emndl

Claude Heiland-Allen

2011–2019

Contents

1 BUGS

```
git     soundtrack breaks if `pwd` contains certain characters (blame pd)
v0.2      --speed default missing, workaround: set it explicitly
v0.1    --auto breaks if emndl_autotune not installed in dirname(emndl.sh)
```

2 complex.h

```
//minimal implementation
#ifndef MY_COMPLEX_TYPE
#define MY_COMPLEX_TYPE

#include <cmath>

template<typename T>
class fcomplex{
public:
    fcomplex(){re=T(0.); im=T(0.);}
    fcomplex(T rv){re=rv; im=T(0.);}
    fcomplex(T rv, T iv){re=rv; im=iv;}
    fcomplex(const fcomplex<T>& cv){re=cv.re; im=cv.im;}
        template<typename U> fcomplex(const fcomplex<U>& cv){re=T(cv.real()); im=T(cv.imag());}
    fcomplex<T>& operator=(const fcomplex<T>& cv){re=cv.re; im=cv.im; return *this;}
    fcomplex<T>& operator+=(const fcomplex<T>& cv){re+=cv.re; im+=cv.im; return *this;}
    fcomplex<T>& operator-=(const fcomplex<T>& cv){re-=cv.re; im-=cv.im; return *this;}
        fcomplex<T>& operator*=(const fcomplex<T>& cv){T re1 = re*cv.re-im*cv.im; im=re*cv.im+im*cv.re; re=re1; return *this;}

    T& real(){return re;}
    const T& real()const {return re;}
    T& imag(){return im;}
    const T& imag()const {return im;}
public:
    T re,im;
};

template<typename T>
inline fcomplex<T> operator+(const fcomplex<T> &c1, const fcomplex<T>& c2){
    return fcomplex<T>(c1.re+c2.re, c1.im+c2.im);
}

template<typename T>
inline fcomplex<T> operator+(T c1, const fcomplex<T>& c2){
    return fcomplex<T>(c1+c2.re, c2.im);
}

template<typename T>
inline fcomplex<T> operator+(const fcomplex<T> &c1, T c2){
    return fcomplex<T>(c1.re+c2, c1.im);
}

template<typename T>
inline fcomplex<T> operator-(const fcomplex<T> &c1, const fcomplex<T>& c2){
    return fcomplex<T>(c1.re-c2.re, c1.im-c2.im);
}

template<typename T>
inline fcomplex<T> operator-(const fcomplex<T> &c){
    return fcomplex<T>(-c.re, -c.im);
```

```
}

template<typename T>
inline fcomplex<T> operator*(const fcomplex<T>& c1, const fcomplex<T>& c2){
    return fcomplex<T>(c1.re*c2.re-c1.im*c2.im, c1.re*c2.im+c1.im*c2.re);
}

template<typename T>
inline fcomplex<T> operator*(T c1, const fcomplex<T>& c2){
    return fcomplex<T>(c1*c2.re, c1*c2.im);
}

template<typename T>
inline fcomplex<T> operator*(const fcomplex<T>& c2, T c1){
    return fcomplex<T>(c1*c2.re, c1*c2.im);
}

template<typename T>
inline fcomplex<T> operator/(const fcomplex<T> &c1, const fcomplex<T>& c2){
    return T(1.)/norm(c2) * c1*conjugate(c2);
}

template<typename T>
inline fcomplex<T> operator/(const fcomplex<T> &c1, T c2){
    return T(1.)/c2 * c1;
}

template<typename T>
inline fcomplex<T> operator/(T c1, const fcomplex<T>& c2){
    return T(1.)/norm(c2) * c1 * conjugate(c2);
}

template<typename T>
inline std::ostream& operator<<(std::ostream& os, const fcomplex<T>& c)
{
    os << '(' << c.real() << ',' << c.imag() << ')';
        return os;
}

/*
template<typename T>
inline std::istream& operator>>(std::istream &is, fcomplex<T>& c)
{
    // TODO: use cout::hexfloat and other flags to setup base
    std::string tmp;
    is >> tmp;
    mpfr_set_str(v.mpfr_ptr(), tmp.c_str(), 10, mpreal::get_default_rnd());
    return is;
}*/

//Functions
template<typename T>
inline fcomplex<T> conjugate(const fcomplex<T> &c){
    return fcomplex<T>(c.re, -c.im);
}

template<typename T>
```

```
inline fcomplex<T> sqr(const fcomplex<T> &c){
    return fcomplex<T>(c.re*c.re-c.im*c.im, T(2.)*c.re*c.im);
}

template<typename T>
inline T norm(const fcomplex<T> &c){
    return c.re*c.re+c.im*c.im;
}

template<typename T>
inline T abs(const fcomplex<T> &c){
  using std::sqrt;
    return sqrt(norm(c));
}

template<typename T>
inline T real(const fcomplex<T> &c){
    return c.real();
}

template<typename T>
inline T imag(const fcomplex<T> &c){
    return c.imag();
}

template<typename T>
inline fcomplex<T> pow(const fcomplex<T> &c, int p){
    T r = abs(c);
        T theta = std::atan2(c.real(), c.imag());
        r = std::pow(r,p);
        theta *= p;
        return fcomplex<T>(r*cos(theta),r*sin(theta));
}

#endif // COMPLEX
```

3 Complex.hs

```
{-
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011,2018  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
-}

module Complex where
```

```
20
    -- this is ugly: can't use Data.Complex because no RealFloat for MPFR
    data Complex c = !c :+ !c deriving (Read, Show, Eq)

    -- complex number arithmetic, with extra strictness and cost-centres
25  instance Num c => Num (Complex c) where
      (a :+ b) + (c :+ d) = (a + c) :+ (b + d)
      (a :+ b) - (c :+ d) = (a - c) :+ (b - d)
      (a :+ b) * (c :+ d) = (a * c - b * d) :+ (a * d + b * c)
      negate (a :+ b) = negate a :+ negate b
30    abs x = error $ "Complex.abs"
      signum x = error $ "Complex.signum"
      fromInteger x = fromInteger x :+ 0

    instance (Fractional a) => Fractional (Complex a)  where
35    (a:+b) / (c:+d) = ((a*c + b*d) / m) :+ ((b*c - a*d) / m) where m = c*c + d*d
      fromRational a = fromRational a :+ 0

    magnitude :: (Floating c) => Complex c -> c
    magnitude = sqrt . magnitude2
40
    magnitude2 :: (Num c) => Complex c -> c
    magnitude2 (re:+im) = re * re + im * im

    cis :: Floating c => c -> Complex c
45  cis a = cos a :+ sin a

    mkPolar :: Floating c => c -> c -> Complex c
    mkPolar r a = (r * cos a) :+ (r * sin a)

50  phase :: (Floating c, Ord c) => Complex c -> c
    phase (re:+im) = atan2' im re

    atan2' :: (Floating c, Ord c) => c -> c -> c
    atan2' y x
55    | x > 0            =  atan (y / x)
      | x == 0 && y > 0  =  pi/2
      | x <  0 && y > 0  =  pi + atan (y / x)
      | x <= 0 && y < 0  = -atan2' (-y) x
      | y == 0 && x < 0  =  pi
60    | x == 0 && y == 0 =  y
      | otherwise = error "Complex.atan2'"

    normalize :: Floating c => Complex c -> Complex c
    normalize z@(re:+im) = let m = magnitude z in (re / m) :+ (im / m)
```

4 COPYING

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

```
 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
5 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
```

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other

than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program,

in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be

included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by

this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of

rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is

conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future

versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

5 emndl_audio.pd

```
#N canvas 0 0 548 531 10;
#X obj 11 198 catch~ \$0-o-l;
#X obj 100 198 catch~ \$0-o-r;
#X obj 35 178 send~ \$0-i-l;
#X obj 129 178 send~ \$0-i-r;
#X obj 218 149 send~ \$0-f;
#X obj 30 347 emndl_compress~ 36;
#X obj 73 113 noise~;
#X obj 166 115 noise~;
#X obj 48 156 rpole~ 0.975;
#X obj 138 156 rpole~ 0.975;
#X obj 351 498 table \$0-in-l;
#X obj 351 474 table \$0-in-r;
#X obj 49 39 tabplay~ \$0-in-l;
#X obj 159 39 tabplay~ \$0-in-r;
#X obj 322 108 r emndl;
```

```
#X obj 206 346 pack s s s;
#X obj 206 388 soundfiler;
#X obj 226 321 symbol \$0-in-l;
#X obj 322 320 symbol \$0-in-r;
#X obj 26 388 tabwrite~ \$0-out-l;
#X obj 342 426 table \$0-out-l;
#X obj 342 447 table \$0-out-r;
#X msg 206 410 resize \$1;
#X obj 197 432 s \$0-out-l;
#X obj 215 452 s \$0-out-r;
#X obj 62 411 tabwrite~ \$0-out-r;
#X obj 251 214 delay 1000;
#X obj 176 251 unpack s s;
#X obj 251 252 symbol;
#X obj 263 294 pack s s s;
#X obj 285 233 t s b b;
#X obj 302 254 symbol \$0-out-l;
#X obj 324 273 symbol \$0-out-r;
#X msg 279 347 write -wave -bytes 2 -rate 48000 \$1 \$2 \$3;
#X obj 279 389 soundfiler;
#X obj 47 457 delay 1000;
#X msg 47 498 \; pd quit;
#X msg 208 502 \; emndl in.wav out.wav;
#X obj 51 369 dac~;
#X msg 206 367 read -maxsize 2.88e+08 -resize \$1 \$2 \$3;
#X obj 137 134 *~;
#X obj 47 134 *~;
#X obj 83 18 vline~;
#X obj 31 279 *~;
#X obj 81 279 *~;
#X obj 47 478 spigot 1;
#X obj 104 479 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0
1;
#X obj 160 81 emndl_comb~;
#N canvas 0 0 658 306 \$0-filters 0;
#X obj 370 184 emndl_filter 32 \$0;
#X obj 11 16 emndl_filter 1 \$0;
#X obj 130 184 emndl_filter 16 \$0;
#X obj 12 39 emndl_filter 2 \$0;
#X obj 11 61 emndl_filter 3 \$0;
#X obj 11 88 emndl_filter 4 \$0;
#X obj 12 115 emndl_filter 5 \$0;
#X obj 12 136 emndl_filter 6 \$0;
#X obj 12 157 emndl_filter 7 \$0;
#X obj 10 184 emndl_filter 8 \$0;
#X obj 132 39 emndl_filter 10 \$0;
#X obj 131 88 emndl_filter 12 \$0;
#X obj 132 136 emndl_filter 14 \$0;
#X obj 251 88 emndl_filter 20 \$0;
#X obj 250 184 emndl_filter 24 \$0;
#X obj 371 88 emndl_filter 28 \$0;
#X restore 215 186 pd \$0-filters;
#X obj 219 131 sig~ 45;
#X obj 160 61 hip~ 2.5;
#X obj 50 60 hip~ 2.5;
#X obj 50 81 emndl_comb~;
#X msg 132 19 0 \, 1 1000;
```

```
#X obj 25 300 emndl_compress~ 48;
#X obj 35 320 emndl_compress~ 48;
#X obj 166 136 *~ 1e-09;
#X obj 73 134 *~ 1e-09;
#X obj 31 255 rev3~ 120 71 11520 10;
#X obj 31 229 -~;
#X obj 60 229 +~;
#X obj 174 274 t b s;
#X obj 215 275 t s b b;
#X obj 174 320 t b b;
#X obj 173 300 delay 1000;
#X connect 0 0 59 0;
#X connect 0 0 60 0;
#X connect 1 0 59 1;
#X connect 1 0 60 1;
#X connect 5 0 19 0;
#X connect 5 0 38 0;
#X connect 5 1 25 0;
#X connect 5 1 38 1;
#X connect 6 0 57 0;
#X connect 7 0 56 0;
#X connect 8 0 2 0;
#X connect 9 0 3 0;
#X connect 12 0 51 0;
#X connect 13 0 50 0;
#X connect 13 1 26 0;
#X connect 14 0 27 0;
#X connect 15 0 39 0;
#X connect 16 0 22 0;
#X connect 17 0 15 1;
#X connect 18 0 15 2;
#X connect 22 0 23 0;
#X connect 22 0 24 0;
#X connect 26 0 28 0;
#X connect 26 0 35 0;
#X connect 27 0 61 0;
#X connect 27 1 30 0;
#X connect 28 0 29 0;
#X connect 29 0 33 0;
#X connect 30 0 28 1;
#X connect 30 1 31 0;
#X connect 30 2 32 0;
#X connect 31 0 29 1;
#X connect 32 0 29 2;
#X connect 33 0 34 0;
#X connect 35 0 45 0;
#X connect 39 0 16 0;
#X connect 40 0 9 0;
#X connect 41 0 8 0;
#X connect 42 0 40 1;
#X connect 42 0 41 1;
#X connect 42 0 43 1;
#X connect 42 0 44 1;
#X connect 43 0 54 0;
#X connect 43 0 54 1;
#X connect 44 0 55 0;
#X connect 44 0 55 1;
```

```
#X connect 45 0 36 0;
#X connect 46 0 45 1;
#X connect 47 0 40 0;
#X connect 49 0 4 0;
#X connect 50 0 47 0;
#X connect 51 0 52 0;
#X connect 52 0 41 0;
#X connect 53 0 42 0;
#X connect 54 0 5 0;
#X connect 54 1 5 0;
#X connect 55 0 5 1;
#X connect 55 1 5 1;
#X connect 56 0 3 0;
#X connect 57 0 2 0;
#X connect 58 0 43 0;
#X connect 58 1 44 0;
#X connect 59 0 58 0;
#X connect 60 0 58 1;
#X connect 61 0 64 0;
#X connect 61 1 62 0;
#X connect 62 0 15 0;
#X connect 62 1 17 0;
#X connect 62 2 18 0;
#X connect 63 0 12 0;
#X connect 63 0 19 0;
#X connect 63 1 13 0;
#X connect 63 1 25 0;
#X connect 63 1 53 0;
#X connect 64 0 63 0;
```

