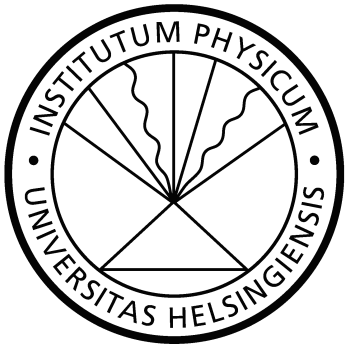


Kandidaatintutkielma
Tähtitiede

AMR-simulaatiotulosten visualisointi Pythonilla `yt`-pakettia käyttäen

Anni Järvenpää
2015

Ohjaajat: Peter Johansson
John Regan
Tarkastaja: Peter Johansson

HELSINGIN YLIOPISTO
FYSIIKAN LAITOS

PL 64 (Gustaf Hällströmin katu 2)
00014 Helsingin yliopisto

Sisältö

1 Johdanto

Simulaatiot ovat tärkeä osa tähtitieteen tutkimusta. Niiden avulla voidaan muun muassa tutkia kohteiden hyvin pitkän aikavälin kehitystä sekä tutkia sellaisiakin kohteita, joista ei vielä ole kattavaa havaintoaineistoa. Samoin voidaan testata teorioiden toimivuutta vertaamalla tietyillä alkuehdoilla ja mallilla saatuja tuloksia havaintoihin.

Eräs paljon käytetty simulaatiotyyppi ovat AMR-simulaatiot, joissa simuloitava alue katetaan päällekkäisillä hiloilla, joiden tiheydet vaihtelevat. Näin joitakin osia simulaatiosta voidaan laskea ympäristöään suuremmalla tarkkuudella, jolloin kiinnostavat alueet saadaan simuloiduksi halutulla tarkkuudella ilman, että laskentatehoa käytetään tarpeettoman paljon vähemmän kiinnostavien alueiden laskemiseen. Eräs esimerkki AMR-koodista on Enzo, joka tukee muun muassa hydrodynamiikkaa, kaasupilvien kemiaa ja säteilynkuljetusta sopien näin ollen hyvin astrofysikaalisten ongelmien ratkaisemiseen. [12]

Jotta simulaatioiden tuloksia voidaan hyödyntää, täytyy data redusoida jotenkin. Tässä hyödyllinen työkalu ovat erilaiset visualisaatiot. Hyvin suunniteltu ja toteutettu plotti välittää tutkittavasta datasta oleellisen tiedon nopeasti ja helppolukuisesti. Tulosten visualisointi on kuitenkin tavallisesti aikaavievää ja vaatii kattavaa ymmärrystä simulaatiodatasta, sillä datasta on löydettävä oikeat alueet ja ne on osattava syöttää jollekin plottaustyökalulle oikeassa muodossa ja oikeissa yksiköissä.

Tätä työtä helpottamaan on kuitenkin kehitetty `yt`, joka on avoimen lähdekoodin Python-paketti, joka abstrahoi simulaatiodatan helposti ymmärrettävään muotoon. Käyttäjän ei esimerkiksi tarvitse huolehtia hiloista tai partikkeleista eikä tietää, missä tiedostossa mikäkin osa datasta on tallennettuna, vaan käyttäjä voi suoraan antaa haluamansa alueen sijainnin ja koon esimerkiksi kiloparsekeissa. `yt`:n vahvuuksiin kuuluu myös se, että sitä voidaan käyttää useiden eri simulaatiokoodien kanssa. Lisäksi se tukee rinnakkaislaskentaa `mpi4py`-moduulia käyttäen ja mahdollistaa visualisoinnin upottamisen simulaatiokoodiin, jolloin visualisointeja voidaan tallentaa lähes mielivaltaisen usein. [21]

Tälläinen työkalu on äärimmäisen hyödyllinen, sillä se helpottaa ja nopeuttaa visualisaatioiden luomista, jolloin tutkijat voivat keskittyä entistä enemmän varsinaiseen tieteeseen. Lisäksi koska `yt`:n lähdekoodi on avointa, voidaan havaitut ongelmat ja puutteet korjata hyvinkin nopeasti ja sitä voi muokata ja laajentaa itse paremmin tarpeisiinsa sopivaksi.

Tässä työssä esitellään pääpiirteet sekä Enzosta että `yt`:stä ja annetaan esimerkkejä `yt`:n soveltamisesta Enzo-dataan. Esimerkkiskriptit on pyritty valitsemaan siten, että ne tuovat mahdollisimman hyvin esiin `yt`:n ominaisuuksia ja ne ovat helposti laajennettavissa ja sovellettavissa. Lisäksi esitellään esimerkkinä käytettävä data ja sen taustat ja eritellään `yt`:n lähdekoodiin monen kuvaajan plottien luomiseen käytettävän työkalun parantamiseksi tehdyt muutokset.

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. [12]

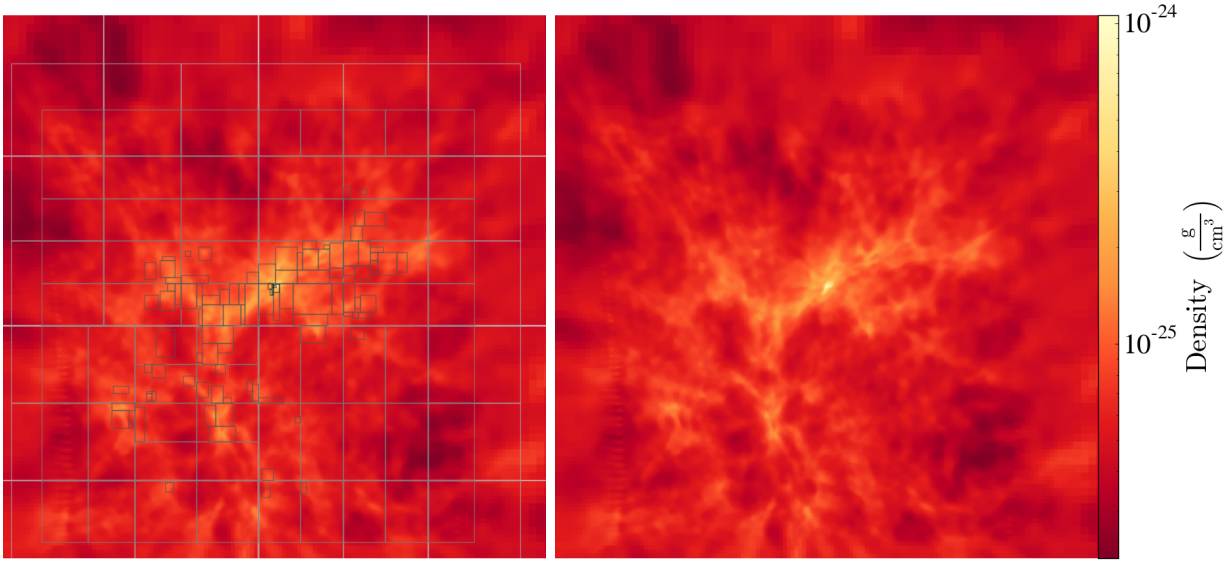
Enzo hyödyntää mukautuvaa hilantihennystä (*Adaptive Mesh Refinement*, AMR), joka mahdollistaa aika- ja paikkaresoluution kasvattamisen simulaation kiinnostavilla alueilla. Tämä on tärkeää, sillä usein melko pienellä alueella tarvitaan suurta resoluutiota, mutta koko simulaation ajaminen näin suurella tarkkuudella veisi kohtuuttoman paljon aikaa. Kullekin paikalle valitaan sopiva resoluutio automaattisesti käyttäjän määrittelemien ehtojen mukaan. [12]

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.[12]

Kaikki hilat ovat karteesisia ja suorakulmaisia. Alueille, joilla tarvitaan esimerkiksi suuremman tiheyden takia 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 käytetään pienempiä ja tiheämpiä hiloja kuin muualla. [12]

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. [12]



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). Tiheämmillä alueilla solut ovat pienempiä. Resoluution heikentyminen simulaation reuna-alueilla sijaitsevissa suuremmissa ja harvemmissa hiloissa on selvästi nähtävissä oikeanpuoleisessa kuvassa.

