Rapport

Rebekka Oskarsen

3. Desember 2024

Innholdsfortegnelse

1 Innledning

I denne rappoerten skal jeg snakke om to hoveddeler fra mappeoppgaven. Først visualisering av terrenget, så vise b-spline flate med baller. Målet med arbeidet er å visualisere normaler og triangulering, med også bevegelse av objecter.

2 Metode

I del 1 av mappeoppgaven brukte jeg terrengdata fra Canvas, hvor læreren har gjort datafilen tilgjengelig til bruk. Datafilen har læreren konvertert til en tekstfil med antall punkter på første linje, og x y z koordinater for hvert punkt på egene linjer.

Visualisering av terrenget ble gjort med tilpasset kamera og renderingsvindu. Punktene fra datasettet blir vist som en punktsky.

Listing 1: Triangulering

```
if (topLeft != -1 && topRight != -1 && bottomLeft != -1
   && bottomRight != -1)
{
    // F rste trekant
    indices.push_back(topLeft);
    indices.push_back(bottomLeft);
    indices.push_back(topRight);

    // Andre trekant
    indices.push_back(topRight);
    indices.push_back(bottomLeft);
    indices.push_back(bottomRight);
}
```

Koden er fra PunktSky.cpp, linje 192-203.

Her lager jeg 2 trekanter som blir til en firkant. Se figur 1 for resultat. Dette

er viktig for å kunne beregne normalvektoren og for å lage phong shader.

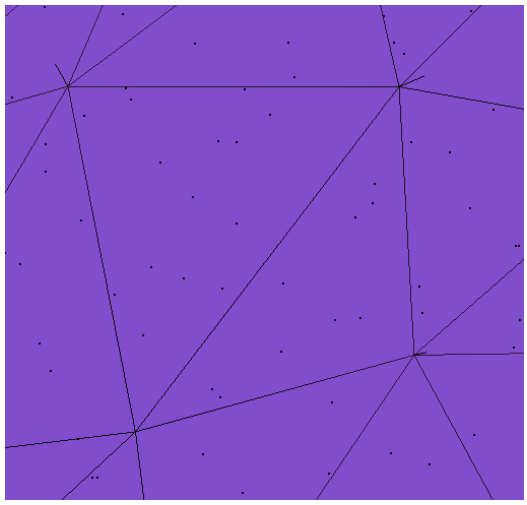

Figure 1: Trangulering

For å finne normalen brukte jeg kryssproduktet men også må ha punktene til hjørnet på en trekant.

Listing 2: Normalen

```
    for (size_t i = 0; i < indices.size(); i += 3)
    {
        unsigned int i0 = indices[i];
        unsigned int i1 = indices[i + 1];
        unsigned int i2 = indices[i + 2];

        glm::vec3 v0 = points[i0];
        glm::vec3 v1 = points[i1];
        glm::vec3 v2 = points[i2];

        glm::vec3 normal = glm::normalize(glm::cross(v1
           - v0, v2 - v0)); // Kryssproduktet for
           beregne trekantens normale

        // Normale vektorer til 3 punkter
        normals[i0] += normal;
```

```
        normals[i1] += normal;
        normals[i2] += normal;
    }
```

Koden er fra PunktSky.cpp, linje 212-228.

Måtte sentere punktene og da fikk jeg hjelp fra en i klassen. Dette er koden dem hjalp meg med.

Listing 3: Sentrere punktene

```
  glm::vec3 min(FLT_MAX), max(-FLT_MAX);

  // Oppdaterer min- og maksverdier for    finne
     bounding box
    min = glm::min(min, position);
    max = glm::max(max, position);

// Beregner midtpunktet og justerer alle punktene
    glm::vec3 center = (min + max) / 2.0f;
    for (auto& position : points)
    {
        position -= center;
    }

    glm::vec3 min(FLT_MAX), max(-FLT_MAX);
    for (const auto& point : points)
    {
     min = glm::min(min, point);
     max = glm::max(max, point);
    }
```

Koden her er fra litt forskjelig steder på PunkSky.cpp, de ligger under void "PunktSky::loadAndCenterPoints(const std::string filename)"

I del 2 brukte jeg litt av forelesningsnotatene. Brukte informasjon fra forelesningsnotatene til læreren. Kapittel 12 om B-Spline flater, der brukte jeg skjøtevektorene og kontrollpunktene som man finner på 12.1.

Listing 4: Kontrollpunkter og Skjøtevektor

```
// Kontrollpunkter for en bikvadratisk B-spline flate
  controlPoints =
  {
      {0, 0, 0}, {1, 0, 0}, {2, 0, 0}, {3, 0, 0},
      {0, 1, 0}, {1, 1, 2}, {2, 1, 2}, {3, 1, 0},
      {0, 2, 0}, {1, 2, 0}, {2, 2, 0}, {3, 2, 0}
  };

  // Skjotevektorer
  uKnots = { 0, 0, 0, 1, 2, 2, 2 };
  vKnots = { 0, 0, 0, 1, 1, 1 };
```

Koden er fra BSplineSurface.cpp, linje 11-20.

