TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Jaagup Kuhi

Volumeetriliste pilvede renderdamine

Bakalaureusetöö (9 EAP)

Juhendaja: Jaanus Jaggo, MSc

Tartu 2018

Volumeetriliste pilvede renderdamine

Lühikokkuvõte:

Käesolev bakalaureusetöö annab ülevaate erinevatest volumeetriliste pilvede renderdamise tehnoloogiatest. Neist ühte uuritakse lähemalt ning implementeeritakse mängumootoris Unity. Töös kirjeldatakse nii pilvede renderdamiseks kasutatud kiirte marssimise algoritmi kui ka pilvede kuju moodustamist ja valguse arvutamist. Pilvede asukohta ja tüüpi kontrollib protseduuriliselt genereeritud ilma tekstuur. Samuti käsitletakse töös pilvede renderdamise optimeerimist. Töö tulemusena valmib demonstratiivne rakendus, mis renderdab volumeetrilisi pilvi ja nende välimust saab reaalajas parameetrite kaudu muuta.

Võtmesõnad:

Volumeetrilised pilved, Unity, kiirte marssimine, arvutigraafika, arvutimängud

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Volumetric cloud rendering

Abstract:

This bachelor thesis provides an overview of various volumetric cloud rendering technologies. One of which will be studied in detail and implemented in Unity game engine. The thesis describes volumetric cloud rendering algorithm, as well as formation of the cloud shapes and cloud light calculations. Cloud position and shape is controlled by procedurally generated weather texture. Thesis also describes the optimization of the cloud rendering. As a result of the thesis, a demonstration application is built, which renders volumetric clouds. The appearance of the clouds can be changed in real-time by changing the parameters.

Keywords:

Volumetric clouds, Unity, ray marching, computer graphics, computer games

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

1 Sissejuhatus 5

2 Pilvede renderdamise tehnoloogiad 6

2.1 Reaalsed pilved 6
2.2 Varasemad pilvede renderdamise tehnoloogiad 7
2.3 Mängus Reset 8
2.4 Nubis 8
2.5 Frostbite 9
2.6 Mängus Witness 9
2.7 Tehnoloogiate võrdlus 10

3 Renderdamine 11

3.1 Volumeetriline renderdamine 11
3.2 Sfääriline atmosfäär 12
3.2.1 Pilvede sees ja peal 13

3.3 Mürad 14
3.3.1 Perlini müra 14

3.3.2 Worley müra 14

3.3.3 Curl 15

3.4 Pilve tekstuurid 15
3.5 Ilma tekstuur 17
3.6 Pilve kuju 17
3.7 Kahemõõtmelised kõrged pilved 20
3.8 Tuul 20

4 Valgustamine 21

4.1 Ülevaade valgustuse mudelist 21
4.2 Otsene valgus 22
4.2.1 Valguse marssimine koonuses 22

4.2.2 Beeri seadus 24

4.2.3 Henyey-Greenstein'i faasi funktsioon 25

4.2.4 Sise-hajumise funktsioon 26

4.3 Otsese valguse kombineerimine 28
4.4 Ümbritsev valgus 29
4.5 Valguse kombineerimine 29
4.6 Pilvedevahelised varjud 29
5 Implementatsioon 31
5.1 Unity 31
5.2 Varjutajate ülevaade 31
5.3 Pilve varjutajad 32
5.4 Ilma varjutajad 33
6 Optimeerimine 34
6.1 Stohhastiline sämplimine 34
6.2 Madalam resolutsioon 35
6.3 Varajane lõpetus 35
6.4 Suured sammud 36
6.5 Muud optimeeringud 36
7 Tulemus 38
8 Kokkuvõte 40
9 Viidatud kirjandus 41
Lisad 46
I. Terminid 46
II. Demorakendus 49
III. Lähtekood 50
IV. Litsents 51

1 Sissejuhatus

Pilved on arvutimängude välistseenides oluline osa. Pilved koos taevaga katavad suure osa stseenist ning neil on tähtis roll loo visuaalsel jutustamisel. Vastavalt mängu tüübile ja mängumaailma suurusele kasutatakse erinevaid pilvede renderdamise tehnoloogiaid. Näiteks võistlusliku mitmiktulistamise mängu Counter Strike: Global Offensive'i [1] juures on tähtis kõrge kaadrisagedus ning seal kasutatakse arvutuslikult võimalikult kiireid lahendusi. Samas kasutavad loopõhised seiklus- ja rollimängud, nagu The Witcher 3: Wild Hunt [2] või Horizon Dawn Zero [3], visuaalselt ilusamaid ja dünaamilisemaid tehnoloogiaid. Viimastel aastatel on videokaartide jõudlus oluliselt kasvanud ning see võimaldab rakendada keerulisemaid pilvede renderdamise tehnoloogiaid.

Käesoleva bakalaureusetöö eesmärk on implementeerida realistlikud volumeetrilised pilved mängumootoris Unity, pilvesüsteemi Nubis [4] põhjal. Pilvede asukoha ja tüübi taevas määrab protseduuriliselt genereeritud ilma tekstuur. Implementeerimisel kasutatakse optimeerimiseks stohhastilist sämplimist (*stochastic sampling*) [5], mis hajutab pilvede renderdamise üle mitme kaadri ja võimaldab teha sarnase tulemuse saamiseks vähem sämpleid. Töö raames valmib ka demonstratiivne rakendus, kus on võimalik muuta pilvede välimust mõjutavaid parameetreid ning näha kuidas pilved muutuvad.

Peatükis 2 antakse ülevaade pilvedest päris maailmas, samuti vaadeldakse nii varasemaid pilvede renderdamise tehnoloogiaid kui ka tänapäevaseid volumeetriliste pilvede renderdamise tehnoloogiaid. Peatükis 3 kirjeldatakse volumeetriliste pilvede renderdamise algoritmi, erinevaid mürafunktsioone ja pilvede kuju määramist. Peatükis 4 kirjutatakse täpsemalt valguse arvutamisest. Peatükis 5 kajastatakse loodud rakenduse implementatsiooni Unity's. Peatükis 6 kirjeldatakse algoritmi optimeerimist. Lõpuks peatükis 7 võrreldakse loodud rakenduse jõudlust ning pildid hinnatakse loodud rakenduse visuaalset kvaliteeti.

Töö lisades on toodud kasutatud terminid (Lisa I), pilvede demonstratiivne rakendus (Lisa II) ja lähtekood (Lisa III).

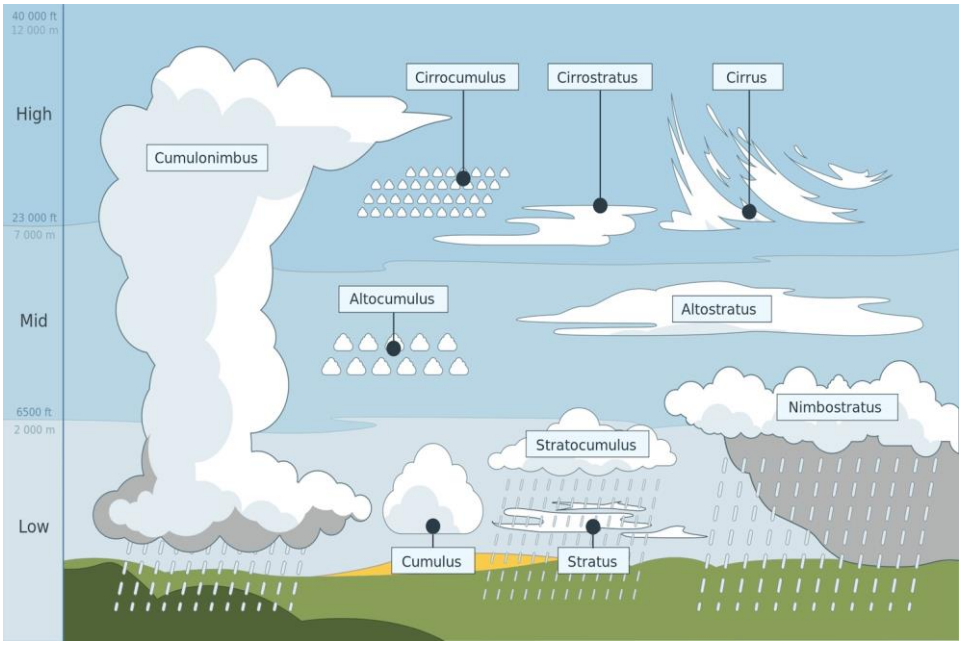
2 Pilvede renderdamise tehnoloogiad

Selles peatükis antakse ülevaade pilvedest päris maailmas ning millised füüsikalised tegurid neid mõjutavad. Samuti vaadeldakse erinevaid pilvede renderdamise tehnoloogiaid arvutigraafikas, võrreldakse neid omavahel ja valitakse neist sobivam, mida implementeerida mängumootoris Unity.

2.1 Reaalsed pilved

Jüri Kamenik [6] kirjutab, et päike soojendab maapinda, mis omakorda soojendab ümbritsevat õhku. Soe õhk mahutab rohkem veeauru kui jahe õhk, samuti on see hõredam ja tõuseb seetõttu kõrgemale. Kuna kõrgel on soojusvahetus väike, siis kulub õhuosakeste siseenergia paisumisele ehk protsess on adiabaatiline. Selle tulemusena langeb veeauru temperatuur ning mahutavus õhus, veeaur hakkab kondenseeruma väikesteks veetilkadeks ning moodustuvad pilved [6].

Artiklis „Pilvede klassifitseerimine“ [8] kirjutatakse, et pilvi klassifitseeritakse moodustumise, välimuse ja kõrguse alusel. Kõrguse järgi jaotatakse pilvi kolmeks: madalad, keskmised ja kõrged. Joonisel 1 on näidatud levinud pilvetüüpe, mis on klassifitseeritud kõrguse järgi. Välimuse järgi jaotatakse pilvi kiulisteks (*cirriform*), kihilisteks (*stratiform*) ja rünklikuteks ehk konvektiivseteks (*cumuliform*). Kiulised ja kihilised pilved on õhemad kui rünklikud pilved, mis võivad ulatuda kõrgele. Moodustumise klasse on neli: alumised, keskmised, kõrged ja vertikaalse arenguga pilved. Klassifitseerimisel tuleb meeles pidada,

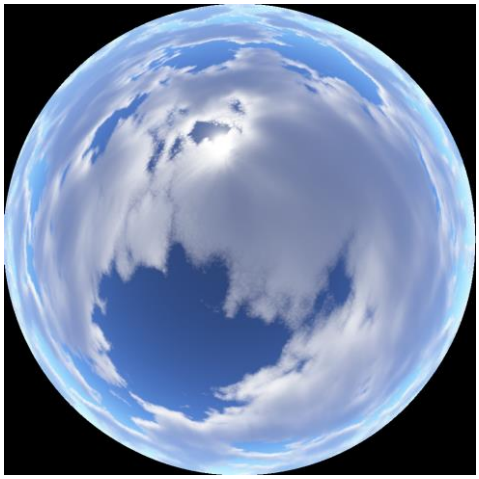


Joonis 1: Pilve tüübid kõrguse järgi [7].

et tegu on kokkuleppega ning mõned pilved võivad kuuluda mitmesse klassi korraga. Näiteks kihtrünkpilved (*stratocumulus*) kuuluvad välimuse järgi nii kihtpilvedesse kui ka rünkpilvedesse [8].

Käesolevas töös klassifitseerime pilvi peamiselt kõrguse järgi, sest selle põhjal saab kõige paremini renderdamist lihtsustada. Näiteks õhukeste kõrgede pilvede paksus on 200-400 meetrit, mis on nende kõrgusega (6-8 kilomeetrit) võrreldes väike [9]. Seetõttu arvutigraafikas võib neid vaadelda kui lamedaid kahemõõtmelisi pilvi. Samas madala kihi pilvi tuleks kujutada volumeetrilistena, sest nad on vaatlejale lähemal ja näha on pilvede ruumilist kuju. Töös renderdatakse volumeetriliselt madala klassi kolme pilve tüüpi: kihtpilved (*stratus*), kihtrünkpilved ja rünkpilved (*cumulus*). Kahemõõtmeliste kõrge pilvede klassist renderdatakse kiudpilvi (*cirrus*).

2.2 Varasemad pilvede renderdamise tehnoloogiad



Joonis 2: Panoraamne taevakupli tekstuur [10].

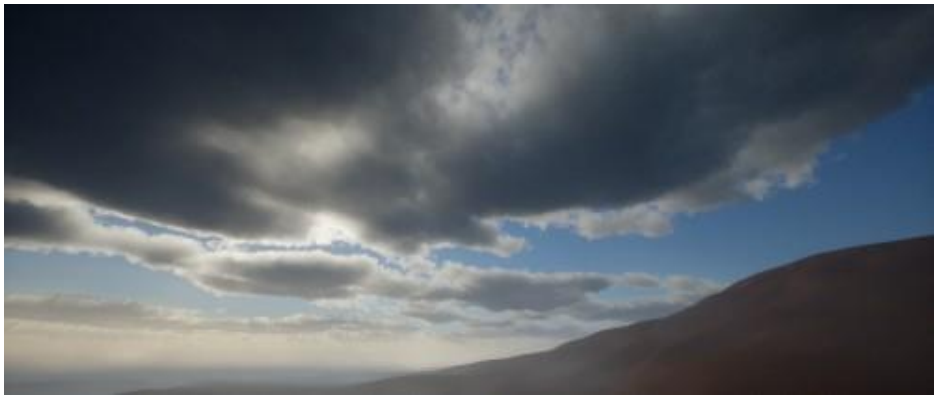
Pilvede renderdamiseks on loodud mitmeid erinevaid lahendusi. Üks levinud viisidest on lisada pilved panoraamsele taevakupli tekstuurile (kujutatud joonisel 2), mis projekteeritakse taevasse. Tavaliselt on arvutimängududes mängija alati taevakupli keskel, ehk taevakuppel liigub mängijaga kaasa. See lahendus annab realistliku välimusega pilved, mille arvutamine on kiire. Rakendustesse, kus on vaja dünaamilist ilma või öö ja päeva vaheldumist, see tehnika ei sobi, sest tegemist on staatilise pildiga. Panoraamse taevakupliga pilvi kasutatakse mängus PUBG [11].

Teine viis on lisada taevasse kahemõõtmeliste pilvede tekstuurid. See lahendus sobib kõrgetele õhukestele pilvedele, kuid ei sobi pilvedele, mis on vaatlejale lähedal, sest on näha, et pilved on lamedad. Kahemõõtmelisi liikuvaid pilvi kasutatakse mängus Holdfast: Nations At War [12].

2.3 Mängus Reset

Reset [13] on arenduses olev esimeses vaates pusle mäng, kus on kasutatud volumeetrilisi pilvi. Seal on atmosfäär jagatud kolmeks osaks: ülemine, keskmine ja alumine atmosfääri kiht. [14]. Ülemises atmosfääri kihis pole ilmastiku nähtusi ega pilvi, taeva väljanägemist mõjutab seal ainult päikese suund.

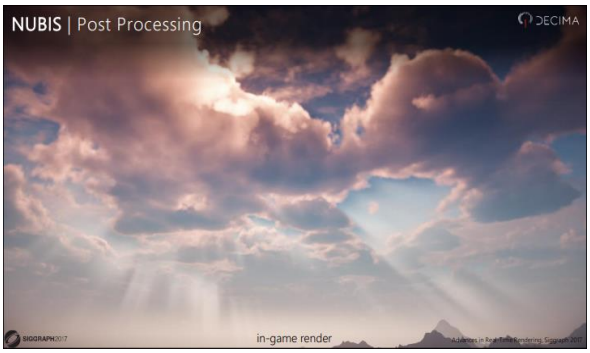
Keskmises kihis on volumeetrilised pilved, mida on kujutatud joonisel 3. Pilvede kuju leitakse kasutades mitut Perlini müra [14] ja teisi matemaatilisi funktsioone, mille väärtused on salvestatud 3D-tekstuuri. Pilvede värvi arvutamiseks leitakse pilve tihedus päikese suunas ning pilve tihedus otse ülesse. Päikese suunas leitud tiheduse põhjal arvutatakse pilvedele otse peegelduva valguse värv, otse ülesse arvutatud tiheduse põhjal lisatakse atmosfäärist peegelduva kiirguse (*radience*) värv. Madalamas kihis simuleeritakse ka teisi ilmastiku nähtuseid, nagu udu.