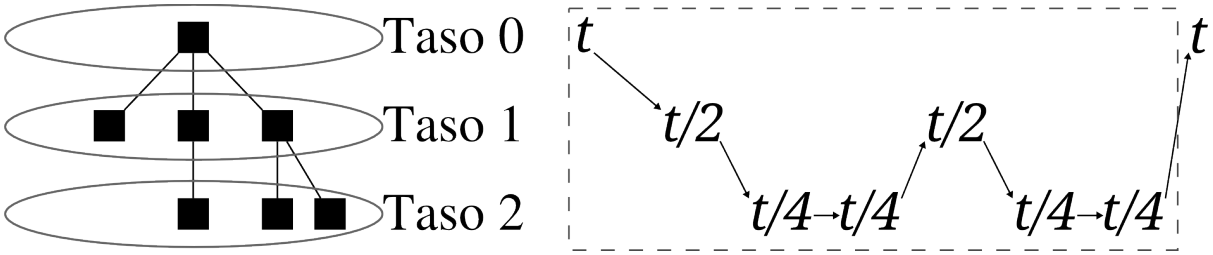
Hienompia hiloja luotaessa valitaan hilan solujen koko siten, että hila rajoittuu reunoistaan vanhempansa solujen tahkoihin. Lisäksi solun vanhemman sivun pituuden tulee olla jokin monikerta solun sivun pituudesta. [12]

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.[17, 12]

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). [12]

W-syklissä kunkin tason kaikille hiloille lasketaan aluksi niiden tila yhden aika-askeleen kuluttua. Laskeminen aloitetaan juurihilasta ja 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. [12]

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. [12]

3 yt

yt on avoimen lähdekoodin Python-paketti, joka 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. [21]

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. [21]

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 keskeltä 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 [21, 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. [21, 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 suureiden laskemisen jälkeen yt tarjoaa laajan valikoiman työkaluja tulosten visualisointiin. Aiemmin esiteltyjen poikkileikkausten lisäksi yt:llä voi luoda muun muassa erilaisia profiileja, painottamattomia tai painotettuja projektioita tai 3D-renderöintejä. [21]

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. [21]

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`. [21, 4]

yt pystyy käsittelemään myös useita datasettejä tai objekteja rinnakkain. Käyttämällä jokerimerkkejä kuten * ja ? dataa ladattaessa voidaan kerralla valita 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 (ohjelmointirajapinta, *Application programming interface*) 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. [21]

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. [21]

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 kuitenkin 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. Sekä läpileikkauksia että projektioita voi tehdä myös muissa kuin simulaation koordinattiakselien suunnissa käyttäen `OffAxisSlicePlot`- ja `OffAxisProjectionPlot`-luokkia. [3]

Projektiota luotaessa pikselien arvot voidaan määrittää usealla eri tavalla. Yleisimpiä näistä ovat painotettu ja painottamaton integrointi, jotka 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, 7]

3.4.2 3D

`yt` tarjoaa myös mahdollisuuden kolmiulotteisten rakenteiden tarkastelemiseen tilavuusrenderöimällä. Kuten kaksiulotteisissakin ploteissa myös tilavuusrenderöinnissä voidaan käyttää mitä tahansa simulaatiodatan kenttää ja esimerkiksi katselusuuntaa kääntämällä tai zoomaamalla 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. Täten color transfer function kertoo kunkin pisteen värin paikan funktiona siten, että $f(v) \rightarrow (r, g, b, a)$. Sen voi määrätä käyttäen mitä tahansa simulaation

kenttä joko painottamattomana tai painotettuna toisella kentällä. [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 Supermassiiviset mustat aukot

Tutkielmassa esiteltävät plotit on luotu John Reganin et al. supermassiivisten mustien aukkojen syntyä mallintavasta simulaatiodatasta. Simulaatio on ajettu käyttäen aiemmin esiteltyä Enzo-simulaatiokoodia. Sen tarkoituksena on tutkia, kuinka hyvin varhaisessa maailmankaikkeudessa voi syntyä havaintoja vastaavia supermassiivisia mustia aukkoja. Simulaatiossa tutkitaan mahdollisuutta kaasupilven romahtamiseen suoraan noin 10^4 – 10^6 $M_\odot$ massaiseksi mustaksi aukoksi vetymolekyylejä hajottavan Lyman-Werner -kaistassa säteilevän säteilylähteen läheisyydessä. [19]

4.1 Havainnot

Kvasaareista on toisistaan riippumattomia havaintoja jopa punasiirtymillä $z \gtrsim 6$, jotka vastaavat aikaa noin 0,9 miljardin vuoden sisällä maailmankaikkeuden synnystä. Näitä havaintoja on tehty muun muassa SDSS- sekä VIKING-datasta. Optisen alueen havainnoissa joudutaan rajoittumaan alueeseen $z \lesssim 6,4$, mutta lähi-infrapunassa on havaittu kohde jopa punasiirtymällä $z = 7,085$. [13, 16, 22]

Kvasaarihavainnot osoittavat, että jo näillä varhaisilla punasiirtymillä on ollut olemassa supermassiivisia mustia aukkoja, joiden massa on luokkaa 10^9 $M_\odot$. Tämä on yllättävää, sillä mustan aukon on kasvettava hyvin nopeasti tai synnyttävä melko suurena, jotta se ehtii saavuttaa näin suuren massan. Näiden varhaisten supermassiivisten mustien aukkojen synnyn selittäminen onkin haasteellista nykyisillä teoreettisilla malleilla. [15]

4.2 Eddingtonin luminositeetti

Vaikka mustan aukon läheisyydessä olisi runsaasti ainetta, ei aukko voi kasvaa mielivaltaisen nopeasti. Kun aukkoon putoaa ainetta, aineen lepomassan energiasta jopa 40%, tyypillisestikin noin 10%, vapautuu säteilynä [18]. Tästä aiheutuva säteilypaine vaikuttaa muihin kertymäkiekon hiukkasiin työntäen niitä kauemmas, jolloin aukkoon putoavan aineen määrä vähenee jälleen. [10]

Alla olevassa käsittelyssä tutkitaan pelkästään elektronien tuntemia voimia. Mustaa aukkoa ympäröivä aine koostuu kuitenkin varhaisessa maailmankaikkeudessa lähinnä vetyplasmasta, jossa on elektronien lisäksi protoneja. Protonin vaikutusala Thomsonin sironnassa on kuitenkin elektroniin verrattuna noin 1836-kertaisesta massasta johtuen noin 3.2×10^6 kertaa pienempi kuin elektronin, joten protoneihin vaikuttava säteilynpaine on mitättömän pieni verrattuna elektroneihin vaikuttavaan. Coulombin voiman ansiosta elektronien tuntema säteilypaine välittyy kuitenkin myös protoneihin. Näin ollen kaasuun kohdistuvien voimien laskemiseksi tulee hiukkasen massana m käyttää keskimääräistä massaa per elektroni, vetyplasman tapauksessa siis käytännössä protonin massaa. [18]

