

Mappeoppgave Visualisering og Simulering

Kandidatnummer: 843

Høgskolen i Innlandet

Sammendrag

Jeg har implementert prosessering av punktskydata i C++, og simulering av regn og ball på trekantflate basert på punktskydata i Unity 2022.3.7f1. Prosjektet er tilgjengelig her: <https://github.com/FunkMarvel/VisSimMappe.git>.

Innhold

Introduksjon	2
Metode	2
Punktsky og triangulering	2
Vertekser	2
Indeksing	3
Ball på trekantflate	4
Kollisjonsdeteksjon	4
Glidefysikk	5
Rulling på flere trekanter	7
B-Splines og simulering av vassdrag	7
Resultater	9
Punktsky og glatting	9
Overflate og ball	9
Spline og vassdrag	10
Diskusjon	10
Punktsky	10
Flate og ball	10
Spline og vassdrag	11
Konklusjon	11
Prosjektfiler	12
Utdrag av vertices.txt	12
Utdrag av indices.txt	13

Introduksjon

Et viktig aspekt av moderne beredskap er forståelsen av ekstremvær og dets effekt på lokal natur. Derfor er det nødvendig å kunne lage digitale representasjoner av virkelige terrenget, samt simulere fysikk ved bruk av datamaskin på slike representasjoner. I denne rapporten utforsker jeg en metode for å modellere, samt simulere effekten av nedbør på terrenget konstruert fra punktskydata.

Rapporten fokuserer på hvordan lage en regulært indeksert trekantflate som representerer terrenget, samt hvordan bruke B-Splines til å kartlegge vassdrag som dannes ved ekstrem nedbør, og hvordan simulere effekten et slik vassdrag har på løsmateriale.

Metode

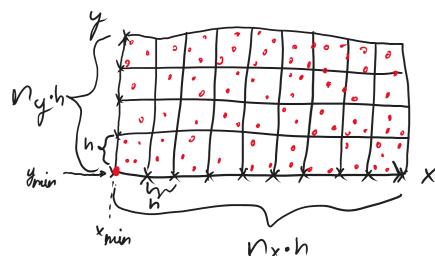
Punktsky og triangulering

Vertekser

Moderne målinger av terrenget gjøres med LiDAR, og resulterer i rådata i form av en uorganisert samling punkter i \mathbb{R}^3 relativt til et valgt koordinatsystem (Berger mfl., 2017). Jeg har valgt å laste ned punktdata fra Kartverket sin database '[hoydedata.no](#)' i '.laz'-format, som jeg så konverterer til en rentekst-fil ved hjelp av programvaren 'LASzip' (Isenburg, 2019). Den resulterende teksten inneholder da n antall punkter fordelt på formen

x_0	y_0	z_0
x_1	y_1	z_1
...
x_{n-1}	y_{n-1}	z_{n-1}

hvor hver linje svarer til et punkt.



Figur 1

Illustrasjon av røde punkter i et regulært rutenett i xy -planet.

For å kunne visualisere punktskyen som en sammenhengende overflate, så konstruerer jeg et regulært rutenett i xy -planet med dimensjoner **width** = $x_{\max} - x_{\min}$ og **height** = $y_{\max} - y_{\min}$. Hvor x_{\max} , x_{\min} , y_{\max} og y_{\min} er henholdsvis største og minste verdi for x - og y -koordinatene til punktene. Jeg velger så at hver rute i det regulære rutenettet er et kvadrat med areal 10 m^2 (steglengde $h = 10 \text{ m}$), slik at $n_x = \lceil \frac{\text{width}}{h} \rceil$ og

