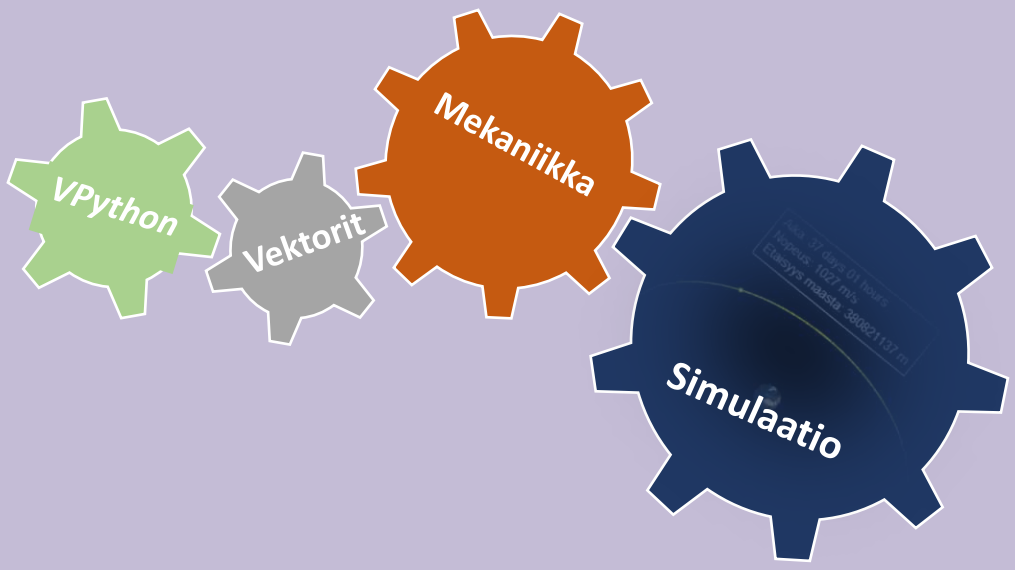


Simulaatioharjoituksia mekaniikasta

VPython 7

© Kari Peisa 2020

Sisällysluettelo

1. JOHDANTO 2

2. DETERMINISTINEN JA STOKASTINEN SIMULAATIO 3

2.1. Kinematiikan perusteet simulaatioissa 3

2.2. Ilmanvastukseton heittoliike (gravitaatiovoiman alainen) 5

3. DYNAMIIKAN PERUSLAUSE SIMULAATION RATKAISUNA 8

3.1. Kitkavoima 9

3.2. Ilmanvastuksellinen heittoliike 11

3.3. Vuorovaikutusta *VPython* -simulaatioon widget-elementtien avulla 12

4. YMPYRÄLIIKKEEN DETERMINISTISIÄ SIMULAATIOITA 16

4.1. Ympyräliike koordinaattiakseleiden muodostamassa tasossa 16

4.2. Kierretyn ratakäyrän vektorifunktio (*) 18

4.3. Objektien kiertäminen VPythonin funktiolla rotate() 20

5. LIIKEMÄÄRÄN MUUTOS JA IMPULSSI 23

6. KIERTOLIIKKEIDEN STOKASTISIA SIMULAATIOITA 24

6.1. Gravitaatiovoiman alainen kiertoliike ja liikemäärän säilymislaki 24

6.2. Maa-kuu systeemi 25

6.3. Satelliittien kiertoradat ja Runge-Kutta-Nyström menetelmä (**) 27

7. JÄYKÄN KAPPALEEN DYNAMIIKKA 29

7.1. Kulmasuureet ja ratasuureet pyörimisliikkeiden kinematiikassa ja dynamiikassa 29

7.2. 3D heilahdusliikkeen stokastinen simulointi 32

8. TÖRMÄYKSIEN KÄSITTELY 35

8.1. Pallon ja tason kohtaaminen 35

8.2. Impulssiperiaate ja pallomaisten kappaleiden kitkaton törmäys 37

8.3. Pallomaisten kappaleiden kitkallinen törmäys 39

8.5. Useiden kappaleiden törmäyksien hallinnasta 41

9. KIRJALLISUUTTA 43

10. LIITTEET 44

1. Johdanto

Simulaatioiden harjoitusmateriaali käsittää tämän dokumentin lisäksi siihen suunnitellut 15 harjoitusta, jotka on suunniteltu versiolle *VPython* 7. Harjoitukset kattavat mekaniikan keskeiset aihealueet sekä lisäksi jonkin verran ympyräliikkeen matematiikkaa. Harjoitukset ovat python version 3.7.3 kooditiedostoja, joissa tarkemmat ohjeet, ympäristö ja alkuasetukset ovat valmiiksi kirjoitettu. Koodi on heti ajettavissa, jolloin alkutilanne nähdään samalla. Toiminnan fysiikka pitää ohjelmoida sekä myös tiettyjä algoritmillisia ratkaisuja. Harjoitukset 1-5 sisältävät kineettisen simulaation ja ohjelmointiympäristön perusharjoitteita, joten ne kannattaa tehdä ensimmäiseksi. Sen jälkeen harjoitusten aiheet etenevät fysiikan kannalta samalla periaatteella, mitä tyypillisissä oppikirjoissa asteittaisesti vaikeutuen. Harjoitukset 6, 7 ja 8 liittyvät kuitenkin vain ympyräliikkeen matemaattiseen ohjelmointiin.

Harjoituskoodit on pyritty varustamaan runsailla suomenkielisillä kommenteilla. Koodi on jäsennetty niin, että fysiikan teorian vaatima osuus erottuisi selkeästi *VPython* 7 -ympäristöön liittyvistä koodiosuuksista. Harjoituksissa myös edetään systemaattisesti kohti toisaalta *VPython* 7 -ympäristön ja toisaalta koko mekaniikan alueen hallintaa. Materiaali sopii heti kaikille, joilla on hyvät pohjatiedot ja taidot vektorilaskennasta ja mekaniikasta sekä ovat joskus hankkineet perustaidot jollakin perinteisellä ohjelmointikielellä. Jos edellä mainitut pohjatiedot puuttuvat kokonaan, on materiaali vaativaa. Siinä ei ole esitetty vektorilaskennan ja mekaniikan pohjatietoja alusta lähtien kuten oppikirjoissa. Sekin käy, että samalla opiskelee näitä aiheita oppikirjoista.

Dokumentti harjoituksineen on vapaasti käytettävissä ”huvin vuoksi” harjoittelumateriaalina sekä myös kerho- tai opetusmateriaalina julkisissa oppilaitoksissa. Materiaalin julkaisemiseen ja kaupalliseen käyttöön tulee saada lupa allekirjoittaneelta.

Dokumentin taustahistoriaa

Materiaali tähän dokumenttiin on syntynyt hiljalleen vuosien saatossa. Toimiessani Rovaniemen (myöhemmin Lapin) amk:ssa matemaattisten aineiden opettajana, tuli monien uudistusmylläköiden yhteydessä tarve uudistaa myös fysiikan opetus tietotekniikan insinööriopiskelijoille enemmän ohjelmointia tukevaksi. Vanha ”paperifysiikka” ei tukenut ohjelmointitaitojen kehittymistä.

Tuohon aikaan noin vuoden 2012 tienoilla aika monissa tekniikan alan kouluissa otettiin käyttöön pelifysiikan nimisiä fysiikan kursseja ja tätä nimeä käytettiin myös Rovaniemellä. Uusi kurssi vaati myös täysin erilaisen oppimisympäristön. Tuolloin kollegani Jouko Teeriaho löysi *VPython* ohjelmiston (Visual Python moduuli), joka osoittautui erittäin soveliaaksi ympäristöksi fysiikan perusteiden opiskeluun simulaatioiden avulla. Joukon osalle tuli englanninkielisen koulutusohjelman pelifysiikan kursin kehittäminen ja minä vastasin suomenkielisestä. Nyt valmistunut dokumentti sisältää entisen opetusmateriaalini lisäksi paljon täydennettyä materiaalia, jota on syntynyt oman mielenkiinnon ja harrastukseni pohjalta. Lisäksi tässä välissä VPython 6 on vaihtunut VPython 7:ksi, ja nämä ympäristöt eivät ole yhteensopivia. Materiaali löytyy kokonaisuudessaan web-sivuiltani http://www.kpeisa.fi/simulaatiot/.

Kari Peisa 2020

2. Deterministinen ja stokastinen simulaatio

2.1. Kinematiikan perusteet simulaatioissa

	Kinematiikan lait
Tasainen liike	$\bar{\boldsymbol{r}} = \bar{\boldsymbol{r}}_0 + \bar{\boldsymbol{v}}_0 t$
Tasaisesti kiihtyvä liike	$\bar{\boldsymbol{v}} = \bar{\boldsymbol{v}}_0 + \bar{\boldsymbol{a}} t$ $\bar{\boldsymbol{r}} = \bar{\boldsymbol{r}}_0 + \bar{\boldsymbol{v}}_0 t + \frac{1}{2}\bar{\boldsymbol{a}}\, t^2$

$\bar{\boldsymbol{r}}_0, ja\ \bar{\boldsymbol{v}}_0$ ovat alkunopeuksia, t = juokseva aika

Simulaatiot toteutetaan tässä dokumentissa kahden periaatteeltaan erilaisen menetelmän avulla, joita kutsun erotukseksi nimillä *deterministinen* ja *stokastinen* simulaatio. Menetelmät eroavat toisistaan kuitenkin vain algoritmillisesti. Mallinnuksessa kummassakin käytetään samoja fysiikan ja matematiikan lakeja. Usein simulaatio voidaan myös toteuttaa molemmilla tavoilla.

Deterministisessa simulaatiossa me voimme muodostaa ajasta riippuvan vektorifunktion $\bar{\boldsymbol{r}}(t) = (x(t), y(t), z(t))$, joka kertoo kappaleen paikan $\bar{\boldsymbol{r}}$ kullakin ajan hetkellä t. Aika juoksee simulaatiossa pienin hyppäyksin eteenpäin. Tämä merkitään ohjelmoinnissa $t = t + dt$ tai myös $t += dt$, millä tarkoitetaan, että ajan edelliseen arvoon lisätään dt. Kappaleen paikka lasketaan kullakin ajanhetkellä saman vektorifunktion $\bar{\boldsymbol{r}}(t)$ avulla.

$$obj.pos = \bar{\boldsymbol{r}}(t)$$
$$t += dt$$

Stokastisessa simulaatiossa me tiedämme kullakin ajanhetkellä vain kappaleen paikan ja sen, miten kappaleen kiihtyvyys ja nopeus tuossa tilanteessa voidaan laskea. Edellisten avulla kappaletta siirretään joko tasaisen tai tasaisesti kiihtyvän liikkeen periaatteella eteenpäin pienen ajanhetken dt verran seuraavaan paikkaan. Tämä menetelmä noudattaa nk. *Eulerin algoritmin* periaatetta, jota käytetään myös differentiaaliyhtälöiden numeerisissa ratkaisuissa. Kappaleessa 6. C. ja liitteessä 2 tutustutaan myös tarkempaan *Runge-Kutta-Nyström* -menetelmään simulaatioiden ratkaisuna. Kappale on merkitty kahdella tähdellä, mikä merkitsee, että siinä käytetään matemaattisesti vaativampia menetelmiä kuin muissa dokumentin kappaleissa.

Stokastisessa simulaatiossa kussakin pysähdyskohdassa lasketaan kappaleen kiihtyvyys (ellei ole vakio) ja nopeus aina uudestaan, ja näiden perusteella siirrytään seuraava askel eteenpäin joko tasaisen tai tasaisesti kiihtyvän liikkeen periaatteella. Esimerkiksi, kun kiihtyvyys $\bar{\boldsymbol{a}}$ = vakio, lasketaan uusi sijainti kaksivaiheisen algoritmin mukaan

$obj.vel += \bar{\boldsymbol{a}} * dt$	1. päivitetään ensin uusi nopeus
$obj.pos += obj.vel * dt + \frac{1}{2}\bar{\boldsymbol{a}} * dt^2$	2. päivitetään kappaleen paikka
$t += dt$	

Deterministinen algoritmi voidaan yleensä aina muuttaa stokastiseksi, kuten seuraava esimerkkikin osoittaa. Toisinpäin tilanne ei ole ollenkaan sama kuten esimerkiksi useiden kappaleiden törmäyssimulaatioissa.

Esimerkki 1. Koordinaatistossa 1 yksikkö vastaa 1 metriä. Tunnetaan kolme pistettä $A = (-30,20,0)$, $B = (-30,-20,0)$ ja $C = (0,0,0)$. a) On luotava paikkaan A pallo 1, joka siirtyy vakionopeudella 5.0 m/s linjan AC suuntaisesti b) On luotava paikkaan B pallo 2, joka siirtyy alkunopeudella 1.5 m/s ja vakiokiihtyvyydellä 2.0 m/s^2 linjan BC suuntaisesti.

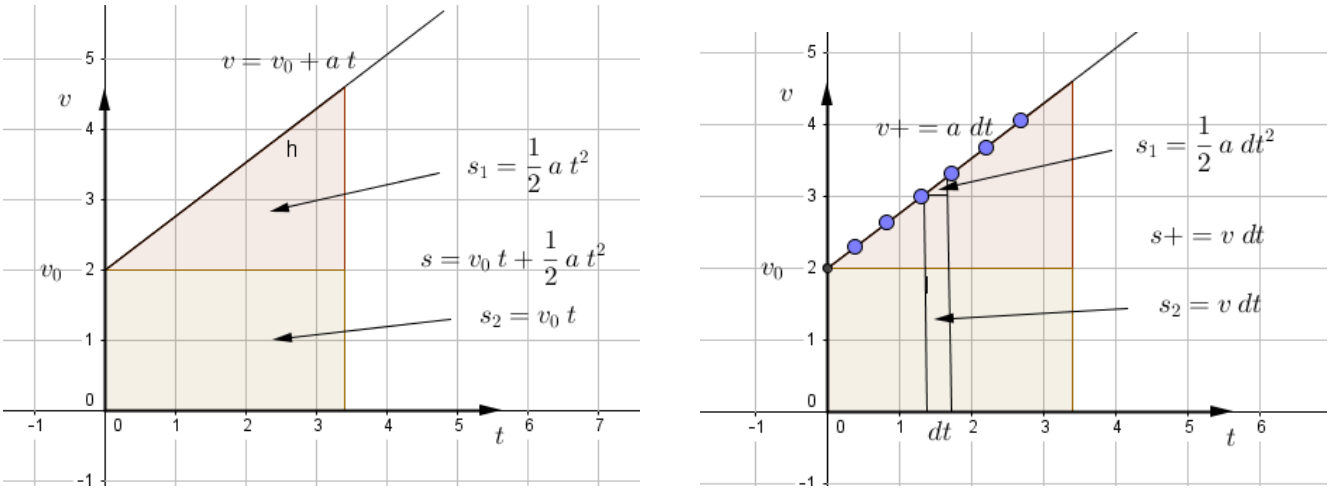
Deterministinen ratkaisu

```
# Liikkeiden alku- ja päätepisteet
A=vector(-30,20,0)
B=vector(-30,-20,0)
C=vector(0,0,0)

# Yksikkövektorit A:sta C:hen ja B:stä C:hen
s10=norm(C-A)
s20=norm(C-B)

# Pallot ja niiden alkuarvojen alustus
ball1=sphere(pos=A,radius=1,color=color.red)
velocity1=5*s10
ball2=sphere(pos=B,radius=1,color=color.blue)
initVelocity2=2*s20
acceleration2=1.5*s20


# Simulaatioaskel ja -luuppi
t=0
dt=0.01

while (1):
    rate(100) # simulaation tajuus
    if ball1.x < C.x : # pysähtymisehto
        ball1.pos = A + velocity1*t
    if ball2.x < C.x : # pysähtymisehto
        ball2.pos = B + initVelocity2*t +0.5*acceleration2*t**2
    t+=dt
```

Stokastinen ratkaisu

```
# Liikkeiden alku- ja päätepisteet
A=vector(-30,20,0)
B=vector(-30,-20,0)
C=vector(0,0,0)

# Yksikkövektorit A:sta C:hen ja B:stä C:hen
s10=norm(C-A)
s20=norm(C-B)

# Pallot ja niiden alkuarvojen alustus
ball1=sphere(pos=A,radius=1,color=color.red)
velocity1=5*s10
ball2=sphere(pos=B,radius=1,color=color.blue)
ball2.velocity=2*s20
ball2.acceleration=1.5*s20


# Simulaatioaskel ja -luuppi
t=0
dt=0.01

while (1):
    rate(100) # simulaation tajuus
    if ball1.x < C.x : # pysähtymisehto
        ball1.pos += velocity1*dt
    if ball2.x < C.x : # pysähtymisehto
        ball2.velocity += ball2.acceleration*dt
        ball2.pos += ball2.velocity*dt + 0.5*ball2.acceleration*dt**2
    t+=dt
```

Determinististä ja stokastista menetelmää voi havainnollistaa myös oheisilla kuvioilla.



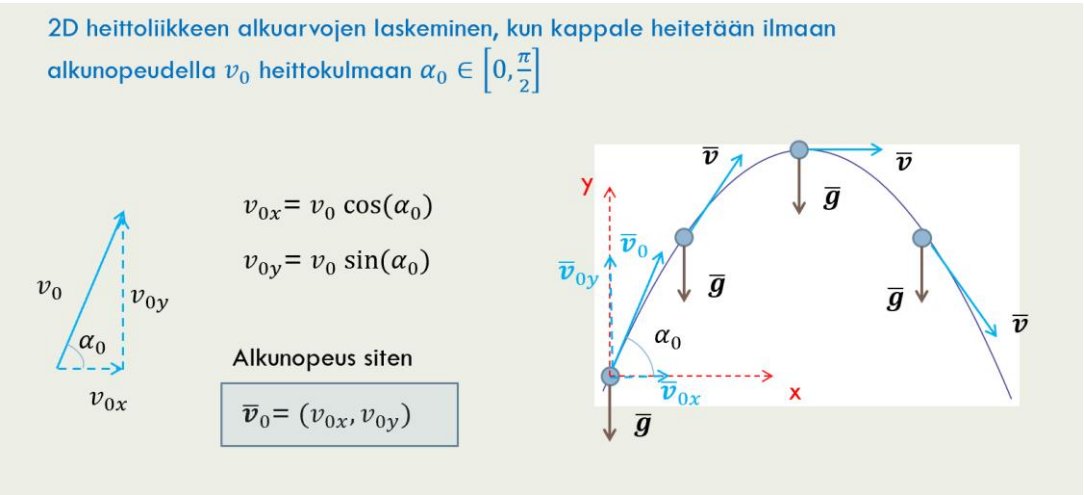
Kuva 1. Deterministinen (vas.) ja stokastinen (oik.) menetelmä (t, v) -koordinaatistossa

Kuvasta 1 nähdään, että kun dt on pieni, on paikan päivityksessä $\frac{1}{2}\bar{\boldsymbol{a}} * dt^2 \approx 0$. Useimmiten ei huomaa eroa, jos kyseisen termin jättää kokonaan pois. Tällöin liikettä mallinnetaan aika-askeleella dt tasaisen nopeuden periaatteella. Kun tarvitaan mahdollisimman tarkkaa approksimaatiota todelliselle käyttäytymiselle, käytetään Eulerin menetelmän sijasta Runge-Kutta-Nyström -menetelmää (Liite 2).

Tee harjoitukset 1, 2

2.2. Ilmanvastukseton heittoliike (gravitaatiovoiman alainen)

Ilmanvastukseton heittoliike on yhden voiman, gravitaatiovoiman, aiheuttamaa tasaisesti kiihtyvää liikettä, jonka vakiokiihtyvyys on maan vetovoiman kiihtyvyys $g = 9.81\ \frac{m}{s^2}$ ja suunta kohtisuoraan alas. Siten mitään uutta fysiikan teoriaa edelliseen kappaleeseen nähden emme sen simuloimiseen tarvitse. Sen sijaan joudumme turvautumaan trigonometriaan alkuarvojen laskemisessa.



Kun siirrytään 3D heittoliikkeeseen, meidän tulee nousukulman α_0 lisäksi tietää alkunopeuden suunta maan pinnan tasossa, joka *VPythonissa on oletusarvoisesti xz-taso*. Tämä suunta on sama, jonka nopeusvektori projisoi xz-tasolle. Tämä suunta voidaan esittää yksikäsitteisesti trigonometristen funktioiden perusominaisuuksiin nojautuen käyttämällä kiertokulmaa $\varphi \in [0,2\pi]$ (tai vaihtoehtoisesti $\varphi \in [-\pi, \pi]$). Alla olevan 3D-koordinaatistokuvan mukaisten suorakulmaisten kolmioiden avulla voidaan laskea alkunopeuden komponenttien suuruudet kulmien α_0 ja φ avulla sekä muodostaa alkunopeusvektori kaikissa mahdollisissa nousukulmissa ja xz-tason suunnissa.

3D heittoliikkeen alkuarvojen laskeminen, kun kappale heitetään ilmaan alkunopeudella v_0 nousukulmaan $\alpha_0 \in \left[0, \frac{\pi}{2}\right]$ ja suuntaan $\varphi \in [0, 2\pi]$. Nousukulma α_0 kiertyy xz-tasosta pos. y-akselin suuntaan ja suuntakulma φ z-akselista vastapäivään katsottaessa pos. y-akselin suunnasta kohti origoa.

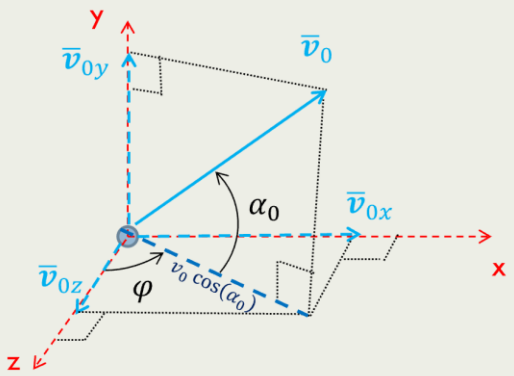
$$v_{0x} = v_0 \cos(\alpha_0) \sin(\varphi)$$

$$v_{0z} = v_0 \cos(\alpha_0) \cos(\varphi)$$

$$v_{0y} = v_0 \sin(\alpha_0)$$

Alkunopeus siten

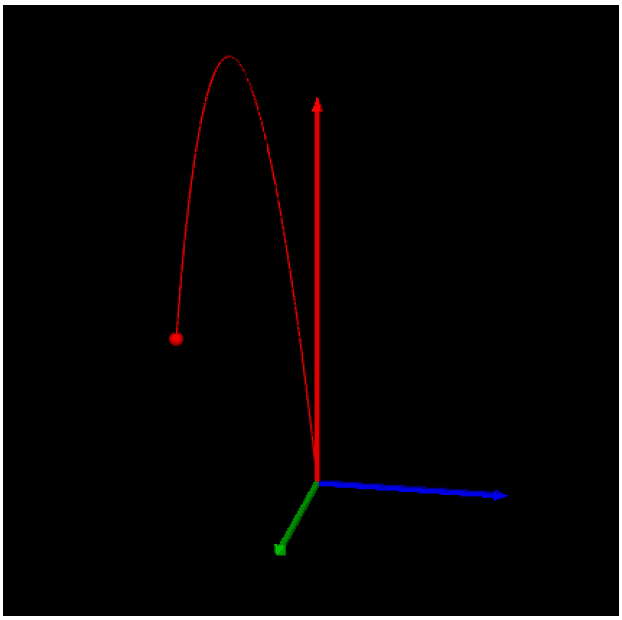
$$\boxed{\bar{\boldsymbol{v}}_0 = (v_{0x}, v_{0y}, v_{0z})}$$



Alkuasetuksien jälkeen itse heittoliikkeen mallinnus on yhtä helppo kuin suoraviivaisenkin tasaisesti kiihtyvän liikkeen tapauksessa. Lentoradan muodostuminen, kun kiihtyvyys on vakio, noudattaa samoja kinematiikan lakeja riippumatta alkunopeuden ja kiihtyvyyden keskinäisistä suunnista. Tämä selviää sitten dynamiikassa, miksi näin on.

