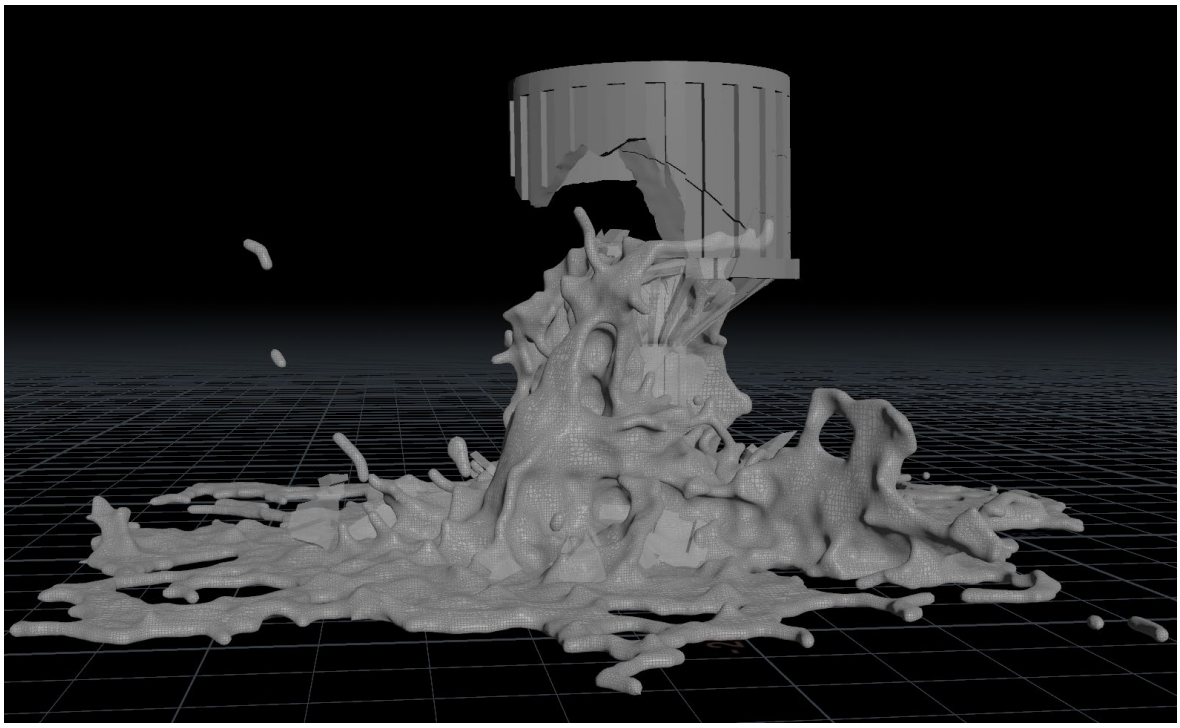
Neste saadaan vesitornin sisälle laittamalla sylinterin muotoinen objekti sen sisälle ja muuttamalla siitä FLIP-objekti. Tässä vaiheessa määritetään simulaation tarkkuus simuloitavien partikkelien määrää säätämällä eli muuttamalla niiden etäisyyttä toisistaan. Partikkelien määrään vaikuttaa myös objektin suuruus. Kuva 18 näyttää miten yksityiskohdat heikkenevät, kun partikkelien määrä vähenee.



Kuva 18. Particle separation -arvon suhde simulaation tarkkuuteen

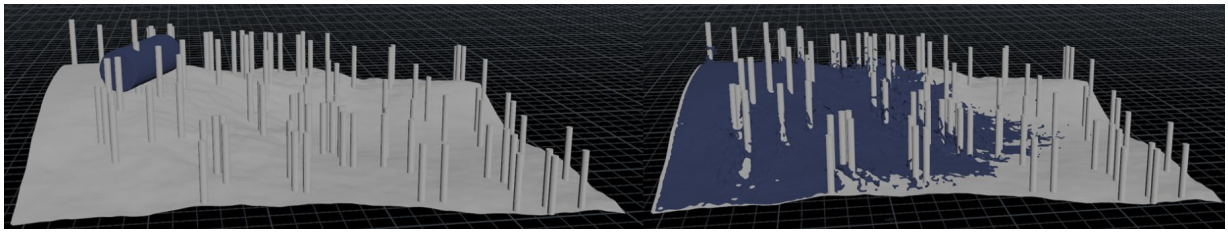
FLIP-solverilla määritetään nesteen käyttäytyminen ja koostumus. Houdini antaa kaksi vaihtoehtoa, miten nestettä käsitellään. Aiemmin mainittua APIC-tekniikkaa käytetään viskoosisemman materiaalin saavuttamiseen, mutta kun kyseessä on virtaava vesi, käytetään FLIP (Splashy) -nopeusmetodia. Laskentakertoja framen aikana joudutaan myös lisäämään monimutkaisen törmäysobjektin takia.

Kun vesi liikkuu halutulla tavalla, siitä muodostetaan geometria Particle Fluid Surface -nodella. Tämä node muodostaa pinnan uloimpien nestepartikkelien paikalle. Vokselien kokoa säätämällä saadaan muutettua pintaa myös sulavammaksi. Simulaatiosta muodostuu pinnan kanssa animaatio, joka näyttää vesimassalta, mikä näkyy kuvassa 19. Se tallennetaan cache-tiedostoon parempaa esikatselua ja Alembic-tiedoston tekemistä varten. Tiedoston koko riippuu simulaation monimutkaisuudesta ja koosta.