$n_y = \lceil \text{height}/h \rceil$ er henholdsvis antall ruter i x - og y -retning. Hvert punkt kan så sorteres inn i ruten som dekker x - og y -koordinatene dens ved å regne ut rad- og kolonneindeks i og j i rutenettet som

$$i = \lfloor \frac{x - x_{\min}}{h} \rfloor$$

$$j = \lfloor \frac{y - y_{\min}}{h} \rfloor.$$

For hver rute kan det regnes ut en gjennomsnittlig z -koordinat ved å ta gjennomsnittet av z -koordinatene til alle punktene med tilsvarende rad- og kolonneindeks

$$\bar{z}_{i,j} = \frac{1}{n_{i,j}} \sum_{k=0}^{n_{i,j}} z_{i,j,k}, \text{ eller for tomme ruter } \bar{z}_{i,j} = \frac{1}{8} \sum_{k=i-1}^{i+1} \sum_{\substack{l=j-1 \\ k \neq i \vee l \neq j}}^{j+1} \bar{z}_{k,l}.$$

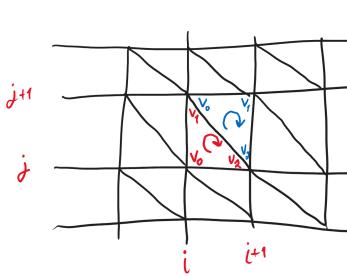
hvor $n_{i,j}$ er antall punkter innenfor ruten som tilsvarer rad i kolonne j . Gjennomsnittshøyden i en tom rute regnes som gjennomsnittet av gjennomsnittshøydene til naborutene (Nylund, 2023, ss.140-141). Hvis dette må regnes for en rute på kanten av rutenettet, så må man ekskludere de ikke-eksisterende naboen. Settet med regulært fordelte punkter kan så sentreres rundt origo, og skrives til fil på formatet

$$\begin{aligned} & n_x \cdot n_y \\ & (i \cdot h - n_x \cdot h/2, \quad \bar{z}_{i,j} - (z_{\max} - z_{\min})/2, \quad j \cdot h - n_y \cdot h/2) \\ & \dots \end{aligned}$$

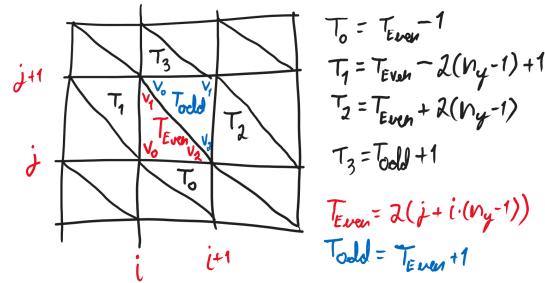
for alle $j = 0, 1, 2, \dots, n_y - 1$ for alle $i = 0, 1, 2, \dots, n_x - 1$. Første linjen i filen inneholder antall punkter. Videre er hver koordinat translatert slik at origo er sentrert, og y - og z -koordinaten er byttet om for innlesning i Unity («Unity Engine», 2023) som bruker et venstrehendt koordinatsystem med positiv y -akse som oppoverretning. Se '[Utdrag av vertices.txt](#)' for eksempel på resulterende punktdata.

Indeksering

For å kunne tegne en trekantflate av de regulært fordelte punktene i underseksjonen '[Vertekser](#)', så må det konstrueres en regulær triangulering med punktene som vertekser i trekkanter (Nylund, 2023, ss.140-141). Dette gjøres ved å ta for seg kvadratet med hjørner $v_0 = j + i \cdot n_y$, $v_1 = (j + 1) + i \cdot n_y$, $v_2 = j + (i + 1) \cdot n_y$ og $v_3 = (j + 1) + (i + 1) \cdot n_y$, hvor \vec{v}_k er de regulært fordelte punktene, og $j = 0, 1, 2, \dots, n_y - 1$ for alle $i = 0, 1, 2, \dots, n_x - 1$.



(a) Indeksering av vertekser i trekantene.



(b) Indeksering av nabotrekantene.

Figur 2

Illustrasjon av indeksering av trekantene og naboinformasjon per kvadrat i rutenettet. Trekantene er indeksert med klokka fordi de skal brukes i Unity sitt venstrehendte koordinatsystem.