Esimerkki 2. Pallo heitetään alkunopeudella 30 m/s origosta 70 asteen nousukulmassa suuntaan, joka on z-akselista 210 astetta vastapäivään. Simuloi pallon ilmanvastukseton liike.

```
from visual import *

#Python

#Koordinaatisto, jotta 3D kliike voidaan havainnoida
arrow(pos=(0,0,0), length=20,axis=(1,0,0), shaftwidth=0.5,color=color.blue)
arrow(pos=(0,0,0), length=40,axis=(0,1,0), shaftwidth=0.5,color=color.red)
arrow(pos=(0,0,0), length=20,axis=(0,0,1), shaftwidth=0.5,color=color.green)

ball = sphere(pos=(0,0,0), radius=1, color=color.red,make_trail=True)
# Vakiot
g=vector(0,-9.81,0)
v0=30
alpha0=radians(70)
fii=radians(210)



# Alkunopeuden määritys
v0x=v0*cos(alpha0)*sin(fii)
v0z=v0*cos(alpha0)*cos(fii)
v0y=v0*sin(alpha0)
ball.velocity=vector(v0x,v0y,v0z)

# toistoluuppi
dt=0.01

while(1):
    rate(100)
    if (ball.y>=0):
        ball.velocity += g*dt
        ball.pos += ball.velocity *dt
```



Kuva 2. Esimerkin 2 ajo. Pallon jälki saadaan asettamalla pallon konstruktorissa attribuutti `make_trail=True`.

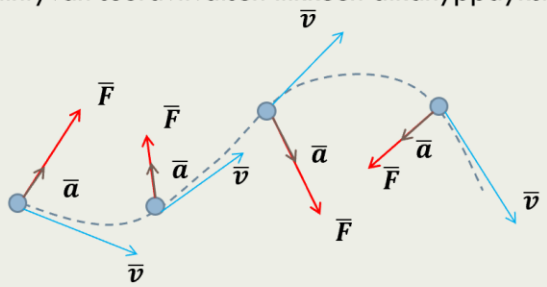
Tee harjoitus 3

3. Dynamiikan peruslause simulaation ratkaisuna

- Kun $\overline{\boldsymbol{F}}$ on kappaleeseen vaikuttava voima (tai kaikkien voimien vektorisumma) on kiihtyvyysvektori $\overline{\boldsymbol{a}}$ aina voiman $\overline{\boldsymbol{F}}$ suuntainen, koska **dynamiikan peruslauseen** mukaan

$$\boxed{\sum \overline{\boldsymbol{F}} = m\,\overline{\boldsymbol{a}}}$$

- Kappaleen nopeus ja paikka voidaan päivittää voiman ja sitä kautta lasketun kiihtyvyyden avulla approksimoiden liikettä pienillä tasaisen tai tasaisesti kiihtyvän suoraviivaisen liikkeen aikahyppäyksillä.



Kun **Eulerin algoritmin** mukainen menetelmä toteutetaan dynamiikan peruslakiin nojautuen, saadaan seuraava algoritmi, joka pitää laskea simulaation luupissa:

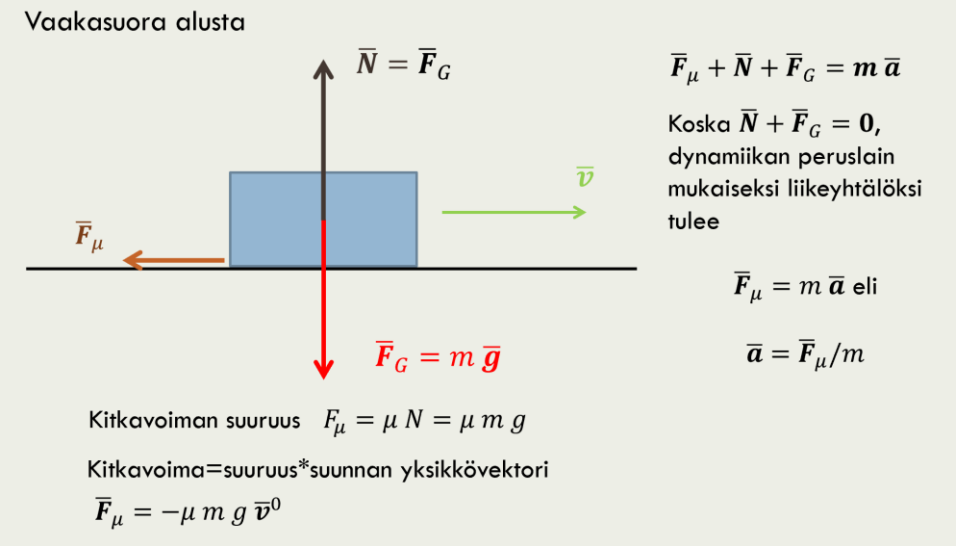
1. Laske kappaleeseen vaikuttava kokonaisvoima $\sum \overline{\boldsymbol{F}}$
2. Laske kiihtyvyys $\overline{\boldsymbol{a}} = \sum \overline{\boldsymbol{F}}\,/m$
3. Päivitä nopeus $\overline{\boldsymbol{v}} \mathrel{+}= \overline{\boldsymbol{a}}\,dt$
4. Päivitä paikka $\overline{\boldsymbol{pos}} \mathrel{+}= \overline{\boldsymbol{v}}\,dt$

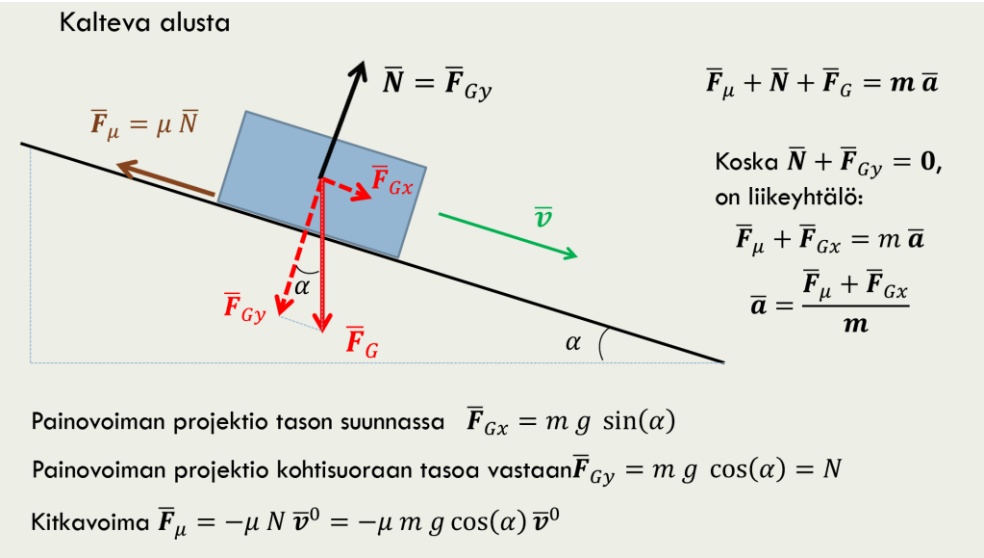
Kun simulaatioluupin kullakin kierroksella on tiedossa kappaleen, jonka massa on m, sen hetkinen paikka, sekä voidaan laskea kappaleeseen vaikuttavien voimien resultantti, voidaan yllä olevalla algoritmilla approksimoida kappaleen seuraava paikka pienen ajanhetken dt jälkeen, ja jatkaa toistoluuppia alusta.

Tämä mahdollista monien uusien kineettisten ilmiöiden simuloinnin jopa sellaisten, joissa deterministisen vektorifunktioon perustuvan ratkaisun löytäminen on mahdotonta.

3.1. Kitkavoima

Tässä käsitellään liukumis- ja vierimiskitka. Niiden suuruus voidaan laskea materiaaleista riippuvan kitkakertoimen μ ja pintojen välisen puristusvoiman $\overline{\boldsymbol{N}}$ eli normaalivoiman avulla. Tämän suuruus riippuu tarkasteltavissa tilanteissa kappaleen painovoimasta $\overline{\boldsymbol{F}}_{\boldsymbol{G}}$ ja kaltevuuskulmasta α.





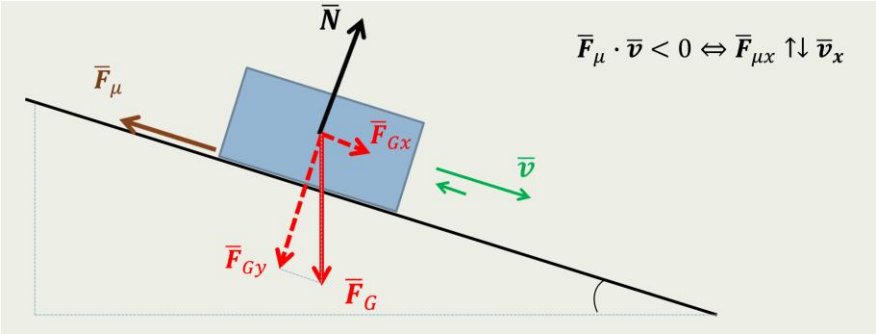
Pysähtyminen kitkavoimien aiheuttamana

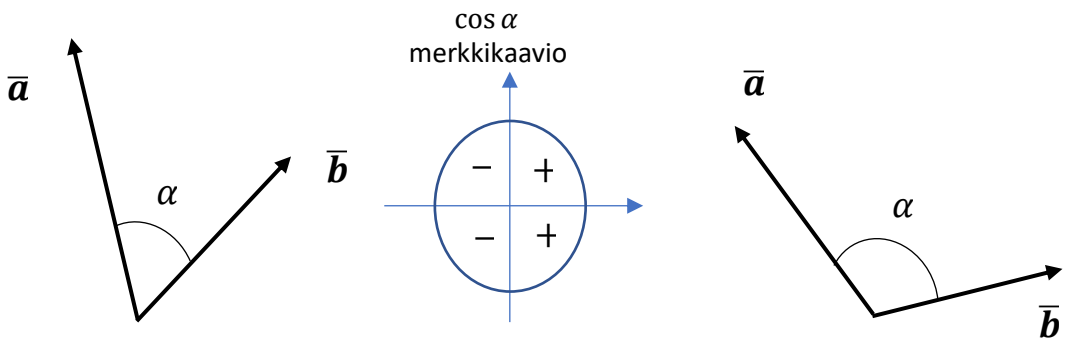
Kun kappale hidastuu kitkavoiman aiheuttamana, tulee hetki, jolloin kappaleen pitää pysähtyä.

Simulaatiossa voi joskus kappaleen liike jatkua pysähtymisen jälkeen taaksepäin kiihtyvällä nopeudella. Tämä johtuu siitä, että simulaation laskennassa nopeus pienenee askelittain ja ei välttämättä saa koskaan arvoa nolla, vaan ”hyppää” suoraan negatiivisiin arvoihin.

Edellinen on tyypillinen algoritmillinen ongelma, joita fysiikan simulaatioissa tulee usein vastaan. Ongelma voidaan korjata useallakin tavalla. Hyvin toimiva yleinen keino on määritellä pienin **kynnysarvo** nopeuden muutokselle, jonka jälkeen nopeus asetetaan pysyvästi nollaksi.

Toisinaan voi myös voimien suuntien avulla selvittää pitääkö nopeus asettaa nollaksi. Esimerkiksi, jos kitkavoiman suunta lasketaan geometrian perusteella niin, että sen suunta ei muutu kappaleen nopeuden muuttuessa, voi pysähtymishetken määritellä **vektoreiden pistetulon** avulla. Kun nopeusvektorin ja kitkavoiman välinen pistetulo on negatiivinen, on se merkki siitä, että niiden suunnat ovat vielä vastakkaiset (vektoreiden välinen kulma on tylppä, kts vektorilaskenta alla). Toistoluupissa tulee laittaa ehto, jossa kitkavoiman vaikutus huomioidaan vain, jos sen ja nopeuden välinen pistetulo on negatiivinen.





$$\bar{\boldsymbol{a}} \cdot \bar{\boldsymbol{b}} = a_x b_x + a_y b_y + a_z b_z = |\bar{\boldsymbol{a}}||\bar{\boldsymbol{b}}| \cos\alpha$$

$$\cos\alpha = \frac{\bar{\boldsymbol{a}} \cdot \bar{\boldsymbol{b}}}{|\bar{\boldsymbol{a}}||\bar{\boldsymbol{b}}|}$$

Tee harjoitus 4

3.2. Ilmanvastuksellinen heittoliike

Empiirinen kaava ilmanvastukselle

$\overline{\boldsymbol{F}}_v = -bv^2\,\overline{\boldsymbol{v}}^0, \qquad b = 0.5\,C\,\rho\,A$

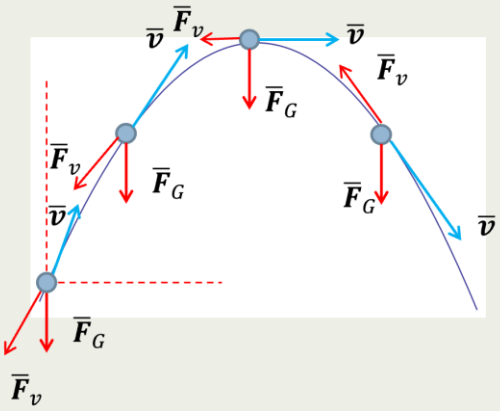
Painovoima

$\overline{\boldsymbol{F}}_G = m\,\overline{\boldsymbol{g}}$

Kokonaisvoima

$\overline{\boldsymbol{F}} = \overline{\boldsymbol{F}}_v + \overline{\boldsymbol{F}}_G$

C = muotokerroin
ρ = ilman tiheys
A = poikkipinta-ala liikesuuntaan nähden



Ilmanvastuksen laskeminen perustuu empiirisiin kaavoihin, joista yllä on esitetty Newtonin vastuslaki **väliaineen (ilman) vastukselle** riittävän suurilla nopeuksilla. Se on karkeasti ottaen voimassa, kun nopeus on niin suuri, että kappaleen pinnalle muodostuu turbulenssia. Vastuksen tarkka laskeminen on vaikeaa, koska sen suuruus riippuu oleellisesti turbulenssin kehittymisestä. Muotokerroinkaan C ei ole vakio, vaan riippuu kappaleen muodon lisäksi väliaineen materiaalisista ominaisuuksista sekä virtausnopeudesta. Alhaisemmilla nopeuksilla väliaine virtaa liikkuvan kappaleen pinnalla laminaarisesti, jolloin vastus on suoraan verrannollinen vain nopeuden 1. potenssiin.

Ilmanvastuksen suunta on aina vastakkainen kappaleen nopeuteen nähden. Nopeus puolestaan muuttaa jatkuvasti suuntaansa. Tämän vuoksi ilmanvastuksellisen heittoliikkeen mallintaminen deterministisesti on hyvin vaikeaa. **Eulerin menetelmällä ilmanvastuksellista heittoliikettä voidaan kuitenkin approksimoida helposti sivulla 7 olevan algoritmin avulla**.

Ilmanvastuksen laskemisessa kuten hyvin usein fysiikan vektorisuureiden laskemisessa käytetään periaatetta: **fysiikan kaavan avulla lasketaan suureen suuruus, joka muunnetaan oikean suuntaiseksi ja suuruiseksi vektoriksi sopivan yksikkövektorin avulla**. Yksikkövektorilla kertominen ei muuta lasketun suureen suuruutta. Ilmanvastuksen suuruus lasketaan Newtonin vastuskaavan avulla ja suunta saadaan sen hetkisen nopeusvektorin vastayksikkövektorista $-\overline{\boldsymbol{v}}^0$. Kaavassa käytetään nopeusvektorin neliötä, joka saadaan laskettua pituuksien tulona tai myös vektoreiden pistetulona $\overline{\boldsymbol{v}}^2 = |\overline{\boldsymbol{v}}|^2 = \overline{\boldsymbol{v}} \cdot \overline{\boldsymbol{v}}$ (pistetulo, kts. sivu 10). *VPython*issa edellinen voidaan laskea myös funktion `mag2()` avulla.

Yleisesti käytettävät vakiot ilmanvastuksen laskemisessa:

C = 1, tasapaksu levy kohtisuorassa ilmavirtausta vasten
C = 0.5, pallo
C = 0.06, putoava pisara
C = 0.2-0.4, henkilöauto
$\rho = 1.29\ \frac{kg}{m^3}$ (NTP-tilassa lämpötila 0 °C, paine 1 013 hPa)

3.3. Vuorovaikutusta *VPython* -simulaatioon widget-elementtien avulla

VPythonin simulaatiossa on usein tarve kokeilla simulaatiota erilaisilla alkuarvoilla sekä tulostaa suureiden arvoja simulaation ollessa käynnissä. Tähän tarkoitukseen on mahdollista käyttää graafisia widget -elementtejä. Alla olevissa koodilistauksissa käsitellään esimerkkiä widget -elementtien ja tekstikehyksen (label) käytöstä simulaation vuorovaikutteisuuden lisäämiseen. Widget -elementeiksi voidaan lisätä mm. *painikkeita* (button) ja *liukusäätimiä* (slider) ja valintamenuja (menu). Elementeille tulee määritellä *tapahtumankäsittelijäfunktiot*. Parhaiten näiden hyödyntämisen oppii kopioimalla harjoituksissa valmiina olevia koodilohkoja ja tekemällä niihin tarpeelliset muutokset omiin tarkoituksiinsa. Tarkoitus ei ole perehtyä syvällisemmin näiden graafisten elementtien ominaisuuksiin.

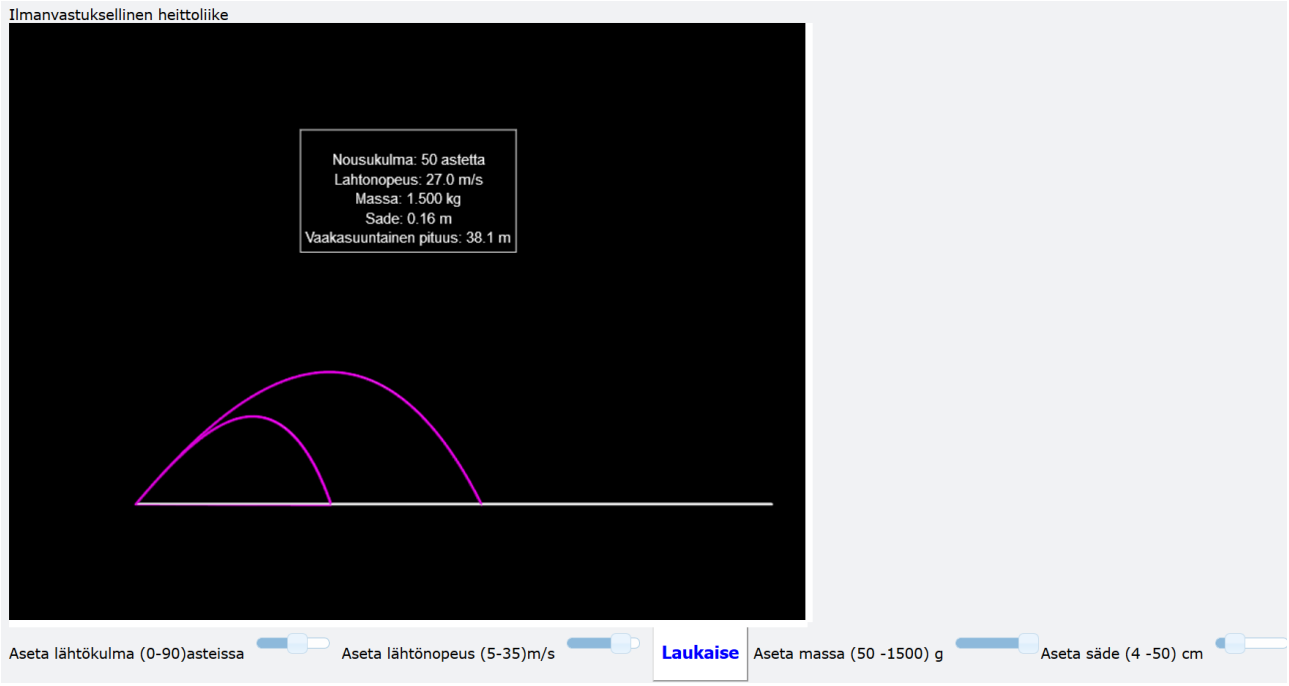
VPython 7:ssa simulaatio toteutetaan selainohjelman sivulla. Widget -elementit sijoittuvat aina web-sivulla varsinaisen simulaatioikkunan ylä- tai alapuolelle. Myös erilaiset graafit (graph) sijoittuvat simulaatioikkunassa web-sivun align -komentojen mukaisesti joko vasemmalle tai oikealle.

Tekstikehykset (label) mahdollistavat muutettavien alkuarvojen sekä simulaatioluupissa laskettavien muuttujien arvojen tulostuksen reaaliaikaisesti näytölle (Kuva 3.). Tekstikehyksen määrittelyssä kannattaa käyttää erillistä funktiota, jolla päivittää sisältöä alla olevan koodiesimerkin mukaisesti. Huomioitavaa on, että widget-elementtien arvot näkyvät tekstikehyksessä reaaliaikaisesti vain, jos niille on sidottu (bind) tapahtumankäsittelijäfunktio.

```
#tekstikehys ikkunaan ja sen päivitysfunktio
textLabel = label(pos=vector(0,20,0))
def updTextLabel(txtlabel):
    text= "\nNousukulma: %.0f astetta" % slider1.value
    text += "\nLahtonopeus: %.1f m/s" % slider2.value
    text += "\nMassa: %.3f kg" % slider3.value
    text += "\nSade: %.2f m" % slider4.value
    text += "\nVaakasuuntainen pituus: %.1f m" % (ball.pos.x-start.x)
    txtlabel.text=text
```

Simulaation toistoluupissa tekstikehys päivitetään kutsumalla funktiota `updTextLabel()`.

```
while True:
    rate(100)
    .
    .
    .
    # tekstikehyksen päivitys näytölle
    updTextLabel(textLabel)
```



Kuva 3. Simulaatioikkuna, jossa tekstikehys, sekä widget-elementit

Tekstikehyksen text-elementin tulostus noudattaa yleistä print() -lauseen Python-syntaksia. Tällöin voi tulostaa muuttujien arvoja monella tapaa, esimerkiksi merkkijonona

```
text="Muuttujan x arvo = " + str(x)
```

Tällöin muuttujan `x` esityksen muotoilu voi olla pettymys. Mikäli haluaa tulostaa reaaliluvut halutulla muotoilulla, voi käyttää arvon tyypin esittelyä (`i` tai `d` kokonaisluku, `f` liukuluku) yhdistettynä esityksen tarkkuuteen operaattorin `%` avulla esim. seuraavasti (liukuluku, 1 desimaalin tarkkuus):

```
text="Nopeuden v suuruus = %.1f m/s" % mag(v)
```

Widget-elementit ovat sisäänrakennettu moduuliin VPython 7, joka ladataan koodin alussa. Oheiset kaksi riviä tulee olla kaikissa tämän dokumentin harjoituksissa

```
from __future__ import division # (3/2=1.5 ei 1)
# interaktiivinen grafiikka
from vpython import *
```

Ensimmäinen rivi huolehtii siitä, että kokonaislukujen jakolaskuperiaatetta ei noudateta kuten kommentista ilmenee.