6 emndl_automu.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <mpfr.h>

#if MPFR_VERSION_MAJOR < 3
#define MPFR_RNDN GMP_RNDN
#endif

int max(int a, int b) { return a > b ? a : b; }

void progressReport(int c) { fputc(c, stderr); fflush(stderr); }

int muatom(int period, mpfr_t x, mpfr_t y, mpfr_t z) {
  const mpfr_prec_t accuracy = 12;
  mpfr_prec_t bits = max(mpfr_get_prec(x), mpfr_get_prec(y));
#define VARS nx, ny, bx, by, wx, wy, zz, Ax, Ay, Bx, By, Cx, Cy, Dx, Dy, Ex, Ey,↙
    ↳ t1, t2, t3, t4, t5, t6, t7, t8, t9, t0, t01, t02, t03, t04
  mpfr_t VARS;
  mpfr_inits2(bits, VARS, (mpfr_ptr) 0);
  mpfr_set(nx, x, MPFR_RNDN);
  mpfr_set(ny, y, MPFR_RNDN);
  mpfr_set_prec(zz, mpfr_get_prec(z));
  int n_ok, w_ok, b_ok;
```

```
25    int r = 0;
      progressReport('\n');
      do { progressReport('X');
        mpfr_set_prec(t0,  bits - accuracy);
        mpfr_set_prec(t01, bits - accuracy);
30      mpfr_set_prec(t02, bits - accuracy);
        mpfr_set_prec(t03, bits - accuracy);
        mpfr_set_prec(t04, bits - accuracy);
        int limit = 40;
        do { progressReport('n');
35        // refine nucleus
          mpfr_set_ui(Ax, 0, MPFR_RNDN);
          mpfr_set_ui(Ay, 0, MPFR_RNDN);
          mpfr_set_ui(Dx, 0, MPFR_RNDN);
          mpfr_set_ui(Dy, 0, MPFR_RNDN);
40        for (int i = 0; i < period; ++i) {
            // D <- 2AD + 1   : Dx <- 2 (Ax Dx - Ay Dy) + 1 ; Dy = 2 (Ax Dy + Ay Dx)
            mpfr_mul(t1, Ax, Dx, MPFR_RNDN);
            mpfr_mul(t2, Ay, Dy, MPFR_RNDN);
            mpfr_mul(t3, Ax, Dy, MPFR_RNDN);
45          mpfr_mul(t4, Ay, Dx, MPFR_RNDN);
            mpfr_sub(Dx, t1, t2, MPFR_RNDN);
            mpfr_mul_2ui(Dx, Dx, 1, MPFR_RNDN);
            mpfr_add_ui(Dx, Dx, 1, MPFR_RNDN);
            mpfr_add(Dy, t3, t4, MPFR_RNDN);
50          mpfr_mul_2ui(Dy, Dy, 1, MPFR_RNDN);
            // A <- A^2 + n   : Ax <- Ax^2 - Ay^2 + nx ; Ay <- 2 Ax Ay + ny
            mpfr_sqr(t1, Ax, MPFR_RNDN);
            mpfr_sqr(t2, Ay, MPFR_RNDN);
            mpfr_mul(t3, Ax, Ay, MPFR_RNDN);
55          mpfr_sub(Ax, t1, t2, MPFR_RNDN);
            mpfr_add(Ax, Ax, nx, MPFR_RNDN);
            mpfr_mul_2ui(t3, t3, 1, MPFR_RNDN);
            mpfr_add(Ay, t3, ny,MPFR_RNDN);
          }
60        // n <- n - A / D  = (nx + ny i) - ((Ax Dx + Ay Dy) + (Ay Dx - Ax Dy)i) / ↙
              ↪ (Dx^2 + Dy^2)
          mpfr_sqr(t1, Dx, MPFR_RNDN);
          mpfr_sqr(t2, Dy, MPFR_RNDN);
          mpfr_add(t1, t1, t2, MPFR_RNDN);

65        mpfr_mul(t2, Ax, Dx, MPFR_RNDN);
          mpfr_mul(t3, Ay, Dy, MPFR_RNDN);
          mpfr_add(t2, t2, t3, MPFR_RNDN);
          mpfr_div(t2, t2, t1, MPFR_RNDN);
          mpfr_sub(t2, nx, t2, MPFR_RNDN);
70
          mpfr_mul(t3, Ay, Dx, MPFR_RNDN);
          mpfr_mul(t4, Ax, Dy, MPFR_RNDN);
          mpfr_sub(t3, t3, t4, MPFR_RNDN);
          mpfr_div(t3, t3, t1, MPFR_RNDN);
75        mpfr_sub(t3, ny, t3, MPFR_RNDN);

          mpfr_set(t01, t2, MPFR_RNDN);
          mpfr_set(t02, t3, MPFR_RNDN);
          mpfr_set(t03, nx, MPFR_RNDN);
80        mpfr_set(t04, ny, MPFR_RNDN);
```

```
      mpfr_sub(t01, t01, t03, MPFR_RNDN);
      mpfr_sub(t02, t02, t04, MPFR_RNDN);
      n_ok = mpfr_zero_p(t01) && mpfr_zero_p(t02);

85    mpfr_set(nx, t2, MPFR_RNDN);
      mpfr_set(ny, t3, MPFR_RNDN);
    } while (! n_ok && --limit);
    if (! limit) goto cleanup;
    mpfr_set(wx, nx, MPFR_RNDN);
90  mpfr_set(wy, ny, MPFR_RNDN);
    mpfr_set(bx, nx, MPFR_RNDN);
    mpfr_set(by, ny, MPFR_RNDN);
    limit = 40;
    do { progressReport('b');
95    // refine bond
      mpfr_set(Ax, wx, MPFR_RNDN);
      mpfr_set(Ay, wy, MPFR_RNDN);
      mpfr_set_ui(Bx, 1, MPFR_RNDN);
      mpfr_set_ui(By, 0, MPFR_RNDN);
100   mpfr_set_ui(Cx, 0, MPFR_RNDN);
      mpfr_set_ui(Cy, 0, MPFR_RNDN);
      mpfr_set_ui(Dx, 0, MPFR_RNDN);
      mpfr_set_ui(Dy, 0, MPFR_RNDN);
      mpfr_set_ui(Ex, 0, MPFR_RNDN);
105   mpfr_set_ui(Ey, 0, MPFR_RNDN);
      for (int i = 0; i < period; ++i) {
        // E <- 2 ( A E + B D )
        mpfr_mul(t1, Ax, Ex, MPFR_RNDN);
        mpfr_mul(t2, Ay, Ey, MPFR_RNDN);
110     mpfr_sub(t1, t1, t2, MPFR_RNDN);
        mpfr_mul(t2, Ax, Ey, MPFR_RNDN);
        mpfr_mul(t3, Ay, Ex, MPFR_RNDN);
        mpfr_add(t2, t2, t3, MPFR_RNDN);
        mpfr_mul(t3, Bx, Dx, MPFR_RNDN);
115     mpfr_mul(t4, By, Dy, MPFR_RNDN);
        mpfr_sub(t3, t3, t4, MPFR_RNDN);
        mpfr_add(Ex, t1, t3, MPFR_RNDN);
        mpfr_mul(t3, Bx, Dy, MPFR_RNDN);
        mpfr_mul(t4, By, Dx, MPFR_RNDN);
120     mpfr_add(t3, t3, t4, MPFR_RNDN);
        mpfr_add(Ey, t2, t3, MPFR_RNDN);
        mpfr_mul_2ui(Ex, Ex, 1, MPFR_RNDN);
        mpfr_mul_2ui(Ey, Ey, 1, MPFR_RNDN);
        // D <- 2 A D + 1
125     mpfr_mul(t1, Ax, Dx, MPFR_RNDN);
        mpfr_mul(t2, Ay, Dy, MPFR_RNDN);
        mpfr_mul(t3, Ax, Dy, MPFR_RNDN);
        mpfr_mul(t4, Ay, Dx, MPFR_RNDN);
        mpfr_sub(Dx, t1, t2, MPFR_RNDN);
130     mpfr_mul_2ui(Dx, Dx, 1, MPFR_RNDN);
        mpfr_add_ui(Dx, Dx, 1, MPFR_RNDN);
        mpfr_add(Dy, t3, t4, MPFR_RNDN);
        mpfr_mul_2ui(Dy, Dy, 1, MPFR_RNDN);
        // C <- 2 ( B^2 + A C )
135     mpfr_sqr(t1, Bx, MPFR_RNDN);
        mpfr_sqr(t2, By, MPFR_RNDN);
        mpfr_sub(t1, t1, t2, MPFR_RNDN);
```

```
            mpfr_mul(t2, Bx, By, MPFR_RNDN);
            mpfr_mul_2ui(t2, t2, 1, MPFR_RNDN);
            mpfr_mul(t3, Ax, Cx, MPFR_RNDN);
            mpfr_mul(t4, Ay, Cy, MPFR_RNDN);
            mpfr_sub(t3, t3, t4, MPFR_RNDN);
            mpfr_add(t1, t1, t3, MPFR_RNDN);
            mpfr_mul(t3, Ax, Cy, MPFR_RNDN);
            mpfr_mul(t4, Ay, Cx, MPFR_RNDN);
            mpfr_add(t3, t3, t4, MPFR_RNDN);
            mpfr_add(t2, t2, t3, MPFR_RNDN);
            mpfr_mul_2ui(Cx, t1, 1, MPFR_RNDN);
            mpfr_mul_2ui(Cy, t2, 1, MPFR_RNDN);
            // B <- 2 A B
            mpfr_mul(t1, Ax, Bx, MPFR_RNDN);
            mpfr_mul(t2, Ay, By, MPFR_RNDN);
            mpfr_mul(t3, Ax, By, MPFR_RNDN);
            mpfr_mul(t4, Ay, Bx, MPFR_RNDN);
            mpfr_sub(Bx, t1, t2, MPFR_RNDN);
            mpfr_mul_2ui(Bx, Bx, 1, MPFR_RNDN);
            mpfr_add(By, t3, t4, MPFR_RNDN);
            mpfr_mul_2ui(By, By, 1, MPFR_RNDN);
            // A <- A^2 + b
            mpfr_sqr(t1, Ax, MPFR_RNDN);
            mpfr_sqr(t2, Ay, MPFR_RNDN);
            mpfr_mul(t3, Ax, Ay, MPFR_RNDN);
            mpfr_sub(Ax, t1, t2, MPFR_RNDN);
            mpfr_add(Ax, Ax, bx, MPFR_RNDN);
            mpfr_mul_2ui(t3, t3, 1, MPFR_RNDN);
            mpfr_add(Ay, t3, by,MPFR_RNDN);
        }
        // (w) <- (w) - (B-1  D)^-1 (A - w)
        // (b) <- (b)   ( C   E)    (B + 1)
        //
        // det = (B-1)E - CD
        // inv = (E    -D)
        //       (-C (B-1) / det
        //
        // w - (E(A-w) - D(B+1))/det
        // b - (-C(A-w) + (B-1)(B+1))/det    ; B^2 - 1
        //
        // A -= w
        mpfr_sub(Ax, Ax, wx, MPFR_RNDN);
        mpfr_sub(Ay, Ay, wy, MPFR_RNDN);
        // (t1,t2) = B^2 - 1
        mpfr_mul(t1, Bx, Bx, MPFR_RNDN);
        mpfr_mul(t2, By, By, MPFR_RNDN);
        mpfr_sub(t1, t1, t2, MPFR_RNDN);
        mpfr_sub_ui(t1, t1, 1, MPFR_RNDN);
        mpfr_mul(t2, Bx, By, MPFR_RNDN);
        mpfr_mul_2ui(t2, t2, 1, MPFR_RNDN);
        // (t1,t2) -= C(A-w)
        mpfr_mul(t3, Cx, Ax, MPFR_RNDN);
        mpfr_mul(t4, Cy, Ay, MPFR_RNDN);
        mpfr_sub(t3, t3, t4, MPFR_RNDN);
        mpfr_sub(t1, t1, t3, MPFR_RNDN);
        mpfr_mul(t3, Cx, Ay, MPFR_RNDN);
        mpfr_mul(t4, Cy, Ax, MPFR_RNDN);
```

```
195         mpfr_add(t3, t3, t4, MPFR_RNDN);
            mpfr_sub(t2, t2, t3, MPFR_RNDN);
            // (t3, t4) = (B-1)E - CD
            mpfr_sub_ui(t3, Bx, 1, MPFR_RNDN);
            mpfr_mul(t4, t3, Ex, MPFR_RNDN);
200         mpfr_mul(t5, By, Ey, MPFR_RNDN);
            mpfr_sub(t4, t4, t5, MPFR_RNDN);
            mpfr_mul(t5, t3, Ey, MPFR_RNDN);
            mpfr_mul(t6, By, Ex, MPFR_RNDN);
            mpfr_sub(t5, t5, t6, MPFR_RNDN);
205         mpfr_mul(t6, Cx, Dx, MPFR_RNDN);
            mpfr_mul(t7, Cy, Dy, MPFR_RNDN);
            mpfr_sub(t6, t6, t7, MPFR_RNDN);
            mpfr_sub(t3, t4, t6, MPFR_RNDN);
            mpfr_mul(t6, Cx, Dy, MPFR_RNDN);
210         mpfr_mul(t7, Cy, Dx, MPFR_RNDN);
            mpfr_add(t6, t6, t7, MPFR_RNDN);
            mpfr_sub(t4, t5, t6, MPFR_RNDN);
            // (t3, t4) = 1/(t3, t4)  ; z^-1 = z* / (z z*)
            mpfr_sqr(t5, t3, MPFR_RNDN);
215         mpfr_sqr(t6, t4, MPFR_RNDN);
            mpfr_add(t5, t5, t6, MPFR_RNDN);
            mpfr_div(t3, t3, t5, MPFR_RNDN);
            mpfr_div(t4, t4, t5, MPFR_RNDN);
            mpfr_neg(t4, t4, MPFR_RNDN);
220         // (t1, t2) *= (t3, t4)
            mpfr_mul(t5, t1, t3, MPFR_RNDN);
            mpfr_mul(t6, t2, t4, MPFR_RNDN);
            mpfr_mul(t7, t1, t4, MPFR_RNDN);
            mpfr_mul(t8, t2, t3, MPFR_RNDN);
225         mpfr_sub(t1, t5, t6, MPFR_RNDN);
            mpfr_add(t2, t7, t8, MPFR_RNDN);