Videre er det to trekantene T_{even} og T_{odd} per kvadrat med henholdsvis partallsindeks og oddetallsindeks. Altså kan området deles opp i $(n_x - 1) \cdot (n_y - 1)$ antall kvadrater med en partallstrekant

$$T_{\text{even}} = 2(j + i \cdot (n_y - 1)), \quad T_{\text{odd}} = T_{\text{even}} + 1.$$

Indeksering med naboinformasjon kan skrives til fil på formen

$$\begin{aligned} & 2(n_x - 1) \cdot (n_y - 1) \\ & v_0 \ v_1 \ v_2 \ T_{\text{odd}} \ T_0 \ T_1 \\ & v_1 \ v_2 \ v_3 \ T_2 \ T_{\text{even}} \ T_3 \\ & \dots \end{aligned}$$

Hvor første linje er totalt antall trekantene, og hvert par med linjer etter holder indekser og naboinformasjon for partall- og oddetallstrekanter i for et vilkårlig kvadrat. Se 'Utdrag av indices.txt' for eksempel på resulterende indekseringsdata.

Ball på trekantflate

For å simulere baller lagde jeg klassen `BallPhysics` som legges til som komponent på en instance av Unitys innebygde sfære-primitiv. Klassen har en referanse til `TriangleSurface`-objektet den skal rulle på. Videre definerer jeg parametere for ballens masse [kg], radius [m], friksjon [dimensjonsløs] og bounciness [dimensjonsløs].

Kollisjonsdeteksjon

For å dettektere kollisjon med trekantflaten, så vil ballen kalle `GetCollision`-metoden i `TriangleSurface` objektet i hver `FixedUpdate`. Ballen gir da posisjonen til senteret sitt `p` til `TriangleSurface`, som bruker barysentriske koordinater til å finne trekanten ballen er på (Nylund, 2023, ss.78-79). Videre finner `TriangleSurface` punktet `s` på trekanten som er på linje med senteret i ballen langs `y`-aksen.

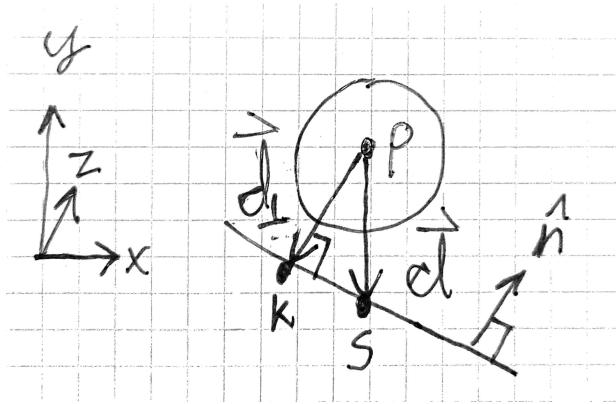
**Figur 3**

Diagram av kollisjonsdeteksjon. p er posisjonen til ballens senter, s er punktet på flaten med samme x - og z -koordinat som ballens senter, og k er punktet på flaten nærmest ballens senter.

For å sjekke om ballen er i kontakt med flaten beregner jeg først punktet k på flaten som er nærmest ballens senter p

$$k = p + ((s - p) \cdot \hat{n}) \hat{n},$$

og sjekker om differansen $|p - k|$ er mindre eller lik radiusen r til ballen. I tilfellet hvor $|p - k| \leq r$, så er det kollisjon. For å unngå at ballen beveger seg gjennom kollisjonsflaten setter jeg så ballens posisjon til å være i radius avstand fra kontaktpunktet langs enhetsnormalen

$$p_{\text{korrigeret}} = k + r \cdot \hat{n}.$$

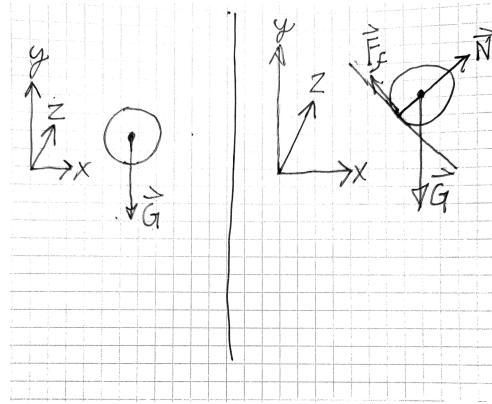
For å simulere spretting endrer jeg ballens hastighet til å bli

