1. Käyttöohje

Mandel on X-Windows-ympäristössä toimiva ohjelma, jolla voi tarkastella Mandelbrotin ja Julian fraktaaleja. Komentoriviparametrejä ei ole, joten käynnistäminen tapahtuu yksinkertaisesti komennolla

```
mandel
```

Tällöin avautuu 400x400 pikselin kokoinen ikkuna, johon piirtyy Mandelbrotin joukko.

Ohjelmalle voidaan antaa syötteitä sekä hiirellä että näppäimistöllä.

Hiiri:

Pitämällä **vasen näppäin** pohjassa voidaan kuviosta rajata suorakulmainen alue, joka zoomataan koko ikkunan kokoiseksi.

Keskinäppäimellä vaihdetaan kuvaaja Mandelbrotin joukosta Julian joukoksi tai päinvastoin. Mikäli hiirtä näpäytetään Mandelbrotin kuvaajassa, käytetään näpäytyspistettä Julian iteraatiovakiona. Siirryttäessä Juliasta Mandelbrotiin ei näpäytyspisteellä ole merkitystä.

Pitämällä **oikea näppäin** pohjassa voidaan kuviota vetää vaakatasossa muuttamatta zoomausta.

Näppäimistö:

c asettaa ikkunan vasempaan alakulmaan koordinaattiruudun tai poistaa sen. Ruudussa näkyy joka hetki sen joukon pisteen koordinaatit, jonka päällä hiiriosoitin on.

j asettaa tai poistaa Julia-esikatseluruudun, mikäli nykyinen joukko on Mandelbrot. Esikatselun ollessa päällä ikkunan oikeassa yläkulmassa on hiiren osoittamaa pistettä vastaavan Julian joukon kuva käänteisfunktiomenetelmällä laskettuna. Tällöin kuva päivitetään aina, kun hiirtä liikutetaan.

i asettaa ikkunan alareunaan syöttörivin, jolla käyttäjä voi antaa uuden arvon kuvaajan laskemiseen käytettävien iteraatioiden maksimimäärälle (sallitut arvot 1 - 99 999). Ohjelma ei siis automaattisesti muuta iteraatiomääriä zoomattaessa.

z zoomaa kuvaa ulospäin pienentämällä zoomauskertoimen kolmasosaan. Vasen alakulma pysyy tällöin paikallaan.

ctrl-x lopettaa ohjelman.

2. Ratkaisutavan selitys

2.1 Kuvaajan laskeminen ja piirtäminen

Kuvaajan piirtämisen suorittaa aina repaint-funktio riippumatta siitä, onko joukko Mandelbrot vai Julia. Funktio saa ainoaksi parametrikseen osoittimen struct picture_info -tyyppiseen muuttujaan, joka sisältää tiedot piirrettävästä joukosta sekä piirrettävää aluetta rajaavan suorakulmion koordinaatit. Alue voi olla siis koko ikkuna tai mikä tahansa sen suorakulmainen osa. Funktio käy läpi jokaisen tähän alueeseen kuuluvan pikselin, skaalaa koordinaatit ja siirtää origon annettujen parametrien avulla, jolloin saadaan vastaavan Mandelbrotin tai Julian joukon pisteen koordinaatit z[0] = x + iy. Näille suoritetaan iteraatio

```
while( (n < max_iterations)  &&  (x*x + y*y < 4) )
{
  tmp = x*x - y*y + x_increment;
  y   = 2*x*y + y_increment;
  x   = tmp;
  n++;
}
```

jossa x_increment ja y_increment ovat Julian joukossa iteraatiovakioita ja Mandelbrotin joukossa pisteen alkuperaiset koordinaatit. Mikäli iteraation jälkeen n = max_iterations kuuluu piste kyseiseen joukkoon ja vastaavalle pikselille annetaan väri 1 eli musta. Hajaantuville pisteille annetaan väri 1 - 15 (2=tummansininen, 15=kirkkaanpunainen) hajaantumisnopeuden mukaan siten, että väli [0, max_iterations-1] jaetaan tasavälisesti 15 osaan, jolloin samaan väliin kuuluvat n:n arvot vastaavat samaa väriä.