            // (t1, t2) = b - (t1, t2)
            mpfr_sub(t1, bx, t1, MPFR_RNDN);
230         mpfr_sub(t2, by, t2, MPFR_RNDN);

            mpfr_set(t01, t1, MPFR_RNDN);
            mpfr_set(t02, t2, MPFR_RNDN);
            mpfr_set(t03, bx, MPFR_RNDN);
235         mpfr_set(t04, by, MPFR_RNDN);
            mpfr_sub(t01, t01, t03, MPFR_RNDN);
            mpfr_sub(t02, t02, t04, MPFR_RNDN);
            b_ok = mpfr_zero_p(t01) && mpfr_zero_p(t02);

240         // (t5, t6) = E (A-w)
            mpfr_mul(t5, Ex, Ax, MPFR_RNDN);
            mpfr_mul(t6, Ey, Ay, MPFR_RNDN);
            mpfr_sub(t5, t5, t6, MPFR_RNDN);
            mpfr_mul(t6, Ex, Ay, MPFR_RNDN);
245         mpfr_mul(t7, Ey, Ax, MPFR_RNDN);
            mpfr_add(t6, t6, t7, MPFR_RNDN);
            // B += 1
            mpfr_add_ui(Bx, Bx, 1, MPFR_RNDN);
            // (t7, t8) = D (B+1)
250         mpfr_mul(t7, Dx, Bx, MPFR_RNDN);
            mpfr_mul(t8, Dy, By, MPFR_RNDN);
```

```
        mpfr_sub(t7, t7, t8, MPFR_RNDN);
        mpfr_mul(t8, Dx, By, MPFR_RNDN);
        mpfr_mul(t9, Dy, Bx, MPFR_RNDN);
255     mpfr_add(t8, t8, t9, MPFR_RNDN);
        // (t5, t6) -= (t7, t8)
        mpfr_sub(t5, t5, t7, MPFR_RNDN);
        mpfr_sub(t6, t6, t8, MPFR_RNDN);
        // (t7, t8) = (t5, t6) * (t3, t4)
260     mpfr_mul(t7, t3, t5, MPFR_RNDN);
        mpfr_mul(t8, t4, t6, MPFR_RNDN);
        mpfr_sub(t7, t7, t8, MPFR_RNDN);
        mpfr_mul(t8, t3, t6, MPFR_RNDN);
        mpfr_mul(t9, t4, t5, MPFR_RNDN);
265     mpfr_add(t8, t8, t9, MPFR_RNDN);

        // (t3, t4) = w - (t7, t8)
        mpfr_sub(t3, wx, t7, MPFR_RNDN);
        mpfr_sub(t4, wy, t8, MPFR_RNDN);
270
        mpfr_set(t01, t3, MPFR_RNDN);
        mpfr_set(t02, t4, MPFR_RNDN);
        mpfr_set(t03, wx, MPFR_RNDN);
        mpfr_set(t04, wy, MPFR_RNDN);
275     mpfr_sub(t01, t01, t03, MPFR_RNDN);
        mpfr_sub(t02, t02, t04, MPFR_RNDN);
        w_ok = mpfr_zero_p(t01) && mpfr_zero_p(t02);

        mpfr_set(t01, t3, MPFR_RNDN);
280     mpfr_set(t02, t4, MPFR_RNDN);
        mpfr_set(t03, wx, MPFR_RNDN);
        mpfr_set(t04, wy, MPFR_RNDN);
        mpfr_sub(t01, t01, t03, MPFR_RNDN);
        mpfr_sub(t02, t02, t04, MPFR_RNDN);
285     w_ok = mpfr_zero_p(t01) && mpfr_zero_p(t02);

        mpfr_set(bx, t1, MPFR_RNDN);
        mpfr_set(by, t2, MPFR_RNDN);
        mpfr_set(wx, t3, MPFR_RNDN);
290     mpfr_set(wy, t4, MPFR_RNDN);
      } while (!(w_ok && b_ok) && --limit);
      if (! limit) goto cleanup;
      // t1 = |n - b|
      mpfr_sub(t1, nx, bx, MPFR_RNDN);
295   mpfr_sqr(t1, t1, MPFR_RNDN);
      mpfr_sub(t2, ny, by, MPFR_RNDN);
      mpfr_sqr(t2, t2, MPFR_RNDN);
      mpfr_add(t1, t1, t2, MPFR_RNDN);
      mpfr_sqrt(t1, t1, MPFR_RNDN);
300   bits = bits + accuracy;
      mpfr_prec_round(nx, bits, MPFR_RNDN);
      mpfr_prec_round(ny, bits, MPFR_RNDN);
      mpfr_prec_round(wx, bits, MPFR_RNDN);
      mpfr_prec_round(wy, bits, MPFR_RNDN);
305   mpfr_prec_round(bx, bits, MPFR_RNDN);
      mpfr_prec_round(by, bits, MPFR_RNDN);
      mpfr_set(zz, t1, MPFR_RNDN);
      mpfr_set_prec(Ax, bits);
```

```
      mpfr_set_prec(Ay, bits);
      mpfr_set_prec(Bx, bits);
      mpfr_set_prec(By, bits);
      mpfr_set_prec(Cx, bits);
      mpfr_set_prec(Cy, bits);
      mpfr_set_prec(Dx, bits);
      mpfr_set_prec(Dy, bits);
      mpfr_set_prec(Ex, bits);
      mpfr_set_prec(Ey, bits);
      mpfr_set_prec(t1, bits);
      mpfr_set_prec(t2, bits);
      mpfr_set_prec(t3, bits);
      mpfr_set_prec(t4, bits);
      mpfr_set_prec(t5, bits);
      mpfr_set_prec(t6, bits);
      mpfr_set_prec(t7, bits);
      mpfr_set_prec(t8, bits);
      mpfr_set_prec(t9, bits);
    } while (mpfr_zero_p(zz) || bits + mpfr_get_exp(zz) < mpfr_get_prec(zz) + ↙
      ↳ accuracy);

    // copy to output
    bits = accuracy - mpfr_get_exp(zz);
    r = 1;
    mpfr_set_prec(x, bits);
    mpfr_set_prec(y, bits);
    mpfr_set(x, nx, MPFR_RNDN);
    mpfr_set(y, ny, MPFR_RNDN);
    mpfr_set(z, zz, MPFR_RNDN);
  cleanup:
    mpfr_clears(VARS, (mpfr_ptr) 0);
  #undef VARS
    return r;
  }

  #define LOGSIZE 8
  #define SIZE (1 << LOGSIZE)

  struct pixel {
    enum { done, active, queued } state;
    enum { white, black } colour;
    int n;
    mpfr_t cx, cy, zx, zy, dx, dy;
  };

  struct image {
    struct pixel p[SIZE][SIZE*2];
  };

  void initialize(struct image *img, int bits) {
    for (int j = 0; j < SIZE; ++j) {
      for (int i = 0; i < SIZE*2; ++i) {
        mpfr_init2(img->p[j][i].cx, bits);
        mpfr_init2(img->p[j][i].cy, bits);
        mpfr_init2(img->p[j][i].zx, bits);
        mpfr_init2(img->p[j][i].zy, bits);
        mpfr_init2(img->p[j][i].dx, bits);
```

```
365         mpfr_init2(img->p[j][i].dy, bits);
          }
        }
      }

370   void deinitialize(struct image *img) {
        for (int j = 0; j < SIZE; ++j) {
          for (int i = 0; i < SIZE*2; ++i) {
            mpfr_clear(img->p[j][i].cx);
            mpfr_clear(img->p[j][i].cy);
375         mpfr_clear(img->p[j][i].zx);
            mpfr_clear(img->p[j][i].zy);
            mpfr_clear(img->p[j][i].dx);
            mpfr_clear(img->p[j][i].dy);
          }
380     }
      }

      void render(struct image *img, mpfr_t x, mpfr_t y, mpfr_t z) {
        mpfr_prec_t bits = max(mpfr_get_prec(x), mpfr_get_prec(y)) - 8;
385     mpfr_t cx0, cy0, cz0, zx2, zy2, zxy, zmag2, dzx, dzy, t;
        mpfr_inits2(bits, cx0, cy0, cz0, zx2, zy2, zxy, zmag2, dzx, dzy, t, (mpfr_ptr)↙
            ↳ 0);
        mpfr_set(cx0, x, MPFR_RNDN);
        mpfr_set(cy0, y, MPFR_RNDN);
        mpfr_set(cz0, z, MPFR_RNDN);
390     mpfr_div_2ui(cz0, cz0, LOGSIZE, MPFR_RNDN);
        mpfr_mul_ui(cz0, cz0, 6, MPFR_RNDN);
        double rx = rand() / (double) RAND_MAX;
        double ry = rand() / (double) RAND_MAX;
        for (int j = 0; j < SIZE; ++j) {
395       for (int i = 0; i < SIZE*2; ++i) {
            img->p[j][i].state = (i == 0 || j == 0 || i == SIZE*2 - 1 || j == SIZE - ↙
                ↳ 1) ? active : queued;
            img->p[j][i].colour = white;
            img->p[j][i].n = 0;
            mpfr_set_prec(img->p[j][i].cx, bits);
400         mpfr_set_prec(img->p[j][i].cy, bits);
            mpfr_set_prec(img->p[j][i].zx, bits);
            mpfr_set_prec(img->p[j][i].zy, bits);
            mpfr_set_prec(img->p[j][i].dx, bits);
            mpfr_set_prec(img->p[j][i].dy, bits);
405         mpfr_set(img->p[j][i].cx, cz0, MPFR_RNDN);
            mpfr_mul_d(img->p[j][i].cx, img->p[j][i].cx, (i - SIZE + rx)/2, MPFR_RNDN)↙
                ↳ ;
            mpfr_add(img->p[j][i].cx, img->p[j][i].cx, cx0, MPFR_RNDN);
            mpfr_set(img->p[j][i].cy, cz0, MPFR_RNDN);
            mpfr_mul_d(img->p[j][i].cy, img->p[j][i].cy, j - SIZE/2 + ry, MPFR_RNDN);
410         mpfr_add(img->p[j][i].cy, img->p[j][i].cy, cy0, MPFR_RNDN);
            mpfr_set_ui(img->p[j][i].zx, 0, MPFR_RNDN);
            mpfr_set_ui(img->p[j][i].zy, 0, MPFR_RNDN);
            mpfr_set_ui(img->p[j][i].dx, 0, MPFR_RNDN);
            mpfr_set_ui(img->p[j][i].dy, 0, MPFR_RNDN);
415       }
        }
        int progressed = 0;
        int progress = 1;
```

```
  int maxiters = 256;
420 while ((!progressed) || progress > LOGSIZE) {
    progressReport('.');
    progress = 0;
    int finished = 0;
    while (! finished) {
425   finished = 1;
      for (int j = 0; j < SIZE; ++j) {
        for (int i = 0; i < SIZE*2; ++i) {
          if (active == img->p[j][i].state && img->p[j][i].n < maxiters) {
            finished = 0;
430         int n = img->p[j][i].n;
            mpfr_t *cx = &img->p[j][i].cx;
            mpfr_t *cy = &img->p[j][i].cy;
            mpfr_t *zx = &img->p[j][i].zx;
            mpfr_t *zy = &img->p[j][i].zy;
435         mpfr_t *dx = &img->p[j][i].dx;
            mpfr_t *dy = &img->p[j][i].dy;
            // |z|^2
            mpfr_sqr(zx2, *zx, MPFR_RNDN);
            mpfr_sqr(zy2, *zy, MPFR_RNDN);
440         mpfr_add(zmag2, zx2, zy2, MPFR_RNDN);
            while (n < maxiters && mpfr_cmp_d(zmag2, 65536) < 0 ) {
              // d <- 2 d z + 1
              mpfr_mul(dzx, *zx, *dx, MPFR_RNDN);
              mpfr_mul(t, *zy, *dy, MPFR_RNDN);
445           mpfr_sub(dzx, dzx, t, MPFR_RNDN);
              mpfr_mul(dzy, *zx, *dy, MPFR_RNDN);
              mpfr_mul(t, *zy, *dx, MPFR_RNDN);
              mpfr_add(dzy, dzy, t, MPFR_RNDN);
              mpfr_mul_2ui(dzx, dzx, 1, MPFR_RNDN);
450           mpfr_add_ui(*dx, dzx, 1, MPFR_RNDN);
              mpfr_mul_2ui(*dy, dzy, 1, MPFR_RNDN);
              // z <- z^2 + c
              mpfr_mul(zxy, *zx, *zy, MPFR_RNDN);
              mpfr_sub(*zx, zx2, zy2, MPFR_RNDN);
455           mpfr_add(*zx, *zx, *cx, MPFR_RNDN);
              mpfr_mul_2ui(*zy, zxy, 1, MPFR_RNDN);
              mpfr_add(*zy, *zy, *cy, MPFR_RNDN);
              // |z|^2
              mpfr_sqr(zx2, *zx, MPFR_RNDN);
460           mpfr_sqr(zy2, *zy, MPFR_RNDN);
              mpfr_add(zmag2, zx2, zy2, MPFR_RNDN);
              // count
              ++n;
            }
465         img->p[j][i].n = n;
            if (!(mpfr_cmp_d(zmag2, 65536) < 0)) {
              // escaped
              mpfr_sqr(*dx, *dx, MPFR_RNDN);
              mpfr_sqr(*dy, *dy, MPFR_RNDN);
470           mpfr_add(*dx, *dx, *dy, MPFR_RNDN);
              mpfr_div(*dx, zmag2, *dx, MPFR_RNDN);
              mpfr_sqrt(*dx, *dx, MPFR_RNDN);
              mpfr_log(zmag2, zmag2, MPFR_RNDN);
              mpfr_mul(zmag2, zmag2, *dx, MPFR_RNDN);
475           mpfr_div(zmag2, zmag2, cz0, MPFR_RNDN);
```