Jos pääohjelman funktioissa viitataan pääohjelmassa määriteltyyn muuttujaan, tulee se esitellä määreellä `global` (kts. koodilistaus jäljessä). Muutoin Python ylikirjoittaa muuttujasta paikallisen muuttujan, jonka arvo ei välity pääohjelmaan eikä ohjelmoija sitä välttämättä havaitse. Ongelma ei välttämättä esiinny oliotyyppisillä muuttujilla, mutta turvallisinta on aina edellisessä tilanteessa käyttää määrettä `global`.

Samoin tulee olla tarkkana, että kaikki tapahtumankäsittelijäfunktion tekemät muutokset tulevat myös simulaation toistoluupissa huomioiduksi. Esimerkiksi koodilistauksessa gravitaatiovoima voidaan päivittää widget-ikkunan shot -painikkeen tapahtumankäsittelijässä pääohjelman

toistoluupin sijasta, koska se on vakioarvoinen massan ollessa vakio toistoluupissa. Sen sijaan ilmanvastus tulee laskea toistosilmukan jokaisella kierroksella.

Liukulukujen ja painikkeiden tapahtumankäsittelyssä on järkevää asettaa simulaatio jonkinlaiseen aloitustilaan. Tällöin kannattaa rakentaa erillinen funktio `setStart()`, jossa tämä tehdään ja jota voidaan kutsua kaikista tapahtuman käsittelyn funktioista. Huomaa, että jälkimmäisille annetaan aina muodollinen argumentti evt (event = tapahtuma), jota ei tarvitse kuitenkaan mitenkään käsitellä. Sen hoitaa käyttöjärjestelmä, joka ”kuuntelee” jatkuvasti elementtien lähettämiä viestejä ja reagoi suorittamalla tapahtuman käsittelyfunktion. Tapahtumankäsittelijäfunktiossa ei välttämättä tarvitse olla mitään toimintaa, mutta funktion olemassaolo vaaditaan esimerkiksi pääohjelman tekstikehyksen päivityksessä, kuten edellä mainittiin.

```
def setStart():
    global start,ball
    ball.pos=start #  Globaalin muuttujan arvon lukeminen
    ball.acceleration=vector(0,0,0)
    ball.velocity=vector(0,0,0)

### Tapahtuman käsittelyfunktiot
# setAngle(evt) muuttaa slider1:n mukaisesti nousukulmaa
def setAngle(evt):
    setStart()
# setSpeed(evt) muuttaa slider2:n mukaisesti lahtonopeutta
def setSpeed(evt):
    setStart()
```

Button-painikkeella laukaistaan pallo liikkeelle ja samalla luetaan liukusäätimien arvot sekä lasketaan heittoliikkeen alkuarvot.

```
# shot() ampuu pallon liikkeelle painikkeesta shotButton ja asettaa heiton alkuarvot
def shot(evt):
    global Fgravitation, ball
    setStart()
    Fgravitation=ball.mass*vector(0,-9.81,0)
    angle0=radians(slider1.value) # huom! paik. muutt.
    v0mag=slider2.value # huom! paik. muutt.
    vy=v0mag*sin(angle0) # huom! paik. muutt.
    vx=v0mag*cos(angle0) # huom! paik. muutt.
    ball.velocity=vector(vx,vy,0)
    ball.acceleration=vector(0,-9.8,0)  #Asetetaan vetovoima päälle
```

Varsinaiset widget-elementit määritetään alla olevan koodiesimerkin mukaisesti. Elementit asettuvat html-sivulla järjestykseen vasemmalta oikealle sitä mukaa, kun ne määritellään.

```
# tekstit, liukusäätimet ja painike asettuvat web-sivun alareunaan (caption)
# järjestyksessä vasemmalta oikealle

wtext(pos=scene.caption_anchor,text='Aseta lähtökulma (0-90)asteissa')
slider1=slider(pos=scene.caption_anchor,bind=setAngle,min=0,max=90,step=1,
               bottom=10,top=10,length=100)
wtext(pos=scene.caption_anchor,text='Aseta lähtönopeus (5-35)m/s')
slider2=slider(pos=scene.caption_anchor,bind=setSpeed,min=5,max=35,step=1,
               bottom=10,top=10,length=100)

shotButton=button(pos=scene.caption_anchor,bind=shot,text='<h3>Laukaise</h3>',
                  color=color.blue)
```

Liukusäätimien `min` ja `max` arvot ovat liukulukuja. Jos ne pitää laskennassa muuttaa eri yksiköiksi, tulee muutos tehdä siinä vaiheessa, kun arvoa kysytään value-attribuutilla. Edellinen voi olla joskus tarpeen, jos liukusäätimissä halutaan antaa käyttäjälle säädettäväksi sopivia yksikköjä kuten esimerkiksi pallon säde, joka annetaan cm:na, mutta laskennassa tulee käyttää metrejä.

Huom*! Virheiden välttämiseksi kannattaa aina muuntaa kaikkien kaavoihin sijoitettavien suureiden yksiköt SI-järjestelmän mukaisiksi perusyksiköiksi, jolloin kaavan tuloksena laskettavalle suureelle saadaan myös perusyksikkö.*

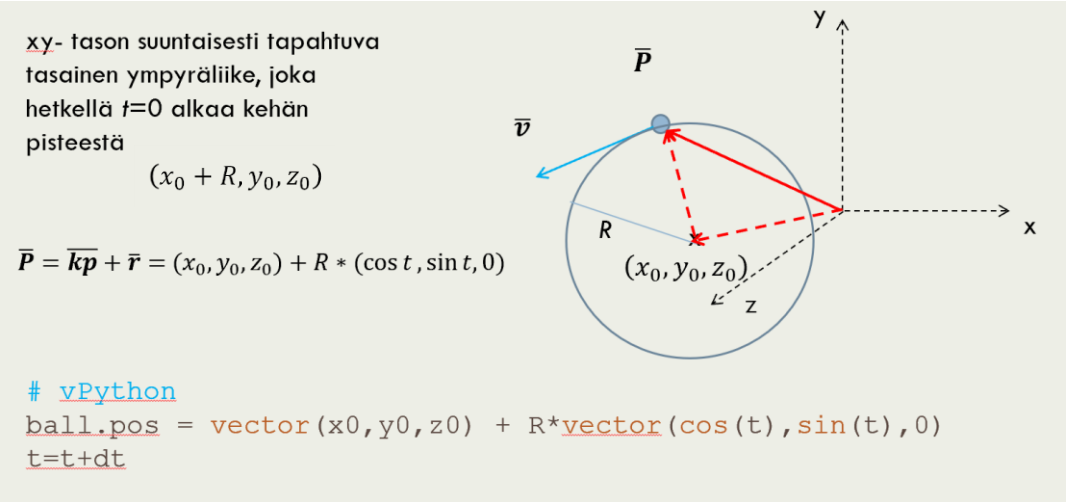
Tee harjoitus 5

4. Ympyräliikkeen deterministisiä simulaatioita

Seuraavissa kappaleissa 4.A, 4.B ja 4.C tarkastellaan erilaisten kiertoliikkeiden determinististä määrittämistä koordinaatistossa. Kappaleissa 4.A ja 4.B käsitellään matemaattista perustaa sille, miten vektorifunktioilla $\bar{\boldsymbol{r}}(t) = (x(t), y(t), z(t))$ saadaan aikaan massapisteiden ympyräliikkeitä. Kappaleessa 4.C tarkastellaan VPython ympäristön sisäänrakennetun funktion `rotate()` toimintaa.

4.1. Ympyräliike koordinaattiakseleiden muodostamassa tasossa

Ympyräliike saadaan simuloitua **vektorifunktiolla** samalla periaatteella, jolla trigonometriset funktiot sin ja cos määritellään yksikköympyrässä. Alla esimerkki ympyräliikkeestä, jossa massakappale kiertää tasaisesti keskipisteen (x_0, y_0, z_0) ympäri x- ja y-akseleiden suuntaisessa tasossa, etäisyydellä R ja vastapäivään katsottuna z-akselilta kohti origoa.



Kun halutaan vaihtaa tason muodostavat akselit, tapahtuu se vaihtamalla trigonometristen funktioiden paikkaa. Kun halutaan muuttaa kiertosuuntaa, vaihdetaan argumentin t etumerkki. Kun halutaan muuttaa kierron aloituskohtaa, tapahtuu se lisäämällä sopiva vaihekulma φ trigonometristen funktioiden argumenttiin.

$$\overline{\boldsymbol{pos}}(t) = (x_0, y_0, z_0) + R * (\cos(\pm t + \varphi), \sin(\pm t + \varphi), 0)$$

Usein simulaatiossa tulostetaan myös virtuaaliaikaa, jonka parametri t ilmoittaa. Silloin on usein tarve saada ympyräliikkeen kiertonopeus sopivaksi halutun kiertoajan T suhteen kuten esimerkiksi kuun kiertoaika maan ympäri. Tämä toteutetaan ajan t sopivalla kertoimella. Olkoon esimerkiksi kiertoaika tunnettu arvo T. Nyt tiedetään, että parametrin t saadessa arvon T, on tapahtunut yksi täysi kierros. Tällöin kertoimen ja parametrin t tulon on saatava arvo 2π. Näin ollen ympyräliikkeen hienosäätö voidaan tehdä seuraavasti:

$$(\cos(\pm \frac{2\pi}{T} t + \varphi), \sin(\pm \frac{2\pi}{T} t + \varphi), 0)$$

Virtuaaliajan ja todellisen reaaliajan täsmääminen riippuu jonkin verran jo tietokoneen laskentatehostakin. Mutta likipitäen oikean suhteen voi säätää simulaation toistoluupissa olevalla pakollisella `rate(N)` funktiolla, jossa argumentti N määrää jokaisen toistokierroksen viiveen

vähimmäismäärän muodossa $\frac{1}{\mathrm{N}}$ sek. Eli, jos $dt = 0.01\,sek$, kannattaa asettaa $\mathrm{N} = \frac{1}{dt} = 100$, mikäli haluaa reaaliajan toteutuvan myös virtuaalisesti kuvaruudulla.

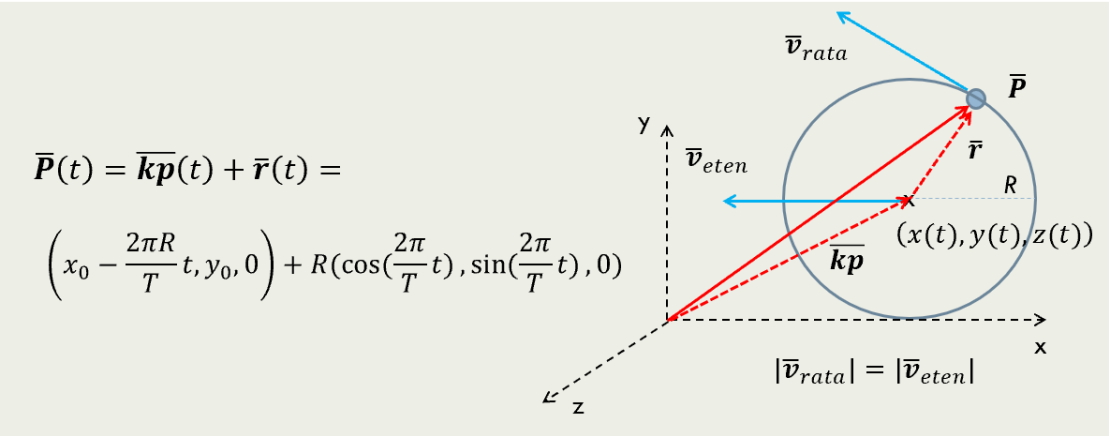
Yhdistettyjen liikkeiden, joissa ympyräliike on mukana, mallinnuksen perusidea on seuraava:

1. Määritä ympyräliikkeen kierrosaika T
2. Määritä ympyräliikkeen säde, keskipisteen $\overline{\boldsymbol{kp}}$ alkupaikka sekä kuvattavan kehäpisteen $\overline{\boldsymbol{P}}$ alkupaikka (vaihekulma φ) ympyrällä
3. Muodosta kehäpisteen $\overline{\boldsymbol{P}}$ ympyräliikkeen ratakäyrän yhtälö muodossa
   $\overline{\boldsymbol{P}}(\boldsymbol{t}) = \overline{\boldsymbol{kp}}(t) + \overline{\boldsymbol{r}}(t)$,

   missä $\overline{\boldsymbol{kp}}(t) = (x(t), y(t), z(t))$ mallintaa keskipisteen etenemisliikettä (joka voi olla myös ympyräliikettä) ja $\overline{\boldsymbol{r}}(t)$ mallintaa ympyräliikettä edellä olevan keskipisteen ympäri. Ympyräliikkeessä ajalla t on kertoimena $\frac{2\pi}{T}$ kuten edellä tuli esille.

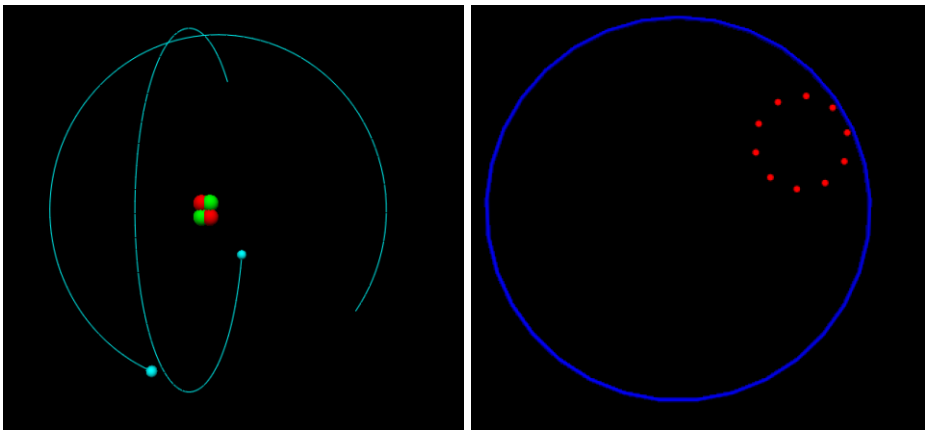
Esimerkiksi kun yhdistetään keskipisteen etenemisliike ja ympyräliike saadaan vierimisliike.

Alla esimerkki xy-tasossa tapahtuvasta vierimisestä, jossa nopeus $\boldsymbol{v}_{rata}$ on ratanopeus, joka on laskettu keskipisteessä sijaitsevan koordinaatiston suhteen.



Yllä keskipisteen etenemisliikkeessä ajan t kerroin tulee kaavasta $s = v * t$, missä nopeus saadaan siitä, kun ajassa T keskipiste etenee matkan $2\pi R$.

Vihje harjoitukseen 7. (kts. seur. sivu): Tarkastellaan tilannetta, jossa sisäympyrä vierii ulkoympyrän kehällä, joka voi myös vieriä omalla alustallaan. Tällöin sisä- ja ulkoympyrän sivuamispiste kiertää ulkoympyrän keskipistettä samassa vaiheessa keskipisteen kanssa. Kun sisäympyrän keskipiste on kiertänyt täyden kierroksen ulkoympyrän keskipisteen ympäri, on itse asiassa sivuamispiste kiertänyt myös täyden kierroksen sisäympyrän keskipisteen ympäri (vaikka liikkuukin ulkoympyrän kehällä). Tämä merkitsee, että *vierivän ympyrän etenemisnopeus eli sisäympyrän keskipisteen ratanopeus ulkoympyrän keskipisteen suhteen on yhtä suuri kuin sivuamispisteen ratanopeus sisäympyrän keskipisteen suhteen, joka on kahden ratanopeuden summa (tai erotus).* Edellisestä saadaan laskentakaava keskipisteen ja sivuamispisteen kulmanopeudelle.



Kuva 4. Helium-atomin malli ja ympyräradan sisäkehällä ”vierivä” kehäpisteiden joukko. Tee harjoitukset 6 ja 7

Mielenkiintoinen kysymys on, voidaanko edellä oleva deterministinen ympyräliikkeen algoritmi muuttaa stokastisella periaatteella toimivaksi niin, ettei vektorifunktiota ja kokonaisaikaa t käytetä paikan määrityksessä. Tämä voidaan toteuttaa **keskihakukiihtyvyyden** eli **normaalikiihtyvyyden** avulla

$$a_n = \frac{v^2}{r}.$$

Normaalikiihtyvyyden suunta on koko ajan kohti keskipistettä, ja sen avulla päivitetään kappaleen nopeutta. Deterministinen ratkaisu on hyvin vakaa ja tarkka, mitä ym. stokastinen ratkaisu ei välttämättä ole. Toisaalta reaalimaailmassa yleisemmin joudutaan simuloimaan ratakäyriä, jotka eivät ole ympyröitä. Silloin normaalikiihtyvyys on simulaatiossa välttämätön.

4.2. Kierretyn ratakäyrän vektorifunktio (*)

Edellä ympyräliikkeitä on tarkasteltu vain erikoistapauksissa, joissa liike tapahtuu kahden koordinaattiakselin muodostamassa tasossa. Tässä tarkastellaan yleisesti minkä tahansa vektorifunktion $\bar{\boldsymbol{r}}(t) = (x(t), y(t), z(t))$, missä t on virtuaaliaika, määrittelemän ratakäyrän kiertymistä jonkin kiertoakselin suhteen. Aluksi tarkastellaan kiertymistä koordinaattiakseleiden suhteen. Kierto voidaan kuvata yhtälöryhmästä satavalla matriisilaskennan esityksellä. Sen jälkeen tarkastellaan yleistä kiertymistä mielivaltaisen kiertoakselin suhteen.

Koordinaatiston pisteen (x, y, z) kiertäminen x-akselin suhteen kulman φ verran pisteeksi (x', y', z') voidaan ratkaista yhtälöryhmällä, joka on esitetty alla matriisilaskennan keinoin.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Kerroinmatriisi yllä on kiertomatriisi x-akselin suhteen. Vastaavasti kiertomatriisi y-akselin suhteen

$$\begin{pmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{pmatrix}$$

ja kiertomatriisi z-akselin suhteen

$$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Edelliset kuvaukset kiertävät pistettä siten, että kiertosuunta on vastapäivään katsottuna origosta positiivisen kiertoakselin suuntaan. Matemaattisesti tarkastellen koordinaatiston pisteen kierrot koordinaattiakseleiden suhteen ovat lineaarikuvauksia $f: \mathbb{R}^3 \to \mathbb{R}^3$, jotka voidaan toteuttaa matriisien laskutoimituksilla. Lineaarikuvaukselle f on aina voimassa ominaisuudet:

$$f(\bar{\boldsymbol{a}} + \bar{\boldsymbol{b}}) = f(\bar{\boldsymbol{a}}) + f(\bar{\boldsymbol{b}})$$

$$f(c\,\bar{\boldsymbol{a}}) = c\,f(\bar{\boldsymbol{a}})$$

Kiertomatriiseja voidaan soveltaa myös symboliseen esitykseen eli vektorifunktioon $\bar{\boldsymbol{r}}(t) = (x(t), y(t), z(t))$. Esimerkiksi origokeskinen xy-tason ympyrärata $\bar{\boldsymbol{r}}(t) = R(cos(t), sin(t), 0)$, joka on kiertynyt x-akselin suhteen kulman φ verran, saadaan matriisitulosta

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) \\ 0 & -\sin(\varphi) & \cos(\varphi) \end{pmatrix} \begin{pmatrix} R\cos(t) \\ R\sin(t) \\ 0 \end{pmatrix} = \begin{pmatrix} R\cos(t) \\ R\cos(\varphi)\sin(t) \\ -R\sin(t)\sin(\varphi) \end{pmatrix}$$

Kierto voidaan tehdä myös jonkin muun pisteen kuin origon kautta kulkevan kiertoakselin suhteen. Se saadaan peräkkäisillä operaatioilla

1. origon siirto→ 2. kierto origon suhteen → 3. siirto takaisin

Käsitellään aluksi koordinaattiakselin suuntaisia kiertoakseleita. Esimerkiksi jos kappale kiertää xy-tason suuntaisesti ympyrärataa pisteen (x_0, y_0, z_0) ympäri etäisyydellä R, on kappaleen kiertoradan vektoriesitys $(x_0 + R\,cos(t), y_0 + R\,sin(t), z_0)$ (kpl 4.A). Jos nyt halutaan kallistaa kyseistä kiertorataa niin, että kiertoakselina on keskipisteen (x_0, y_0, z_0) kautta kulkeva x-akselin suuntainen akseli ja kiertokulma on φ, saadaan uudeksi kiertoradaksi symbolinen esitys em. kolmen operaation avulla seuraavasti:

$$\underbrace{\overbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) \\ 0 & -\sin(\varphi) & \cos(\varphi) \end{pmatrix} \left(\overbrace{\begin{pmatrix} R\cos(t) + x_0 \\ R\sin(t) + y_0 \\ z_0 \end{pmatrix} - \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}}^{1.} \right)}^{2.} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}}_{3.} = \begin{pmatrix} x_0 + R\cos(t) \\ y_0 + R\cos(\varphi)\sin(t) \\ z_0 - R\sin(t)\sin(\varphi) \end{pmatrix}$$

Tuloksesta voidaan kuitenkin nähdä, että kierto *tässä erityistapauksessa, missä kiertoakseli kulkee rataympyrän keskipisteen kautta*, sievenee hyvin yksinkertaiseksi kuvaukseksi *'siirto + kierto origon suhteen'*. Yllä olevasta matriisiesityksestä saadaan helposti sievennettyä tämä tulos.

*VPython*issa ei ole sisäänrakennettua matriisien algebraa, joten kiertomatriisien käyttö symbolisten esitysten muodostamiseen tulee perustua siihen, että **matriisin kertomisen tuloksena syntyvä symbolinen ratakäyrän esitys lasketaan valmiiksi muilla matemaattisilla laskentaohjelmilla**.

