3D-visualisointi

1 Yleiskuvaus

Tämä ohjelma on eräs yksin tehty ensimmäisen vuoden ohjelmointiprojekti.

Ohjelma lukee tiedostosta kuvauksen kolmiulotteisesta ympäristöstä, jossa on seiniä. Ympäristössä voi liikkua melko vapaasti ja ohjelma näyttää visualisoinnin kyseisen sijainnin perusteella. Näkymä päivittyy reaaliaikaisesti liikkeen ja katselusuunnan mukana. Seinät ovat yksivärisiä ja niiden väri riippuu niiden etäisyydestä.

2 Käyttöohje

Ohjelma voidaan ajaa `sbt`:llä. `Sbt` toimii Windows 10:llä asennuksen jälkeen ainakin git bashissa. Ohjelma ajetaan projektikansiossa komennolla `sbt run`.

Ohjelma näyttää aluksi tyhjän alueen. Varsinainen ympäristö ladataan valitsemalla yläpalkista `Tiedosto` ja valikosta `Avaa`. Jos valikosta valitsee myöhemmin `Sulje`, pääsee takaisin tyhjään ympäristöön.

Kun ympäristö on latautunut, näyttää ohjelma heti visualisaation alueesta. Alueessa voi nyt alkaa liikkua.

Yksinkertainen liikkuminen on mahdollista alapalkin nappuloiden avulla, mutta varsinaisesti ohjelmassa on tarkoitus liikkua näppäimistön avulla. Näppäimet ja niitä vastaavat komennot on selitetty taulukossa 1.

Seinät ladataan tiedostosta `worlddata.csv`, jonka täytyy sijaita projektin hakemistossa. Tiedoston sisällön voi muokata itse, joten omia alueita on mahdollista luoda ja muokata.

Ohjelman voi sulkea milloin tahansa painamalla rastia ikkunan yläkulmassa.

näppäin	**toiminto**
`W`	liiku eteenpäin
`A`	liiku vasemmalle
`S`	liiku taaksepäin
`D`	liiku oikealle
`R`	palaa alkutilanteeseen
välilyönti	liiku ylöspäin
shift	liiku alaspäin
nuoli ylös	katso ylöspäin
nuoli alas	katso alaspäin
nuoli vasemmalle	katso vasemmalle
nuoli oikealle	katso oikealle

Taulukko 1: näppäintoiminnot

3 Ohjelman rakenne

Ohjelma on jaettu erikseen sisäiseen malliin ja käyttöliittymään. Käytettyjä luokkia ovat Wall, World, ProjectionSurface ja Camera. Lisäksi käytössä ovat yksittäisoliot Transforms sekä UI. Matriisien kuvaamiseen käytettiin EJML-kirjaston SimpleMatrix-luokkaa.

Ohjelma sisältää seiniä (Wall) 3-ulotteisessa avaruudessa[1]. Seinien koordinaatteja voidaan muuntaa matriisikertolaskuilla. Seinän koordinaateista otetaan 2-ulotteinen projektio ProjectionSurface:n avulla ja projektio piirretään UI-luokassa.

- Wall on mallinnuksen keskeinen luokka, joka kuvaa piirrettäviä seiniä. Seinä sisältää erikseen alkuperäiset, nykyiset sekä seuraavat 3-ulotteiset koordinaatit. Seinä sisältää kokoelman "törmäyspisteitä", joilla seinä täytetään. Näitä käytetään seinän etäisyyden määrittämiseen. Seinä tietää nykyiset ja seuraavat törmäyspisteet. Lisäksi seinä tietää 2-ulotteiset koordinaattinsa sekä 2-ulotteiset törmäyspisteet (jotta ne voidaan halutessa piirtää).

  Seinä pystyy tarkistamaan törmäyksen, tekemään pisteilleen muunnoksen annetulla matriisilla sekä pyytämään annetulta projektiotasolta itselleen uusia 2D-koordinaatteja. Muunnokset tehdään kahdessa vaiheessa, jotta ne voidaan perua törmäyksen takia.

- World kuvaa ohjelman aluetta. World sisältää kokoelman kaikista käytetyistä seinistä, viittauksen käytettävään projektiotasoon sekä viittauksen Transforms-olioon.