$$\vec{v}_{\text{etter}} = \vec{v}_{\text{før}} - (1 + b) (\vec{v}_{\text{før}} \cdot \hat{n}) \hat{n},$$

hvor $b \in [0, 1]$ er bounciness-parameteren til ballen, og representerer hvor stor brøkdel av normalkomponenten av hastigheten som skal reflekteres ved kollisjon. For $b = 0$ så mister ballen all hastighet normalt på planet. Hvis $b = 1$ så vil ballen få like stor hastighet normalt på planet som før kollisjonen, men i motsatt retning.

Glidefysikk

For å simulere fysikken på ballen, så finner jeg først alle kreftene på ballen i både tilfellet når den er og ikke er i kontakt med trekantflaten.

**Figur 4**

Frilegmediagram av ballen i både fritt fall, og på trekantflaten, med Unity sitt venstrehendte koordinatsystem tegnet inn.

I begge tilfeller er tyngdekraften på ballen lik

$$\vec{G} = m\vec{g} = m(-9.81 \text{ m/s}^2)\hat{j}$$

hvor m er ballens masse i kilogram.

I tilfellet hvor ballen er i kontakt med trekantflaten, så opplever ballen en normalkraft

$$\vec{N} = -(\vec{G} \cdot \hat{n})\hat{n},$$

hvor \hat{n} er enhetsnormalvektoren til trekanten ballen er i kontakt med. Videre modellerer jeg også en glidefriksjonskraft

$$\vec{F}_f = -\mu|\vec{N}|\hat{v}_{\parallel}, \quad \hat{v}_{\parallel} = \frac{\vec{v} - (\vec{v} \cdot \hat{n})\hat{n}}{|\vec{v} - (\vec{v} \cdot \hat{n})\hat{n}|}.$$

hvor μ er en kinetisk friksjonskonstant, og \vec{v} hastighetsvektoren til ballen. Friksjonskraften virker da alltid mot bevegelsesretningen langs flaten ballen er i kontakt med.

Jeg finner så akselerasjonen til ballen ved Newton's andre lov, slik at

$$\vec{a} = \frac{\vec{G} + \vec{N} + \vec{F}_f}{m},$$

og integrerer hastighet \vec{v} og posisjon \vec{p} med Forward-Euler algoritmen

$$\begin{aligned}\vec{v}_{t+\Delta t} &= \vec{v}_t + \vec{a} \cdot \Delta t, \\ p_{t+\Delta t} &= p_t + \vec{v}_{t+\Delta t} \cdot \Delta t,\end{aligned}$$

Hvor $\Delta t = 0.02 \text{ s}$.

Rulling på flere trekanter

I tilfellet hvor ballen krysser fra en trekant til neste, så har jeg implementert to separate måter å håndtere det på. I tilfellet hvor ballen har en bounciness som er større enn 0, så håndteres all kollisjon med den delvise refleksjonen beskrevet i underseksjon 'Kollisjonsdeteksjon'

$$\vec{v}_{\text{etter}} = \vec{v}_{\text{før}} - (1 + b) (\vec{v}_{\text{før}} \cdot \hat{n}) \hat{n}.$$

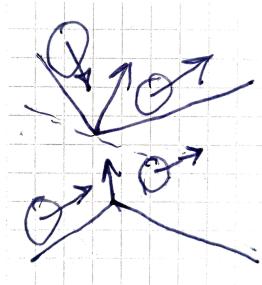
I tilfellet hvor ballens bounciness er satt til å være 0, så bruker jeg i stedet en refleksjon av planet med normal lik gjennomsnittsnormalen til de to involverte trekantene (Nylund, 2023, ss.120-124).

