Sisältö

1 Johdanto

2 Enzo

Enzo on astrofysikaalisten fluidien simulointiin käytettävä ilmainen ja avoin simulaatiokoodi, joka on suunniteltu kosmologisten rakenteiden simulointiin. Se tukee muun muassa hydrodynamiikkaa, ideaalista ja epäideaalista magnetohydrodynamiikkaa, N kappaleen simulaatioita, kaasupilvien kemiaa, säteilynkuljetusta, tähtien syntyä sekä maailmankaikkeuden laajenemista. [7]

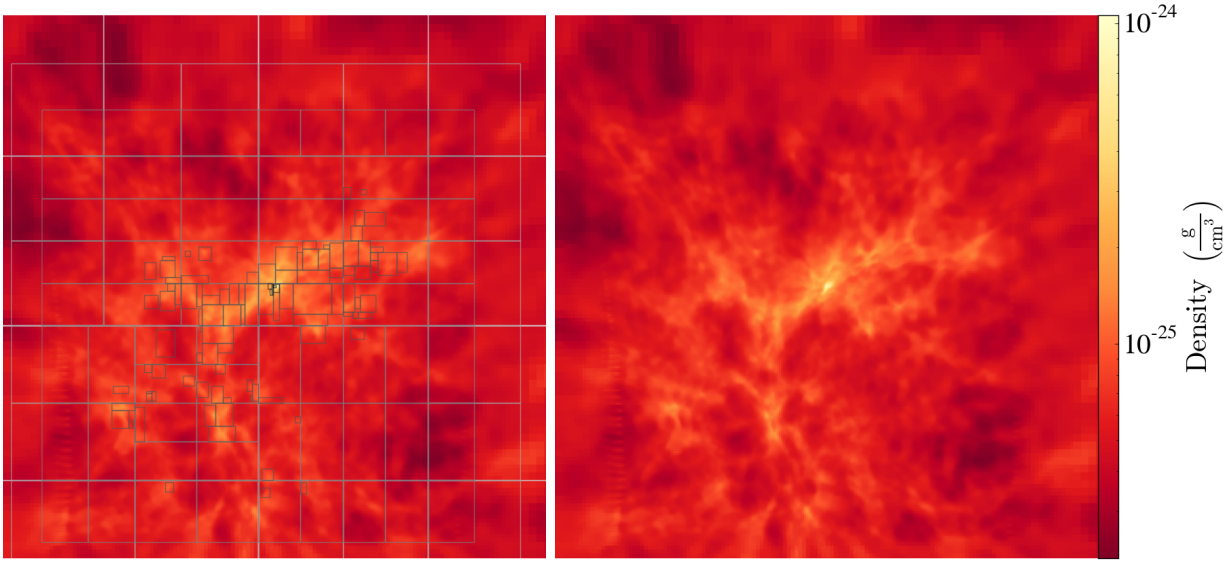
Enzo hyödyntää mukautuvaa hilantihennystä (*Adaptive Mesh Refinement*, AMR), joka mahdollistaa aika- ja paikkaresoluution kasvattamisen simulaation kiinnostavilla alueilla. Tämä on tärkeää, sillä usein melko pienellä alueella tarvitaan suurta resoluutiota, mutta koko simulaation ajaminen näin suurella tarkkuudella veisi kohtuuttoman paljon aikaa. Kullekin paikalle valitaan sopiva resoluutio automaattisesti käyttäjän määrittelemien ehtojen mukaan. [7]

2.1 Hila

Simuloitava alue katetaan kokonaan hilalla, jonka tiheys valitaan sellaiseksi, että saavutetaan pienin haluttava resoluutio. Tämä niinkutsuttu juurihila toimii juurena muiden hilojen muodostamalle hierarkialle, joka muodostuu, kun juurihilasta valitaan alueita simuloitavaksi korkeammalla resoluutiolla.[7]

Kaikki hilat ovat karteesisia ja suorakulmaisia. Alueille, joilla tarvitaan korkeampaa resoluutiota, asetetaan juurihilan kanssa päällekkäin toinen, hienompi hila. Näitä sisäkkäisiä hiloja voi tarvittaessa olla teoriassa mielivaltaisen monta. Kuvassa 1 on selvästi nähtävillä hilojen mukauttaminen tutkittavaan alueeseen: tiheillä tai muuten kiinnostavilla alueilla kätetään pienempiä ja tiheämpiä hiloja. [7]

Kullakin hilalla juurihilaa lukuun ottamatta on vanhempi, joka sisältää hilan kokonaan. Yhdellä hilalla voi olla useita lapsia, mutta aina vain yksi vanhempi. Näin hilat muodostavat puumaisen rakenteen. Tavallisesta puita koskevasta nimeämiskäytännöstä poiketen sisaruksiksi kutsutaan kaikkia niitä hiloja, joiden resoluutio on sama eli ne sijaitsevat puussa samalla tasolla. [7]



Kuva 1: Läpileikkaus 100 pc^2 alueesta Enzo-simulaatiosta (julkaisua Regan et al. 2015 varten), jonka hilojen rajat näkyvillä (vasen kuva) sekä ilman niitä (oikea kuva). Kiinnostavammilla alueilla solut ovat pienempiä. Resoluution heikentyminen simulaation reuna-alueilla sijaitsevissa suuremmissa ja harvemmissa hiloissa on selvästi nähtävissä oikeanpuoleisessa kuvassa.

