

Luku 1

Pallogeometriaa

1.1 Kuun mitat ja pinnanmuodot

Kuun keskimääräinen säde on 1737.1 kilometriä. Suurimmat pinnanmuodot ovat varhaisia törmäyskraattereita. Kappaleen törmätessä kuun pintaan sulla basalttinen laava täytti kraatterin muodostaen tumman tasaisen alangon, joita nykyisin nimitämme kuun meriksi (*maria*), järviksi (*lacus*), lahdiiksi (*sinus*) ja soiksi (*paludes*).

Oletamme, että käytössämme on kuvan 1 mukainen listaus kuun merkittävimmistä pinnanmuodoista tekstitiedostona. Kukin rivi koostuu tabulaattorimerkein erotetuista kentistä. Kentät ovat muodostuman latinankielinen nimi, suomenkielinen nimi, latitudi, longitudi ja halkaisija.

1.2 Funktio `splitOn`

Kun annamme muuttujan `str` arvoksi esimerkkirivin tiedostosta, voimme jakaa merkkijonon osiin kirjaston `Data.List.Split` funktiolla `splitOn`. Funktio saa argumentteinaan katkaisevan ja katkaistavan merkkijonon. Funktio palauttaa listan syntyneistä merkkijonon osista.

```
> import Data.List.Split
```

Oceanus Procellarum	Myrskyjen valtameri	18.4 N	57.4 W	2568
Mare Frigoris	Kylmyyden meri	56.0 N	1.4 E	1596
Mare Imbrium	Sateiden meri	32.8 N	15.6 W	1123
Mare Fecunditatis	Hedelmällisyyden meri	7.8 S	51.3 E	909
Mare Tranquillitatis	Rauhallisuuden meri	8.5 N	31.4 E	873
Mare Nubium	Pilvien meri	21.3 S	16.6 W	715
Mare Serenitatis	Hiljaisuuden meri	28.0 N	17.5 E	707
Mare Australe	Eteläinen meri	38.9 S	93.0 E	603
Mare Insularum	Saarten meri	7.5 N	30.9 W	513
Mare Marginis	Reunameri	13.3 N	86.1 E	420
Mare Crisium	Vaarojen meri	17.0 N	59.1 E	418
Mare Humorum	Kosteuden meri	24.4 S	38.6 W	389
Mare Cognitum	Tunnettu meri	10.0 S	23.1 W	376
Mare Smythii	Smythin meri	1.3 N	87.5 E	373
Mare Nectaris	Nektarinmeri	15.2 S	35.5 E	333
Mare Orientale	Itäinen meri	19.4 S	92.8 W	327
Mare Ingenii	Nerokkuuden meri	33.7 S	163.5 E	318
Mare Moscoviense	Moskovan meri	27.3 N	147.9 E	277
Mare Humboldtianum	Humboldtin meri	56.8 N	81.5 E	273
Mare Vaporum	Höyryjen meri	13.3 N	3.6 E	245
Mare Undarum	Aaltojen meri	6.8 N	68.4 E	243
Mare Anguis	Käärmeitten meri	22.6 N	67.7 E	150
Mare Spumans	Vaahdon meri	1.1 N	65.1 E	139
Lacus Veris	Kevään järvi	16.5 S	86.1 W	396
Lacus Somniorum	Unelmien järvi	38.0 N	29.2 E	384
Lacus Excellentiae	Erinomaisuuden järvi	35.4 S	44.0 W	184
Lacus Autumnii	Syksyn järvi	9.9 S	83.9 W	183
Lacus Mortis	Kuoleman järvi	45.0 N	27.2 E	151
Lacus Solitudinis	Yksinäisyyden järvi	27.8 S	104.3 E	139
Lacus Temporis	Ajan järvi	45.9 N	58.4 E	117
Lacus Timoris	Pelon järvi	38.8 S	27.3 W	117
Lacus Gaudii	Ilon järvi	16.2 N	12.6 E	113
Lacus Doloris	Kärsimyksen järvi	17.1 N	9.0 E	110
Lacus Bonitatis	Hyvyyden järvi	23.2 N	43.7 E	92
Lacus Aestatis	Kesän järvi	15.0 S	69.0 W	90
Lacus Felicitatis	Onnellisuuden järvi	19.0 N	5.0 E	90
Lacus Lenitatis	Pehmeiden järvi	14.0 N	12.0 E	80
Lacus Spei	Toivon järvi	43.0 N	65.0 E	80
Lacus Odii	Vihan järvi	19.0 N	7.0 E	70
Lacus Perseverantiae	Sinnikkyiden järvi	8.0 N	62.0 E	70
Lacus Hiemalis	Talven järvi	15.0 N	14.0 E	50
Lacus Luxuriae	Ylellisyyden järvi	19.0 N	176.0 E	50
Lacus Oblivionis	Unohduksen järvi	21.0 S	168.0 W	50
Sinus Medii	Keskilahti	2.4 N	1.7 E	335
Sinus Aestuum	Helteen lahti	10.9 N	8.8 W	290
Palus Epidemiarum	Tautien suo	32.0 S	28.2 W	286
Sinus Iridum	Sateenkaarten lahti	44.1 N	31.5 W	236
Sinus Asperitatis	Kovuuden lahti	3.8 S	27.4 E	206
Sinus Roris	Aamukasteen lahti	54.0 N	56.6 W	202
Palus Putredinis	Mätänemisen suo	26.5 N	0.4 E	161
Palus Somni	Unien suo	14.1 N	45.0 E	143
Sinus Concordiae	Sopusoinnun lahti	10.8 N	43.2 E	142
Sinus Successus	Menestyksen lahti	0.9 N	59.0 E	132
Sinus Amoris	Rakkauden lahti	18.1 N	39.1 E	130
Sinus Lunicus	Lunan lahti	31.8 N	1.4 W	126
Sinus Honoris	Kunnian lahti	11.7 N	18.1 E	109
Sinus Fidei	Luottamuksen lahti	18.0 N	2.0 E	70