$$\vec{v}_{\text{etter}} = \vec{v}_{\text{før}} - 2 (\vec{v}_{\text{før}} \cdot \hat{n}) \hat{n}, \quad \hat{n} = \frac{\hat{n}_{\text{før}} + \hat{n}_{\text{etter}}}{|\hat{n}_{\text{før}} + \hat{n}_{\text{etter}}|}.$$

Dette bevarer farten til ballen over trekantbyttet. Jeg har også valgt å kun gjøre denne refleksjonen i tilfellet hvor

$$(\hat{n}_{\text{før}} \times \hat{n}_{\text{etter}}) \cdot (\hat{n}_{\text{før}} \times \hat{v}_{\parallel}) < 0,$$

hvor \hat{v}_{\parallel} er enhetsvektoren i samme retning som komponenten av ballens hastighet som er parallel med trekanten. Betingelsen over, sørger for at hastigheten kun er bevart over konvekse skjøter relativt til positiv y -retning, slik at ballen kan komme i fritt fall hvis den sklir utfør et fall.



Figur 5

Korrigering av hastighetsretning ved konvekse, men ikke konkave skjøter. Positiv y -akse rettet oppover på figuren.

B-Splines og simulering av vassdrag

For å simulere vassdrag som produseres under ekstremnedbør, modelerer jeg et sett med enkeltregndråper i form av baller som følger fysikken beskrevet i forrige sekjson. Videre får hver ball en tilfeldig generert startposisjon innenfor et valgt volum over terrenget. Jeg velger så å lagre posisjonen til hver av regndråpene 15 gjevnt fordelte ganger over en tidsperiode på 30 s. De lagrede posisjonene blir så brukt som kontrollpunkter i xz -planet for en 2D kvadratisk B-Spline, implementert med deBoors algoritme (Nylund, 2023, ss.98-102), som skal representere enkeltdråpene sine bevegelse horisontalt. Splinen kan så tegnes ved å

evaluere punkter for 750 uniformt fordelt verdier av parameteren t , som jeg tegner rette linjer mellom. De tegnede punktene løftes opp på terrengoverflaten ved å beregne høyde med barysentriske koordinater.

For å simulere effekten av et slik vassdrag på en kampestone eller annet løsmateriale, så bruker jeg likningen for motstandskraft i fluid

$$\vec{F}_D = \frac{1}{2} C_D \rho A |\vec{v}| \vec{v},$$

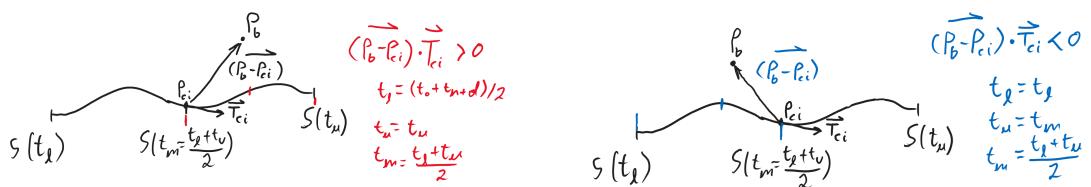
hvor C_D er dragkoeffisienten til legemet som påvirkes, $\rho = 1000 \text{ kg m}^{-3}$ er tettheten til fluidet, A er tversnittarealet som står normalt på fluidets bevegelsesretning, og $\vec{v} = \vec{v}_f - \vec{v}_b$ er fluidets hastighet relativt til legemet (Alexander & Cooker, 2016, s.7). I min simulasjon bruker jeg en ball med masse $m_b = 50 \text{ kg}$ og radius $r_b = 1 \text{ m}$ plasert på terrenget. Jeg har valgt en dragkoeffisient på $C_D = 0.5$ siden det gjelder for sfærer i mange forskjellige typer flyt (Hall, 2023). Videre har ballen et tversnittareal $A = \pi r_b^2$. Hastighetsvektoren til fluidet regner jeg ut som