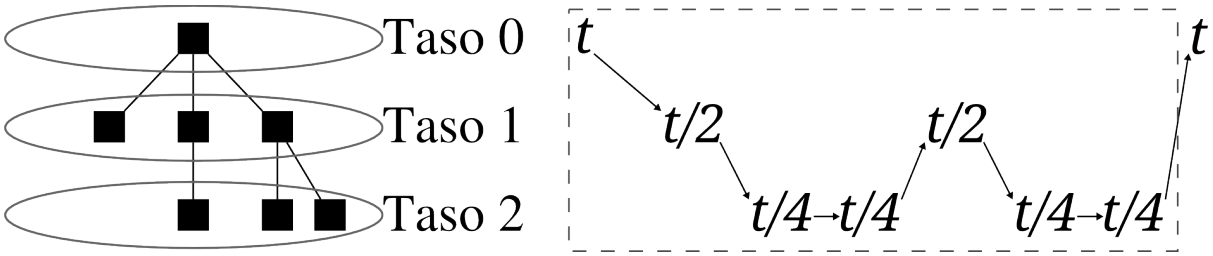
Hienompia hiloja luotaessa valitaan hilan solujen koko siten, että hila rajoittuu reunoistaan vanhempansa solujen tahkoihin. Lisäksi solun vanhemman sivun pituuden tulee olla jokin monikerta solun sivun pituudesta. [7]

Kukin hila koostuu varsinaisen datan tallettavien aktiivisten vyöhykkeiden (*active zone*) lisäksi haamuvyöhykkeistä (*ghost zone*), joita käytetään laskennassa tarvittavien aktiivisten vyöhykkeiden arvojen päivittämiseen tarvittavien naapurisolujen varastoimiseen väliaikaisesti. Hydrodynamiikkaa varten haamuvyöhykkeitä on kolme kerrosta kullakin aktiivisen vyöhykkeen tahkolla. Haamuvyöhykkeiden arvot saadaan interpoloimalla hilan vanhemmasta tai kopioimalla sisaruksista.[8, 7]

2.2 Numeerinen ratkaiseminen

Enzo tarkastelee kutakin hilaa omana ongelmanaan ratkaisten tarvittavat yhtälöt kullekin hilalle erikseen käyttäen reunaehtoina haamuvyöhykkeistä saatavia arvoja. Aluksi määritetään halutun tarkkuuden saavuttamiseksi tarvittava aika-askel kullekin puun tasolle. Tämän jälkeen aletaan tasoja käydä läpi W-syklin mukaisesti (kuva 2). [7]

W-syklissä kunkin tason kaikille hiloille lasketaan aluksi niiden tila yhden aika-askeleen kuluttua. Laskeminen aloitetaan juurihilasta se etenee tasoittain, kunnes kaikki AMR-



Kuva 2: Vasemmalla kolmitasoinen hierarkia hiloja ja oikealla niiden läpikäyntijärjestys W-syklissä ja aika-askeleiden pituudet kun oletetaan, että tason $l+1$ aika-askel on puolet tason l aika-askeleen pituudesta ja tason 0 aika-askeleen pituus on t. Katkoviivalla on erotettu vaiheet, joiden jälkeen kunkin tason hilat ovat edenneet ajan t verran eli ollaan jälleen alkutilannetta vastaavassa tilanteessa, jolloin sykli alkaa uudelleen.

puun lehdet on käsitelty. Seuraavaksi puun alimmalla tasolla olevia hiloja edistetään niin monta aika-askelta kuin tarvitaan, jotta saavutetaan ylempi taso, jonka hiloja edistetään myös yhden aika-askeleen verran. Tämän jälkeen alimman tason hiloja edistetään taas, kunnes ylempi taso on jälleen saavutettu. [7]

Hilojen edistämistä jatketaan näin, edistäen aina tason l hiloja, kunnes ne saavuttavat tason $l-1$ hilat, jolloin tason $l-1$ hiloja edistetään jälleen yhden aika-askeleen verran. Näin seuraava aika-askel lasketaan aina sille tasolle, jolla aikaa simulaation alusta on kulunut vähiten. Kun hiloja on edistetty niin pitkälle, että on jälleen aika edistää tason 0 hilaa, sykli alkaa alusta. [7]

