TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Einar Linde

Delta õppehoone visualisatsioon – visuaalsed efektid

Bakalaureusetöö (9 EAP)

Juhendaja: Raimond-Hendrik Tunnel, MSc

Tartu 2019

Delta õppehoone visualisatsioon – visuaalsed efektid

Lühikokkuvõte:

Käesolev töö kirjeldab aastal 2017 alanud Delta õppehoone visualisatsiooni projekti täiustamist. Töö käigus optimeeriti visualisatsiooni valgustust ja loodi uusi ilmastiku efekte. Võrreldakse Unity mängumootori erinevaid valgustamise võimalusi. Lisati igasse hoone õpperuumi oma valgustus, et oleks võimalik valgust sisse-välja lülitada. Loodi uued lume ja vihma efektid kasutades varjutajat. Töö lõppeb visualisatsiooni jõudluse testimisega ja visauaalste efektide testimisega kasutajate peal.

Võtmesõnad:

Arvutigraafika, ilmastik, Unity, valgustus, varjutaja, visuaalsed efektid, visualisatsioon

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

Delta Building Visualization – Visual Effects

Abstract:

This thesis is improvement to Delta building visualization project, which started in 2017. During work for this thesis the visualization lighting and weather effects was optimized. The thesis compares different lighting strategies in Unity game engine. Lighting to every room was added such that light is toggleable. The thesis describes the process to make improved snow and rain effects using shaders. The thesis concludes with performance testing and visual testing on users.

Keywords:

Computer graphics, weather, Unity, lighting, shader, visual effects, visualization

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

Sisukord

1. Sissejuhatus

Delta õppehoone on Tartu Ülikooli uus õppehoone, mis valmib 2020 aasta alguses. Uude õppehoonesse kolivad majandusteaduskond, arvutiteaduste instituut ning matemaatika ja statistika instituut. Maja on väga suur ning esimesel korrusel on 50 õpperuumi ja teisel korrusel 73 õpperuumi. Õppehoones toimuva tegevuse ja hoonet külastavate inimeste inspireerimiseks otsustati luua maja fuajeesse visualisatsioon, mille pealt saab õppehoonet külastav inimene vaadata õppehoones olevate inimeste liikumist, Tartu hetke ilma, hetkel käivaid tunde ning klasside asukohti. Esimese versiooni visualisatsioonist tegid tudengid 2017/2018 õppeaastal bakalaureusetööna.

Antud töö eesmärk on optimeerida ja täiustada A. Nikolajevi ja A. Voitenko poolt alustatud Delta õppehoone visualisatsiooni (DBV) rakendust. A. Nikolajevi bakalaureusetöö keskendub mudeli loomisele ja inimeste liikumisele visualisatsioonis. Täpsemalt saab lugeda A. Nikolajevi bakalaureusetööst „Delta õppehoone visualiseerimine ja optimeerimine“ [1]. A. Voitenko bakalaureusetöö kirjeldab liidestust Cumulocity süsteemiga, mis võimaldab hallata majas olevatest anduritest tulevat informatsiooni ja keskkonna visualiseerimist. Täpsemalt saab lugeda A. Voitenko bakalaureusetööst „Delta õppehoone keskkonna visualiseerimine“ [2]. Edaspidi viidatakse neile kui DBV eelmised arendajad.

Käesoleva töö eesmärgiks on kirjeldada protsessi, mille käigus optimeeriti Delta hoone visualisatsioon, parandati ja täiustati hoone mudelit ning loodi uued visuaalsed efektid. Uuteks efektideks on realistlikumad vihma ja lume efektid ning valgustuse lisamine ruumidesse.

Käesoleval aastal täiustatakse Delta õppehoone visualisatsiooni paralleelselt kolme tudengi poolt. Meelis Perli poolt tehtav bakalaureusetöö “Delta õppehoone visualisatsioon – agentide loogika”[3], mis keskendus inimeste visualiseerimisele õppehoones, Daniel Kütti bakalaureusetöö “Delta õppehoone visualisatsioon – administaratiivne tööriist”[4], mille raames loodi visualisatsioonile tööriist DBV kontrollimiseks üle interneti, ja käesolev töö.

Töö on jaotatud kuute ossa. Peatükis 2 kirjeldatakse eelmiste DBV arendajate tööd, mis on aluseks käesolevale tööle, ja nende poolt loodud rakenduse jõudlust. Peatükis 3 kirjeldatakse nõudeid, mis seadis käesoleva töö autor valminud rakendusele. Peatükk 4 kirjeldab olemasoleva rakenduse optimiseerimist. Peatükk 5 kirjeldab Delta õppehoone ruumidesse valgustuse lisamise protsessi. Peatükk 6 kirjeldab vihma ja lume efektide loomist. Peatükk 7 kirjeldab loodud visuaalide testimist kasutajate peal ja jõudlusteste. Töös kasutatud terminid leiab lisast I ja informatsiooni kaasapandud failide kohta lisast II.

2. Eelnevad tööd

Aleksander Nikolajevi ja Andrei Voitenko loodud visualisatsioon on tehtud Unity mängumootoris[1] ning kasutades programmeerimiskeelt C#. Eelnevate arendajate bakalaureusetöödest [1,2] on kirjeldatud järgmisi protsesse: õppehoone mudeli loomist, õppehoones liikuvate inimeste visualiseerimist ja keskkonna visualiseerimist. Nikolajevi töös [1] kirjutatakse järgmist: Õppehoone mudel on tehtud jooniste järgi, mis on saadud firmalt Arhitekt11 OÜ. Mudelis on puudu maja seintele tulevad kuldsed triibud, mis jäeti välja visualisatsiooni parema jõudluse saavutamiseks. Mööbel, mis on pandud mudelisse, on tehtud ilma joonisteta ning ruumide mahutavus ei vasta plaanitud mahutavusele. Voitenko töös [2] kirjutatakse järgmist: Informatsioon ilma kohta saadakse ilmateeenuse OpenWeatherMap[2] rakendusliidesest. Selle informatsiooni põhjal visualiseeriti järgmised keskkonna nähtused: lumi, vihm, pilvisus, äike ning ööpäeva tsükkel. Sademete jaoks kasutatakse Unity osakeste süsteemi[3] (*particle system*) ning maapinna visualiseerimiseks rakenduses Blender[4] loodud objekte ja Unity animeerimissüsteemi[5]. Ööpäeva tsükkel saavutatakse suundvalgusallika suuna muutmisega vastavalt reaalse päikese asukohale Delta õppehoone suhtes ning valgusallika värvi muutmisega. Õppehoone ruume valgustati ööpäeva tsükli loomiseks kasutatud suundvalgusallikat.

2.1 Visualisatsiooni jõudlus

Eelmiste arendajate tehtud töö jõudluse võrdlemiseks käesoleva töö tulemusega, mõõdeti nende tehtud töö jõudlus uuesti üle. Mõõtmised tehti uuesti, sest kahe töö tulemuste võrdlemiseks on tarvilik kasutada samade spetsifikatsioonide arvutit. Jõudluse mõõtmiseks kasutati lauaarvutit järgmise spesifikatsioonidega:

Operatsioonisüsteem: Windows 10 Pro

Protsessor: Intel(R) Core(TM) i5-6400 CPU @ 2.70 GHz

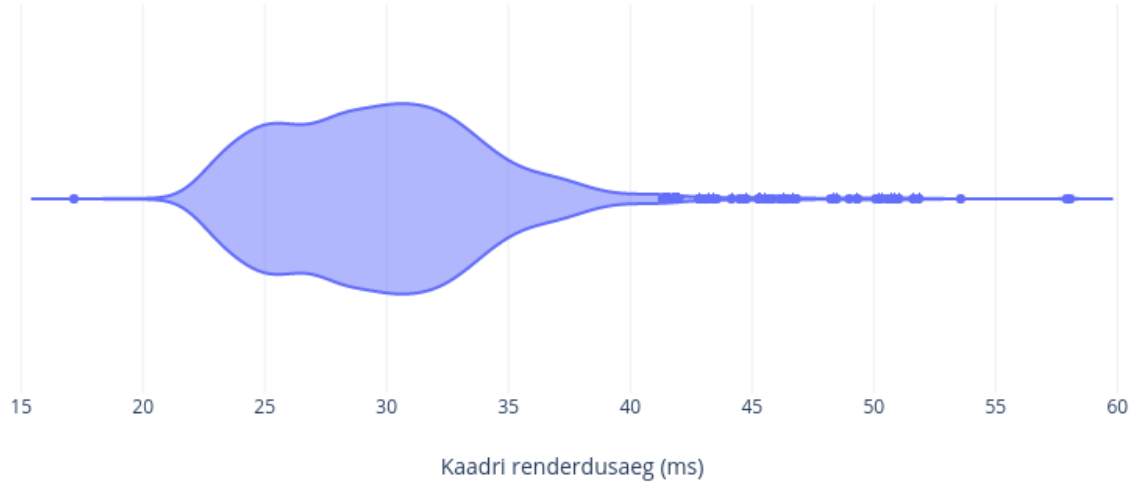
Graafikakaart: Nvidia GeForce GTX 1060, 6 GB RAM

Muutmälu: 8 GB

Ekraan: LG 24MP48HQ resolutsioonil 1920 × 1080

[1] https://unity.com/
[2] https://openweathermap.org/
[3] https://docs.unity3d.com/Manual/ParticleSystems.html
[4] https://www.blender.org/
[5] https://docs.unity3d.com/Manual/AnimationOverview.html

Mõõtmiste tegemiseks kasutati visualisatsiooni eraldiseisvat versiooni (*build*), sest Unity redaktoris lisandub visualisatsiooni jõudlusele veel Unity enda töötamiseks vajav jõud. Mõõtmiseks kasutati rakendust Fraps[6]. Fraps on rakendus, millega on võimalik mõõta rakenduste kaadrisagedust ja ühe kaadri renderdamiseks kuluvat aega[6]. Mõõdetavateks suurusteks oli kaadri renderdamiseks kuluv aeg ja kaadrite arv sekundis (FPS). Visualisatsioon käivitamisel valiti graafika kvaliteedi seadeks kiireim (*fastest*). Visualisatsioon käivitati maksimaalse inimeste arvuga, mis on 1471, sest paljudes ruumides on kohti vähem kui plaani peal. Inimesed hakkasid määratud alguspunktidest liikuma klassiruumide poole. Mõõtmise kestvuseks valiti kaks minuti, sest selle ajaga jõuvad inimesed alguspunktist klassiruumi ja igat vaadet visualiseeritakse täpselt 4 korda. Testimise tulemusena saadi mediaan renderdusajaks 29.66 millisekundit (vt. Illustratsioon 1). Arvutatuna kaadreid sekundis on see 33.7 FPS. Maksimaalne renderdusaeg oli 58.05 ms ehk 17.2 FPS ning minimaalne renderdusaeg 17.17 ms ehk 58.2 FPS.



Illustratsioon 1. Visualisatsiooni renderdusaeg

Muutmälu (RAM) kasutuse mõõtmiseks kasutati Unity mängumootori tööriista nimega Profiler[7]. Profiler on Unity redaktorisse sisseehitatud tööriist rakenduse jõudluse mõõtmiseks[7]. Profiler võimaldab uurida, kui palju kasutab rakenduse erinevad osad mälu, protsessori aega, graafika protsessori aega jne[7]. Saadud info põhjal on võimalik otsustada, milline koht rakendusest vajab optimeerimist kõige rohkem. Visualisatsiooni sätted jäid samaks nagu kasutades rakendust Fraps ja mõõdeti samuti eraldiseisvast versioonist. Mõõdeti kahe minuti jooksul muutmälu kasutust. Tulemuseks saadi, et visualisatsioon kasutab kokku keskmiselt 510 MB muutmälu.