Joonis 3: Pilved mängust Reset [14].

2.4 Nubis

Pilvedesüsteem Nubis [4] on loodud Guerilla Gamesi poolt ja see on implementeeritud mängumootoris Decima. Neid pilvi kasutatakse mängus Horizon Dawn Zero [15], mis on saadaval mängukonsoolil PlayStation 4. Pilvesüsteemis Nubis on kolm pilvetüüpi: kihtpilved, kihtrünkpilved ja rünkpilved. Need pilved on volumeetrilised ja nad on realiseeritud kahe 3D-tekstuuri ja kahe 2D-tekstuuri abil. Volumeetrilised pilved on renderdatud kiirte marssimise abil. Kõrge kihi pilvede renderdamiseks kasutavad nad aga kahemõõtmelisi



Joonis 4: Nubis pilvede renderdus [4].

tekstuure kiirte marssimise asemel. Pilvede tüübi, asukoha ja vihmarohkuse määrab ilmasüsteemi poolt genereeritud ilma tekstuur, milles on info vastavate ilma andmete kohta. Joonisel 4 on näidatud pilvi Nubis pilvesüsteemist.

Pilvede liikumise animeerimiseks tuule mõjul liigutatakse 3D-tekstuure. Tuule suund on madalatel volumeetrilistel pilvel ja kõrgetel kahemõõtmelistel pilvedel erinev.

2.5 Frostbite

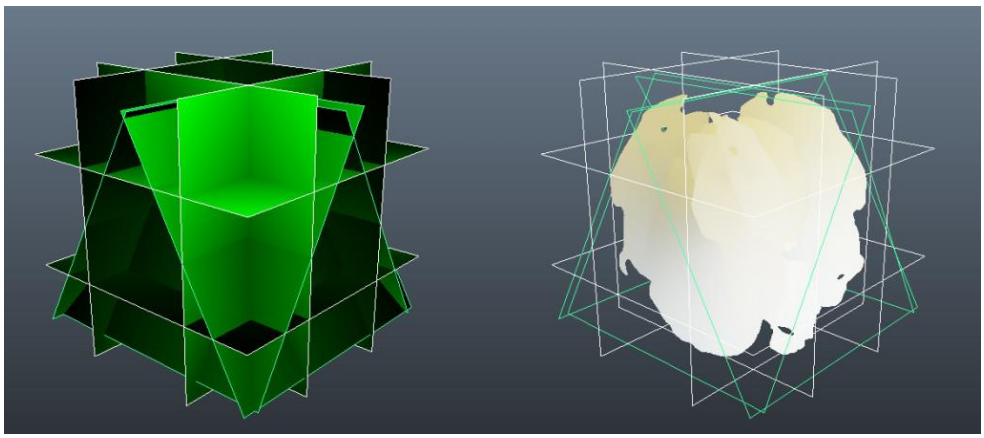
Esimese põlvkonna Frostbite mängumootor loodi mängule Battlefield Bad Company. Nüüdseks on Frostbite mängumootor arenenud firma EA üheks põhiliseks tööriistaks [16].

Frostbite mängumootori taeva renderdamine sisaldab volumeetriliste pilvede implementatsiooni [17], mis põhineb Nubis pilvesüsteemil. Joonisel 5 on pilved Frostbite mootorist. Sarnaselt Nubise pilvedele kasutab Frostbite samuti kahte 3D-tekstuuri ja renderdamisel kiirte marssimist. Erinevalt Nubis süsteemist on Frostbite mängumootoris lisaks ilmatekstuurile kasutusel ka pilve tüübi tekstuuri, mis määrab täpsemalt pilve kõrguse taevas [17].

Joonis 5: Pilved Frostbite mootoris [17].

2.6 Mängus Witness

The Witness [18] on avatud maailmaga seiklusmäng. Witnessi pilved [19] on loodud lõikuvate geomeetriate abil, kujutatud joonisel 6, mis grupeeritakse kokku suurtemateks pilvedeks. Pilved renderdatakse kahes möödumises (*pass*), kus esimene kasutab alfa kattuvumist (*alpha to coverage*) ning teine alfa sujutamist (*alpha blending*). Alfa kattumine



Joonis 6: Lõikuvad geomeetriad [19].

on tehnoloogia, mis on loodud riistvaralise MSAA sakitõrje (*anti-aliasing*) parandamiseks kohtades, kus sakid esinevad mujal kui geomeetria servades [20]. Näiteks muru sakiliste äärte eemaldamiseks olukorras, kus nelinurksele geomeetriale (*quad*) on renderdatud ainult muru tekstuuri läbipaistmatud pikslid. Seda renderdamise tehnoloogiat nimetatakse ka alfa testimiseks (*alpha-testing*) ning kombineerides alfa kattuvusega saab määrata, millistele pikslitele sakitõrje rakendub. Esimene möödumine tehakse eest tahapoole, ehk pilve osad, mis on kaamerale lähemal, renderdatakse esimesena. Teises möödumises tehakse vastupidi, tagant ettepoole. The Witnessis loodud pilved on stiliseeritud, mida on näha joonisel 7. Disaini aluseks on valitud maalid pilvedest, mitte päris pilvede fotod.

Joonis 7: Pilt pilvedest mängus Witness [19].

2.7 Tehnoloogiate võrdlus

Eelnevalt vaadeldud volumeetriliste pilvede renderdamise tehnoloogiatest kolmes kasutatakse 3D-tekstuure ja kiirte marssimist ning ühes kasutatakse lõikuvaid geomeetriaid. Kiirte marssimisel põhinevad tehnoloogiad võimaldavad kõige täpsemalt simuleerida päris pilvede füüsikalisi omadusi. Erinevalt teistest kirjeldatud volumeetriliste pilvede renderdamise tehnoloogiatest on mängus Witness võetud pilvede disaini aluseks pilvede maalid. Seetõttu on need rohkem stiliseeritud ja ei sobi selle töö eesmärgiga implementeerida realistlikud pilved.

Pilvesüsteem Nubis ja Frostbite mängumootori pilved kasutavad lisaks 3D tekstuuridele ka ilma tekstuuri, mis määrab pilvede asukoha ja tüübi. Mängus Reset sõltuvad pilved vaid 3D-tekstuuri väärtustest, mida on keerulisem kontrollida, kui kahemõõtmelist ilma tekstuuri. Frostbite ja Nubis tehnoloogiad on üksteisele väga sarnased, kuid pilvesüsteemist Nubis on kättesaadaval inforikkad allikad, näiteks artikkel raamatus GPU Pro 7 [21] ja esitluse slaidid SIGGRAPH konverentsilt aastast 2015 [15] ja 2017 [4]. Seetõttu on käesolevas töös valitud Unity’s implementeerimiseks pilvesüsteem Nubis.

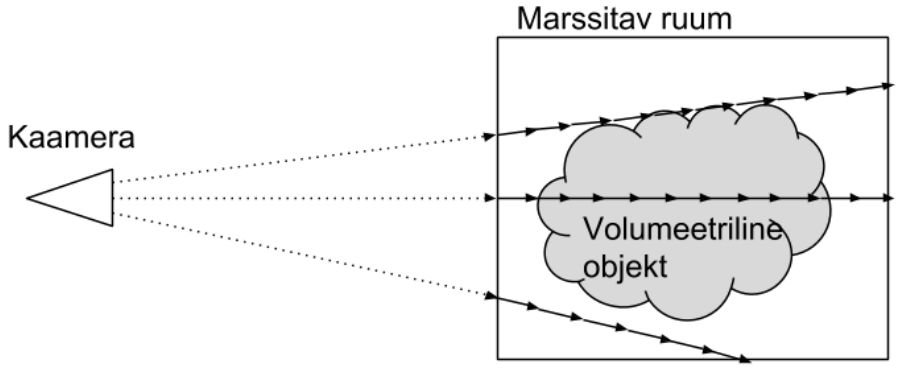
3 Renderdamine

Käesolevas peatüki alguses antakse ülevaade volumeetrilisest renderdamisest arvutigraafikas ja hiljem täpsustatakse, kuidas seda rakendada pilvede renderdamiseks. Peatükis kirjeldatakse atmosfääri kujutamist ja võrreldakse pilvede renderdamise erinevusi atmosfääri sees ja peal. Samuti antakse ülevaade pilvede loomisel kasutatud müradest ja tekstuuridest, kirjeldatakse üldiseid volumeetriliste pilvede kuju moodustamise etappe ning ka kõrgete pilvede renderdamist. Lõpuks vaadeldakse, kuidas mõjutab tuul pilvi ja kuidas seda arvutigraafikas simuleerida.

3.1 Volumeetriline renderdamine

Tavapärase renderdamise käigus kujutatakse geomeetrilised objektid pindadena. Need pinnad rasteriseeritakse ehk jagatakse fragmentideks, mis vastavad üldjuhul pikslitele ekraanil. Igale fragmendile arvutatakse selle värv, kaugus kaamerast ja vajadusel muid andmeid. See renderdamise meetod sobib küll hästi tahkete kehade kujutamiseks, kuid ei võimalda minna objekti sisse. Erandiks on ainult läbipaistvad objektid, näiteks klaas, mis renderdatakse pärast tahket geomeetriat selle peale. Mitte tahkete, näiteks gaasiliste, ainete renderdamiseks traditsiooniline renderdamise viis hästi ei sobi, sest ei saa hästi määratleda aine piire ega tihedust.

Arvutigraafikas kasutatakse selliste volumeetriliste objektide renderdamiseks kiirte marssimist (*ray marching*), mida on näidatud joonisel 8. Kiirte marssimine toimub mingis piiritletud ruumis, selleks võib olla lihtne geomeetriline objekt, näiteks kuup, või terve kaamera tüvipüramiid (*viewing frustum*), ehk see osa stseenis, mis renderdatakse ekraanile. Ruumi sees on volumeetriline objekt, mida saab kujutada matemaatilise funktsiooniga, 3D-tekstuuridega või nende kombinatsiooniga. See ruum jaotatakse osadeks nii, et iga osa vastab ühele ekraanile renderdatavale pikslile. Iga osa renderdamiseks saadetakse ruumi



Joonis 8: Kiirte marssimine.

kiir, mida mööda saab seda ruumi konstantsete sammudega sämplida. Sämplimise igal sammul arvutatakse ruumi tihedus ja valgustatus. Saadud sämpli värv akumuleeritakse, võttes arvesse, kui sügaval volumeetrilises objektis see asub. Sämplite arvu suurendamisel muutub pildi kvaliteet paremaks, kuid arvutuslikult aeglasemaks.

Kuna pilvede näol on tegemist väikeste osakestega, mille piire on raske määrata, sobib nende renderdamiseks kõige paremini kiirte marssimine. Pilvi võib aga kujutada 3D-skalaar väljadena, kus igale ruumi punktile antakse väärtus, mis tähistab pilve tihedust antud punktis. Erinevalt tahketest objektidest liigub valgus sügavale pilvede sisse, kus ta põrkab ringi, kuni sumbub või väljub mingis teises suunas. Valguse teekonda pilves saab samuti kiirte marssimisega ligikaudselt simuleerida.

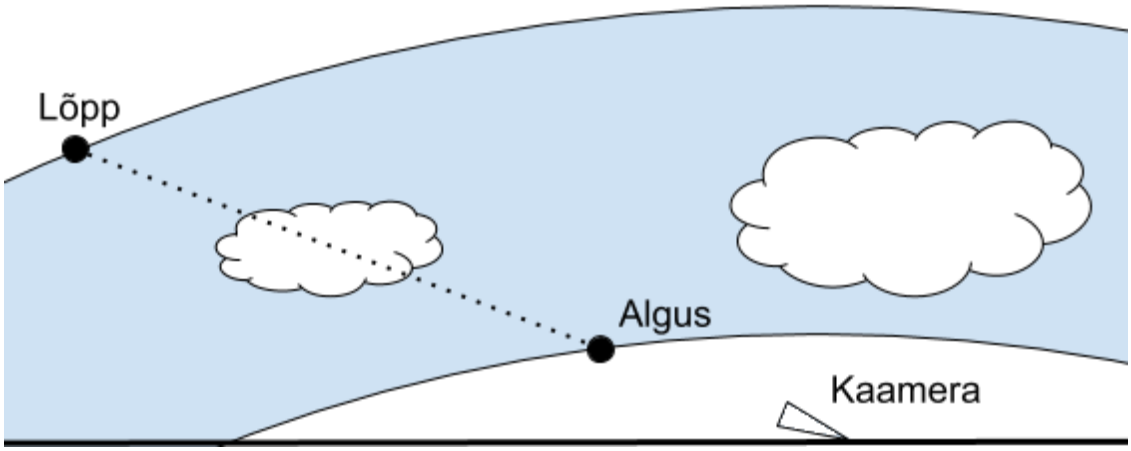
3.2 Sfääriline atmosfäär

Planeet Maa on ellipsoidi kujuga, mille raadius on ekvaatoril veidi suurem kui poolustel. Kuna see erinevus, 21 kilomeetrit, on aga kogu maa raadiuse, milleks on 6371 kilomeetrit [22], suhtes väga väike, siis arvutigraafika rakendustes võime selle lihtsustamiseks kasutada kera. Maa atmosfäär on maast kuni 120 kilomeetri kõrgusel ning seda võime samuti kerana vaadelda.

Kasutades päris maa raadiust kaovad pilved horisondi taha alles väga kaugel, mistõttu on ka pilvi näha kaugele. Sellega kaasneb probleem, kus pilved hakkavad horisondi lähedal korduma (vaata joonist 9), sest ilma tekstuur ei kata nii suurt osa taevast. Lisaks paikneb horisondil mitu kihti pilvi üksteise ees, neile tuleb teha tiheduse ja valguse arvutusi mitu korda, mis teeb renderdamise aeglasemaks. Sellepärast kasutatakse lihtsustamise eesmärgil väiksemat raadiust, näiteks käesolevas töös 400 kilomeetrit. Lihtsustuse tulemusel katab ilma tekstuur piisavalt suure osa taevast ära ning kordumist ei ole märgata, kuna pilved kaovad horisondi taha enne, kui kordumine nähtavaks muutub.

Joonis 9: Planeedi raadius 400km (vasakul), 6371km (paremal).

Kiirte marssimise algus- ja lõpp-punkt arvutatakse kiire ja kera lõikepunkti leidmise algoritmiga [23]. Alguspunkti kõrgus planeedi pinnalt on 1500 meetrit ning lõpp-punkti kõrgus on 5500 meetrit, ehk atmosfääri läbimõõt on 4000 meetrit. Atmosfääri sügavus on horisondi lähedal suurem kui otse kaamera kohal, näidatud joonisel 10. Seega sõltub ka kiirte marssimisel tehtavat sammude arv sellest, kui lähedal on kiir horisondile. Käesolevas töös tehakse horisondil kaks korda rohkem kiire marssimise samme, kui otse ülesse.



Joonis 10: Atmosfääri sügavus kujutatud horisondi lähedal.

3.2.1 Pilvede sees ja peal

Pilvi saab renderdada ka siis, kui kaamera on atmosfääri sees või selle peal. Kui kaamera on atmosfääri peal, siis kasutatakse kiirte marssimise alguspunktiks joonisel 10 kujutatud lõpp-punkti. Atmosfääri sügavust on pealtpoolt raske arvutada, sest kiir võib lõikuda välimise sfääriga, kuid mitte sisemisega. Selle pärast kasutatakse pealtpoolt renderdades fikseeritud marssimise kaugust, milleks on käesolevas töös 10 kilomeetrit.