3 yt

yt on avoimen lähdekoodin Python-paketti, joka on mahdollistaa simulaatiotulosten helpon lukemisen, analysoinnin ja visualisoinnin. Kehitystyön alkuvaiheessa se sopi käytettäväksi vain Enzo-simulaatioiden kanssa, mutta nykyään sillä voidaan suoraan lukea sekä muiden AMR-simulaatioiden (kuten RAMSES tai BoxLib) että N kappaleen simulaatioiden (esimerkiksi Gadget) tuloksia. [9]

3.1 Käyttäjän ja simulaatiodatan välinen rajapinta

yt abstrahoi hyvin voimakkaasti lukemansa datan, jolloin käyttäjä voi keksittyä fysikaalisiin rakenteisiin, joita simulaatio edustaa. Kun käyttäjä on ladannut simulaatiodatan, voi siitä tarkastella esimerkiksi pallomaista aluetta antamalla alueen keskustan koordinaatit ja pallon säteen ilman, että käyttäjän tarvitsee esimerkiksi etsiä niitä hiloja, jotka kattavat alueen parhaalla mahdollisella resoluutiolla tai tietää, missä tiedostoissa data on tallennettuna. Lisäksi yt huolehtii yksikkömuunnoksista. [9]

Koska dataa käsitellään abstrakteina olioina, voidaan samaa ohjelmaa käyttää myös eri simulaatioista saadun datan kanssa vaihtamalla luettavaa datasettiä. Esimerkiksi alla oleva koodisegmentti lataa sijainnissa ”data” olevan simulaation muistiin ja plottaa sen jälkeen sivultaan 500 kpc olevan alueen poikkileikkauksen tiheyden ja tallentaa sen. Ohjelma ei ota kantaa luettavan datan muotoon, ja onkin siksi helposti käytettävissä minkä tahansa yt:n tukeman datasetin kanssa ainoastaan vaihtamalla datasettiä ladattaessa annettavaksi parametriksi halutun datan sijainti. [9, 1]

```
1 import yt
2 dataset = yt.load("data")
3 plot = yt.SlicePlot(dataset, "x", "density", width = (500, "kpc
    ↪ "))
4 plot.save("slice.png")
```

Simulaatiodatassa olevien tietojen lisäksi yt pystyy laskemaan datasta myös monia muita suureita kuten vaikkapa kulmaliikemäärän, maksimi- ja minimiarvoja sekä niiden sijainteja tai massakeskipisteen sijainnin. Lisäksi käyttäjän on mahdollista luoda omia kenttiä.

Alla on yt:n dokumentaatiosta[1] mukailtu esimerkki koodinpätkästä, jolla lisättäisiin paine yt:n tuntemien kenttien joukkoon. Uudelle kentälle määritellään nimen ja funktion lisäksi myös yksikkö. [9, 2]

```
1 def _pressure(field, data):
2     return (data.ds.gamma - 1.0) * data["density"] * data["
      ↪ thermal_energy"]
3 yt.add_field("pressure", function=_pressure, units="dyne/cm**2"
  ↪ )
```

Laskettujen suureidin laskemisen jälkeen yt tarjoaa laajan valikoiman työkaluja tulosten visualisointiin. Aiemmin esiteltyjen halkileikkausten lisäksi yt:llä voi luoda muun muassa erilaisia profiileja, painottamattomia tai painotettuja projektioita tai 3D-renderöintejä. [9]

3.2 Rinnakkaistaminen

Usein visualisoitavaa dataa on hyvin paljon ja tietokoneiden kehittyessä sen määrä edelleen lisääntyy, jolloin rinnakkaislaskennan käyttö myös datan analysoinnissa tulee tärkeämmäksi ja tärkeämmäksi. yt käyttää mpi4py-moduulia mahdollistaakseen datan rinnakkaisen käsittelyn niissä tehtävissä, jotka ovat helposti rinnakkaistettavissa. Tälläisiä ovat esimerkiksi tehtävät, joissa simuloitava alue voidaan jakaa eri prosessorien kesken. Muun muassa projistointi on mahdollista toteuttaa siten, että kukin prosessori laskee tietyn joukon näkösäteitä, jotka lopuksi yhdistetään yhdeksi plotiksi.[9]