Aukkoa ympäröivän aineen elektroneihin vaikuttaa säteilypaineen aiheuttama voima

$$\vec{F}_R = \frac{L}{4\pi R^2 c}\sigma_T \vec{e}, \tag{4}$$

joka työntää elektroneja kauemmas aukosta. Yhtälössä L on aukon luminositeetti, R elektronin etäisyys aukosta ja σ_T elektronin vaikutusala Thomsonin sironnassa. Samalla aukon painovoima vetää niitä kohti aukkoa yhtälön 5 mukaisesti.

$$\vec{F}_G = -\frac{GM_{BH}m_p}{R^2}\vec{e} \tag{5}$$

Hiukkanen putoaa kohti aukkoa kun $|\vec{F}_G| > |\vec{F}_R|$. Kun oletetaan ionisoituneen aineen jakautuneen tasaisesti aukon ympärille ja aukon säteilevän pallosymmetrisesti, voidaan tästä ehdosta laskea luminositeetti, jolla säteilypaine on yhtä suuri kuin aukon painovoima, jolloin ainetta ei enää putoa aukkoon. Tällöin aukko on saavuttanut *Eddingtonin luminositeetin*, jolle voidaan yhtälöiden 4 ja 5 avulla ratkaista yhtälön 6 mukainen lauseke. [18]

$$L_{Edd} = \frac{4\pi G M_{BH} m_p}{\sigma_T} c \tag{6}$$

Pallosymmetrisesti massaa keräävä musta aukko ei voi kasvaa Eddingtonin luminositeetin asettamaa rajaa nopeammin. On kuitenkin kehitetty joitakin malleja, joissa sopivilla kertymäkiekon ominaisuuksilla voidaan hetkellisesti saavuttaa esimerkiksi 100–200 -kertainen kasvunopeus Eddingtonin luminositeetin asettamaan ylärajaan verrattuna. Tämä on mahdollista, mikäli mustan aukon säteily kohdistuu hyvin pieneen avaruuskulmaan tai energiaa siirtyy muilla tavoilla. [20]

4.3 Supermassiivisten mustien aukkojen mahdolliset syntytavat

Selkein tapa mustan aukon synnylle on syntyä tähden kehityksen lopputuotteena. Mikäli esimerkiksi massaltaan luokkaa 100–300 $M_\odot$ oleva populaation III tähti romahtaa mustaksi aukoksi punasiirtymällä $z \approx 20$, se voi ehtiä kasvaa supermassiiviseksi punasiirtymään $z \approx 7$ mennessä. Tämä edellyttää kuitenkin sitä, että saatavilla on riittävästi materiaa, jotta aukko voi kasvaa lähes koko ajan Eddingtonin luminositeetin asettamalla rajalla. [23, 11]

Tämä on kuitenkin ongelmallista, sillä populaation III tähdet yleensä menettävät elinaikanaan kaasupilven, jossa ne ovat syntyneet. Lisäksi jotkin simulaatiot viittaavat siihen, että tyypillinen populaation III tähti oli massaltaan vain joitakin kymmeniä $M_\odot$, jolloin kasvaminen supermassiiviseksi tarkasteltavalla aikavälillä on vielä epätodennäköisempää. Lisäksi mustan aukon alettua kerätä materiaa, sen säteilemä ionisoiva säteily hajottaa entisestään aukkoa ympäröivää pilveä, jolloin kasvu rajoittuu murto-osaan Eddingtonin rajasta. [23, 9, 11]

Mitä massiivisempi musta aukko on syntyessään, sitä hitaampi kasvu riittää selittämään havainnot. Romahtaminen suoraan mustaksi aukoksi vaatii kuitenkin sopivat olosuhteet, sillä tavallisesti kaasupilvi fragmentoituu ja syntyy yhden mustan aukon sijaan paljon pienempiä tähtiä. Fragmentoitumisessa syntyvien kohteiden koon määrää pilven Jeansin massa M_J, jolle pätee $M_J \propto T^{\frac{3}{2}}$ missä T on kaasun kineettinen lämpötila. [11, 19, 14]

Ainoastaan mikäli kaasupilvi on tarpeeksi suuri ja kuuma, se voi romahtaa luokkaa 10^4 $M_\odot$ massaiseksi mustaksi aukoksi. Näin suuren aukon kasvaminen havaintoja vastaavaksi pystytään jo selittämään uskottavasti. Korkean lämpötilan ylläpitämiseksi pilven jäähtymisen on kuitenkin oltava hyvin tehotonta, eli pilvi ei saa sisältää metalleja, pölyä tai molekulaarista vetyä. [11, 19]

Varhaisessa maailmankaikkeudessa metalleja ja pölyä oli hyvin vähän, mutta molekulaarista vetyä voi kuitenkin syntyä tiheillä alueilla. Siksi jonkin prosessin onkin hajotettava vetymolekyylejä atomeiksi. Tämän voi tehdä esimerkiksi säteily Lyman-Werner -kaistassa (11,2–13,6 eV), joka virittää molekyylin elektronit korkeammille energiatasoille, jolloin molekyyli hajoaa takaisin atomeiksi eikä kaasun jäähtyminen tehostu niin paljon, että se fragmentoituisi tähdenmassaisiksi osiksi. [19]

4.4 Tutkittava simulaatio

LW-kaistan säteilyn vaikutusta varhaisiin kaasupilviin on tutkittu jo useissa julkaisuissa ja tulokset ovat vaikuttaneet lupaavilta, mutta aiemmissa simulaatioissa säteily on ollut isotrooppista. Tutkielman esimerkkidatana käyttämässäni John Reganin et al. simulaatiossa on tästä poiketen käytetty pistemäistä säteilylähdettä isotrooppisen LW-taustan sijaan. Tämä todennäköisesti vastaa aiempia simulaatioita paremmin todellisia olosuhteita. [19]

Tutkittava halo on peräisin aiemmin ajetuista simulaatioista. Niitä varten ajettiin ensin melko epätarkkoja pelkkää pimeää ainetta sisältäviä simulaatioita, joista poimittiin tiheimmät alueet punasiirtymällä $z = 30$, joka vastaa noin kymmenen miljoonan vuoden ikäistä maailmankaikkeutta. Näillä alueilla ajettiin uudelleen alusta aloittaen tarkemmat simulaatiot, joissa myös hydronynamiikka otettiin huomioon. Datasta valittiin yksi halo, jonka romahtamista tutkittiin. Punasiirtymällä $z = 30$ halon massa oli noin $10^6\ M_\odot$ ja punasiirtymään $z = 21$ mennessä se kasvoi noin massaan $6 \times 10^7 M_\odot$. [19]

Simulaatiossa tutkittiin halon romahtamista kun 3 kpc päässä siitä on LW-kaistassa säteilevä kohde. Simulaatio ajettiin useita kertoja erilaisilla säteilyn voimakkuuksilla tarvittavan intensiteetin selvittämiseksi. Simulaation tulokset viittavat siihen, että noin 10^{54} fotonia/s vuo LW-kaistassa riittää vähentämään H_2 määrää riittävästi, jotta noin massaltaan luokkaa $10^5\ M_\odot$ oleva kappale voi syntyä. Tätä pienemmillä vuon arvoilla syntyy vain noin $10^3\ M_\odot$ massa, joka romahtaa H_2 vaikutuksesta Populaation III tähdeksi. [19]

5 yt:n sovelluksia

Aloitin yt:hen tutustumisen selaamalla dokumentaatiota sekä cookbookia[2] ja ensin kopioimalla ja sittemmin muokkaamalla siellä annettuja malliohjelmia käyttäen esimerkkidataa. Tutustuin tarkimmin poikkileikkauksiin, 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, voidaan käyttää `eps_writer`-luokkaa. Sen tarjoama tuki profiileille oli kuitenkin puutteellinen, joten jouduin myös parantelemaan yt:n lähdekoodia.