[6] http://www.fraps.com/
[7] https://unity3d.com/learn/tutorials/temas/performance-optimization/profiler-window

3. Nõuded

Selles peatükis on Delta õppehoone visualisatsiooni funktsionaalsed ja mittefunktsionaalsed nõuded, mis on koostatud lõputöö eesmärkide põhjal.

3.1 Mittefunktsionaalsed nõuded

Selles alapeatükis on toodud mittefunktsionaalsed nõuded Delta õppehoone visualisatsioonile. Nõuete defineerimiseks kasutati peatükis 2.1 saadud testimise tulemusi.

MF1: Visualisatsiooni mediaan kaadri renderdamiseks kuluv aeg ei ole suurem kui eelnevalt olemasoleva rakenduse mediaan kaadri renderdamisaeg (29.66 ms).

MF2: Visualisatsiooni maksimaalne kaadri renderdusaeg ei ole suurem kui eelnevalt olemasoleva rakenduse maksimaalne renderdusaeg (58.05 ms).

MF3: Visualisatsiooni muutmälu kasutuses ei esine lekkeid.

MF5: Visualisatsiooni ruumide valguse sisse-välja lülitamiseks kuluv renderdamisaeg ei ole suurem kui eelnevalt olemasoleva rakenduse kaadri renderdamisaeg (29.66ms).

MF6: Visualisatsiooni lume efekti renderdamine ei ole suurem kui eelnevalt olemasoleva rakenduse kaadri renderdamisaeg (29.66ms).

MF7: Visualisatsiooni vihma efekti renderdamine ei ole suurem kui eelnevalt olemasoleva rakenduse kaadri renderdamisaeg (29.66ms).

3.2 Funktsionaalsed nõuded

Funktsionaalsete nõuded peatükis on toodud nõuded visualisatsiooni visuaalsetele efektidele.

F1: Valgustus visualisatsiooni õpperuumides on välja lülitatud, kui vastavas ruumis ei ole ühtegi inimest.

F2 Valgustus visualisatsiooni õpperuumis on sisse lülitatud, kui selles ruumis on inimesi.

F3: Visualisatsiooni maapinnale tekib lumi, kui lund sajab.

F4: Visualisatsiooni maapinnal olevale lumel on mudased teerajad.

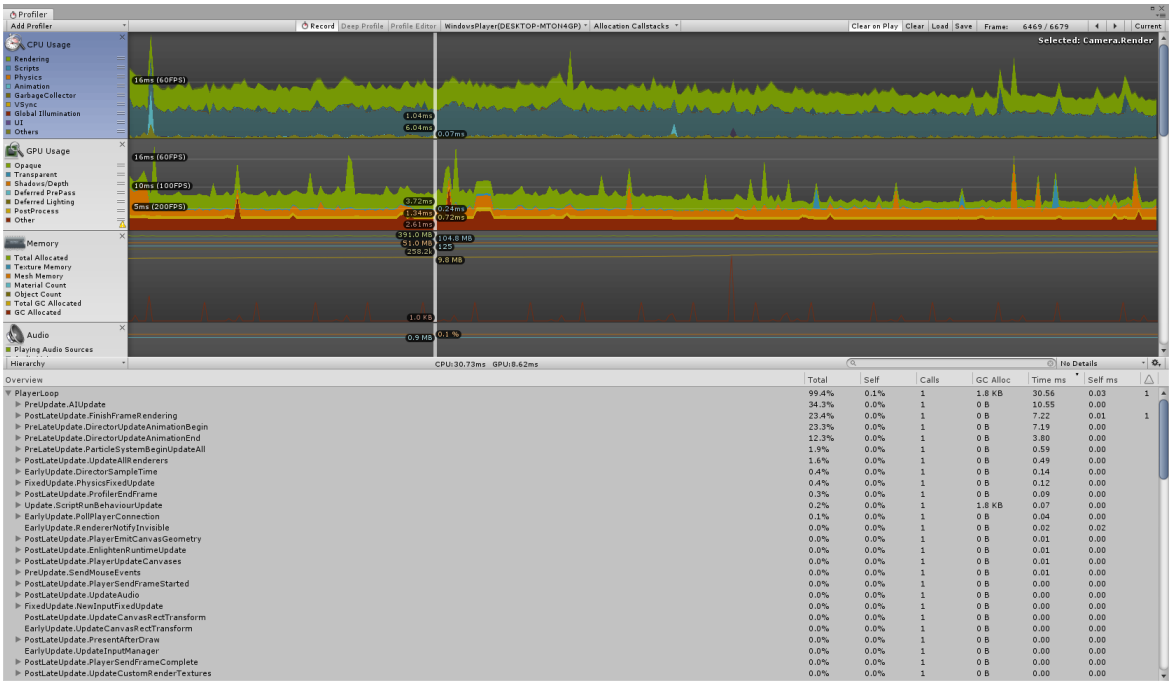
F5: Visualisatsiooni maapind muutub visuaalselt märjaks, kui vihma sajab.

F6: Visualisatsiooni maapinnale tekivad lombid, kui vihma sajab.

F7: Visualisatsiooni maapinnal olevate lompide peale tekivad vihmapiisad, kui vihma sajab.

4. Optimeerimine

Selles peatükis kirjeldatakse DBV eelmiste arendajate töö optimeerimist. Esimese asjana kellegi teise rakenduse täiustamiseks on tarvilik esmalt rakendusega tutvuda ning leida kohad, mis vajavad parandamist või optimeerimist. Rakenduse optimeerimine lihtsustab uute visuaalsete efektide lisamist, sest peale optimeerimist ollakse rakendusega tuttav. Optimeerimine parandab rakenduse renderdusaega ja seetõttu on võimalik lisada rakendusse uusi featuure. Probleemsete kohtade üles leidmiseks kasutati Unity Profiler tööriista. Profiler tööriist võimaldab uurida rakenduse sisemust ning teha kindlaks optimeerimist vajavad kohad.



Illustratsioon 2. Profilier tööriist

Üheks suurimaks protsessori aja kulutajaks oli inimeste teeradade arvutamine (vt. Illustratsioon 2). Selle optimisieerimise kohta saab lugeda Meelis Perli bakalaureusetööst [3]. Järgmiseks suureks protsessori aja kulutajaks oli visualisatsiooni vaadete renderdamine. Visualisatsioonis on kuus vaadet, mida kasutajale näidatakse ning vahetatakse iga 5 sekundi tagant. Viimaseks suureks aja kulutajaks oli animatsioonide taasesitamine. DBV visualisatsioon kasutab animatsioonide taasesitamiseks Unity komponenti Animator[8]. Inimeste visualiseerivate agentide animatsioonid oli pandud objektide külge nagu näiteks toolid ja lauad. Seepärast töötasid animatsioonid koguaeg. Selle parandamiseks tuleks panna animatsioonid ainult agentide külge, kuna agentide arv on muutuv ning harva tekib olukord, kui

[8] https://docs.unity3d.com/ScriptReference/Animator.html

hoone on inimestest täis. Animatsioonide optimeerimist käesolevas töös ei tehtud, kuna jäi antud töö mahust välja.

Peale põhjaliku Profilier tööriista uurimist selgus, et üks vaadete renderdamisaja mõjutajaks oli järeltöötlusefektid[9] (*post processing effects*). Järeltöötlusefekte kasutatakse, et lisada vaatele erinevaid filtreid. Visualisatsioonis kasutati järeltöötlusefektide paketi esimest versiooni[10]. Parandamiseks asendati paketi see järeltöötlusefektide paketi teise versiooniga[11].

4.1 Renderdamisviis

Delta õppehoone visualisatsiooni parema jõudluse saavutamiseks on oluline valida stseeni ekraanile visualiseerimiseks õige tehnika. Stseeni ekraanile visualiseerimiseks on Unity mängumootoris kaks põhilist viisi: edaspidine renderdamine[12] (*forward rendering*, FR) ja edasilükatud renderdamine[13] (*deferred rendering*, DR).

Unity mängumootori dokumentatsioonis[14] on edaspidine renderdamise kohta öeldud järgmist. Edaspidine renderdamine on levinuim visualiseerimismeetod mängumootorites. FR eeliseks on see, et ekraanile on võimalik renderdada objekte kiiresti. Lisaks võimaldab kasutada erinevaid kohaldatuid varjutamise tehnikaid ja riistvarapoolt võimaldatavaid tehnikaid. Näiteks *multi-sample anti-aliasing*[15] (MSAA) on võimalik kasutada FR korral, aga ei ole võimalik kasutada DR-ga[12]. Owens kirjutab oma artiklis [5], et FR suureks puuduseks on selle jõudlus paljude valgusallikate korral. Valgustuse visualiseerimiseks tehakse keerulised valgusarvutused iga objekti fragmendile stseenis. Fragmendiks nimetatakse informatsiooni, mis on vajalik piksli genereerimiseks[16]. Fragment jõuab ekraanile, kui teda sügavustestide ajal ei eemaldata. StatCounter Global Stats[17] andmetel on enim kasutatud ekraani resolutsioon 1366×768. See teeb üle miljoni piksli (täpsemalt 1 049 088 pikslit), mis tuleb igal kaadril uuesti arvutada. Fragmentide sügavustestimine tehakse peale valgusarvutusi. Sügavusteste mitte läbinud fragmente ei kasutata piksli arvutamisel ja nende jaoks tehtud arvutused lähevad raisku. Kasutades FR visualiseeritakse stseen iga valgusallika korral uuesti. Valgusarvutuste keerukus on seega:

[9] https://docs.unity3d.com/Manual/PostProcessingOverview.html
[10] https://assetstore.unity.com/packages/essentials/post-processing-stack-83912
[11] https://github.com/Unity-Technologies/PostProcessing/wiki
[12] https://docs.unity3d.com/Manual/RenderTech-ForwardRendering.html
[13] https://docs.unity3d.com/Manual/RenderTech-DeferredShading.html
[14] https://unity3d.com/learn/tutorials/topics/graphics/choosing-rendering-path
[15] https://www.khronos.org/opengl/wiki/Multisampling
[16] https://enacademic.com/dic.nsf/enwiki/2473503
[17] http://gs.statcounter.com/screen-resolution-stats/desktop/worldwide

$$O(geomeetria\ fragmentide\ arv \cdot valgusallikate\ arv) \quad (1)$$

Edasilükatud renderdamine kohta kirjutab Owens järgmist: DR vähendab arvutamisel osalevate fragmentide hulka, kasutades arvutuste tegemisel ekraani resolutsiooni kogu fragmentide hulga asemel. DR arvutuste keerukus on:

$$O(ekraani\ resolutsioon \cdot valgusallikate\ arv) \quad (2)$$

Eelnevat informatsiooni arvesse võttes on DR parem valik kui stseenis on rohkelt valgusallikaid, kuid Unity mängumootoris ei saa DR kasutada ortograafilise kaamera[18] (*orthographic camera*) korral. DF kasutamine takistab veel riistvara poolt võimaldatavate tehnikate kasutamist. Kuna Delta õppehoone visualisatsioon kasutab stseeni visualiseerimiseks ortograafilist kaamerat, ei ole võimalik kasutada edasilükatud renderdamist. Stseeni visualiseerimiseks jäädi käesoleva töö raames kasutama edaspidist renderdamist.