Käyttäjä voi ottaa rinnakkaistuksen käyttöön ohjelmassaan yksinkertaisesti lisäämällä ohjelmansa alkuun rivin `yt.enable_parallelism()`. Lisäksi ohjelman suorittamiseen on luonnollisesti käytettävä mpirun-komentoa. Tällöin yt osaa rinnakkaistaa esimerkiksi projektioiden, poikkileikkausten, profiilien ja 3D-renderöintien luonnin sekä halojen etsimisen. Tarvittaessa joitain operaatioita voidaan suorittaa sarjallisesti esimerkiksi tarkastamalla funktion `yt.is_root()` palauttama arvo, joka palauttaa arvon `True` jos kutsuvalla prosessorilla on MPI rank 0, muuten `False`. [9, 4]

yt pystyy käsittelemään myös useita datasettejä tai objekteja rinnakkain. Käyttämällä jokerimerkkejä kuten * ja ? dataa ladattaessa, voidaan kerralla ladata useita datasettejä tai objekteja käsiteltäväksi rinnakkain. Tämän jälkeen ne voidaan käydä läpi käyttäen `piter()`-funktiota, joka jakaa käsiteltävät oliot prosessorien kesken. [4]

[1] http://yt-project.org/doc/developing/creating_derived_fields.html

3.3 Visualisoinnin upottaminen simulaatiokoodiin

Vaikka simulaation tila voidaan tarvittaessa tallentaa vaikka jokaisen aika-askeleen jälkeen, on prosessi hidas ja vaatii suuria tai pitkäkestoisia simulaatioita ajettaessa kohtuuttoman paljon levytilaa. Python/C API mahdollistaa muistissa olevan datan antamisen Python-tulkille simulaation sisällä. `yt` puolestaan tarjoaa API:n jolla simulaation informaatio voidaan välittää analyysipaketille ja `yt`:tä voidaan käyttää ajossa olevan simulaation analysointiin. [9]

Kun simulaation outputteja ei tarvitse tallentaa visualisointia varten, voidaan kuvia tai plotteja tallentaa huomattavasti useammin ja saavutetaan huomattavasti parempi tarvittavan tallennustilan ja tallennetun hyödyllisen informaation määrän suhde, sillä simulaation outputin koko liikkuu usein gigatavujen luokassa kun taas esimerkiksi kuvat ovat vain joitain kymmeniä tai satoja kilotavuja. Tyypillisesti dataa redusoidaan voimakkaasti joka tapauksessa, joten alkuperäisen outputin tallentaminen ei välttämättä ole mielekästä niin usein kuin esimerkiksi sulavan animaation tuottamiseksi on tarpeen. [9]

3.4 Kuvien luominen

`yt` piilottaa paljon plottien luomiseen tarvittavista esimerkiksi datan tallennusmuotoon tai numeeristen arvojen väreiksi muuttamiseen liittyvistä toimista. On kuitenkin ohjelman suorituskyvyn ja tuotettujen kuvien ymmärtämisen kannalta tärkeää ymmärtää jonkin verran myös `yt`:n toimintaa. Alla on eritelty muutamien kuvien luomisen mekaniikkaa hieman käyttäjälle näytettävää pintaa syvemmältä.

3.4.1 Läpileikkaukset ja projektiot

Läpileikkausten ja projektioiden luominen `yt`:llä on pääpiirteissään melko samanlaista, tärkeimpänä erona käsiteltävän data-alueen laajuus. Läpileikkausten luominen on tyypillisesti melko nopeaa, sillä niiden tekemiseksi täytyy käydä läpi vain ohut viipale datasetistä. Plotti luodaan hakemalla ensin tarvittava data datasetistä suurimmalla mahdollisella resoluutiolla `FixedResolutionBuffer`-luokkaan (FRB), josta muodostetaan kuva käyttäjän määrittelemästä kohdasta. Samaa FRB:tä voidaan käyttää useiden kuvien renderöintiin mikäli kuvaa halutaan esimerkiksi zoomata tai panoroida. Tämä nopeuttaa kuvien luomista hieman. [3]

Projektioiden luominen `yt`:llä onnistuu pääpiirteissään samalla tavoin kuin läpileikkaus-

tenkin. Projektiota luotaessa täytyy käydä läpi koko näkösäde kutakin lopullisen kuvan pikseliä kohden, joten niiden luominen on tyypillisesti hieman hitaampaa kuin läpileikkausten. Ne tuovat kuitenkin usein läpileikkauksia paremmin esiin näkösäteen suunnassa laajempia rakenteita. Kuten läpileikkauksiakin, myös projektioita voi tehdä myös muissa kuin simulaation koordinattiakselien suunnissa käyttäen `OffAxisProjectionPlot`-luokkaa. [3]

Projektiota luotaessa pikselien arvot voidaan määrittää usealla eri tavalla. Yleisimpiä näistä ovat painotettu ja painottamaton integrointi, joka määritetään asettamalla `method=integrate`. Mikäli `weight_field`-parametria ei ole asetettu, lasketaan painottamaton keskiarvo yhtälön 1 mukaisesti integroimalla haluttu kenttä $f(x)$ näkösäteen $\hat{n}$ suunnassa tutkittavan alueen yli. Tällöin projisoidun kentän yksikkö on alkuperäisen kentän yksikkö kerrottuna pituusyksiköllä. Mikäli `weight_field` on määritelty, lasketaan kentän painotettu keskiarvo yhtälön 2 mukaisesti. Tällöin myös yksikkö säilyy projisoinnissa samana. [3]

$$g(X) = \int f(x)\hat{n} \cdot dx \tag{1}$$

$$g(X) = \frac{\int f(x)w(x)\hat{n} \cdot dx}{\int w(x)\hat{n} \cdot dx} \tag{2}$$