Pilvede sees valitakse kiirte marssimise alguspunktiks kaamera asukoht. Lõpp-punkti võib leida kiire-sfääri lõikepunkti algoritmiga, nagu tehakse siis, kui renderdatakse pilvi alt. Sellise lähenemisega tekib probleem, kus pilvi marssitakse liiga hõredalt, sest planeedil raadiusega 400 kilomeetrit on horisondil pilvi näha umbes 35-45 kilomeetri kaugusele. Näiteks kui kasutada kiirte marssimisel 64 sammu, on ühe sammu pikkus 650 meetrit, mis on liiga suur. Seda saab lahendada tõstes sammude arvu, kuid sellega kaasneb ka arvutuskiiruse langus. Teine lahendus on lõpetada pilvede sees marssimine varem ära, määrates fikseeritud kaugus, mida on tehtud ka käesolevas töös. Selleks kauguseks on mõistlik valida kaamera renderdamise kaugus ehk vaate tüvikoonuse kaugeim külg (*far-plane*). Sellisel lahendusel on nähtavate pilvede kaugus programmeerijatel lihtsasti kontrollitav ning valitud kaugusele saab valida ka sobiva sammude arvu.

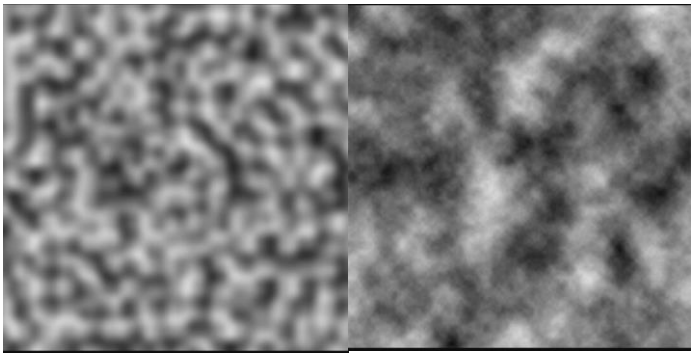
3.3 Mürad

Arvutigraafikas tähistab müra sellist matemaatilist funktsiooni, mis annab numbrilisele sisendile korrapäratu välimusega numbrilise väljundi. Müra sisendeid võib olla rohkem kui üks ning nende arv määrab müra dimensiooni, ehk mitme mõõtmelise ruumi jaoks seda saab arvutada. Näiteks tasandile katmiseks on vaja kahemõõtmelist müra. Müra funktsioon on deterministlik ehk annab samale sisendile alati sama väljundi. Mürasid kasutatakse arvutigraafikas kõige sagedamini protseduurilisel genereerimisel ja tekstuuride loomisel, kus on vaja loomulikke ja juhusliku välja nägemisega väärtused. Üheks levinud kasutusalaks on näiteks maastiku loomine.

3.3.1 Perlini müra

Perlini müra loodi Ken Perlini poolt aastal 1983, et luua protseduurilisi tekstuure filmile Tron [24]. Perlini müra on gradientmüra [25], ehk erinevalt väärtusmürast [26], kus kasutatakse juhulikke numbrilisi väärtuseid, kasutab see juhuslikke vektoreid. Joonisel 11 on kujutatud vasakul Perlini müra.

Et muuta müra välja nägemist huvitavamaks, liidetakse enamasti mitu erinevalt skaleeritud müra kokku. Saadud müra nimetatakse fraktaalseks müraks (*fractional Brownian Motion,* fBM), joonisel 11 paremal on kujutatud mitmest Perlini mürast koostatud fraktaalne müra.

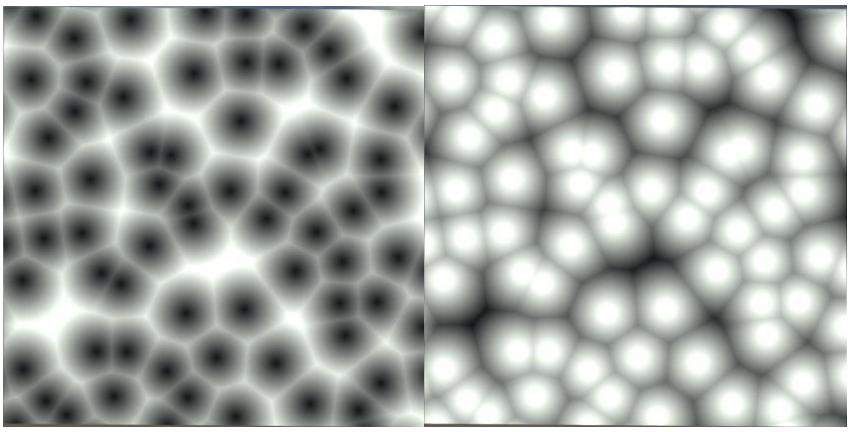


Joonis 11: Perlini müra (vasakul), Perlini müra fBM (paremal) [27].

3.3.2 Worley müra

Worley müra [28] on loodud aastal 1996 ja selle autor on Steven Worley. Müra genereerimine põhineb korrapäratult, kuid ühtlaste vahedega paigutatud funktsiooni punktidel (*feature points*). Worley müra väärtus on võrdne sisendi koordinaadi kaugusega lähimast funktsiooni punktist. Worley müraga saab luua Perlini mürast erineva mustriga

tekstuure, mida on kujutatud joonisel 12 [28]. Pildil vastab must värv müra funktsiooni väärtusele null ja valge värv vastab väärtusele üks. Paremal on sama müra ümberpööratult, mis tähendab, et müra funktsiooni väärtus on lahutatud ühest.

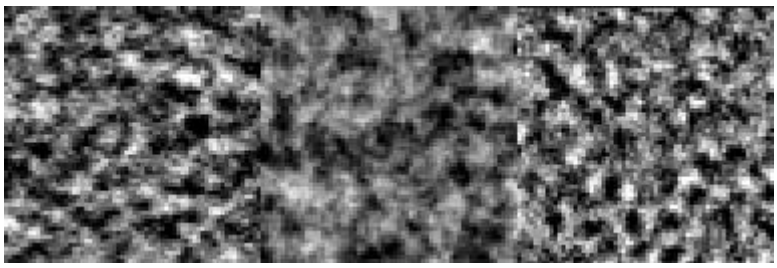


Joonis 12: Worley müra (vasakul) ümberpööratud Worley müra (paremal).

3.3.3 Curl

Curl müra [29] kasutatakse arvutigraafikas vedelike või gaaside turbulentse liikumise genereerimiseks, ilma et peaks kasutama simuleerimiseks keerulisi ja aeglaseid vedeliku liikumise võrrandeid. Erinevalt Perlini ja Worley mürast on n-dimensioonilise Curl müra väljund n-dimensiooniline vektor, mitte number. See vektor tähistab õhu või vedeliku osakese kiiruse vektorit. Müra genereerimiseks kasutatakse mitut Perlini müra sämplit, millele rakendatakse erinevaid matemaatilisi funktsioone [29].

Töös kasutatud Curl müra tekstuuri värvikanalid on toodud joonisel 13.

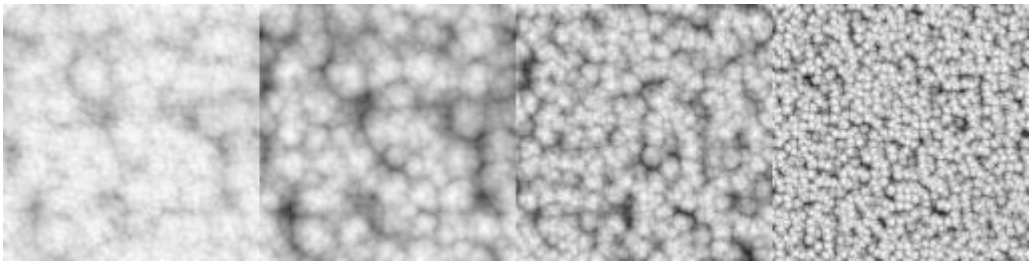


Joonis 13: Curl müra tekstuuri RGB kanalid.

3.4 Pilve tekstuurid

Pilvede loomiseks kasutatakse nelja tekstuuri, millest kaks on 3D-tekstuurid ja kaks 2D-tekstuurid. Kolmemõõtmelised tekstuurid on genereeritud C++'is kirjutatud programmiga TileableVolumeNoise [30] ja neid kasutatakse pilve kuju määramiseks. Esimene 2D-tekstuur on eelnevalt kirjeldatud Curl müra, mis on genereeritud kasutades koodi blogi postituse „Intro to Curl Noise" [31] slaididelt. Teine 2D-tekstuur on ilma tekstuur, mis genereeritakse protseduuriliselt programmi töö käigus.

Esimene 3D-tekstuur määrab pilve kuju. See tekstuur koosneb neljast kanalist ja on mõõtmetega 128^3 pikslit ning sisaldab erineva sagedusega müra. Läbilõikeid selle 3D tekstuuri kanalitest on kujutatud joonisel 14. Tekstuuri punases kanalis on Perlin-Worley müra, mis koosneb kolmest erineva sagedusega Perlini ja Worley mürast. Järgnevates värvikanalites on Worley mürad kasvavate sagedustega.

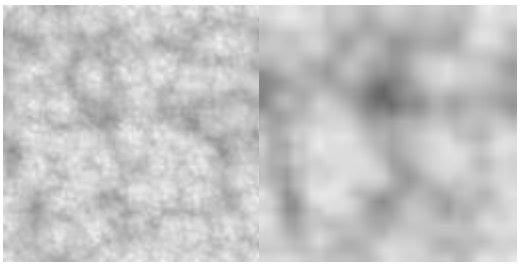


Joonis 14: Pilve kuju müra tekstuuri RGBA kanalid.

Joonisel 15 on kujutatud pilvedele detailide lisamise 3D-tekstuuri läbilõiked. See tekstuur koosneb kolmest kanalist ja on mõõtmetega 32^3 pikslit. Tekstuuri värvikanalites on kasvava sagedusega Worley müra. Kuigi jooniste põhjal võib tunduda, et see müra on madalama sagedusega kui pilve kuju määramiseks kasutatud müra, siis tegelikult see nii ei ole, sest tekstuuri mõõtmed on samuti palju väiksemad.

Joonis 15: Pilve detaili müra tekstuuri RGB kanalid.

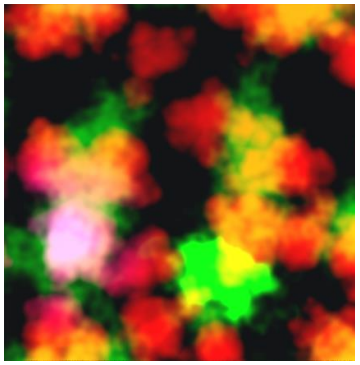
Mõlema 3D-tekstuuri värvikanalid kombineeritakse renderdamisel kokku üheks väärtuseks. Seda tehakse iga kord täpselt ühtemoodi, seega saab selle protsessi optimeermiseks pakkida värvikanalid üheks kanaliks kokku juba enne tekstuuri kasutamist. Pakkimisel liidetakse mürad kokku järjekorras punane, roheline, sinine ja viimasena alfa kanal, vähendades järk-järguliselt iga kanali panust. Pakkimise tulemusena saadud tekstuuri on näidatud joonisel 16.



Joonis 16: Pilve kuju tekstuur (vasakul), detaili tekstuur (paremal).

3.5 Ilma tekstuur

Ilma tekstuur, mis on näidatud joonisel 17, määrab taevas pilvede asukoha, tüübi ja valguse sumbumise vihmapilvedes. Andmed on salvestatud punases, rohelises ja sinises värvikanalis. Punases kanalis on pilvede kattuvuse andmed, mis määravad ära pilvede asukohad taevas. Rohelises kanalis on pilvede tüübi info, kus väärtus 0 tähistab kihtpilvi, 0,5 tähistab kihtrünkpilvi ja 1 tähistab rünkpilvi. Sinine kanal määrab pilvede vihmarohkuse. Vihmapilved erinevad tavalistest pilvedest selle poolest, et neisse neeldub rohkem valgust ja nad paistavad teistest pilvedest tumedamana. Ilma tekstuur katab taevast 20x20 kilomeetrilise ala.



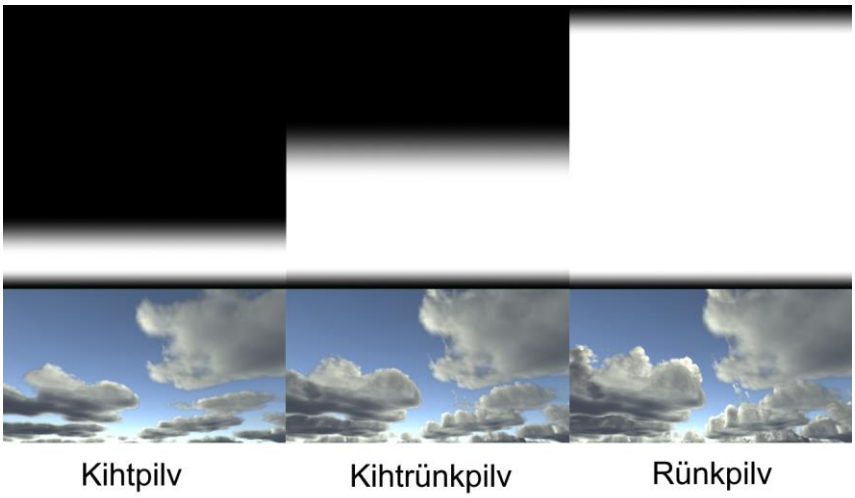
Joonis 17: Ilma tekstuur.

Ilma tekstuuri genereeritakse protseduuriliselt või loetakse sisse eelnevalt loodud ilma tekstuur. Protseduurilisel genereerimisel kasutatakse Worley müra ja Simplex müra, mis sarnaneb Perlini müraga, kuid on kiirem arvutada. Et minna sujuvalt üle ühelt ilma tekstuurilt teisele ilma tekstuurile, interpoleeritakse 30 sekundi jooksul kahe tekstuuri väärtuse vahel lineaarselt.

3.6 Pilve kuju

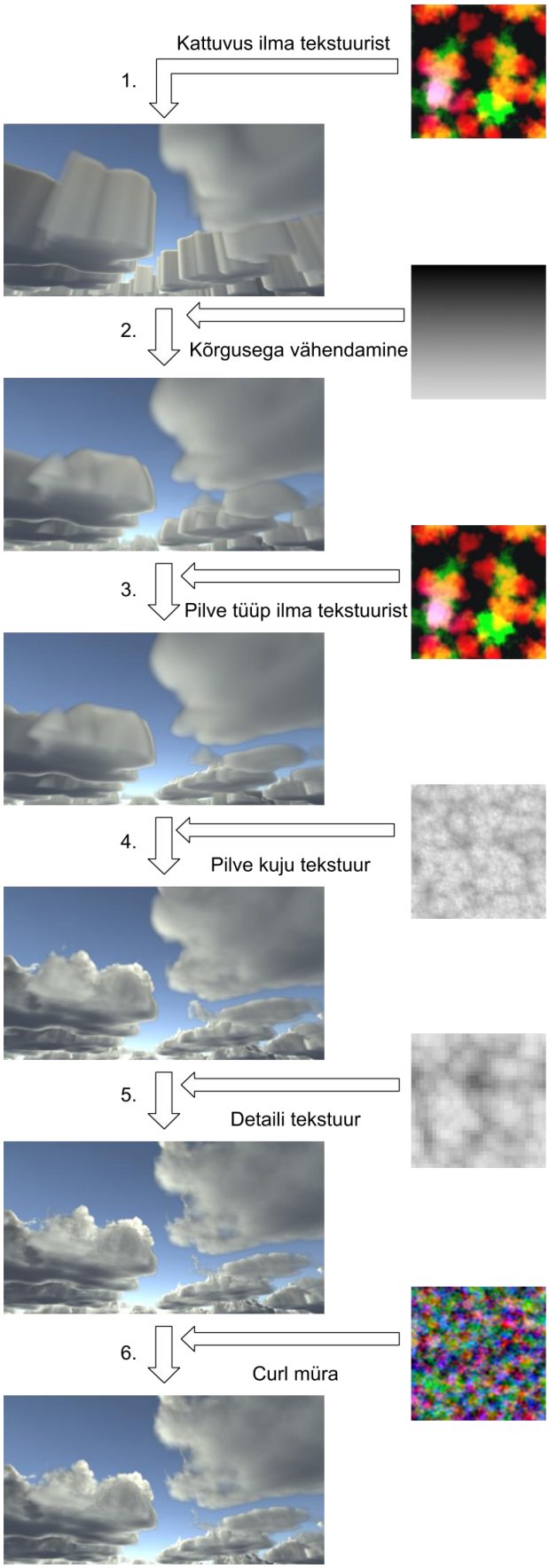
Pilve kuju moodustumine toimub läbi kuue etapi. Joonisel 19 on toodud need pilve kuju sämplimise etapid ja igale etapile vastav pilve välimus. Kõigepealt sämplitakse ilma tekstuuri, joonis 19.1, selleks, et teada saada, kas sämplimise punkt asub pilve sees. Seejärel, joonisel 19.2, vähendatakse kattuvuse väärtust sõltuvalt sämplitava punkti kõrgusest pilve kihis. Niimoodi saavutatakse pilvede hõrenemine kõrguse kasvades.