Kuva 1. Kuun merkittävimmät pinnanmuodot tekstitiedostona. Kentät on erotettu tabulaattorimerkein.

```
> str = "Mare Smythii\tSmythin meri\t1.3 N\t87.5 E\t373"
> splitOn "\t" str
["Mare Smythii","Smythin meri","1.3 N","87.5 E","373"]
```

1.3 Pallokoordinaatisto

Voimme muuntaa koordinaatteja pallokoordinaatistosta karteesiseen koordinaatistoon kaavalla (<http://mathworld.wolfram.com/SphericalCoordinates.html>)

$$\begin{aligned}x &= r \cos \theta \sin \phi \\y &= r \sin \theta \sin \phi \\z &= r \cos \phi\end{aligned}$$

Tässä r on säde eli etäisyys origosta, θ kulma x -akselista xy -tasossa ja ϕ kulma ylöspäin osoittavasta z -akselista.

Maantieteellisessä koordinaatistossa merkitsemme leveysastetta (*latitudi*) symbolilla δ , jolloin $\phi = 90^\circ - \delta$ sekä pituusastetta (*longitudi*) symbolilla λ ($\lambda = \theta$).

Koordinaattilyhenteissä kirjain N (*north*) merkitsee pohjoista leveyttä, S (*south*) eteläistä leveyttä, E (*east*) itäistä pituutta ja W (*west*) läntistä pituutta. Maapallolla leveysaste δ kasvaa päiväntasaajalta pohjoiseen kuljettaessa ja pituusaste λ Greenwichin nollameridiaanilta itään kuljettaessa. Nimitykset leveys ja pituus juontuvat Välimeren alueen kulttuureista: Välimeri on ”pitkä” itä-länsi-suunnassa ja ”leveä” pohjois-etelä-suunnassa. Pituuspiirejä sanotaan myös meridiaaneiksi. Termi meridiaani johtuu latinan puolipäivää tai etelää merkitsevästä sanasta *meridies*.

Määrittelemme tietotyypin `Point3D` pisteelle kolmiulotteisessa karteesisessa xyz -koordinaatistossa. Pallokoordinaatistossa määrittelemme pisteen `Spheric3D` kulmien θ ja ϕ avulla.

```
-- | Point3D x y z, RH cartesian coordinates
data Point3D = Point3D Double Double Double
```

```
-- | Spheric3D theta phi, where phi =
```

```
-- polar angle measured from a fixed zenith direction
data Spheric3D = Spheric3D Angle Angle
```

Asetamme kuun säteeksi $r = 1737.1$ km. Yksinkertaisimman muunnoksen kolmiulotteisesta koordinaatistosta kaksiulotteiseen koordinaatistoon saamme pudottamalla x -koordinaatin pois.

```
r = 1737.1
```

```
dropX (Point3D x y z) = Point y z
```

```
perspective = dropX
```

```
cartesian (Spheric3D lambda delta) = Point3D x y z
```

```
  where
```

```
    x = r * cos1 theta * sin1 phi
```

```
    y = r * sin1 theta * sin1 phi
```

```
    z = r * cos1 phi
```

```
    theta = lambda
```

```
    phi = (DEG 90) `subAngles` delta
```

Saamme hahmotelman leveyspiireistä pallon etupuoliskolla algoritmilla

```
latitudes = [PolyLine [(perspective . cartesian)
```

```
  (Spheric3D (DEG 1) (DEG d))
```

```
  | l <- lambda]
```

```
  | d <- delta]
```

```
  where
```

```
    delta = [-90,-75..90]
```

```
    lambda = [-90,-70..90]
```

Etupuoliskon pituuspiirit eli meridiaanit saamme algoritmilla

```
meridians = [PolyLine [(perspective . cartesian)
```

```
  (Spheric3D (DEG 1) (DEG d))
```

```
  | d <- delta]
```

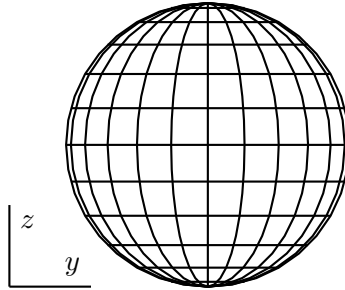
```
  | l <- lambda]
```

```
  where
```

```
    delta = [-90,-80..90]
```

```
    lambda = [-90,-75..90]
```

Olemme esittäneet leveys- ja pituuspiirien muodostaman kuvion kuvassa 2.



Kuva 2. Karttapallon puolisko, jossa kuvattuna leveyspiirit ja pituuspiirit eli meridiaanit 15 asteen välein.

1.4 Vinoprojektion perspektiivimatriisi

Niin sanotussa *vinoprojektiossa* kuvaamme kaksi akselia suoraan kulmaan toistensa kanssa ja kolmannen akselin tiettyyn kulmaan näiden välillä.

Kuvassa 3 esiintyvät vakiot a , b , c ja d olemme määritelleet seuraavasti kulman α avulla:

$$a = 1/2 \cdot \cos \alpha$$

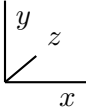
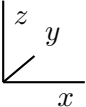
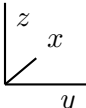
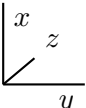
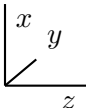
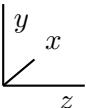
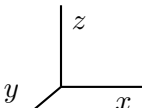
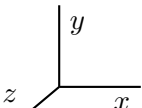
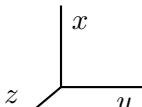
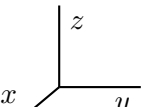
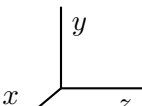
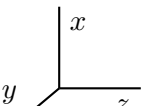
$$b = 1/2 \cdot \sin \alpha$$

$$c = -a$$

$$d = -b$$

Matriiseista järjestysluvultaan parittomat antavat kuvauskoordinaatistoksi vasenkätisen ja parilliset oikeakätisen koordinaatiston. Haskell-kielelle muunnettuna voimme esittää perspektiivimatriisit $M_{1..12}$ `case`-lauseen avulla.

```
matrix1 m alpha = case m of
  1 -> [ [1,0,a], [0,1,b], z]
  2 -> [ [1,a,0], [0,b,1], z]
  3 -> [ [a,1,0], [b,0,1], z]
  4 -> [ [0,1,a], [1,0,b], z]
```