2.2 Julian esikatselu inverse function -tekniikalla

Inverse function -menetelmän perusideana on soveltaa Julian rekursiokaavaa käänteiseseti, eli z[n+1] = sqrt(z[n] - c), z[0] on mielivaltaisesti valittu alkuarvo. Tällöin *jokaisella* iteraatiokerralla saadaan jokin Julian joukon piste z[n], tai itse asiassa kaksi pistettä, ±z[n]. Tulokset voisi kerätä binaaripuuhun ja käydä läpi kaikki haarat, mutta menetelmä on turhan työläs nopeaa esikatselua varten, joten käytännössä joka iteraatiolla valitaan haaroista toinen. Tällöin ei tarvita mitään tietorakenteita kulkemattomien polun alkujen tallennukseen, vaan voidaan edetä suoraviivaisesti silmukassa. Kuljettava haara voidaan tietysti valita millä periaatteella tahansa, mutta samalla on huomioitava käänteismenetelmän ilmeinen heikkous; etukäteen ei voi tietää, kuinka suuren osan joukosta iteraatio kattaa. Onhan mahdollista, että se juuttuu vain muutaman pisteen mittaiselle radalle, jolloin esikatselusta ei paljoakaan hyötyä ole. Yksinkertainen ja tehokas menetelmä tämän välttämiseksi on satunnainen juurien valinta random()-funktiolla, jota tässäkin ohjelmassa käytetään. Mikäli tällöin samaan pisteeseen joudutaan toistamiseen, saatetaan silti päästä eri radalle kuin edellisellä kerralla.

Toinen ongelma on kompleksisten neliöjuurien laskeminen. Olkoon z = x + iy luvun a + ib neliöjuuri, siis

$$(x+iy)^2 = a+ib$$

$$\Leftrightarrow \begin{cases} x^2 - y^2 = a \\ 2xy = b \end{cases}$$

$\underline{x \neq 0:}$

$$y = \frac{b}{2x}$$

$$\Rightarrow 4x^4 - 4ax^2 - b^2 = 0$$

$$\Leftrightarrow x^2 = \frac{a \pm \sqrt{a^2+b^2}}{2}$$

$$\begin{cases} x = \pm\sqrt{\frac{a+\sqrt{a^2+b^2}}{2}} \\ y = \frac{b}{2x} \end{cases}$$

$\underline{x = 0:}$

$$y = \pm\sqrt{-a}$$

Jälkimmäistä vaihtoehtoa, x = 0, tarvitaan äärimmäisen harvoin; sehän merkitsee, että juurrettava on negatiivinen reaaliluku, eli kompleksiosan on doublen tarkkuudella (n. 18 desimaalia) oltava = 0. Tietysti sekin mahdollisuus on huomioitava ettei synny nollalla jakamista.
Koska tällä menetelmällä laskettuna piste joko kuuluu joukkoon tai ei kuulu, ei pisteitä voida värittää hajaantumisnopeuden mukaan kuten varsinaisessa kuvaajassa, joten värejä ei tarvita kuin kaksi. Ohjelmassa tämä on toteutettu siten, että ikkunan oikeasta yläkulmasta varataan neliön muotoinen alue, jonka sivu on kolmasosa ikkunan lyhyemmästä sivusta. Alue maalataan mustaksi ja siihen plotataan valkoisella iteraation tuloksena saadut pisteet.

2.3 Käyttöliittymä

Ohjelma toimii täysin graafisesti yhdessä ikkunassa ilman minkäänlaisia valikoita, joten käyttöliittymä on varsin pelkistetty. Täysipainoinen käyttö vaatiikin kokeilumieltä tai käyttöohjeeseen