Õige pilve tüübi renderdamiseks kasutatakse ilma tekstuuris määratud pilve tüüpi, joonisel 19.3. Igale pilve tüübile vastab erinev gradient, mis on kujutatud joonisel 18 koos renderdatud pilvedega.



Joonis 18: Pilve gradiendid koos renderdatud pilvedega.

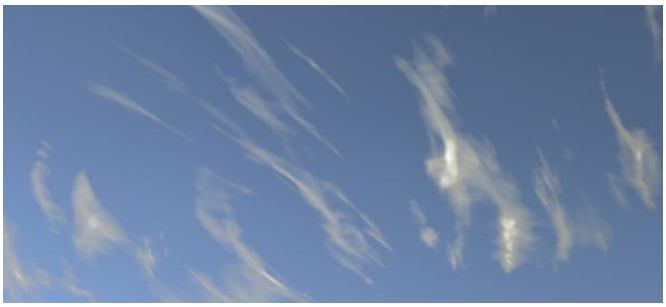
Pilvede peamine kuju leitakse pilve kuju tekstuuri sämplides, joonis 19.4. Saadud tulemus sarnaneb reaalsete pilvedega, suured ümarad pilvekogud, millest on puudu vaid detailid pilvede äärtes. Nende lisamiseks sämplitakse pilve detaili tekstuuri, joonis 19.5, mille abil uuristatakse eelnevalt saadud pilvede ääri. Selle käigus tekivad äärtesse väikesed ümarad pilvekogud. Viimasel etapil lisatakse turbulentse õhu liikumine, mis neid väikeseid pilvekogusid moonutab. Selleks sämplitakse Curl müra tekstuuri, mille põhjal luuakse õhu osakese liikumise vektor, mille igale värvikanalile vastab x-, y- või z-koordinaattelg. Vektori pikkus sõltub sämplimise positsiooni kõrgusest, mida kõrgemal seda pikem, sest tuule kiirus sõltub ka atmosfääri kõrgusest [32]. See vektor liidetakse pilvede detaili tekstuuri sämplimise positsioonile juurde, mis selle kuju moonutab, joonis 19.6. Curl müra efekt tuleb kõige paremini nähtavale tuulega.



Joonis 19: Pilve kuju moodustumise töövoog.

3.7 Kahemõõtmelised kõrged pilved

Lisaks volumeetrilistele madalatele pilvedele on töösse lisatud ka kõrged pilved, kujutatud joonisel 20. Päris maailmas on need pilved kõrgustel 6-8 kilomeetrit ja nende paksus jääb vahemikku, 200-400 meetrit, seega võrreldes kõrgusega on nad õhukesed [9]. Aarvutigraafikas võib nende pilvede renderdamise lihtsustamiseks kasutada kahemõõtmelisi tekstuure. Kõrgete pilvede sämplimise asukoht arvutatakse sama kiire-sfääri lõikepunkti algoritmiga, mida kasutati ka sfäärilise atmosfääri leidmiseks. Kõrgete pilvede kattuvuse määrab sama ilma tekstuur, mida kasutati ka volumeetrilistel pilvedel. Et madalate ja kõrgete pilvede asukohad paistaksid teistsugused, ilma tekstuuri sämplimise positsiooni nihutatakse ja skaleeritakse erinevalt.



Joonis 20: Kõrged pilved.

3.8 Tuul

Päris maailmas tuul nii liigutab kui ka moonutab pilvi. Tuul võib olla erinevatel kõrgustel erineva suuna ja kiirusega [32]. Kuna volumeetrilised pilved on kõrgusel 1,5-5,5 kilomeetrit ja kõrged pilved on 8,5 kilomeetri kõrgusel, kasutatakse nii madalatel volumeetrilistel kui ka kõrgetel kahemõõtmelistel pilvedel oma tuule suunda ja kiirust. Tuule kiirus tõuseb koos kõrgusega ja sellepärast lisatakse volumeetrilistele pilvedele kõrgusest sõltuv nihe (*shear*), mis liigutab kõrgel olevaid pilvi tuule suunas natuke edasi. Juurde on lisatud ka aeglane ülespoole liikumine, mis simuleerib sooja õhu tõusmist.

Kui volumeetrilistel pilvedel kasutada sama tuule suunda ja kiirust, et liigutada nii 3D-tekstuuri kui ka ilma tekstuuri, siis saadud pilvede liikumine pole kõige huvitavam, kuna pilved on mööda taevast liikudes muutumatud. Sellepärast kasutatakse ilma ja 3D tekstuuridel erinevat tuule suunda ja kiirust. Nii saab määrata neile natuke erineva liikumise ning see aitab simuleerida pilve kuju muutumist. Kui mänguloojatel on vaja luua spetsiaalselt disainitud taevas kindlate pilvede asukohaga, aga jätta alles tuule tõttu pilvede muutumise illusioon, saab ilma tekstuuri tuule kiiruse määrata nulliks. Niimoodi liiguvad ainult 3D-tekstuurid tuule suunas, kuid ilma tekstuur koos pilvede asukohaga jääb paigale.

4 Valgustamine

Käesolev peatükk annab ülevaate arvutigraafika klassikalisest valguse mudelist Phong. Pärast seda kirjeldatakse pilvede valguse mudelit, otsese valguse arvutamist ning otsese ja ümbritseva valguse kombineerimist.

4.1 Ülevaade valgustuse mudelist

Kolmemõõtmelistele objektidele värvi arvutamiseks kasutatakse arvutigraafikas valgustusmudelit, mis võtab arvesse nii valgusallikad kui ka objekti materjali omadusi. Üks klassikaline valgustusmudel on Phong'i mudel [33], kus on kolm komponenti: ümbritsev valgus (*ambient*), hajus valgus (*diffuse*) ja kaamerasse otse peegeldunud valgus (*specular*).

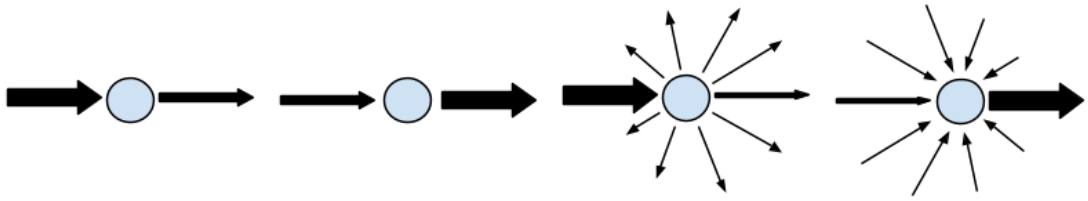
Reaalses maailmas peegeldub valgus objektidelt tagasi ning põrkab keskkonnas ringi, kuni kogu energia on sumbunud. Seda nimetatakse Phong'i mudelis ümbritsevaks valguseks ning see valgustab objekti ka sealt kuhu otsene valgus ei jõua. Ümbritseva valguse täpne arvutamine on aeglane, sest selleks peab simuleerima paljude valguskiirte põrkeid keskkonnas. Seetõttu kasutatakse selle lihtsustamiseks Phong mudelis konstantset väärtust, mis liidetakse teistele valguskomponentidele juurde.

Hajus valgus tuleneb sellest, kui valguskiir langeb pinnale, siis valgus põrkab pinna molekulides ringi ning väljub suvalises suunas. Hajusa valguse tugevus sõltub valguse langenurgast, sest pinnaga risti langev valgus katab väiksema pindala kui nurga all langev. Otse peegelduv valgus on osa pinnale langevast valgusest, mis peegeldub sellelt otse vaataja silma. Phongi mudelis leitakse otse peegelduva valguse arvutamiseks kosiinus peegeldunud valguse vektori ja pinnalt vaataja suunas vektori vahelisest nurgast ning võetakse see mingisse suurde astmesse, et läike suurust teha väiksemaks.

Pilvede valgustamisel kasutatakse ainult otsest valgust ja ümbritsevat valgust. Need arvutatakse iga pilve sees sämplitud punkti kohta. Otsese valguse leidmiseks arvutatakse, kui palju valgust jõuab sellesse ruumi punkti. Selleks leitakse optiline tihedus sämplitava punkti ja valgusallika vahel. Mida suurem on tihedus, seda vähem valgust vaadeldava punktini jõuab. Ümbritsev valgus tekib päikese valguse peegeldumisel maapinnalt, teistelt pilvedelt ja atmosfääris molekulidelt hajudes. Ümbritseva valguse arvutamisest kirjutatakse täpsemalt peatükis 4.4.

4.2 Otsene valgus

Päris maailma pilvedel pole kindlat pinda, millelt valgus saab peegelduda. Selle asemel liiguvad valguse footonid pilve sisse, kus nad põrkavad teiste pilve osakestega. Valgus võib osakeste mõjul sumbuda (*attenuation*), kiirguda (*emission*), välja hajuda (*out-scattering*) või sisse hajuda (*in-scattering*) [34], neid protsesse on kujutatud joonisel 21. Valguse energia väheneb sumbumisel, mille käigus muundub valguse energia osakese siseenergiaks ehk soojuseks, või hajudes teistesse suundadesse. Samuti võib ka valguse energia suureneda läbi kiirgumise või sise-hajumise tõttu. Kiirgumist pilvedel ei esine, see toimub näiteks tuleleegil. Välja ja sisse hajumise tõttu valguse energia mõnes suunas väheneb ning teistes suundades suureneb, kuna hajumisel muudab valguskiir suunda või muutub mitmeks väiksema energia kuid erineva suunaga kiireks.



Joonis 21: Vasakult paremale: valguse sumbumine, kiirgumine, välja hajumine, sise-hajumine.

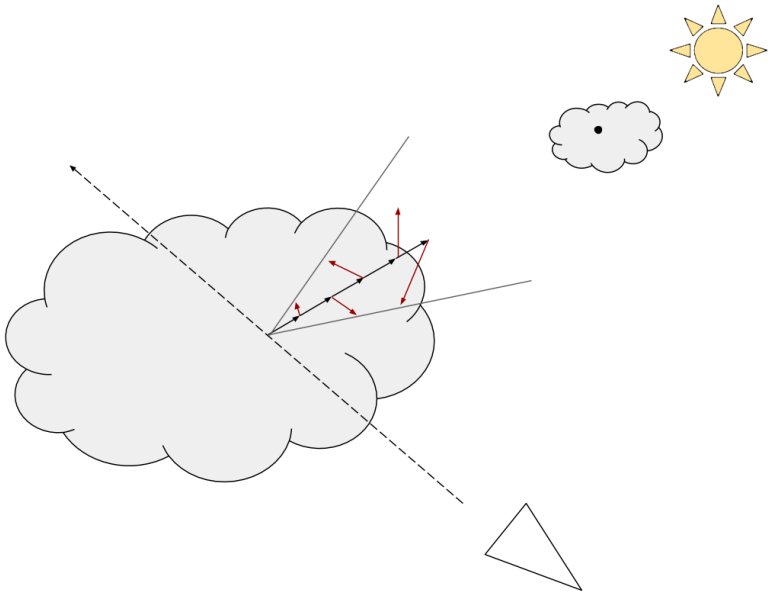
Otsene valgus arvestab päikesest tulnud valgust ning selle liikumist pilvedes. Otsese valguse arvutamisel kasutatakse:

1. Beeri seadust
2. Henyey-Greenstein faasi funktsiooni
3. Sise-hajumise funktsiooni

4.2.1 Valguse marssimine koonuses

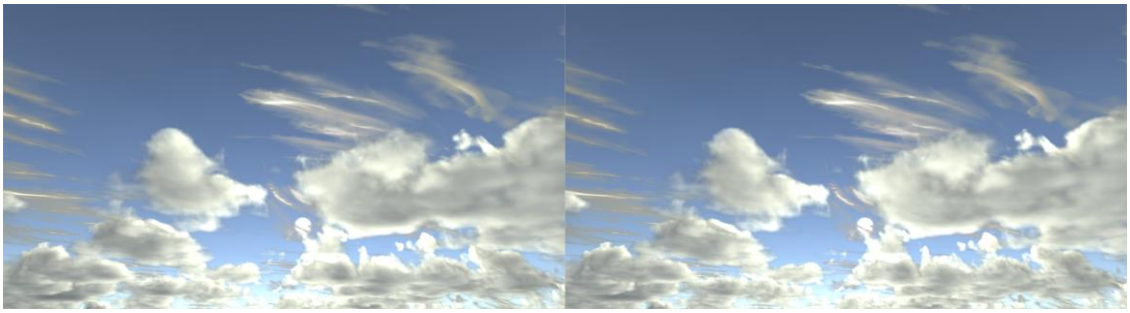
Pilved ei ole jaotunud pilvekihis ühtlaselt, mis tähendab, et erinevates punktides on pilve tiheduse väärtused erinevad. Lisaks võib mõni pilv olla ka vihmapilv, kus optiline tihedus on suurem kui tavalistel pilvedel. Sellepärast tuleb pilve optilise tiheduse arvutamiseks teha uuesti kiirte marssimist sämplitavast punktist valgusallika suunas, et leida, kui palju valgust sämplitavasse punkti jõuab. Samuti pole õige seda üksnes sirgjoones marssida, sest valgus pilvedes hajub ja liigub siksakiliselt. Sellepärast tehakse kiire marssimist koonusena valgusallika poole poole [21], kujutatud joonisel 22. Valgusele tehakse kokku kuus sämplit, viis koonuses ja üks kaugemal, et saada pilvedele varje ka teistelt pilvedelt. Kuus sämplit on valitud sellepärast, et vähemate sämplite arvu korral hakkab valgustuse kvaliteet langema ning suuremal sämplite arvul on arvutuskiirus liiga aeglane.

Koonuse kuju saamiseks kasutatakse viit juhusliku ettearvutatud vektorit, mis asuvad ühiksfääril. Marssides valgust, võetakse igal sammul järgmine vektor, tehakse see pikemaks ja liidetakse nihkena marssitavale positsioonile juurde. Kuna igal sammul suurendati vektori pikkust, moodustub vektoreid sisaldavast ruumist koonus, mis laieneb valgusallika suunas. Juhuslikud vektorid on joonisel 22 kujutatud punaselt. Selle lähenemise probleemiks on see, et kiirte marssimisel on valguse koonused igas sämplitavas punktis täpselt samad. Kuna koonusese loomiseks kasutatakse ainult viit vektorit, siis võib samuti juhtuda, et kõik vektorid on suunaga ühele sfääri poolele. Selle lahendamiseks saab juhuslikud vektoreid leida ka dünaamiliselt, kui kasutada räsifunktsiooni valguse marssimise positsioonidel ning luua igale positsioonile uus juhuslik vektor.



Joonis 22: Valguse marssimine koonuses.