		$M_1 = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 0 \end{pmatrix}$	$M_2 = \begin{pmatrix} 1 & a & 0 \\ 0 & b & 1 \\ 0 & 0 & 0 \end{pmatrix}$
		$M_3 = \begin{pmatrix} a & 1 & 0 \\ b & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$	$M_4 = \begin{pmatrix} 0 & 1 & a \\ 1 & 0 & b \\ 0 & 0 & 0 \end{pmatrix}$
		$M_5 = \begin{pmatrix} 0 & a & 1 \\ 1 & b & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$M_6 = \begin{pmatrix} a & 0 & 1 \\ b & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
		$M_7 = \begin{pmatrix} 1 & c & 0 \\ 0 & d & 1 \\ 0 & 0 & 0 \end{pmatrix}$	$M_8 = \begin{pmatrix} 1 & 0 & c \\ 0 & 1 & d \\ 0 & 0 & 0 \end{pmatrix}$
		$M_9 = \begin{pmatrix} 0 & 1 & c \\ 1 & 0 & d \\ 0 & 0 & 0 \end{pmatrix}$	$M_{10} = \begin{pmatrix} c & 1 & 0 \\ d & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$
		$M_{11} = \begin{pmatrix} c & 0 & 1 \\ d & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$M_{12} = \begin{pmatrix} 0 & c & 1 \\ 1 & d & 0 \\ 0 & 0 & 0 \end{pmatrix}$

Kuva 3. Vinoprojektoiden perspektiivimatriiseja.

```

5 -> [ [0,a,1], [1,b,0], z]
6 -> [ [a,0,1], [b,1,0], z]
7 -> [ [1,c,0], [0,d,1], z]
8 -> [ [1,0,c], [0,1,d], z]
9 -> [ [0,1,c], [1,0,d], z]
10 -> [ [c,1,0], [d,0,1], z]
11 -> [ [c,0,1], [d,1,0], z]
12 -> [ [0,c,1], [1,d,0], z]

```

where

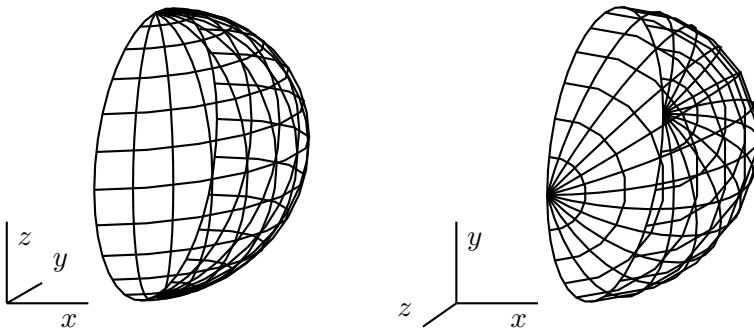
```

a = 0.5 * cos1 alpha
b = 0.5 * sin1 alpha
c = -a; d = -b
z = [0,0,0]

```

Asetamme nyt muunnosmatriiseiksi matriisit M_2 ja M_8 (kuva 4).

$$M_2 = \begin{pmatrix} 1 & 1/2 \cdot \cos 30^\circ & 0 \\ 0 & 1/2 \cdot \sin 30^\circ & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad M_8 = \begin{pmatrix} 1 & 0 & -1/2 \cdot \cos 35^\circ \\ 0 & 1 & -1/2 \cdot \sin 35^\circ \\ 0 & 0 & 0 \end{pmatrix}$$



Kuva 4. Karttapallon puoliskot muunnosmatriiseja M_2 ja M_8 käyttäen.

```

matrixTimes3 a b =
  [sum [x * y | (x,y) <- zip a1 b] | a1 <- a]

matr1 pv pAlpha (Point3D x1 y1 z1) = Point x y
where

```

```
[x,y,z] = matrixTimes3 (matrix1 pv alpha) [x1,y1,z1]
alpha = DEG pAlpha
```

```
r = 1737.1
```

```
dropX (Point3D x y z) = Point y z
```

```
perspective = matr1 pv pAlpha
```

```
where
```

```
pv = 8
```

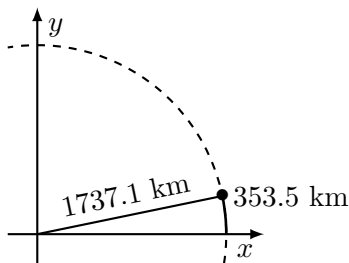
```
pAlpha = 35
```

1.5 Mare Serenitatis

Haluamme seuraavaksi kuvata Hiljaisuuden meren karttapallolle. Kraatterin läpimitta on $d = 707$ km, ja säde näin ollen $r = d/2 = 353.5$ km. Hiljaisuuden meren keskipisteen koordinaatit ovat (28.0° N, 17.5° E).

Kuun säteen ollessa $r = 1737.1$ km, saamme kuvan 5 merkinnöillä keskuskulmaksi

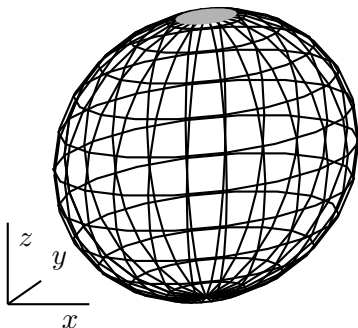
$$\theta = 2\pi \cdot \frac{353.5}{2\pi \cdot 1737.1} = 0.2035 \text{ rad} = 11.65^\circ$$



Kuva 5. Kraatterin säteen $r = 353.5$ km muodostama keskuskulma.