Yleisen kierron vektorifunktio (**)

Yleisessä kierrossa, missä kiertoakseli ei välttämättä ole koordinaattiakseleiden suuntainen, on tiedossa kiertokulman suuruuden lisäksi jokin kiertoakseli sekä koordinaatiston piste, jonka kautta kiertoakseli kulkee. Myös tällainen kierto voidaan toteuttaa yhden kiertomatriisin avulla. Mikäli haluaa konstruoida ja implementoida itse mielivaltaisen pyörimisakselin suhteen toimivan ja myös symbolisen tuloksen palauttavan kiertofunktion, on liitteessä 1 selostettu tällaisen funktion konstruointi kvaterniotekniikkaan perustuen. Menetelmän laajempi selostus *Mathematica*n symbolisen ohjelmoinnin keinoin toteutettuna löytyy julkaisusta
(Peisa 2005, http://www.kpeisa.fi/work/Quaternions%20AITC%202005.pdf).

4.3. Objektien kiertäminen VPythonin funktiolla rotate()

Moduliin VPython on toteutettu sisäänrakennettu oma kiertofunktio `rotate()`, joka on määritelty erikseen vektoreihin kohdistettuna

```
newVec=rotate(vector,angle=theta,axis=vector(a,b,c))
```

ja graafisiin objekteihin kohdistuvana (Funktion syntaksi ja semantiikka on esitetty hyvin *VPython*in dokumentaatiossa).

```
obj.rotate(angle=theta, axis=vector(a,b,c))
obj.rotate(angle=theta, axis=vector(a,b,c),origin=vector(a,0,0))
```

Funktio `rotate()` toteuttaa siten numeerisesti laskemalla saman, mitä edellä mainittu liitteen 1 funktio, joka tarvittaessa tuottaa myös kierretyn ratakäyrän symbolisen esityksen.

Siihen, kumpi toteutustapa on parempi, sisäänrakennettu funktio vai kiertoesityksen symbolinen muodostaminen, ei ole yksiselitteistä vastausta ja usein molemmat tavat onnistuvat. Keskeistä on ymmärtää, minkä keskipisteen ja minkä kiertoakselin suhteen kierto tapahtuu.

Tarkastellaan esimerkkinä objektia, joka kiertää aluksi origokeskistä xy-tason ympyrärataa $\bar{\boldsymbol{r}}(t) = R(cos(t), sin(t), 0)$. Jos nyt halutaan kiertää tätä ympyrärataa x-akselin suhteen kulman φ verran, saadaan edellä johdettu kierto aikaan funktiolla `rotate()` kiertämällä sen paikkavektoria seuraavasti:

```
newPos=R*vector(cos(t),sin(t),0)
obj.pos=rotate(newPos,φ,vector(1,0,0))
t += dt
```

Tai edellinen esitys yhdistettynä

```
obj.pos=rotate(R*vector(cos(t),sin(t),0),φ,vector(1,0,0))
t +=dt
```

Tällöin olipa kappale missä tahansa kohtaa `t` alkuperäistä ympyrärataa, saadaan uusi paikka aina kiertämällä kyseistä paikkavektoria x-akselin suhteen kulman φ verran.

Objektia voidaan kiertää ”kallistettuja kiertoympyröitä pitkin” myös stokastisella periaatteella käyttämällä sisäänrakennettua rotate-funktiota seuraavasti. Kierretään ensin kappale uuteen kierrettyyn paikkaan sekä lasketaan uusi kiertoakseli samalla tavalla. Lasketaan sitten, kuinka suuri kulmamuutos kiertoradalla vastaa ajanmuutosta `dt`. Ympyräliikkeellä, jonka kiertoaika on T, suuruus lasketaan seuraavasti (kts. kpl 4.1)

```
dΘ=2*pi/T*dt.
```

Edellisessä esimerkissä objektia ja kiertoakselia (z-akseli) kierretään ensin x-akselin suhteen kulman φ verran. Sen jälkeen ohjelmasilmukassa kierretään objektia saman keskipisteen (origo) mutta uuden kiertoakselin suhteen kulman `dΘ` verran .

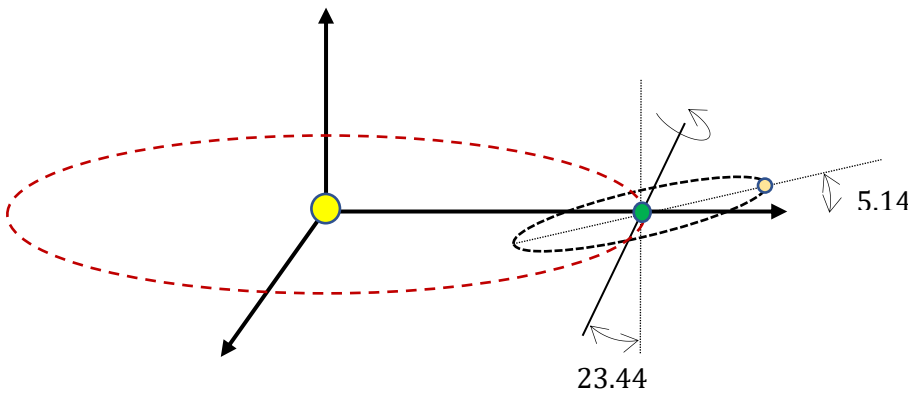
```
obj.pos=rotate(obj.pos,φ,vector(1,0,0)) # objektille uusi paikka
rotAxis= rotate(vector(0,0,1),φ,vector(1,0,0)) # uusi kiertoakseli

… Loopissa …
obj.rotate(angle=dΘ,axis=rotAxis) # kiertyminen ajassa dt
t +=dt
```

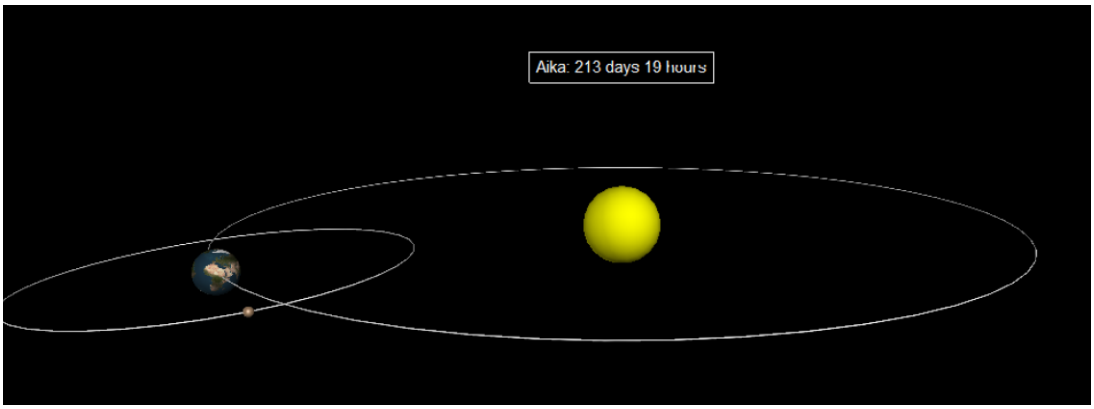
Deterministinen ympyräliike sopii esimerkiksi aurinko-maa-kuu -systeemin karkeaan mallintamiseen. Johdetaan symbolinen kuun ratakäyräesitys, kun kuun kiertorata on kallistunut maan kiertoradan muodostamaan tasoon nähden inklinaatiokulman $\varphi = 5.145°$ verran.

Koordinaatiston origo sijoitetaan auringon keskipisteeseen. Alkutilanteessa maa sijoitetaan x-akselille (tai vaihtoehtoisesti z-akselille). Sen jälkeen muodostetaan kuun ympyrärata, kun se kiertää maata ensin maan ratatasossa. Tämän jälkeen kuun rataa kierretään maan keskipisteen eli kuun radan keskipisteen kautta kulkevan z-akselin (tai vaihtoehtoisesti x-akselin) suuntaisen kiertoakselin suhteen inklinaatiokulman verran. On huomioitavaa, että kuun ratatason suunta samoin kuin maan ja kuun napa-akselitkin säilyvät kiertäessään auringon ympäri. Tosiasiassahan ne kuitenkin muuttuvat hiljalleen määrätyin pitkien ajan jaksoin.

Maapallon pyöriminen akselinsa ympäri voidaan toteuttaa deterministiseen simulaatioon edellä kuvatun stokastisen periaatteen ja sisäänrakennetun rotate-funktion avulla huomioimalla, että kiertoakseli on maan kallistettu (23,44°) napa-akseli, joka kulkee maan keskipisteen kautta.

Kuun elliptisen todellisen liikeradan mallintamiseen palataan myöhemmin stokastisten kiertoliikkeiden simulaatioissa.





Kuva 5. Aurinko-maa-kuu systeemi. Tee harjoitus 8.

5. Liikemäärän muutos ja impulssi

Liikemäärä $\overline{\boldsymbol{p}} = m\,\overline{\boldsymbol{v}}$ ja liikemäärämomentti $\overline{\boldsymbol{L}}$ (myös impulssimomentti, kulmaliikemäärä tai pyörimismäärä) ovat kaksi simulaatioissa laajasti käytettyä käsitettä, joilla luonnossa on voimassa lähes kaiken määräävät säilymislait. Liikemäärämomenttia käsitellään myöhemmin jäykän kappaleen dynamiikan yhteydessä. Seuraavassa esitetään peruste stokastisen simulaation ratkaisumalliin, joka perustuu liikemäärän ja impulssin käsitteisiin.

Liikemäärän ja impulssin käsitteisiin päästään dynamiikan peruslaista kirjoittamalla se uuteen muotoon (oletetaan massa muuttumattomaksi).

$$\overline{\boldsymbol{F}} = m\,\overline{\boldsymbol{a}} = m\,\frac{d\boldsymbol{v}}{dt} \Leftrightarrow\ m\,d\boldsymbol{v} = \overline{\boldsymbol{F}}\,dt$$

Jälkimmäinen esitys kertoo sen, että voiman lyhyessä ajassa dt kappaleelle antama impulssi $\overline{\boldsymbol{F}}\,dt$ aiheuttaa kappaleen liikemäärän muutoksen $d\overline{\boldsymbol{p}} =\ m\,d\boldsymbol{v}$. Edelleen voidaan jälkimmäinen muoto kirjoittaa muodossa

$$m\,(\overline{\boldsymbol{v}}_{\mathbf{2}} - \overline{\boldsymbol{v}}_{\mathbf{1}}) = \overline{\boldsymbol{F}}\,dt \rightarrow\ \ \overline{\boldsymbol{v}}_{\mathbf{2}} = \frac{m\,\overline{\boldsymbol{v}}_{\mathbf{1}} + \overline{\boldsymbol{F}}\,dt}{m}$$

Jälkimmäinen esitys antaa **impulssiin ja liikemäärän muutokseen** perustuvan algoritmin simulaatiolle.

1. Laske kappaleeseen vaikuttava kokonaisvoima $\sum \overline{\boldsymbol{F}}$
2. Päivitä kappaleen liikemäärä $\overline{\boldsymbol{p}} \mathrel{+}= \sum \overline{\boldsymbol{F}}\,dt$
3. Laske kappaleen uusi nopeus $\overline{\boldsymbol{v}} = \frac{\overline{\boldsymbol{p}}}{m}$
4. Päivitä paikka $\overline{\boldsymbol{pos}} \mathrel{+}= \overline{\boldsymbol{v}}\,dt$

Algoritmi poikkeaa vain hieman edellä esitetystä dynamiikan peruslakiin pohjautuvasta muodosta. Tämän algoritmin soveltamisessa liikemäärää pidetään kappaleobjektin yhtenä päivitettävänä attribuuttina. Impulssiin perustuvassa mallintamisessa kiihtyvyyksiä ei tarvitse kappaleille erikseen laskea eikä päivittää. **Edellä esitettyjä simulaation pääalgoritmeja voi varioida monellakin tavalla**.

Impulssien laskeminen on keskeinen periaate myöhemmin erilaisiin törmäyksiin liittyvien simulaatioiden ratkaisuissa.

6. Kiertoliikkeiden stokastisia simulaatioita

6.1. Gravitaatiovoiman alainen kiertoliike ja liikemäärän säilymislaki

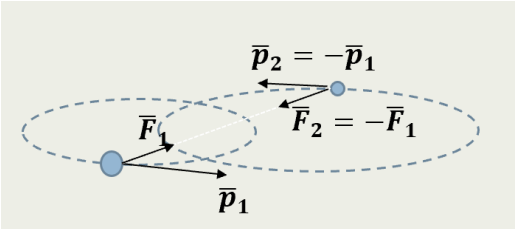
Stokastisissa simulaatioissa meidän tulee tietää kappaleiden alkutilanne tarkasti, jonka jälkeen approksimoimme niiden liiketilaa pienin askelin sen perusteella, mitä tiedämme niihin kohdistuvista voimista.

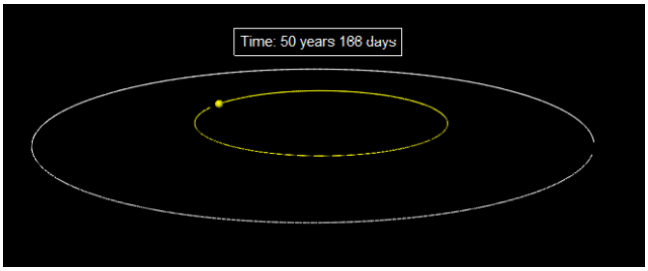
Kun mallinnetaan taivaankappaleiden liikettä gravitaatiovoimien muodostamissa systeemeissä, itse voimien vaikutuksen mallintaminen on usein helpoin tehtävä. Pallomaiset taivaankappaleet voidaan ajatella pistemäisiksi ja ne vaikuttavat toisiinsa toisistaan riippumattomilla keskinäisillä gravitaatiovoimilla, joiden suunnat määräytyvät näiden keskipisteiden välisinä ja suuruudet lasketaan Newtonin gravitaatiolailla

$$\overline{\boldsymbol{F}}_G = G\frac{m_1 m_2}{r^2},$$

missä $G \cong 6.67384e-11\frac{\mathrm{m}^3}{\mathrm{kg\,s}^2}$, m_1 ja m_2 kappaleiden massat ja r kappaleiden (keskipisteiden) välinen etäisyys. Gravitaatiovoimavektori lasketaan simulaatiossa samalla periaatteella kuin ilmanvastuksen tapauksessa, suuruus saadaan yo. fysiikan kaavasta ja suunta kappaleiden keskipisteiden välisestä yksikkövektorista.

Edellä oleva liikemäärän päivitykseen perustuva algoritmi on erittäin käytännöllinen tilanteissa, joissa tarkasteltava systeemi koostuu esimerkiksi kahdesta gravitaatiovoiman alaisesta liikkuvasta kappaleesta, joiden kokonaisliikemäärä on nolla eli kokonaissysteemi on levossa. **Liikemäärää päivitettäessä riittää laskea vain toisen kappaleen liikemäärä, sillä joka hetki toisen kappaleen liikemäärä on liikemäärän säilymislain nojalla edellisen vastavektori**. Tällä periaatteella voidaan helposti simuloida esimerkiksi kahden samaa massakeskipistettä kiertävien taivaankappaleiden liikkeet gravitaatiovoiman vaikutuksesta, kunhan alkuasetelma saadaan määritettyä oikein.





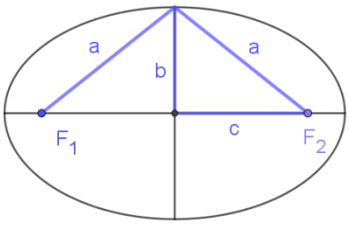
Kuva 6. Kaksoistähtisimulaatio

Tee harjoitus 9

6.2. Maa-kuu systeemi

Gravitaatiokentän alaisissa stokastisissa simulaatioissa alkupaikkojen ja -nopeuksien määrittäminen tai arvioiminen on ratkaisevaa. Siinä joudutaan turvautumaan mittaamalla ja laskemalla saatuun taivaanmekaniikan tutkittuun tietoon ja/tai ratakäyrien geometriaan. Itse liikettä ohjaa vain gravitaatiovoima.

Tarkastellaan esimerkkinä maan ja kuun systeemiä ja sen alkutilanteen määrittelyä.



Kuun kiertorata on ellipsi. Palautetaan mieliin ellipsin parametrit

- a isoakselin puolikas
- b pikkuakselin puolikas
- c polttopisteiden välin puolikas (polttoväli)
- e eksentrisyys = polttopisteiden välimatkan suhde isoakselin pituuteen = $\frac{c}{a}$

Ellipsin kehän pisteet muodostavat sellaisen joukon, joiden etäisyyksien summa polttopisteistä on vakio = $2a$. Edellisillä parametreilla on voimassa

$$b^2 + c^2 = a^2$$

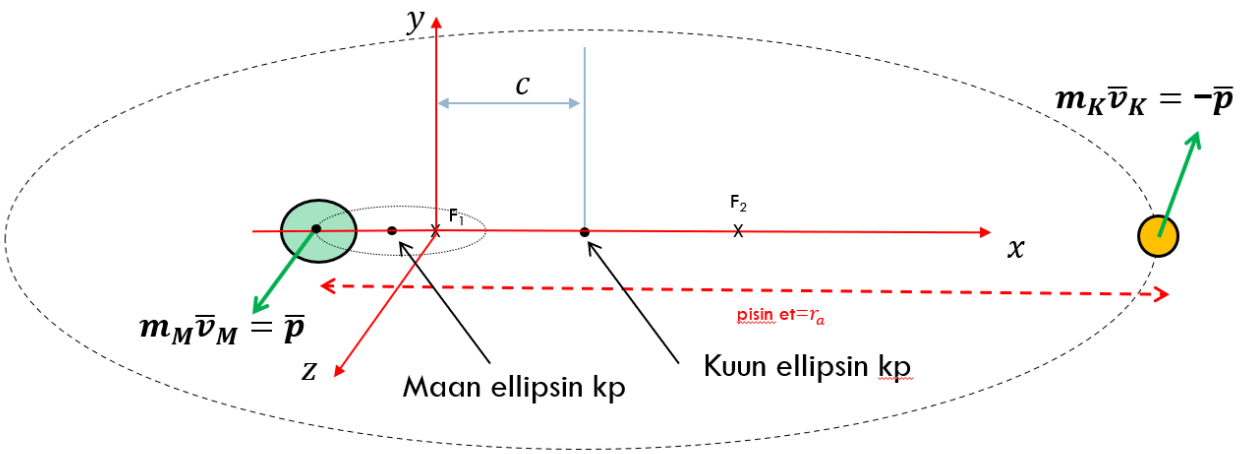
$$e = \frac{c}{a} = \sqrt{\frac{a^2-b^2}{a^2}}$$

Taivaanmekaniikalla on tiedossa maan lyhin ja pisin etäisyys kuusta sekä kuun radan eksentrisyys e. Merkitään lyhin etäisyys (perigeum) on r_p ja pisin etäisyys (apogeum) on r_a. Jos maan ja kuun yhteinen massakeskipiste muodostaa origon, saadaan pisimmästä etäisyydestä laskettua maan ja kuun alkupaikat ao. kuvan mukaisesti.

Maan ja kuun alkukoordinaattien x_M ja x_K laskeminen

Massakeskipiste: $\frac{\sum_i m_i x_i}{\sum_i m_i} = \frac{m_M x_M + m_K x_K}{m_M + m_K} = 0$

Pisin etäisyys: $-x_M + x_K = r_a$



Massakeskipisteestä saadaan $x_K = -\frac{m_M}{m_K} x_M$. Kun se sijoitetaan pisimmän etäisyyden lausekkeeseen, saadaan kuun alkupaikalle $x_K = \frac{m_M}{m_M + m_K} r_a$ ja vastaavasti maan alkupaikalle

$x_M = -\frac{m_K}{m_M + m_K} r_a$.

Itse ellipsiratojen parametreja ei tarvitse ratkaista, koska simulaatio toimii stokastisesti. Taivaanmekaniikan mukaisesti keveämmät massakappaleet kiertävät suurempaa massaa pitkin ellipsin rataa siten, että suurempi massa sijaitsee radan toisessa polttopisteessä. Tämä pitää paikkaansa suhteellisen tarkasti vain silloin, kun massojen ero on hyvin suuri ja raskaamman kappaleen liike tuskin havaittava. Esimerkiksi kuun vaikutus maan liikkeeseen on hyvinkin havaittavaa. Oikein ilmaistuna maa ja kuu kiertävän yhteistä massakeskipistettä, joka samalla on molempien elliptisten kiertoratojen yhteinen polttopiste. Maan ja kuun kiertoratojen toiset polttopisteet tulevat päinvastaisiin suuntiin yhteisestä polttopisteestä. Edellä mainittujen kaavojen avulla voidaan nyt kiinnittää maan ja kuun alkupaikat koordinaatistossa aluksi ilman kuun radan inklinaatiokulman huomioimista. Kulma huomioidaan sitten alkupaikan kiertona.

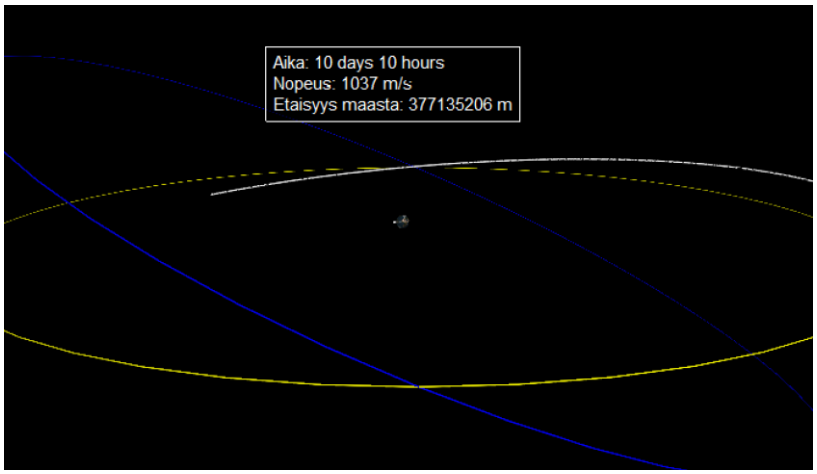
Alkupaikkojen tarkan laskemisen lisäksi on arvioitava maan ja kuun alkunopeudet oikein. Kun ma-kuu-systeemiä tarkastellaan erillisenä, on kokonaisliikemäärä nolla. Maan ja kuun alkunopeuksien suuruuksien suhde saadaan ratkaistua liikemäärän yhtälöstä, kun massat tunnetaan.

$$m_M v_M - m_K v_K = 0$$

Kuun liikkeen keskimääräinen ratanopeus maan suhteen on 1.022 km/s ja kuun kierrosaika (sideraalinen) on noin 27.322 vrk. Inklinaatiokulmalla ei ole vaikutusta alkunopeuden suuntaan, kun alkupaikka sijoitetaan xy-tasolle. Tällöin nopeusvektori osoittaa aina z-akselin suuntaisesti.