Joonisel 23 on võrreldud visuaalset erinevust, mis tekib konstantse ja juhusliku koonuse kasutamisel. Joonisel on ka koonuse raadiust suurendatud, et erinevust oleks paremini näha. Nagu näha, siis erinevus on üsna väike ning väljendub peamiselt selles, et pilve keskosad on juhusliku koonuse puhul veidi heledamad. Kuna juhuslike väärtuste arvutamine on

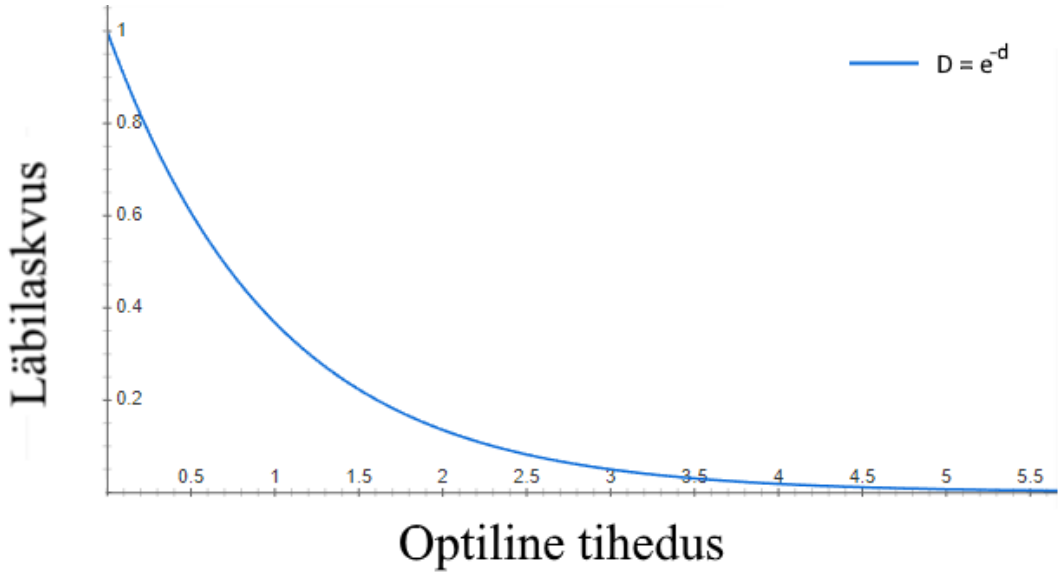


Joonis 23: Konstante koonus (vasakul), juhuslik koonus (paremal).

aeglasem kui eelnevalt defineeritud vektoreid kasutades, siis pole seda reaalaja simulatsioonides mõistlik rakendada.

4.2.2 Beeri seadus

Beeri seadus kirjeldab valguse sumbumist materjalis [35]. Selle abil saab arvutada, kui palju valgust jõuab materjali sisse ning kui palju see materjali sumbub. Beeri seaduse funktsiooni graafikut on kujutatud joonisel 24.



Joonis 24: Beeri seadus. Valguse läbilaskvuse sõltuvus optilisest tihedusest.

Beeri seadusega arvutatakse ainult valguse sumbumist, kuid sise-hajumisega valguse energia kasvu see ei arvesta. Seetõttu näevad pilved välja liiga tumedad, näidatud joonisel 25. Selle parandamiseks arvutatakse Beeri seaduse funktsiooni kaks korda, kus ühel korral vähendatakse optilist tihedust, et valgus tungiks sügavamale pilve aga vähendatakse ka funktsiooni väärtust [4]. Lõplik tulemus on mõlema funktsiooni maksimaalne väärtus. Näidisel 1 on toodud Beeri funktsioonist implementatsioon varjutaja koodis.

```
float beerLaw(float density) {
   return max(exp(-density), exp(-density * 0.5) * 0.7);
}
```

Näidis 1: Beeri seaduse implementeering koodis.

Joonis 25: Vasakul on valgustus kasutades ühte Beeri funktsiooni, paremal kasutatakse kahte funktsiooni.

4.2.3 Henyey-Greenstein'i faasi funktsioon

Volumeetrilise nähtuste renderdamisel kasutatakse Henyey-Greenstein (HG) faasi funktsiooni, et modelleerida valguse edasi-hajumise tõenäoust. Edasi-hajumine on nähtus, kus enamik footoneid jäävad liikuma enamvähem samas suunas kui enne hajumist. Nurk enne hajumise ja pärast hajumise suuna vahel on alla 90 kraadi [36]. Algselt loodi see funktsioon selleks, et kirjeldada valguse hajuvust tähtedevahelise tolmu pilvedes [37]. Henyey-Greenstein faasi funktsiooni väärtus sõltub vaatleja ning valguse vektori suunast ja numbrilisest sisendparameetrist lõigus [-1, 1]. HG funktsiooni mõju pilvedele on kujutatud joonisel 26.

Joonis 26: Koos HG faasi funktsiooniga (vasakul), ilma HG faasi funktsioonita (paremal)

Henyey-Greenstain'i faasi funktsiooni implementatsioon on toodud näidises 2. Muutuja *cosAngle* on võrdne kiire suuna ja päikese valguse suuna vahelise nurga kosiinusega ning muutja *g* on funktsiooni parameeter.

```
float HenyeyGreensteinPhase(float cosAngle, float g){
   float g2 = g * g;
   return ((1.0 - g2) / pow(1.0 + g2 - 2.0 * g * cosAngle, 1.5)) / 4.0 *
          3.1415;
}
```

Näidis 2: Henyey-Greenstain'i faasi funktsiooni implementatsioon koodis

Kasutades ühte HG faasi funktsiooni selgub, et päikese loojangul muutuvad päikese vastasküljel olevad pilved liiga tumedaks (vaata joonist 27), sellepärast tuleb kasutada kahte HG faasi funktsiooni [4]. Ühel neist on negatiivse funktsiooni parameetriga, kus toimub tagasi-hajumine (*back-scattering*) [38] ehk valguse põrkavad otse tagasi. Mõlemast HG funktsiooni väärtustest valitakse edaspidiseks kasutamiseks maksimaalne väärtus.

Joonis 27: Pilved päikesest eemal. Üks HG faasi funktsioon (vasakul), kaks HG faasi funktsiooni (paremal).

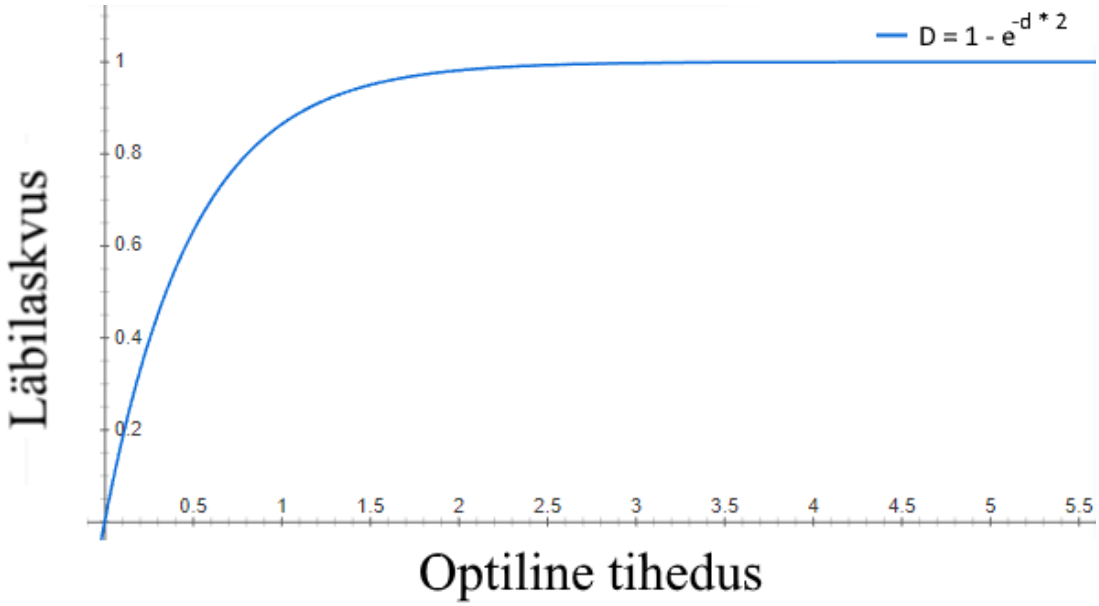
4.2.4 Sise-hajumise funktsioon

Pilve päikesepoolsel küljel paistavad selle ääred tihti tumedamad kui ümara osa keskel (näidatud joonisel 28). See nähtus tekib valguse sise-hajumisest pilves ning Beeri seaduse abil seda modelleerida ei saa [21]. Beeri seadus modelleerib ainult valguse sumbimist meediumis, kuid sisehajumise korral peab modelleerima ka valguskiirte liitumist. Tavaliselt hajub valgusega samas suunas edasi-hajumise tõttu, aga kui optiline tihedus on piisavalt suur, siis võib valgus osakeste vahel põrgates pöörata kuni 180-kraadi. Pilvede ääres see ei juhtu, sest ümbritsevaid osakesi on seal liiga vähe. Sügavale jõudnud valguskiired sumbuvad Beeri seaduse järgi enne kui nad jõuavad põrgata 180 kraadi ja väljuda pilvest.

Erandiks on lõhed ümarate pilve osade vahel, kus valgus siseneb pilve ümara osa keskelt, hajub seal ning väljub lõhest ja jõuab vaatlejani.

Joonis 28: Foto [54] tumedatest äärtest.

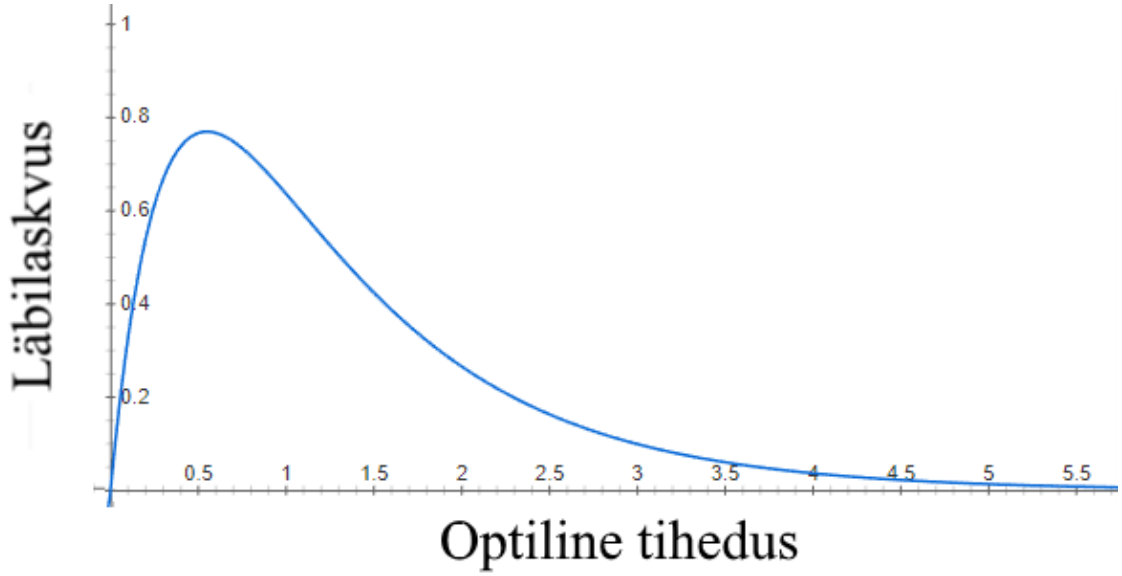
Sarnane nähtus esineb ka tuhksuhkrul ja edaspidi nimetatakse seda tuhksuhkru efektiks (tuhksuhkur on inglise keeles *powdered sugar* ja nähtuse nimi tuleb seda lühendades *powder-effect*) [21]. Tuhksuhkru efekti on kõige paremini näha tihedatel ümaratel pilvealadel, kus nende vahelised lõhed on heledamad kui ümar osa ise. Seda efekti saab ligikaudselt arvutada kasutades ümberpööratud Beeri funktsiooni [21], tuhksuhkru funktsiooni graafikut on kujutatud joonisel 29.



Joonis 29: Tuhksuhkru efekti funktsioon.

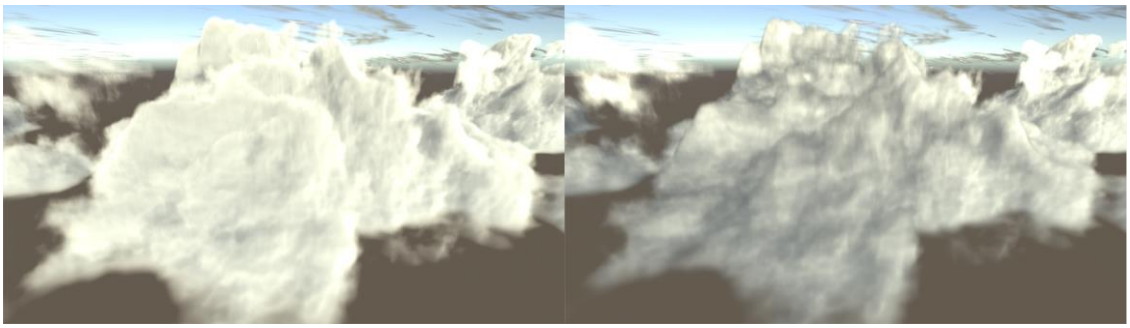
4.3 Otsese valguse kombineerimine

Beeri seadus ja tuhksuhkru efekti kombineeritakse korrutades mõlema funktsiooni tulemused kokku. Kuna mõlema funktsiooni väärtused on nulli ja ühe vahel, siis nende korrutamisel saadud väärtus on veelgi tumedam. Selle lahendamiseks korrutatakse lõplik väärtus kahega, et muuta seda uuesti heledamaks. Kombineeritud funktsiooni graafik on kujutatud joonisel 30.



Joonis 30: Beeri-tuhksuhkru funktsioon kombineeritult.

Joonisel 31 on kujutatud pilvede erinevus ainult Beeri funktsiooni kasutades ning Beeri-tuhksuhkru funktsiooni kasutades. Pildilt on märgata, et ümarate osade ääred muutusid tumedamaks.

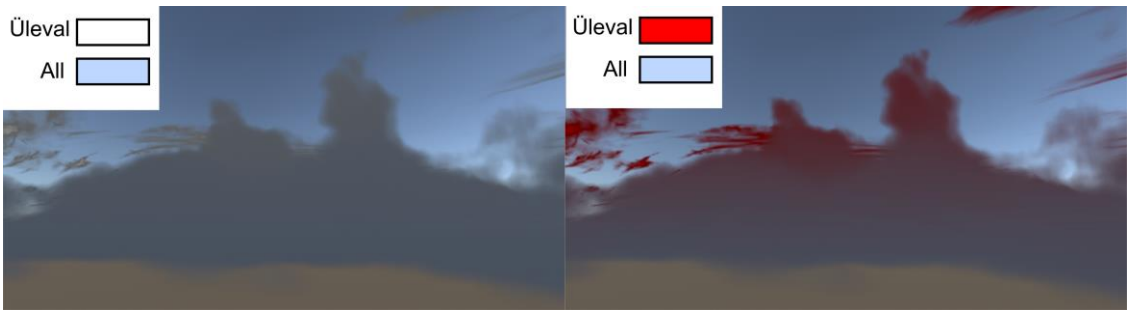


Joonis 31: Ainult Beeri funktisoon (vasakul), Beeri-tuhksuhkru funktsioon (paremal).

Beeri-tuhksuhkru funktsiooni ja HG faasi funktsioon kombineeritakse samuti üksteise väärtuste korrutamisega. Sellega on otsene valgus arvutatud.

4.4 Ümbritsev valgus

Ümbritsev valgus tekib kui päikese valgus peegeldub maapinnalt või hajub atmosfääri molekulitelt. Pilvede värvi mõjutab altpoolt maapealt peegelduv valgus, ülevalt aga atmosfäärilt peegelduv valgus, seega ümbritseva valguse värv ja tugevus on pilvede all ja peal erinev. Selle protsessi füüsikaliselt täpne simuleerimine on aga väga aeglane. Kuna pilvedeni jõudev ümbritsev valgus on küllaltki hajutatud, võib seda lihtsustada, võttes kasutusele kaks konstantset väärtust [17]: üks alumisele ja teine ülemisele valgusele. Joonisel 32 vasakul on kujutatud ümbritsev valgus koos ülemise ja alumise värviga, paremal on ülemist värvi muudetud, et ümbritseva valguse mõju paremini esile tekiks.



Joonis 32: Ümbritsev valgus (vasakul), suur värvide erinevus (paremal).

Ümbritseva valguse koodi implementatsioon on toodud näidises 3.

```
ambientLight = lerp(_CloudBaseColor, _CloudTopColor, height);
```

Näidis 3: Ümbritseva valguse arvutamine koodis.