Aiemmin esitellyn perusteella osaamme jo sijoittaa pohjoisnavalle ympyrän, jonka säde on annettu. Käytämme tällöin Hiljaisuuden merelle laske-

maamme keskuskulmaa ylimääräisenä leveyspiirinä, jonka piirrämme täytettynä monikulmiona (kuva 6).



Kuva 6. Hiljaisuuden meri pohjoisnavalle sijoitettuna.

```
serenitatis = [ Filled $ Polygon [
  (perspective . cartesian) (Spheric3D (DEG 1) t)
  | 1 <- lambda]]
where
  t = RAD (halfpi - 0.2035)
  lambda = [-180,-160..160]
```

1.6 Kiertomatriisit kolmessa ulottuvuudessa

Kiertomatriisit kolmessa ulottuvuudessa kulman θ verran akselien x , y ja z suhteen ovat (https://en.wikipedia.org/wiki/Rotation_matrix)

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix} \quad R_y = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Haskell-kielelle muunnettuna nämä ovat

```
rotationX t = [
  [1,      0,      0],
  [0, cos1 t, -sin1 t],
  [0, sin1 t,  cos1 t]
]
```

```
rotationY t = [
  [cos1 t, 0, sin1 t],
  [0,      1,      0],
  [-sin1 t, 0, cos1 t]
]
```

```
rotationZ t = [
  [cos1 t, -sin1 t, 0],
  [sin1 t,  cos1 t, 0],
  [      0,      0, 1]
]
```

Latitudin δ komplementtikulma $\phi = 90^\circ - \delta$ määrittää kierron y -akselin suhteen ja longitudi λ kierron z -akselin suhteen.

```
rotYZ delta lambda (Point3D x1 y1 z1) = Point3D x y z
  where
    [x,y,z] = foldr matrixTimes3 [x1,y1,z1] rts
    rts = [rotationZ lambda, rotationY phi]
    phi = DEG 90 `subAngles` delta
```

Valitsemme perspektiivimatriisiin M_{10} .

```
perspective = matr1 pv pAlpha
  where
    pv = 10
    pAlpha = 35
```

Yleisessä muodossaan määrittelemme kuun meren piirtoalgoritmin funktiossa `mare`, joka saa parametrinaan `d` meren halkaisijan ja parametrinaan `pos` maantieteellisen pohjois-itä-koordinaatin tyyppiä `GeographicNE`.

```
data GeographicNE = GeographicNE Angle Angle
```

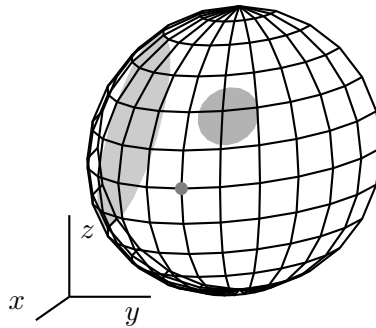
```
marePg1 d pos = Filled $ Polygon $ marePts d pos

marePts d pos = [ perspective $ rotYZ delta lambda $
  cartesian $ Spheric3D (DEG l) phi | l <- lambdaRim ]
where
  GeographicNE delta lambda = pos
  phi = DEG 90 `subAngles` (RAD theta)
  theta = (d/2) / r
  lambdaRim = [-180,-160..160]
```

Hiljaisuuden meri saa nyt muodon

```
serenitatis = marePg1 d pos
where
  d = 707
  pos = GeographicNE (DEG 28) (DEG 17.5)
```

Piirrämme kuvaan 7 myös Myrskyjen valtameren, jonka halkaisija on $d = 2568$ km, ja jonka keskipiste sijaitsee pisteessä $(18.4^\circ \text{ N}, 57.4^\circ \text{ W})$.



Kuva 7. Hiljaisuuden meri ja Myrskyjen valtameri.

```
procellarum = marePg1 d pos
where
  d = 2568
  pos = GeographicNE (DEG 18.4) (DEG (-57.4))
```

Merkitsemme myös koordinaatiston nollapisteen funktiolla `proto0`.

```
proto0 = marePg1 160 (GeographicNE (DEG 0) (DEG 0))
```

1.7 Monikulmion paloittelu

Olemme koordinaattimuunnoksissa huomioineet ainoastaan täytetyn monikulmion reunapisteen, joten esimerkiksi Myrskyjen valtameren keskiosat piirtyivät väärin kuvassa 7.

Parempaan tulokseen päädyimme paloittelemalla monikulmiot asteverkon mukaisesti. Käytämme aluksi tasavälistä lieriöprojektiota (*equirectangular projection*), jossa pituus- ja leveysasteet kuvautuvat sellaisenaan koordinaattipisteiksi.

```
equirect (Spheric3D lambda delta) = Point l d
  where
    DEG l = degrees lambda
    DEG d = degrees delta
```

Mittakaavakertoimenä on seuraavassa $\frac{2\pi \cdot r}{360}$, missä $r = 1737.1$ km on kuun säde.

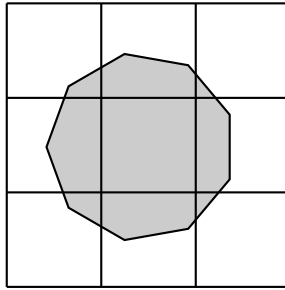
```
marePg2 d pos = Polygon (marePts2 d pos)
```

```
marePts2 d pos = [ pt0 `addCoords`
  pointFromPolar (DEG l) r2 | l <- lambdaRim ]
  where
    r2 = (d/2) / (twopi * r / 360)
    pt0 = equirect (Spheric3D lambda delta)
    GeographicNE delta lambda = pos
    lambdaRim = [-180,-140..140]
```

Muunnamme polaarikoordinaatit pisteeksi `Point` funktiolla `pointFromPolar`.

```
pointFromPolar t s = Point x y
  where
    x = s * cos1 t
    y = s * sin1 t
```

Olemme kuvassa 8 esittäneet suorakulmaisessa koordinaatistossa Hedelmälisyyden meren, jonka halkaisija on $d = 909$ km, ja jonka keskipiste sijaitsee pisteessä (7.8° S, 51.3° E).