Kun simulaatio on toiminnassa, voidaan edellisten tietojen valossa arvioida kuun nopeus aloitustilanteessa (ja laskea vastaava maan nopeus) niin, että kuun kiertoajaksi tulee oikea. Näin saadaan jo todellisuutta suhteellisen hyvin vastaava maa-kuu-simulaatio. Todellisuudessa kuun rata ei ole kovin vakio ja kuun liike sisältää kaksikin eri prekessioliikettä. Simulaatio toimii oikein, mikäli maa-kuu-systeemi ei vaella avaruudessa eli kokonaisliikemäärä on koko ajan nolla.

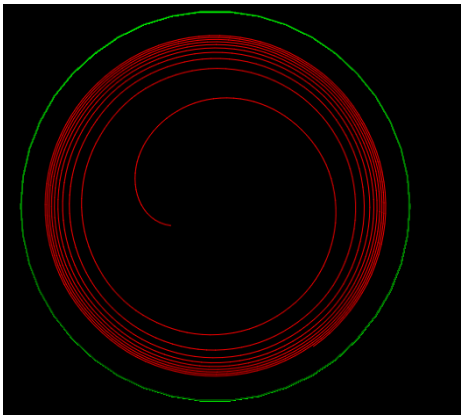
Kun kuulle kokeilee eri alkunopeuden suuruuksia, huomaa, miten kuun liikerata ja kiertoaika vaihtelevat hieman erilaisten ellipsien myötä. *VPython*issa voi tarkastella myös maan liikkumista jäljen avulla, kun laittaa materiaalin riittävän läpinäkyväksi (attribuutti `opacity=0.3`). Simulaatiossa voi tulostaa tekstikehykseen virtuaaliajan lisäksi esimerkiksi kuun ja maan välisen etäisyyden. Tällöin voi verrata myös pisimmän ja lyhimmän etäisyyden arvoja oikeisiin.



Kuva 7. Maa-kuu -systeemi stokastisella simulaatiolla.

Tee harjoitus 10.

6.3. Satelliittien kiertoradat ja Runge-Kutta-Nyström menetelmä (**)



Satelliittien lentoratoja simuloitaessa liikutaan jo toisinaan maan ilmakehän yläosissa, joissa tulee mallintaa myös väliaineen vastusta saadakseen reaalista käyttäytymistä. Se onkin hyvin mielenkiintoinen ilmiö, miten satelliitin nopeus kehittyy ja mitä tapahtuu, kun ilmakehä jarruttaa sitä heikolla voimalla. Satelliitin lentoradan kuvaaminen 50 km – 200 km korkeuksilla havainnollistetaan parhaiten siten, että maapallo jätetään kokonaan kuvaamatta tai se kutistetaan pisteeksi origoon, ja kuvataan satelliitin paikka pelkästään kiertoliikkeenä origon ympäri. Tämä voidaan toteuttaa asettamalla satelliitti kiertämään koordinaattiakseleiden muodostamassa tasossa. Kullakin ajanhetkellä ratkaistaan satelliitin kiertokulma $\varphi \in]-\pi, \pi]$ Pythonin funktiolla `math.atan2(y,x)`, joka palauttaa radiaaneissa kiertokulman välille $]-\pi, \pi]$ riippuen koordinaattien x ja y etumerkeistä. Satelliitin paikka kuvataan tuon kiertokulman ja satelliitin korkeuden avulla).

Ilmakehän vastusvoiman mallintamiseen voi käyttää edellä esitettyä ilmanvastuskaavaa muunnetuin kertoimin, mutta sen paikkansa pitävyydestä niin erikoisissa olosuhteissa kuin termosfäärissä yli 50 km:n korkeuksilla ei ole mitään varmaa tietoa. Kuitenkin erittäin alhaisessa paineessa ja tiheydessä oleva aine käyttäytyy noilla korkeuksilla vielä kaasumaisesti. Vastuskaavan kertoimia voi muunnella ja yrittää saada vastaavuutta tunnettuihin tapauksiin kuten esimerkiksi kiinalaisen satelliitin Tiangong-1 maahan syöksyyn 2018, josta löytyy tietoa verkosta mm. sivuilla

https://www.heavens-above.com/OrbitHeight.aspx?satid=37820.

Ilmantiheyden mallintaminen on myös melko ongelmallista. Voisi kuvitella, että regressiosovituksella (tarvitaan matemaattinen ohjelma) löytyy sopiva polynomi tai eksponenttifunktio, joka sovitettuna alla olevan ilmantiheystaulukon numeerisiin arvoihin mallintaisi tiheyttä riittävän hyvin.

Lähde: http://www.taulukot.com/index.php?search_id=mekaniikka_termodynamiikka&lng=fi

h (km)	0	1	2	3	4	5	10	15	20	30
ρ (kg/m^3)	1.22	1.11	1.01	0.91	0.82	0.73	0.41	0.19	0.088	0.018

h (km)	40	50	100	200	300	400	500	600	700
ρ (kg/m^3)	3.9E-03	1.0E-03	4.9E-07	3.3E-10	3.5E-11	6.4E-12	1.5E-12	4E-13	1.5E-13

Näin ei kuitenkaan tapahdu ihan helposti. Polynomilla ilmenee liian suuria kaareutumisia yläkorkeuksilla ja eksponenttifunktio sukeltaa liian kaukaa pisteiden ohi joko ala- tai sitten yläkorkeuksilla. Täytyykin laskea ensin logaritmit (ln) tiheyden arvoista ja sovittaa logaritmisiin arvoihin polynomi $p(h)$ (asteluku 3 tai 5, voi kokeilla). Sen jälkeen voi kuvata tiheyden funktiona $\rho(h) = e^{p(h)}$. Tällä tavalla pääsee ainakin 200 km:n korkeuteen niin, että data ja funktio täsmäävät erittäin hyvin. Pitää muistaa, että mallilla ei kärsi kuitenkaan ekstrapoloida, koska polynomista johtuen funktion käyttäytyminen muuttuu oleellisesti sovitusalueen ulkopuolella.

Koska väliaineen vastusvoima termosfäärissä on erittäin heikkoa ja gravitaatiovoiman vaikutuksen virheet pyrkivät kumuloitumaan jokaisella kierroksella, tarvitaan tässä Eulerin menetelmää

tarkempaa approksimaatiota, jona käytetään Runge-Kutta-Nyström -menetelmää. Sen käyttöönotto on selostettu liitteessä 2.

Joskus simulaatiosta haluaa rekisteröidä tietoa tiedostoon myöhempää käsittelyä varten jollakin toisella ohjemistolla. Python-koodissa ascii-muotoisen tekstitiedoston (kooditiedoston kansioon) saa kirjoitettua helposti oheisilla komennoilla.

```
file=open("data.txt","w")
while (1):
  ...
  file.write("%i,%.1f")%(intVar,floatVar)
file.close()
```

Tiedostoon kirjoitetaan jokaisella silmukan kierroksella, mikä voi olla aivan liian tiheään. Jos aika-askeleen `dt` arvo on kokonaisluku, voidaan jakojäännösoperaattorilla % helposti tulostaa vain tietyillä ajan `t` kokonaislukuarvoilla esimerkiksi minuutin välein `if(t%60)==0`. Kun `dt`:n arvo on desimaaliosa sekunnista (0.01, 0.02 tai 0.05) saa tulostukset vain tietyillä ajan kokonaislukuarvoilla esimerkiksi oheisella `if`-lauseella. Jos `dt` annettu yhden desimaalin tarkkuudella, käytetään `round()` funktiolle yhden desimaalin tarkkuutta.

```
if(int(t//1)%60==0 and round(t%1,2)==0):
  file.write(...
```

Tee harjoitus 11.

Satelliitin simulaatioon liittyvä teoreettinen tarkastelu löytyy esim. oheisella www-sivulla

https://physicsfromplanetearth.wordpress.com/2017/02/13/work-energy-and-the-satellite-drag-paradox/?fbclid=IwAR1dTKTA69_nNmqRKXqaE7aqMfip45maJYL89b5DGjRYLVvPsGYclXS-XEY

7. Jäykän kappaleen dynamiikka

7.1. Kulmasuureet ja ratasuureet pyörimisliikkeiden kinematiikassa ja dynamiikassa

Kun simulaatioissa kontrolloidaan yleisiä 3-ulotteisia kiertoliikkeitä tai kappaleiden pyörimisliikkeitä, on välttämätöntä käsitellä pyörimiseen liittyviä suureita vektorisuureina. Kinematiikan ja dynamiikan teorian rakenne kiertoliikkeissä jonkin akselin suhteen on hyvin analoginen etenemisliikkeeseen nähden. Meillä on selkeät vastaavuudet etenemis- ja kiertoliikkeiden suureilla.

skalaarisuureet

m massa ←→	hitausmomentti I

vektorisuureet kinematiikassa

$\bar{\boldsymbol{r}}$ paikka ←→	kulma $\bar{\boldsymbol{\varphi}}$
$\bar{\boldsymbol{v}}$ nopeus ←→	kulmanopeus $\bar{\boldsymbol{\omega}}$
$\bar{\boldsymbol{a}}$ kiihtyvyys ←→	kulmakiihtyvyys $\bar{\boldsymbol{\alpha}}$

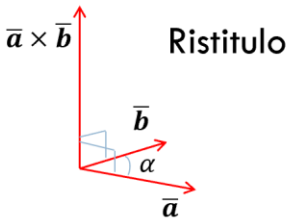
vektorisuureet dynamiikassa

$\bar{\boldsymbol{F}}$ voima ←→	voiman (vääntö)momentti $\bar{\boldsymbol{\tau}}$
$\bar{\boldsymbol{p}}$ liikemäärä ←→	liikemäärämomentti $\bar{\boldsymbol{L}}$
$\bar{\boldsymbol{F}}\,dt$ voiman impulssi ←→	(vääntö)momentin impulssi $\bar{\boldsymbol{\tau}}\,dt$

Tässä yhteydessä rajoitutaan käsittelemään ns. *puhdasta pyörimistä*, jossa kappaleen jokainen massapiste kiertää ympyrärataa saman pyörimisakselin suhteen. Itse pyörimisakseli voi kulkea minkä tahansa avaruuden pisteen kautta. Jäykän kappaleen pyöriminen sekä kiertäminen jonkin akselin ympäri noudattavat tällöin samaa teoriaa, joka johdetaan massapisteen ympyräliikkeestä.

Hitausmomentti I (moment of inertia) on suure, joka huomioi jäykän kappaleen pyörimisessä massan lisäksi sen etäisyyden tai jakautumisen pyörimisakseliin nähden. Pistemäiseksi laskettavan kappaleen hitausmomentti $I = m\,r^2$, missä r on massapisteen etäisyys ympyräliikkeen kiertoakselista. Erilaisia kappaleiden hitausmomentteja eri tilanteissa voi katsoa taulukoista tai laskea itse mm. *Steinerin säännöllä* (teoriaa ei tässä esitellä).

Pyörimisliikkeiden vektorisuureiden hahmottaminen vaatii mm. matematiikasta vektorilaskennan ristitulon käsitteen ymmärtämistä.



Ristitulo

$$\bar{a} \times \bar{b} = \begin{vmatrix} \bar{\boldsymbol{\imath}} & \bar{\boldsymbol{\jmath}} & \bar{\boldsymbol{k}} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

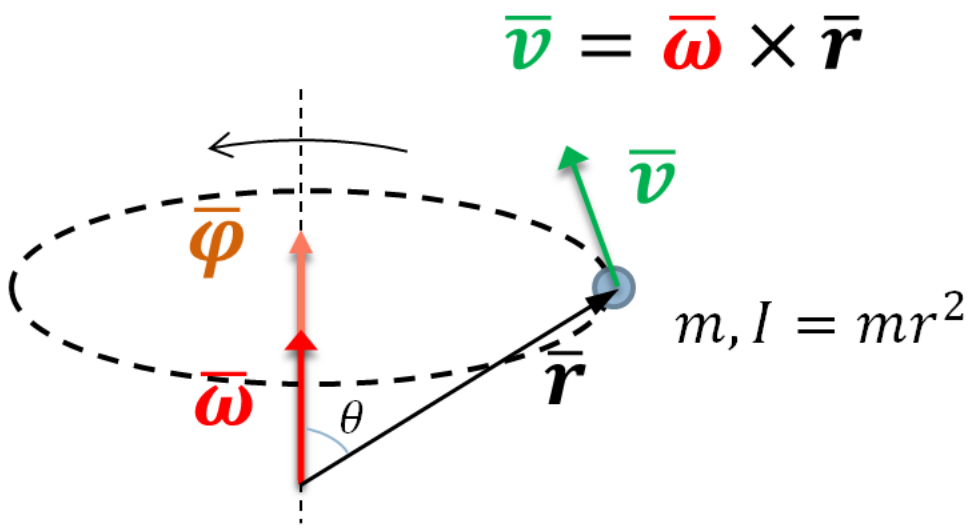
Geometrinen tulkinta $|\bar{a} \times \bar{b}| = |\bar{a}||\bar{b}|\sin(\alpha)$, $\alpha = \measuredangle(\bar{a}, \bar{b})$

Yleensä ristitulon laskeminen on implementoitu vektoreita käyttäviin ohjelmistoympäristöihin valmiiksi kuten myös *VPython*issa on laita.

Huomionarvoista on, että ristitulovektori on aina kohtisuorassa molempia ristitulon tekijävektoreita vastaan. Lisäksi jos tekijöiden välinen kulma on suora (90°), niin $|\bar{a} \times \bar{b}| = |\bar{a}||\bar{b}|$ (kts. geometrinen tulkinta). On myös huomioitava, että ristitulo määrittää aina ristitulovektorin suunnan ns. oikean käden säännön mukaisesti (peukalo ($\bar{\boldsymbol{a}} \times \bar{\boldsymbol{b}}$), etusormi ($\bar{\boldsymbol{a}}$), keskisormi ($\bar{\boldsymbol{b}}$)).

Alla olevat kulmasuureiden ja ratasuureiden matemaattiset määrittelyt ja niiden väliset yhteydet luodaan matematiikan alueella, jota kutsutaan **differentiaaligeometriaksi**. Matemaattiset määrittelyt tuottavat fysikaalisen kiertoliikkeen, jossa massapiste kiertyy origon kautta kulkevan kiertoakselin suhteen. Kaikki vektorikulmasuureet ovat kiertoakselin suuntaisia. Kierron suunta on myötäpäivään, kun sitä tarkastellaan vektorikulmasuureiden suuntaisesti.

Seuraavassa oletetaan, että vektori $\bar{\boldsymbol{r}}$ on massapisteen m paikkavektori, ja kiertoakseli kulkee origon kautta. Massapisteen m (rata)nopeuden ja kulmanopeuden välinen yhteys saadaan ristitulolla. Nopeusvektori on siten kohtisuorassa sekä kulmanopeutta että paikkavektoria vastaan.



Ristitulon perusteella ratanopeuden suuruus saadaan kaavasta

$$v = |\bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{r}}| = \omega r \sin(\theta),$$

missä $\theta = \sphericalangle(\bar{\boldsymbol{\omega}}, \bar{\boldsymbol{r}})$ ja $r\, sin(\theta)$ on massapisteen m etäisyys pyörimisakselista. Kun tämä etäisyys pysyy vakiona, ristitulo antaa saman nopeusvektorin kulmasta θ riippumatta. Tämän voi perustella myös ristitulon osittelulain perusteella.

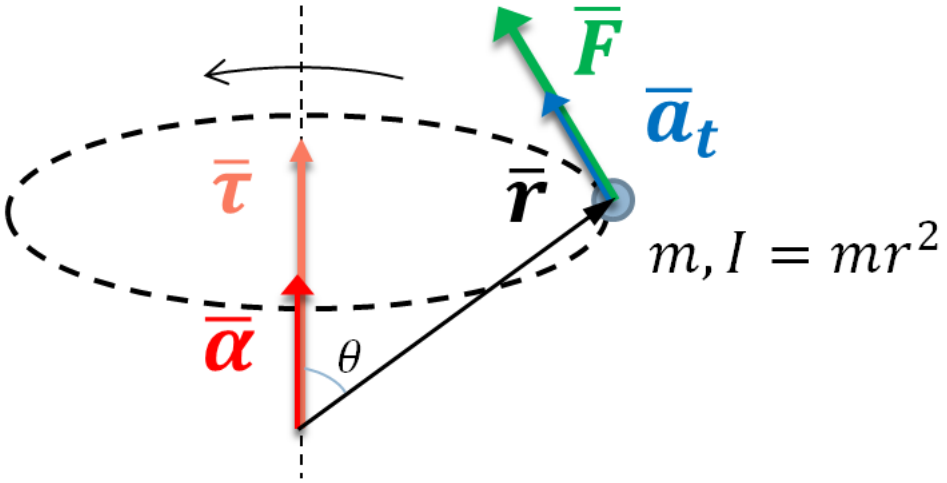
$$\bar{\boldsymbol{\omega}} \times (\bar{\boldsymbol{r}} + s\, \bar{\boldsymbol{\omega}}) = \bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{r}} + s\, \bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{\omega}} = \bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{r}}$$

Kun paikkavektori $\bar{\boldsymbol{r}}$ on kohtisuorassa pyörimisakseliin nähden, saadaan skalaariyhtälö $v = \omega r$.

Oikean käden säännön mukaan nopeuksien suunnat saadaan järjestyksestä peukalo $\parallel \bar{\boldsymbol{v}}$, etusormi $\parallel \bar{\boldsymbol{\omega}}$ ja keskisormi $\parallel \bar{\boldsymbol{r}}$. On syytä huomioida, että ristituloyhtälö $\bar{\boldsymbol{v}} = \bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{r}}$ ei määritä yksikäsitteisesti ristitulon tekijävektoreita.

Vääntömomentti $\bar{\boldsymbol{\tau}}$ (torque) aiheuttaa massalle kulmakiihtyvyyttä (analogisesti etenevän liikkeen tilanteeseen, jossa voima aiheuttaa kiihtyvyyttä).

$$\bar{\tau} = \bar{r} \times \bar{F} = I\,\bar{\alpha}$$

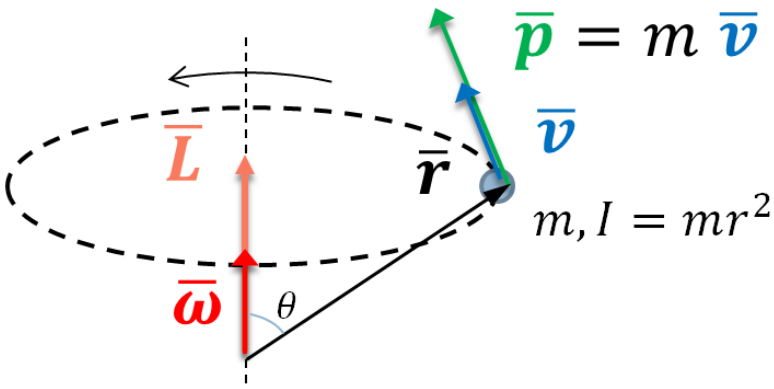


Yleinen pyörimisen liikeyhtälö $\bar{\boldsymbol{\tau}} = I\,\bar{\boldsymbol{\alpha}}$ on myös analoginen etenemisliikkeen vastaavan yhtälön $\bar{\boldsymbol{F}} = m\,\bar{\boldsymbol{a}}$ kanssa. Kun paikkavektori $\bar{\boldsymbol{r}}$ on kohtisuorassa pyörimisakseliin nähden, merkitään usein skalaarista momenttia kirjaimella M (moment, momentti) ja merkitään $M = F\,r = I\,\alpha$. Tällöin kulmakiihtyvyyttä α vastaava ratasuure massalle m on ***tangentiaalinen kiihtyvyys***, jonka suuruus $a_t = \alpha\,r$.

Liikemäärää vastaavaan pyörimissuure on liikemäärämomentti $\bar{\boldsymbol{L}}$ (angular momentum). Myös siihen liittyy yleinen säilymislaki, minkä vuoksi käsitteestä käytetään suomenkielessä usein myös nimitystä *pyörimismäärä*.

$$\bar{L} = \bar{r} \times \bar{p}$$

$$\bar{p} = m\,\bar{v} = m\,(\bar{\omega} \times \bar{r})$$



Huom! Puhtaassa pyörimisessä liikemäärämomentti $\bar{\boldsymbol{L}}$ ja kulmanopeus $\bar{\boldsymbol{\omega}}$ ovat yhdensuuntaiset. Mutta esimerkiksi prekessioliikkeessä ne eivät ole. Kuten edellä todettiin, tässä käsitellään vain puhdasta pyörimistä.

Kun paikkavektori $\bar{\boldsymbol{r}}$ on kohtisuorassa pyörimisakseliin nähden, on skalaarisuureille voimassa $L = p\,r = m\,v\,r = I\,\omega$. Liikemäärämomentti on tärkeä käsite etenkin kitkallisten törmäysten hallinnassa.

Kun etenemisliikkeessä voiman antama impulssi $\bar{\boldsymbol{F}}\,\Delta t$ aiheuttaa kappaleen liikemäärän muutoksen $m\,\Delta\bar{\boldsymbol{v}}$, aiheuttaa vastaavasti pyörimisliikkeissä ja kiertoliikkeissä *voiman vääntömomentin impulssi* $\bar{\boldsymbol{\tau}}\,dt$ (*kulmaimpulssi* eli *momentin impulssi*) liikemäärämomentin $d\bar{\boldsymbol{L}}$ muutoksen eli

$$\bar{\boldsymbol{\tau}}\,dt = (\bar{\boldsymbol{r}} \times \bar{\boldsymbol{F}})dt = d\bar{\boldsymbol{L}} = I d\bar{\boldsymbol{\omega}}$$

Kulmaimpulssista saadaan pistemäisen kappaleen ympyräliikkeen stokastiseen simulaatioon kulmanopeuden muutoksen laskentakaava tilanteessa, jossa vaikuttava kokonaisvoima tunnetaan.

$$d\bar{\boldsymbol{\omega}} = \frac{(\bar{\boldsymbol{r}} \times \bar{\boldsymbol{F}})dt}{I}$$

$$\bar{\boldsymbol{\omega}} += d\bar{\boldsymbol{\omega}}$$

Kulmasuureiden vektorimatematiikka määrää pyörimisen suunnan siten, että katsottaessa pyörimisakselin suuntaisen kulmanopeusvektorin positiiviseen suuntaan pyöriminen tapahtuu myötäpäivään. Tätä ominaisuutta käytetään simulaation algoritmissa kappaleen paikan päivittämiseen kappaleessa 4.3 esitellyllä kiertofunktiolla `rotate()`. Seuraavassa kappaleessa käydään läpi esimerkki, jossa vektorikulmasuureet ja edellä mainittu kiertofunktio ovat ratkaisun avaimina.

7.2. 3D heilahdusliikkeen stokastinen simulointi

Tämän kappaleen esimerkissä simuloidaan reaalista 3D jäykkävartista heiluria, jossa myös ilmanvastus on huomioitu. Stokastisessa ratkaisussa heilurin 3D liikerataa simuloidaan erisuuntaisten voimien, ilmanvastus ja painovoima, aiheuttaman momentin impulssin avulla. Tämä ratkaisu sopii myös esimerkiksi Foucault'n heilurin simuloimiseen. Tässä esimerkissä heilurin 3-ulotteinen heilahdusliike tuotetaan alkunopeudella, joka poikkeaa gravitaatiovoiman määräävästä heilahdustasosta, jolloin heilurin liike muistuttaa kukan terälehdistöä.