$$\vec{v}_f = v \hat{T}_{\text{mean}}, \quad v = 3 \text{ m s}^{-1},$$

Hvor v er en konstant hastighet jeg har valgt for vassdraget, og \hat{T}_{mean} er den gjennomsnittet av enhetstangentvektoren i nærmeste punkt på hver B-Spline innenfor en radius på 5 m fra ballen. Jeg tilnærmer tangentvektoren $\vec{T} = S(t + \Delta t) - S(t)$ for en parameterverdi t som et lite steg $\Delta t = 0.5$ langs splinen S . For å finne parameterverdien tilsvarende nærmeste punkt P_c for posisjonen til ballen P_b på splinen S , så bruker jeg at

$$(P_b - P_c) \cdot \vec{T}_c = 0,$$

som jeg kan tilnærme en løsning for iterativt med Newton's midtpunktsmetode



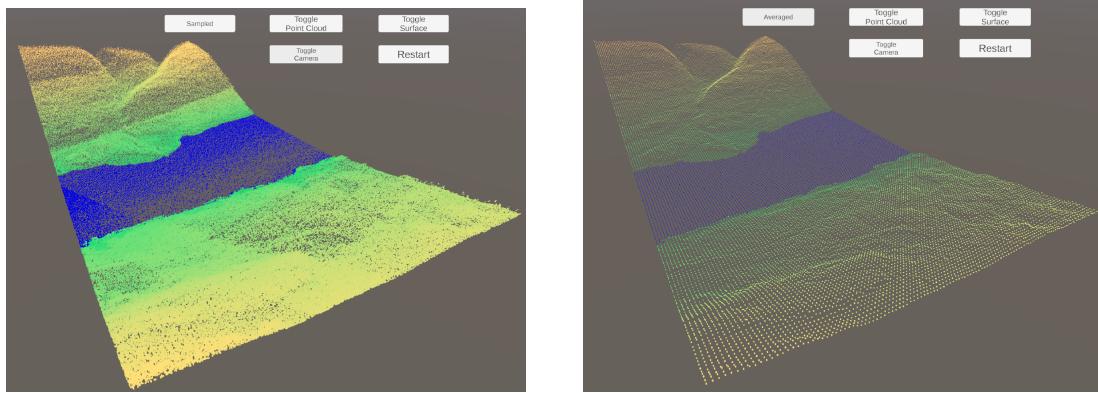
- (a) Velger øvre interval hvis prikkproduktet er positivt.
(b) Velger nedre interval hvis prikkproduktet er negativt.

Figur 6

Iterativ innskrenkning av parameterverdi for punkt $P_{c,i}$ på splinen S som er nærmest punktet P_b .

Resultater

Punktsky og glatting



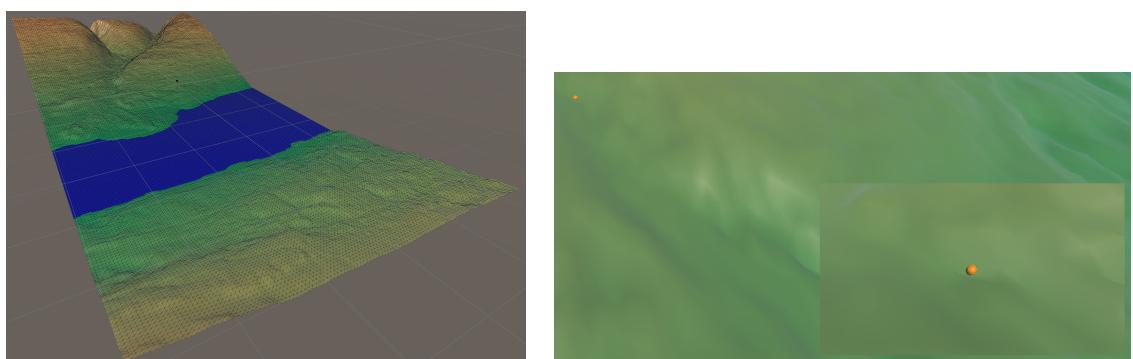
(a) Hvert hundrede punkt fra rådataene.