Kuva 8. Hedelmällisyyden meri suorakulmaisessa koordinaatistossa.

```
fecunditatis = marePts2 d pos
  where
    d = 909
    pos = GeographicNE (DEG (-7.8)) (DEG 51.3)
```

1.8 Pisteet monikulmion sisä- ja ulkopuolella

Käytämme monikulmion paloitteluun *Sutherland-Hodgmanin algoritmia* (https://en.wikipedia.org/wiki/Sutherland-Hodgman_algorithm). Paloittelussa muokkaamme monikulmion kärkipistejoukkoa vertaamalla sitä leikkaavan monikulmion sivuihin sivu kerrallaan. Leikkauksen lähtöjoukkona toimii aina edellisessä vaiheessa saatu kärkipistejoukko. Kukin sivu leikkaa osan kärkipisteistä pois sekä muodostaa uusia kärkipisteitä paloitteltavan ja leikkaavan monikulmion sivujen leikkauspisteisiin. Paloittelualgoritmia varten tarvitsemme tiedon siitä, kummalla puolella annettua sivua tietty kärkipiste sijaitsee.

Saamme selville kummalla puolella sivua piste sijaitsee muodostamalla kolmion, jonka kärkipisteet ovat sivun alkupiste, sivun loppupiste ja vertailtava piste. Kun piste sijaitsee sivun oikealla puolella, muodostuneen kolmion kiertosuunta on myötäpäivään, jolloin sen ala determinanttisäännön mukaan on negatiivinen. Kun piste sijaitsee sivun vasemmalla puolella, kiertosuunta on vastapäivään ja muodostuneen kolmion ala positiivinen.

```
data InOut = In | Out
```

deriving Show

```
sign x = if x < 0 then (-1) else 1
around xs = zip xs ((tail . cycle) xs)
```

```
inOut1 p1 p2 pts = [
  (inOut . sign . area . Polygon) [p1,p2,p3] | p3 <- pts]
where
  inOut 1 = In
  inOut (-1) = Out
```

```
gridGreatCircles = concat [[[
  xpt l1 d1, xpt l2 d1, xpt l2 d2, xpt l1 d2]
 | (d1,d2) <- zip vb3 (tail vb3)]
 | (l1,l2) <- zip vb2 (tail vb2)]
where
  xpt l d = equirect (Spheric3D (DEG l) (DEG d))
  vb3 = visible3 delta
  vb2 = visible2 lambda
  delta = [-90,-75..90]
  lambda = [-90,-75..90]
```

```
visible2 = filter (\l -> l > 29 && l < 76)
visible3 = filter (\d -> d > -31 && d < 16)
```

Determinantisääntöä käyttämme, kun määrittelemme funktion `area` monikulmiolle `Polygon`. Determinantin saamme matematiikasta tutulla kaavalla

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = a \cdot d - b \cdot c$$

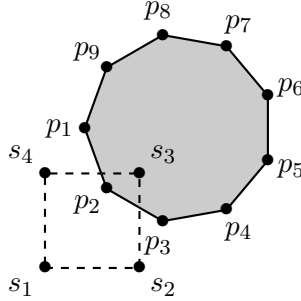
```
-- | http://mathworld.wolfram.com/PolygonArea.html
-- Polygon area: vertices counterclockwise
area (Polygon pts) = half (sum [det [[x1,y1],[x2,y2]]
 | (Point x1 y1,Point x2 y2)
   <- zip pts (tail pts ++ take 1 pts)])
```

```
half = (0.5 *)
```

```
det [[a,b],[c,d]] = a * d - b * c
```

Aloitamme kuvion paloittelun neliöstä $(s_1s_2s_3s_4)$ alueen vasemmassa alannurkassa (kuva 9).

```
parte = 0
[s1,s2,s3,s4] = gridGreatCircles !! parte
fc0 = fecunditatis
```



Kuva 9. Ensimmäinen tarkasteltava ruutu.

Rajattavan monikulmion sivut jakautuvat neljään ryhmään suhteessa rajaavaan monikulmioon:

- (In,In): sivu alkaa sisäpuolelta ja päättyy sisäpuolelle.
- (In,Out): sivu alkaa sisäpuolelta ja päättyy ulkopuolelle.
- (Out,Out): sivu alkaa ulkopuolelta ja päättyy ulkopuolelle.
- (Out,In): sivu alkaa ulkopuolelta ja päättyy sisäpuolelle.

Sutherland-Hodgmanin algoritmin mukaiset toimenpiteet sivutyypeille ovat

- (In,In): säilytämme kärkipisteet.
- (In,Out): säilytämme lähtöpisteen ja siirrämme loppupisteen.
- (Out,Out): poistamme kärkipisteet.
- (Out,In): siirrämme alkupisteen ja säilytämme loppupisteen.

Haskell-kielelle muunnettuna saamme uudet kärkipisteet monikulmion pistejoukosta `fc` suoran (s_1,s_2) suhteen funktiokutsulla `nextGen fc s1 s2`.

```
nextGen fc s1 s2 = concat [new i1 i2 p1 p2
  | ((i1,i2),(p1,p2)) <- zip io2 pts]
```

```

where
  io1 = inOut1 s1 s2 fc
  io2 = around io1
  pts = around fc
  new In In  p1 p2 = [p1]
  new In Out p1 p2 = [p1,
    fromJust (intersection s1 s2 p1 p2)]
  new Out Out p1 p2 = []
  new Out In  p1 p2 = [
    fromJust (intersection s1 s2 p1 p2)]

```

Ensimmäinen rajaava suora on neliön alareuna s_1s_2 . Monikulmion pistejoukon $fc0$ kaikki kärkipisteet kuuluvat alueen sisäpuolelle.

```

> io1 = inOut1 s1 s2 fc0
> io1
[In,In,In,In,In,In,In,In,In,In]