5.1 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 haluttaessa luoda mielivaltaisesta suunnasta [3].

Lisäksi plotin väripalkin alueeksi asetetaan $3 \times 10^{-26} - 2.5 \times 10^{-25}$ $\frac{\mathrm{g}}{\mathrm{cm}^3}$ (noin 0,01–0,11 vetyatomia kuutiosenttimetrissä) 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.

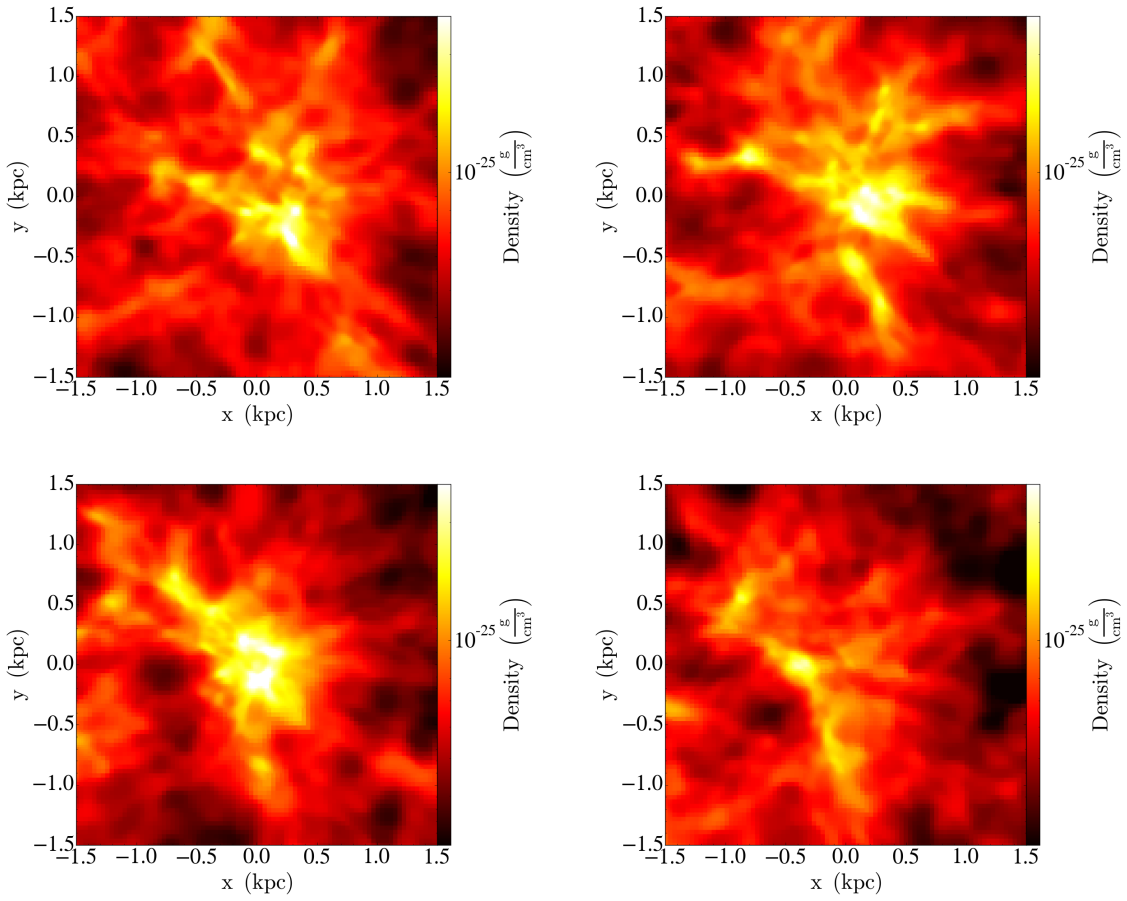
[2] `http://yt-project.org/docs/3.1/cookbook/`
[3] `http://yt-project.org/doc/visualizing/colormaps/`

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.

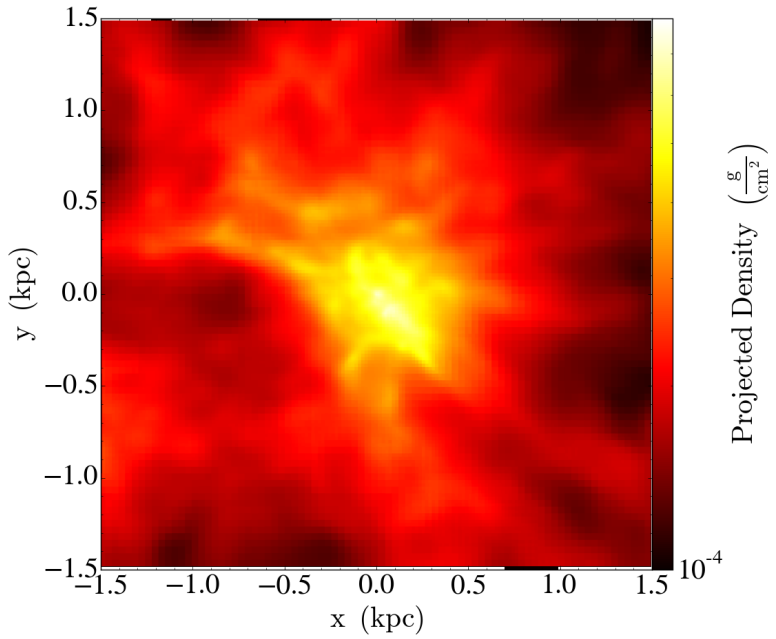
```
1  # -*- coding: utf-8 -*-
2  import yt
3  import numpy as np
4
5  ds = yt.load("data/dataset")
6  v, maxtiheys = ds.find_max("density")
7  frame = 0
8
9  for i in np.arange(-0.5, 0.5, 0.01):
10     siirto = yt.YTArray([0, 0, i], 'kpc')
11     keskusta = maxtiheys - siirto
12
13     image = yt.SlicePlot(ds, '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
```