```
                double distance = mpfr_get_d(zmag2, MPFR_RNDN);
                img->p[j][i].colour = (distance < 0.5 * sqrt(2)) ? black : white;
                img->p[j][i].state = done;
                for (int dj = -1; dj <= 1; ++dj) {
480               int jj = j + dj;
                  if (jj < 0 || SIZE <= jj) continue;
                  for (int di = -1; di <= 1; ++di) {
                    int ii = i + di;
                    if (ii < 0 || SIZE*2 <= ii) continue;
485                 if (img->p[jj][ii].state == queued) {
                      img->p[jj][ii].state = active;
                    }
                  }
                }
490             ++progress;
              }
            }
          }
        }
495   }
      maxiters *= 2;
      progressed = progressed || progress;
    }
    progressReport('\n');
500 mpfr_clears(cx0, cy0, cz0, zx2, zy2, zxy, zmag2, dzx, dzy, t, (mpfr_ptr) 0);
    const unsigned char blocks[16][4] =
      { { 0x20, 0x00, 0x00, 0 } // empty
      , { 0xe2, 0x96, 0x98, 0 } // top left
      , { 0xe2, 0x96, 0x9d, 0 } // top right
505   , { 0xe2, 0x96, 0x80, 0 } // top
      , { 0xe2, 0x96, 0x96, 0 } // bottom left
      , { 0xe2, 0x96, 0x8c, 0 } // left
      , { 0xe2, 0x96, 0x9e, 0 } // bottom left top right
      , { 0xe2, 0x96, 0x9b, 0 } // not bottom right
510   , { 0xe2, 0x96, 0x97, 0 } // bottom right
      , { 0xe2, 0x96, 0x9a, 0 } // top left bottom right
      , { 0xe2, 0x96, 0x90, 0 } // right
      , { 0xe2, 0x96, 0x9c, 0 } // not bottom left
      , { 0xe2, 0x96, 0x84, 0 } // bottom
515   , { 0xe2, 0x96, 0x99, 0 } // not top right
      , { 0xe2, 0x96, 0x9f, 0 } // not top left
      , { 0xe2, 0x96, 0x88, 0 } // full
      };
    for (int j = SIZE - 1; j > 0; j -= 2) {
520   for (int i = 0; i < SIZE*2 - 1 && i < 360; i += 2) {
        int b =  (img->p[j][i].colour == black)
              | ((img->p[j][i+1].colour == black) << 1)
              | ((img->p[j-1][i].colour == black) << 2)
              | ((img->p[j-1][i+1].colour == black) << 3);
525     fprintf(stderr, "%s", blocks[b]);
      }
      fputc('\n', stderr);
    }
    fflush(stderr);
530 }

int locate(mpfr_t x, mpfr_t y, mpfr_t z, int *period) {
```

```
  progressReport('L');
  mpfr_prec_t bits = max(mpfr_get_prec(x), mpfr_get_prec(y)) + 8;
  mpfr_prec_round(x, bits, MPFR_RNDN);
  mpfr_prec_round(y, bits, MPFR_RNDN);
  mpfr_t cz0, zx[4], zy[4], cx[4], cy[4], dx, dy, t;
  mpfr_inits2(bits, cz0, zx[0], zx[1], zx[2], zx[3], zy[0], zy[1], zy[2], zy[3],↙
      ↳ cx[0], cx[1], cx[2], cx[3], cy[0], cy[1], cy[2], cy[3], dx, dy, t, (↙
      ↳ mpfr_ptr) 0);
  mpfr_set(cz0, z, MPFR_RNDN);
  mpfr_sub(cx[0], x, cz0, MPFR_RNDN);
  mpfr_sub(cy[0], y, cz0, MPFR_RNDN);
  mpfr_add(cx[1], x, cz0, MPFR_RNDN);
  mpfr_sub(cy[1], y, cz0, MPFR_RNDN);
  mpfr_add(cx[2], x, cz0, MPFR_RNDN);
  mpfr_add(cy[2], y, cz0, MPFR_RNDN);
  mpfr_sub(cx[3], x, cz0, MPFR_RNDN);
  mpfr_add(cy[3], y, cz0, MPFR_RNDN);
  mpfr_set_ui(zx[0], 0, MPFR_RNDN);
  mpfr_set_ui(zy[0], 0, MPFR_RNDN);
  mpfr_set_ui(zx[1], 0, MPFR_RNDN);
  mpfr_set_ui(zy[1], 0, MPFR_RNDN);
  mpfr_set_ui(zx[2], 0, MPFR_RNDN);
  mpfr_set_ui(zy[2], 0, MPFR_RNDN);
  mpfr_set_ui(zx[3], 0, MPFR_RNDN);
  mpfr_set_ui(zy[3], 0, MPFR_RNDN);
  int origin;
  int p0 = *period;
  int p = 0;
  do {
    ++p;
    for (int i = 0; i < 4; ++i) {
      mpfr_mul(t, zx[i], zy[i], MPFR_RNDN);
      mpfr_sqr(zx[i], zx[i], MPFR_RNDN);
      mpfr_sqr(zy[i], zy[i], MPFR_RNDN);
      mpfr_sub(zx[i], zx[i], zy[i], MPFR_RNDN);
      mpfr_add(zx[i], zx[i], cx[i], MPFR_RNDN);
      mpfr_mul_2ui(zy[i], t, 1, MPFR_RNDN);
      mpfr_add(zy[i], zy[i], cy[i], MPFR_RNDN);
    }
    int preal = 0;
    for (int i = 0; i < 4; ++i) {
      int j = (i + 1) % 4;
      if (!(mpfr_sgn(zy[i]) < 0 && mpfr_sgn(zy[j]) < 0) && !(mpfr_sgn(zy[i]) > 0↙
          ↳ && mpfr_sgn(zy[j]) > 0)) {
        mpfr_sub(dy, zy[j], zy[i], MPFR_RNDN);
        mpfr_sub(dx, zx[j], zx[i], MPFR_RNDN);
        int s = mpfr_sgn(dy);
        mpfr_mul(dy, dy, zx[i], MPFR_RNDN);
        mpfr_mul(dx, dx, zy[i], MPFR_RNDN);
        mpfr_sub(dy, dy, dx, MPFR_RNDN);
        if (mpfr_sgn(dy) == s && s != 0) {
          ++preal;
        }
      }
    }
    origin = preal & 1;
  } while ((!origin) && p < p0 * 12);
```

```
  if (origin) {
    *period = p;
  }
  mpfr_clears(cz0, zx[0], zx[1], zx[2], zx[3], zy[0], zy[1], zy[2], zy[3], cx[0], cx[1], cx[2], cx[3], cy[0], cy[1], cy[2], cy[3], (mpfr_ptr) 0);
  return origin;
}

int pick(struct image *img, mpfr_t x, mpfr_t y, mpfr_t z, int *period) {
  mpfr_prec_t bits = max(mpfr_get_prec(x), mpfr_get_prec(y)) + 8;
  mpfr_prec_round(x, bits, MPFR_RNDN);
  mpfr_prec_round(y, bits, MPFR_RNDN);
  mpfr_t cz0;
  mpfr_init2(cz0, mpfr_get_prec(z));
  mpfr_set(cz0, z, MPFR_RNDN);
  mpfr_div_2ui(cz0, cz0, LOGSIZE, MPFR_RNDN);
  mpfr_mul_ui(cz0, cz0, 6, MPFR_RNDN);
  int r = 0;
  int n = 0;
  for (int j = 0; j < SIZE; ++j) {
    for (int i = 0; i < SIZE*2; ++i) {
      if (img->p[j][i].colour == black) {
        ++n;
      }
    }
  }
  if (n > 0) {
    int p0 = *period;
    int p = 0;
    while (n > 0 && (p < p0 || (p0 > 1 && (p % p0) == 0) || p > p0 * 8)) {
      progressReport(',');
      int m = rand() % n + 1;
      for (int j = 0; j < SIZE; ++j) {
        for (int i = 0; i < SIZE*2; ++i) {
          if (img->p[j][i].colour == black) {
            --m;
          }
          if (m == 0) {
            --n;
            img->p[j][i].colour = white;
            mpfr_set(x, img->p[j][i].cx, MPFR_RNDN);
            mpfr_set(y, img->p[j][i].cy, MPFR_RNDN);
            break;
          }
        }
        if (m == 0) {
          break;
        }
      }
      if (m == 0) {
        p = p0;
        if (! locate(x, y, cz0, &p)) {
          p = 0;
          r = 0;
        } else {
          r = 1;
        }
```

```
      } else {
        puts("internal error in pick()\n");
        exit(1);
      }
    }
    *period = p;
  }
  mpfr_clear(cz0);
  return r;
}

void main2(mpfr_prec_t prec0, int period, mpfr_t x, mpfr_t y, mpfr_t z, struct ↙
    ↳ image *img) {
  if (muatom(period, x, y, z)) {
    int period2 = period;
    mpfr_t x2, y2, z2;
    mpfr_init2(x2, mpfr_get_prec(x)); mpfr_set(x2, x, MPFR_RNDN);
    mpfr_init2(y2, mpfr_get_prec(y)); mpfr_set(y2, y, MPFR_RNDN);
    mpfr_init2(z2, mpfr_get_prec(z)); mpfr_set(z2, z, MPFR_RNDN);
    int bits = mpfr_get_prec(x);
    mpfr_printf("%d %d %Re %Re %Re\n", bits, period, x, y, z);
    fflush(stdout);
    if (bits > prec0) exit(0);
    if (period == 1) {
      initialize(img, 24);
      render(img, x, y, z);
      while (pick(img, x, y, z, &period)) {
        main2(prec0, period, x, y, z, img);
        period = period2;
        mpfr_set_prec(x, mpfr_get_prec(x2)); mpfr_set(x, x2, MPFR_RNDN);
        mpfr_set_prec(y, mpfr_get_prec(y2)); mpfr_set(y, y2, MPFR_RNDN);
        mpfr_set_prec(z, mpfr_get_prec(z2)); mpfr_set(z, z2, MPFR_RNDN);
      }
      deinitialize(img);
    } else {
      mpfr_prec_round(x, mpfr_get_prec(x) * 2, MPFR_RNDN);
      mpfr_prec_round(y, mpfr_get_prec(y) * 2, MPFR_RNDN);
      for (int j = 0; j < 8 * SIZE; ++j) {
        double angle  = 6.283185307179586 * rand() / (double) RAND_MAX;
        double radius = (1.0 + rand() / (double) RAND_MAX) / 0.75;
        mpfr_t p;
        mpfr_init2(p, mpfr_get_prec(z2));
        mpfr_set_d(p, 0.995, MPFR_RNDN);
        mpfr_pow(x, z2, p, MPFR_RNDN);
        mpfr_pow(y, z2, p, MPFR_RNDN);
        mpfr_clear(p);
        mpfr_set(z, z2, MPFR_RNDN);
        mpfr_mul_d(x, x, radius * cos(angle), MPFR_RNDN);
        mpfr_mul_d(y, y, radius * sin(angle), MPFR_RNDN);
        mpfr_add(x, x, x2, MPFR_RNDN);
        mpfr_add(y, y, y2, MPFR_RNDN);
        mpfr_div_d(z, z, log2(period), MPFR_RNDN);
        int period0 = period;
        if (locate(x, y, z, &period) && (period % period0)) {
          main2(prec0, period, x, y, z, img);
        }
        period = period2;
```

```
          mpfr_set_prec(x, mpfr_get_prec(x2)); mpfr_set(x, x2, MPFR_RNDN);
          mpfr_set_prec(y, mpfr_get_prec(y2)); mpfr_set(y, y2, MPFR_RNDN);
          mpfr_set_prec(z, mpfr_get_prec(z2)); mpfr_set(z, z2, MPFR_RNDN);
        }
      }
      mpfr_clear(x2);
      mpfr_clear(y2);
      mpfr_clear(z2);
    }
  }

  int main(int argc, char **argv) {
    srand(time(0));
    if (argc != 1 && argc != 5) {
      fprintf(stderr, "usage: ./emndl_automu prec period cx cy\n");
      return 1;
    }
    mpfr_prec_t prec = 16;
    if (argc == 5) {
      prec = atoi(argv[1]);
    }
    int period = 1;
    mpfr_t x, y, z;
    mpfr_inits2(prec, x, y, z, (mpfr_ptr) 0);
    mpfr_set_ui(x, 0, MPFR_RNDN);
    mpfr_set_ui(y, 0, MPFR_RNDN);
    if (argc == 5) {
      period = atoi(argv[2]);
      mpfr_set_str(x, argv[3], 10, MPFR_RNDN);
      mpfr_set_str(y, argv[4], 10, MPFR_RNDN);
    }
    mpfr_set_prec(z, 24);
    struct image *img = calloc(1, sizeof(*img));
    main2(prec, period, x, y, z, img);
    mpfr_clears(x, y, z, (mpfr_ptr) 0);
    mpfr_free_cache();
    return 0;
  }
```

7 emndl-autotune.cabal