(b) Regulært glattede punkter.

Figur 7

Plott av punktskydataene.

Overflate og ball



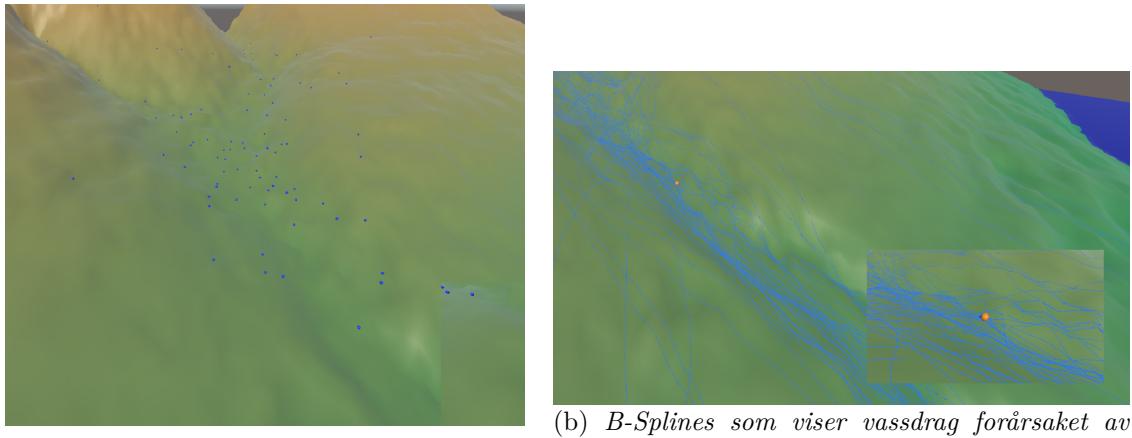
(a) Regulært indeksert trekantflate

(b) Oransj ball som sklir på flaten.

Figur 8

Plott av trekantflaten og to perspektiver av samme ball.

Spline og vassdrag



Figur 9

Plott av regn og vassdrag. Den oransje ballen flyter med vassdraget.

Antall splines	Bildefrekvens
100	167 fps
250	53 fps
500	4 fps

Tabell 1

Antall B-Splines (1 per regndråpe simulert), og tilsvarende gjennomsnittlig bildefrekvens.

Diskusjon

Punktsky

Fra [figur 7](#) ser jeg at punktskydataene er blitt prosesert riktig. Delfigur [\(a\)](#) viser vært hundrede av punktene innlest fra textfilen som LASzip genererte fra .laz-filen med rådata. Det er totalt 318905 punkter som rendres. Fargen er satt basert på høyden i vertex-shaderen 'GPUInstancing.shader'. Dataene er som forventet ugjevnt fordelt i det horisontale planet, og har varierende tetthet. Delfigur [\(b\)](#) viser de regulært glattede verteksene generert av `PointCloudSurfaceParser.cpp`, og er som forventet gjevnt fordelt i det horisontale planet med konstant tetthet innenfor området.

Flate og ball

[Figur 8](#) viser den regulært indekserte trekantflaten generert av `PointCloudSurfaceParser.cpp`, og at en ball kan plaseres og bevege seg på trekantflaten.

Spline og vassdrag

Som illustrert i [Figur 9](#), så er 100 individuelle regndroper i stand til å trille på trekantflaten samtidig. Dette kan brukes for å simpelt og raskt estimere hvor regn kommer til å flyte langs terrenget. Videre vises B-Splinene som visualiserer det estimerte vassdraget som antas følges av videre nedbør. Ballen som representerer løsmateriale blir fraktet med vassdraget og, basert på start posisjon, kan ende opp i bunnen av dalen som terrenget viser, eller bli sittende fast på veien på grunn av friksjonen på $\mu = 0.04$ for ballen.