```

Pistejoukon kaikki pisteet kuuluvat luokkaan (In, In) , joten alareuna säilyttää kaikki monikulmion pisteet $(p_1 \cdots p_9)$.

```

> around io1
[ (In,In), (In,In), (In,In), (In,In), (In,In),
  (In,In), (In,In), (In,In), (In,In) ]

```

Saamme uuden pistejoukon $fc1$ funktiokutsulla `nextGen fc0 s1 s2`.

```
fc1 = nextGen fc0 s1 s2
```

Toinen rajaava suora on neliön oikea reuna s_2s_3 . Nyt rajaavan suoran vasemmalle puolelle eli alueen sisäpuolelle jäävät monikulmion pisteet $(p_1 p_2 p_9)$. Alueen ulkopuolelle jäävät monikulmion pisteet $(p_3 \cdots p_8)$.

```

> io2 = inOut1 s2 s3 fc1
> io2
[In,In,Out,Out,Out,Out,Out,Out,In]

```

Monikulmion sivut suhteessa rajaavan neliön oikeaan reunaan s_2s_3 kuuluvat nyt seuraaviin luokkiin:

```

> around io2
[ (In,In), (In,Out), (Out,Out), (Out,Out), (Out,Out),

```

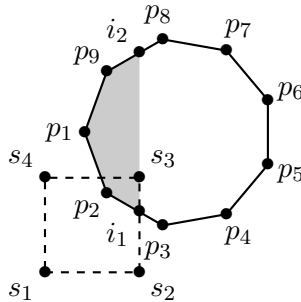

$(\text{Out}, \text{Out}), (\text{Out}, \text{Out}), (\text{Out}, \text{In}), (\text{In}, \text{In})$]

Luokat (In, Out) ja (Out, In) tuottavat uuden kärkipisteen i_1 suorien $s_2 s_3$ ja $p_2 p_3$ leikkauspisteeseen sekä pisteen i_4 suorien $s_2 s_3$ ja $p_8 p_9$ leikkauspisteeseen.

`i1 = intersection s2 s3 p2 p3`

`i2 = intersection s2 s3 p8 p9`

Kuvan 10 merkinnöillä suoran $s_2 s_3$ suhteen leikattu toinen monikulmio koostuu kärkipisteistä $(p_1 p_2 i_1 i_2 p_9)$.



Kuva 10. Toinen leikkaus antaa monikulmion kärkipisteet $(p_1 p_2 i_1 i_2 p_9)$.

Saamme nyt uuden pistejoukon `fc2` funktiokutsulla `nextGen fc1 s2 s3`.

`fc2 = nextGen fc1 s2 s3`

Kolmas leikkaus tapahtuu suoran $s_3 s_4$ suhteen edellä saadulle kärkipistejoukolle $(i_1 p_2 p_3 p_4 i_2)$.

`fc3 = nextGen fc2 s3 s4`

Saamme pisteväleille uudet luokat

```
> io3 = inOut1 s3 s4 fc2
```

```
> io3
```

```
[Out, In, In, Out, Out]
```

```
> around io3
```

```
[(Out, In), (In, In), (In, Out), (Out, Out), (Out, Out)]
```

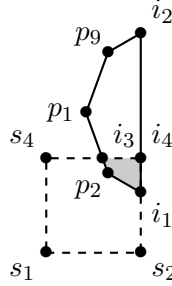
Tässä luokat (Out, In) ja (In, Out) tuottavat uuden kärkipisteen i_3 suorien

s_3s_4 ja p_1p_2 leikkauspisteeseen sekä pisteen i_4 suorien s_3s_4 ja i_1i_2 leikkauspisteeseen.

`i3 = intersection s3 s4 p1 p2`

`i4 = intersection s3 s4 i1 i2`

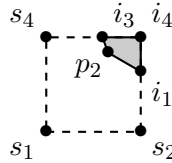
Leikattu monikulmio koostuu nyt kärkipisteistä $(i_3p_2i_1i_4)$ (kuva 11).



Kuva 11. Kolmas leikkaus antaa monikulmion kärkipisteet $(i_3p_2i_1i_4)$.

Viimeinen leikkaus tapahtuu suoran s_4s_1 suhteen. Leikkaus säilyttää kärkipistejoukon $(i_3p_2i_1i_4)$ sellaisenaan (kuva 12).

`fc3 = nextGen fc2 s3 s4`



Kuva 12. Ensimmäisen alueen valmis kärkipistejoukko $(i_3p_2i_1i_4)$.

Kun kokoamme yhteen funktiokutsut

`fc0 = fecunditatis`

`fc1 = nextGen fc0 s1 s2`

`fc2 = nextGen fc1 s2 s3`

`fc3 = nextGen fc2 s3 s4`

`fc4 = nextGen fc3 s4 s1`

saamme seuraavan rekursiivisen määrittelyn funktiolle `fc`:

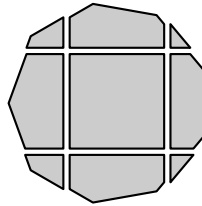
```
fc 0 = fecunditatis
fc n = nextGen1 (fc (n - 1)) ((around square1) !! (n - 1))
  where
    nextGen1 f (a,b) = nextGen f a b
```

```
block1 = fc 4
```

Neljästä suunnasta leikattu valmis pistejoukko on nyt muuttujassa `block1`.

Voimme nyt esittää Hedelmällisyyden meren kokonaisuudessaan paloitetuna (kuva 13).

```
blocksA = c
  where
    c = [map (`addCoords` Point x y) b1
         | (b1,(x,y)) <- b]
    b = zip blocks [(x,y) | x <- [-1..1], y <- [-1..1]]
```



Kuva 13. Hedelmällisyyden meri paloitetuna.