Muita mahdollisia projisointitapoja ovat ovat `mip` ja `sum`. Näistä ensimmäinen valitsee projisoitavan kentän maksimiarvon kullakin näkösäteellä ja jälkimmäinen integroinnin sijaan summaa kentän arvot näkösäteellä ottamatta huomioon kuljetun polun pituutta. Siksi `sum` sopiikin käytettäväksi vain sellaisten gridien kanssa, joiden solut ovat vakiokokoisia, sillä muuten syntyvä kuva saattaa olla hyvin epäfysikaalinen. Kumpikin säilyttää alkuperäisen kentän yksiköt. [3, 6]

3.4.2 3D

`yt` tarjoaa myös mahdollisuuden kolmiulotteisten rakenteiden tarkastelemiseen tilavuusrenderöimällä. Kuten kaksiulotteisissakin ploteissa myös tilavuusrenderöinnissä voidaan käyttää mitä tahansa simulaatiodatan kenttää, jolloin esimerkiksi katselusuuntaa kääntämällä tai zoomaamalla kohti kiinnostavia alueita voidaan simulaation kiinnostavia piirteitä tuoda esiin joissain tilanteissa havainnollisemmin kuin läpileikkauksilla tai projektioilla. [5]

Tilavuusrenderöintiä varten on ensin luotava color transfer function, jonka perusteella määritetään tarkasteltavan tilavuuden emissio ja absorptio kussakin RGBA-värijärjestelmän kaistassa. Ne voi määrätä mikä tahansa simulaation kenttä joko painottamat-

tomana tai painotettuna toisella kentällä. Täten color transfer function määrää kunkin pisteen värin paikan funktiona siten, että $f(v) \rightarrow (r, g, b, a)$. [5]

Seuraavaksi luodaan `camera`-olio, jolloin datalohkot jaetaan kuvattavan alueen täysin kattaviin "tiiliin" (*bricks*), joihin valitaan parhaan mahdollisen resoluution data jokaisesta pisteestä. Nämä tiilet järjestetään näkösäteen suunnassa takaa eteen, jotta ne on helppo käydä läpi siinä järjestyksessä, jossa silmään saapuva säde kohtaa ne. Seuraavaksi luodaan kuvatason kanssa yhdensuuntainen taso säteitä kuvattavan alueen perälle alkuarvolla nolla. Näitä säteitä liikutetaan kohti katsojaa ja pikselien arvot päivitetään kullakin askeleella. [5]

Kullekin tiilelle lasketaan, mitkä säteet leikkaavat sen. Kustakin säteestä otetaan `sub_samples`-parametrilla määrättävä määrä näytteitä (oletusarvoisesti 5) ja arvot näytteenottokohdissa lasketaan solun kärkien arvoista trilineaarisella interpoloinnilla. Tämän jälkeen kussakin pisteessä lasketaan transfer function arvo, jolloin saadaan emission j ja absorption α voimakkuus kussakin kaistassa. Näiden perusteella kullekin pikselille lasketaan uusi arvo ottaen huomioon säteen kulkema matka. Pikselin uusi arvo v^{n+1} kaistassa i saadaan pikselin aiemman arvon v^n perusteella yhtälöstä 3, jossa Δs on säteen kulkema matka näytteenottopisteiden välillä.

$$v_i^{n+1} = j_i \Delta s + (1 - \alpha_i \Delta s) v_i^n \tag{3}$$

4 Oma työ

Aloitin `yt`:hen tutustumisen selaamalla dokumentaatiota sekä cookbookia[2] ja ensin kopioimalla ja sittemmin muokkaamalla siellä annettuja malliohjelmia käyttäen esimerkkidataa. Tutustuin tarkimmin halkileikkauksiin, projektioihin ja 3D-renderöinteihin sekä muun muassa niiden akselien tekstien ja jaotuksen tai väriskaalojen muokkaamiseen. Lisäksi kokeilin `yt`:n soveltamista animaatioiden tekemiseen.

Opittuani perusasiat aloin soveltaa niitä tutkimusryhmämme tuottamaan dataan kaasupilven romahtamista suoraan mustaksi aukoksi koskevista simulaatioista (Regan et al. 2015). Samalla otin selvää `yt`:n tarjoamista mahdollisuuksista usean plotin yhdistämiseksi yhteen kuvaan. Kun samaan kuvaan halutaan useantyyppisiä kuvia, tarvitaan `eps_writer`-luokkaa. Sen tarjoama tuki profiileille oli kuitenkin puutteellinen, joten jouduin myös parantelemaan `yt`:n lähdekoodia.