  World-luokalla voidaan tehdä asioita, jotka vaikuttavat koko kuvattavaan alueeseen. World voi pyytää seiniä tekemään halutun muunnoksen tai päivittämään omia 2d-koordinaattejaan. World voi myös ladata seinät tiedostosta, poistaa seinät tai palauttaa alueen alkutilaan.

- ProjectionSurface kuvaa mallinnuksessa käytettävää tasoa, johon lasketaan projektiot 3d-koordinaateista. Ajatuksena on, että katselupiste on aina pisteessä $(0, 0, 0)$, ja tämän edessä on projektiotaso, joka on myös aina samassa paikassa. Sen sijaan ympärillä oleva alue muuttuu. Projektiotaso on yz-tason suuntainen ja sijaitsee oletuksena kohdassa $x = 1$ (joskin sijainti on mielivaltainen). Projektioihin liittyvä laskenta ja algoritmit on toteutettu tässä luokassa.

- Camera luokka kuvaa näkökulmaan liittyvää tietoa ja sitä käytetään käyttöliittymässä World-luokan lisäksi. Piirrettävään alueeseen liittyy katselukulman leveys (FOV), joka annetaan tässä luokassa. Tämän arvon perusteella lasketaan muita arvoja.

  Lisäksi Camera-luokka sisältää "katselukulman" sekä "korkeuden". Käyttöliittymä muuttaa katselukulmaa aina y-suuntaisen kierron yhteydessä. Tämän arvon avulla liikkuminen voidaan toteuttaa suhteessa "maanpintaan" – alkuperäiseen xy-tasoon. Lisäksi tämän kulman avulla voidaan liikuttaa horisonttia kierron yhteydessä. "Korkeutta" muutetaan käyttöliittymässä aina kun liikutaan maanpinnan normaalin suunnassa. Tämän

[1] Oikeastaan ohjelma käyttää homogeenisia koordinaatteja, joissa on neljäntenä koordinaattina arvo 1. Tämä mahdollistaa matriisien käytön myös translaatioon.

arvon avulla voidaan varmistaa, että ohjelma ei anna liikkua maanpinnan alapuolelle.

- Transforms-yksittäisolio luo muunnoksiin soveltuvia matriiseja annettujen parametrien pohjalta.
- Käyttöliittymää kuvaa UI-yksittäisolio. UI luo uudet World- sekä Camera-oliot. UI on toteutettu Swingin avulla ja käyttääkin Swingin luokkia. UI perii SimpleSwingApplicationin. UI koostuu MainFramesta, jonka sisällä on BorderPanel sekä MenuBar. BorderPanel koostuu edelleen nappulat sisältävästä FlowPanelista sekä Panelista, jossa on toteutettu varsinainen piirtäminen.

  UI käyttää Timer-oliota tapahtumien toistamiseen tietyin väliajoin.

  UI:ssa tapahtuva piirtäminen toimii Graphics2D:n avulla. Piirtäminen käyttää tämän fillPolygon-metodia.

  Olio sisältää myös piirtämisen apumetodeja.

4 Algoritmit

4.1 Muunnokset

Ympäristössä liikkuminen on toteutettu matriisikertolaskuilla. Jos seinän jokin koordinaatti on $\vec{v}$, on uusi koordinaatti $\vec{v}' = \mathbf{M}\vec{v}$, jossa M on haluttu muunnos.

Esimerkiksi y-suuntaista translaatiota vastaa matriisi

$$\mathbf{T}_{\mathrm{y,\ d}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -d \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Jos y-suunnassa on käännytty θ astetta ja halutaan esim. liikkua "alkuperäisen" x-akselin suuntaan, täytyy y-suunnan käännös ensin kumota, sitten tehdä translaatio ja lopuksi tehdä uusi y-suuntainen käännös, siis

$$\vec{v}' = \mathbf{R}_\theta \mathbf{T}_{\mathrm{x,d}} \mathbf{R}_\theta^{-1} \vec{v} \tag{2}$$

Tämän vuoksi käyttöliittymä päivittää y-suuntaisten käännösten yhteydessä käännöskulmaa. Käytännössä ohjelma ei laske rotaatiomatriisille käänteismatriisia vaan pyytää uuden rotaatiomatriisin jonka kiertokulma on nykyisen kulman vastaluku. Tällaista yhdistelmää käytetään x- ja z-suuntaisten translaatioiden sekä z-suuntaisen kierron yhteydessä.