Paloittelimme monikulmiot asteviivojen mukaan, joten joudumme palaa-
maan karteesisesta koordinaatistosta takaisin maantieteelliseen koordi-
naatistoon. Määrittelemme tätä varten funktion `geog`. Maantieteellisen
koordinaatin saamme kahdella erillisellä suuntakulman laskennalla, joista
ensimmäinen on kulma λ xy -tasossa ja toinen kulma δ edellisen ja z -akselin
muodostamassa tasossa.

```
geog (Point3D x y z) = GeographicNE delta lambda
  where
    delta = directionAngle vect2
    vect2 = Vector rProjXy z
```

```

rProjXy = sqrt (sqr x + sqr y)
lambda = theta
theta = directionAngle vect1
vect1 = Vector x y
r = sqrt (sqr x + sqr y + sqr z)
sqr x = x * x

```

Olemme luetteloineet merien pintabasaltin iät ja piirrämme ne harmaan eri sävyinä. Monikulmion muunnoskaavat olemme keränneet funktioon `marePg`.

```

marePg d pos = pg
  where
    pg = map Polygon blocks2
    blocks2 = [map (perspective . cartesian .
      ptToSpheric3D) pts | pts <- blocks1]
    blocks1 = cutEqui pts3
    pts3 = map (equirect . geog) pts2
    pts2 = [ (rotYZ delta lambda . cartesian)
      (Spheric3D (DEG th) phi)
      | th <- lambdaRim]
    GeographicNE delta lambda = pos
    phi = RAD ((d/2) / r)
    lambdaRim = [-180,-160..160]

```

```

mare2 t = map (FilledWith rgb) (marePg d pos)
  where
    rgb = RGB v v v
    v = 1.0 - (0.05 + 0.8 * ((g - 3100) / 1000))
    (name,n,e,d,g) = t
    pos = GeographicNE (DEG n) (DEG e)

```

```

maria ts = ts2
  where
    ts2 = concatMap mare2 (filter visible ts)
    visible (name,n,e,d,g) = e >= -90 && e <= 90

```

Teemme funktiosta `cartesian` monimuotoisen, jolloin se muuntaa sekä tyyppin `Spheric3D` että tyyppin `GeographicNE` muuttujan karteesisen koordi-

naatistoon. Muunnokset tasaväliseen lieriöprojektitioon säilyvät sellaisenaan.

```
data SphericP = Spheric3D Angle Angle
  | GeographicNE Angle Angle
```

```
ptToSpheric3D (Point x y) = Spheric3D theta phi
  where
    theta = lambda
    phi = (DEG 90) `subAngles` delta
    delta = DEG y
    lambda = DEG x
```

```
cartesian (GeographicNE delta lambda) =
  cartesian (Spheric3D theta phi)
  where
    theta = lambda
    phi = (DEG 90) `subAngles` delta
```

```
cartesian (Spheric3D theta phi) = Point3D x y z
  where
    x = r * cos1 theta * sin1 phi
    y = r * sin1 theta * sin1 phi
    z = r * cos1 phi
```

Myös varsinainen paloittelualgoritmi on hyvin samankaltainen aiemman kanssa, mutta olemme parametrisoineet siinä monikulmiot.

```
fc mre sq 0 = mre
fc mre sq n = nextGen1 (fc mre sq nm) ((around sq) !! nm)
  where
    nm = n - 1
    nextGen1 f (a,b) = nextGen f a b
```

```
blocksEqui mre = b2
  where
    b2 = filter (not . null) b1
    b1 = map (blockE mre) squares
    blockE mre sq = fc mre sq 4
```

```
squares = gridGreatCircles

cutEqui mre = blocksEqui mre
```

1.9 Paikannimet

Luemme paikannimet ja pintabasaltin iät kahdesta eri tiedostosta pääohjelmassa.

```
main = do
  content <- readFile "../moon-random/moon-list.txt"
  content2 <- readFile "../moon-random/surface-basalt-age.txt"
  let
    moon = filter (not . null) (lines content)
    ageText = filter (not . null) (lines content2)
    c1 = map tabulated moon
    aged1 = map aged ageText
    c2 = map (lookup1 aged1) c1
    c3 = filter valid1 c2
    c4 = map aged2 c3
  putStrLn (tpict c4)
```

Kentät on tiedostoissa erotettu tabulaattorimerkein ("`\t`"), joten pilkomme tekstin niiden mukaan. Luemme numerot standardikirjaston funktiolla `read`. Funktio `read` on monimuotoinen funktio, ja vaatii siksi kohdetyypin tyyppimäärittelyn.

```
tabulated str = map trim (splitOn "\t" str)

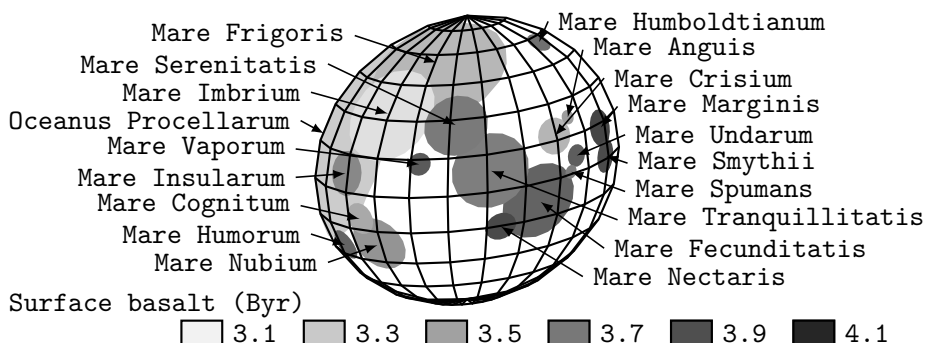
aged2 xs = (a,b1,c1,d1,e1)
  where
    [a,b,c,d,e] = xs
    [b1,c1,d1,e1] = map readd [b,c,d,e]
    readd x = read x :: Double
```

Voimme käyttää listoja kuin ne olisivat tietokannan tauluja, mutta helpommin. Funktio `lookup` palauttaa arvon `Just x`, jos haku onnistui, muutoin se

palauttaa arvon Nothing.

```
lookup1 table2 table1 = [a,c1,d1,e,f]
  where
    [c1,d1] = map brt [c,d]
    [a,b,c,d,e] = table1
    f = case lookup a table2 of
      Just x -> x
      Nothing -> ""
    brt s = addMinus s ++ takeWhile (/= ' ') s
    addMinus s = if last s `elem` "SW" then "-" else ""
```

Asettellemme vielä paikannimet kartan vasemmalle ja oikealle puolelle. Olemme esittäneet syntyneen kartan kuvassa 14.