4.5 Valguse kombineerimine

Pilve lõplik valgus saadakse liites kokku otsene ja ümbritsev valgus, koodi implementatsioon on näidises 4. Saadud valgus korrutatakse pilve tiheduse väärtusega, mille tagajärjel saadakse pilve sämpli värv ning see akumuleeritakse kiirte marssimisel koos teiste sämplitega kokku.

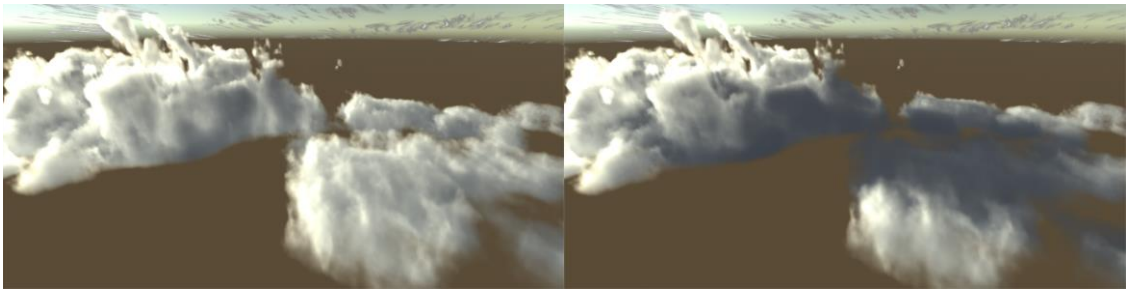
```
lightColor = directLight * _DirectFactor + ambientLight * _AmbientFactor;
```

Näidis 4: Otsese ja ümbritseva valguse kombineerimine.

4.6 Pilvedevahelised varjud

Pärast koonusena pilve tiheduse sämplimist tehakse pilvest väljaspool ainult üks sämple, mis kinnitab, et on jõutud pilvest välja. Selline lähenemine ei tekita aga pilvede vahelisi varje, kuna võib juhtuda, et pilv on koonuse ja eemal oleva sämpli vahel või pärast seda. Selle saab lahendada, kui ühe sämpli asemel teha neid rohkem, näiteks kokku 25, sel juhul heidavad pilved üksteisele varje kaugemalt ja detailsemalt. Pilvede vaheline varjude

heitmise kaugus sõltub sammude arvust ja pikkusest. Pilt erinevusest on toodud joonisel 33. Nii saab küll parema tulemuse, kuid reaalaja rakendustes ei tasu see ära, sest lisatud valguse sämplimine nõuab palju arvutusjõudlust ja on aeglane.



Joonis 33: Kui teha rohkem kui 6 valguse sämplit, siis heidab pilv varju teisele pilvele. Vasakul 6 valguse sämplit, paremal 30 (25 + 5 koonuses) valguse sämplit.

5 Implementatsioon

Selles peatükis antakse ülevaade pilvede implementatsioonist Unity's. Kirjeldatakse loodud varjutajaid (*shader*) ning nende parameetreid.

5.1 Unity

Unity [39] on mängumootor, mille peamine eesmärk on 3D- ja 2D-mängude või simulatsioonide loomine erinevatele platvormidele, nagu arvutid, konsoolid ja mobiilseadmed. Unity toetab kokku 27 erinevat platvormi [40]. Mänguloogika programmeerimine Unitys käib keeltes C# ja JavaScript [41].

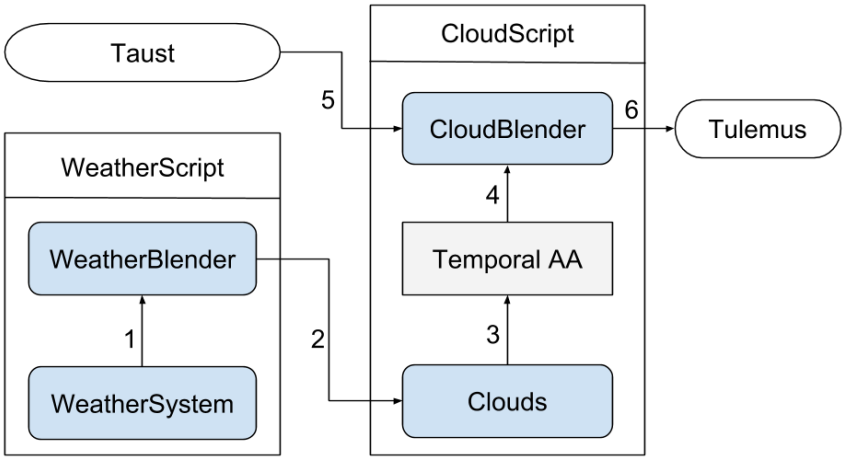
Unity võimaldab luua verteks-, fragment- [42], arvutusvarjutajaid [43] (*vertex, fragment and compute shaders*), seda saab teha programmeerimiskeeltes HLSL, GLSL ja Cg [44]. Lisaks neile saab luua ka pinna varjutajaid (*surface shader*) kirjutamist, mis hiljem teisteks shaderi tüüpideks kompileeritakse. [45].

Unityst on saadaval nii tasuta (Unity Personal) kui ka tasulised versioonid (Unity Plus, Unity Pro) [46]. Nende versioonide peamiseks erinevuseks on see, et tasuta versiooni võib kasutada, kuni sellega loodud mängude tulu on alla $100 000 aastas, Unity Plus võib kasutada siis, kui aastane tulu on alla $199 000 ning Unity Pro'd tuleb kasutada, kui aastane tulu on üle $199 000 [47].

5.2 Varjutajate ülevaade

Peatükis 3.1 on kirjeldatud, et kiirte marssimist tehakse mingis piiritletud ruumis. Selleks võib olla lihtne kuup või kogu kaamera vaate tüvipüramiid. Kui kiirte marssimist tehakse geomeetrias, näiteks kuubi sees, siis seda saab implementeerida tavalise varjutajana, luua materjali ning rakendada seda materjali geomeetriale. Kuna taevas katab suure osa kaamera vaateväljast ning kaamera võib paikneda kogunisti pilvede sees, siis on nende renderdamiseks mõistlik seda teha kaamera vaates ja selleks tuleb luua järeltöötluse efekt (*post-processing effect*).

Nii pilvede renderdamiseks kui ka ilma tekstuuri genereerimine toimub eraldi skriptides. Joonisel 34 on kujutatud pilvede renderdamise töövoog. Varjutajas *WeatherSystem* genereeritakse ilma tekstuur, mis saadetakse varjutajasse *WeatherBlender* (joonis 34.1), kus interpoleeritakse eelmise ja uue ilma tekstuuri vahel. Sealt saadetakse ilma tekstuur skripti *CloudScript* (joonis 34.2), kus varjutaja *Clouds* renderdab pilved eraldi tekstuurile, mille peal tehakse ajalist sakitõrjet (*Temporal anti-aliasing*, TAA), joonis 34.3. Pilved lisatakse stseenile varjutajas *CloudBlender*, mis saab sisendiks renderdatud pilved (joonis 34.4) ja renderdatud stseeni (joonis 34.5).



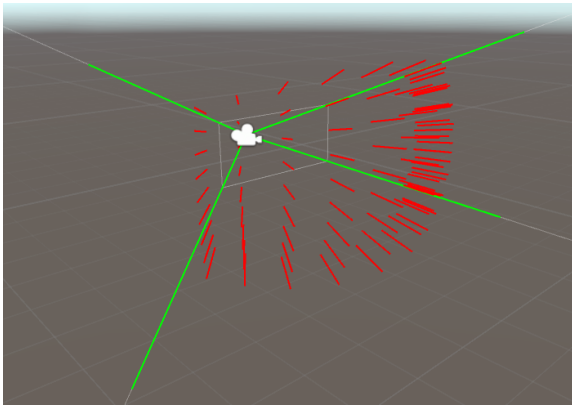
Joonis 34: Varjutajate ja skriptide töövoog.

5.3 Pilve varjutajad

Kiirte marssimise aluseks on võetud Generic Raymarch Unity [48] kood. Seal on loodud skripti ja varjutaja aluskood kiirte marssimiseks mängumootoris Unity. Skriptis luuakse nelinurk (*quad*), mis katab kogu kaamera vaate ja mille peal tehakse kiirte marssimist kaamera vaate tüvipüramiidis. Kiirte marssimiseks tuleb leida igale pikslile vastav kiire suund ning saata see fragmendi varjutajasse. Kiirte suundade arvutamine toimub läbi kolme sammu [49]:

1. Leitakse neli kaamera tüvipüramiidi moodustavat vektorit, joonisel 35 kujutatud rohelistena. Vektorid saadetakse varjutajasse 4x4 maatriksis.
2. Renderdamisel kasutatakse kohandatud Graphics.Blit() funktsiooni, mis renderdab ekraani katva nelinurga ja rakendab sellele mingit varjutajat. Seega saab selle abil muuta iga ekraani piksli värvi ning seda kasutatakse enamasti järeltöötluse efektide loomiseks. Et teostada kiirte marssimist, lisatakse igale nelinurga tipule ka vastava tüvipüramiidi vektori indeks. Seda indeksit verteksvarjutajas kasutatakse, et leida vastava vektori indeks maatriksist.

3. Verteksvarjutajast saadetakse iga nurga kiire suunad fragmendivarjutajasse. Varjutajas interpoleeritakse automaatselt kiirte vahel nii, et saadakse igale pikslile vastav kiire suund, mida on joonisel 35 kujutatud punaselt.



Joonis 35: Visualisatsioon kaamerast väljuvatest kiirtest [47].

Varjutajas *Clouds* toimub pilvede renderdamine ja valgustamine, mis on kirjeldatud peatükkides 3 ja 4. Pilved renderdatakse eraldi tekstuurile, millele rakendatakse ka ajalist sakitõrjet, sellest kirjeldatakse täpsemalt peatükis 6.1. Pilved lisatakse taustale varjutajas *CloudsBlender*, kus sujutatakse (*blending*) pilved taustaga kasutades eelkorrutatud alfasujutamist (*premultiplied alpha blending*).

5.4 Ilma varjutajad

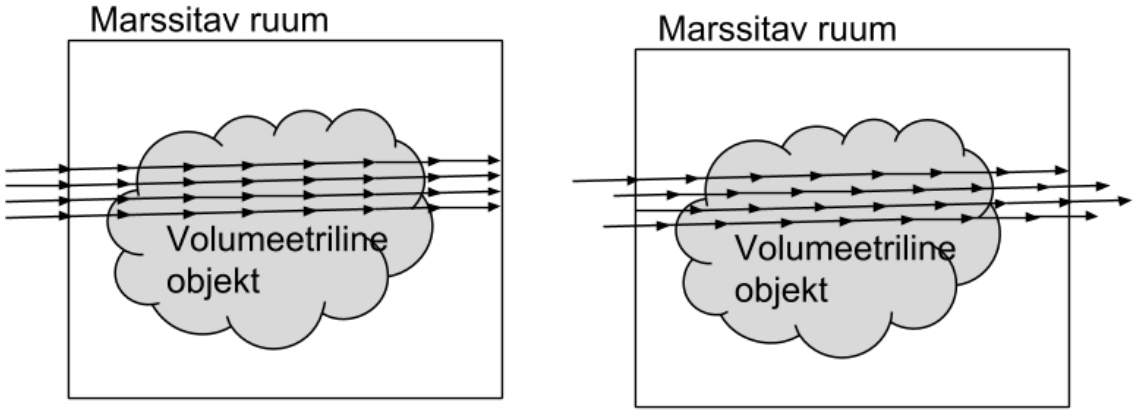
Ilma tekstuuri genereerimine ja vahetamine toimub skriptis *WeatherScript* ja ilma varjutajates. Uus ilma tekstuur genereeritakse ainult siis kui vastav meetod skriptis välja kutsutakse. Tekstuuri genereerimine toimub varjutajas *WeatherSystem*, mille käigus kasutatakse Worley müra implementatsiooni Turbulence Library'st [50] ja Simplex müra [51]. Iga ilma tekstuur renderdatakse eraldi tekstuurile. Peale selle renderdamist alustatakse tekstuuride vahetamise protsessi, kus interpoleeritakse lineaarselt uue ja vana tekstuuri vahel. Interpoleerimine toimub varjutajas *WeatherBlender*, mis võtab kolm sisendparameetrit: vana ilma tekstuur, uus ilma tekstuur ja koefitsent, mis määrab, kui palju vana ja uut tekstuuri võtta. Ilma skriptis võib määrata ka eelnevalt genereeritud ilma tekstuuri, mis võimaldab kasutajal kasutada enda loodud ilma tekstuuri.

6 Optimeerimine

Kiirte marssimine ilma optimeeringuteta on aeglane ja teeb palju ebavajalikke arvutusi. Käesolevas peatükis kirjeldatakse töös rakendatud optimeerimisvõtteid.

6.1 Stohhastiline sämplimine

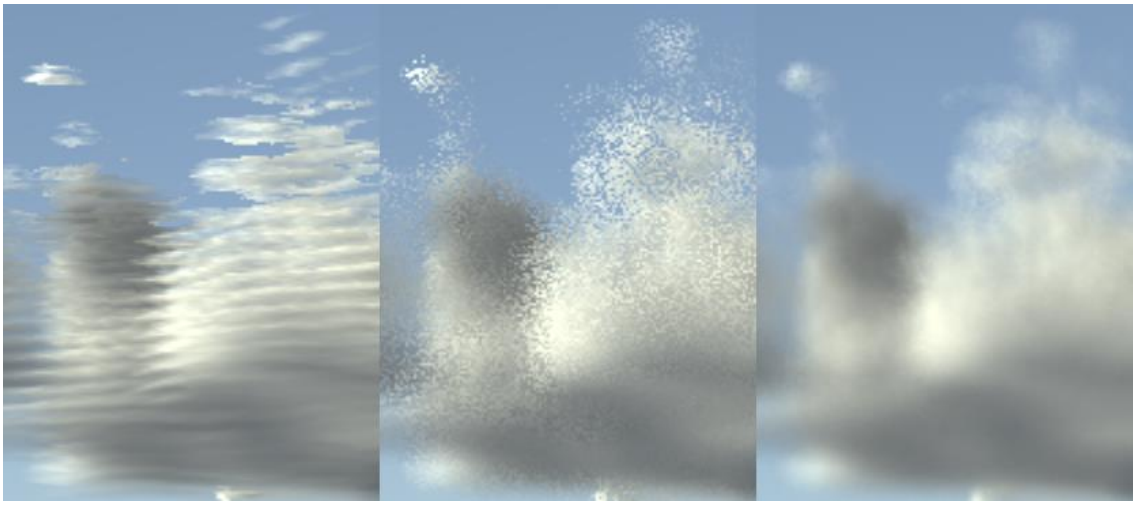
Tõenäosusteoorias ja sellega seotud aladel nimetatakse stohhastiliseks protsessiks (*stochastic process*) sellist protsessi, mille kulg sõltub juhusest ehk seda protsessi muudetakse juhuslike väärtuste abil vähem korduvaks [52]. Kiirte marssimisel saab kasutada stohhastilist sämplimist (*stochastic sampling*), mis tähendab, et kiire alguspunktile lisatakse juurde juhuslik, kuni ühe sammu pikkune, osa. Erinevust tavalise kiirte marssimise ja stohhastilise sämplimisega marssimise vahel on kujutatud joonisel 36. Selline lähenemine lubab kiirte marssimisel tehtavate sämplite arvu vähendada ilma, et tekiks joonisel 37 vasakul kujutatud artefakte. Juhuslik suurus saadakse sinise müra tekstuuri sämplides [53].



Joonis 36: Vasakul on tavaline kiirte marssimine, paremal stohhastilise sämplimisega kiirte marssimine.

Ainult juhusliku suuruse juurde lisamisest ei piisa, sest saadud tulemus on mürane, kujutatud joonise 37 keskel. Igal kaadril on juhuslik suurus erinev, mistõttu sämplitakse erinevaid positsioone ja lühikese aja möödudes sämplitakse pilves peaaegu kõiki punkte. Eelmiste kaadrite info kaasamiseks kasutatakse ajalist sakitõrjet (*temporal anti-aliasing*, TAA), mis on võte, kus kasutatakse sakitõrjeks eelnevalt renderdatud kaadreid. Käesolevas töös kasutatakse müra kaotamiseks avatud lähtekoodiga Playdead'i TAA implementatsiooni Unitys [54], mida kasutati ka mängus INSIDE [55]. Sellist lahendust kasutatakse ka

Frostbite implementatsioonis [17]. Joonisel 37 paremal on pilt, kus kasutatakse nii stohhastilist sämplimist kui ka ajalist sakitõrjet.