perehtymistä, varsinkaan näppäinkomennot eivät ole mitenkään itsestäänselviä, vaikkakin (toivottavasti) helposti muistettavia.
SRGP:n input-funktioita käytetään sekä hiiren että näppäimistön lukemiseen silmukassa, jonka ctrl-x:n painaminen katkaisee.
Näppäimistöä luetaan EVENT-moodissa, jolloin näppäinpuskuri on käytössä eikä painalluksia menetetä esim. uudelleenmaalauksenkaan aikana. Syöttömoodi on RAW, siis näppäin kerrallaan, paitsi i-komennon jälkeen. Tällöin ikkunan alareunaan tulostetaan kehote uuden iteraatiomaksimin syöttämisestä, joka sitten luetaan EDIT-moodissa funktiolla SRGP_waitEvent(-1), eli odotetaan, kunnes enteriä painetaan. Ohjelma on tällöin muilta osin "pois päältä", siis esim. hiirisyötteeseen ei reagoida mitenkään.
Hiiren lukumoodi on SAMPLE, koska hiirisyötteen puskurionti ei ole järkevää. Kun vasen nappi painetaan pohjaan, muutetaan kursorin tyypiksi RUBBER_RECT osoittamaan zoomausaluetta; vastaavasti oikeaa nappia painettaessa kursori muutetaan RUBBER_LINEksi osoittamaan vetomatkaa. Näin päästään aavistuksen verran helpommalla kuin xor-piirtoa käyttäen.
Silmukan lopuksi tarkistetaan, onko hiiri liikkunut edelliskerrasta. Jos näin on, päivitetään koordinaattinäyttö ja Julia preview, mikäli ne ovat käytössä.

3. Ohjelman toiminnan kuvaus

Ohjelma on jaettu 3 lähdekooditiedostoon: main.c sisältää käynnistettäessä suoritettavat alustukset sekä pääsilmukan, joka lukee hiirtä ja näppäimistöä; graph.c sisältää grafiikan tulostukseen ja käsittelyyn liittyvät funktiot; calc.c sisältää puhtaasti laskennallisia funktioita.

3.1 Funktioiden kuvaus

main.c:

int resize_callback(int, int);
Funktiota kutsutaan käyttäjän muutettua ikkunan kokoa. Asettaa kuvaajan koon koko ikkunan kokoiseksi ja piirtää kuvan uudelleen.

void init_colors();
Lataa väripalettiin värit 2 - 15 karkeana väriliukuna siten, että 2 on tummansininen ja 15 kirkkaanpunainen.

int input_value(rectangle, int);
Tulostaa annetun suorakulmion sisään tekstin "Max iterations (*oletusarvo*):" ja pyytää käyttäjältä uuden iteraatiomaksimin, jonka pituus on enintään INPUT_BUFFER_LEN-1. Lukua syötettäessä ohjelma ei reagoi mihinkään muuhun syötteeseen. Palauttaa annetun arvon tai virheellisellä syötteellä oletusarvon.

void process_input();
Pääsilmukka, joka lukee hiirtä ja näppäimistöä ja kutsuu niiden perusteella alempia funktioita.

graph.c:

repaint(infoptr, rectangle);
Piirtää infoptrin osoittamasta Mandelbrotin/Julian joukosta annetun suorakulmion rajaaman alueen. Tässä ohjelmassa alue on aina joko koko ikkuna tai koordinaattinäytön tai Julia previewin peittämä alue.

void zoom_in(infoptr, point, point);
Muuttaa kuvaajan zoomausta ja origoa siten, että annettujen pisteiden välinen suorakulmainen alue täyttää uudessa kuvaajassa koko ikkunan.

void slide(infoptr, point, point);
Siirtää kuvaajaa ikkunassa sivusuunnassa, siis zoomausta muuttamatta, siten, että vetämisen aloituspistessä oleva kohta siirtyy lopetuspisteeseen.

void change_set(infoptr, point);
Vaihtaa kuvaajan Mandelbrotista Juliaan tai päinvastoin. Annettua pistettä käytetään Julian vakiona.

void print_coords(infoptr, point, rectangle);
Tulostaa ikkunan vasempaan alakulmaan annetun pisteen koordinaatit kompleksitason koordinaateiksi muunnettuna 17 desimaalin tarkkuudella. Tarkkuuden valintaan on syynä doublen tarkkuus, n. 18 merkitsevää numeroa.

paint_julia_prev(infoptr, point);
Piirtää ikkunan oikeaan yläkulmaan annettua pistettä vastaavan Julian joukon esikatselukuvan inverse function-tekniikalla. JULIA_LIMIT (=1000) määrää iteraatiokertojen lukumäärän, eikä käyttäjä voi sitä muuttaa.

calc.c:

dpoint to_dpoint(point, infoptr);
Muuntaa grafiikkaikkunan pikselien koordinaatit infoptrin osoittaman osoittaman Mandelbrotin/Julian joukon koordinaateiksi.