3D heilahdusliikkeen algoritmi:

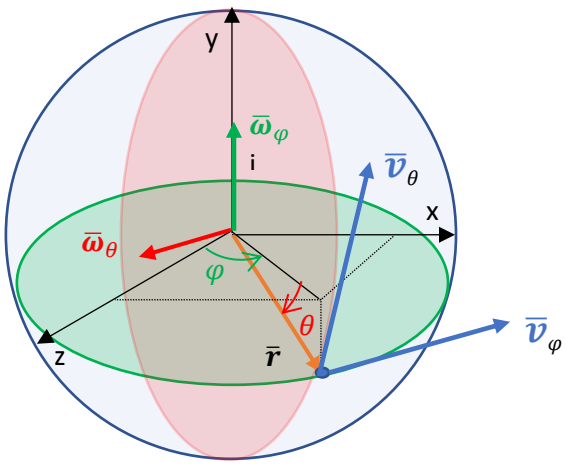
1. Laske heilurimassaan vaikuttavan kokonaisvoiman $\bar{\boldsymbol{F}}$ aiheuttama kulmaimpulssi $d\bar{\boldsymbol{L}} = (\bar{\boldsymbol{r}} \times \bar{\boldsymbol{F}})\,dt$, missä $\bar{\boldsymbol{r}}$ on heilurimassan paikkavektori ja pyörimisakseli kulkee origon kautta.
2. Laske impulssimomentin aiheuttama kulmanopeuden muutos
   $d\bar{\boldsymbol{\omega}} = \frac{d\bar{\boldsymbol{L}}}{I}$
3. Päivitä heilurimassan kulmanopeus (kiertoakseli) $\bar{\boldsymbol{\omega}} += d\bar{\boldsymbol{\omega}}$
4. Kierrä heilurimassaa kulman $d\varphi = \omega\,dt$ verran heilurin kiinnityskohdan (ja origon) kautta kulkevan kiertoakselin $\bar{\boldsymbol{\omega}}$ suhteen
5. Laske heilurimassan nopeus $\bar{\boldsymbol{v}} = \bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{r}}$ (esimerkissä ilmanvastusta varten)
6. Laske kokonaisvoima $\bar{\boldsymbol{F}}$ (esimerkissä ilmanvastus + gravitaatio)

Pallopinnan alkunopeusvektorin määritys

Alkunopeuden antaminen ei ole kovin triviaali juttu. Koska heilurin liike tapahtuu pallopinnalla, mikä tahansa nopeuden suunta ei ole mahdollinen. Tässä voidaan käyttää hyväksi **avaruuden napakoordinaatistoa ja suunnistusta, jota esimerkiksi maapallon pituus- ja leveyspiirit**

noudattavat. Kun annetaan alkunopeuden komponentit näiden piirien tangenttien suuntaisesti, saadaan kaikki mahdolliset nopeuden suunnat esitettyä. Tangenttien suunnat voidaan määrittää, kun heilurin alkupaikka $\bar{\boldsymbol{r}}$ tiedetään, jolloin pituus- ja leveyspiiriä vastaavat kulmat φ, θ voidaan laskea (Kts. kuva alla), tai jos kulmat annetaan, alkupaikka voidaan laskea.

Määritetään seuraavassa avaruuden napakoordinaatisto *VPython* -ympäristössä. Voimme muodostaa avaruuden napakoordinaatiston *samalla periaatteella kuin sivulla 5 on esitetty* 3D heittoliikkeen yhteydessä. Yleensä avaruuden napakoordinaatisto esitetään niin, että z-akseli osoittaa ylös, mutta *tässä noudatamme VPythonin ympäristöä, jossa y-akseli osoittaa ylös* (Kuva 5.).



Pisteen $P = (x, y, z)$ asema avaruudessa esitetään napakoordinaattien (r, φ, θ) avulla, missä

r = pisteen etäisyys origosta

φ = kiertokulma xz-tasossa

θ = korkeuskulma xz-tasoon nähden.

Yleensä asetetaan, että $\varphi \in [-\pi, \pi]$ ja $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, mutta tämä ei ole ainoa mahdollinen valinta. Tällöin trigonometristen funktioiden sin ja cos perusominaisuuksien nojalla avaruuden kaikissa pisteissä saadaan suorakulmaisen koordinaatiston koordinaatit seuraavilla muunnoskaavoilla:

$$x = r \sin\varphi \cos\theta$$
$$y = r \sin\theta$$
$$z = r \cos\varphi \cos\theta$$

Suorakulmaisten koordinaattien muuttaminen napakoordinaateiksi onnistuu ratkaisemalla edellisestä yhtälöryhmästä käänteiset esitykset.

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = arc\ sin\left(\frac{y}{\sqrt{x^2 + y^2 + z^2}}\right)$$
$$\varphi = arc\ tan\ (x, z)$$

Vektorin pituus r lasketaan ensin. Korkeuskulma θ saadaan palautettua oikein, koska $arcsin\left(\frac{y}{r}\right)$ palauttaa aina kulman välille $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Kiertokulma φ saadaan palautettua oikein, mikäli laskentaympäristössä on tai itse valmistaa funktion $arc\ tan(x, z)$ (vastaten muotoa $arc\ tan\left(\frac{z}{x}\right)$), missä koordinaatit x ja z annetaan erikseen, joka palauttaa kulman aina välille $[-\pi, \pi]$.

Oletetaan, että yllä olevan kuvan mukaisesti skalaarisuureet φ, θ, r, v_θ ja v_φ on annettu (tai vektori $\bar{\boldsymbol{r}}$ annettu ja φ, θ ja r laskettu).

Nopeusvektorit $\bar{\boldsymbol{v}}_\varphi$ ja $\bar{\boldsymbol{v}}_\theta$ saadaan määrittämällä ensin vastaavat kulmanopeudet $\bar{\boldsymbol{\omega}}_\varphi$ ja $\bar{\boldsymbol{\omega}}_\theta$. Kulmaa φ vastaavan kiertoliikkeen yksikkövektori $\bar{\boldsymbol{\omega}}_\varphi^0$ löydetäänkin helposti huomaamalla, että se on y-akselin suuntainen eli vektori $\bar{\boldsymbol{\omega}}_\varphi^0 = (0,1,0)$.

Kulmaa θ vastaava yksikkövektori $\bar{\boldsymbol{\omega}}_\theta^0$ on aina tasolla xz, jolloin sen y-koordinaatti on nolla. Tällöin vektori saadaan ottamalla yksikkövektori vektorista $(-r_z, 0, r_x)$, joka on kohtisuorassa vektoria $\bar{\boldsymbol{r}}$ vastaan ($\bar{\boldsymbol{r}} \cdot (-r_z, 0, r_x) = 0$). Täten $\bar{\boldsymbol{\omega}}_\theta^0 = normalize((-r_z, 0, r_x))$.

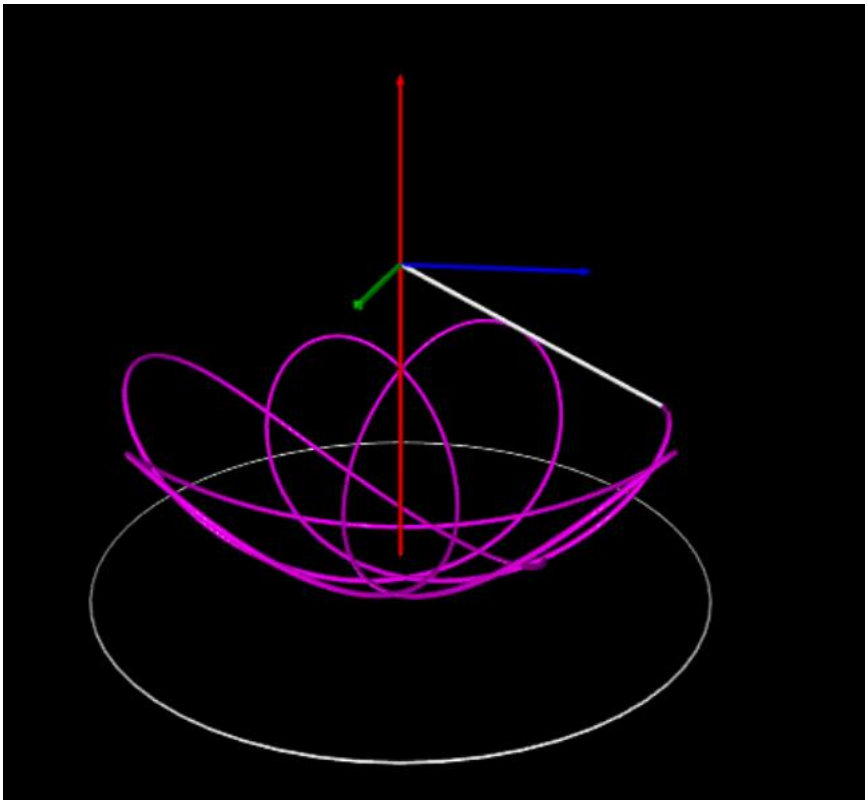
Nyt voidaan laskea vektoreiden $\bar{\boldsymbol{\omega}}$ ja $\bar{\boldsymbol{v}}$ alkuarvot, joita tarvitaan simulaation alussa.

$$\bar{\boldsymbol{\omega}}_\varphi = \frac{v_\varphi}{r}\bar{\boldsymbol{\omega}}_\varphi^0$$

$$\bar{\boldsymbol{\omega}}_\theta = \frac{v_\theta}{r}\bar{\boldsymbol{\omega}}_\theta^0$$

$$\bar{\boldsymbol{\omega}} = \bar{\boldsymbol{\omega}}_\varphi + \bar{\boldsymbol{\omega}}_\theta, \qquad \bar{v} = \bar{\boldsymbol{\omega}} \times \bar{\boldsymbol{r}}$$

Heilurille voidaan nyt antaa alkunopeus mielivaltaiseen suuntaan komponenttien v_θ ja v_φ avulla. Käyttäytyminen on kuten jäykkävartisella heilurilla, jolloin täysi kierto tukipisteen ympäri toteutuu myös. Heilurin varren on helpoin ajatella olevan ohut massaton ja ilmanvastukseton, jolloin sen voi piirtää viivana tukipisteen ja massan välille.



Kuva 8. Reaalinen 3D-heiluri, jossa ilmanvastus huomioitu Tee harjoitus 12.

8. Törmäyksien käsittely

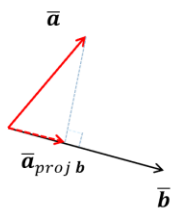
Seuraavissa kappaleissa käsitellään kappaleiden törmäykset siten, että niissä siirrytään yksinkertaisimmasta tilanteesta yleisempiin tapauksiin. Aluksi tarkastellaan pallomaisen kappaleen täysin kimmoista törmäystä eri suuntaisiin tasoihin, josta edetään pallomaisten kappaleiden keskinäisiin kitkattomiin ja lopulta reaalimaailman mukaisiin kitkallisiin törmäyksiin. Kaikkein yleisintä ja vaikeinta tilannetta, jossa epämääräisen muotoiset kappaleet törmäävät, ei käsitellä. Viimeinen kappale tarkastelee useiden kappaleiden törmäyksien kokonaisuuden hallintaa erilaisissa simulointiympäristöissä.

8.1. Pallon ja tason kohtaaminen

Aivan aluksi palautetaan mieliin vektorilaskennan eräs tehokkaimmista menetelmistä, jolla lasketaan projektiovektori ja erityisesti projektiovektorin pituus.

Vektorin $\bar{\boldsymbol{a}}$ projektiovektori vektorille $\bar{\boldsymbol{b}}$

$$\bar{\boldsymbol{a}}_{proj\ \boldsymbol{b}} = \frac{\bar{\boldsymbol{a}} \cdot \bar{\boldsymbol{b}}}{\bar{\boldsymbol{b}} \cdot \bar{\boldsymbol{b}}} \bar{\boldsymbol{b}} = \frac{\bar{\boldsymbol{a}} \cdot \bar{\boldsymbol{b}}}{|\bar{\boldsymbol{b}}|} \bar{\boldsymbol{b}}^0$$

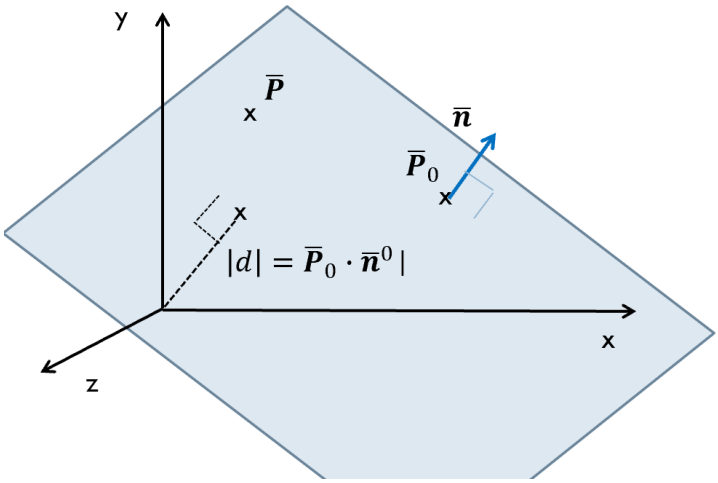


Vektorin $\bar{\boldsymbol{a}}$ skalaariprojektio vektorille $\bar{\boldsymbol{b}}$

$$a_{proj\ \boldsymbol{b}} = \frac{\bar{\boldsymbol{a}} \cdot \bar{\boldsymbol{b}}}{|\bar{\boldsymbol{b}}|}$$

Projektiovektorin pituus on skalaariprojektion itseisarvo

Jos yllä olevassa esityksessä vektori $\bar{\boldsymbol{b}}$ on yksikkövektori, saa skalaariprojektion laskettua hyvin tehokkaasti pistetulolla $\bar{\boldsymbol{a}} \cdot \bar{\boldsymbol{b}}$ ja projektiovektorin pituus saadaan edellisen itseisarvona.



Olkoon $\bar{\boldsymbol{P}}_{\boldsymbol{0}} = (p_x, p_y, p_z)$ tason tunnettu piste, $\bar{\boldsymbol{P}} = (x, y, z)$ tasolla liikkuva piste ja $\bar{\boldsymbol{n}} = (n_x, n_y, n_z)$ tason yksikkönormaali (Kuva 6.).

*Huom! Tässä kappaleessa ja kaikissa myöhemmissä tasoa koskevissa esityksissä merkintä $\bar{\boldsymbol{n}}$ tarkoittaa aina **yksikkövektoria**.*

Matematiikan oppikirjoissa johdetaan perinteisesti tason yhtälö kohtisuoruusehdosta pistetulolla

$$(\overline{\boldsymbol{P}} - \overline{\boldsymbol{P}}_{\mathbf{0}}) \cdot \overline{\boldsymbol{n}} = p_x x + p_y y + p_z z - p_x n_x - p_y n_y - p_z n_z = 0$$

$$p_x x + p_y y + p_z z = p_x n_x + p_y n_y + p_z n_z = d \in \mathbb{R}$$

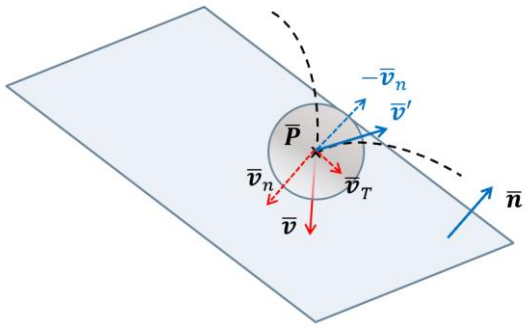
toisin sanoen yleinen tason yhtälö esitetään muodossa

$$a\,x + b\,y + c\,z = d$$

Tämä muoto ei kuitenkaan ole kovin käyttökelpoinen ohjelmointiympäristöissä. On selvää, että paikkavektori $\overline{\boldsymbol{P}}_0$ (tason tunnettu piste) sekä tason suunnan osoittava yksikkönormaali $\overline{\boldsymbol{n}}$ määrittävät tason yksikäsitteisesti. Käyttökelpoisin yksikäsitteinen esitys tasolle saadaan kuitenkin parista, jonka muodostavat yksikkönormaali $\overline{\boldsymbol{n}}$ ja vakio d. Yleensä ne esitetään järjestettynä parina muodossa $\langle \overline{\boldsymbol{n}}, d \rangle$.

Edellisen tason yhtälön johdon perusteella vakio $d = \overline{\boldsymbol{P}}_{\mathbf{0}} \cdot \overline{\boldsymbol{n}}$ antaa pisteen $\overline{\boldsymbol{P}}_{\mathbf{0}}$ skalaariprojektion suunnassa $\overline{\boldsymbol{n}}$**.** Vakion itseisarvo antaa siten tason etäisyyden origosta. Vakio d on erittäin hyödyllinen ja tehokas laskennan kannalta. Sen avulla voidaan tehokkaasti laskea, onko jokin piste tasolla, ja jos ei, niin mikä on pisteen etäisyys tasosta. Tämä taas on tärkeää tutkittaessa törmääkö kappale tasoon.

Tasoon liittyy vielä yksi asia, joka vaatii tarkastelua ennen varsinaisten laskentakaavojen esittelyä. Tasolle voidaan aina määritellä kaksi normaalia, jotka ovat toistensa vastavektoreita. On ensiarvoisen tärkeää tietää, kummalle puolelle tason normaali osoittaa suhteessa johonkin tapahtumapisteeseen. Liitteessä 3 tarkastellaan tätä ongelmaa lähemmin. Tässä rajoitutaan tilanteisiin, joissa tapahtumapisteet oletetaan olevan sillä puolella tasoa, jonne normaali osoittaa.



Yksinkertaisimmin pallon törmääminen tasoon voidaan mallintaa kitkattomasti ja täysin kimmoisasti pelkästään vaihtamalla nopeuden normaalinsuuntaisen komponentin etumerkki törmäyksen tapahduttua (kitkattomuuden vuoksi tason suuntainen ei muutu). Tämä edellyttää nopeusvektorin jakamista kahteen komponenttiin, normaalin $\overline{\boldsymbol{v}}_n$ ja tason suuntaiseen $\overline{\boldsymbol{v}}_T$, mikä voidaan toteuttaa skalaariprojektion avulla.

$$\overline{\boldsymbol{v}}_n = (\overline{\boldsymbol{v}} \cdot \overline{\boldsymbol{n}})\,\overline{\boldsymbol{n}}$$

$$\overline{\boldsymbol{v}}_T = \overline{\boldsymbol{v}} - \overline{\boldsymbol{v}}_n$$

$$\overline{\boldsymbol{v}}' = \overline{\boldsymbol{v}}_T - \overline{\boldsymbol{v}}_n$$

Törmäyksen tapahtumisehto saadaan tason vakion ja normaalin avulla ehdosta

$$\overline{\boldsymbol{P}} \cdot \overline{\boldsymbol{n}} - d < r, \qquad (1)$$

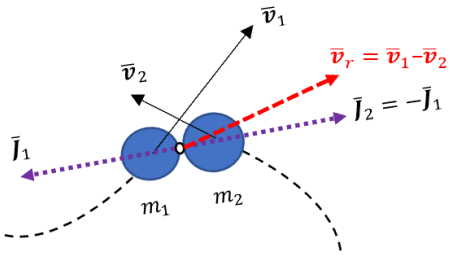
missä r = pallon säde ja erotus $\overline{\boldsymbol{P}} \cdot \overline{\boldsymbol{n}} - d$ ilmoittaa pisteen $\overline{\boldsymbol{P}}$ kohtisuoran etäisyyden tasosta $\langle \overline{\boldsymbol{n}}, d \rangle$. Ehto toimii (ilman itseisarvojakin) kaikissa tilanteissa, kunhan huolehditaan, että piste $\overline{\boldsymbol{P}}$ sijaitsee normaalin osoittamalla puolella tasoa.

Törmäyksiin liittyy hyvin yleinen algoritmillinen ongelma, kappaleiden jumittuminen toisiinsa. Useimmiten tämä johtuu siitä, että laskennallisesti kappaleet tunkeutuvat törmäyksissä toistensa sisään. Kun nopeuden suunta on vaihdettu, ne eivät välttämättä ole irtaantuneet toisistaan seuraavassa tarkistusvaiheessa. Tällöin törmäysehto toteutuu uudelleen ja nopeuden suunta vaihtuu myös uudelleen.

Yksinkertaisin tapa, joka useimmiten toimii, on vain peruuttaa edellinen siirtymä jokaisen törmäyksen yhteydessä ennen uuden nopeuden vaihtamista. Helpoitenhan se tapahtuu paikan päivityksellä $\overline{\boldsymbol{pos}} -= \overline{\boldsymbol{v}}\, dt$.

Seuraavissa kappaleissa käsitellään pallomaisten kappaleiden keskinäisten törmäyksien hallintaa *impulssiperiaatteella* ja niissä esitellään lisää keinoja hoitaa tuo edellä mainittu jumittumisongelma.

8.2. Impulssiperiaate ja pallomaisten kappaleiden kitkaton törmäys



Merkitään $\overline{\boldsymbol{J}}$ on voiman aiheuttama impulssi. Kun pallomaiset kappaleet törmäävät kitkattomasti, törmäyksen vaikutussuora kulkee kappaleiden massakeskipisteiden kautta. Tämän perusteella voidaan impulssin suuruus laskea.

Kappaleiden saamat impulssit ovat yhtä suuret mutta vastakkaissuuntaiset. Liikemäärän säilymisen nojalla:

$$\overline{\boldsymbol{p}}_1{}' = \overline{\boldsymbol{p}}_1 + \overline{\boldsymbol{J}}_1$$
$$\overline{\boldsymbol{p}}_2{}' = \overline{\boldsymbol{p}}_2 + \overline{\boldsymbol{J}}_2 = \overline{\boldsymbol{p}}_2 - \overline{\boldsymbol{J}}_1$$

Törmäyksen kimmoisuuden astetta mallinnetaan **sysäyskertoimella** *e*, joka kuvaa sitä, kuinka suuri osa kineettisestä energiasta muuttuu törmäyksessä johonkin muuhun energiamuotoon seuraavalla tavalla:

$$u_2 - u_1 = e(v_1 - v_2) \Leftrightarrow e = \frac{u_2 - u_1}{v_1 - v_2}.$$

Sysäyskerroin e saa arvon 0 täysin kimmottomassa ($u_2 = u_1$) ja arvon 1 täysin kimmoisassa törmäyksessä. Jälkimmäinen seuraa siitä, että sekä liike-energia että liikemäärä säilyvät täysin kimmoisassa törmäyksessä (yhtälöparin käsittely ohitetaan).

$$\begin{cases} m_1 v_1 + m_2 v_2 = m_1 u_1 + m_2 u_2 \\ \frac{1}{2} m_1 {v_1}^2 + \frac{1}{2} m_2 {v_2}^2 = \frac{1}{2} m_1 {u_1}^2 + \frac{1}{2} m_2 {u_2}^2 \end{cases}$$
$$\Rightarrow u_2 - u_1 = v_1 - v_2$$

Simulaatiossa sysäyskertoimelle tulee antaa kokeilemalla arvo, jolla käyttäytyminen on reaalisinta. Sysäyskerroin voi olla myös negatiivinen (esimerkiksi luoti lävistää hauraan materian) tai myös suurempi kuin 1 (törmäyksessä vapautuu lisäenergiaa esimerkiksi räjähdyksestä).