4.2 Rakenduse kood

Rakenduse koodi kirjutades on oluline, et kirjutatud kood oleks „puhas“. Puhtaks koodiks nimetatakse koodi, mis on arusaadav, meeldiv lugeda ning sellese saab lihtsa vaevaga uuendusi teha [6]. Kuna Delta õppehoone visualisatsiooni projekti arendatakse mitme inimese poolt korraga on siin eriti tähtis, et “musta” koodi leiduks võimalikult vähe. „Must“ kood raskendab arendajate tööd, tehes seda aeglasemaks ja frustreerivamaks. Ükski rakendus ei ole „musta“ koodita. Klassis `WeatherLoad`, mille eesmärk oli uuendada ilmastiku visuaale rakendusliideselt saadud informatsiooni põhjal, leidus rohkelt „musta“ koodi. Klassi peamine meetod oli 242 rida pikk. See viitab koodi kordustele ning, et meetodit kasutati mitme ülesande jaoks. Koodi kordused koodis on halvad, sest see viitab abstraktsiooni kasutamata jätmisele [6]. Ühel meetodil peaks olema üks otstarve, et rakenduse kood oleks kergesti loetav [6]. Klassi `WeatherLoad` peamist meetodit uuendati iga kaader, kuigi ilma rakendusliideselt saadi andmeid iga kümne sekundi tagant. Järgnevalt on näha lõiku koodist, mis uuendas ilma äikese korral.

```
1.  if (weather.weather[0].id >= 200 && weather.weather[0].id <= 232)
2.  {
3.   muudatused äikese puhul
4.   if (!Rain)
5.   {
6.    coroutine = StartParticle(2400, 300, rainPart);
7.    StartCoroutine(coroutine);
8.   }
9.   if (Snow)
10.  {
11.   coroutine = StopAndStart(snowPart, rainPart, 2400);
```

[18] https://docs.unity3d.com/ScriptReference/Camera-orthographic.html

```
12.   StartCoroutine(coroutine);
13.   if (SnowHills)
14.   {
15.    LerpController lc = new LerpController();
16.    coroutine = lc.LerpColor_toGrass(current, grass);
17.    StartCoroutine(coroutine);
18.    snowGrass = false;
19.    ObjectController ob = new ObjectController();
20.    coroutine = ob.deleteSnowHills(snowHillsObj, snowHillsAnim, 1000);
21.    StartCoroutine(coroutine);
22.    SnowHills = false;
23.   }
24.  }
25.  if (!puddlesBool)
26.  {
27.   ObjectController ob = new ObjectController();
28.   coroutine = ob.makePuddles(puddles, puddleAnim, 500);
29.   StartCoroutine(coroutine);
30.   puddlesBool = true;
31.  }
32.  lightningBool = true;
33.  StartCoroutine("lightning");
34. }
```

Sarnast koodijuppi kasutati iga ilma efekti korral. Koodi korduste ja meetodile ühe ülesande andmiseks, tehti igale ilma efektile oma meetod.

Klassis `WeatherLoad` kasutati paljusid klassi muutujaid tõeväärtustüübiga. Järgnevast koodi jupist on näha kasutatud tõeväärtustüübiga väljasid.

```
1. bool first = true;
2. bool isNetwork = false;
3. public bool Rain = false;
4. public bool Snow = false;
5. public bool lightningBool = false;
6. public bool SnowHills = false;
7. public bool RainPuddles = false;
8. public bool puddlesBool = false;
9. public bool snowGrass = false;
```

Neid muutujaid muudetakse mitmes erinevas meetodis, mis tegi nende otstarbest arusaamise väga raskeks. Klassi `WeateherLoad` ülesandeks on kontrollida ilma visualiseerimist ning sellel ei ole vaja teada hetke visualisatsiooni maapinna olukorda. Seetõttu tehti uus komponent nimega `WeatherMaterial`, mis kontrollib maapinna visualiseerimist. Selle tulemusena sai klassist eemaldada kõik tõeväärtus muutujad, tehes koodi loetavamaks.

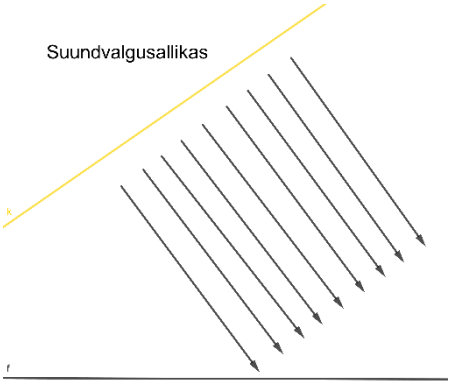
Illustratsioon 3. Ilma ikoon

Visualisatsioonis näidatakse hetke ilma ning selle kõrval ikooni ilmast (vt. Illustratsioon 3). Ikoon saavutati uue komponendi lisamisega kasutajaliidesesse. Komponent lisati iga ilma uuenduse korral ning eelmist komponenti ei kustutatud. See tekitab rakenduse pikka töötamise ajal probleeme, sest komponentide arv läheb iga ilma uuendusega suuremaks ning hakkab jõudlust mõjutama. Selle parandamiseks viidi sisse uuendus, mille tulemusena muudetakse ilma uuenduse korral juba stseenis olevat `WeatherImage` komponent, et vahetada ilma pilti.

5. Valgustus

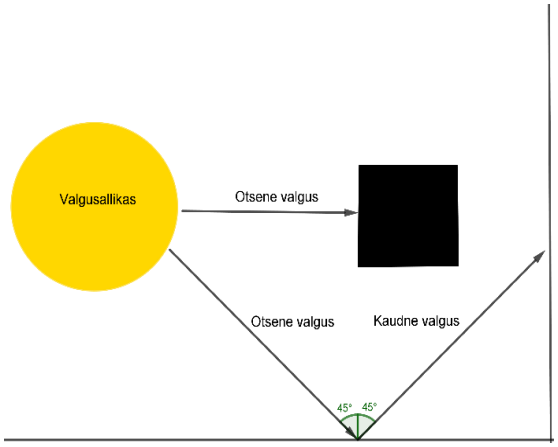
Käesolevas peatükis võrreldakse Unity mängumootori poolt pakutavaid võimalusi ning kirjeldatakse valgustuse implementeerimist Delta õppehoone visualisatsioonis.

DBV eelmised arendajate töö kasutas sise- ja välisvalgustuse visualiseerimiseks ühte suundvalgusallikat[19] (*directional light*) [1,2]. Suundvalgusallikas simuleerib valgusallikat, mis asub lõpmata kaugel ning mille kiired on üksteise suhtes paralleelsed (vt. Illustratsioon 4)[19]. Selle eeliseks on võimalus valgustada stseeni kõiki objekte samamoodi. Valguskiired langevad objektile sama nurga all olenemata valgusallika asukohast. Seepärast kasutatakse suundvalgusallikat kõige sagedamini päikese ja kuu valguse kujutamiseks.



Illustratsioon 4. Suundvalgusallikas

Unity mängumootoris jaotatakse valgustehnikad kahte suurte gruppi: reaalaja valgustus (*realtime lighting*) ja eelarvutatud valgustus (*precomputed lighting*). Reaalaja valgustus arvestab valgustuse arvutamisel ainult otsest valgust (*direct light*) ning see arvutatakse igal kaadril uuesti. Otseseks valguseks nimetatakse valgust, mis tuleb otse valgusallikast (vt. Illustratsioon 5). Kasutades stseeni valgustamiseks ainult reaalaja valgusallikat paistavad täisvarjud ühtlaselt musta/hallina, sest reaalaja valgustus arvestab varjude arvutamisel ainult globaalselt valgust. Kaudseks valguseks nimetatakse teistelt objektidelt peegelduvat valgust (vt. Illustratsioon 5). [7,8]



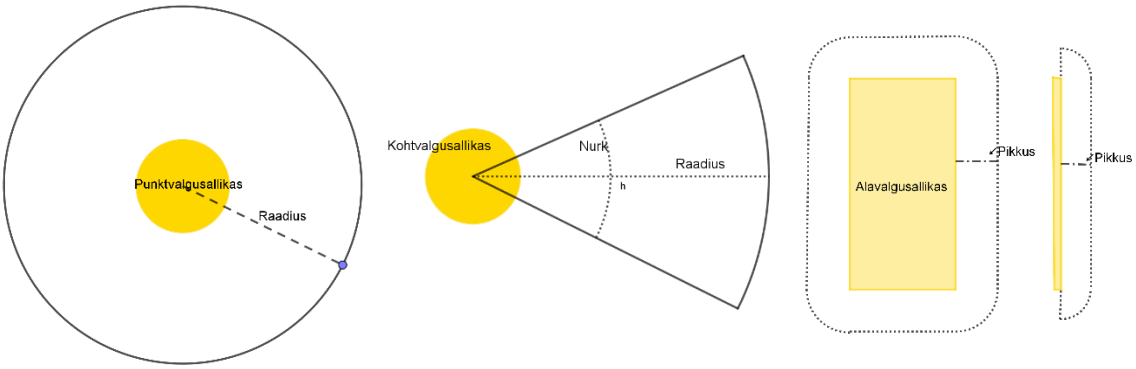
Illustratsioon 5. Otsene ja kaudne valgus

Eelarvutatud valgus ehk küpsetatud valgus (*baked lighting*) arvutatakse enne rakenduse kompileerimist valgustekstuuridesse ning rakenduse töötamise ajal dekodeeritakse ja näidatakse kasutajale. Eelnevalt arvutatud valgus jaguneb veel omakorda kaheks: eelarvutatud reaalaja globaalne valgustus (*precomputed realtime global illumination*, reaalaja GI) ja küpsetatud globaalne valgustus (*baked global illumination*, küpsetatud GI). Reaalaja ja küpsetatud GI erinevus seisneb selles, et küpsetatud GI korral on

[19] https://docs.unity3d.com/Manual/Lighting.html

valgustekstuurid staatilised ehk neid ei ole võimalik muuta rakenduse töötamise ajal. Reaalaja GI korral on arvutatakse valgustekstuuridesse osaline informatsioon stseeni geomeetriast ning rakenduse töötamise ajal arvutatakse lõplik valgustus, mis kasutajale kuvatakse. See annab võimaluse valgust rakenduse töötamise ajal muuta. Näiteks võimaldab see valgust sisse-välja lülitada, valguse värvi ja intensiivsust muuta. Mõlema GI tüüpide valgustektuuride arvutamisel arvestatakse nii otsest valgust kui ka kaudset valgust. [7,8]

Unity dokumentatsioonis kirjeldatakse kuute valgusallikat. Nendeks on (eelnevalt nimetatud suundvalgusallikas), punktvalgusallikas (*point light*), kohtvalgusallikas (*spot light*), alavalgusallikas (*area light*), ümbritsev valgus (*ambient light*) ja valgust kiirgav materjal (*emissive material*). Punktvalgusallikas (vt. Illustratsioon 6) valgustab ümbritsevat ala saates valguskiired ühest punktist igasse suunda. Valguse mõju tugevus sõltub objekti kaugusest allika asukohani. Kohtvalgusallikas (vt. Illustratsioon 6) töötab sarnaselt punktvalgusallikale ainult, et saadab valguskiired välja stseeni suhtes nurga määratud koonuse sees. Alavalgusallikas (vt. Illustratsioon 6) on ristkülikukujuline ning kiirgab valgust tervelt pinnalt, kuid selle suurimaks probleemiks on see, et seda on võimalik kasutada ainult küpsetatud GI-ga. Järelikult puudub võimalus seda valgusallikat sisse-välja lülitada vastavalt kasutaja soovidele. Ümbritseval valgusel ei ole stseenis kindlat asukohta. Nimetatud allikaga tõstetakse terve stseeni valgustus taset. Viimaks on veel olemas valgust kiirgav materjal. Valgust kiirgav materjal valgustab ümbritsevat ala sarnaselt ala valgusallikale kiirates valgust tervelt pinnalt, kuid seda on võimalik korrigeerida sarnaselt teistele valgusallikatele rakenduse töötamise ajal muutes valguse intensiivsust, värvi jne.[20]



Illustratsioon 6. Valgusallikad:
Punktvalgusallikas (vasakul), kohtvalgusallikas (keskel) ja alavalgusallikas (paremal)