Kuva 19. Vesisimulaatiolle annettu pintageometria

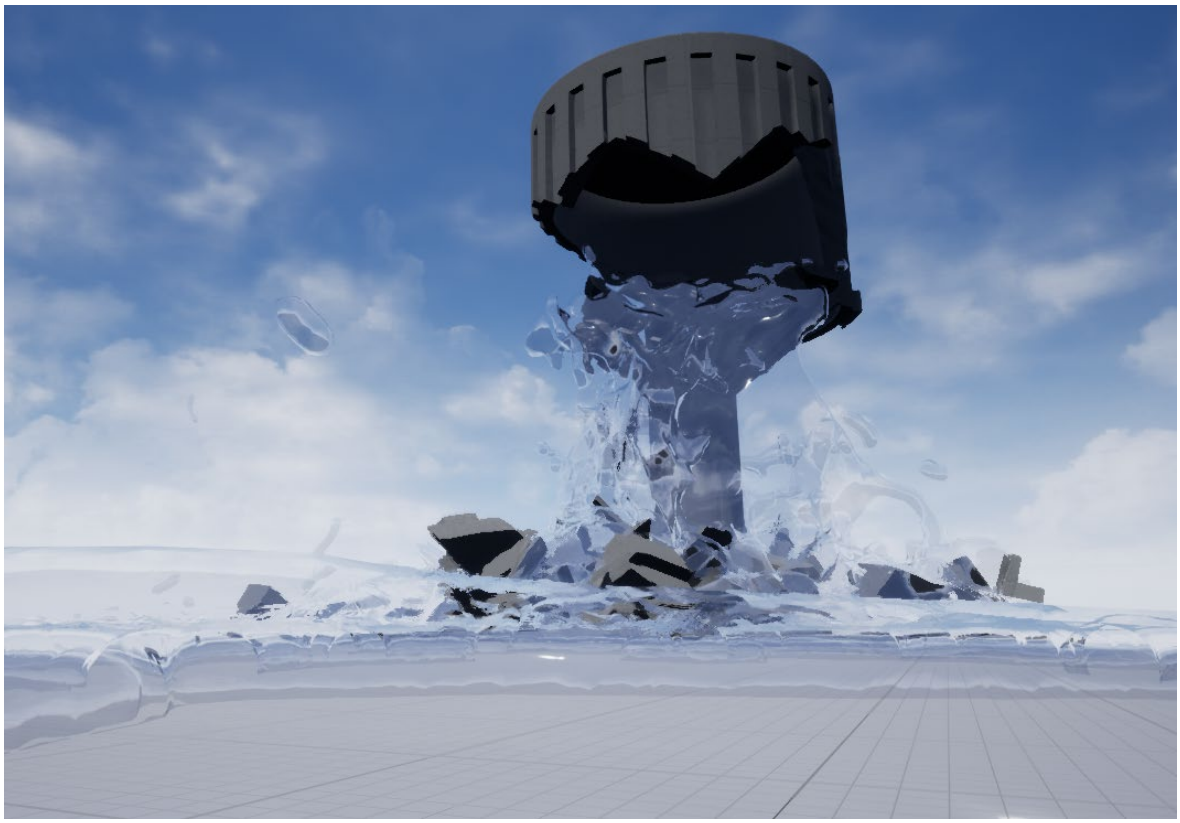
Ympäristö luodaan pelimoottoria varten, tässä tapauksessa Unreal Enginellä, joten tarvitaan vain oikeanlainen maanmuoto sekä sylinterit puiden paikoille. Vesitornin ympärystä on suhteellisen tasainen, joten normaali taso riittää vesitornin simulaatioon. Metsäkohtauksessa vesi virtaa rinnettä alas ja räiskyy puihin törmätessä. Puut ovat tässä tapauksessa hyvin ohuita maan geometriaan verrattuna, joten törmäysgeometrialle on annettava isompi resoluutio, että simulaatio ottaa puiden geometrian huomioon. Tämä tekee myös simulaation laskemisesta raskaampaa ja hitaampaa. Kuvasta 12 huomaa myös, että vedelle on täytynyt asettaa rajat, jotka poistavat simulaatiosta tason ulkopuolelle jäävät osat. Näin saadaan pidettyä prosessointinopeus minimissä.



Kuva 20. Rinne, jossa vesisimulaatio vuorovaikuttaa puiden kanssa

Pelimoottoriin tuonti

Unreal Engine -pelimoottoriin on mahdollista ladata Houdini Engine -lisäosa, joka pystyy lukemaan komplekseja Houdinilla tehtyjä assetteja. Koska projektin tämän osan on tarkoitus olla vain välikohtausanimaatio, simulaation koolla ei ole merkitystä. Pyrkimys oli siis vain tehdä assetteja animaatiota varten, joka näyttää tapahtuvan itse pelimaailmassa. Metsäkohtauksessakin puiden paikoille voi asetella Unreal Enginen assetteja manuaalisesti. Tällöin lopputulos on sama kuin että sylinterit olisi korvattu pelimoottorin omilla puuasseteilla Houdini Engine -lisäosaa käyttämällä. Koska vesitornin ei tarvinnut olla responsiivinen pelaajan kanssa, se voitiin siirtää pelimoottoriin FBX-animaationa. Tämä tallentaa muuttuvan 3D-mallin liikkeen vaiheet jokaisen framen kohdalla erikseen. Vesitorni ja vesimassa sijoitetaan samaan pisteeseen pelimoottorin koordinaatistossa, kuin ne olivat Houdinissa, jolloin ne näyttävät kuin ne reagoisivat toisiinsa. Vesimassa tuotiin Unreal Engineen Alembic-tiedostona. Mitä enemmän yksityiskohtia vesimassaan haluaa, sitä raskaampi Alembic-tiedostosta tulee. Tällöin pelimoottorin suorituskyvyllä on rajansa tiedoston käsittelemisessä. Vesitorni on assetti, jota ei vielä pystytty käyttämään pelin keskeneräisyydestä johtuen. Projektin tavoite oli täten luoda assetti, joka sisältää välikohtauksessa varmasti sisällytettävät asiat, jotka kuvataan sitten, kun pelin tarinan yksityiskohdat ovat tarkentuneet. Vesitornin lopputulos nähdään kuvassa 21.