5.2 Projektiot

Listingissä 2 on esitetty yksinkertaisen projektion tallentava skripti. Se käyttää listingin 1 kanssa samaa simulaatiodataa, 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")
```

5.3 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 molekulaarisen vedyn osuus kaikesta vedystä on suurin hyödyntämällä `yt`:n tarjoamia `find_max`- ja `max_location`-funktioita. Tämän jälkeen riveillä 16–19 määrätään tutkittavaksi alueeksi säteeltään 1 kpc pallo löydetyn suurimman molekulaarisen vedyn pitoisuuden kohdan ympärillä. Lisäksi täytyy määrittää kameran ja kuvan keskustan välinen vektori, kuvan leveys ja resoluutio.

Kun tarvittavat parametrit on alustettu, voidaan camera-olio luoda. Kameran kuvaamiksi kentiksi asetetaan ainoastaan H2_fraction. Lisäksi määritetään, että kuvassa näytetään ainoastaan aiemmin luodun 1 kpc säteisen pallon sisällä oleva aine, jolloin vältytään siltä riskiltä, että tutkittavan alan eteen tulee toinen huomattavasti molekulaarista vetyä sisältävä alue, joka estää tutkittavan alueen näkemisen.

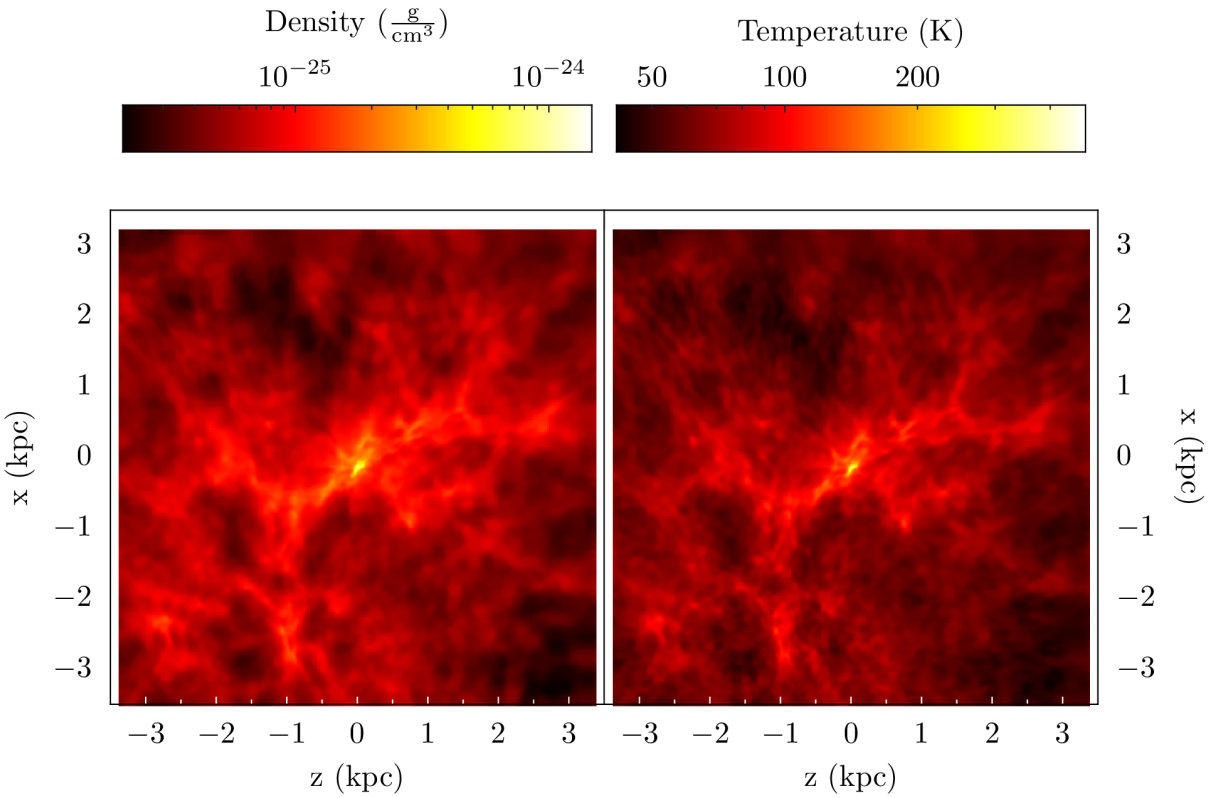
Animaation luomiseksi käytetään kameran `zoomin`-funktiota, jolla on mahdollista iteroida haluttu zoomaus halutulla määrällä vakiokokoisia askelia. Esimerkin animaatiossa zoomauskertoimeksi on valittu 3 ja askelten määrä on valittu sellaiseksi, että kohde voidaan kiertää kerran 0,04 radiaanin askelin. Kierron toteuttamiseksi kameraa käännetään kunkin zoomauksen jälkeen. Tämän jälkeen `zoomin`-funktion palauttama kuva tallennetaan numeroituna.

5.4 Usean kuvaajan plotit

`yt` tarjoaa mahdollisuuden kuvaajien palauttamiseen taulukkona, jota voidaan edelleen käyttää esimerkiksi `matplotlib`in[4] kanssa usean kuvaajan yhdistämiseksi samaan kuvaan. Tähän on kuitenkin myös apufunktio, jolle voidaan antaa suoraan joko kuvaajat tai jpg-kuvat, jotka asemoidaan haluttaessa väripalkkien kanssa ruudukkoon.

Liitteessä B on nähtävillä `eps_writer`iä hyödyntävä skripti, joka plottaa tiheyden ja lämpötilan 7 kpc leveällä alueella simulaation tiheimmän kohdan ympäriltä ja tallentaa ne sekä erikseen että yhdessä. Plotti, jossa molemmat ovat, on nähtävissä kuvassa 5. Plotin muodostamista voidaan ohjata erilaisilla parametrina annettavilla lipuilla. Esimerkiksi kuvan 5 luomisessa on hyödynnetty parametreja `xaxis_flags` ja `yaxis_flags`, joilla asetetaan akselit plottien ulkoreunoille sekä parametria `cb_location`, jolla väripalkit

[4] `http://matplotlib.org/mpl_toolkits/axes_grid/api/axes_grid_api.html`



Kuva 5: Liitteessä B esitetyn skriptin tuottama kuva.

asetetaan kuvaajien yläpuolelle.

5.4.1 Ongelmat

`yt` on kuitenkin vielä kehittyvä paketti, mikä on selvästi huomattavissa käyttämäni `yt`-version 3.2 `eps_writer`-luokassa. Kuvasta 5 nähdään esimerkiksi, että plotit eivät ole keskellä niille piirrettyjä reunuksia vaan liian alhaalla. Lisäksi mikäli esimerkiksi käytettyä väriskaalaa vaihdetaan, täytyy yksittäiset kuvat tallentaa ennen niiden käyttämistä multiplotissa. Jos esimerkiksi liitteen B rivit 21 ja 30 kommentoidaan pois koodista, saadaan tuloksena kuva, jossa väripalkit noudattavat hot-colormappia mutta kuvat oletuksena käytettävää jet-väritystä.

Lisäksi dokumentaation [6] ja lähdekoodin [8] mukaan `eps_writer` tukee myös profiileja. Esimerkiksi liitteen C skriptin ajaminen johtaa kuitenkin virheilmoitukseen "`AttributeError: 'ProfilePlot' object has no attribute '_redraw_image'`"ohjelman suorituksen edettyä riville 85, jossa plotit yhdistetään. Yksittäiset kuvat tallentuvat kuitenkin normaalisti, mikä osoittaa skriptin alkuosan toimivan. Todennäköisesti `ProfilePlot`-luokkaa tai jotain muuta `yt`:n osaa on muokattu, ja testien riittämättömän kattavuuden takia vaikutukset muualla koodissa ovat jääneet huomaamatta. `yt` on kuitenkin avointa

lähdekoodia, joten ongelman korjaaminen itse oli mahdollista.

5.4.2 Muutokset yt:n lähdekoodissa

Virheilmoituksen perusteella ongelmakohdan paikantaminen oli hyvin yksinkertaista. Liitteen D listingissä 3 on esitetty ongelman aiheuttanut profiilien käsittelyyn kirjoitettu `eps_writer`-luokan `insert_image_yt`-funktion osa, jonka on tarkoitus hakea multiplottiin lisättävä kuva ja poistaa profiilin tapauksessa tarpeeton väripalkki.

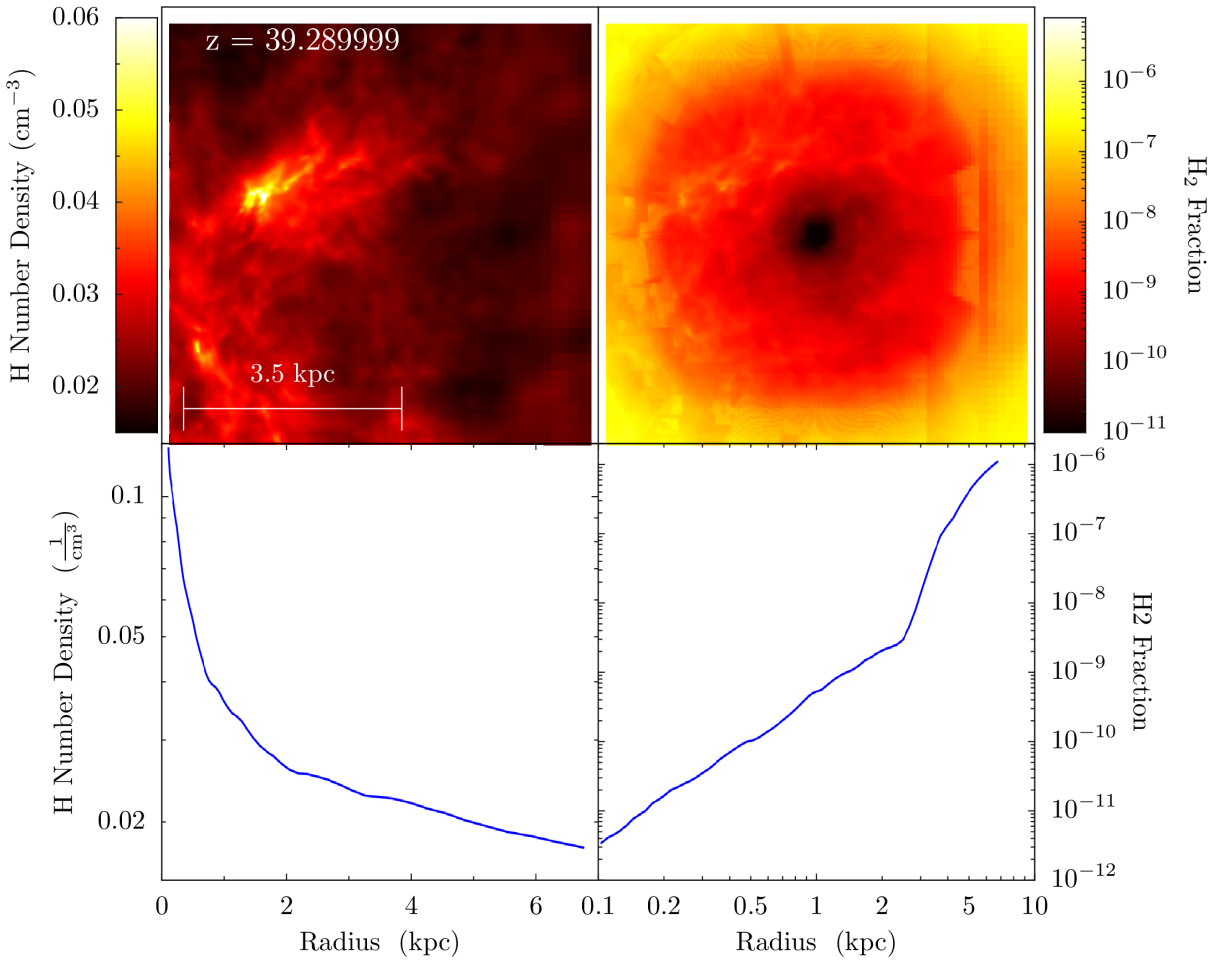
`ProfilePlot`-luokalla ei kuitenkaan ole `_redraw_image()`-funktiota, joten ohjelman suoritus keskeytyy. Myöskään `_figure` ei ole määritelty. Sen sijaan `figures`-attribuuttiin on tallennettu `FigureContainer`, josta haluttu figure voidaan hakea. Lisäksi väripalkin poistaminen ei ole tarpeellista. Sama lohko näiden korjausten jälkeen on nähtävissä liitteen D listingissä 4.

Tämän lisäksi alkuperäisessä koodissa käytetään akselien rajojen, tekstien ja logaritmisuuden määrittämiseen samaa lohkoa sekä `ProfilePlot`- että `PhasePlot`-luokkien olioiden käsittelyyn. Sen käyttäminen profiileihin johtaa kuitenkin kaatumiseen. Siksi kirjoitin profiilien käsittelyyn oman, erillisen lohkonsa, joka on nähtävissä liitteen D listingissä 5.

Lisäksi profiilien kanssa väripalkin käyttäminen ei ole koskaan mielekästä, joten lisäsin `colorbar_yt`-funktioon tarkastuksen, jolla varmistetaan, ettei sitä käytetä profiilien kanssa. Mikäli käyttäjä yrittää tehdä näin, heitetään `RuntimeError` viestillä "When using ProfilePlots you must either set yt_nocbar=True or provide colorbar flags so that the profiles don't have colorbars".

Näiden muutosten jälkeen liitteen C skripti voidaan ajaa virheettä. Kukin plotti täytyy edelleen tallentaa yksin, jotta plottiin tehdyt muutokset välittyvät luotavaan nelikenttään. Lisäksi `yt` piirtää väripalkit niin kauas varsinaisista ploteista, että tarvittaessa väliin mahtuisi akselin selite. Koska sitä ei kuitenkaan tässä plotissa tarvita, muutin tätä plottia varten väripalkin sijoittelua määräääviä vakioita siten, että ne siirtyivät hieman lähemmäs plotteja. Tuloksena saatava nelikenttä on nähtävissä kuvassa 6.

Koodin toimivuuden testaamisen jälkeen loin pull requestin muutosteni sulauttamiseksi `yt`:n kehitysversioon. Kokeneemmat kehittäjät arvioivat kaikki pull requestit ja joko ehdottavat parannuksia tai suoraan hyväksyvät tai hylkäävät requestin. Muutokseni hyväksyttiin, joten nyt ne ovat kaikkien käytettävissä ja paranneltavissa.