4.2 Vektorin leikkauspiste

Olkoon pisteen paikkavektori

$$\vec{v} = \begin{pmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \\ 1 \end{pmatrix} \tag{3}$$

Koska projektiotaso on yz-tason suuntainen, saadaan paikkavektorien leikkauspisteet laskemalla

$$\vec{v}_{\text{leikkaus}} = \frac{\vec{v}}{x} x_{\text{taso}} \tag{4}$$

, jossa x_{taso} on tason x-koordinaatti. Tästä saadaan halutut 2D-koordinaatit valitsemalla vektorin y- ja z-komponentit.

Ohjelma vaatii, että $x > 0$.

Seinien ja törmäyspisteiden 2D-muunnokset tehdään kutsumalla kaikille pisteille vektorien leikkauspisteet ottavaa apumetodia. Voi kuitenkin olla, että seinä täytyy leikata ennen muunnoksen laskemista.

4.3 Seinien leikkaaminen

Jos osa seinän kulmista on x-akselin negatiivisella puolella, ei niille lasketa projektiota. Seinä voidaan kuitenkin piirtää, koska seinästä voidaan leikata jokin x-akselin positiiviselle puolelle jäävä osa ja piirtää seinä leikkauksen perusteella.

Algoritmi käyttää leikkauksen laskemiseen apumetodia
`cut(x1: SimpleMatrix, x2: SimpleMatrix): SimpleMatrix`
Se vastaanottaa vektorin päätepisteet $\vec{x}_1$ ja $\vec{x}_2$, joista jälkimmäinen on tason negatiivisella puolella. Ensin se laskee niiden välisen vektorin $\vec{v}$ ja sen yksikkövektorin $\vec{u}$. Aluksi on $t = 0$. Apumetodi käy silmukalla läpi lauseketta

$$\vec{c} = \vec{x}_1 + t\ \vec{u} \tag{5}$$

kasvavilla t:n kokonaislukuarvoilla niin kauan kun $\vec{c}$ pysyy positiivisella puolella. Lopuksi apumetodi palauttaa leikatun pisteen $\vec{c}$.

Algoritmi tunnistaa ensin, mikä leikkaus tunnetuista tapauksista on kyseessä, ja sen mukaan luo uudet pisteet, joista osa on korvattu ennalta määrätyillä apumetodin kutsuilla.

4.4 Törmäyspisteiden luonti

Jos seinistä tiedetään vain kulmien koordinaatit, on vaikea laskea seinän etäisyys ja siten tunnistaa törmäys. Tämän takia seinät täytetään törmäyspisteillä (kokoelma pisteiden paikkavektoreita), joista etäisyys on helpompi määrittää.

Olkoon seinän vasen yläkulma $\vec{v}$, vektori vasemmasta yläkulmasta oikeaan yläkulmaan $\vec{a}$ ja vektori vasemmasta yläkulmasta vasempaan alakulmaan $\vec{b}$. Näitä vastaavat normaalivektorit $\vec{u}_a$ ja $\vec{u}_b$.

Algoritmi käy läpi kaikki mahdolliset kokonaislukuarvot s ja t, joiden avulla muodostetaan törmäyspiste

$$\vec{p} = \vec{v} + s\ \vec{u}_a + t\ \vec{u}_b \tag{6}$$

, kun on oltava $0 \leq s \leq ||\vec{a}||$ ja $0 \leq t \leq ||\vec{b}||$. Algoritmi palauttaa kaikki näin saadut pisteet.

Tämä menetelmä törmäysten tunnistamiseen ei kuitenkaan ole tehokas.