Kuva 21. Vesitornin sortuminen ja vesisimulaatio Unreal Engine -pelimoottorissa

Muita menetelmiä ja huomioita

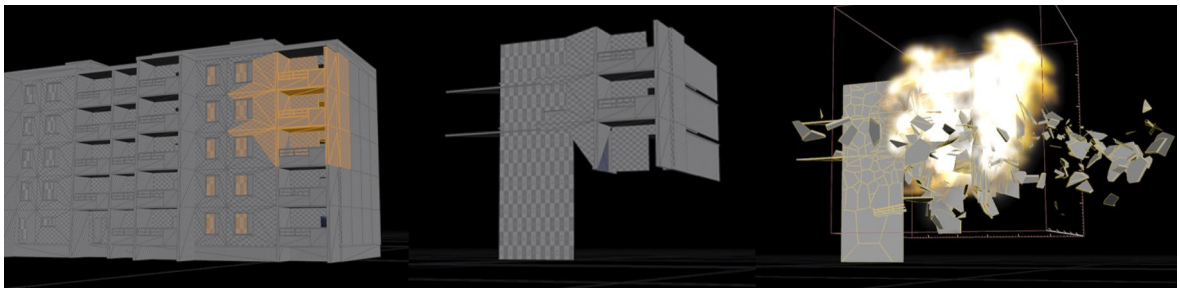
Peleissä on yleisempää käyttää Alembic-tiedostoja hyvin pienessä mittakaavassa, kuten esimerkiksi roiskeissa. Tällöin yksityiskohtien määrä ei ole suuri, koska animaatio on nopea ja pieni. Tämä säästää tiedoston koossa ja pelin suorituskyvyssä. Staattisia nesteitä, kuten jokia ja meriä kuvannetaan peleissä usein tekstuurien avulla. Niitä voidaan ohjailla myös Houdinia käyttäen, että saadaan halutulla tavalla kulkeva tekstuuri. Unreal Enginellä on myös oma Water Surface Mesh -työkalu, jolla voidaan helposti saada dynaamisia tasaisia vesimassoja, kuten meriä ja jokia. Pelimoottoreissa ei ole vielä selkeää, helppoa ja yksinkertaista ratkaisua responsiiviselle nesteelle.

Vesitornin tuhoutumissimulaatiosta saa yksilöllisemmän Houdinin avulla kuin silloin, jos sen tekisi pelkällä Unreal Enginellä. Vapautta antoi se, että simulaatiosta tuli peliin video, eikä toimiva responsiivinen simulaatio pelin aikana. Työtä tehdessä huomasi myös erilaiset tekniikat, joilla voisi pyrkiä samankaltaisiin, mutta rajoitetumpiin, lopputuloksiin. Näitä ovat esimerkiksi edellä mainitut tekstuurit tai Unreal Engine -pelimoottorin omat dynaamiset partikkeliefektit.

Suurimpina ongelmina olivat tekniset rajoitteet, kuten hidas prosessointiteho, monimutkaiset asetukset pelimoottoriin siirrossa, simulaation viemä suuri muistin määrä, sekä simulaation kohdistaminen peliympäristöön. Unreal Engine vaatii juuri oikeat import-asetukset, jotta simulaatiosta tehty animoitu malli ja sen materiaalit näkyvät pelissä oikein. Koska simulaation Alembic-tiedostosta saattoi tulla jopa 60GB kokoinen tiedosto, työhön käytetty tietokone ei pystynyt pitämään pelimoottoria vakaana. Tämän seurauksena pelimoottori kaatui useaan kertaan, kunnes Alembic-tiedostosta saatiin riittävän hyvän näköinen videolle, mutta tarpeeksi pieni kooltaan pelimoottorin pyöritettäväksi. Ongelmana oli myös se, että kyseisellä Houdinin Indie lisenssillä ei voinut tehdä peliin muuta kuin valmiiksi laskelmoituja assetteja, eikä sellaisia, joita Unreal Engine voisi käyttää Houdini Enginen kanssa. Tästä syystä esimerkiksi puiden paikanpitäjiä ei voida määrätä suoraan Houdinilla vaan ne täytyy sijoittaa manuaalisesti.

6.3 Pyrosimulaatio

Räjähdyksen on määrä tapahtua vain yhdessä huoneessa, joten pelin suorituskyvyn kannalta animaatio kannattaa tehdä vain rajatussa osassa taloa. Itse talon on mallintanut Mikael Häggvist. Talo jaetaan samoilla menetelmillä paloiksi kuin vesitornikin. Tässä tapauksessa tuhoutuminen aiheutetaan samaa räjähdystä hyödyntäen, mistä myöhemmin tehdään savupartikkelisimulaatio Unreal Engine -pelimoottoria varten. Tuhoutuvalle kerrostalon osalle annetaan attribuutti, joka seuraa, onko räjähdys tarpeeksi lähellä talon osia, jolloin talon osat lähtevät lentämään räjähdyksen nopeusvektorin mukaan (kuva 22).