3 Resultater

I del 1 fikk jeg resultatet jeg ville ha. Terrenget ble som jeg ville at den skulle være, se figur 2. Phong shaderen fremhevde terrengets overflate detaljer. Dette var mulig fordi jeg la til punktene og så fikk triangler mellom punktene, deretter normaler.

Fikk ikke resulatet jeg ville ha på del 2. B-spline flaten har 3 baller som jeg ville ha men de beveger seg ikke på flaten og med hvordan flaten er. De beveger seg under flaten og ser dem bare når jeg har på wireframe eller blir dem bare dekket av flaten. Se figur 3.

Flaten har phong shader på seg, se på figur 4.

Ballene kolliderer med hverandre og beveger seg ikke utenfor flaten. Flaten har treangler og normaler for å få en glatt overflate. Man må velge startposisjonen til ballene og når man har gjort det så ser man hvor dem er på flaten

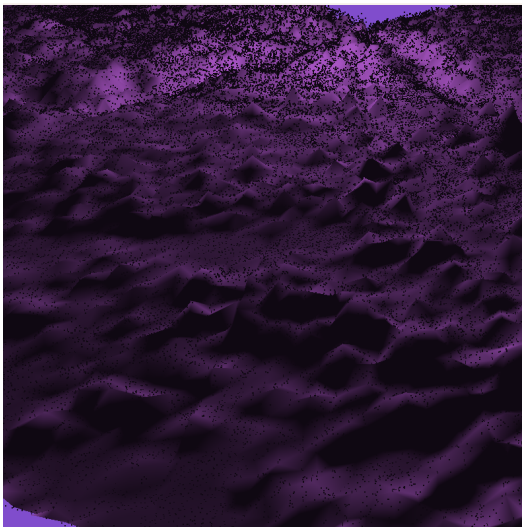

Figure 2: Terreng med phong shader

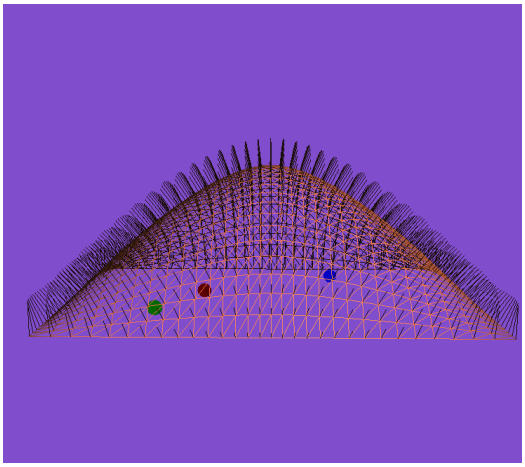

Figure 3: B-spline flate med wireframe og 3 baller

og ved å trykke på opp pilen på tastaturen så starter hastigheten til ballene. La inn hastigheten som 0 som startsfart for å kunne se at startposisjonene man har valgt for dem, stemmer.

Når ballene kolliderer med hverandre så spretter dem vekk fra hverandre.

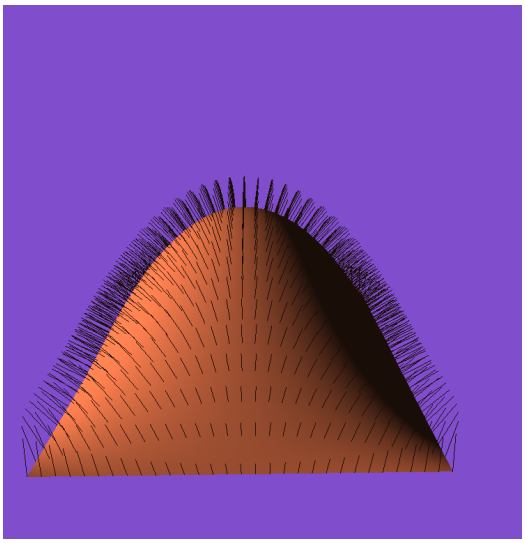

Figure 4: Flaten med phong shader

4 Diskusjon

Resultatet fra del 1: Jeg hadde ikke klart å få vist terrenget hvis det ikke var for at jeg fikk hjelp av en i klassen med å sentrere punktene fra datafilen. Jeg misforsto oppgave 1.4 når det sto "som i oblig 2". Derfor bruker jeg b-spline fra oblig 2.

Resultatet fra del 2: Jeg burde ha gjort mer arbeid med koden for å teste mer med høydeverdiene på flaten.

5 GitHub Link

https://github.com/RebekkaOskarsen/234.git

6 Ressurser

Lys shader
https://learnopengl.com/Lighting/Basic-Lighting

Ball og kollisjon
https://github.com/RebekkaOskarsen/Oblig1_Spillmotorarkitektur.git

Terrengdata
https://drive.google.com/file/d/1aQLgZpqdRgPml9yJLr5Y5IMpm5uoLLN5/view?usp=sharing

Forelesningsnotater
https://drive.google.com/file/d/1iOIm-Orpi-zYynCo7TyEQccLohRI5HBh/view?usp=sharing