4.1 Läpileikkaukset

Aloitin tutustumalla `yt`:n yksinkertaisimpiin plotteihin aloittaen läpileikkauksista. Niitä voidaan käyttää yksittäisinä näyttämään jokin alue simulaatiosta tai niitä voidaan renderöidä useita ja koostaa niistä animaatio. Esimerkiksi listing 1 lukee annetun datasetin ja tallentaa läpileikkauksen tiheyksistä simulaation tiheimmän kohdan ympäriltä sadasta kohdasta liikkuen z-akselia pitkin 1 kpc matkan. Esimerkissä tarkastellaan kohdetta koordinaattiakselin suunnasta, mutta läpileikkauksia voidaan luoda mielivaltaisesta suunnasta [3].

Lisäksi plotin colorbarin alueeksi asetetaan 3×10^{-26} – 2.5×10^{-25} $\frac{\mathrm{g}}{\mathrm{cm}^3}$ ja colormapiksi hot[3]. Lisäksi akseleilla käytettävän fontin kokoa kasvatetaan ja lopullinen kuva tallennetaan juoksevaa numerointia käyttäen. Neljä sadasta tuloksena saatavista plotista tasaisin välein valittua läpileikkausta on nähtävissä kuvassa 3.

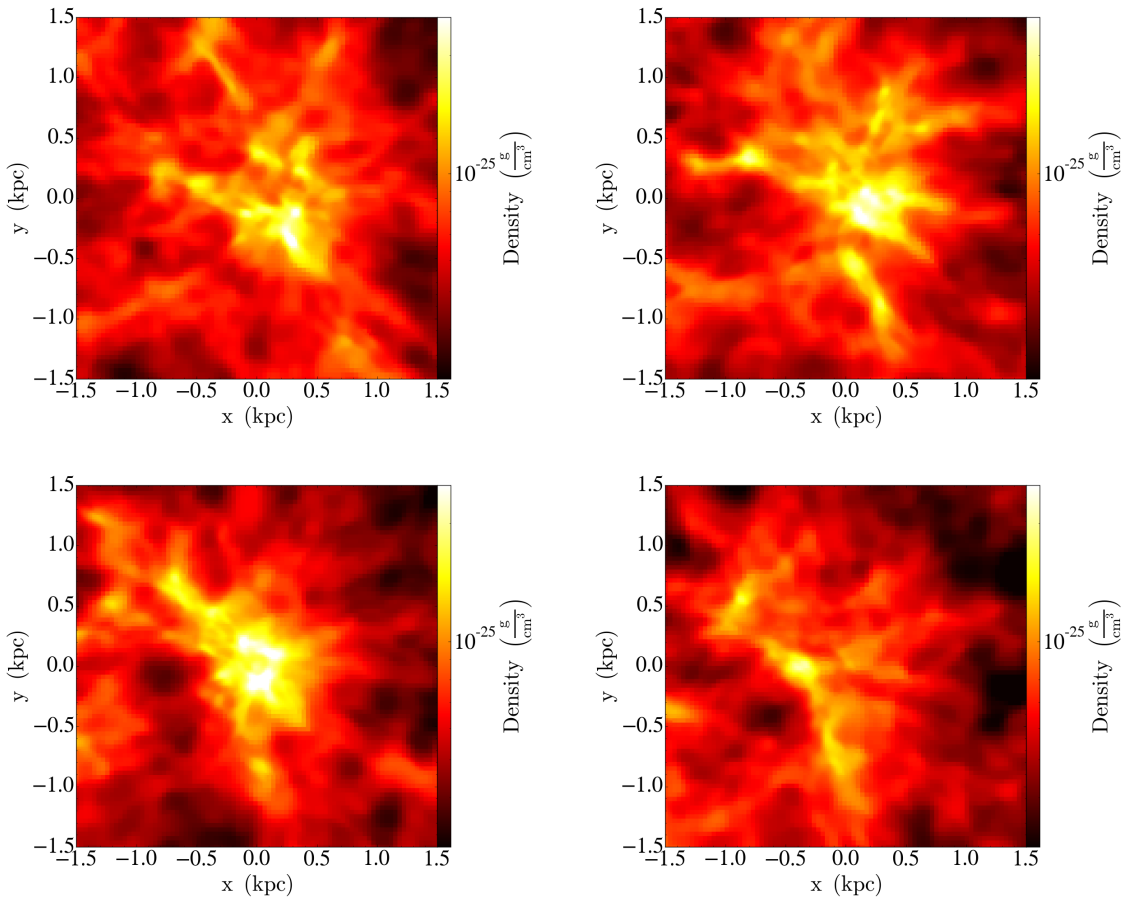
[2] `http://yt-project.org/docs/3.1/cookbook/`
[3] `http://yt-project.org/doc/visualizing/colormaps/`

Listing 1: Python-ohjelma, joka tallentaa 100 esimerkiksi animoitavaksi sopivaa framea, joissa sivultaan 3 kpc oleva läpileikkaus liikkuu simulaation z-akselin suunnassa läpi simulaation tiheimmän kohdan.