5 Tiedostot

Ohjelma lukee ympäristön seinät csv-tyyppisestä tekstitiedostosta. Tiedostossa on listattuna seinien kulmien koordinaatteja puolipisteellä erotettuna, järjestyksessä x y z. Seinän pisteet tulee esittää järjestyksessä kiertäen seinän ympärystä

pitkin. Seuraava piste on seuraavalla rivillä. Jos seinän pisteiden jälkeen halutaan lisätä seuraavan seinän pisteet, tulee välissä olla rivi, jossa on kaksi puolipistettä (siis ei mitään pisteitä, mutta erottimet ovat jäljellä). Tällainen rivi voi olla myös tiedoston lopussa, mutta ei ole välttämätön. Rivin lopussa saa olla kommentti, joka alkaa risuaidalla "#".

```
x1;y1;z1
x2;y2;z2
x3;y3;z3
x4;y4;z4
;;
```

mutta ei esimerkiksi

```
x1;y1;z1
x3;y3;z3
x2;y2;z2
x4;z4;y4
;;
```

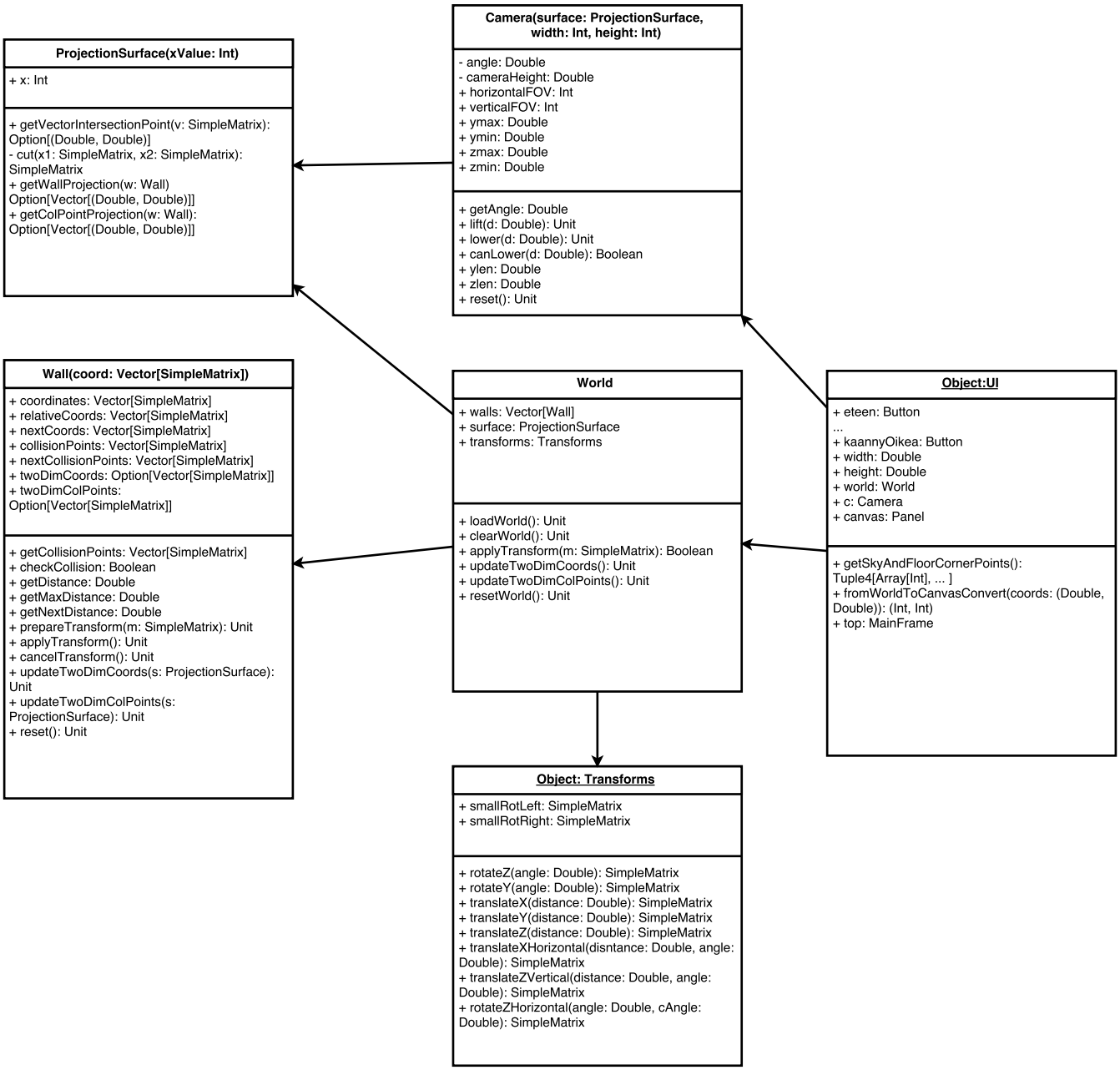
Esimerkiksi valmiina mukana olevassa tiedostossa kaksi ensimmäistä seinää on esitetty muodossa

```
4;-3;3  #vasemman käytävän lähempi seinä
4;-11;3
4;-11;-1
4;-3;-1
;;
4;-11;3  #vasemman käytävän pääty
8;-11;3
8;-11;-1
4;-11;-1
```