point to_point(dpoint, infoptr);
Muuntaa Mandelbrotin/Julian joukon koordinaatit vastaaviksi ikkunan koordinaateiksi.

void init_coords(infoptr);
Alustaa kuvaajan koordinaatiston, kun se näytetään ensimmäistä kertaa. Kuva-alueen vasemmassa alakulmassa on piste -2-2i ja kuvaajan leveys ja korkeus ovat vähintään 4 yksikköä.

void fit_region(infoptr, double, double);
Muuttaa kuvaajan skaalauksen siten, että annetun levyinen ja korkuinen kompleksialue täyttää ikkunan; mikäli kuva-alue ja sovitettava kompleksialue ovat erimuotoisia, valitaan skaala niin, että alue sopii ikkunaan kokonaan eikä leikkaudu.

int iterate(dpoint, dpoint, int);
Suorittaa Mandelbrot/Julia -iteraation ja palauttaa iteraatiokertojen määrän.

int select_color(int, int);
Käytettyjen iteraatiokertojen ja maksimi-iteraatiomäärän perusteella valitsee pisteelle värin 1-15 kohdassa 2.1 esitetyllä tavalla.

dpoint c_sqrt(dpoint);
Palauttaa annetun kompleksiluvun neliöjuuren.

rectangle get_prev_rect(infoptr);
Palautaa Julia-esikatselualueen, eli ikkunan oikeassa yläkulmassa olevan neliön, jonka sivu on kolmasosa ikkunan lyhyemmästä sivusta.

3.2 Tietorakenteet ja algoritmit

Ohjelma hyödyntää SRGP:n tyyppejä point ja rectangle, joiden lisäksi on määritelty vastaavanlainen dpoint. Siihen voidaan tallentaa pisteen koordinaatit doublen tarkkuudella. Käytännössä näitä hyödynnetään siten, että point-muuttujat sisältävät graafisia pikselikoordinaatteja ja dpoint-muuttujat Mandelbrotin ja Julian joukkojen liukulukukoordinaatteja. Muunnokset näiden välillä suoritetaan funktioilla to_dpoint() ja to_point().

Kaikkein olennaisin tietorakenne on kuitenkin määritelty seuraavasti:

```
struct picture_info{
  rectangle area;
  int       is_mandel,
            max_iterations;
  dpoint    orig,
            julia_c;
  double    scale;
};
```

Tämä karakterisoi täysin ruudulla esitettävän fraktaalin. Samaa rakennetta käytetään sekä varsinaisen kuvaajan yhteydessä, että Julia preview -ruudun tietojen tallennukseen. Jälkimmäisessä tapauksessa kentillä is_mandel ja max_iterations ei kuitenkaan ole käyttöä.
Kenttien merkitykset ovat seuraavanlaiset: area tarkoitaa sitä SRGP-ikkunan suorakulmiota, jonka sisään kuvaaja piirretään. Mikäli kyseessä on pääkuvio eikä siis esikatseluruutu, area rajaa aina koko ikkunan.
is_mandel kertoo onko kuvaaja Mandelbrot (TRUE) vai Julia (FALSE). max_iterations tarkoittaa tietenkin yhden pisteen laskemiseksi käytettävien iteraatioiden maksimimäärää. orig sisältää sen kompleksitason pisteen, jonka kuva on ikkunassa kohdalla area.bottom_left.
julia_c on Julia-iteraation vakiotermi. scale tarkoittaa Mandelbrotin/Julian yksikköjärjestelmässä mitatun pituuden suhdetta vastaavaan pituuteen pikseleissä. Se on siis periaatteessa zoomauskertoimen käänteisluku.

Tärkeät algoritmit on selvitetty kohdissa 2.1 ja 2.2.

3.3 Globaalit muuttujat

Ohjelmassa on yksi globaali muuttuja:

```
struct picture_info picture;
```

Valintaan on selkeä ja pakottava syy: kun käyttäjä muuttaa ikkunan kokoa, kutsutaan SRGP_registerResizeCallback()-funktiolla rekisteröityä funktiota resize_callback(). Tämän tehtävänä on piirtää kuvaaja uudelleen, mutta se onnistuu vain, jos funktio saa tarpeelliset kuvaajaa yksilöivät parametrit. Koska kuitenkin ko. funktiota kutsuu X-Windows eikä itse ohjelma, ei tarvittavia tietoja voida välitää funktioparametreinä, vaan ne on sijoitettava globaaliin muuttujaan.
Funktioiden monikäyttöisyys ei tästä kuitenkaan kärsi; muuttujaa käyttävät suoraan ainoastaan em. resize_callback() sekä päätason funktiot main() ja process_input(). Muille funktioille vastaava tieto välitetään osoitinparametrina.