Kuva 6: Liitteessä C esitetyn skriptin tuottama kuva. Ylemmissä ploteissa on esitetty samalta alueelta säteilevän hiukkasen ympäristöstä vedyn lukumäärätiheyden projektio ja läpileikkaus molekulaarisen vedyn osuudesta. Näiden alla olevissa profiileissa on esitetty samat suureet profiileina, vedyn lukumäärätiheys keskitettynä simulaation tiheimpään kohtaan ja molekulaarisen vedyn osuus keskitettynä säteilylähteeseen.

6 Johtopäätökset

`yt` helpottaa simulaatiodatan visualisointia huomattavasti ja tekee visualisointien luomisesta intuitiivista ja vaivatonta. Tästä huolimatta on hyödyllistä ymmärtää ohjelman toimintaa jonkin verran, jotta plotteja ei esimerkiksi luo vahingossa tarpeettoman tehottomasti. Samoin simulaatiodatan ja niiden luomiseen käytettyn simulaatiokoodin tunteminen on hyödyllistä, jotta datasta voidaan valita kiinnostavat alueet ja datan erityispiirteet kuten esimerkiksi AMR-simulaatioiden tapauksessa paikan mukaan muuttuva resoluutio osataan ottaa huomioon.

Erilaisia plotteja on lukuisia, ja niiden joukosta kannattaa pyrkiä valitsemaan tutkittavan datan erityispiirteet parhaiten esille tuovat. Esimerkiksi läpileikkaus näyttää tarkasti yhden tason simulaatiosta, mutta kadottaa näkösäteen suuntaiset rakenteet, kun taas projektiolla on mahdollista tarkastella laajempia rakenteita, mutta niiden sijoittumista kolmiulotteisesti on mahdotonta nähdä. Plottityypin valinnan lisäksi myös parametrit kannattaa valita huolellisesti. Usein paras tapa tähän on tehdä runsaasti erilaisia kokeiluja muutellen arvoja.

Animaatiot tuovat usein simulaation piirteet yksittäisiä plotteja paremmin esiin. Erityisesti aikasarjoissa ja kolmiulotteisia rakenteita tutkittaessa animoinnista on hyötyä, sillä animaatiosta ajan kuluessa tapahtuvat muutokset on helpompi hahmottaa. Samoin yksittäinen 3D-renderöinti ei useinkaan anna kovin selkeää kuvaa kohteiden etäisyyksistä, mutta esimerkiksi kameran siirtäminen tai kääntäminen saa aikaan illuusion kolmiulotteisuudesta. Animaatioiden huonona puolena on kuitenkin yksittäisiin kuviin verrattuna hyvin suuri levytilan tarve ja framejen luomiseen kuluva aika.

Kaikenkaikkiaan `yt` on hyvin lupaava apuväline ja kehitystyö jatkuu yhä sen parantamiseksi. Kuten eps_writerin tapauksessa huomattiin, on joidenkin vähemmän käytettyjen ominaisuuksien toiminta vielä epävarmaa eivätkä esimerkiksi automaattiset testit kata kaikkea koodia. Ydinominaisuudet ovat kuitenkin luotettavia ja hyvin testattuja ja dokumentoituja.

Myös tämän työn tekemisen aikana tehdyt muutokset `yt`:n lähdekoodiin päätyivät osaksi `yt`:n kehitysversiota. Korjasin ongelman, jossa profiilien käyttäminen `eps_writer`in kanssa johti ohjelman kaatumiseen ja tein muutoksistani pull requestin. Se hyväksyttiin muutoksitta, joten nyt parannukseni ovat kaikkien käytettävissä.

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.eps_writer.multiplot. http://yt-project.org/doc/reference/api/generated/yt.visualization.eps_writer.multiplot.html. Accessed: 30.7.2015.

[7] 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.

[8] yt source: yt.visualization.eps_writer.multiplot. http://yt-project.org/doc/_modules/yt/visualization/eps_writer.html#multiplot. Accessed: 30.7.2015.

[9] M. A. Alvarez, J. H. Wise, and T. Abel. Accretion onto the First Stellar-Mass Black Holes. *Astrophys. J., Lett.*, 701:L133–L137, August 2009.

[10] M. L. Bartelmann. *Theoretical astrophysics : : an introduction /.* Wiley-VCH,, Weinheim :, cop. 2013. Verkkoaineisto: Ebrary Academic Complete.

[11] Volker Bromm and Naoki Yoshida. The first galaxies. *Annual Review of Astronomy and Astrophysics*, 49(1):373–407, 2011.

[12] 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.

[13] X. Fan, J. F. Hennawi, G. T. Richards, M. A. Strauss, D. P. Schneider, J. L. Donley, J. E. Young, J. Annis, H. Lin, H. Lampeitl, R. H. Lupton, J. E. Gunn, G. R. Knapp, W. N. Brandt, S. Anderson, N. A. Bahcall, J. Brinkmann, R. J. Brunner, M. Fukugita, A. S. Szalay, G. P. Szokoly, and D. G. York. A Survey of z $>$ 5.7 Quasars in the Sloan Digital Sky Survey. III. Discovery of Five Additional Quasars. *Astron. J.*, 128:515–522, August 2004.

[14] Hannu. Karttunen. *Tähtitieteen perusteet /.* Tähtitieteellinen yhdistys Ursa,, Helsinki :, 2010.

[15] Y. Li, L. Hernquist, B. Robertson, T. J. Cox, P. F. Hopkins, V. Springel, L. Gao, T. Di Matteo, A. R. Zentner, A. Jenkins, and N. Yoshida. Formation of z $\sim$ 6 Quasars from Hierarchical Galaxy Mergers. *ApJ*, 665:187–208, August 2007.

[16] D. J. Mortlock, S. J. Warren, B. P. Venemans, M. Patel, P. C. Hewett, R. G. McMahon, C. Simpson, T. Theuns, E. A. Gonzáles-Solares, A. Adamson, S. Dye, N. C. Hambly, P. Hirst, M. J. Irwin, E. Kuiper, A. Lawrence, and H. J. A. Röttgering. A luminous quasar at a redshift of z = 7.085. *Nature*, 474:616–619, June 2011.

[17] 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.

[18] J. E. Pringle. Accretion discs in astrophysics. *Ann. Rev. Astron. Astrophys.*, 19:137–162, 1981.

[19] J. A. Regan, P. H. Johansson, and J. H. Wise. The Direct Collapse of a Massive Black Hole Seed under the Influence of an Anisotropic Lyman-Werner Source. *ApJ*, 795:137, November 2014.

[20] A. Sądowski, R. Narayan, J. C. McKinney, and A. Tchekhovskoy. Numerical simulations of super-critical black hole accretion flows in general relativity. *Mon. Not. R. Astron. Soc.*, 439:503–520, March 2014.

[21] 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.

[22] B. P. Venemans, J. R. Findlay, W. J. Sutherland, G. De Rosa, R. G. McMahon, R. Simcoe, E. A. González-Solares, K. Kuijken, and J. R. Lewis. Discovery of Three z $>$ 6.5 Quasars in the VISTA Kilo-Degree Infrared Galaxy (VIKING) Survey. *ApJ*, 779:24, December 2013.

[23] D. J. Whalen and C. L. Fryer. The Formation of Supermassive Black Holes from Low-mass Pop III Seeds. *Astrophys. J., Lett.*, 756:L19, September 2012.

A 3D-animaatio spiraaliradalla kohdetta lähestyen