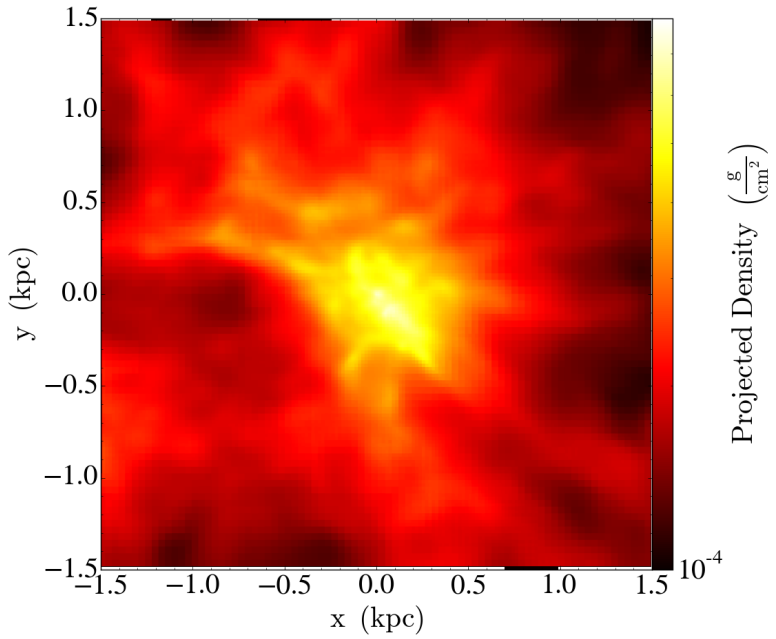
```
1  # -*- coding: utf-8 -*-
2  import yt
3  import numpy as np
4
5  ds = yt.load("data/dataset")
6  v, maxtiheys = ds.find_max("density")
7  frame = 0
8
9  for i in np.arange(-0.5, 0.5, 0.01):
10     siirto = yt.YTArray([0, 0, i], 'kpc')
11     keskusta = maxtiheys - siirto
12
13     image = yt.SlicePlot(ds, 'z', center=keskusta, fields=['
           ↪ density'], width=(3, 'kpc'))
14     image.set_zlim('density', 3e-26, 25e-26)
15     image.set_cmap("density", "hot")
16     image.set_font_size(35)
17     image.save("kuvat/flythrough/%04i.png" %frame)
18     frame+=1
```

4.2 Projektiot

Listingissä 2 on esitetty yksinkertaisen projektion tallentava skripti. Se käyttää samaa simulaatiodataa kuin 1, josta luodaan yksi projektio integroimalla z-akselin suuntaisen näkösäteen suunnassa. Skriptin tuottama kuva on nähtävissä kuvassa 4. Vertaamalla tätä kuvan 3 läpileikkauksiin huomataan, että projisointi kadottaa näkösäteen suunnassa pienet yksityiskohdat, mutta projektiosta saa paremman kuvan suuremman skaalan rakenteista. Myös yksiköiden huomataan poikkeavan toisistaan, sillä projektiota luotaessa käytettiin painottamatonta integrointia.



Kuva 3: Neljä listingissä 1 esitetyn koodin tallentamista kuvista, kun se ajetaan julkaisua Regan et al. 2015 varten lasketulla datalla. Alueessa edetään syvemmälle riveittäin vasemmalta oikealle.



Kuva 4: Listingin 2 luoma projektio. Verrattuna kuvan 3 läpileikkauksiin nähdään vähemmän yksityiskohtia, mutta projektio antaa paremman kokonaiskuvan koko alueesta kerralla.

Listing 2: Yksinkertaisen projektion luomiseen soveltuva ohjelma. Käytetty alue datassa on sama kuin listingissä 1, mutta useiden läpileikkausten sijaan luodaan yksi projektio.

```
1  # -*- coding: utf-8 -*-
2  import yt
3
4  ds = yt.load("data/dataset")
5  v, keskusta = ds.find_max("density")
6  vasen_kulma = keskusta + yt.YTArray([-1.5, -1.5, -0.5], 'kpc')
7  oikea_kulma = keskusta + yt.YTArray([1.5, 1.5, 0.5], 'kpc')
8  region = ds.box(vasen_kulma, oikea_kulma)
9
10 plot = yt.ProjectionPlot(ds, 'z', fields=['density'], center='
       ↪ max', data_source = region, width=(3, 'kpc'))
11 plot.set_zlim("density", 8e-4, 1e-4)
12 plot.set_cmap("density", "hot")
13 plot.set_font_size(30)
14 plot.save("kuvat/projection.png")
```