[20] https://docs.unity3d.com/Manual/Lighting.html

Küpsetatud valguse töötamiseks on tarvilik, et objekt oleks staatiline. Staatilist objekti ei ole võimalik stseenis liigutada ega pöörata. Objekti, mille omadused nagu asukoht ja pööre peavad rakenduse töötamise aeg muutuma, nimetatakse dünaamiliseks. Dünaamilisi objekte ei saa kasutada küpsetatud valgusega. Selleks, et kasutada dünaamilisi objekte küpsetatud valgusega, on Unity mängumootoris valgussondi grupp (*light probe group*). Valgussondi grupp koosneb mitmest sondist. Iga sond paigutatakse stseenis arendaja valitud kohale ning salvestatakse selle asukoha valguse informatsioon. Selle informatsiooni kasutamine rakenduse töötamise aeg nõuab vähe jõudlust. Informatsioon dekodeeritakse objektidele kasutades sfäärilise harmoonia (*spherical harmonics*) matemaatilisi funktsioone. Sfäärilised harmooniad on kera pinnal defineeritud funktsioonid, mida kasutatakse tihti osaliste diferentsiaalide lahendamiseks[21] [8,9]

5.1 Sisevalgustus

Delta õppehoones asub esimesel ja teisel korrusel kokku 123 õpperuumi. Iga ruumi valgustuse muutmiseks on tarvilik, et igas ruumis on oma valgusallikas. Sobiva valgusallika valikul sai määravaks allika mõju visualisatsiooni jõudlusele ning võimalus valgust välja lülitada vastavalt soovile. Peatüki alguses on toodud kaks võimalust, millal saab valgust sisse-välja lülitada vastavalt kasutaja soovidele. Esimeseks variandiks on reaalaja valguse kasutamine. Reaalaja valgus nõuab palju renderdusaega. Kasutades seda varianti ei ole võimalik saavutada mittefunktsioonaalset nõuet MF1. Teiseks variandiks on kasutada eelarvutatud reaalaja globaalset valgustust. Reaalaja GI korral on võimalik muuta valguse intensiivsust ning lülitada valgust sisse-välja rakenduse töötamise ajal. Valgusallikaks valiti valgust kiirgav materjal, sest see võimaldab valgusallika kuju ise määrata.

Üks esimestest probleemidest oli õige valgusallika kuju valimine. Valgusallika kujuks oli kaks erinevat võimalust. Potentsiaalseks lahenduseks oli kindla suurusega valgusallikas, mille suurus ei sõltu ruumist. Sellise valgusallika puhul esines mitmeid probleeme. Suurimaks probleemiks oli valgustuse ebaühtlus ruumide lõikes. Selle põhjuseks on, et hoones on palju erineva suuruse ja kujuga ruume ning valgus ei jõudnud igasse toa nurka. Ruumid, mis olid väiksemad ja ruudukujulised olid heledamad ning suured ja rohkemate nurkadega

[21] https://en.wikipedia.org/wiki/Spherical_harmonics

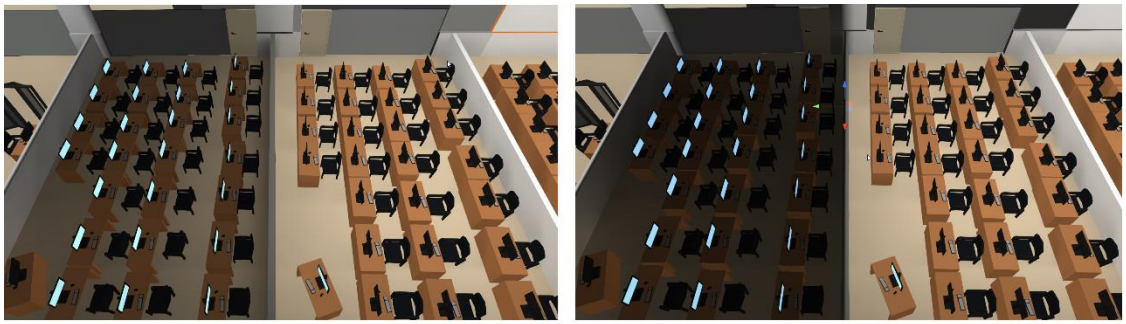
ruumid poolenisti valgustatud. Illustratsioon 7 kujutab olukorda kui valgusallikas on Unity tasand ja väiksem kui ruumi pindala.

Illustratsioon 7. Kindla suurusega valgusallikas

Järgmiseks variandiks oli teha ruumi suurune valgusti (vt. Illustratsioon 8). Sellega sai kaotatud ära ruumide valgustuse ebaühtlus. Valgustite tegemiseks kasutati rakendust Blender. Blenderis on tehtud esialgne DBV mudel ning igasse ruumi sobiva kuju ja suurusega valgusallika tegemine oli lihtsam kui Unity. Valgustiteks sai valitud tasand (*plane*), kuna tasandi eelis teiste 3D objektide ees on see, et tasand koosneb vähematest tippudest kui teised 3D objektid. Tippude arvu madalal hoidmine parandab visualisatsiooni kaadri renderdamiseks kuluvat aega. Tasand kiirgab valgust ainult ühest küljest, vältides sellega liigse valguse tekkimist.

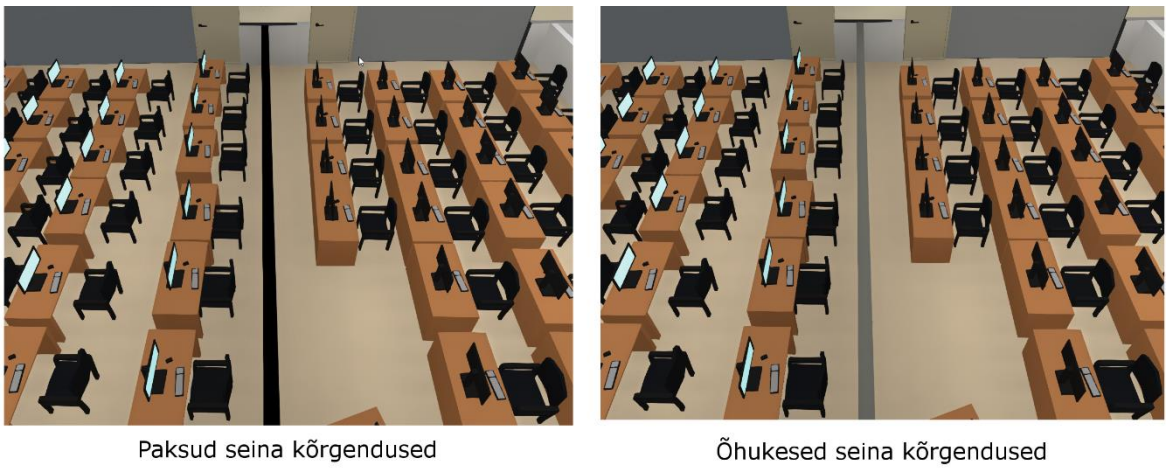
Illustratsioon 8. Valgustid

Valgust kiirgav materjal kiirgab valgust objekti pinnalt nähtavate objektide suunda ning seetõttu on tarvilik takistada valguse levikut naaberruumidesse. DBV eelmised arendajad tegid ruumide ja agentide parema nähtavuse saavutamiseks seinad poolkõrged. Agentideks nimetatakse visualisatsioonis liikuvaid inimesi. Valgustuse seisukohalt on poolkõrged seinad kehvad, sest need ei takista valguse levikut kõrvalruumidesse. Selle lahendamiseks sai loodud seinatele nähtamatud seina kõrgendused, mis ulatusid laeni. See takistas valguse levikut kõrvalruumidesse. Illustratsioon 9 näitab seina kõrgenduste võrdlust.



Illustratsioon 9. Seinakõrgenduste võrdlus

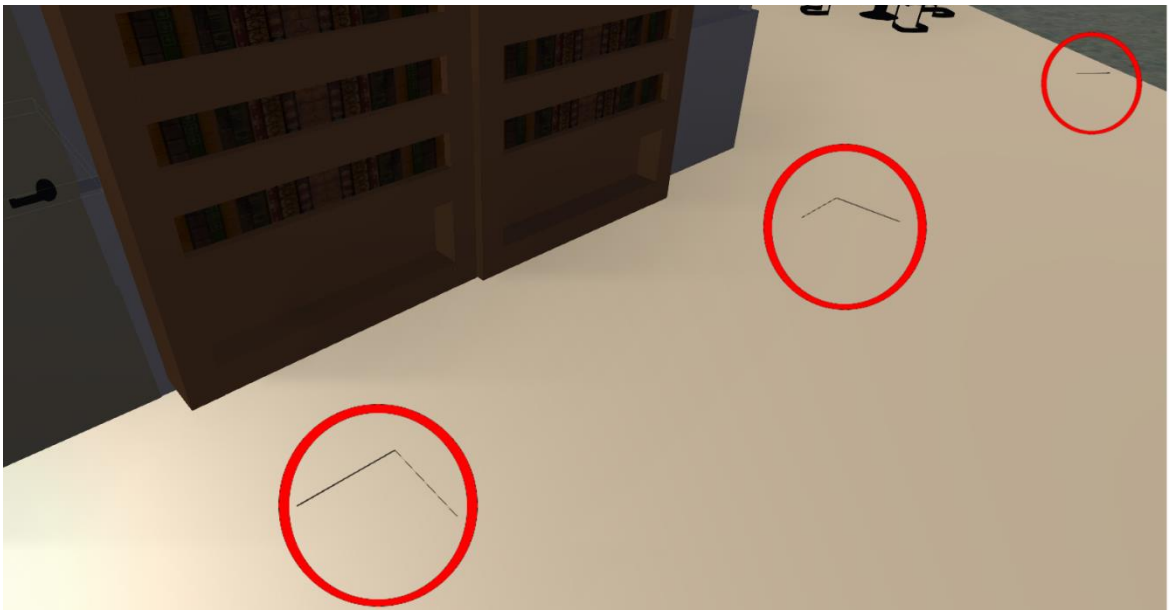
Seina kõrgenduste laiuseks sai valitud seina laiusest väiksem suurus, kuna vastasel juhul, kui seina kõrgenduse laius on võrdne või suurem kui sein ise, siis ei pääse valgus seina pealmisele küljele ning see osa seinast paistab visualisatsioonis mustana (vt. Illustratsioon 10).



Illustratsioon 10. Seinapaksuse võrdlus

Eelnevalt oli mudelis põrandad tehtud erinevate suurustega tükkidest. Osadel ruumidel oli oma põrand, teised ruumid jagasid põrandat naaberruumiga ning kolmandad ruumid jagasid põrandat koridori ning poole korrusega. Valguse levimise peatamiseks kõrvalruumidesse sai loodud igale ruumile oma põrand. Väiksemate põrandate loomine ühe suure asemel

vähendas ka valgustuse küpsetamisaega. Ruumidele, millel oli varasemalt oma põrand olemas, sai ka loodud uus põrand, kuna vanadel põrandatel oli augud sees, mis valguse sisse-välja lülitamisel selgelt näha olid (vt. Illustratsioon 11).