```
1  # -*- coding: utf-8 -*-
2  import yt
3  import numpy as np
4  import math
5
6  ds = yt.load("data/dataset")
7  ad = ds.all_data()
8
9  # etsitään tihein kohta ja tiheimmän kohdan ympäristöstä
       suurin H2-pitoisuus
10 v, tiheysmaksimi = ds.find_max("density")
11 densSphere = ds.sphere(tiheysmaksimi, (0.5, "kpc"))
12 maxlocs = densSphere.quantities.max_location(("gas",
       "H2_fraction"))
13 H2max = maxlocs[2:5]
14
15 # etsitään suurimman H2-pitoisuuden sijainnin ympärillä 1 kpc
       sisältä suurin ja pienin H2-pitoisuus ja luodaan niiden
       logaritmien avulla ColorTransferFunction
16 H2sphere = ds.sphere(H2max, (1.0, "kpc"))
17 minimi, maksimi = H2sphere.quantities.extrema("H2_fraction")
18 tf = yt.ColorTransferFunction((np.log10(minimi),
       np.log10(maksimi)))
19 tf.add_layers(100, w=0.01, colormap="hot")
20
21 katselusuunta = [1, 1, 1]
22 leveys = (1.5, 'kpc')
23 resoluutio = 512
24
25 # luodaan camera-olio
26 cam = ds.camera(H2max, katselusuunta, leveys, resoluutio, tf,
       fields=["H2_fraction"], data_source=H2sphere)
27
28 askel=0.02 # (radiaania)
29 frames = int(math.floor(2*math.pi/askel))
30
31 # zoomataan ja käännetään kameraa ja tallennetaan kuva
       numeroituna
32 for frame, snapshot in enumerate(cam.zoomin(3, frames,
       clip_ratio=8.0)):
33     cam.rotate(askel)
```

```
34 |     snapshot.write_png("kuvat/volume-spiraali%04i.png" %frame)
```

B Kaksi läpileikkausta samassa plotissa

```
# -*- coding: utf-8 -*-
import yt
import yt.visualization.eps_writer as eps
import matplotlib

sarakkeet = 2
rivit = 1
leveys = 7.0 # (kpc)
xaxis_flags = []
yaxis_flags = []
cb_location=[]
plotit = []

ds=yt.load("data/dataset")
v, tiheysmaksimi = ds.find_max("density")