Kuva 14. Valmis kartta selitteineen.

```
data LeftRight = L | R
  deriving Eq

getSide (Point x1 y1) (Point x2 y2)
  | x1 <= x2 = L
  | otherwise = R

name2 t = (name,side,posXY)
  where
    side = getSide posXY pt1
```

```

pt1 = Point (-70) (-70)
posXY = (perspective . cartesian)
      (GeographicNE (DEG n) (DEG e))
(name,n,e,d,g) = t

sortOnY = sortOn (\(n,s,Point x y) -> -y)

names ts = concat (lx ++ rx)
  where
    lx = map (\(n,s,p,pt) -> [Texttt pt n "left",
      Arrow "" pt p]) l3
    rx = map (\(n,s,p,pt) -> [Texttt pt n "right",
      Arrow "" pt p]) r3
    l3 = map f3 l2
    r3 = map f3 r2
    f3 = \ (y,(n,s,p)) -> (n,s,p,crc s y)
    l2 = zip [dy*sL-d,dy*(sL-1)-d..] l1
    r2 = zip [dy*sR-d,dy*(sR-1)-d..] r1
    d = 70
    sL = intToDouble (length l1) / 2
    sR = intToDouble (length r1) / 2
    dy = 2 * r / max ln rn
    [ln,rn] = map (intToDouble . length) [left,right]
    l1= refineOrder L l0
    r1= refineOrder R r0
    [l0,r0] = map sortOnY [left,right]
    right = filter (\(n,s,p) -> s == R) n1
    left  = filter (\(n,s,p) -> s == L) n1
    n1 = map name2 (filter visible ts)
    visible (name,n,e,d,g) = e > -90 && e < 90

crc s y = p4
  where
    p4 = if d1 < d2 then p1 else p2
    [d1,d2] = map (dist (p3 s)) [p1,p2]
    [p1,p2] = intersect1 circle1 pt1 pt2
    pt1 = Point (-r) y

```



```

p0 = Point (-70) (-70)
circle1 = Circle (r+300) p0
pt2 = Point r y
p3 L = pt1
p3 R = pt2

```

1.10 Coastlines

The header structure: GSHHG C-structure used to read and write the data.

```

struct GSHHG { /* Global Self-consistent Hierarchical
                High-resolution Shorelines */
int id; /* Unique polygon id number, starting at 0 */
int n; /* Number of points in this polygon */
int flag; /* = level + version << 8 + greenwich << 16
            + source << 24 + river << 25 */
/* flag contains 5 items, as follows:
 * low byte: level = flag & 255: Values: 1 land, 2 lake,
            3 island_in_lake, 4 pond_in_island_in_lake
 * 2nd byte: version = (flag >> 8) & 255: Values:
            Should be 12 for GSHHG release 12 (i.e., version 2.2)
 * 3rd byte: greenwich = (flag >> 16) & 1: Values:
            Greenwich is 1 if Greenwich is crossed
 * 4th byte: source = (flag >> 24) & 1: Values:
            0 = CIA WDBII, 1 = WVS
 * 4th byte: river = (flag >> 25) & 1: Values:
            0 = not set, 1 = river-lake and level = 2 */
int west, east, south, north; /* min/max extent in micro-degrees */
int area; /* Area of polygon in 1/10 km^2 */
int area_full; /* Area of original full-resolution polygon
                in 1/10 km^2 */
int container; /* Id of container polygon that encloses
                this polygon (-1 if none) */
int ancestor; /* Id of ancestor polygon in the full resolution set
                that was the source of this polygon (-1 if none) */
};

```

Following each header structure is n structures of coordinates:

```
struct GSHHG_POINT {  
    /* Each lon, lat pair is stored in micro-degrees in 4-byte  
       signed integer format */  
    int32_t x;  
    int32_t y;  
};
```

Some useful information:

- A) To avoid headaches the binary files were written to be big-endian. If you use the GMT supplement gshhg it will check for endian-ness and if needed will byte swab the data automatically. If not then you will need to deal with this yourself.
- B) In addition to GSHHS we also distribute the files with political boundaries and river lines. These derive from the WDBII data set.
- C) As to the best of our knowledge, the GSHHG data are geodetic longitude, latitude locations on the WGS-84 ellipsoid. This is certainly true of the WVS data (the coastlines). Lakes, riverlakes (and river lines and political borders) came from the WDBII data set which may have been on WGS072. The difference in ellipsoid is way less then the data uncertainties. Offsets have been noted between GSHHG and modern GPS positions.
- D) Originally, the gshhs_dp tool was used on the full resolution data to produce the lower resolution versions. However, the Douglas-Peucker algorithm often produce polygons with self-intersections as well as create segments that intersect other polygons. These problems have been corrected in the GSHHG lower resolutions over the years. If you use gshhs_dp to generate your own lower-resolution data set you should expect these problems.
- E) The shapefiles release was made by formatting the GSHHG data using the extended GMT/GIS metadata understood by OGR, then using ogr2ogr to build the shapefiles. Each resolution is stored in its own subdirectory (e.g., f, h, i, l, c) and each level (1-4) appears in its own shapefile. Thus, GSHHS_h_L3.shp contains islands in lakes for the

high res data. Because of GIS limitations some polygons that straddle the Dateline (including Antarctica) have been split into two parts (east and west).

- F) The netcdf-formatted coastlines distributed with GMT derives directly from GSHHG; however the polygons have been broken into segments within tiles. These files are not meant to be used by users other than via GMT tools (pscoast, grdlandmask, etc).