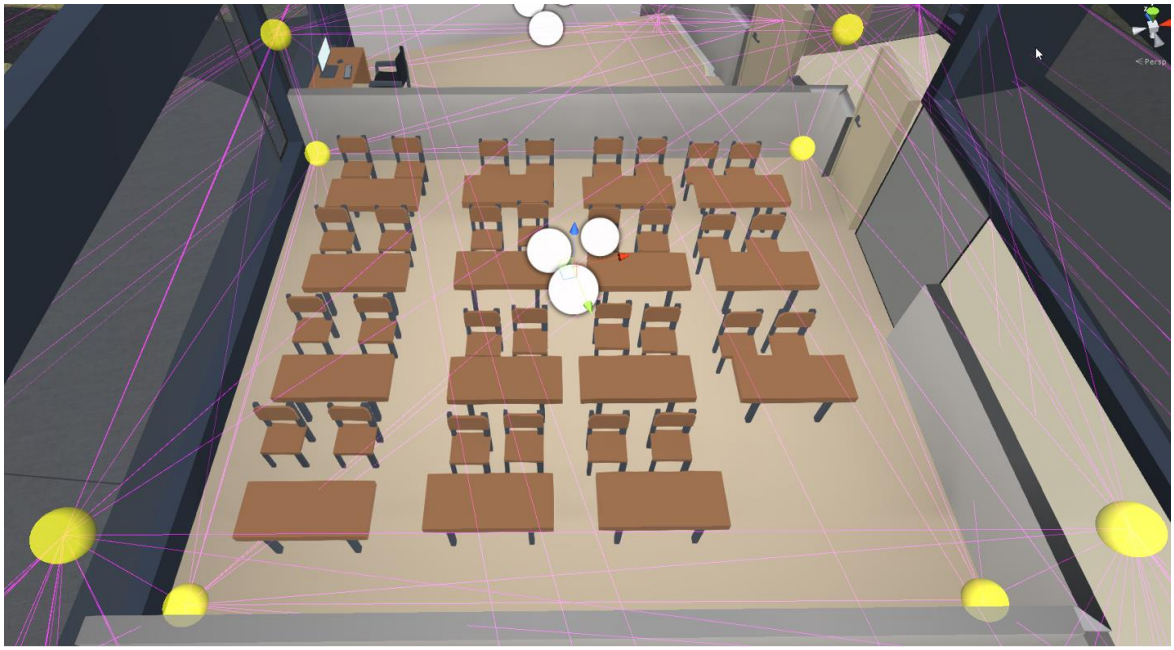


Illustratsioon 11. Augud põrandas

Ruumi valgusti, seinad (seina kõrgendused) ja põrand tehti staatiliseks, sest need need ei liigu ega pöörle ning on vajalik kiirgava materjali töötamiseks.

Ruumis olevad esemed tehti staatiliseks ning eemaldati valgustekstuuride staatilisus (*light-map static*). Valgustekstuuride staatilisus kontrollib, kas objektile genereeritakse valgustekstuur või mitte. DBV projektis on mitu tuhat väikest objekti nagu lauad, toolid jne. Väiksematelt objektidelt valgustekstuuride staatilisus eemaldamine vähendas valgustuse küpsetamisaega ja ühtlustas obektide valgustust. Dünaamiliste objektide valgustamiseks kasutatakse valgussondide gruppe. Nende kasutamine ühtlustas esemete valgustamist ruumis. Igasse ruumi sai paigutatud valgussondi grupp, mis valgustas selles ruumis olevaid objekte. Sondid sai paigutatud ruumis olevate objektide ümber (vt. Illustratsioon 12). Keeruliseks tegi sondide paigutamisel asjaolu, et need tuli asetada kahele erinevale korrusele. Sai

jälgitud, et ruumis olevad objektid saaks valguse enda ruumi sondidest mitte alumise/ülemise korruselt või kõrvalruumist.



Illustratsioon 12. Valgussondid

Koridori valgustamisel on kasutatud eeldust, et hoone visualisatsiooni saab ainult vaadata ainult õppehoone fuajees. Sellest sai järeldatud, et hoone koridorid peavad olema alati valgustatud. Püsivalt põlevate tulede simuleerimiseks kasutati ühte suundvalgusallikat. Selle valgusallika eesmärgiks on valgustada ainult kihte (*layer*), mis peavad olema püsivalt valgustatud. Kihte kasutatakse Unity mängumootoris sarnaste objektide grupeerimiseks ja võimaldab piirata operatsioonid nendele kihtidele[22]. Neid kihte on hetkel ainult kaks: "static", mis valgustab esimese korruse püsivalt valgustatuid objekte ning "secondFloorStatic", mis valgustab teise korruse objekte.

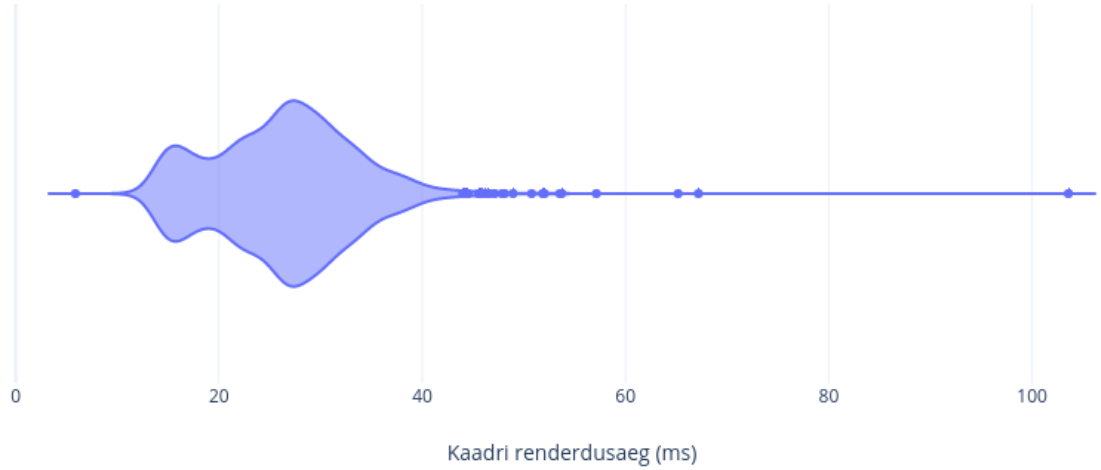
Tulemused

Selle peatüki tulemusena loodi visualisatsiooni tulede sisse-välja lülitamise lahendus. Sellega täideti funktsionaalsed nõuded F1 ja F2. Visulisatsioon töötab ilma tõrgeteta.

Valgustuse jõudluse mõõtmiseks kasutati rakendust Fraps. Tulede sisse-välja lülitamist mõõdeti koos agentidega, sellepärast, et agentide ruumi sisenedes, süttib ruumi valgustus. Visualisatsiooni kasutati 1471 agenti. Mõõtmine algas, kui agentid hakkasid oma alguspunktist klassiruumi poole liikuma. Mõõtmine kestis kaks minutit, et andmed oleksid võrreldavad peatükis 2.1 mõõdetud andmetega. Tulemuseks saadi mediaan kaadri

[22] https://docs.unity3d.com/560/Documentation/Manual/class-TagManager.html

renderdusajaks 26.73 ms ehk 37.4 FPS (vt. Illustratsioon 13)



Illustratsioon 13. Valguse sisse-välja lülitamise kaadri renderdusaeg

Mittefunktsionaalne nõue MF5 nõudis, et mediaan renderdusaeg oleks väiksem kui 29.66 ms. See nõue täideti käesolevas peatükis.

5.2 Välisvalgustus

Eelmiste arendajate töödest oli võimalik lugeda, et ööpäeva tsükli visualiseerimiseks kasutatakse suundvalgusallika suuna muutmist. Rakenduse koodi põhjalikult uurides selgus, et skripti, mille ülesandeks oli pöörata valgusallikat vastavalt reaalmaalilma päiksele ei kasutatud stseenis. Selle põhjuseks oli uuema Unity mängumootori versiooni kasutamine. Eelmised arendajad kasutasid Unity 2017 aasta versiooni, aga käesolevas töös kasutati Unity 2018 aasta versiooni. 2018 aasta versioon tegi kokkupakitud objektide (*prefab*) haldamise lihtsamaks[23]. Ööpäeva tsükli parandamiseks lisati skript `Sun` päikest kujutava suundvalgusallika külge. Skriptis olev kood parandati ära ning integreeriti Daniel Kütti bakalaureusetööga. Illustratsioon 14 kujutab öist õppehoonet.

[23] https://blogs.unity3d.com/2018/06/20/introducing-new-prefab-workflows/

Illustratsioon 14. Öine Delta õppehoone

Delta õppehoone ümbruse valgustamiseks asetati objektid, mis asuvad väljaspool hoonet eraldi kihtidesse. See võimaldas valgustada suundvalgusallikaga ainult neis kihtides olevaid objekte. Objektid jaotati kolme kihti: “outside”, “outsideRoad” ja “outsideActor”. See võimaldas erinvate objektidele teha erinevaid operatsiooni. Näiteks kihti “outsideRoad” kasutati lume loomiseks.

6. Ilmastiku visualiseerimine

Käesolevas peatükis kirjeldatakse Delta õppehoone visualisatsiooni ilmastiku efektide täiustamist.

Eelnevalt oli visualisatsioonis implementeeritud järgmised ilmastiku nähtused: vihm, lumi, pilvisus ja äike. Sademete visualiseerimiseks kasutati Unity osakeste süsteemi ning maapinna visualiseerimiseks kasutati rakenduses Blender loodud 3D objekte ning Unity animeerimissüsteemi. Unitys animeeriti objektide suurust (*scale*). [2]

Sellise lahenduse puuduseks on vähene variatsiooni. Käesolevas peatükis kasutati lumise ja vihmase maapinna täiustamiseks varjutajaid (*shader*). Varjutaja on hulk instruktsioone, mis jooksutatakse GPU (*Graphical Processing Unit*) peal [10]. Vivo ja Lowe ütlevad oma raamatus [11] järgmist. Ekraanil resolutsiooniga 800×600 on 480 000 pikslit ja kaadrisagedusel 30 kaadrit sekundis tuleb teha 14 400 000 arvutust sekundis. CPU-s (*Central Processing Unit*) on tavaliselt neli tuuma ning üle 14 miljoni arvutuse sekundis on sellele koormav tegevus. GPU-s on seevastu, aga mitusada mikroprotsessorit, mis on mõeldud kindlat tüüpi ülesannete lahendamiseks. GPU-s tehakse pikslite arvutused paralleelselt ja see teeb varjutaja arvutamise GPU-s väga kiireks [11].

Varjutajaid[24] on mitut tüüpi. Nendeks on tipu varjutaja (*vertex shader*), tesselatsiooni varjutaja (*tesselation shader*), geomeetria varjutaja (*geometry shader*), fragmendi varjutaja (*fragment shader*) ja arvutus varjutaja(*compute shader*). Varjutaja võtab sisendiks erinevad parameetrid nagu näiteks mudeli tipud, tekstuurid, valguse informatsiooni jm ning renderdamise standartse töövoo viimane varjutaja tagastab fragmendi värvi [10]. Unitys on valgusega varjutaja lihtsamaks kasutamiseks loodud pinna varjutaja (*surface shader*)[25]. Pinna varjutaja genereerib saadud info põhjal tipu ja fragmendi varjutaja.

Dokumentatsioonis on kasutavate platvormide kohta kirjutatud järgmist[26]: Unity mängumootorit saab kasutada mitmel platvormil: Direct3D, OpenGL, Metal ja konsoolidel. Need saab jagada kahte gruppi Direct3D sarnased, kuhu kuulub veel Metal ja konsoolid, ning OpenGL sarnased. Direct3D kasutab koordinaatsüsteemi, mis algab ülevalt vasakult ning y koordinaat suureneb allapoole. OpenGL kasutab koordinaatsüsteemi, mille algus on all vasakul ning y koordinaat suureneb ülesse. Kindlustamaks, et varjutajas kasutatav

[24] https://www.khronos.org/opengl/wiki/Shader
[25] https://docs.unity3d.com/Manual/SL-SurfaceShaders.html
[26] https://docs.unity3d.com/Manual/SL-PlatformDifferences.html

koordinaatsüsteem on sama igal platvormil, lisati varjutajasse kontroll ning vajadusel pööratakse tipu varjutajas tekstuuri y koordinaat ümber.