# ensimmäinen kuva
tiheys = yt.SlicePlot(ds, 1, "Density", center=tiheysmaksimi,
   width=(leveys, 'kpc'))
tiheys.set_unit("Density", "g/cm**3")
tiheys.set_cmap(field="Density", cmap='hot')
tiheys.save("kuvat/tiheys.png")
plotit.append(tiheys)
xaxis_flags.append(0) # x-akseli alapuolelle
yaxis_flags.append(0) # y-akseli vasemmalle
cb_location.append("top") # väripalkki yläpuolelle

#toinen kuva
lampo = yt.SlicePlot(ds, 1, "Temperature",
   center=tiheysmaksimi, width=(leveys, 'kpc'))
lampo.set_cmap(field="Temperature", cmap='hot')
lampo.save("kuvat/lampo.png")
plotit.append(lampo)
xaxis_flags.append(0) # x-akseli alapuolelle
yaxis_flags.append(1) # y-akseli oikealle
cb_location.append("top") # väripalkki yläpuolelle

multi = eps.multiplot(sarakkeet, rivit, plotit,
   bare_axes=False,  xaxis_flags = xaxis_flags, yaxis_flags =
   yaxis_flags, cb_location=cb_location)
multi.save_fig("kuvat/EPSMultiPlot", format="png")
```

C Nelikenttä

```
# -*- coding:utf-8 -*-
import yt
import pyx
import yt.visualization.eps_writer as eps
from yt.units.yt_array import YTQuantity, YTArray
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

ds = yt.load("data/dataset")
dd=ds.all_data()

sarakkeet = 2
rivit = 2
leveys = yt.YTQuantity(7.0, 'kpc')
white=pyx.color.cmyk.white
linewidth = 3
prof_xrange = [0.1, 7.0] #kpc
colorbar_label = []
plots = []
colorbar_flags = []
xaxis_flags = []
yaxis_flags = []

# tutkittavat alueet
try:
    keskusta = dd["RadiationParticle", "particle_position"][0]
except:
    print "No particle position found"
    keskusta = [0.5, 0.5, 0.5]
leftedge = keskusta - leveys /2
rightedge = keskusta + leveys/2
myregion = ds.region(keskusta, leftedge, rightedge)

# H Number Density
H_proj = yt.ProjectionPlot(ds, 1, "H_number_density",
   center=keskusta, width=leveys, data_source=myregion,
   weight_field='density')
H_proj.set_log('H_number_density', False)
H_proj.set_zlim('H_number_density', 15e-3, 6e-2)
H_proj.set_cmap(field="H_number_density", cmap='hot')
#H_proj.save("kuvat/Hproj.png")
plots.append(H_proj)
colorbar_label.append("H Number Density (cm$^{-3}$)")
```

```
43 colorbar_flags.append(True)
44 xaxis_flags.append(-1)
45 yaxis_flags.append(-1)
46
47 # H2 Fraction
48 H2_slice = yt.SlicePlot(ds, 1, "H2_fraction", center=keskusta,
       width=leveys, north_vector = [1,0,0])
49 H2_slice.set_zlim('H2_fraction', 1e-11, 8e-6)
50 H2_slice.set_cmap(field="H2_fraction", cmap='hot')
51 #H2_slice.save("kuvat/H2fracproj.png")
52 plots.append(H2_slice)
53 colorbar_label.append("H$_2$ Fraction")
54 colorbar_flags.append(True)
55 xaxis_flags.append(-1)
56 yaxis_flags.append(-1)
57
58 # H profile
59 val, maxTiheys = ds.find_max("density")
60 sphere = ds.sphere(maxTiheys, leveys) # keskitetään H number
       density -kuvaaja tiheimpään kohtaan
61 H_prof = yt.ProfilePlot(sphere, "radius", ["H_number_density"])
62 H_prof.set_unit("radius", "kpc")
63 H_prof.set_xlim(prof_xrange[0], prof_xrange[1])
64 H_prof.set_ylim("H_number_density", 0.015, 0.13)
65 H_prof.set_line_property("linewidth", linewidth)
66 H_prof.x_log = False
67 #H_prof.save("kuvat/Hprof.png")
68 plots.append(H_prof)
69 colorbar_flags.append(False)
70 xaxis_flags.append(0)
71 yaxis_flags.append(0)
72
73 # H2 profile
74 sphere = ds.sphere(keskusta, leveys) # keskitetään H2 fraction
       -kuvaaja säteilylähteeseen
75 H2_prof = yt.ProfilePlot(sphere, "radius", ["H2_fraction"])
76 H2_prof.set_unit("radius", "kpc")
77 H2_prof.set_xlim(prof_xrange[0], prof_xrange[1])
78 H2_prof.set_ylim("H2_fraction", 1e-12, 2e-6)
79 H2_prof.set_line_property("linewidth", linewidth)
80 H_prof.x_log = False
81 plots.append(H2_prof)
82 colorbar_flags.append(False)
83 xaxis_flags.append(0)
84 yaxis_flags.append(1)
85 #H2_prof.save("kuvat/H2prof.png")
86
87 multi = eps.multiplot(sarakkeet, rivit, plots,
       bare_axes=False, cb_labels=colorbar_label, cb_flags =
       colorbar_flags, xaxis_flags = xaxis_flags, yaxis_flags =
       yaxis_flags)
```

```
88 multi.scale_line(label="%.1f kpc" % (0.5*leveys),size=0.5,
       loc=(0.05,1.08))
89 multi.title_box("z = %2.6f" % (ds.current_redshift),
       loc=(0.1,1.95),
90 color=white, bgcolor=None, text_opts=[pyx.text.size.large])
91 multi.save_fig("kuvat/nelikko", format="png")
```

D yt:n lähdekoodin muutokset

Listing 3: Alkuperäinen profiilin hakemiseen käytetty koodinpätkä.

```
elif isinstance(plot, ProfilePlot):
    plot._redraw_image()
    # Remove colorbar
    _p1 = plot._figure
    _p1.delaxes(_p1.axes[1])
```

Listing 4: Korjattu ohjelmanpätkä, joka korvaa listingin 3 koodin.

```
elif isinstance(plot, ProfilePlot):
    _p1 = plot.figures.items()[0][1]
```

Listing 5: Profiilin akselien rajojen, tekstien ja logaritmisuuden asettamiseksi `eps_writer`-luokkaan lisätty koodi.

```
        elif isinstance(plot, ProfilePlot):
            subplot = plot.axes.values()[0] # siirra muualle?
            # limits for axes
            xlimits = subplot.get_xlim()
            _xrange = (YTQuantity(xlimits[0], 'm'),
               YTQuantity(xlimits[1], 'm')) # unit hardcoded
               but afaik it is not used anywhere so it doesn't
               matter
            if list(plot.axes.ylim.viewvalues())[0][0] is
               None: #mieti tata, etsi parempi tapa
                 ylimits = subplot.get_ylim()
            else:
                 ylimits = list(plot.axes.ylim.viewvalues())[0]
            _yrange = (YTQuantity(ylimits[0], 'm'),
               YTQuantity(ylimits[1], 'm')) # unit hardcoded
               but afaik it is not used anywhere so it doesn't
               matter
            # axis labels
            xaxis = subplot.xaxis
            _xlabel = pyxize_label(xaxis.label.get_text())
            yaxis = subplot.yaxis
            _ylabel = pyxize_label(yaxis.label.get_text())
            # set log if necessary
            if subplot.get_xscale() == "log":
                 _xlog = True
            else:
```

```
20                      _xlog = False
21                  if subplot.get_yscale() == "log":
22                      _ylog = True
23                  else:
24                      _ylog = False
25                  _tickcolor = None
```