Seuraavassa tarkastellaan impulssiperiaatteen matemaattista perustaa ja johdetaan laskentakaavat, joilla simulaatiossa nopeudet päivitetään törmäyksissä. Ratkaisun idea on se, että toisaalta voimme geometriasta selvittää impulssin vaikutussuunnan, ja toisaalta liikemäärän säilymisen yhtälöistä voimme ratkaista impulssin suuruuden. Ratkaisu edellyttää, että jaamme nopeudet komponentteihin, jotka ovat impulssien vaikutussuoran suuntaiset (alla alaindeksi j) ja sen kanssa kohtisuorassa. Jälkimmäiset eivät vaikuta törmäyksessä, koska kitkaa ei ole. Tällöin riittää, että tarkastelemme pelkästään vaikutussuoran suuntaisia nopeuksia myös sysäyskertoimen osalta. Nopeudet voidaan esittää nyt skalaarisuureina, joiden etumerkit määräävät suunnan.

Saamme seuraavan yhtälöryhmän.

$$\begin{cases} m_1 u_{1j} = m_1 v_{1j} - J \\ m_2 u_{2j} = m_2 v_{2j} + J \\ e = \frac{u_{2j} - u_{1j}}{v_{1j} - v_{2j}} \end{cases}$$

Ratkaistaan yhtälöryhmästä kolme tuntematonta u_{1j}, u_{2j} ja J, jolloin impulssille J voidaan saada pienen ryhmittelyn jälkeen seuraava laskentakaava:

$$J = \frac{(1+e)(v_{1j} - v_{2j})}{\frac{1}{m_1} + \frac{1}{m_2}}$$

Laskentakaavassa nimittäjässä on lauseke kirjoitettu muotoon $\frac{1}{m_1} + \frac{1}{m_2}$ erityisestä syystä. Jos toinen kappale on hyvin massiivinen, jonka nopeuden voi ajatella törmäyksessä pysyvän muuttumattomana, paljastaa esitysmuoto, että sitä koskeva massalauseke voidaan jättää pois, koska $\frac{1}{m} \approx 0$.

Törmäyksen vaikutussuoran suunta on sama kuin törmäyskohdan pinnan normaalin $\overline{\boldsymbol{n}}$ suunta.

Tällöin impulssin laskentakaavassa nopeuksien erotus $v_{1j} - v_{2j}$ saadaan laskettua kappaleiden (törmäävien pintojen) **suhteellisen nopeuden**

$$\overline{\boldsymbol{v}}_r = \overline{\boldsymbol{v}}_1 - \overline{\boldsymbol{v}}_2 \tag{2}$$

ja normaalin $\overline{\boldsymbol{n}}$ skalaariprojektiona $\overline{\boldsymbol{v}}_r \cdot \overline{\boldsymbol{n}}$. Suhteellista nopeutta voidaan pitää nopeutena, joka toiselle kappaleelle muodostuu vastaavassa törmäyksessä, missä toisen kappaleen nopeus on asetettu nollaan (pääkoordinaatiston vaihto toiseen kappaleeseen). Suhteellinen nopeus voidaan ajatella jaettavaksi kahteen komponenttiin, joista toinen on normaalin suuntainen ja toinen kohtisuorassa sitä vastaan.

Impulssin suuruus voidaan nyt laskea muodossa:

$$J = \frac{(1+e)|\overline{\boldsymbol{v}}_r \cdot \overline{\boldsymbol{n}}|}{\frac{1}{m_1} + \frac{1}{m_2}} \tag{3}$$

ja nopeudet voidaan päivittää seuraavasti (normaali $\overline{\boldsymbol{n}}$ valittu massan 1 nopeutta vastaan):

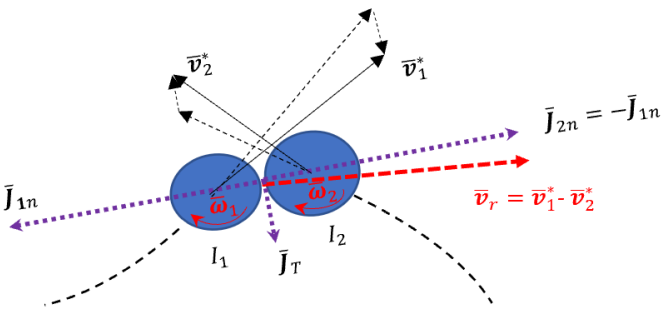
$$\begin{aligned} \overline{\boldsymbol{v}}_1 &+= J\,\overline{\boldsymbol{n}}\,/m_1 \\ \overline{\boldsymbol{v}}_2 &+= -J\,\overline{\boldsymbol{n}}\,/m_2 \end{aligned} \tag{4}$$

Kaavan (1) avulla laskettuun törmäyksen tapahtumisehtoon ’ *Pallojen välinen etäisyys pitää olla alle 2xsäde*’ voidaan lisätä myös toinen lisäehto, jolla on tarkoitus estää jumittuminen. Kun suhteellinen nopeus on määritelty kappaleesta 1 käsin eli $\overline{\boldsymbol{v}}_r = \overline{\boldsymbol{v}}_1 - \overline{\boldsymbol{v}}_2$, on positiivinen suunta kappaleen 1 massakeskipisteestä poispäin. Jos laskettu pinnan normaali $\overline{\boldsymbol{n}}$ osoittaa myös pois kappaleesta 1 kohti kappaletta 2, *kertoo pistetulon* $\overline{\boldsymbol{v}}_r \cdot \overline{\boldsymbol{n}}$ *etumerkki sen, ovatko kappaleet siirtymässä kauemmaksi toisistaan vai lähentymässä*.

$$\overline{\boldsymbol{v}}_r \cdot \overline{\boldsymbol{n}} < 0 \Leftrightarrow \sphericalangle(\overline{\boldsymbol{v}}_r, \overline{\boldsymbol{n}}) > 90° \tag{5}$$

Negatiivinen tulos merkitsee, että vektoreiden välinen kulma on yli 90° ja törmäystä ei synny. Molempien ehtojen toteutuminen merkitsee, että kappaleet ovat törmäämässä. Tällöin on usein välttämätöntä siirtää kappaleet irti toisistaan ennen uuden nopeuden soveltamista, jotta jumittumista ei tapahdu.

8.3. Pallomaisten kappaleiden kitkallinen törmäys



Kun pallomaiset kappaleet törmäävät kitkallisesti, on kappaleiden pyörimisnopeudet ja hitausmomentit huomioitava. Pintojen normaalin suuntaisen impulssin laskeminen ei eroa kitkattoman törmäyksen tapauksesta muuten, kuin laskettaessa törmäävien pintojen suhteellista nopeutta $\overline{\boldsymbol{v}}_r$ pitää pyörimisliikkeet ottaa huomioon.

Merkitään R_1 ja R_2 ovat pallojen säteet ja $\overline{\boldsymbol{n}}$ kappaleen 1 pinnan normaalin yksikkövektori törmäyspisteessä. Tällöin törmäävien pintojen todelliset nopeudet ja suhteellinen nopeus ovat ennen törmäystä

$$\begin{aligned}
\overline{\boldsymbol{v}}_1^* &= \overline{\boldsymbol{v}}_{\boldsymbol{1}} + \overline{\boldsymbol{\omega}}_{\boldsymbol{1}} \times R_1 \overline{\boldsymbol{n}} \\
\overline{\boldsymbol{v}}_2^* &= \overline{\boldsymbol{v}}_{\boldsymbol{2}} + \overline{\boldsymbol{\omega}}_{\boldsymbol{2}} \times (-R_2 \overline{\boldsymbol{n}}) \\
\overline{\boldsymbol{v}}_r &= \overline{\boldsymbol{v}}_{\boldsymbol{1}}^* - \overline{\boldsymbol{v}}_{\boldsymbol{2}}^*
\end{aligned} \tag{6}$$

Merkitään törmäyksen vaikutussuoran eli normaalin suuntaista impulssia alaindeksillä n ja sen kanssa kohtisuorassa olevaa tangentiaalista impulssia indeksillä t. Nyt $\overline{\boldsymbol{J}} = \overline{\boldsymbol{J}}_n + \overline{\boldsymbol{J}}_t$, missä J_n lasketaan edelleen kaavalla (3) käyttäen kaavan (6) mukaista pintojen välistä todellista suhteellista nopeutta.

Kitkavoiman aiheuttama tangentin suuntaisen impulssin suuruus syntyy pintojen liukuessa hivenen toistensa ohi törmäyksen aikana. Kitkavoiman määritelmän mukaan $\overline{\boldsymbol{F}}_\mu = \mu\, \overline{\boldsymbol{F}}_n$, missä $\overline{\boldsymbol{F}}_n$ on pintojen välinen (keskimääräinen) normaalivoima, joka puolestaan tuottaa normaalin

suuntaisen impulssin $\bar{\boldsymbol{J}}_n$ sekä μ on kitkakerroin. Tällöin tangentiaalisen impulssin suuruus lasketaan kaavalla

$$J_t = \mu\, F_n\, dt = \mu\, J_n \tag{7}$$

Erityisesti kovien esineiden törmäyksissä kitkakerroin μ ei kuitenkaan voi olla sama kuin vastaavien pintojen liukukitkakerroin, koska tangentiaalisen impulssin suuruus kasvaa selvästi epäreaaliseksi. Muutenkin tangentiaalisen impulssin vaikutuksen havaitseminen törmäyksissä on vaikeaa. Kovien Curling -kivien kohdalla kerroin μ on asetettava hyvin pieneksi.

Tangentiaalisen impulssin suunta riippuu suhteellisen nopeuden $\bar{\boldsymbol{v}}_{\boldsymbol{r}}$ suunnasta, joka on sama molemmille kappaleille. Normaalin suuntainen impulssi vaikuttaa vain massakeskipisteen etenemisnopeuteen, mutta tangentin suuntainen impulssi vaikuttaa sekä edelliseen että myös kappaleiden kulmanopeuteen.

Kohtisuora tangentin suuntainen yksikkövektori törmäyskohdassa lasketaan vektoreiden $\bar{\boldsymbol{v}}_{\boldsymbol{r}}$ ja $\bar{\boldsymbol{n}}$ sekä edellisen vektoriprojektion avulla.

$$\bar{\boldsymbol{t}} = \text{normalisoi}(\bar{\boldsymbol{v}}_{\boldsymbol{r}} - (\bar{\boldsymbol{v}}_{\boldsymbol{r}} \cdot \bar{\boldsymbol{n}})\, \bar{\boldsymbol{n}}) \tag{8}$$

Tangentiaalinen impulssi saadaan nyt laskettua etumerkkiä vaille kaavasta

$$\bar{\boldsymbol{J}}_t = \mu J_n\, \bar{\boldsymbol{t}}, \tag{9}$$

missä etumerkit törmääville kappaleille ovat vastakkaiset. On huomioitavaa, että mikäli suhteellinen nopeus on normaalin suuntainen, tulee tangentiksi 0-vektori, jolloin myös tangentiaalinen impulssi on nolla kuten pitääkin.

Kappaleiden keskipisteiden etenemisnopeudet päivitetään nyt kaavoilla

$$\begin{aligned} \bar{\boldsymbol{J}} &= \bar{\boldsymbol{J}}_t + \bar{\boldsymbol{J}}_n \\ \bar{\boldsymbol{v}}_1 &+= \bar{\boldsymbol{J}}/m_1 \\ \bar{\boldsymbol{v}}_2 &+= -\bar{\boldsymbol{J}}/m_2 \end{aligned} \tag{10}$$

Tangentiaalisen impulssin muuttamat kulmanopeudet päivitetään kaavan $d\bar{\boldsymbol{\omega}} = \frac{(\bar{\boldsymbol{r}} \times \bar{\boldsymbol{F}})dt}{I}$ mukaisesti.

$$\begin{aligned} \bar{\boldsymbol{\omega}}_1 &+= (R_1 \bar{\boldsymbol{n}}) \times \bar{\boldsymbol{J}}_T / I_1 \\ \bar{\boldsymbol{\omega}}_2 &+= \frac{(-R_2 \bar{\boldsymbol{n}}) \times (-\bar{\boldsymbol{J}}_T)}{I_2} = (R_2 \bar{\boldsymbol{n}}) \times \bar{\boldsymbol{J}}_T / I_2 \end{aligned} \tag{11}$$

Jälkimmäisessä kulmanopeuden kaavassa miinusmerkit ovat mukana sen vuoksi, että ne havainnollistavat vastakkaisia suuntia. Toisaalta kappaleen 2 normaalin suunta on vastakkainen ja toisaalta tangentiaalinen impulssi pyrkii muuttaman pallon 2 kulmanopeutta vastakkaiseen suuntaan. Miinusmerkit kumoavat toistensa vaikutuksen.

Kaavat (1) – (11) toimivat samalla algoritmeina, joilla kitkaton tai kitkallinen pallomaisten kappaleiden törmääminen voidaan ohjelmoida.

8.5. Useiden kappaleiden törmäyksien hallinnasta

Ratkaistavien algoritmillisten ongelmien piiriin kuuluu myös se, kuinka kaikki törmäykset pystytään hoitamaan oikeassa järjestyksessä. Automaattisten fysiikkamoottoreiden yleinen ratkaisuperiaate on se, että vapaa liikkuminen, törmäysten havainnoiminen ja niiden käsittely pidetään erillisinä toimintoina. Fysiikkamoottoreissa objekteihin liitetään pintoja myötäileviä yksinkertaistettuja törmäysten havainnointirakenteita. Ne ovat tapahtumien kirjaajia, jotka tarkkailevat seuraamaansa kappaletta yleensä hieman suurempaa ympäröivää tilaa ja pitävät kirjaa siitä, mitkä muut objektit ovat tunkeutuneet tähän tilaan. Samoin nämä havainnoijat ajavat automaattisesti määrätyt törmäysten käsittelyfunktiot, joihin ohjelmoija voi kirjoittaa koodiansa.

Omien törmäyssimulaatioiden rakentamisessa *VPython*issa ei ole mahdollista käyttää valmista fysiikkamoottoria eikä automaattisia törmäysten havainnoijia. Törmäysten hallinta on hoidettava siten itse. Ratkaisu voi olla esimerkiksi seuraava:

Kaikki tilassa liikkuvat kappaleet luetteloidaan niin, että ne voidaan identifioida tallennusrakenteista. Kaikille mahdollisesti törmääville pareille muodostetaan 2-ulotteinen taulukko, jossa kuhunkin alkioon $a_{ij}, i \neq j$, liittyy tieto (True/False) siitä ovatko kappaleet i ja j törmäämässä. Oletuksena on siis False. Itse asiassa taulukosta riittää läpikäydä vain lävistäjä ja sen yläpuolinen kolmio, jolloin jokainen pari on läpikäyty.

Simulaation silmukassa kullakin toistokierroksella ensin siirretään vapaasti liikkuvia kappaleita. Sen jälkeen tarkistetaan ja kirjataan ylös kaikki mahdolliset törmäävät parit. Viimeisenä vaiheena välitetään kaikki törmäävät parit törmäysten käsittelijälle ennen kuin siirrytään taas vapaaseen siirtymiseen. Näin mahdollistetaan se, että törmäyksien käsittelyn aikana ei tapahdu kappaleiden vapaata liikkumista, mikä yleensä riittää estämään kappaleiden liikkeiden ”karkaamisen käsistä”.

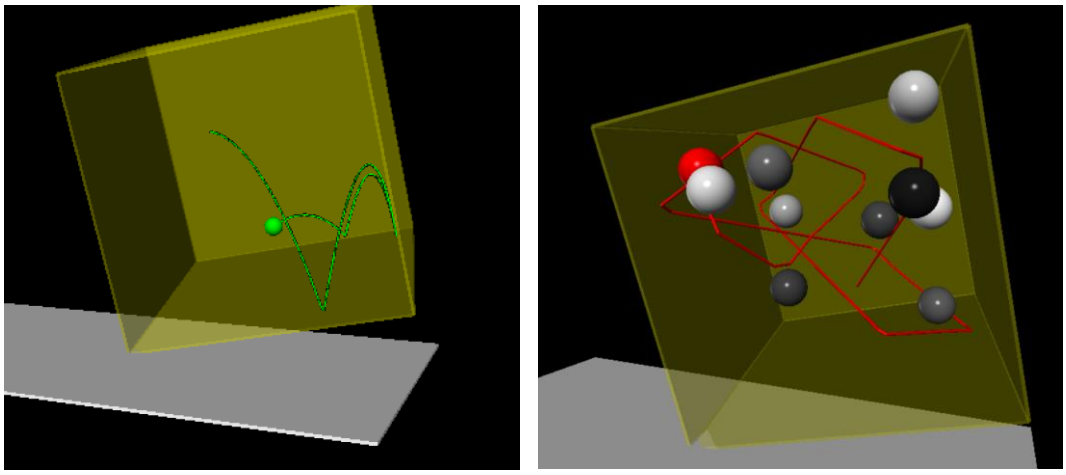
Törmäävien kappaleiden jumittuminen on myös välttämätön seuraus jossakin vaiheessa ilman sen hoitamista. Sen voi tehdä esimerkiksi siirtämällä toisiinsa tunkeutuneet kappaleet erilleen törmäystilanteessa (kts. kpl 8.2 loppu).

Useiden kitkattomien pallomaisten kappaleiden törmäyksien hallinta-algoritmi:

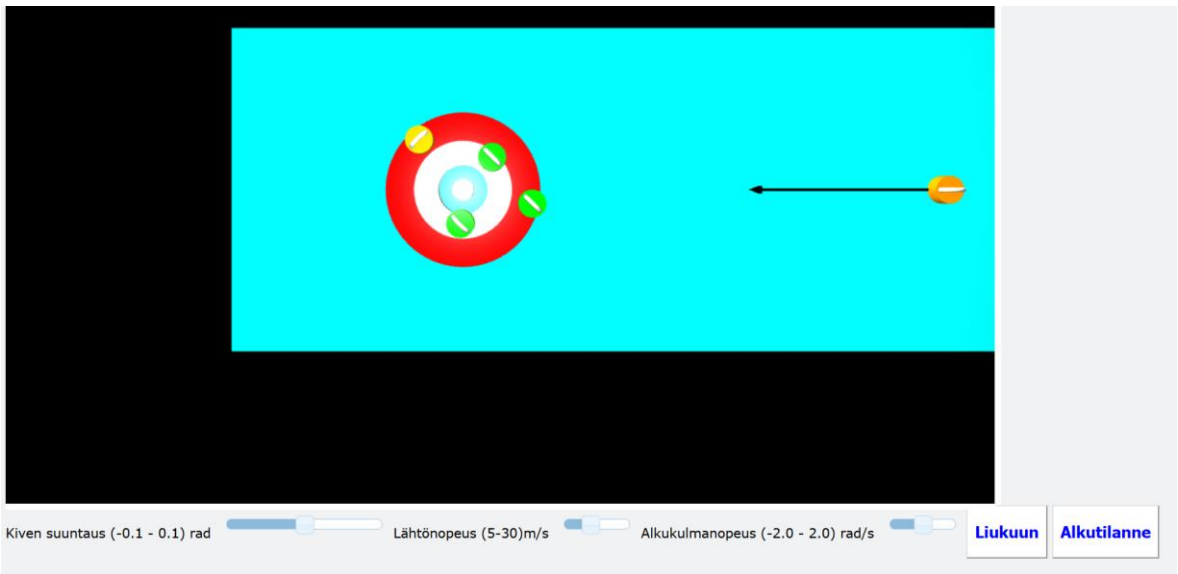
```
Recording structures:
List(s) of colliding bodies
Table(s) of colliding pairs initialized by False

1. ApplyMoving()
     free moving (flying, sliding, …)

2. CheckForCollision()
     repeat for each possible colliding pair
       compute relative velocity (suhteellinen nopeus)
       compute unit normal
       if (distance < minimum)
         move bodies off to prevent jamming
         if (bodies are moving toward each other)
           set colliding pair = True
```

```
3. Repeat for each colliding pairs in array
    if (colliding pair == True)
      onCollision(colliding pair)
        compute relative velocity
        compute unit normal
        compute impulse
        update velocity affected by the impulse
        update position
      set colliding pair = False
```

Kitkallisten törmäysten hallinta eroaa edellisestä siinä, että myös kappaleiden kulmanopeuksista on huolehdittava törmäyksien käsittelyssä. Laskettaessa suhteellista nopeutta on kappaleiden kulmanopeuksien vaikutus pintojen törmäysnopeuteen huomioitava. Lisäksi törmäystilanteissa on laskettava ja huomioitava tangentiaalisen impulssin vaikutus sekä etenemisnopeuteen että myös kulmanopeuteen sivujen 37-38 mukaisesti. Itse törmäysten käsittelyn algoritmi ei muutu.



Kuva 9. Kitkaton törmäys Tee harjoitukset 13 ja 14.



Kuva 6. Kitkallinen törmäys, curling-kivet Tee harjoitus 15.

9. Kirjallisuutta

Dokumentissa on hyödynnetty Wikipedian lisäksi erilaisia web-sivustoja, joiden osoitteet löytyvät tekstistä kulloinkin käsiteltävän asian yhteydestä. Lisäksi olen käyttänyt alla lueteltuja teoksia, mutta niihin ei ole tekstissä viitattu yksilöidysti.

1. Bourg David M. & Bywalec Bryan, 2013, Physics for Game Developers, ISBN: 978-1-449-39251-2

2. Dunn Fletcher & Parberry Ian, 2002, 3D Math Primer for Graphics and Game Development, ISBN: 1-55622-911-9

3. Lengyel Eric, 2012, Mathematics for 3D Game Programming and Computer Graphics, Third Ed., ISBN-13: 978-1-4354-5886-4, ISBN-10: 1-4354-5886-9

Mikäli haluaa perehtyä mahdollisimman reaaliseen taivaanmekaniikkaan, löytyy siihen todella hyviä suomenkielisiä teoksia palkitun ja kiitetyn Hannu Karttusen kirjoittamana.