6.1 Lumi

Lume visualiseerimiseks saadi inspiratsiooni Wolfgang Engeli raamatust "GPU pro 7: Advanced rendering techniques" [12]. Peatükis "Deferred snow deformation in Rise of the Tomb Rider" kirjeldati kuidas visualiseeriti lund arvutimängus Rise of the Tomb Rider. Seal kasutati 4 erinevat varjutajat, et luua lumme jalajäljed, mis ajapikku lund täis sadasid. Sellest tulenevalt üritati luua sarnane varjutaja jalajälgede jaoks.

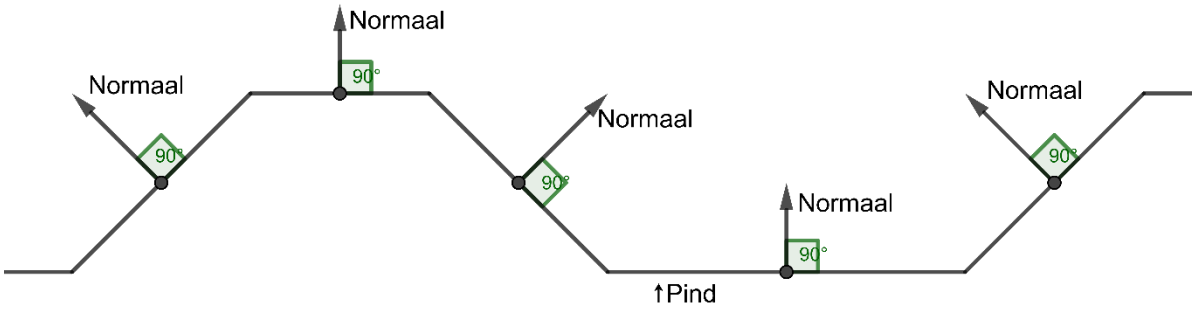
Lume kohta kirjutatakse artiklis „Snow Characterics" järgmist: Lumi paistab inimesele valgena. Selle põhjuseks on, et nähtav valgus on valge ja lumi peegeldab enamuse valgusest tagasi. Kuigi mõnikord tundub lumi vaatajale sinakas. See juhtub, sest valgusallikalt tulev valgus peegeldub lume terade vahel mitu korda ning punast valgust neeldub rohkem kui sinist valgust. Mida kaugemal vaataja lumest on seda sinisem lumi tundub. [13]

Lumele välimuse andmiseks loodi pinna varjutaja. Varjutajas kasutati valgustuseks Unity füüsikal põhinevat valgustuse mudelit (*physically based lighting model*)[27]. Füüsikaline valgusmudel imiteerib reaalse maailma valgustust. Seda kasutades on võimalik lisada varjutajale erinevaid omadusi nagu näiteks värv, normaali tekstuur, karedus. Visualisatsiooni kaamerad asuvad maja kohal ja seetõttu asuvad lumest üsna kaugel. Lume värvuseks valiti sinakas tekstuur (vt. Illustratsioon 15). Kõik lume tekstuurid pärinevad veebilehelt www.textures.com, kui pole öeldud vastupidist.

Illustratsioon 15. Lumevärvuse tekstuur

Sujuva ülemineku loomiseks maapinna tekstuuri ja lume tekstuuri vahel kasutatakse lineaarset interpoleerimist kasutades funktsiooni `lerp`. `Lerp` funktsioon kasutab muutujat `SnowAmount`, mis määrab ära lume tekstuuri osakaalu maapinna värvi arvutamisel. Ainult värvi tekstuuriga tundub lume pind liialt sile. Selle parandamiseks kasutati normaali tekstuuri. Normaalideks[28] nimetatakse pinnaga risti olevaid vektoreid (vt. Illustratsioon 16).

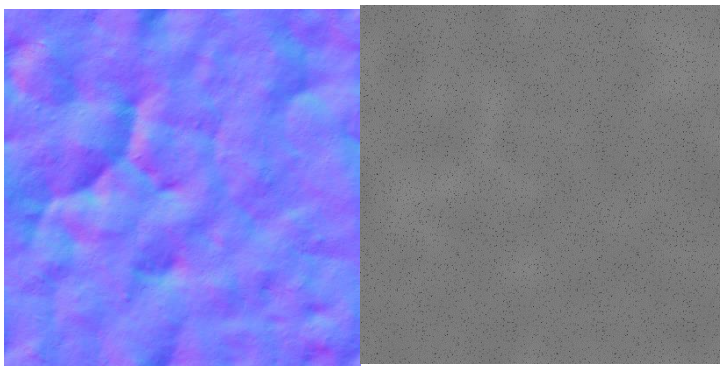
[27] https://docs.unity3d.com/Manual/shader-StandardShader.html
[28] http://www.opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading/



Illustratsioon 16. Normaalid

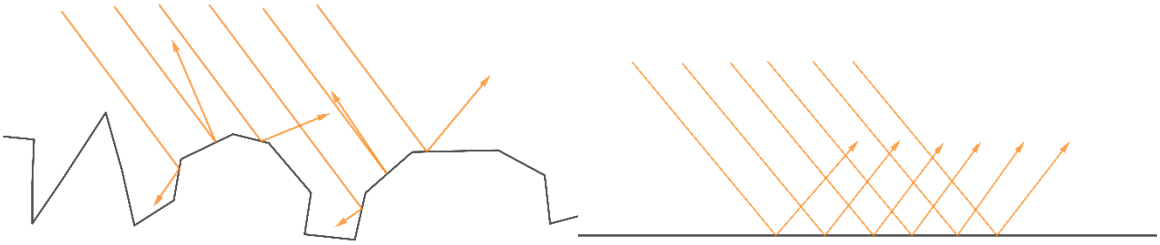
Neid kasutatakse kõige sagedamini valgustuse arvutamisel. Normaalide abil määratakse ära peegeldumisnurk, mille all valgus objektilt tagasi peegeldub vaataja silma (kaamerasse). Lisades varjutajale normaali tekstuuri (vt. Illustratsioon 17) muutus lume pind krobelisemaks.

Lume peegeldumisvõime tõstmiseks lisati varjutajasse karedus (*roughness*) tekstuur ja sujuvuse (*smoothness*) väärtus. Karedus määrab ära kui sarnane on materjal metallile[29]. Metallisem materjal peegeldab rohkem valgust. Sujuvus kujutab pinna mikro detaile ehk näitab kui ühtlane on pind mikrotasandil (vt. Illustratsioon 18) [30].



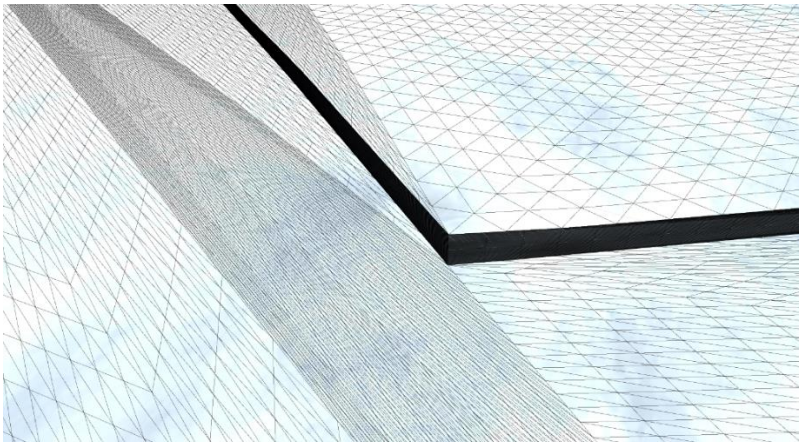
Illustratsioon 17. Lume normaali ja kareduse tekstuur

[29] https://docs.unity3d.com/Manual/StandardShaderMaterialParameterMetallic.html
[30] https://docs.unity3d.com/Manual/StandardShaderMaterialParameterSmoothness.html



Illustratsioon 18. Vasakul pinna sujuvus minimaalne ja paremal maksimaalne

Lumele jalajälgede joonistamiseks sai proovitud kasutada raamatus GPU Pro[7]: Advanced Techniques kirjeldatud meetodit [12]. Lumele jalajälje tekitamiseks prooviti viia jalajäljele vastavad tipud tagasi algasendisse ehk asendisse, kus nad olid enne lumele paksuse lisamist. DBV maapind koosneb väga paljudest erineva suuruse ja kujuga tahkudest. Kuna tahud on suuremad kui jalajäljed, siis jalajälje mõjutatud tippude muutmisega ei õnnestunud jalajälge tekitada. Selle probleemi lahendamiseks otsustati teha tahud väiksemaks kasutades tesselatsiooni[31]. Tesselatsioon on pinna katmine kujunditega nii, et ei ole ülekatet ega vahet kujundite vahel. Mängumootoris Unity on tesselatsioonis kasutatavaks kujundiks kolmnurk[32]. Ühtlase tesselatsiooni saavutamiseks on tarvis, et aluspinna tahud oleksid võimalikult sarnased. Kuna DBV visualisatsioon maapind koosneb paljudest erinevad suurusega ja kujuga tahkudest, siis ei tessaleerinud maapind ühtlaselt. Illustratsioon 19 on näha halvasti tessaleerinud maapind. Osad tahud on suuremad kui teised.



Illustratsioon 19. Tessaleerinud maapind

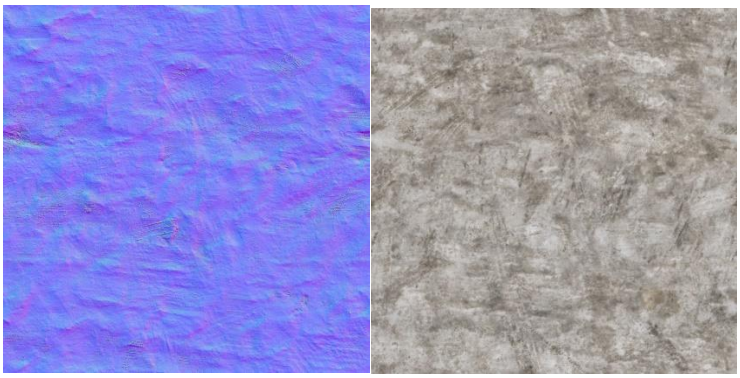
Seetõttu otsustati jalajälgede visualiseerimiseks kasutada ainult pikslite värvi muutmist.

Delta õppehoone ümbritseval alal käib päevas sadu inimesi. Visualisatsioonis olevad agendid luuakse kindlates kohtades teeradadel. Seetõttu ainult agentide jalajälgede kujutamine

[31] https://www.mathsisfun.com/geometry/tessellation.html
[32] https://docs.unity3d.com/Manual/SL-SurfaceShaderTessellation.html

ei anna täpset infot Delta hoone ümbritseval alal käidavate inimeste kohta ja otsustati jalajälgede asemel visualiseerida lumele mudased teerajad. Teeradade jaoks loodi sügavus tekstuur, kus on peal kõik teerajad. Tekstuuri loomiseks kasutati kaamerat ja renderduse tekstuuri[33] (*render texture*). Renderduse tekstuur on tekstuur, mida on võimalik muuta rakenduse töötamise ajal. Maapinna alla, vaate suunaga üles asetati kaamera, mis salvestas maapinnal asuvaid teeradu renderduse tekstuuri. Sellise lahenduse eeliseks võrreldes käsitsi tegemise ees on, et lisades stseeni uusi objekte saab kerge vaevaga ja kiiresti teha uue tekstuuri. Peale renderduse tekstuuri loomist salvestati renderduse tekstuur faili ning anti varjutajale.

Tänu sellel sai render tekstuuri loomiseks kasutatud kaamera desaktiveerida säästes sellega visualisatsiooni jõudlust. Varjutaja kasutab loodud tekstuuri ning renderdab teeradade asemele mudase lume kasutades mudase lume tekstuuri, normaale (vt. Illustratsioon 20). Viimistletud lumist maapinda kujutab Illustratsioon 21.