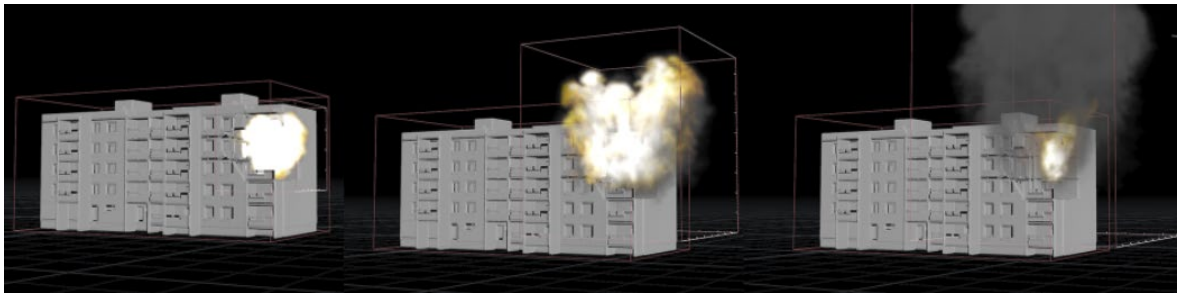


Kuva 22. Pyrosimulaatiota varten eristettävä talon osa

Tulen muodostus

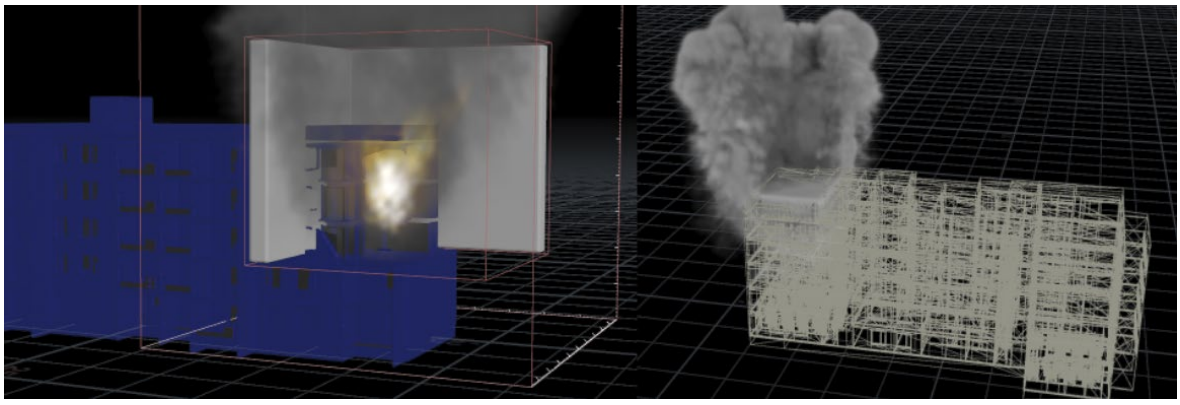
Tulen lähde on pallo, joka asetetaan rakennuksen sisälle. Pyrosimulaatiota ohjataan tekniikka osuudessa mainituilla arvoilla: temperature, density, velocity, fuel ja heat. Nämä imitoivat simulaatiossa palamista. Ne määrittävät esimerkiksi kuinka kirkas tai kuinka kauan

tuli palaa, tai sen, onko liekkiä ollenkaan. Tässä työssä polttoainetta on enemmän alkuvaiheessa, mutta se hyytyy loppuvaiheessa. Tämä saa aikaan suuremman räjähdyksen alkuun, jonka jälkeen savuaminen kuitenkin jatkuu, vaikka räjähdys on jo tapahtunut (kuva 23).



Kuva 23. Kerrostalon räjähdyksen luonne

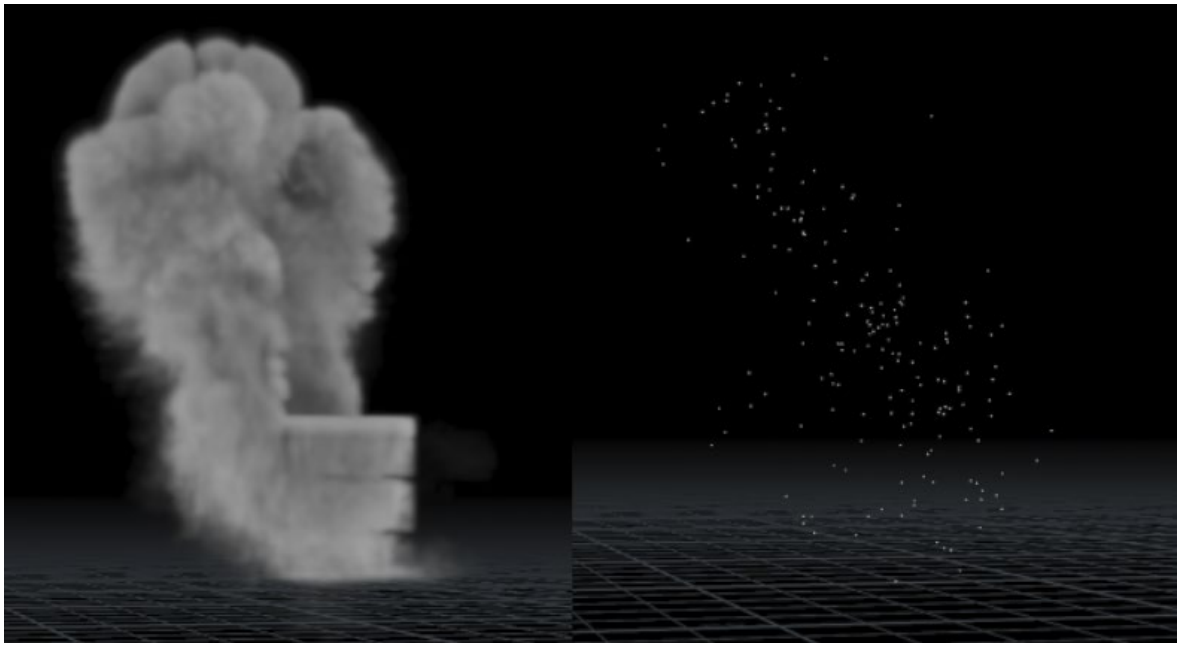
Räjähdyksen ei ole tarkoitus levitä talon sisälle, joten talosta täytyi tehdä VDB-objekti, että savu reagoisi siihen oikein. Koska talo on monimutkainen malli, simulaation laskentatehoa saatiin säästettyä tekemällä yksinkertainen väliseinä räjähtävän huoneen rajoihin (kuva 24). Tämä estää simulaation laskennan talon pikkutarkkojen sisäosien kanssa, mitä ei edes näkyisi.