4. Karttunen Hannu, 2015, johdatus taivaanmekaniikkaan, ISBN 978-952-5329-20-9, ISBN 0357-7937

5. Karttunen Hannu, 2017, Tähdet ja maailman kaikkeus, ISBN 978-951-1-32036-4

10. Liitteet

Liite 1 Kiertofunktio

Tehtävänä on konstruoida kiertofunktio, joka kiertää 3D koordinaatiston pisteen $\overline{\boldsymbol{P}} = (p_x, p_y, p_z)$ pisteeksi $\overline{\boldsymbol{P}}'$ origon kautta kulkevan mielivaltaisen pyörimisakselin $\overline{\boldsymbol{v}} = (v_x, v_y, v_z)$ suhteen kulman α verran. Funktio muodostetaan kvaterniotekniikkaa hyödyntämällä mutta ilman kvaternioalgebran laskentafunktioita konstruoimalla 3×3 kiertomatriisi R, joka toteuttaa halutun kierron matriisioperaationa

$$\overline{\boldsymbol{P}}' = R.\overline{\boldsymbol{P}} \tag{1}$$

Kvaternio q on reaaliluvuista muodostettu nelikko, jolle voidaan esittää seuraavat esitysmuodot:

$$q = \langle w, x, y, z \rangle = w + \overline{\boldsymbol{u}}, \tag{2}$$

missä $\overline{\boldsymbol{u}} = (x, y, z) \in \mathbb{R}^3$ on vektori ja muut symbolit reaalilukuja. Kvaternioille tulee määritellä omat laskutoimitukset, minkä esitys (2) jo osoittaa. Reaaliluvun ja vektorin yhteenlasku ei ole sallittu operaatio ilman, että ne määritellään kvaternioiksi. Menemättä tarkemmin kvaternioiden laskutoimituksiin, todetaan tässä, että yksikkökvaternio

$$\hat{q} = \langle w_0, x_0, y_0, z_0 \rangle, \qquad |\hat{q}| = \sqrt{w_0^2 + x_0^2 + y_0^2 + z_0^2} = 1$$

voidaan liittää annettuun konstruointitehtävään siten, että sen alkiot w_0, x_0, y_0 ja z_0 pitävät sisällään tiedon kiertokulmasta ja origon kautta kulkevasta kiertoakselista. Yksikkövektorille löytyy aina esitysmuoto

$$\hat{q} = \cos\theta + \sin\theta\, \overline{\boldsymbol{u}}^0 \,,$$

mistä kiertokulma ja kiertoakseli saadaan. Reaaliluvusta $w_0 = \cos\theta$ saadaan kiertokulma, joka on 2θ, sekä vektorista $\overline{\boldsymbol{u}} = (x_0, y_0, z_0) = \sin\theta\, \overline{\boldsymbol{u}}^0$ tai tietenkin myös yksikkövektorista $\overline{\boldsymbol{u}}^0$ saadaan kiertoakseli.

Näin ollen, kun kierretään pistettä $\overline{\boldsymbol{P}} = (p_x, p_y, p_z)$ origon kautta kulkevan kiertoakselin $\overline{\boldsymbol{v}} = (v_x, v_y, v_z)$ suhteen kulman α verran, muodostetaan ensin seuraava yksikkökvaternio:

$$\hat{q} = \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\overline{\boldsymbol{v}}^0 = \langle \cos\left(\frac{\alpha}{2}\right), \sin\left(\frac{\alpha}{2}\right) v_x^0, \sin\left(\frac{\alpha}{2}\right) v_y^0, \sin\left(\frac{\alpha}{2}\right) v_z^0 \rangle \tag{3}$$

missä $\overline{\boldsymbol{v}}^0$ on vektorin $\overline{\boldsymbol{v}}$ yksikkövektori.

Voidaan osoittaa, että yksikkökvaternio (3) toteuttaa pisteen $\overline{\boldsymbol{P}} = (p_x, p_y, p_z)$ yllä määritellyn kierron kvaternioiden laskutoimituksella

$$\dot{P}' = \hat{q} \cdot \dot{P} \cdot \hat{q}^{-1}, \tag{4}$$

missä konjugaatti $\hat{q}^{-1} = \cos\left(\frac{\alpha}{2}\right) - \sin\left(\frac{\alpha}{2}\right)\overline{\boldsymbol{v}}^0$ sekä $\dot{P} = \langle 0, p_x, p_y, p_z \rangle$ ja $\dot{P}' = \langle 0, p'_x, p'_y, p'_z \rangle$ ovat vektoreiden $\overline{\boldsymbol{P}}$ ja $\overline{\boldsymbol{P}}'$ kvaternioesityksiä.

Seuraavaksi kiertomatriisin konstruoimiseksi tarvitsemme kvaternioiden kertolaskun $(\cdot)$ määrittelyn, joka voidaan esittää vektorilaskennan operaatioilla seuraavasti:

$$q_1 \cdot q_2 = (w_1 + \overline{\boldsymbol{u}}_1)(w_2 + \overline{\boldsymbol{u}}_2) = w_1 w_2 + w_1 \overline{\boldsymbol{u}}_2 + w_2 \overline{\boldsymbol{u}}_1 + \overline{\boldsymbol{u}}_1 \times \overline{\boldsymbol{u}}_2 \tag{5}$$

Nyt voimmekin konstruoida kiertomatriisin R ilman kvaternioalgebran laskentafunktioita.

Käyttämällä yhtälöitä (3), (4) ja (5) voimme ratkaista symbolisen esityksen halutulle kiertomatriisille vertaamalla symbolista kiertomatriisin tulosta $R.\overline{\boldsymbol{P}}$ yhtälöstä (5) saatavaan tulokseen. Toki aika paljon sieventämistä ja symbolista prosessointia saa tehdä alla olevan sievennetyn esityksen (6) saamiseksi. Huomioitavaa on, että yksikkökvaterniolle $\hat{q} = \langle q_0, q_x, q_y, q_z \rangle$ on voimassa $q_x^2 + q_y^2 + q_z^2 = 1 - q_0^2$, koska $|\hat{q}| = 1$.

Kiertomatriisi R, joka kiertää koordinaatiston pisteen $\overline{\boldsymbol{P}} = (p_x, p_y, p_z)$ origon kautta kulkevan mielivaltaisen pyörimisakselin $\overline{\boldsymbol{v}} = (v_x, v_y, v_z)$ suhteen kulman α verran, konstruoidaan yksikkökvaternion

$$\hat{q} = \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\overline{\boldsymbol{v}}^0 = \langle \cos\left(\frac{\alpha}{2}\right), \sin\left(\frac{\alpha}{2}\right) v_x^0, \sin\left(\frac{\alpha}{2}\right) v_y^0, \sin\left(\frac{\alpha}{2}\right) v_z^0 \rangle$$

komponenteista, joita merkitään seuraavassa esityksessä lyhyesti symbolein $\langle q0, qx, qy, qz \rangle$.

$$R = \begin{pmatrix} 2\mathrm{q0}^2 + 2\mathrm{qx}^2 - 1 & 2\mathrm{qx\ qy} - 2\mathrm{q0\ qz} & 2\mathrm{q0\ qy} + 2\mathrm{qx\ qz} \\ 2\mathrm{qx\ qy} + 2\mathrm{q0\ qz} & 2\mathrm{q0}^2 + 2\mathrm{qy}^2 - 1 & 2\mathrm{qy\ qz} - 2\mathrm{q0\ qx} \\ 2\mathrm{qx\ qz} - 2\mathrm{q0\ qy} & 2\mathrm{q0\ qx} + 2\mathrm{qy\ qz} & 2\mathrm{q0}^2 + 2\mathrm{qz}^2 - 1 \end{pmatrix} \tag{6}$$

Ohessa vielä matriisin esitys koodaukseen sopivammassa muodossa:

```
R={{2*q0^2 + 2*qx^2 - 1, 2*qx*qy - 2*q0*qz, 2*q0*qy + 2*qx*qz},
  {2*qx*qy + 2*q0*qz, 2*q0^2 + 2*qy^2 - 1, 2*qy*qz - 2*q0*qx},
  {2*qx*qz - 2*q0*qy, 2*q0*qx + 2*qy*qz, 2*q0^2 + 2*qz^2 - 1}}
```

Edellä olevan konstruoinnin mukaisen kierron $R.\overline{\boldsymbol{P}}$ voi toteuttaa helposti yhdessä funktiossa, jossa argumenttina annetaan kierrettävä vektori $\overline{\boldsymbol{P}}$, kiertokulma α ja kiertoakseli $\overline{\boldsymbol{v}}$. Funktion voi laajentaa tapaukseen, jossa pyörimisakseli ei kulje origon kautta. Tällöin toimitaan samoin kuin kiertojen yhteydessä sivulla 16 on esitetty eli periaatteella *siirto origoon – kierto origon kautta – siirto takaisin*.

Huom. 1. Vaikka saamme yksikäsitteisesti saman kierron sekä yhtälön (1) mukaisesti kuin myös yhtälön (4) mukaisesti, eivät nämä operaatiot ole täysin isomorfiset. Nimittäin kutakin kerroinmatriisia kohti löytyy kaksi yksikkökvaterniota $\hat{q}$ ja $-\hat{q}$, jotka tekevät saman kierron eli

$$\dot{P}' = \hat{q} \cdot \dot{P} \cdot \hat{q}^{-1} = (-\hat{q}) \cdot \dot{P} \cdot (-\hat{q}^{-1})$$

Huom. 2. Matriisille R on kiertomatriisina oltava voimassa $R.R^T = R^T.R = I$ ja $\det(R) = 1$. Tietokonearitmetiikassa syntyy laskennassa usein pyöristysvirheitä, jotka saattavat tuottaa myös konstruoitavaan kiertomatriisiin hieman virheellisiä alkioita, niin että edellä olevat ehdot, joita voidaan käyttää tarkistuksessa, eivät tarkkaan ottaen toteudu. Usein likiarvoistamalla itseisarvoltaan hyvin pienet luvut nolliksi virhe poistuu.

Liitteessä 1 olevan menetelmän laajempi selostus *Mathematica*n symbolisen ohjelmoinnin keinoin toteutettuna löytyy julkaisusta

Peisa 2005, http://www.kpeisa.fi/work/Quaternions%20AITC%202005.pdf.

Stokastisissa menetelmissä on tilanteita, esimerkiksi satelliitin kiertoradan simulointi, joissa Eulerin menetelmä tuottaa jatkuvasti hieman liian suuren arvon satelliitin korkeudelle eli virhe kumuloituu. Virhettä voidaan pienentää pienentämällä aika-askeleen pituutta. Mutta siinäkin tulee raja vastaan, jolloin simulaation esittäminen alkaa hidastua liikaa.

Edellä olevan kaltaisten ongelmatapausten yhteydessä voi käyttää paranneltuja menetelmiä kuten tässä esitettävää Runge-Kutta-Nyström (R-K-N) -menetelmää, joissa virhettä pyritään pienentämään ilman, että askelpituutta tarvitsee lyhentää.

Matemaattisesti tässä esitetty menetelmä on sama kuin yleisen *2. kertaluvun differentiaaliyhtälön*

$$y''(t) = f(t, y, y'), \qquad y(t_0) = y_0 \text{ ja } y'(t_0) = y'_0$$

numeerinen ratkaiseminen R-K-N -menetelmällä. Tämä menetelmä sopii suoraan etenevän liikkeen simuloimiseen sivulla 7 olevan algoritmin mukaisesti. 3D simulaatioissa tarkastellaan tällöin vektoriarvoisia funktioita differentiaaliyhtälössä

$$\bar{\boldsymbol{r}}''(t) = \bar{\boldsymbol{f}}(\bar{\boldsymbol{r}}, \bar{\boldsymbol{r}}') \qquad \text{eli} \qquad \bar{\boldsymbol{a}}(t) = \bar{\boldsymbol{f}}(\bar{\boldsymbol{r}}, \bar{\boldsymbol{v}}).$$

Vektoriarvoinen funktio $\bar{\boldsymbol{f}}(\bar{\boldsymbol{r}}, \bar{\boldsymbol{v}})$ palauttaa kappaleen kiihtyvyyden. Se voi sisältää sekä paikkaan $\bar{\boldsymbol{r}}$ että nopeuteen $\bar{\boldsymbol{v}}$ liittyviä lausekkeita. Kiihtyvyys määräytyy kappaleeseen vaikuttavista voimista, jotka voivat olla paikkaan (gravitaatiovoima) tai nopeuteen (ilmanvastus) liittyviä.

R-K-N -menetelmässä ikään kuin tunnustellaan tuntemattoman ratakäyrän $\bar{\boldsymbol{r}}(t)$ ja nopeuden $\bar{\boldsymbol{v}}(t)$ muutoksia siirryttäessä eripituisin askelein eteenpäin. Lopuksi muodostetaan tunnustelujen tuottamien arvioiden ”eräänlaiset” painotetut keskiarvot nopeuden muutokselle $d\bar{\boldsymbol{v}}$ ja paikan muutokselle $d\bar{\boldsymbol{r}}$. Sitaatit sen vuoksi, että nuo kertoimet määräytyvät tosiasiassa Taylorin polynomin mukaisesti.

Tunnetussa paikassa $\bar{\boldsymbol{r}}$ ja tunnetulla nopeudella $\bar{\boldsymbol{v}}$ ajanhetkellä t liikkuvalle kappaleelle lasketaan "tunnusteluarviot" aika-askeleella dt:

1.arvio $d\bar{\boldsymbol{r}}_1 = \bar{\boldsymbol{v}}\, dt$
$d\bar{\boldsymbol{v}}_1 = \bar{\boldsymbol{f}}(\bar{\boldsymbol{r}}, \bar{\boldsymbol{v}})\, dt$

2.arvio $d\bar{\boldsymbol{r}}_2 = (\bar{\boldsymbol{v}} + \frac{1}{2} d\bar{\boldsymbol{v}}_1)\, dt$
$d\bar{\boldsymbol{v}}_2 = \bar{\boldsymbol{f}}(\bar{\boldsymbol{r}} + \frac{1}{2} d\bar{\boldsymbol{r}}_1, \bar{\boldsymbol{v}} + \frac{1}{2} d\bar{\boldsymbol{v}}_1)\, dt$

3.arvio $d\bar{\boldsymbol{r}}_3 = (\bar{\boldsymbol{v}} + \frac{1}{2} d\bar{\boldsymbol{v}}_2)\, dt$
$d\bar{\boldsymbol{v}}_3 = \bar{\boldsymbol{f}}(\bar{\boldsymbol{r}} + \frac{1}{2} d\bar{\boldsymbol{r}}_2, \bar{\boldsymbol{v}} + \frac{1}{2} d\bar{\boldsymbol{v}}_2)\, dt$

4.arvio $d\bar{\boldsymbol{r}}_4 = (\bar{\boldsymbol{v}} + d\bar{\boldsymbol{v}}_3)\, dt$
$d\bar{\boldsymbol{v}}_4 = \bar{\boldsymbol{f}}(\bar{\boldsymbol{r}} + d\bar{\boldsymbol{r}}_3, \bar{\boldsymbol{v}} + d\bar{\boldsymbol{v}}_3)\, dt$

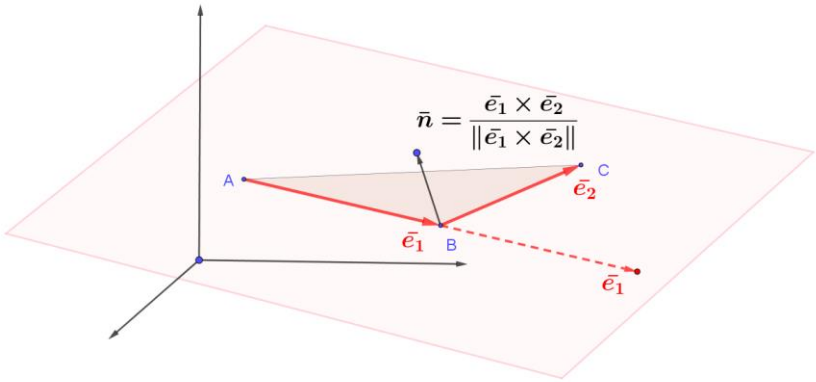
Lopuksi lasketaan muutokset

$$d\bar{\boldsymbol{v}} = \frac{1}{6}(d\bar{\boldsymbol{v}}_1 + 2d\bar{\boldsymbol{v}}_2 + 2d\bar{\boldsymbol{v}}_3 + d\bar{\boldsymbol{v}}_4)$$
$$d\bar{\boldsymbol{r}} = \frac{1}{6}(d\bar{\boldsymbol{r}}_1 + 2d\bar{\boldsymbol{r}}_2 + 2d\bar{\boldsymbol{r}}_3 + d\bar{\boldsymbol{r}}_4)$$

Koko edellinen laskenta voidaan antaa erillisen funktion suoritettavaksi, jolle argumenttina annetaan dt, $\bar{\boldsymbol{r}}$ ja $\bar{\boldsymbol{v}}$. Myös funktio $\bar{\boldsymbol{f}}(\bar{\boldsymbol{r}}, \bar{\boldsymbol{v}})$, joka laskee kiihtyvyyden, voidaan toteuttaa erillisenä itsenäisenä funktiona, jolloin R-K-N -funktiosta tulee suoraan yllä olevan määrittelyn mukainen esitys.

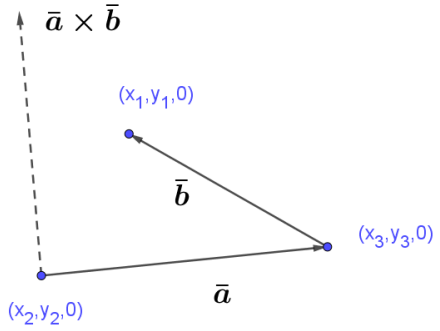
Liite 3 Kierron suunnan määrittäminen tason pisteiden kautta

Tason voi määritellä yksikäsitteisesti 3D avaruudessa yhden tason tunnetun pisteen ja tason normaalin avulla. Tason $\langle \bar{\boldsymbol{n}}, d \rangle$ voi määritellä myös vektoreiden ristitulolla käyttäen kolmea tason pistettä A, B ja C, jotka eivät ole samalla suoralla. Oheisen kuvan mukaisesti lasketaan ensin yksikkönormaali $\bar{\boldsymbol{n}}$, jonka jälkeen lasketaan vakio d pistetulolla jonkin kärkipisteen kanssa $d = \bar{\boldsymbol{n}} \cdot \bar{\boldsymbol{A}}$.



Mutta kumpaan suuntaan tason normaali pitää asettaa osoittamaan eli kumpi on tason etupuoli ja kumpi takapuoli? Tämä ongelma nousee vastaan esimerkiksi tilanteessa, jossa kappale törmää tasoon, jota edellä oleva kolmio edustaa. Törmäyksessä kappaleet usein tunkeutuvat tason toiselle puolelle. Miten tietokoneen ohjelma voi tietää, kummalle puolelle tason normaali osoittaa? Vastaus on, ettei mitenkään pelkästään tason pisteiden avulla. Me emme voi päätellä 3-ulotteisessa avaruudessa lueteltujen monikulmion pisteiden avulla, mihin suuntaan asetettu normaali osoittaa, ellemme samalla tiedä, ovatko pisteet lueteltu normaaliin nähden myötäpäivään vai vastapäivään.

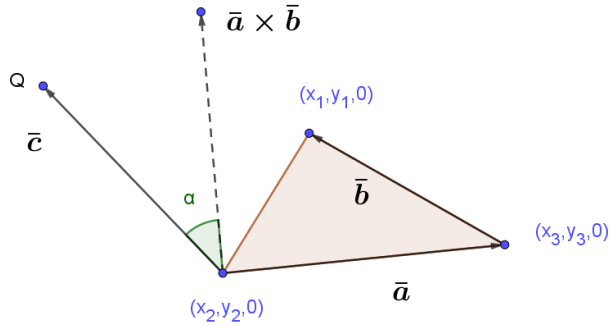
2-ulotteisessa avaruudessa katselemme aina tasoa vain yhdeltä suunnalta eli ikään kuin ”ylhäältä”. Tällöin voimme selvittää laskemalla, onko kolme ei-samalla-linjalla olevaa pistettä lueteltu myötäpäivään vai vastapäivään. Tämä voidaan perustella esimerkiksi vektoreiden ristitulon avulla. Koska tarkastelemme 2D pisteitä, on meidän kuitenkin lisättävä kolmas ulottuvuus, jotta voimme soveltaa ristituloa. Pisteisiin lisätään 0 kolmanneksi koordinaatiksi, jolloin käsittelemme edelleen samaa tasoa mutta 3-ulotteisen avaruuden aliavaruutena. Lasketaan ristitulo peräkkäisten pisteiden muodostamille vektoreille samaan suuntaan kierrettynä.



$$\bar{\boldsymbol{a}} \times \bar{\boldsymbol{b}} = (0,0,-x_2y_1 + x_3y_1 + x_1y_2 - x_3y_2 - x_1y_3 + x_2y_3) =$$
$$= (0,0,(x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1))$$

Edellä ristitulon 3. koordinaatti, joka voidaan esittää sivujen päätepisteiden koordinaattien erotuksien ristitulona, on positiivinen (oikean käden koordinaatistossa), jos ristitulovektori osoittaa ylös, jolloin pisteet on lueteltu vastapäivään. Jos 3. koordinaatti on negatiivinen, osoittaa ristitulovektori alas ja pisteiden järjestys on annettu myötäpäivään.

3-ulotteisessa avaruudessa me emme voi määritellä kolmion kärkipisteiden määrittämän kierron suuntaa, koska tasoa voidaan katsoa kahdesta vastakkaisesta suunnasta, jolloin saamme päinvastaiset tulokset. Mutta jos kiinnitetään piste, josta kolmea pistettä katsotaan (piste Q kuvassa, esimerkiksi tason kohtaava kappale), voidaan kolmen ei-samalla-linjalla olevan pisteen muodostaman kierron suunta määritellä. Tämä tapahtuu esimerkiksi skalaarikolmitulolla.



$$\bar{\boldsymbol{c}} \cdot \bar{\boldsymbol{a}} \times \bar{\boldsymbol{b}} > 0 \Leftrightarrow \text{kulma } \alpha \text{ terävä } (< 90°),$$

Mikäli kuvanmukaisesti kolmen pisteen järjestys on vastapäivään, osoittaa ristitulovektori ”ylös” kolmion tasosta. Tällöin piste Q sijaitsee samalla puolella kuin ristitulovektori, jos ja vain jos ristitulovektorin ja vektorin $\bar{\boldsymbol{c}}$ välinen kulma on terävä, jolloin pistetulo > 0 eli skalaarikolmitulo on positiivinen.

Edellinen merkitsee toisaalta myös sitä, että jos pisteiden lisäksi tiedetään tallennettu pisteiden kiertosuunta asetettuun normaaliin nähden, voidaan siitä päätellä normaalin suunta pisteeseen Q nähden ja se, sijaitseeko piste Q kolmion edustaman pinnan etu- vai takapuolella.

Kolmioiden muodostaman mesh-rakenteen, jolla kappaleet tyypillisesti ympäröidään 3D tietokonegrafiikassa, käsittely onkin melkoista kirjanpitoa. Kolmioiden kärkipisteiden lisäksi tallennetaan paljon muutakin tietoa mm. se, *onko pisteet lueteltu vastapäiväisessä vai myötäpäiväisessä järjestyksessä* suhteessa asetettuun normaaliin. Mesh-rakenteen käsittely on kuitenkin tämän dokumentin tarkastelun ulkopuolella.