Illustratsioon 20. Mudase lume värvi ja normaali tekstuur

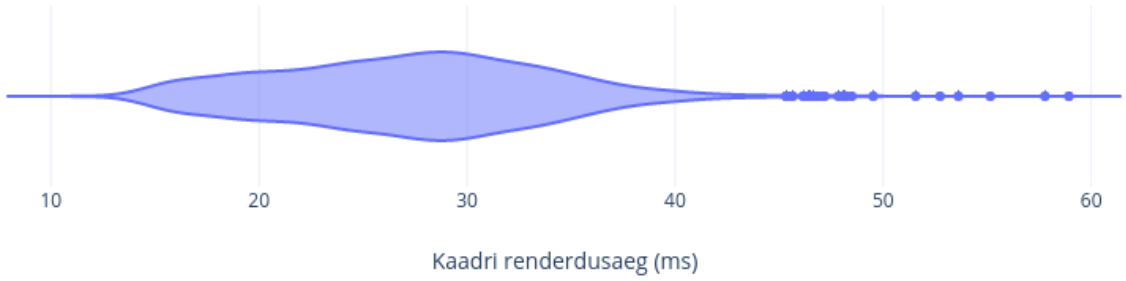
[33] https://docs.unity3d.com/Manual/class-RenderTexture.html

Illustratsioon 21. Viimistletud lumine maapind

Tulemused

Selle peatüki tulemusena sai täidetud funktsionaalsed nõuded F3 ja F4. Visualisatsioon töötab tõrgeteta.

Lume efekti jõudluse testimiseks kasutati rakendust Fraps. Frapsiga mõõdeti visualisatsiooni kaadri renderdusaega lumisel päeval. Mõõtmiseks kasutati samasugust meetodit nagu peatükis 2.1, et mõlemaid andmeid saaks oma vahel võrrelda. Lume efekti mediaan kaadri renderdusajaks saadi 27.52 (vt. Illustratsioon 22).



Illustratsioon 22. Lume efekti kaadri renderdusaeg

Sellega täideti mittefunktsinaalne nõue MF6.

6.2 Vihm

Varasemalt kasutati vihmase maapinna visualiseerimiseks käsitsi asetatud objekte ning Unity animeerimissüsteemi muutes objektide suurust [2]. Käesolevas töös loodi vihmase maapinna kujutamiseks varjutaja.

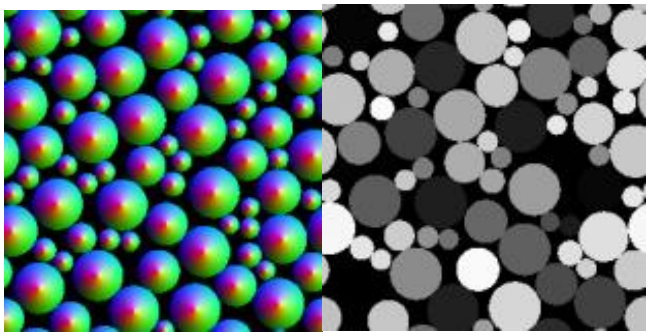
Märg pind erineb kuivast tema peegeldumisvõime poolest. Märg pind peegeldab rohkem valgust kui kuiv pind. Pinna peegelduvaks tegemiseks muutsin pinna sujuvust. Mida sujuv pind on seda rohkem valgust peegeldub vaataja silma (vt. Illustratsioon 23 ja Illustratsioon 24).

Illustratsioon 23. Kuiv maapind

Illustratsioon 24. Märg maapind

Kui vihma sajab ja maapind on märg tekivad maapinnale veelombid. Veelombile langev veepiisk tekitab laineid. Lainetuse animatsioon saavutatakse kahe varjutajaga. Esimene varjutaja on pinna varjutaja ning sellega renderdatakse maapind. Maapinna renderdamisel võetakse arvesse peatükis 6.1 kirjeldatud lund.

Teine varjutaja renderdab veelombid ja vihmapiiskade animatsiooni lompidele. Vihmapiiskade varjutaja loomisel oli inspiratsiooniks Sébastien Lagarde blogi [14]. Vihmapiiskade renderdamisel kasutatakse kahte erinevat tekstuuri (vt. Illustratsioon 25). Tekstuurid on pärit Legarde blogist [14].



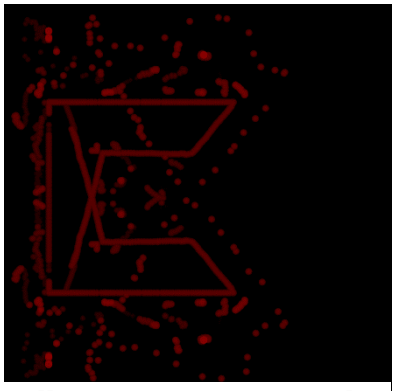
Illustratsioon 25. Vihmapiiskade tekstuurid

Vasakpoolne tekstuur määrab ära tekkiva lainetuse suuruse ja kuju ning parempoolset kasutatakse vihmapiisa animatsiooni algusaja määramiseks. Sellega saavutatakse piiskade asümmeetrilisus ajas.

Vihmasaju visualiseerimiseks on varjutajal vihma intensiivsus muutuja. Muutuja väärtus on nulli ja ühe vahel. Vihma korral kasutatakse väärtust 1 ja vastasel korral väärtust 0.

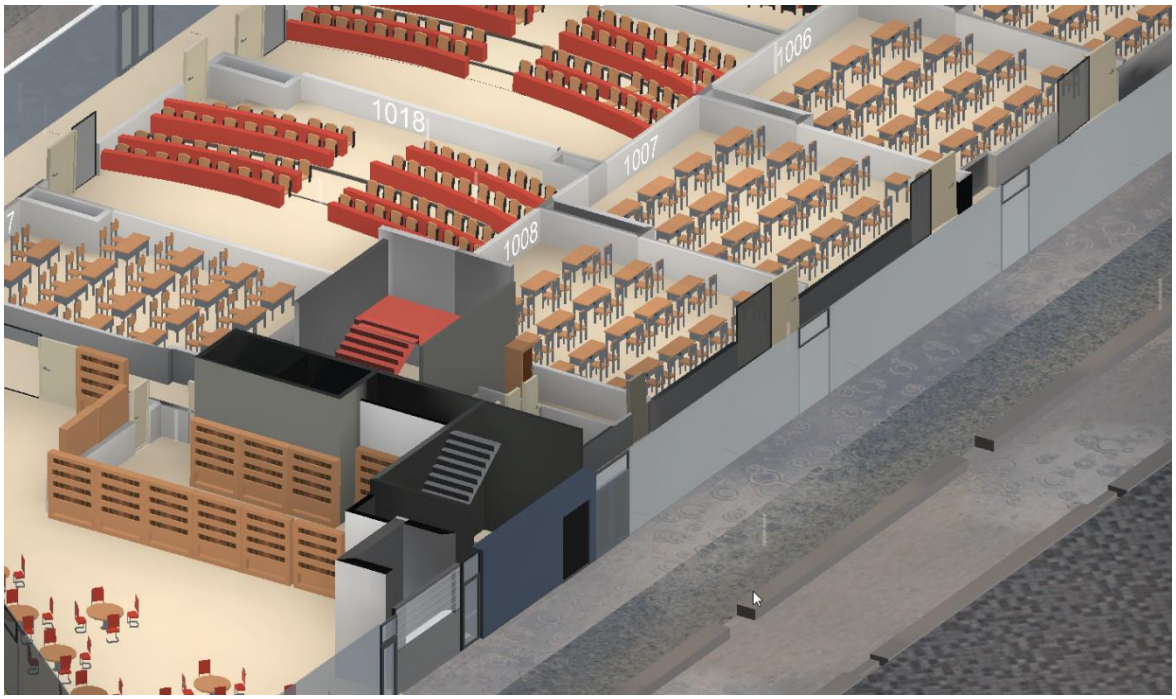
Lompide paremaks kujutamiseks lisati visualisatsiooni taevas. Varem oli taevaks Unity standard taevas, kus ei olnud midagi. Uue taeva lisamisega tekkisid pilved ning nende peegeldust on lompide pealt näha. Varjutaja kasutab taevast lompidele peegelduse andmiseks.

Veelombid paigutati maapinnale käsitsi, sest DBV maapind koosneb paljudest erinevatest tahkudest ning lombid, mis asuvad mitme tahu peal, osutusid probleemseteks. Lompide asetuse tekstuur tehti kasutades renderduse tekstuuri, kaamerat ja kera objekti. Kera asetati maapinnale lombiks sobivale kohale ja kaamera salvestas objekti asukoha tekstuuri (vt. Illustratsioon 26).



Illustratsioon 26. Lompide asetuse tekstuur

Jõudluse säästmiseks desaktiveeriti lompide tekstuuri loomiseks kasutatavad tööriistad ning tekstuur salvestati faili. Viimistletud vihmast ilma kujutatakse Illustratsioon 27 ja Illustratsioon 28.



Illustratsioon 27. Vihmane ilm

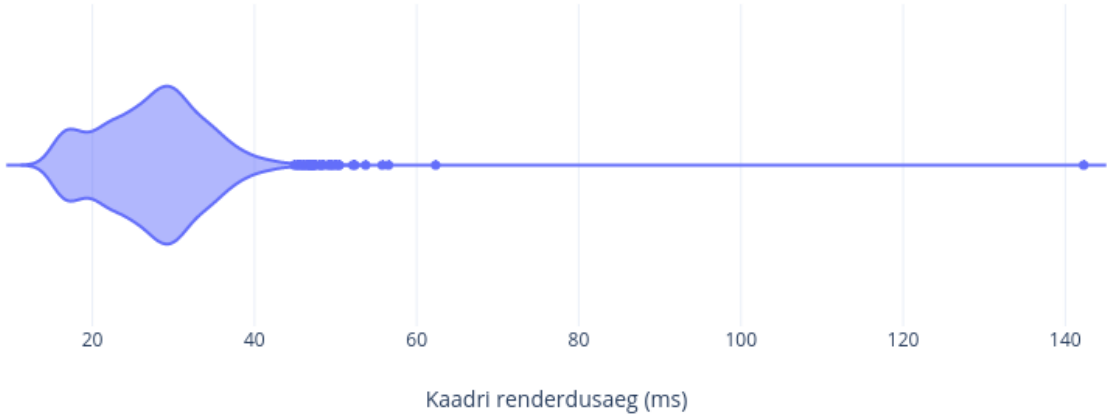
Illustratsioon 28. Vihmane ilm

Vihma lompe ei looda muru peale, sest vihm imbub kiiresti pinnasesse ja sinna ei teki lompe.

Tulemused

Selle peatüki tulemusena täideti funktsionaalsed nõuded F5, F6 ja F7. Visualisatsioon töötab tõrgeteta.

Jõudluse mõõtmiseks kasutati rakendust Fraps. Kaadri renderdusaja mõõtmiseks kasutati samasugust meetodit nagu peatükis 2.1. Mõõteajaks oli kaks minutit ning visualisatsioon sisaldas 1471 agenti. Visualisatsiooni ilm oli vihmane. Mediaan kaadri renderdusajaks saadi 27.93 ms (vt. Illustratsioon 29).



Illustratsioon 29. Vihmase ilma kaadri renderdusaeg

Sellega täideti mittefunktsionaalne nõue MF7.