```
Name:                emndl-autotune
Version:             0.2.0.1
Synopsis:            automated tuning for Mandelbrot Set exploration
Description:         Run "emndl_autotune +RTS -N -RTS 192" for 192 bits
                     precision.  Defaults to 128 bits, using MPFR.
                     Best with terminal supporting Unicode block chars.
                     Note that hmpfr might require a GHC compiled with
                     integer-simple instead of the default integer-gmp.
Homepage:            http://code.mathr.co.uk/emndl
License:             GPL-3
License-file:        COPYING
Author:              Claude Heiland-Allen
Maintainer:          claude@mathr.co.uk
Category:            Graphics
Build-type:          Simple
Extra-source-files:  README THANKS
```

```
Cabal-version:       >=1.4

Executable emndl_autotune
  Main-is:             emndl_autotune.hs
  Other-modules:       Complex GridScan MuAtom Number Roots
  Build-depends:       base, array, parallel >= 3 && < 4, random, hmpfr >= 0.3.2, ↙
    ↳ Vec >= 0.9.8, ad >= 1.0.3 && < 4.4, reflection
  GHC-Options:         -O3 -Wall -threaded -rtsopts -fspec-constr-count=50
```

8 emndl_autotune.hs

```
{-
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
-}

module Main (main) where

import Control.Monad (guard, when)
import Data.Either (lefts)
import Data.List (minimumBy)
import Data.Maybe (listToMaybe, fromMaybe)
import Data.Ord (comparing)
import Data.Vec (NearZero(nearZero))
import System.Random (RandomGen, randomR, newStdGen) -- StdGen)
import System.Environment (getArgs)
import System.IO (hPutStrLn, stderr)
import Control.Parallel.Strategies (parMap, rseq)
import Data.Number.MPFR (toString, RoundMode(Near), set)
import Data.Number.MPFR.Instances.Near()

import Number (I, R, C)
import Complex (Complex((:+)), magnitude2)
import MuAtom (refineNucleus)
import GridScan (gridEdge, gridShow, gridConverge, gridStep, gridScan, gridSpace↙
    ↳ )

default (Int)

straddlesOrigin :: [C] -> Bool
straddlesOrigin ps = odd . length . filter id . zipWith positiveReal ps $ (drop ↙
    ↳ 1 ps ++ take 1 ps)
```

```
positiveReal :: C -> C -> Bool
positiveReal (u:+v) (x:+y)
  | v < 0 && y < 0 = False
  | v > 0 && y > 0 = False
  | (u * (y - v) - v * (x - u)) * (y - v) > 0 = True
  | otherwise = False

maxPeriod :: I -> I
maxPeriod p = 12 * p

locateNucleus :: I -> R -> C -> Maybe I
locateNucleus p r c =
  let cs = [ c + (r:+r), c + (r:+(-r)), c + ((-r):+(-r)), c + ((-r):+r) ]
      zs = iterate (zipWith (\cc z -> z * z + cc) cs) [0,0,0,0]
  in  fmap fst . listToMaybe . dropWhile (not . straddlesOrigin . snd) . zip [0 ..
      ↪ maxPeriod p] $ zs

offsetFromOrigin :: [C] -> R
offsetFromOrigin = magnitude2 . sum

rescan :: I -> R -> C -> Maybe (R, C)
rescan p r0 c0
  = case filter (straddlesOrigin . snd)
  . parMap rseq (fmap (parMap rseq (\c -> iterate (\z->z * z + c) 0 !! p)))
  . fmap (\(r, c) -> ((r, c), [c + (r:+r), c + ((-r):+r), c + ((-r):+(-r)), c +
      ↪ (r:+(-r))]))
  $( [ (r', c0 + d) | i <- [-1,1], j <- [-1,1 ], let d = ((r'*i) :+ (r'*j)) ]
  ++ [ (r', c0 + d) | i <- [  0 ], j <- [  0 ], let d = ((r'*i) :+ (r'*j)) ]) of
    [] -> Nothing
    xs -> Just . fst . minimumBy (comparing $ offsetFromOrigin . snd) $ xs
  where r' = r0 / 2

shuffle :: RandomGen g => [a] -> g -> ([a], g)
shuffle [] g = ([], g)
shuffle xs g =  let (i, g') = randomR (0, length xs - 1) g
                    (h, x:t) = splitAt i xs
                    (xs', g'') = shuffle (h ++ t) g'
                in  (x : xs', g'')

autotunes :: RandomGen g => ((C, R, I), g) -> [Either (C, R, I) String]
autotunes x@((c0, r0, p0), _) = Left (c0, r0, p0) :
  case autotune1 x of
    (Nothing, _) -> []
    (Just str, m) -> Right str : case m of
      Nothing -> []
      Just y -> autotunes y

autotune1 :: RandomGen g => ((C, R, I), g) -> (Maybe String, Maybe ((C, R, I), g
    ↪ ))
autotune1 (x@(c0, r0, _), g0)
  | nearZero r0 = (Nothing, Nothing)
  | otherwise =
      let grid = gridConverge . iterate gridStep . gridScan c0 $ r0
          edge = gridEdge grid
          str = gridShow grid
          (es, g1) = shuffle edge g0
      in  (Just str, autotune' es (x, g1))
```

```
autotune' :: RandomGen g => [C] -> ((C, R, I), g) -> Maybe ((C, R, I), g)
autotune' [] _ = Nothing
autotune' (c1:es) x@((_, r0, p0), g0) =
  fromMaybe (autotune' es x) $ do
    let r1 = gridSpace r0
    p1 <- locateNucleus p0 r1 c1
    guard $ p1 > 2 && (p0 == 1 || p1 `mod` p0 /= 0)
    (_, c1') <- (!! 64) . iterate (uncurry (rescan p1) =<<) $ Just (r1, c1)
    let (re, im, r2) = refineNucleus (fromIntegral p1) c1'
        c2 = re :+ im
    return $ Just ((c2, r2, p1), g0)

main :: IO ()
main = do
  args <- getArgs
  let bits = case args of
        [b] -> read b
        _ -> 128 :: Int
      dec = ceiling . ((logBase 10 2 :: Double) *) . fromIntegral $ bits
      r = set Near (fromIntegral bits)
--      g = read "123456789 987654321" :: StdGen
  g <- newStdGen
  let ats = autotunes ((r 0 :+ r 0, r 1, 1), g)
      (fre :+ fim, fzr, _) = last (lefts ats)
      pretty (Left (re:+im,zr,p)) = ["P " ++ show p, "R " ++ toString dec re, "I↙
          ↳ " ++ toString dec im, "@ " ++ toString dec zr]
      pretty (Right s) = [s]
  when verbose $ (mapM_ (hPutStrLn stderr) . concatMap pretty . init) ats
  putStrLn . unwords $ [ toString dec fre, toString dec fim, show (ceiling (0.5 ↙
      ↳ + logBase 2 (256 / fzr) / 8) :: Int) ]

verbose :: Bool
verbose = True
```

9 emndl_calculate.cc

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011,2012,2018  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

    Based in part on knighty's public domain nanomb1/nanomb2 code from:
    <https://fractalforums.org/f/28/t/277/msg8132#msg8132>
    <https://fractalforums.org/f/28/tooming/277/msg7983#msg7983
```

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <cmath>
#include <qd/dd_real.h>
#include <qd/qd_real.h>
#include <qd/fpu.h>
#include <list>
#include <pthread.h>

#include "mp_real.h"

#define NANOMB2

#define pi 3.141592653589793
#define CHANNELS 5

#ifdef NANOMB2
// m.cpp 2018 Knighty
// based on some code by Claude Heiland-Allen
// LICENSE: public domain, cc0, whatever
// acceleration method for rendering Mandelbrot set
// based on approximation of n iterations around minibrot of period n using bivariate polynomial
//-------------------------------------------------------------------------------------------------
// compile:
//  g++ -std=c++11 -Ofast -mfpmath=387 -march=native mb.cpp -lmpfr -lgmp -Wall -Wextra -pedantic -fopenmp
//-------------------------------------------------------------------------------------------------
// run:
//  ./a --help
// view results:
//  display out.ppm
//#undef RC_INVOKED
//#undef __STRICT_ANSI__
//#include <float.h>

//#include <cassert>
#include <cstdlib>
#include <cstring>

#include <limits>

#include <iostream>
#include <fstream>

#if 0
#include <complex>
#define COMPLEX complex
#else
```

```
    //A little problem with mpreal which includes <complex> so I had to change the ↙
        ↳ name of the custom complex class to fcomplex
75  #include "complex.h"
    #define COMPLEX fcomplex
    #endif

    #include <vector>
80
    #include "mp_real.h"

    using namespace std;

85  // type aliases
    typedef int N;
    typedef long double R_lo;
    typedef mp_real R_hi;
    typedef COMPLEX<R_lo> C_lo;
90  typedef COMPLEX<R_hi> C_hi;

    #define TABSIZ 128
    //-------------------------------------------------------------------------------------------
        ↳
    // Conversion routines
95  inline R_lo r_lo(const R_hi &z)
    {
      return (long double)(z);//MPFR_RNDN);//mpfr_get_ld(z.backend().data(), ↙
          ↳ MPFR_RNDN);
    }

100 inline R_lo r_lo(const char *s)
    {
      return strtold(s, NULL);
    }

105 inline C_lo c_lo(const C_hi &z)
    {
      return C_lo(r_lo(real(z)), r_lo(imag(z)));
    }

110 //-------------------------------------------------------------------------------------------
        ↳
    // simple RGB24 image
    class image
    {
    public:
115   N width;
      N height;
      vector<uint8_t> rgb;

      // construct empty image
120   image(N width, N height)
      : width(width)
      , height(height)
      , rgb(width * height * 3)
      { };
125
      // plot a point
```

```
  void plot(N x, N y, N r, N g, N b)
  {
    N k = (y * width + x) * 3;
    rgb[k++] = r;
    rgb[k++] = g;
    rgb[k++] = b;
  };

  // save to PPM format
  void save(const char *filename)
  {
    FILE *out = fopen(filename, "wb");
    fprintf(out, "P6\n%d %d\n255\n", width, height);
    fwrite(&rgb[0], width * height * 3, 1, out);
    fflush(out);
    fclose(out);
  }
};
//-------------------------------------------------------------------------------------------------
  ↪
class refClass{
        N m_n;
        vector<C_lo> m_v;
public:
        refClass(): m_n(0), m_v(0) {}
        void adde(C_lo c){ m_v.push_back(c); m_n++;}
        C_lo & operator[](N i){ return m_v[i % m_n];}
};

class perturbationClass{
        C_lo m_d0;
        C_lo m_d;
        N    m_n0;
        R_lo m_col;
        bool m_escaped;
public:
        perturbationClass(C_lo d0, C_lo d, N n0): m_d0(d0), m_d(d), m_n0(n0), ↙
            ↪ m_col(0), m_escaped(false) {}
        void run(refClass &ref, N maxiter){
                for(N i=m_n0; i < maxiter; i++){
                        C_lo zn(ref[i]);
                        m_d = m_d * (R_lo(2) * zn + m_d) + m_d0;
                        R_lo r(norm(zn + m_d));
                        if (r > R_lo(256)){
                                m_escaped = true;
                                m_col = R_lo(i+1) - std::log(std::log(r))/std::↙
                                    ↪ log(R_lo(2));
                                return;
                        }
                }
        }
        R_lo getCol(){return m_col;}
        bool haveEscaped(){return m_escaped;}
};


class perturbationClassD{
```

```
        C_lo m_d0;
        C_lo m_d;
        C_lo m_dd;
        N    m_si;
        N    m_n0;
        R_lo m_col;
        bool m_escaped;
        int m_iters;
        float m_trans;
public:

        perturbationClassD(C_lo d0, C_lo d, C_lo dd, N si, N n0): m_d0(d0), m_d(d), m_dd(dd), m_si(si), m_n0(n0), m_col(0), m_escaped(false) {}

        //Experimental: Usually, after series approximation delta_0 (m_d0 here) have no effect.
        //              Here we do a first loop where we add delta_0 then, when delta_0 is too small,
        //             a second loop is performed where delta_0 is not used anymore. (idea from Pauldelbrot)
        // It works! ToDo: ...
        void run1(refClass &ref, N maxsi, N maxiter){
                N si = m_si, i = m_n0;
                for( ; si < maxsi && i < maxiter; si++, i++){
                        C_lo zn(ref[i]);
                        if(abs(m_d0.re)+abs(m_d0.im) < 1e-20 * (abs(zn.re)+abs(zn.im))) break;
                        m_dd = R_lo(2) * m_dd * (m_d + zn) + R_lo(1);
                        m_d = m_d * (R_lo(2) * zn + m_d) + m_d0;
                        R_lo r(norm(zn + m_d));
                        if (r > R_lo(1e10)){
                                m_escaped = true;
                                R_lo m = R_lo(i+1) - std::log(std::log(r))/std::log(R_lo(2));
                                m_iters = floor(m);
                                m_trans = m - m_iters;
                                m_col = R_lo(2) * std::sqrt(r) * std::log(r) / abs(m_dd);
                                return;
                        }
                }
                for( ; si < maxsi && i < maxiter; si++, i++){
                        C_lo zn(ref[i]);
                        m_dd = R_lo(2) * m_dd * (m_d + zn);// + R_lo(1);
                        m_d = m_d * (R_lo(2) * zn + m_d);// + m_d0;
                        R_lo r(norm(zn + m_d));
                        if (r > R_lo(1e10)){
                                m_escaped = true;
                                R_lo m = R_lo(i+1) - std::log(std::log(r))/std::log(R_lo(2));
                                m_iters = floor(m);
                                m_trans = m - m_iters;
                                m_col = R_lo(2) * std::sqrt(r) * std::log(r) / abs(m_dd);
                                return;
                        }
                }
```