4. Arvio ohjelmasta ja parannusehdotuksia

Ohjelman selvimpänä heikkoutena ovat näppäinkomennot - ohjelma itsessään ei sisällä mitään tietoa siitä, mitä mikäkin näppäin tekee, vaan ne on opeteltava käyttöohjeesta. Kuitenkin koska tärkeimmät toiminnot suoritetaan hiirellä ja koska näppäinkomentoja on vain muutama, olisi valikkojen käyttäminen ollut ehkä hieman liioiteltua. Tosin jo parinkin uuden toiminnon lisääminen käytännössä vaatisi valikkojen käyttöönottoa.

Nykyisellään tarkasteltava joukko täyttää kokonaisuudessaan ohjelman ikkunan, mutta tietorakenteet ja grafiikkafunktiot sallivat kuvaajan rajoittamisen mille tahansa ikkunan suorakulmaiselle osa-alueelle. Niinpä ikkuna voitaisiin vaikkapa jakaa kahtia ja esittää Mandelbrotin ja Julian joukot samanaikaisesti. Tämä kuitenkin vaatisi käyttöliittymäfunktioiden mittavaa uudelleenkirjoittamista.

Eräs vaikeasti kierrettävä ongelma aiheutuu koneen laskentatarkkuudesta; double-tyypin muuttujassa on noin 18 merkitsevää numeroa, mikä käytännössä tarkoittaa sitä, että jos kuvaa zoomataan noin 10^{16}-kertaiseksi alkutilasta, se muuttuu palikkamaiseksi eikä uusia yksityiskohtia enää saada. Vaikka em. luku tuntuu suurelta, on se varsin helposti saavutettavissa.

Pienempiä, osittain SRGP:stä johtuvia kauneusvirheitä:

- RUBBER_RECT- ja RUBBER_LINE-tyyppiset kursorit piirtyvät alpha- ja hp-koneissa eri väreillä. Ellei alla oleva piirtoväri ole musta, näkyvät ne alphoissa erittäin huonosti, mikä saattaa haitata työskentelyä.
- Ikkunan maksimointinappi tekee ikkunasta monta kertaa koko työpöytää suuremman, eli on käyttökelvoton.
- Jos ikkunan korkeutta muutetaan silloin kun ruudulla on iteraatiomaksimin syöttörivi, ei rivi siirrykään uuden ikkunan alareunaan, vaan jää roikkumaan aikaisemmalle korkeudelleen ikkunan yläreunasta laskettuna. Myös kaikki rivin tekstit pyyhkiytyvät tietysti pois.

5. Suunnitelmaan tehdyt muutokset

Suurin muutos on Julian esikatselumahdollisuuden lisääminen siihen liittyvine algoritmeineen sekä grafiikka- ja käyttöliittymätoimintoineen.

Alkuperäisessä suunnitelmassa jäi avoimeksi kuhunkin tilanteeseen sopivien iteraatiomaksimien määrittäminen. Ongelma osoittautui varsin hankalaksi, ja sen onkin nyt jätetty käyttäjän ratkaistavaksi; i-komennolla voi syöttää minkä tahansa maksimiarvon. Tämän menettelyn etuna on ensinnäkin se, että tällöin on mahdollista zoomata nopeasti pienillä iteraatiomäärillä jollekin mielenkiintoiselle alueelle. Toisekseen näin voidaan helposti tutkia iteraatiomäärien vaikutusta kuvion muotoutumiseen.

Pienempänä muutoksena on näppäinkomennon z, zoom out, lisääminen. Hiirellä ulos zoomaaminen on mahdollista vetämällä zoomausalue riittävän paljon ikkunan rajojen ulkopuolelle. Tämä on kuitenkin melko tehotonta, jos ikkuna on suurikokoinen.