6 Viitteet

http://ejml.org/javadoc/org/ejml/simple/SimpleMatrix.html
http://ejml.org/javadoc/org/ejml/simple/SimpleBase.html
https://en.wikipedia.org/wiki/Camera_matrix

7 Liitteet


ProjectionSurface(xValue: Int)
+ x: Int
+ getVectorIntersectionPoint(v: SimpleMatrix): Option[(Double, Double)]
- cut(x1: SimpleMatrix, x2: SimpleMatrix): SimpleMatrix
+ getWallProjection(w: Wall) Option[Vector[(Double, Double)]]
+ getColPointProjection(w: Wall): Option[Vector[(Double, Double)]]
Camera(surface: ProjectionSurface, width: Int, height: Int)
- angle: Double
- cameraHeight: Double
+ horizontalFOV: Int
+ verticalFOV: Int
+ ymax: Double
+ ymin: Double
+ zmax: Double
+ zmin: Double
+ getAngle: Double
+ lift(d: Double): Unit
+ lower(d: Double): Unit
+ canLower(d: Double): Boolean
+ ylen: Double
+ zlen: Double
+ reset(): Unit
Wall(coord: Vector[SimpleMatrix])
+ coordinates: Vector[SimpleMatrix]
+ relativeCoords: Vector[SimpleMatrix]
+ nextCoords: Vector[SimpleMatrix]
+ collisionPoints: Vector[SimpleMatrix]
+ nextCollisionPoints: Vector[SimpleMatrix]
+ twoDimCoords: Option[Vector[SimpleMatrix]]
+ twoDimColPoints: Option[Vector[SimpleMatrix]]
+ getCollisionPoints: Vector[SimpleMatrix]
+ checkCollision: Boolean
+ getDistance: Double
+ getMaxDistance: Double
+ getNextDistance: Double
+ prepareTransform(m: SimpleMatrix): Unit
+ applyTransform(): Unit
+ cancelTransform(): Unit
+ updateTwoDimCoords(s: ProjectionSurface): Unit
+ updateTwoDimColPoints(s: ProjectionSurface): Unit
+ reset(): Unit
World
+ walls: Vector[Wall]
+ surface: ProjectionSurface
+ transforms: Transforms
+ loadWorld(): Unit
+ clearWorld(): Unit
+ applyTransform(m: SimpleMatrix): Boolean
+ updateTwoDimCoords(): Unit
+ updateTwoDimColPoints(): Unit
+ resetWorld(): Unit
Object:UI
+ eteen: Button
...
+ kaannyOikea: Button
+ width: Double
+ height: Double
+ world: World
+ c: Camera
+ canvas: Panel
+ getSkyAndFloorCornerPoints(): Tuple4[Array[Int], ...]
+ fromWorldToCanvasConvert(coords: (Double, Double)): (Int, Int)
+ top: MainFrame
Object: Transforms
+ smallRotLeft: SimpleMatrix
+ smallRotRight: SimpleMatrix
+ rotateZ(angle: Double): SimpleMatrix
+ rotateY(angle: Double): SimpleMatrix
+ translateX(distance: Double): SimpleMatrix
+ translateY(distance: Double): SimpleMatrix
+ translateZ(distance: Double): SimpleMatrix
+ translateXHorizontal(disntance: Double, angle: Double): SimpleMatrix
+ translateZVertical(distance: Double, angle: Double): SimpleMatrix
+ rotateZHorizontal(angle: Double, cAngle: Double): SimpleMatrix