4.3 3D

Tilavuusrenderöinnit tuovat usein tutkittavan kohteen kolmiulotteisen rakenteen esiin projektioita ja läpileikkauksia intuitiivisemmin. Yksittäiset kuvat saattavat kuitenkin olla

vaikeasti tulkittavia, sillä kohteiden etäisyyttä katsojasta on usein vaikeaa hahmottaa. Kuvista ja etenkin animaatioista on kuitenkin mahdollista saada erittäin näyttäviä.

Liitteessä A on esitetty skripti, jolla voidaan luoda framet animaatioon, jossa kamera sekä kiertää kohdetta että zoomaa sisään. Datan lataamisen jälkeen riveillä 10–13 etsitään datan tiheimmästä kohdasta korkeintaan 0,5 kpc päässä oleva paikka, jossa moelkulaarisen vedyn osuus kaikesta vedystä on suurin hyödyntämällä `yt`:n tarjoamia `find_max`- ja `max_location`-funktioita.

4.4 eps_writer

4.5 Muutokset yt:n lähdekoodissa

5 Tulokset

tulostan tähän tuloksia

6 Loppupäätelmät

—

Viitteet

[1] The cookbook. http://yt-project.org/doc/cookbook/. Accessed: 7.7.2015.

[2] Creating derived fields. http://yt-project.org/doc/developing/creating_derived_fields.html. Accessed: 27.7.2015.

[3] How to make plots. http://yt-project.org/doc/visualizing/plots.html. Accessed: 17.7.2015.

[4] Parallel computation with yt. http://yt-project.org/doc/analyzing/parallel_computation.html. Accessed: 7.7.2015.

[5] Volume rendering: Making 3d photorealistic isocontoured images. http://yt-project.org/doc/visualizing/volume_rendering.html. Accessed: 24.7.2015.

[6] yt api: yt.visualization.plot_window.projectionplot. http://yt-project.org/doc/reference/api/generated/yt.visualization.plot_window.ProjectionPlot.html#yt.visualization.plot_window.ProjectionPlot. Accessed: 20.7.2015.

[7] G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, Y. Li, and Enzo Collaboration. ENZO: An Adaptive Mesh Refinement Code for Astrophysics. *ApJS*, 211:19, April 2014.

[8] Brian W O'Shea, Greg Bryan, James Bordner, Michael L Norman, Tom Abel, Robert Harkness, and Alexei Kritsuk. Introducing enzo, an amr cosmology application. *arXiv preprint astro-ph/0403044*, 2004.

[9] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman. yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data. *ApJS*, 192:9, January 2011.

A 3D-animaatio spiraaliradalla kohdetta lähestyen

```
# -*- coding: utf-8 -*-
import yt
import numpy as np
import math

ds = yt.load("/data/scratch3/extragal/Enzo/data/LWIR/RD0076/RD0076")
ad = ds.all_data()

# etsitään tihein kohta ja tiheimmän kohdan ympäristöstä suurin H2-pitoisuus
v, tiheysmaksimi = ds.find_max("density")
densSphere = ds.sphere(tiheysmaksimi, (0.5, "kpc"))
maxlocs = densSphere.quantities.max_location(("gas", "H2_fraction"))
H2max = maxlocs[2:5]

# etsitään suurimman H2-pitoisuuden sijainnin ympärillä 1 kpc sisältä suurin ja pienin H2-pitoisuus ja luodaan niiden logaritmien avulla ColorTransferFunction
H2sphere = ds.sphere(H2max, (1.0, "kpc"))
minimi, maksimi = H2sphere.quantities.extrema("H2_fraction")
tf = yt.ColorTransferFunction((np.log10(minimi), np.log10(maksimi)))
tf.add_layers(100, w=0.01, colormap="hot")

katselusuunta = [1, 1, 1]
W = 0.022 # vakio, joka määrää, kuinka laaja alue kuvassa nä ytetään
leveys = 512 # kuvan leveys pikseleissä

# luodaan camera-olio
cam = ds.camera(H2max, katselusuunta, W, leveys, tf, fields=["H2_fraction"], data_source=H2sphere)

frame=0
step=0.02
frames = int(math.floor(2*math.pi/step))

# zoomataan ja käännetään kameraa ja tallennetaan kuva numeroituna
```

```
33 for i, snapshot in enumerate(cam.zoomin(3, frames, clip_ratio
   ↪ =8.0)):
34     cam.rotate(0.05)
35     snapshot.write_png("kuvat/volume-spiraali%04i.png" %frame)
36     frame += 1
```