Joonis 37: Ilma stohhastilise sämplimisega (vasakul), stohhastilise sämplimisega (keskel), stohhastilise sämplimise ja TAAga (paremal).

6.2 Madalam resolutsioon

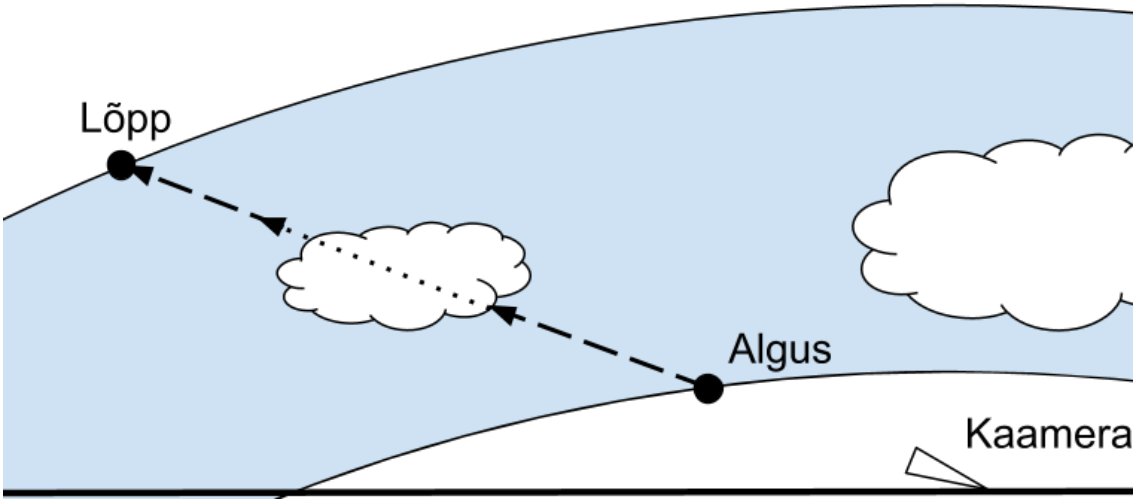
Kuna pilvedel puuduvad teravad ääred, siis saab jõudluse parandamiseks renderdada pilved esmalt madalamal resolutsioonil ning hiljem tagasi originaalsuurusesse venitada, ilma et pildi kvaliteet oluliselt langeks. Madala resolutsiooni implementatsioon on lihtne: varjutajas *Cloud* renderdatakse pilved väiksema resolutsiooniga tekstuurile ning tulemus antakse edasi varjutajale *CloudBlender*, mis interpoleerib tekselite (*texel*) vahelised väärtused automaatselt.

6.3 Varajane lõpetus

Kiirte marssimisel vähendab iga pilvesisene sämpel piksli läbipaistvust (suurendab alfa väärtust). Kui kiirte marssimise käigus on saanud pilve tihedus nii suureks, et see pole enam läbipaistev, näiteks alfa on suurem kui 0.99, siis võib marssimise lõpetada. Edasised sämplid enam piksli värvi ega läbipaistvust ei mõjuta ning seega pilve välimust ei muuda.

6.4 Suured sammud

Väljaspool pilvi on pilvede tihedus null ja ei mõjuta lõpliku piksli värvi. Seega pole mõtet teha palju sämpleid väljaspool pilvi. Kiirte marssimist alustatakse suurte sammudega. Kui pilve tiheduse sämplimisel tuleb nullist suurem, astutakse üks suur samm tagasi ning hakatakse tegema väiksemaid samme. Kui pilve tihedus on null, tehakse väikeseid samme veel 10 korda, siis hakatakse jälle suuremaid samme tegema, kujutatud joonisel 38. Ühe suure sammu pikkus on kolm väikest sammu. Sellise optimeeringuga muutub pilvede välimus väga vähe. Ainult mõned õhukesed pilve ääred võivad natuke väiksemaks muutuda.



Joonis 38: Väljaspool pilvi tehakse suuri samme, sees väikeseid.

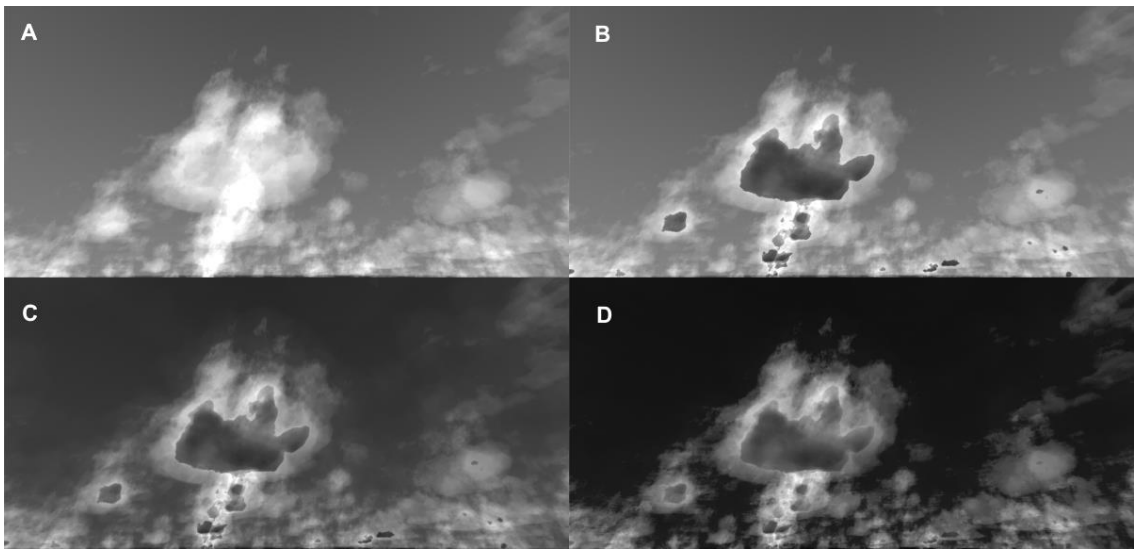
6.5 Muud optimeeringud

Valguse arvutamist tehakse ainult siis, kui pilve tihedus on nullist suurem. Seda võib teha, sest kui pilve tihedus on null, siis sealt ei põrka ega haju valgust kaamerasse.

Kui pärast pilve kuju tekstuuri sämplimist on pilve tiheduse väärtus null, siis detaili ega Curl müra tekstuuri samuti ei sämplita, sest pilve detaili tekstuur ainult vähendab pilve tiheduse väärtust. Sellega hoitakse veelgi tekstuuri sämplimiste arvu kokku. Sama tehakse ka siis, kui ilma tekstuurist saadud väärtus on null.

Joonisel 39 on näidatud, kuidas optimeeringud parandavad jõudlust. Joonisel tähistab must värv 0 korda tekstuuride sämplimist ja valge värv 1000 kordset tekstuuride sämplimist, vahepealsed väärtused on lineaarselt interpoleeritud. Esimesel juhul, pildil A, on ainuke optimeering see, et sämplitakse valgust pilve sees, ehk on näha, et seal tehakse väga palju sämplimisi. Teisel juhul, pildil B kasutatakse lisaks ka varajast lõpetust ning on näha, et

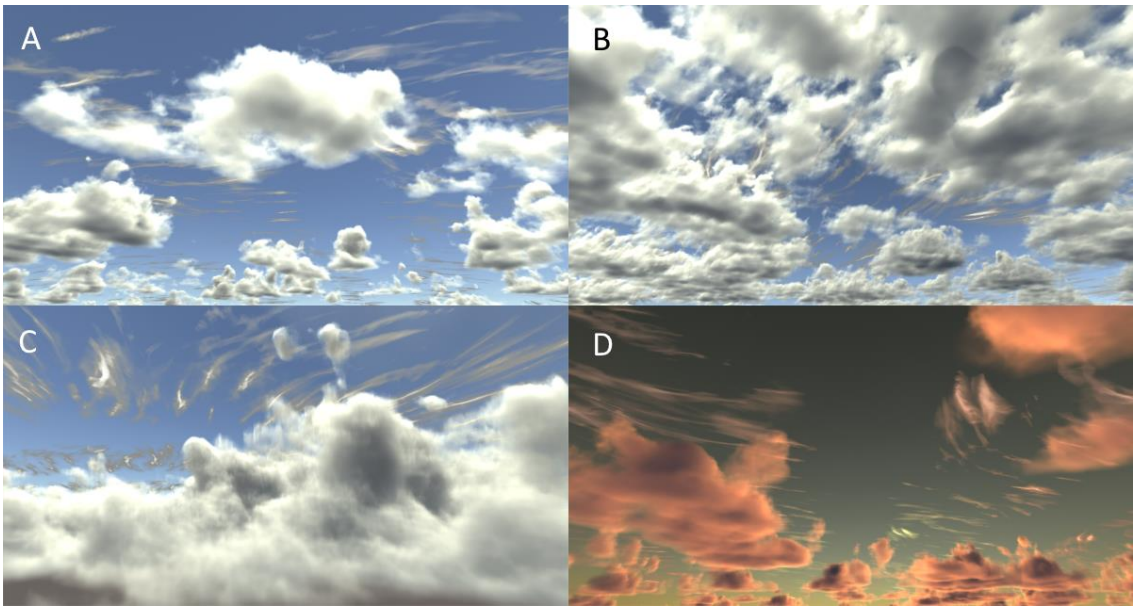
pilvede sees tehakse nüüd vähem sämpleid. Kolmandal juhul, pildil C, sämplitakse detaili ja Curl müra tekstuuri ainult siis, kui pilve kuju tekstuuri väärtus pole null. Sellest on näha, et väljaspool pilvi väheneb tehtavate tekstuuri sämplimiste arv. Neljandal juhul, pildil D, tehakse väljaspool pilvi suuri samme ja pildilt on näha, et väljaspool pilvi läheb taevas tumedamaks ehk sämplite arv langeb.



Joonis 39: A - valguse arvutamine pilve sees, B - varajane lõpetus, C - detaili ja Curl müra sämplimine kui kuju pole null ja D - suured sammud.

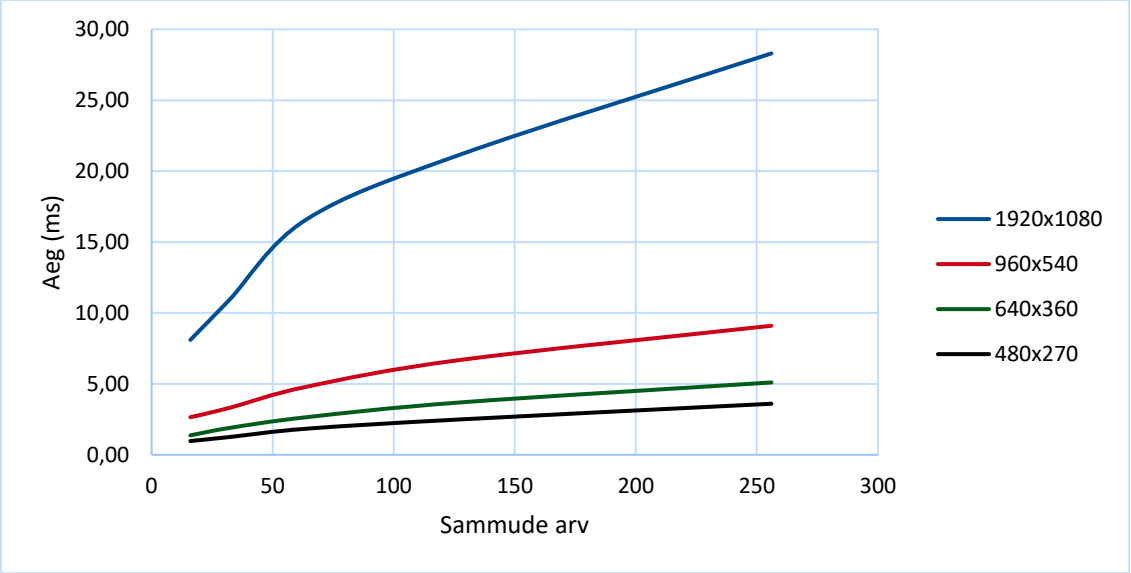
7 Tulemus

Joonisel 40 on renderdatud pildid pilvedest. Piltil A ja B on kaamera pilvede all ning päike on kõrgel. Pildil C on kaamera pilvekihi sees. Pildil D on päikeseloojang ja on näha pilvede värvi muutumist.



Joonis 40: Lõplikud pilvede renderdused.

Pilvede renderdamisel on suurim koormus videokaardil, sest enamus arvutusi tehakse fragmendivarjutajas. Protsessoril ei tehta kulukaid arvutusi, peamiselt saadetakse vaidparameetrid varjutajasse. Demorakendusest on ehitatud arendusversioon (*development build*), kuhu on pilvede varjutaja kiiruse mõõtmiseks lisatud külge Unitys sisseehitatud profiiler (*profiler*). Seal mõõdetakse skripti *CloudScript* videokaardi kasutuse aega. Testarvuti videokaart on NVIDIA GeForce GTX 1060 6GB ja protsessor on Intel® Core™ i5-6500. Mõõtmiseid tehti resolutsioonil 1920x1080 nii, et terve vaade oli kaetud pilvedega. Ajakulu tulemused on joonisel 41, kus x-teljel on kiirte marssimise sammude arv ja y-teljel on renderdamise aeg millisekundites. Erinevad pilvede resolutsioonid on joonisel erinevate värvidega. Sinise joonega on tähistatud pilvede renderdamine tekstuurile mõõtmetega 1920x1080 pikslit, punasega 960x540 pikslit, rohelisega 640x360 pikslit, mustaga 480x270 pikslit.



Joonis 41: Pilvede renderdamise ajakulu sõltuvus sammude arvust erinevatel pilvede resolutsioonidel.

8 Kokkuvõte

Käesolevas töös kirjeldati ja võrreldi erinevaid volumeetriliste pilvede renderdamise tehnoloogiaid ning valiti neist sobivaim, milleks oli pilvesüsteem Nubis. Sarnane volumeetriliste pilvede renderdamise lahendus implementeeriti ka mängumootoris Unity. See valik sobib töö eesmärgiga implementeerida realistlikud pilved ja Nubis süsteemist on kättesaadaval rohkem artikleid võrreldes teiste tehnoloogiatega.

Töös kirjeldati pilvede loomiseks kasutatud mürasid, 3D-tekstuuride genereerimist, pilve loomise algoritmi ja valgustamist. Renderdamise optimeerimisel kasutati stohhastilist sämplimist, pilvede renderdamist madalal resolutsioonil ja muid võtteid sämplimiste arvu vähendamiseks. Lisaks pilvede renderdamisele valmis ka protseduuriline ilma tekstuuri genereerimine kasutades Worley ja Simplex müra. Töös analüüsiti ka loodud pilve varjutaja ajakulu erinevatel kvaliteedi sätetel ühe kaadri arvutamiseks.

Lisaks implementatsioonile valmis ka demonstratiivne rakendus, mis sisaldab lihtsat stseeni koos kasutajaliidesega. Seal saab muuta pilvede välimust mõjutavaid parameetreid ja genereerida uusi ilma tekstuure.

Tulevikuarendustena oleks vaja juurde lisada funktsionaalsus pilvede varjude heitmiseks maapinnale. Lisaks ei luba praeguses lahendus paigutada kahte erinevat volumeetrilist pilve üksteise kohal, sest volumeetriliste pilvede kihil kasutatakse ainult ühte ilma tekstuuri. Üks lahendus võimalus on lisada teise pilvekihi kattuvuse info ilma tekstuuri alfa kanalisse. Edasi arendades võiks kasutada ka mitut erineva resolutsiooniga ilma tekstuuri, et parandada selle kordumine horisondil. Siis saaks kasutada päris maa raadiust. Praegu kasutatakse kõrge kihi pilvede renderdamiseks ühte pilve tekstuuri. Tulevikus võiks lisada juurde teise kõrgete pilvede tekstuuri, et seal oleks rohkem varjeerumist.

