Mappeoppgave H2023 VisSim

Stål Brændeland

October 2023

1 Introduksjon

For dette prosjektet valgte jeg å jobbe i Unity. Det gjør noen ting som innlesing av punkter og visualisering av de samme punktene som punktsky noe mer involvert, men det meste annet er mye bedre tilrettelagt - i tilegg er det da mulighet for å jobbe i Rider fremfor Qt Creator.

2 Konstruksjon og visualisering av 3D-terreng

2.1

Her valgte jeg et vilkårlig areal i nordmarka på hoydedata.no, som ble tilsendt meg på mail som en .laz fil. Deretter konverterte jeg det til enkelt innlesbar xyz data med las2txt, et redskap fra LAStools-samlingen (`https://github.com/LAStools/LAStools`). Filen kan åpnes direkte i fil-utforsker fra prosjekt-mappen (Assets/StreamingAssets/terrain.txt) eller via Unity-prosjektet.

2.2

Referer til Unity-prosjektet for å se det endelig meshet. For å få ok performance (over 20FPS) når jeg skulle tegne nesten fem millioner kuber var det eneste som ga bra nok resultater indirekte GPU-instansiering. (Jeg valgte en kube til å representere ett punkt - her hadde noe som GL_Points i openGL vært nyttig.) For å få indirekte GPU-instansiering til å funke brukte jeg en youtube-video fra tarodev (`https://www.youtube.com/watch?v=6mNj3M1il_c`) og tilsvarende GitHub prosjekt som referanse.

2.3

Maks antall vertekser du kan ha i et enkelt mesh i Unity er 65355 (16 bit buffer), så jeg valgte ut så mange punkter med jevne mellomrom fra de 4878263 punktene med LiDAR-data. Så lager jeg en regulær triangulering som beskrevet i 10.6 i forelesningsnotatene, men med xMin,zMax til xMax,zMin i stedet for yMax og yMin, siden y er opp i Unity. Z er også gitt som opp i høydedataen, så i koden kan du se at jeg bytter om z og y-koordinatene. Jeg justerer også alle koordinat-verdiene ned (i essens bare å flytte meshet nærmere origo) så ikke terrenget skal ende opp så fryktelig langt unna origo. Jeg bruker en klasse, Triangle,

for å holde oversikt over nabo-trekanter. Tekstfil med indekseringsdata og nabodata kan du finne i prosjekt-mappen (Assets/StreamingAssets/triangles.txt), og interaktivt i Unity-prosjektet. Den regeneres hvis du åpner den via prosjektet, med den samme dataen som meshet i **2.4** tegnes med.

2.4

Referer til Unity-prosjektet.

3 Simulering av nedbør og vassdrag

3.1

Newtons andre lov sier at når et legeme blir påvirket av én eller flere krefter, vil det få en akselerasjon i den retningen kreftene virker. Summen av kreftene på legemet er lik legemets masse ganger dets akselerasjon. Denne loven kan beskrives med formelen

$$F = ma$$

Massen til ballen er 1. Tyngdekraften er representert av Unity sin Physics.Gravity, altså vektoren

$$\vec{g} = [0, -9.81, 0]$$

Physics.Gravity har enheten m/s^2, så vi ganger med massen for å få en ny vektor som har enheten N

$$\vec{G} = \vec{g} \times 1$$

Ballen bruker så i hvert tidssteg barysentriske koordinater for å finn ut hvilken trekant den befinner seg på. Når vi vet hvilken trekant det dreier seg om kan trekant-klassen finne enhetsnormalen sin ved å ta kryssproduktet av to av kantene sine og normalisere det. Hvis en trekant består av vertekser a, b, c

$$\vec{ab} = b - a$$

$$\vec{ac} = c - a$$

$$\vec{n} = ab \times ac$$

Vi kaller $\vec{n}$ normalisert for $\hat{n}$. Normalkraften blir da minus en ganger prikkproduktet av denne vektoren og vektoren for tyngdekraften. Vi ganger den med enhetsnormalen for å gi den retning og representere den som en vektor.

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

Den endelige kraften som virker på ballen, x-komponenten til tyngdekraften langs trekant-planet kan vi finne ved å legge sammen de to vektorene.

$$\vec{F} = \vec{G} + \vec{N}$$

Ved å bruke formelen fra Newtons andre lov kan vi finne akselerasjonen

$$a = \frac{F}{m}$$

Med akselerasjonen er det trivielt å finne fart og deretter posisjon (bare å gange med tid) - slik kan vi simulere ballens posisjon ettersom den beveger seg i terrenget.

3.1.1

Foreløpig er det ingen kollisjon mellom ballene.

3.2

Regndråpene bruker akkurat samme logikk basert på samme fysiske lover som ballene til å bevege seg over terrenget. Det er samme klasse (RollingBall), bare med en boolean som bestemmer litt annen logikk i forhold til vassdrag og lignende.

3.3

Når regndråpene har mistet så og si all fart og satt seg i ro danner de et lite vann som kan gro ettersom nye regndråper faller på samme sted. Når de forsvinner, enten fra å danne et nytt vann eller å ha blitt absorbert inn i ett eksisterende ett, lager de en spline i verdenen som består av B-spline segmenter med tre kontrollpunkter hver. Knutevektoren er ikke "clamped" det vil si at vi får en åpen spline som ikke går gjennom endepunktene.

3.4

Baller som har lagt seg til ro i terrenget vil flyte oppå vannet som dannes, slik at hvis det blir nok vann vil de rulle avgårde til lavere terreng.

Diskusjon

Kilder

`https://drive.google.com/file/d/1cKrJ_vrKrcS1igT96_c09cjgJrRo442p`
`https://www.youtube.com/watch?v=6mNj3M1il_c`
`https://github.com/Matthew-J-Spencer/pushing-unity`