Kuva 24. Talon ja savun törmäyksen kontrolloiminen suoritustehon säästämiseksi

Koska simulaatio on saatava pelin sisälle kevyeksi tiedostoksi, käytetään simulaation aiheuttamaa voimaa liikuttamaan partikkelisimulaatiota. Yksinkertainen partikkelisimulaatio liikkuu täten kuin savu Advect by Volumes -noden avulla. Koska partikkelien tilalla käytetään

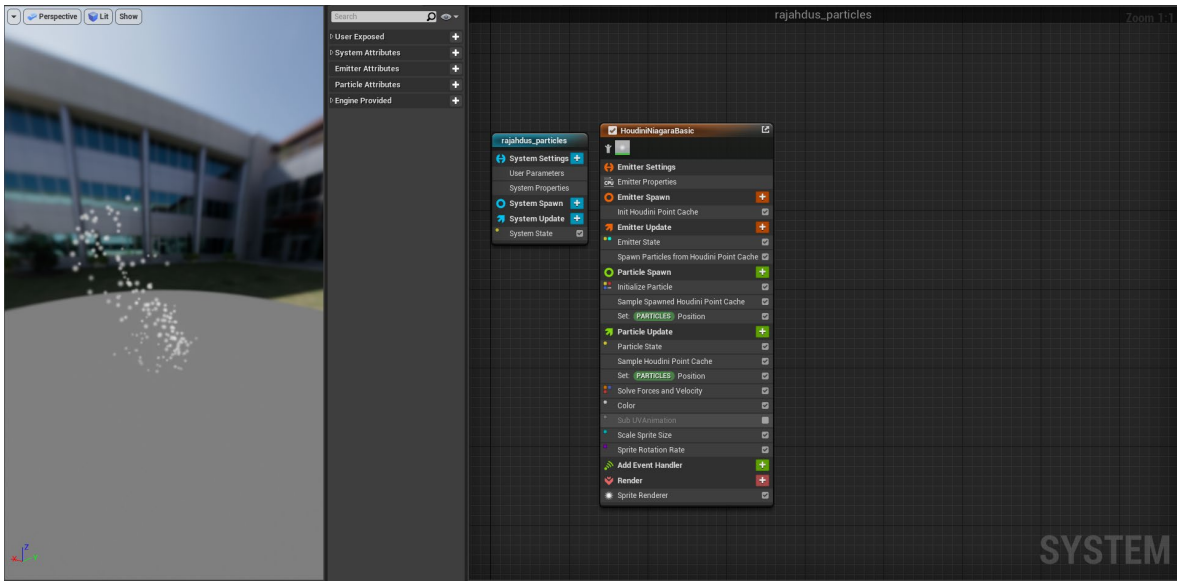
sprite-animaatiota, niiden konvektioasetuksia täytyy muuttaa hieman vapaammaksi. Tarkasti savua seuraavat partikkelit tekevät kipinöille ominaista värähtelyä ilmassa, mikä ei näytä realistiselta isommassa mittakaavassa. Kuvassa 25 huomataan, että simulaatio menettää tässäkin tapauksessa paljon yksityiskohtia.



Kuva 25. Savusimulaatio muutetaan partikkelianimaatioksi

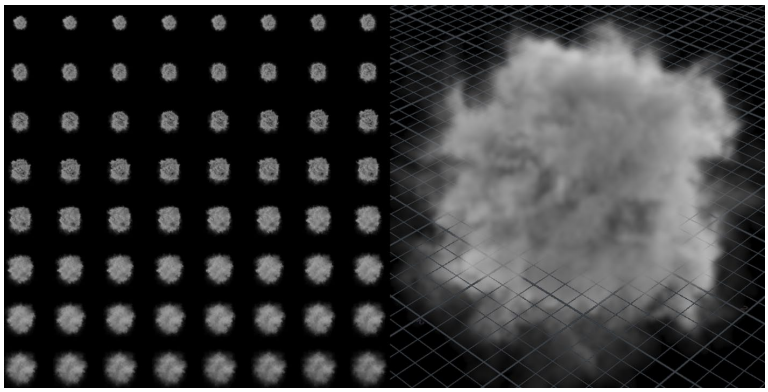
Pelimoottoriin tuonti

Talon räjähdys tuotiin samalla tavoin Unreal Engine -pelimoottoriin kuin vesitornikin. Samalla tuodaan terrain-pohja, johon lentävän talon palaset laskeutuvat. Partikkelit tuotiin point cache -tiedostona Houdinista Unreal Engine -pelimoottoriin käyttäen Niagara-lisäosaa. Tämän avulla partikkelisimulaation mukana saadaan tuotua arvoja, jotka on määritetty Houdinissa, kuten ikä, väri ja koko. Näitä arvoja voidaan muokata ja animoida Niagaran editorilla Unreal Engine -pelimoottorissa (kuva 26).