```
        }
        void run(refClass &ref, N maxsi, N maxiter){
                for(N si = m_si, i = m_n0 ; si < maxsi && i < maxiter; si++, i++){
                        C_lo zn(ref[i]);
                        m_dd = R_lo(2) * m_dd * (m_d + zn) + R_lo(1);
                        m_d = m_d * (R_lo(2) * zn + m_d) + m_d0;
                        R_lo r(norm(zn + m_d));
                        if (r > R_lo(1e10)){
                                m_escaped = true;
                                R_lo m = R_lo(i+1) - std::log(std::log(r))/std::log(R_lo(2));
                                m_iters = floor(m);
                                m_trans = m - m_iters;
                                m_col = R_lo(2) * std::sqrt(r) * std::log(r) / abs(m_dd);
                                return;
                        }
                }
        }
        R_lo getCol(){return m_col;}
        int getIters(){return m_iters;}
        float getTrans(){return m_trans;}
        bool haveEscaped(){return m_escaped;}
};

//-----------------------------------------------------------------------
// Class for probes used to determine when to stop SA computations
// Grabbed from superMB. For future use
//-----------------------------------------------------------------------
class probeClass
{
    C_lo m_dzz;
public:
    C_lo m_dc;// delta c
    C_lo m_dz;// current delta z

        //Constructors
    probeClass(){}
    probeClass(C_lo dc, C_lo dz):m_dc(dc),m_dz(dz){ m_dzz = C_lo(0);}
    probeClass(const probeClass & pC){ m_dc = pC.m_dc; m_dz = pC.m_dz; m_dzz = pC.m_dzz;}

        //Do one iteration. the iterate is stored in the private member variabe m_dzz. One must call apply()
        //member function to store the new iterate in m_dz.
        void advance(const C_lo &rz){
        m_dzz = m_dz * (R_lo(2) * rz + m_dz) + m_dc;
    }

        //In case we need to rebase the SA to a new point.
    void shift(const C_lo &s, const C_lo &sz){
        m_dc -= s;
        m_dz -= sz;
        m_dzz = m_dz;//necessary to avoid the "floral fantasy" glitch. Also because areProbesOk uses dif2ex() below which compares against d_zz.
```

```
    }

        //
    void apply(){m_dz = m_dzz;}

        //Just computes the difference between m_dzz and the given point.
        C_lo dif2ex(const C_lo &ex){ return m_dzz - ex;}
};

//---------------------------------------------------------------------
// Stores informations about the roots that are found during SA.
// For future possible use.
//---------------------------------------------------------------------
class rootInfoClass
{
public:
    N n; //At which iteration this root was found
    N mult; //Multiplicity of the root... for later use
    C_lo RPos; //position of the root w.r.t. current reference point
    rootInfoClass(){}
    rootInfoClass(N _n, N _mult, const C_lo &_RPos):
        n(_n), mult(_mult), RPos(_RPos) {}
};

//class biPolyClass;//Forward declaration

//-------------------------------------------------------------------------------------------------
//Polynomial class
//-------------------------------------------------------------------------------------------------
class polyClass{
        N m_m; //highest exponent
        C_lo tab[TABSIZ];//to store the coefficients. It is more than big enought.
public:
        //Constructor
        polyClass(N m): m_m(m) {
                for(N l=0; l <= m_m; l++)
                        tab[l] = C_lo(0);
        }
        //Copy constructor
        polyClass(const polyClass& p): m_m(p.m_m) {
                for(N l=0; l <= m_m; l++)
                        tab[l] = p.tab[l];
        }

        //evaluation function. It would be nice to add an ()-operator.
        C_lo eval(C_lo u){
                C_lo r(0);
                C_lo ui(1);
                for(N i=0; i <= m_m; i++){
                        r += tab[i] * ui;
                        ui *= u;
                }
                return r;
        }
```

```
        //evaluat derivative.
        C_lo evalD(C_lo u){
                C_lo r(0);
                C_lo ui(1);
                for(N i=1; i <= m_m; i++){
                        r += R_lo(i) * tab[i] * ui;
                        ui *= u;
                }
                return r;
        }

        //Gives the nearest root to the 0. To use if and when applicable (that is the reference is near 0... atom domain thing!)
        //Newton should do the job (otherwise IA-Newton ?).
        C_lo getRoot(){
                C_lo rt(0);
                //R_lo t = abs(eval(rt));
                for(N i=0; i<30; i++){
                        C_lo num = eval(rt);
                        C_lo den = evalD(rt);
                        C_lo delta = num / den;
                        num = rt;
                        rt -= delta;
                        if( rt.re == num.re && rt.im == num.im) break;
                }
                return rt;
        }

        //Shift the polynomial to the given (relative) origin point
        //given a polynomial p(z) this function returns the polynomial q(z)==p(z+shift)
        polyClass shift(C_lo shift){
                polyClass res(m_m);
                R_lo bino[TABSIZ];//vector<R_lo> bino(m_m + 2);//bino could be precomuted.
                for(int i=0;i<=m_m;i++)
                        bino[i]=1;
                for(int i=0; i<=m_m; i++){
                        C_lo v(0);
                        for(int j=m_m-i; j>=0; j--)
                                v = tab[j+i] * bino[j] + shift * v;
                        for(int j=1; j<=m_m; j++)
                                bino[j] += bino[j-1];
                        res.tab[i] = v;
                }
                return res;
        }

        friend class biPolyClass;
};

//-----------------------------------------------------------------------------------------------
//Bivariate polynomial class. Used for SSA.
//-----------------------------------------------------------------------------------------------
```

```
385 class biPolyClass {
        N    m_m, m_n;
        C_lo tab[TABSIZ][TABSIZ];
        C_lo ttab[TABSIZ][TABSIZ];
        C_lo m_shift;
390     N    m_period;
        R_lo m_escRadius;

        //copy tab into ttab. used for the squaring.
        void mcopy(){
395             for(N l=0; l <= m_m; l++)
                    for(N c=0; c<= m_n; c++)
                        ttab[l][c] = tab[l][c];
        }

400     //for the squaring operation: ttab will be the square of tab
        C_lo csqrc(N k, N l){
            C_lo v(0);
            for(N i=0; i <= k; i++)
                for(N j=0; j <= l; j++)
405                 v += ttab[i][j] * ttab[k-i][l-j];
            return v;
        }
public:
        //Constructor
410     biPolyClass(N m, N n): m_m(m), m_n(n), m_shift(0), m_period(0), ↙
            ↳ m_escRadius(0) {
            for(N l=0; l <= m_m; l++)
                for(N c=0; c<= m_n; c++)
                    tab[l][c] = C_lo(0);
            tab[1][0] = C_lo(1);
415     }

        C_lo getShift(){ return m_shift;}
        N    getPeriod(){ return m_period;}
        void setPeriod(N period){ m_period = period;}
420     void setEscRadius(R_lo multiplier = R_lo(.1)){
            m_escRadius = multiplier * getRadius();
        }
        R_lo getEscRadius(){ return m_escRadius;}

425     //squaring
        void sqr(){
            mcopy();
            for(N i=0; i <= m_m; i++)
                for(N j=0; j <= m_n; j++)
430                 tab[i][j] = csqrc(i,j);
        }

        //Apply one Mandelbrot iteration
        void cstep(C_lo z){
435         sqr();
            tab[0][0]  = z;
            tab[0][1] += C_lo(1);
        }

440     //Evaluate the SSA polynomial
```

```
        C_lo eval(C_lo u, C_lo v){
                //v -= m_shift;//Take into account the shift
                C_lo r(0);
                C_lo ui(1);
                for(N i=0; i <= m_m; i+=2){
                        C_lo vj(ui);
                        for(N j=0; j <= m_n; j++){
                                r += tab[i][j] * vj;
                                vj *= v;
                        }
                        ui *= u*u;
                }
                return r;
        }

        //Evaluate the derivative wrt c
        C_lo eval_dc(C_lo u, C_lo v){
                //v -= m_shift;
                C_lo r(0);
                C_lo ui(1);
                for(N i=0; i <= m_m; i+=2){
                        C_lo vj(ui);
                        for(N j=1; j <= m_n; j++){
                                r += R_lo(j) * tab[i][j] * vj;
                                vj *= v;
                        }
                        ui *= u*u;
                }
                return r;
        }

        //Evaluate the derivative wrt z
        C_lo eval_dz(C_lo u, C_lo v){
                //v -= m_shift;
                C_lo r(0);
                C_lo ui(u);
                for(N i=2; i <= m_m; i+=2){
                        C_lo vj(C_lo(i) * ui);
                        for(N j=0; j <= m_n; j++){
                                r += tab[i][j] * vj;
                                vj *= v;
                        }
                        ui *= u*u;
                }
                return r;
        }

        //To see the coefficients
        void print(){
                for(N i=0; i <= m_m; i++){
                        for(N j=0; j <= m_n; j++){
                                cout << "i: " << i << "\tj: " << j << "\tval: " ↙
                                    ↳ << tab[i][j] << endl;
                        }
                        cout << "-----------------------" << endl;
                }
        }
```

```
        //gives the escape radius for this SSA.
        //based on polynomial roots properties: Rouché theorem.
        //ToDo: Document the method by which we find the radius.
        R_lo getRadius(){
                if(abs(tab[0][2])==0) return 1e30;
                return abs(tab[0][1])/abs(tab[0][2]);//Is actually accurate enough
                /*R_lo r(0);
                for(int i = 0; i < 10; i++){
                        C_lo den(0);
                        R_lo rr(1);
                        for(int j = 2; j <=m_n; j++){
                                den += tab[0][j] * rr;
                                rr  *= r;
                        }
                        r = abs(tab[0][1])/abs(den);
                }
                return r;*/
        }
        //version that gives an estimate of the bail out radius in dynamic plane depending on v.
        R_lo getRadius(C_lo v){
                C_lo t2(0), t4(0), vi(1);
                for(N i = 0; i <= m_n; i++){
                        t2 += tab[2][i] * vi;
                        t4 += tab[4][i] * vi;
                        vi *= v;
                }
                return abs(t2)/abs(R_lo(2)*t4);//Is actually accurate enough...
        }

        //Just give the part that depends on v only. It can be used as the classical series approximation.
        polyClass getSAPoly(){
                polyClass SApol(m_n);
                for(N i=0; i<=m_n; i++)
                        SApol.tab[i] = tab[0][i];
                return SApol;
        }

        //Partially evaluate the biPoly for the given v value.
        polyClass getEvalPoly(C_lo v){
                polyClass SApol(m_m);
                //v -= m_shift;
                for(N i=0; i<=m_m; i++){
                        C_lo vj(1);
                        C_lo r(0);
                        for(N j=0; j <= m_n; j++){
                                r  += tab[i][j] * vj;
                                vj *= v;
                        }
                        SApol.tab[i] = r;
                }
                return SApol;
        }
```

```
        //Partially evaluate the biPoly for the given v value. This is for ↙
            ↳ derivative evaluation.
        polyClass getEvalPoly_dz(C_lo v){
                polyClass SApol = getEvalPoly(v);
                //v -= m_shift;
                for(N i=0; i<m_m; i++){
                        SApol.tab[i] = SApol.tab[i+1] * R_lo(i+1);
                }
                SApol.m_m -= 1;
                return SApol;
        }

        //Partially evaluate the biPoly for the given v value. This is for ↙
            ↳ derivative evaluation.
        //To implement
        polyClass getEvalPoly_dc(C_lo v){
                polyClass SApol(m_m);
                //v -= m_shift;
                for(N i=0; i<=m_m; i++){
                        C_lo vi(1);
                        C_lo r(0);
                        for(N j=1; j<=m_n; j++){
                                r  += R_lo(j) * vi * tab[i][j];
                                vi *= v;
                        }
                        SApol.tab[i] = r;
                }
                return SApol;
        }

        //gives a shifted biPoly.We shift only wrt v param therefore we will use↙
            ↳  the shift function of polyClass.
        biPolyClass getShiftedBP(C_lo sv){
                biPolyClass SBP(m_m,m_n);
                for(N i=0; i<=m_m; i++){
                        polyClass pol(m_n);
                        for(N j=0; j<=m_n; j++)
                                pol.tab[j] = tab[i][j];
                        polyClass popol = pol.shift(sv);
                        for(N j=0; j<=m_n; j++)
                                SBP.tab[i][j] = popol.tab[j];
                }
                SBP.m_shift = m_shift + sv;//ToDo: verify if itis necessary to ↙
                    ↳ do add m_shift . This function is meant to be called only ↙
                    ↳ for the main SSA where shift==0. Just in case...
                return SBP;
        }
};

//-------------------------------------------------------------------------------------------
    ↳
//Iterate point till escape then return a "color" (not pretty at all)
//ToDo: use the closure of the biPoly (fp) for the current d0 because it is ↙
    ↳ constant
//Nota 1: It is interesting that Bout is approximately the square root of the "↙
    ↳ size" of the minibrot
//Nota 2: It is also interesting that the bail out ardius around 0 -in dynamic ↙
```

```
    ↪ plane- is the square root of the bailout around the minibrot -in the ↙
    ↪ parameter space-
//Nota 3: In reality Bout depends on d0 (the v parameter (in biPolyClass) ↙
    ↪ corresponding to c. we have d0==v and [C]=c+v )
//        One could recompute Bout for each pixel.
//------------------------------------------------------------------------------------------
    ↪
R_lo iteratePt( C_lo d0, biPolyClass &fp, refClass &ref, N period, N maxiter, ↙
    ↪ R_lo Bout, N &si ){
        C_lo d(0);
        C_lo ld(0);
        N i(0);
        si = N(0);
        if(abs(d0)<Bout){
                while(i<maxiter && norm(d)<Bout){//notice that we are comparing ↙
                    ↪ d^2 with Bout: norm(d)<Bout <=> d<sqrt(Bout)
                        d = fp.eval(d,d0);
                        i+=period;
                        si++;
                }
        }
        if(i>maxiter)
                return R_lo(-1);
        perturbationClass p(d0,d,i);
        p.run(ref,maxiter);
        if(p.haveEscaped())
                return p.getCol();
        else
            return R_lo(-1);
}