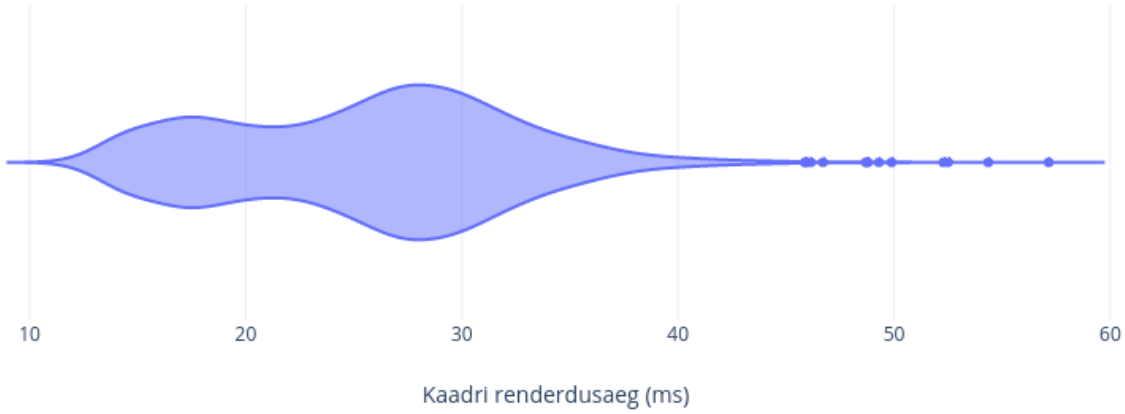
7. Testimine

Kindlustamaks, et visualisatsioon töötab hästi, testiti visualisatsiooni jõudlust kui ka visuaalset poolt inimeste peal. Jõudluse testimiseks kasutati peatükis 2.1 kirjeldatud arvuti ja visuaalne testimine toimus Tartu Ülikooli raamatukogus.

7.1 Jõudluse testimine

Visualisatsiooni jõudlus oleneb kahest faktorist: käesolevas töös lisatud efektidest ja Meelis Perli bakalaureusetööst [4]. Testimiseks kasutati kahte erinevat rakendust. Visualisatsiooni renderdusaega testiti rakendusega Fraps. Fraps eelis Unity Profiler tööriista ees on, et Frapsi nõuab töötamiseks vähem resursse kui Unity Profiler tööriist.

Rakenduse üldise jõudluse testimiseks kasutati sama meetodit nagu peatükis 2.1. Seetõttu saab neid kahte andmeid võrrelda. Visualisatsiooni mediaan renderdusajaks mõõdeti 26.55 millisekundit (vt. Illustratsioon 30). Arvutatuna kaadriks sekundis on see 37.7 FPS. Visualisatsiooni renderdusaeg võrreldes töö algusega vähenes 3.11 millisekundit ehk 10.9 protsendi võrra. Tulemus vastas mittefunktsionaalsele nõudele MF1. Visualisatsiooni maksimaalne kaadri renderdusaeg oli 57.16 ms. Töö alguses oli maksimaalne kaadri renderdusaeg 58.05 ms. Seega vähenes maksimaalne renderdusaeg 0,89 ms ehk 1.5 protsendi võrra. Tulemus vastab mittefunktsioaalsele nõudele MF2.

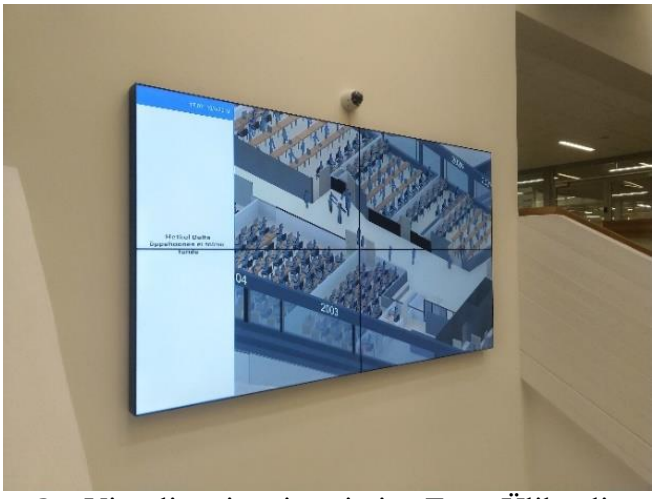


Illustratsioon 30. Visualisatsiooni renderdusaeg

Minimaalseks renderdusajaks oli 11.53 ms. Võrreldes töö algusega vähenes see 34 protsendi võrra. Unity Profiler tööriistaga mõõdeti keskmist muutmälu kasutust. Tulemusega saadi staabiline 580MB. Ei esine mälu lekkeid ja sellega täideti mittefunktsionaalne nõue MF3.

7.2 Visuaalne testimine

Visualisatsiooni visuaalset poolt testiti Tartu Ülikooli raamatukogus. Samaaegselt testiti kolme lõputööd: käesolev töö, Meelis Perli bakalaureusetöö [3] ja Daniel Kütti bakalaureusetöö [4].



Illustratsioon 31. Visualisatsiooni testimine Tartu Ülikooli raamatukogus

Visualisatsiooni testimisel osales viis inimest, aga rakenduse ootamatute tehniliste vigade tõttu viibis algus ning reaalselt sai testida ainult kahe inimese peal. Testimiseks koostati küsitlus koos Meelis Perliga. Küsitlus jaotati nelja ossa. Nendeks oli testija kohta käiv informatsioon, visualisatsioonis olevad agendid, visuaalsed efektid ja üldised küsimused visualisatsiooni kohta (vt. Lisa II). Testijatelt küsiti kuuepalliskaala küsimusi visualisatsiooni väljanägemise kohta. Kuuepalliskaala valiti seetõttu, et ei oleks võimalik valida keskmist väärtust. Visualisatsiooni ööpäeva tsüklil hinnati hindega 4.5 palli. Sise -ja välisvalgustuse hindeks tuli vastavalt 5.0 ja 5.5 palli. Lumise maapinna hindeks anti 6.0 palli. Lumel olevate teeradadele anti hindeks 4.5 palli. Vihmase ilmale anti hindeks 5.5 palli ning lompide peal olevatele vihmapiiskadele 6 palli. Tervele visualisatsioonile kokku anti hindeks 5. Visualisatsiooni arusaadavus oli 5 ning huvitavus 4 palli. Illustratsioon 31 on näha Tartu Ülikooli raamatukogu ekraani, mille peal testiti.

Tulemused

Käesoleva töö autor arvab, et visuaalsest testimisest saadud informatsioon ei ole piisav, et sellest mingeid järeldusi teha. Selle põhjuseks on vähene testijate arv.

8. Kokkuvõte

Käesoleva töö oli jätk 2017/2018 aastal tehtud Delta õppehoone visualisatiooni bakalaureusetöödes alustatud projektidele. Töö eesmärgiks oli optimeerida ja täiustada visualisatsiooni visuaalseid efekte. Töö käigus sai optimeeritud visualisatsiooni valgustust ja ilmastike efekte. Tööst selgus, et visualisatsioon sisaldas kohti, mis ei töötanud või vajasid optimeerimist. Töö käigus parandati olemasolevat koodi, et tulevastel arendajatel oleks kergem koodist aru saada ja seda täiustada. Sai võrreldud erinevaid võimalusi valgustuse visualiseerimiseeks.

Üks töö eesmärke oli täiustada visualisastsiooni valgustust. Selle tarveks tehti igasse õppehoone ruumi oma valgusti, et oleks võimalik tulesid sisse-välja lülitada sõltuvalt inimeste arvust ruumis. Selle optimaalseks implementeerimiseks kasutati kiirgavat materjali ja valgussonde.

Veel üks eesmärk oli täiustada visualisatsiooni ilmastiku efekte. Ilmastiku efektidest täiustati lumist ja vihmast maapinda, luues neile kergesti kohaldatav varjutaja. Lume sadades kattub terve maapind lumega ja tekivad porised teerajad. Maapind saavutati varjutajale erinevate tekstuuri ette andmisega. Vihmasaju korral muutuvad pinnad märjaks ja läigivad vaatajale vastu. Vihmasaju jätkudes tekivad teeradade peale lombid. Lompide peale animeeriti vihmapiiskade lainetused kasutades varjutajat.

Käesoleva tööl on edasi arendamise võimalusi. Üheks võimaluseks on optimeerida visualisatsiooni animatsioone või teha inimese seisukohast vaate visualisatsioonile. Inimese seisukohast olev vaade annaks uue huvitava ilme visualisatsioonile. Veel võimalik visualiseerida erinevad aastaajad (kevad, sügis), udu ja linnud.

Lõputöö andis autorile arvestaval määral uusi teadmisi arvutigraafikas olevate erinevate võtete kohta, varjutajate loomise kohta ning kogemusi lõputöö kirjutamisest.

Tänan Tartu Ülikooli arvutigraafika ja virtuaalreaalsuse (CGVR) laborit võimaluse eest kasutada töö tegemiseks arvuteid. Veel tänan Tartu Ülikooli raamatukogu võimaluse eest testida visualisatsiooni suure ekraani peal. Tänan oma juhendajat Raimond-Hendrik Tunnelit, kes aitas mind lõputöö tegemisel.

Viidatud kirjandus

[1] A. Nikolajev, Delta õppehoone visualiseerimine ja optimeerimine, bakalaureusetöö, Tartu Ülikool, 2018

[2] A. Voitenko, Delta õppehoone keskkonna visualiseerimine, bakalaureusetöö, Tartu Ülikool, 2018

[3] Meelis Perli, Delta õppehoone visualisatsioon – agentide loogika, bakalaureusetöö, Tartu Ülikool, 2019

[4] Daniel Kütt, Delta õppehoone visualisatsioon – administratiivne tööriist, bakalaureusetöö, Tartu Ülikool, 2019

[5] B. Owens, Forward Rendering vs. Deferred Rendering, https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342 (07.05.2019)

[6] R. C. Martin, 2009, Clean Code: A Handbook of Agile Software Craftsmanship,

[7] Choosing a Lighting Technique, https://unity3d.com/learn/tutorials/topics/graphics/choosing-lighting-technique, (08.05.2019)

[8] IronEqual, A Lighting-Fast Presentation of Global Illumination in Unity 5, https://medium.com/ironequal/a-lightning-fast-presentation-of-global-illumination-in-unity-5-112ab5e79c14

[9] Probe Lighting, https://unity3d.com/learn/tutorials/topics/graphics/probe-lighting?playlist=17102 (08.05.2019)

[10] What is shader?, https://www.shadercat.com/what-is-a-shader/ (08.05.2019)

[11] P. G. Vivo, J. Lowe, The Book of Shaders, https://thebookofshaders.com/01/ (08.05.2019)

[12] W. Engel, 2016, GPU Pro^{7}: Advanced Rendering Techniques, leheküljed 3-16

[13] Snow Charateristics, https://nsidc.org/cryosphere/snow/science/characteristics.html (08.05.2019)

[14] S. Lagarde, Water drop 2b – Dynamic rain and its effects, https://seblagarde.wordpress.com/2013/01/03/water-drop-2b-dynamic-rain-and-its-effects/ (08.05.2019)

Lisad

I Terminid

Kaader	Üks pilt esitavate piltide jadas.
Tekstuur	Pinna tunnuste kogum pildina.
Fragment	Informatsioon, mis on vajalik piksli genereerimiseks.
Varjutaja	Hulk instruktsioone, mis jooksutatakse GPU peal.
Normaal	Pinnaga risti olev vektor.

II Kaasapandud failid

Visulisatsioon lähtekood asub ZIP-failis „DeltaBuildingVisualization.zip“. Visualisatsiooni eraldiseisev rakendus asub failis „Build.zip“.

Visualisatsioon lähtekood asub Gitlabi repositooriumis, mis asub aadressil https://gitlab.com/UT-CGVR-Projects/DeltaBuildingVisualization

Failis „README.txt“ info kasutatud Unity versiooni kohta. Veel asub „README.txt“ failis viited töös kasutatud tekstuuridele.

Failis „Küsitlus.pdf“ on visuaalse testimise joaks kasutatud küsitlus.

Kaustas „Testimine“ asuvad jõudluse testide tulemused ja küsitlusele saadud vastused.

III Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Einar Linde,

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

   Delta õppehoone visualisatsioon – visuaalse efektid,

   mille juhendaja on Raimond-Hendrik Tunnel

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Einar Linde

08.05.2019