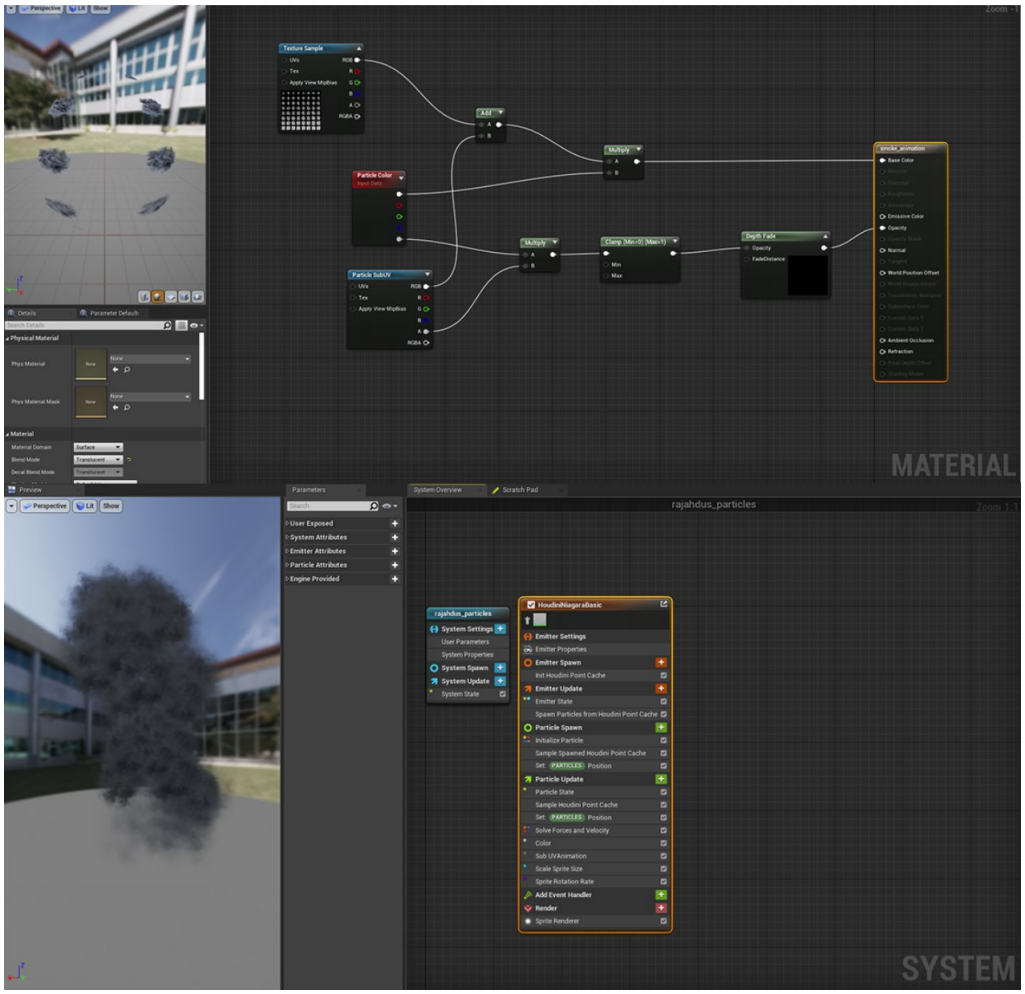


Kuva 26. Unreal Engine -pelimoottorin Niagara-editoriin tuotu Houdini-partikkelisimulaatio

Partikkelit itsenään näyttävät vain kipinöiltä, joten savua täytyy parannella sprite-animaatiolla. Unreal Enigne -pelimoottorin omat savusprite-materiaalit olivat liian epärealistisia käyttökohteeseen. Houdinilla siis täytyi tehdä oma savua kuvaava sprite sheet. Tätä varten toteutettiin uusi savuräjähdys, josta muodostettiin 80 framen kuvasarja (kuva 27). Partikkelit käyvät täten läpi 80 framen animaation elinkaarensa aikana pelimoottorissa. Lopputulos ei tule olemaan sama minkä Houdinin esikatselu näyttää, mutta tämä menetelmä tuottaa kevyemmän ja tarpeeksi aidon tuloksen videopelimaailmaan. Sprite-animaatiota värjätään, skaalataan ja pyöritetään niin, että siitä saadaan kuvan 28 näköinen savupylväs. Kuvan ylemmässä osassa näytetään sprite sheet laitettuna partikkelin materiaaliksi.