//------------------------------------------------------------------------------------------
    ↪
//Iterate point till escape then return the distance estimate
//ToDo: use the closure of the biPoly (fp) for the current d0 because it is ↙
    ↪ constant
//------------------------------------------------------------------------------------------
    ↪
R_lo iteratePtDE( C_lo d0, biPolyClass &fp, refClass &ref, N period, N maxsi, N ↙
    ↪ maxiter, R_lo Bout, N &si ){
        C_lo d(0);
        C_lo dd(0);
        N i(0);
        si = N(0);
        C_lo od0(d0);//Save original delta_c
        d0 -= fp.getShift();
        if(abs(d0)<Bout){
                while(i<maxiter && norm(d)<Bout){//notice that we are comparing ↙
                    ↪ d^2 with Bout: norm(d)<Bout <=> d<sqrt(Bout)
                        dd = dd * fp.eval_dz(d,d0) + fp.eval_dc(d,d0);
                        d = fp.eval(d,d0);
                        i+=period;
                        si++;
                }
        } //else dd = C_lo(1);
        if(i>maxiter)
                return R_lo(-1);
```

```
        //restore original delta_c
        d0 = od0;
        //Compute current d wrt current ref
        d = d - ref[i];
        //Use perturbation
        perturbationClassD p(d0,d,dd,si,i);
        p.run(ref,maxsi,maxiter);
        if(p.haveEscaped())
                return p.getCol();
        else
            return R_lo(-1);
}

//-------------------------------------------------------------------------------------------------
// iterate using the given list of SSAs.
// d0 is relative to the center of the screen. The SSAs transparently transform the point into their own reference frame when called thanks to their m_shift member variable.
//-------------------------------------------------------------------------------------------------
R_lo iteratePtDE1( C_lo d0, vector<biPolyClass> &SSAs, refClass &ref, N maxsi, N maxiter, N &si /*to be removed*/, int &iters, float &trans){
        //try the SSAs from the latest (deepest) to the first. if not possible use perturbation until we get near to 0 again then repeat.
        //ToDo: edit refClass to add another run(inside radius) function that verify if the iterated point returns near 0
        //ToDo2: we need to test for escapes AFTER the iteration --> edit biPolyClass to compute the "other" bailout.

        C_lo d(0);
        C_lo dd(0);
        N i(0);
        si = N(0);
        N nbrSSAs(SSAs.size());
        N curSSA(-1);

        //first iteration with SSA. It uses bailout radius in parameter space.
        for(N k = nbrSSAs - 1; k>=0; k--){
                C_lo d00 = d0 - SSAs[k].getShift();//delta_c coordinates have de be wrt current SSA
                if(abs(d00)<SSAs[k].getEscRadius()){
                        //----cout << "curSSA = " << curSSA << endl;
                        dd = dd * SSAs[k].eval_dz(d,d00) + SSAs[k].eval_dc(d,d00);
                        d = SSAs[k].eval(d,d00);
                        i+= SSAs[k].getPeriod();
                        si++;
                        curSSA = k;
                        break;
                }
        }
        N firstSSA = curSSA;
        R_lo d_norm(1.0/0.0);
        //Use SSAs from lower to upper to iterate
        for(; curSSA>=0 && SSAs[curSSA].getPeriod() >= 1; curSSA--){
                C_lo d00 = d0 - SSAs[curSSA].getShift();//delta_c coordinates
```

```
                    ↪ have de be wrt current SSA
                while(si < maxsi && i<maxiter && norm(d)<SSAs[curSSA].↙
                    ↪ getEscRadius()) {// norm(d)<SSAs[curSSA].getEscRadius() ↙
                    ↪ <=> abs(d)<sqrt(SSAs[curSSA].getEscRadius())
                      if (curSSA == firstSSA && norm(d) < d_norm)
695                   {
                        d_norm = norm(d);
                        // check interior: newton iterations for w0 = f^(si*p)↙
                            ↪ (w0, d0)
                        C_lo w0(d);
                        C_lo dw;
700                     bool converged = false;//------------------
                        for (int step = 0; step < 16; ++step)
                        {
                          C_lo w(w0);
                          dw = C_lo(1);
705                       for (int n = 0; n < si; ++n)
                          {
                            dw = dw * SSAs[curSSA].eval_dz(w, d00);
                            w = SSAs[curSSA].eval(w, d00);
                          }
710                       C_lo w0_next = w0 - (w - w0) / (dw - C_lo(1));
                          R_lo delta = norm(w0_next - w0);
                          //C_lo delta = (w0_next - w0);//-----------------
                          w0 = w0_next;
                          R_lo epsilon2(0); // FIXME
715                       if (delta <= epsilon2) { converged = true;break; }; ↙
                              ↪ // converged
                          //if (abs(delta.re) < abs(w0.re) * R_lo(1e-6) && abs↙
                              ↪ (delta.im) < abs(w0.im) * R_lo(1e-6)) { ↙
                              ↪ converged = true; break↙
                              ↪ ;}//-----------------------
                        }
                        if (converged && norm(dw) < R_lo(1))
                        {
720                       iters = maxiter;
                          trans = 0;
                          return 0;
    #if 0
                          // is interior, calculate interior DE
725                       C_lo z(w0), dz(1), dc(0), dzdz(0), dcdz(0);
                          for (int n = 0; n < si; ++n)
                            up.eval(z, dz, dc, dzdz, dcdz);
                          R_lo de = (R_lo(1) - norm(dz)) / abs(dcdz + (dzdz * ↙
                              ↪ dc) / (C_lo(1) - dz));
                          iters = si * period;
730                       trans = 0.0f;
                          return -de;
    #endif
                        }
                      }
735                   dd = dd * SSAs[curSSA].eval_dz(d,d00) + SSAs[curSSA].↙
                          ↪ eval_dc(d,d00);
                      d  = SSAs[curSSA].eval(d,d00);
                      i += SSAs[curSSA].getPeriod();
                      si++;
                }
```

```
740                 if(si >= maxsi || i >= maxiter) break;
            }

            //We are out of SSAs or maxiter attained
            //R_lo f(1);
745         //if(i == 0) {dd = C_lo(1); f = R_lo(.5);}
            if(si >= maxsi || i>=maxiter)
            {
              iters = maxiter;
              trans = 0;
750           return R_lo(0);
            }

            //Use perturbation to finish the job.
            //ToInvestigate: Usually when arriving here (same thing with SA) delta_c↙
                ↳  (d0 here) is too small wrt d.
755         //              One can use perturbation without adding delta_c. Some ↙
                ↳ other optimizations may become possible. (from comment by ↙
                ↳ Pauldelbrot @fractalforums.com)
            d -= ref[i]; //put d in reference orbit's local frame
            perturbationClassD p(d0,d,dd,si,i);//We use original d0 because the ↙
                ↳ reference orbit was computed there.
            p.run(ref,maxsi,maxiter);

760         if(p.haveEscaped())
            {
              iters = p.getIters();
              trans = p.getTrans();
              return p.getCol();
765         }
            else
            {
              iters = maxiter;
              trans = 0;
770           return R_lo(0);
            }
    }

    //-------------------------------------------------------------------------------------------------
        ↳
775 //
    //-------------------------------------------------------------------------------------------------
        ↳
    void plot(image &i,N x, N y, R_lo v, N si)
    {
      if(v<0){
780         i.plot(x, y, 0, 0, 0);
            return;
      }
      //v=std::sqrt(v);
      N r(N(255.*(.5*sin(0.1*v)+.5) ));
785   N g(N(255.*(.5*sin(0.09*v)+.5) ));
      N b(N(255.*(.5*sin(0.12*v)+.5) ));
      //N r((si & 1) * 255);
      si=si;//for when si is not used and we want the compiler to not say warning!
      i.plot(x, y, r, g, b);
790 }
```

```
void plotDE(image &i,N x, N y, R_lo v, R_lo pixelScale)
{
  if(v<0){
795         i.plot(x, y, 128, 0, 0);
            return;
  }
  v = std::sqrt(v * pixelScale);//std::sqrt(v);
  N r(N(255.*(.5*sin(0.1*v)+.5) ));
800   N g(N(255.*(.5*sin(0.09*v)+.5) ));
  N b(N(255.*(.5*sin(0.12*v)+.5) ));
  i.plot(x, y, r, g, b);
}
//-------------------------------------------------------------------------------------------------
    ↪
805 // entry point

static uint32_t burtle_hash(uint32_t a)
{
    a = (a+0x7ed55d16) + (a<<12);
810     a = (a^0xc761c23c) ^ (a>>19);
        a = (a+0x165667b1) + (a<<5);
        a = (a+0xd3a2646c) ^ (a<<9);
        a = (a+0xfd7046c5) + (a<<3);
        a = (a^0xb55a4f09) ^ (a>>16);
815     return a;
}

// uniform in [0,1)
static double jitter(uint32_t x, uint32_t y, uint32_t c)
820 {
  return burtle_hash(x + burtle_hash(y + burtle_hash(c))) / (double) (0↙
      ↪ x100000000LL);
}

void main1(const C_hi &c, N bm, N bn, N period, N maxsi, N maxiters, N width, N ↙
    ↪ height, float *buffer, R_lo r0)
825 {
  R_lo tmax = r0;

  //Compute bivariate polynomial approximating period iterations↙
      ↪ -------------------------
  //ToDo: compute also the bivariate polynomial for the "upper level" hyperbolic↙
      ↪  components centres (HCC):
830   //   - Use the running bivariate polynomial... actually the part that doesn't ↙
      ↪ depend on "u" (the SA part)
  //   - Find the upper HCCs by solving the zroes of the SA poly when applicable↙
      ↪ .
  //   - We will not suppose that "c" is at the center of a minibrot anymore.
  //   - The last (deepest) HC is the one which center is inside or near the ↙
      ↪ rendered area
  //    This procedure may fail when the SA part of tha bipoly is no longer ↙
      ↪ accurate while we haven't found
835   //  an HCC inside the rendered area and that area is too small to be handled ↙
      ↪ by the previous HCC
  //  (insufficient accuracy). Maybe a kind of "super" perturbation?
  //    For now we will suppose we have a good deepest HCC.
```

```
    //ToDo: what about the case where no deepest minibrot is found... We still ↙
        ↳ need good reference points... maybe subdivide the rendered area?
    biPolyClass fp(bm,bn);
840 vector<biPolyClass> SSAs;//to store the set of the relevent hyperbolic ↙
        ↳ components' SSAs from lowest to highest period.
    R_lo smallest_ref_rad_so_far(1000);//for atom domain
    refClass ref;
    C_hi z(0);
    C_lo zlo(0);
845 bool bNotFinished = true;
    bool stopRecSSAs = false;

    for(N i = 0; i < period && bNotFinished; i++){
        ref.adde(zlo);
850     z = z*z + c;
        zlo = c_lo(z);
        fp.cstep(zlo);
        //See if we have a new atom domain... Atom domain does work but not ver ↙
            ↳ well
        //R_lo eRad = fp.getRadius() * R_lo(1000.);//scale down in order to not ↙
            ↳ choose too far HC center. ToVerify... worse!
855     //doing "if(zmag < /*eRad*/ R_lo(.5)*smallest_ref_rad_so_far)" is better↙
            ↳  (with the .5 factor which removes a lot of SSAs)
        R_lo zmag = abs(zlo);

        if(zmag < /*eRad*/ R_lo(.5)*smallest_ref_rad_so_far /*&& i>7*/ && !↙
            ↳ stopRecSSAs){
            //compute the SSA for current hyperbolic component and store it ↙
                ↳ in SSAs:
860
            //get the SA
            polyClass SA = fp.getSAPoly();

            //compute the corresponding HC center
865         C_lo HCcentre = SA.getRoot();//HCcentre is relative to c.