9 Viidatud kirjandus

[1] „Counter Strike: Global Offensive,“ [Võrgumaterjal]. http://blog.counter-strike.net/. [Kasutatud 13 mai 2018].

[2] „The Witcher 3: Wild Hunt,“ [Võrgumaterjal]. http://thewitcher.com/en/witcher3. [Kasutatud 13 mai 2018].

[3] „Horizon Dawn Zero,“ [Võrgumaterjal]. https://www.guerrilla-games.com/play/horizon. [Kasutatud 13 mai 2018].

[4] A. Schneider, „NUBIS: Authoring Real-Time Volumetric Cloudscapes With The Decima Engine,“ Guerilla Games, 31 juuli 2017. [Võrgumaterjal]. https://www.guerrilla-games.com/read/nubis-authoring-real-time-volumetric-cloudscapes-with-the-decima-engine. [Kasutatud 13 mai 2018].

[5] A. Toft, H. Bowles ja D. Zimmermann, „Optimisations for Real-Time Volumetric Cloudscapes,“ 2016. [Võrgumaterjal]. http://addymotion.com/files/optimisations-cloudscapes.pdf. [Kasutatud 13 mai 2018].

[6] J. Kamenik, „Mis on pilved ja kuidas need tekivad?,“ 04 september 2009. [Võrgumaterjal]. https://ilm.ee/index.php?46318. [Kasutatud 13 mai 2018].

[7] J. Kamenik, „Pilvede klassifitseerimine,“ 08 september 2009. [Võrgumaterjal]. https://ilm.ee/?46330. [Kasutatud 13 mai 2018].

[8] V. d. Bruyn, „Cloud types,“ [Võrgumaterjal]. http://legacy.sciencelearn.org.nz/Science-Stories/Navigating-Without-Instruments/Images/Cloud-types. [Kasutatud 13 mai 2018].

[9] M. Jürissaar, „Kiudrünkpilved,“ [Võrgumaterjal]. http://vana.loodusajakiri.ee/eesti_loodus/EL/vanaweb/9906/pilved.html. [Kasutatud 13 mai 2018].

[10] C. Thames, „Panoraamsed taeva tekstuurid,“ [Võrgumaterjal]. http://www.tutorialsforblender3d.com/Skybox/Page2. [Kasutatud 13 mai 2018].

[11] „PUBG,“ [Võrgumaterjal]. https://playbattlegrounds.com/main.pu. [Kasutatud 13 mai 2018].

[12] „Holdfast: Nations At War,“ [Võrgumaterjal]. http://www.holdfastgame.com/. [Kasutatud 13 mai 2018].

[13] „Reset,“ [Võrgumaterjal]. http://reset-game.net/. [Kasutatud 13 mai 2018].

[14] M. Kallinen, „In Praxis: Atmosphere,“ 19 detsember 2012. [Võrgumaterjal]. http://reset-game.net/?p=284. [Kasutatud 13 mai 2018].

[15] A. Schneider ja N. Vos, „The Real-Time Volumetric Cloudscapes Of Horizon Zero Dawn,“ Guerilla Games, 13 august 2015. [Võrgumaterjal]. https://www.guerrilla-games.com/read/the-real-time-volumetric-cloudscapes-of-horizon-zero-dawn. [Kasutatud 13 mai 2018].

[16] Electronic Arts, „Frostbite, The Engine,“ [Võrgumaterjal]. https://www.ea.com/frostbite/engine. [Kasutatud 13 mai 2018].

[17] S. Hillaire, „Physically Based Sky, Atmosphere and Cloud Rendering in Frostbite,“ 2016. [Võrgumaterjal]. https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/s2016-pbs-frostbite-sky-clouds-new.pdf. [Kasutatud 13 mai 2018].

[18] „The Witness,“ [Võrgumaterjal]. http://the-witness.net/news/about/. [Kasutatud 13 mai 2018].

[19] „The Art of the Witness - Clouds,“ [Võrgumaterjal]. http://www.artofluis.com/3d-work/the-art-of-the-witness/clouds/. [Kasutatud 13 mai 2018].

[20] B. Golus, „Anti-aliased Alpha Test: The Esoteric Alpha To Coverage,“ 12 august 2017. [Võrgumaterjal]. https://medium.com/@bgolus/anti-aliased-alpha-test-the-esoteric-alpha-to-coverage-8b177335ae4f. [Kasutatud 13 mai 2018].

[21] A. Schneider, „Real-Time Volumetric Cloudscapes,“ *GPU Pro 7*, CRC Press, 2016, pp. 97-127.

[22] D. R. Williams, „Earth Fact Sheet,“ [Võrgumaterjal]. https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html. [Kasutatud 13 mai 2018].

[23] „A Minimal Ray-Tracer: Rendering Simple Shapes (Sphere, Cube, Disk, Plane, etc.) - Ray-Sphere Intersection,“ [Võrgumaterjal]. https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection. [Kasutatud 13 mai 2018].

[24] „Perlin noise,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Perlin_noise. [Kasutatud 13 mai 2018].

[25] „Gradient noise,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Gradient_noise. [Kasutatud 13 mai 2018].

[26] „Value noise,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Value_noise. [Kasutatud 13 mai 2018].

[27] „Perlini müra pildid,“ [Võrgumaterjal]. https://stackoverflow.com/questions/36814120/improved-perlin-noise-too-smooth. [Kasutatud 13 mai 2018].

[28] S. Gustavson, „Procedural Textures in GLSL,“ *OpenGL Insights*, CRC Press, 2012, p. 113.

[29] R. Bridson, J. Houriham ja M. Nordenstam, „Curl-noise for procedural fluid flow,“ *ACM Trans. Graph.*, kd. 26, nr 3, July 2007.

[30] S. Hillaire, „Tileable Volume Noise,“ [Võrgumaterjal]. https://github.com/sebh/TileableVolumeNoise. [Kasutatud 13 mai 2018].

[31] P. Werner, „Intro to Curl Noise,“ 19 veebruar 2015. [Võrgumaterjal]. http://petewerner.blogspot.com.ee/2015/02/intro-to-curl-noise.html. [Kasutatud 13 mai 2018].

[32] „Wind gradient,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Wind_gradient. [Kasutatud 13 mai 2018].

[33] B. T. Phong, „Illumination for Computer Generated Pictures,“ 1975. [Võrgumaterjal]. http://www.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf. [Kasutatud 13 mai 2018].

[34] W. Jarosz, „Efficient Monte Carlo Methods for Light Transport in Scattering Media,“ 2008. [Võrgumaterjal]. https://cs.dartmouth.edu/~wjarosz/publications/dissertation/. [Kasutatud 13 mai 2018].

[35] A. Beer, *Bestimmung der Absorption des rothen Lichts in farbigen Flüssigkeiten (Determination of the Absorption of Red Light in Colored Liquids)*, 1852, pp. 78-88.

[36] D. Darling, „Forward scattering,“ [Võrgumaterjal]. http://www.daviddarling.info/encyclopedia/F/forward_scattering.html. [Kasutatud 13 mai 2018].

[37] L. G. Henyey ja J. L. Greenstein, „Diffuse radiation in the Galaxy,“ *Astrophysical Journal,* kd. 93, pp. 70-83, 1941.

[38] „Backscatter,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Backscatter. [Kasutatud 13 mai 2018].

[39] „Unity,“ [Võrgumaterjal]. https://unity3d.com/. [Kasutatud 13 mai 2018].

[40] „Unity - Multiplatform,“ [Võrgumaterjal]. https://unity3d.com/unity/features/multiplatform. [Kasutatud 13 mai 2018].

[41] „Documentation, Unity scripting languages and you,“ [Võrgumaterjal]. https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/. [Kasutatud 13 mai 2018].

[42] „Unity - Shader Reference,“ [Võrgumaterjal]. https://docs.unity3d.com/Manual/SL-Reference.html. [Kasutatud 13 mai 2018].

[43] „Unity - Compute shaders,“ [Võrgumaterjal]. https://docs.unity3d.com/Manual/ComputeShaders.html. [Kasutatud 13 mai 2018].

[44] „Shading Language used in Unity,“ [Võrgumaterjal]. https://docs.unity3d.com/Manual/SL-ShadingLanguage.html. [Kasutatud 13 mai 2018].

[45] „Unity - Writing Surface Shaders,“ [Võrgumaterjal]. https://docs.unity3d.com/Manual/SL-SurfaceShaders.html. [Kasutatud 13 mai 2018].

[46] „Unity - Store,“ [Võrgumaterjal]. https://store.unity.com/. [Kasutatud 13 mai 2018].

[47] „Unity Pro, Unity Plus and Unity Personal Software Additional Terms,“ [Võrgumaterjal]. https://unity3d.com/legal/terms-of-service/software. [Kasutatud 13 mai 2018].

[48] A. Biagioli, „Generic Raymarch Unity,“ [Võrgumaterjal]. https://github.com/Flafla2/Generic-Raymarch-Unity. [Kasutatud 13 mai 2018].

[49] A. Biagioli, „Raymarching Distance Fields: Concepts and Implementation in Unity,“ 1 oktoober 2016. [Võrgumaterjal]. http://flafla2.github.io/2016/10/01/raymarching.html. [Kasutatud 13 mai 2018].

[50] J. St-Amand, „Turbulence Library,“ [Võrgumaterjal]. https://assetstore.unity.com/packages/vfx/shaders/turbulence-library-3957. [Kasutatud 13 mai 2018].

[51] „2D / 3D / 4D optimised Perlin Noise Cg/HLSL library (cginc),“ [Võrgumaterjal]. https://forum.unity.com/threads/2d-3d-4d-optimised-perlin-noise-cg-hlsl-library-cginc.218372/. [Kasutatud 13 mai 2018].

[52] „Stochastic process,“ [Võrgumaterjal]. https://en.wikipedia.org/wiki/Stochastic_process. [Kasutatud 13 mai 2018].

[53] C. Peters, „Free blue noise textures,“ 22 detsember 2016. [Võrgumaterjal]. Free blue noise textures. [Kasutatud 13 mai 2018].

[54] „Temporal Reprojection Anti-Aliasing,“ [Võrgumaterjal]. https://github.com/playdeadgames/temporal. [Kasutatud 13 mai 2018].

[55] M. Gjoel ja M. Svendsen, „Low Complexity, High Fidelity: The Rendering of INSIDE,“ [Võrgumaterjal]. https://youtu.be/RdN06E6Xn9E?t=9m17s. [Kasutatud 13 mai 2018].

[56] „The Great White Clouds Today, and Cauliflowers,“ 1 juuli 2012. [Võrgumaterjal]. https://blog.abhranil.net/2012/07/01/the-great-white-clouds-today-and-cauliflowers/. [Kasutatud 13 mai 2018].

Lisad

I. Terminid

Renderdamine[1] (*rendering*) Renderdamine on arvutigraafikas 2D- või 3D-objektide teisendamine kahemõõtmeliseks pildiks, mida saab kuvada ekraanil.
Taevakuppel[2] (*skybox*) Meetod arvutimängudes teha maailm suuremaks, projekteerides panoraamse tekstuuri taevasse.
Optiline tihedus[3] (*optical density*) Optilisest tihedusest sõltub kui suur osa valgusenergiast materjali läbib. Suur optilise tiheduse väärtus osutub väikesele läbilaskvusele, väike optiline tihedus kõrgele läbilaskvusele.
Sakitõrje[4] (*anti-aliasing*) Arvutigraafikas geomeetria renderdamisel äärtes tekkinud sakkide eemaldamine.
Interpolatsioon[5] (*interpolation*) Interpolatsioon on teadaolevate punktide vahepealsete väärtuste arvutamine.
Kiirte marssimine[6] (*ray marching*) Volumeetriliste objektide renderdamise viis, kus iga piksli kohta saadetakse kiir stseeni.

[1] https://en.wikipedia.org/wiki/Rendering_(computer_graphics)

[2] https://en.wikipedia.org/wiki/Skybox_(video_games)

[3] https://et.wikipedia.org/wiki/Valgusfilter#Optiline_tihedus

[4] https://www.pcmag.com/encyclopedia/term/37810/anti-aliasing

[5] https://en.wikipedia.org/wiki/Interpolation

[6] https://et.wikipedia.org/wiki/Kiire_marssimine_ruumis

Kaamera tüvipüramiid[7] (*viewing frustum*) Kaamera tüvipüramiidiks nimetatakse seda osa stseenist, mis renderdatakse ekraanile.
Varjutaja[8] (*shader*) Videokaardil käivitatav programm, mis algselt loodi 3D-objektide varjutamiseks (värvi arvutamine valguse põhjal), kuid nüüd kasutatakse ka järeltöötluse efektide või muude arvutigraafikas kasutatavate eriefektide loomiseks.
Verteksvarjutaja[9] (*vertex shader*) Varjutaja, mida rakendatakse igale verteksile.
Fragmendivarjutaja[10] (*fragment shader*) Varjutaja, mida rakendatakse igale fragmendile.
Arvutusvarjutaja[11] (*compute shader*) Arvutusvarjutajat kasutatakse paralleelarvutusteks videokaardil.
Järeltöötluse efekt[12] (*post-processing effect*) Pärast geomeetria renderdamist, kuid enne pildi ekraanile kuvamist rakendatud efekt.
Ajaline sakitõrje[13] (*temporal anti-aliasing*) Sakitõrje liik, mis kasutab eelnevalt renderdatud kaadreid.
Sujutamine[14] (*blending*) Arvutigraafikas kahe või enama objekti pikslite värvi kombineerimine.

[7] https://www.pcmag.com/encyclopedia/term/61771/viewing-frustum

[8] https://en.wikipedia.org/wiki/Shader

[9] https://www.pcmag.com/encyclopedia/term/53754/vertex-shader

[10] https://www.khronos.org/opengl/wiki/Fragment_Shader

[11] https://www.khronos.org/opengl/wiki/Compute_Shader

[12] https://docs.unity3d.com/Manual/PostProcessingOverview.html

[13] https://en.wikipedia.org/wiki/Temporal_anti-aliasing

[14] http://www.vallaste.ee/sona.asp?Type=UserId&otsing=3084

Eelkorrutatud alfasujutamine[15] (*pre-multiplied alpha blending*) Sujutamise liik, kus tekstuuri alfa väärtus määrab, kui palju ta varjab tagumisi objekte.
Resolutsioon[16] (*resolution*) Pildi mõõtmed pikslites.
Profiiler[17] (*profiler*) Programmikoodi jõudlusanalüüsi instrument, mis kogub jõudlusandmeid.

[15] https://blogs.msdn.microsoft.com/shawnhar/2009/11/06/premultiplied-alpha/

[16] https://en.wikipedia.org/wiki/Image_resolution

[17] https://akit.cyber.ee/term/8115-profiler

II. Demorakendus

Demorakendus on tööga kaasas olevas ZIP-failis “demorakendus.zip”. Käivitamiseks paki lahti ZIP-fail ja ava “VolumetricCloudsDemo.exe”. Vali sobiv ekraani resolutsioon ja vajuta nuppu “Play!”. Süsteemi nõuded on 64-bitine Window ja soovitavalt eraldiseisev videokaart.

Demorakenduse klahvivajutused on kirjeldatud tabelis 1.

Tabel 1: Demorakenduse klahvide kirjeldused.

Klahv	**Kirjeldus**
W	Edasi liikumine
A	Vasakule liikumine
S	Tagasi liikumine
D	Paremale liikumine
Space	Hüppamine
Shift	Kiire liikumine
Ctrl	Aeglane liikumine
F	Lendamise alustamine/lõpetamine
Q	Alla liikumine (lennates)
E	Üles liikumine (lennates)
Esc	Hiire lukustamine 3D maailma või menüüle

III. Lähtekood

Lähtekood on kättesaadaval GitHubis aadressil: https://github.com/jaagupku/volumetric-clouds

Lähtekoodi käivitamiseks on vaja mängumootorit Unity (testitud versioonil 2018.1.0f2).

IV. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Jaagup Kuhi**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   Volumeetriliste pilvede renderdamine,

   mille juhendaja on Jaanus Jaggo,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.05.2018**