Kuva 27. Houdinin kuvaeditorilla toteutettu sprite sheet



Kuva 28. Sprite sheetin avulla tuotettu savuanimaatio Unreal Engine -pelimoottorin Niagara-lisäosassa

Lopputulos on pelimaailmaan sopivan realistinen ja kevyt prosessoitavaksi. Työn raskain osuus on FBX-animaatio, jota on myös kevennetty sillä, että vain räjähtävä osa taloa on animoitu. Loput talosta on vain malli, josta räjähtävä osa puuttuu. Partikkelien ja spritejen väriä olisi voinut muuttaa enemmän tulen väreihin, mutta se olisi tehnyt tässä tapauksessa tekstuureista epärealistiset, joten pelkkä savu tekee järkevimmän kokonaisuuden. Simulaation taloineen voi siirtää peliprojektiin assettina sille paikalle, jolle räjähtävä talo kuuluu (kuva 29).

Kuva 29. Pyrosimulaatio siirrettynä Unreal Engine -pelimoottoriin

Muita menetelmiä ja huomioita

Toinen tapa, jolla räjähdys voidaan saada pelimoottoriin, on sprite-tekstuurianimaatio, joka näyttäisi koko räjähdyksen. Simulaatio tuotaisiin pelimoottoriin kaksiulotteisena animoituna kuvasarjana. Animaation saa myös seuraamaan pelaajan lokaatiota, jolloin räjähdys näyttäisi joka suunnasta samanlaiselta. Tällöin räjähdys tapahtuu ikään kuin reaaliajassa, eikä kaksiulotteisuutta huomaa tarpeeksi kaukaa. Tämän työn puitteissa kyseinen tapa ei kuitenkaan voinut toimia, sillä simulaation sisällä on geometriaa, jolloin kaksiulotteisuus paljastuisi hyvinkin helposti.

Talon räjähdys halutaan tapahtuvan tietyssä kohdassa peliä. Houdiniin tuotiin tällöin pelissä käytettävä terrain, joka pilkottiin oikean muotoiseksi oikeaan kohtaan, että peliin siirrettävän räjähdyksen osat tippuvat oikeisiin paikkoihin. Räjähdyksen olisi voinut saada aiheutettua itse pelimoottorissa. Tämä kuitenkin vaatisi laskentatehoja peliltä ja se rajoittaisi räjähdyksen ulkoasua, jonka Houdinilla saa tehokkaasti sellaiseksi kuin halutaan. Savun osalta pelimoottoreilla ei ole vielä kevyttä tekniikkaa, jolla tilavuutta, esimerkiksi VDB-tiedostoa, saataisiin näytettyä kolmiulotteisena reaaliajassa.

7 Yhteenveto

Opinnäytetyössä tutkittiin tietokonegrafiikassa ja peleissä käytettäviä menetelmiä simulaatiopohjaisten erikoistehosteiden luomiseksi. Vaikka menetelmät perustuvat fyysisiin periaatteisiin ja pelkistettyihin numeerisen virtausdynamiikan kaavoihin, suuri osa simulaatioihin liittyvästä muokattavuudesta tapahtuu yksinkertaisilla käytännönläheisillä menetelmillä. Esimerkiksi räjähdyksen muodon saaminen halutuksi voi vaikuttaa ylitsepääsemättömän monimutkaiselta, mutta siinäkin tarvitsee vain ohjata räjähdyksen lähtöpisteitä yksinkertaisella geometrialla. Simulaatioihin liittyvässä työnkuvassa on tärkeää ymmärtää vaativa laskentateho. Raskaan simulaation työstössä on keskeistä osata ohjata ohjelman resursseja vain tarvittaviin simulaation osiin sekä käyttää vain sillä hetkellä tarvittavaa tarkkuutta ja yksityiskohtien määrää. Simulaatiota on haastavaa työstää reaaliajassa heti sellaiseksi kuin se tulee olemaan. Kuten teoriaosuudessakin todettiin, simulaatioiden suunnittelu on paljolti sisäisten ja ulkoisten voimien ja arvojen muuttamista, kunnes saadaan haluttu lopputulos.

Opinnäytetyö tehtiin Pixtell-yrityksen peliprojektia varten, jonka nimi tätä opinnäytetyötä kirjoitettaessa on Last Drop -Legend of Seppo. Pelimaailmaan oli tarkoitus luoda elokuvamaisia kohtauksia varten assetteja, jotka vaativat fluidien simulointia yksityiskohtaisella tavalla. Houdinia hyödynnettiin opinnäytetyön aikana sen erinomaisen kyvyn ansiosta luoda realistisia fyysisiä simulaatiota. Houdini osoittautui työlääksi, mutta erittäin hyödylliseksi työkaluksi pelimoottorin kanssa. Sillä oli mahdollista saada haastavista simulaatio-osuuksista toiveiden mukaisia. Ongelmia, joita täytyi korjata työn aikana, olivat sortuvien objektien tekstuurien siirtyvyys pelimoottoriin sekä simulaatiotiedostojen koon ja ominaisuuksien kokoonpanon optimointi. Assettien siirto pelimoottoriin tapahtui pääosin FBX- ja Alembic-tiedostoilla. Partikkelisimulaatio siirrettiin Houdinin point catche -tiedostolla Niagara-lisäosan avulla. Tutkimusta voisi jatkaa selvittämällä erilaisia pelimoottorin sisäisiä tapoja saada aikaan simulaatioita ja sitä, kuinka lähelle niillä päästään verrattuna Houdinilla saavutettuihin simulaatioihin. Simulaatioita voisi myös kokeilla tehdä enemmän tekstuuripainoitteisesti. Unreal Engine 5 versiossa aletaan jo kehittämään Niagaran avulla uusia fluidisimulaatiomahdollisuuksia.