Valget om å simulere retningen til vassdraget med tangentvektoren til B-Splinene er mindre enn optimalt ettersom algoritmen for å finne parameterverdiene for nærmeste punkt per spline skalerer dårlig for økt antall spliner, som vist i [Tabell 1](#). Aktuelle alternativer er å bruke en felt- eller diffusjonsbasert fluidsimmulering til å bestemme hastighetsvektorer per kvadrat i det regulære rutenettet som ligger til grunne for trekantflaten. En slik simulasjon hadde også antagelig kunne modellere et vassdrag med hastighet som varierer i rommet.

Konklusjon

Hensikten med dette prosjektet var å utforske konstruksjonen av trekantflater fra punktsky-måling av terreng, til en mulig modell for simulering av vassdrag og effekt av nedbør på løsmateriale. Konstruksjonen av trekantflate er fult mulig, og gir et modellert tereng som kan brukes til simulering av vær.

Valgt metode for visualisering av vassdrag og simmulering av flom er ikke egnet for fysisk komplekse situasjoner grunnet at ytelsen raskt går ned når man øker antall regndråper. Derimot er modellen aktuelt som verktøy for enkle visualiseringer av effekten av regn på enkeltlegemer.

Referanser

- Alexander, J., & Cooker, M. J. (2016). Moving boulders in flash floods and estimating flow conditions using boulders in ancient deposits (V. Manville, Red.). *Sedimentology*, 63(6), 1582–1595. <https://doi.org/10.1111/sed.12274>
- Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A., & Silva, C. T. (2017). A Survey of Surface Reconstruction from Point Clouds. *Computer Graphics Forum*, 36(1), 301–329. <https://doi.org/10.1111/cgf.12802>
- Hall, N. (2023 november). Drag of a Sphere. Hentet 27. november 2023, fra <https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/drag-of-a-sphere/>
- Isenburg, M. (2019 november). LASzip. <https://laszip.org/>
- Nylund, D. (2023 oktober). *MAT301 Matematikk III VSIM101 Visualisering Og Simulering Forelesningsnotater Og Oppgaver*. Høgskolen i Innlandet. Hentet 29. november 2023, fra https://drive.google.com/file/d/1cKrJ_vrKrcS1igT96_c09cjgJrRo442p/view?usp=sharing&usp=embed_facebook
- Unity Engine. (2023 mars). Hentet 20. april 2023, fra <https://unity.com>

Tillegg Prosjektfiler

Prosjektet er tilgjengelig her: <https://github.com/FunkMarvel/VisSimMappe.git> .
 All kode for prosesering av punktskydata er tilgjengelig i PointCloudSurfaceParser- undermappen i Git repoet,
<https://github.com/FunkMarvel/VisSimMappe/tree/e58f43b7870e46e7956c186ae735f39589018496/PointCloudSurfaceParser>
 All kode for Unity-simuleringen er tilgjengelig i VisSimMappeUnityProsjekt- undermappen av Git repoet,
<https://github.com/FunkMarvel/VisSimMappe/tree/e58f43b7870e46e7956c186ae735f39589018496/VisSimMappeUnityProsjekt>

Utdrag av vertices.txt

```
28800
(-600, 87.3003, -300)
(-600, 86.173, -295)
(-600, 84.6683, -290)
(-600, 82.7369, -285)
(-600, 81.8657, -280)
(-600, 83.2463, -275)
(-600, 81.257, -270)
(-600, 79.8496, -265)
(-600, 79.406, -260)
(-600, 79.7682, -255)
(-600, 78.7479, -250)
(-600, 79.5409, -245)
```

(-600, 80.1995, -240)
(-600, 78.4865, -235)
...

Utdrag av indices.txt

56882
0 1 120 1 -1 -1
1 121 120 238 0 2
1 2 121 3 1 -1
2 122 121 240 2 4
2 3 122 5 3 -1
3 123 122 242 4 6
3 4 123 7 5 -1
4 124 123 244 6 8
4 5 124 9 7 -1
5 125 124 246 8 10
5 6 125 11 9 -1
6 126 125 248 10 12
6 7 126 13 11 -1
7 127 126 250 12 14
...