Tutkimuksessa huomattiin, miten peli- ja elokuvateollisuus linkittyvät yhteen, ja miten molemmissa samantyyliset ammatit ovat eri rakenteisia. Työssä tutkittiin, miten elokuvateollisuuteen tarkoitetulla ohjelmalla saadaan rikastettua pelimoottorilla luotavia kokemuksia. Jatkotutkimusta voisi kehittää vastaavasti siitä, miten elokuvissa aletaan käyttää yhä useammin pelimoottoria sen nopean ja vaivattoman reaaliaikaisen renderöinnin ansiosta.

Lähteet

Braley, C. & Sandu, A. 2009. Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods. Viitattu 7.3.2021. Saatavissa https://www.fxguide.com/fxfeatured/the-science-of-fluid-sims/

Cline, D., Cardon, D. & Egbert, P. 2015. Fluid flow for the rest of us: Tutorial of the marker and cell method in computer graphics. Brigham Young Unviersity. Viitattu 7.3.2021. Saatavissa https://www.researchgate.net/publication/228964362_Fluid_flow_for_the_rest_of_us_Tutorial_of_the_marker_and_cell_method_in_computer_graphics

Crow, J. 2021. Dynamics and Simulation. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 567 – 571.

Cryengine. 2022 Culling Explained. Viitattu 7.3.2022. Saatavissa https://docs.cryengine.com/display/SDKDOC4/Culling+Explained

Electronic Arts. 2021. It Takes Two. Videopeli. Redwood City: Electronic Arts.

Hughes, C.J., Grzeszczuk, R., Sifakis, E., Kim, D., Kumar, S., Selle, A.P., Chhugani, J., Holliman, M. & Chen, Y. 2007. Physical Simulation for Animation and Visual Effects: Parallelization and Characterization for Chip Multiprocessors. 2.3, 3. Viitattu 31.3.2022. Saatavissa https://pages.cs.wisc.edu/~sifakis/papers/physbam_isca.pdf

Johnson, D. 2021a. Disciplines and Job Titles. Teoksessa Okun, J. & Zwerman, S. (toim.)The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 641 – 653.

Johnson, D. 2021b. Future of Gaming. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 660.

Johnson, D. 2021c. Game Engines and Real-Time Rendering. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 638 – 641.

Johnson, D. 2021d. How the Gaming Industry and Film/TV Industries are different. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 638.

Johnson, D. 2021e. Optimization and Runtime Budgets. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 655 – 656.

Johnson, D. 2021f. Pre-Rendered Cinematics vs. Real Time Visuals. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 654.

Kidd, R. 2021. 3D Products, Systems, and Software. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 628 – 636.

Riot Games. 2022. The Call | Season 2022 Cinematic - League of Legends (ft. 2WEI, Louis Leibfried, and Edda Hayes). Youtube-video. Viitattu 22.4.2022. Saatavilla https://www.youtube.com/watch?v=mDYqT0_9VR4

Parmentier, C. 2020. Pipewave flip simulation tutorial. Youtube-video. Viitattu 21.4.2022. Saatavissa https://www.youtube.com/watch?v=FqunirhLNwc

Seymore, M. 2011. The Science of Fluid Sims. Fxguide. Viitattu 7.3.2021. Saatavissa https://www.fxguide.com/fxfeatured/the-science-of-fluid-sims/

SideFX. 2022a. Creating explosions. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/pyro/explode.html

SideFX. 2022b. FLIP Solver 2.0 dynamics node. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/nodes/dop/flipsolver.html

SideFX. 2022c. Introduction to Pyro. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/pyro/intro.html

SideFX. 2022d. Particle Fluid Surface 2.0 geometry node. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/nodes/sop/particlefluidsurface.html

SideFX. 2022e. Shading and rendering. Viitattu 21.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/pyro/shading.html

SideFX. 2022f. Shelf Tools. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/shelf/index.html

SideFX. 2022g. SOP Guide dynamics node. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/nodes/dop/sopguide.html

SideFX. 2022h. VOP Force dynamics node. Viitattu 10.4.2022. Saatavissa https://www.sidefx.com/docs/houdini/nodes/dop/vopforce.html

Tessendorf, J. 2010. TEDxConejo - Jerry Tessendorf - 03/27/10. TEDx Talks. Youtube-video. Viitattu 7.3.2021. Saatavissa https://www.youtube.com/watch?v=XW2LtGtCbk0

Upstill, S. 1990. The RenderMan Companion AProgrammer's Guide to Realistic Computer Graphics. Boston, Massachusetts, USA: Addison-Wesley publishing company.

Zerouni, C. 2021a. Particles. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 571 – 575.

Zerouni, C. 2021b. Rigid-Body Dynamics. Teoksessa Okun, J. & Zwerman, S. (toim.) The VES handbook of visual effects: Industry Standard VFX Practices and Procedures 3rd Edition. New York: Routledge, 576 - 578.

Rare. 2018. Sea of Thieves. Videopeli. Twycross: Rare.

Zhanpeng, H., Guanhong, G., & Liang, H. 2014. Physically-based smoke simulation for computer graphics: a survey. Multimedia Tools and Applications, 74(18), 7573–7577. Viitattu 23.2.2022. Saatavissa https://doi.org/10.1007/s11042-014-1992-4

Zhao, X. 2020. General Pipeline and Lighting Study: Video Games vs. VFX Commercial. 80 Level. Viitattu 05.04.2022. Saatavissa https://80.lv/articles/general-pipeline-and-lighting-study-video-games-vs-vfx-commercial/