            //get shifted SSA
            biPolyClass SSA = fp.getShiftedBP(HCcentre);
            SSA.setPeriod(i+1);
870         SSA.setEscRadius();

            if(SSA.getEscRadius() > R_lo(0.1) * tmax){
                //Store the shifted SSA into SSAs
                //We don't need HC with too low period. They may be ↙
                    ↳ slower than using perturbation.
875             if(i>7)//-----------------------------------------------
                    SSAs.push_back(SSA);

                //if HCcentre is inside or near the rendered area we ↙
                    ↳ should stop!
                //--> We will consider that HCcenter is near when it's ↙
                    ↳ magnitude is less than say 1000th the rendered ↙
                    ↳ area width? to study!!!
880             //if(abs(HCcentre) < R_lo(100) * tmax) bNotFinished = ↙
                    ↳ false;

                //Print infos:
```

```
                        cout << "Atom domain: period: " << i+1 << endl;
                        cout << "\t zmag:       " << zmag << endl;
                        cout << "\t HC centre: " << HCcentre << "\t (relative to location)" << endl;
                        cout << "\t Esc rad:   " << SSA.getEscRadius() << endl;
                        //fp.print();
                        cout << "-----------------------------------------------" << endl;
                    } else stopRecSSAs = true;
                    //
                    smallest_ref_rad_so_far = zmag;
                }
            }
            cerr << "__" << c_lo(z) << endl;
            fp.print();
            cerr << "R == " << fp.getRadius() << endl;

            //return 1;
            //-------------------------------------------------------------------------------------
            //Compute picture
            //-------------------------------------------------------------------------------------

            //N nSSA(SSAs.size()-2);
            //cout << "------------------------------\nCurrent nucleus: period: " << SSAs[nSSA].getPeriod() << endl;
            //SSAs[nSSA].print();
            N progress = 0;
            #pragma omp parallel for schedule(dynamic)
            for (N y = 0; y < height; ++y)
            {
              for (N x = 0; x < width; ++x)
              {
                R_lo t0 = (x + jitter(x, y, 0)) / width * 2 * pi;
                R_lo c0 = cosl(t0);
                R_lo s0 = sinl(t0);
                R_lo t1 = (x + jitter(x, y, 0) + 1) / width * 2 * pi;
                R_lo c1 = cosl(t1);
                R_lo s1 = sinl(t1);
                R_lo l0 = (y + jitter(x, y, 1)) / width * 2 * pi;
                R_lo r0 = expl(8 - l0);
                C_lo dc(r0 * c0, r0 * s0);
                C_lo dc1(r0 * c1, r0 * s1);
                R_lo pixel_spacing = sqrtl(norm(dc1 - dc));
                N si(0);
                int iters = 0;
                float trans = 0.0f;
                //we need to send the nucleus position wrt the center of the picture.
                R_lo v = iteratePtDE1( dc, SSAs, ref, maxsi, maxiters, si, iters, trans);//
                //R_lo v = iteratePtDE( dc, fp, ref, period, maxiters, Bout, si, iters, trans);

                float dwell = v <= 0 ? -1 : 16 + iters + trans;
                float distance = fabsl(v / pixel_spacing);
```

```
        float angle = 0;
        float isperiod = 0;
        float isnewton = 0;
935     float *result = buffer + size_t(y) * width * CHANNELS + x * CHANNELS;
        result[0] = dwell;
        result[1] = distance;
        result[2] = angle;
        result[3] = isperiod;
940     result[4] = isnewton;

        }
        #pragma omp critical
        cerr << "\timage scanline " << ++progress << " / " << height << "\r";
945   }
    }
    #endif

    #ifdef NANOMB1
950 // m.cpp 2018 Knighty
    // based on some code by Claude Heiland-Allen
    // LICENSE: public domain, cc0, whatever
    // acceleration method for rendering Mandelbrot set
    // based on approximation of n iterations around minibrot of period n using ↙
        ↪ bivariate polynomial
955 //-------------------------------------------------------------------------------------------
        ↪
    // Knighty (2018-07-10): Edited the code in order to make it possible to render ↙
        ↪ locations where the
    //                       center is not exactly on the nucleus:
    //                       - added the tmpPolyClass: used to find the position if ↙
        ↪ the nucleus.
    //                       - added code to compute the reference orbit wrt the ↙
        ↪ nucleus.
960 //                       - added iteratePtDE2() which is almost exactly the same↙
        ↪  as iteratePtDE() but
    //                         designed to take into account the new reference orbit↙
        ↪ .
    //-------------------------------------------------------------------------------------------
        ↪
    // compile:
    //  g++ -std=c++11 -Ofast -mfpmath=387 -march=native mb.cpp -lmpfr -lgmp -Wall -↙
        ↪ Wextra -pedantic -fopenmp
965 //-------------------------------------------------------------------------------------------
        ↪
    // run:
    //  ./a --help
    // view results:
    //  display out.ppm
970 //#undef RC_INVOKED
    //#undef __STRICT_ANSI__
    //#include <float.h>

    //#include <cassert>
975 #include <cstdlib>
    #include <cstring>

    #include <limits>
```

```
980 #include <iostream>
    #include <fstream>

    #if 0
    #include <complex>
985 #define COMPLEX complex
    #else
    //A little problem with mpreal which includes <complex> so I had to change the ↙
        ↳ name of the custom complex class to fcomplex
    #include "complex.h"
    #define COMPLEX fcomplex
990 #endif

    #include <vector>

    using namespace std;
995
    // type aliases
    typedef int N;
    typedef mp_real R_hi;
    typedef COMPLEX<R_hi> C_hi;
1000
    //-------------------------------------------------------------------------------------------------
        ↳
    // Conversion routines

    inline double r_lo(const R_hi &z, const double &dummy)
1005 {
      (void) dummy;
      return mpfr_get_ld(z.m, MPFR_RNDN);
    }

1010 inline long double r_lo(const R_hi &z, const long double &dummy)
    {
      (void) dummy;
      return mpfr_get_ld(z.m, MPFR_RNDN);
    }
1015
    template<typename R>
    inline R r_lo(const char *s, const R &dummy)
    {
      return r_lo(R_hi(s), dummy);
1020 }

    template <typename R>
    inline COMPLEX<R> c_lo(const C_hi &z, const R &dummy)
    {
1025   return COMPLEX<R>(r_lo(real(z), dummy), r_lo(imag(z), dummy));
    }

    //-------------------------------------------------------------------------------------------------
        ↳
    // simple RGB24 image
1030 class image
    {
    public:
```

```
    N width;
    N height;
1035  vector<uint8_t> rgb;

    // construct empty image
    image(N width, N height)
    : width(width)
1040  , height(height)
    , rgb(width * height * 3)
    { };

    // plot a point
1045  void plot(N x, N y, N r, N g, N b)
    {
      N k = (y * width + x) * 3;
      rgb[k++] = r;
      rgb[k++] = g;
1050  rgb[k++] = b;
    };

    // save to PPM format
    void save(const char *filename)
1055  {
      FILE *out = fopen(filename, "wb");
      fprintf(out, "P6\n%d %d\n255\n", width, height);
      fwrite(&rgb[0], width * height * 3, 1, out);
      fflush(out);
1060  fclose(out);
    }
};
//-------------------------------------------------------------------------------------------------
    ↳
template <typename R_lo>
1065 class refClass{
        typedef COMPLEX<R_lo> C_lo;
        N m_n;
        vector<C_lo> m_v;
public:
1070    refClass(): m_n(0), m_v(0) {}
        void adde(C_lo c){ m_v.push_back(c); m_n++;}
        const C_lo & operator[](N i) const { return m_v[i % m_n];}
};

1075 template <typename R_lo>
class perturbationClass{
        typedef COMPLEX<R_lo> C_lo;
        C_lo m_d0;
        C_lo m_d;
1080    N    m_n0;
        R_lo m_col;
        bool m_escaped;
public:
        perturbationClass(C_lo d0, C_lo d, N n0): m_d0(d0), m_d(d), m_n0(n0), ↙
            ↳ m_col(0), m_escaped(false) {}
1085    void run(const refClass<R_lo> &ref, N maxiter){
            for(N i=m_n0; i < maxiter; i++){
                C_lo zn(ref[i]);
```

```
                        m_d = m_d * (R_lo(2) * zn + m_d) + m_d0;
                        R_lo r(norm(zn + m_d));
                        if (r > R_lo(256)){
                                m_escaped = true;
                                m_col = R_lo(i+1) - R_lo(std::log(std::log(↙
                                    ↳ double(r)))/std::log(2.0));
                                return;
                        }
                }
        }
        R_lo getCol(){return m_col;}
        bool haveEscaped(){return m_escaped;}
};

template <typename R_lo>
class perturbationClassD{
        typedef COMPLEX<R_lo> C_lo;
        C_lo m_d0;
        C_lo m_d;
        C_lo m_dd;
        N    m_n0;
        R_lo m_col;
        bool m_escaped;
        int   m_iters;
        float m_trans;
public:
        perturbationClassD(C_lo d0, C_lo d, C_lo dd, N n0): m_d0(d0), m_d(d), ↙
            ↳ m_dd(dd), m_n0(n0), m_col(0), m_escaped(false) {}
        void run(const refClass<R_lo> &ref, N maxiter){
                for(N i=m_n0; i < maxiter; i++){
                        C_lo zn(ref[i]);
                        m_dd = R_lo(2) * m_dd * (m_d + zn) + R_lo(1);
                        m_d = m_d * (R_lo(2) * zn + m_d) + m_d0;
                        R_lo r(norm(zn + m_d));
                        if (r > R_lo(1e10)){
                                m_escaped = true;
                                double j = i+1 - log(log(double(r)))/log(2.0);
                                m_iters = floor(j);
                                m_trans = j - m_iters;
                                m_col = R_lo(2 * std::sqrt(double(r)) * std::log↙
                                    ↳ (double(r))) / abs(m_dd);
                                return;
                        }
                }
        }
        int   getIters(){return m_iters;}
        float getTrans(){return m_trans;}
        R_lo  getCol(){return m_col;}
        bool haveEscaped(){return m_escaped;}
};

template <typename R_lo> class uniPolyClass;
template <typename R_lo> class tmpPolyClass;
template <typename R_lo>
class biPolyClass {
        friend class uniPolyClass<R_lo>;
```

```
        friend class tmpPolyClass<R_lo>;
        typedef COMPLEX<R_lo> C_lo;
        N m_m, m_n;
        C_lo tab[128][128];
        C_lo ttab[128][128];
        void mcopy(){
                for(N l=0; l <= m_m; l++)
                        for(N c=0; c<= m_n; c++)
                                ttab[l][c] = tab[l][c];
        }
        C_lo csqrc(N k, N l){
                C_lo v(0);
                for(N i=0; i <= k; i++)
                        for(N j=0; j <= l; j++)
                                v += ttab[i][j] * ttab[k-i][l-j];
                return v;
        }
public:
        biPolyClass(N m, N n): m_m(m), m_n(n) {
                for(N l=0; l <= m_m; l++)
                        for(N c=0; c<= m_n; c++)
                                tab[l][c] = C_lo(0);
                tab[1][0] = C_lo(1);
        }

        void sqr(){
                mcopy();
                for(N i=0; i <= m_m; i++)
                        for(N j=0; j <= m_n; j++)
                                tab[i][j] = csqrc(i,j);
        }

        void cstep(C_lo z){
                sqr();
                tab[0][0]  = z;
                tab[0][1] += C_lo(1);
        }

        C_lo eval(C_lo u, C_lo v){
                C_lo r(0);
                C_lo ui(1);
                for(N i=0; i <= m_m; i+=2){
                        C_lo vj(ui);
                        for(N j=0; j <= m_n; j++){
                                r += tab[i][j] * vj;
                                vj *= v;
                        }
                        ui *= u*u;
                }
                return r;
        }

        C_lo eval_dc(C_lo u, C_lo v){
                C_lo r(0);
                C_lo ui(1);
                for(N i=0; i <= m_m; i+=2){
                        C_lo vj(ui);
```

```
                        for(N j=1; j <= m_n; j++){
                                r += R_lo(j) * tab[i][j] * vj;
                                vj *= v;
                        }
                        ui *= u*u;
                }
                return r;
        }

        C_lo eval_dz(C_lo u, C_lo v){
                C_lo r(0);
                C_lo ui(u);
                for(N i=2; i <= m_m; i+=2){
                        C_lo vj(C_lo(i) * ui);
                        for(N j=0; j <= m_n; j++){
                                r += tab[i][j] * vj;
                                vj *= v;
                        }
                        ui *= u*u;
                }
                return r;
        }

        void print(){
                for(N i=0; i <= m_m; i++){
                        for(N j=0; j <= m_n; j++){
                                cout << "i: " << i << "\tj: " << j << "\tval: " ↙
                                    ↳ << tab[i][j] << endl;
                        }
                        cout << "-----------------------" << endl;
                }
        }

        R_lo getRadius(){
                //return abs(tab[0][1])/abs(tab[0][2]);
                R_lo r(0);
                for(int i = 0; i < 10; i++){
                        C_lo den(0);
                        R_lo rr(1);
                        for(int j = 2; j <=m_n; j++){
                                den += tab[0][j] * rr;
                                rr  *= r;
                        }
                        r = abs(tab[0][1])/abs(den);
                }
                return R_lo(0.5) * r;
        }
};

//temporary poly class for solving for nucleus relative position. It is ↙
    ↳ initialized with the part of the SSA that depends only on c.
template <typename R_lo>
class tmpPolyClass {
        typedef COMPLEX<R_lo> C_lo;
        N m_m;
        C_lo b[128];
public:
```

```
        tmpPolyClass(const biPolyClass<R_lo> &p): m_m(p.m_n) {
                for(N i = 0; i <= m_m; i++)
                        b[i] = p.tab[0][i];
        }

        //evaluation function. It would be nice to add an ()-operator... or not. :)
        C_lo eval(C_lo u){
                C_lo r(0);
                C_lo ui(1);
                for(N i=0; i <= m_m; i++){
                        r += b[i] * ui;
                        ui *= u;
                }
                return r;
        }

        //evaluate derivative.
        C_lo evalD(C_lo u){
                C_lo r(0);
                C_lo ui(1);
                for(N i=1; i <= m_m; i++){
                        r += R_lo(i) * b[i] * ui;
                        ui *= u;
                }
                return r;
        }

        //Gives the nearest root to the 0. To use if and when applicable (that is the reference is near 0... atom domain thing!)
        //Newton should do the job (otherwise IA-Newton ?).
        C_lo getRoot(){
                C_lo rt(0);
                //R_lo t = abs(eval(rt));
                for(N i=0; i<30; i++){
                        C_lo num = eval(rt);
                        C_lo den = evalD(rt);
                        C_lo delta = num / den;
                        num = rt;
                        rt -= delta;
                        if( rt.re == num.re && rt.im == num.im) break;
                }
                return rt;
        }
};

// https://fractalforums.org/f/28/t/277/msg7952#msg7952
template <typename R_lo>
class uniPolyClass {
        typedef COMPLEX<R_lo> C_lo;
        N m_m;
        C_lo b[128];
        C_lo dbdc[128];
public:
        uniPolyClass(const biPolyClass<R_lo> &p, C_lo c): m_m(p.m_m) {
                for(N i = 0; i <= m_m; i += 2)
                {
```

```
                C_lo s(0), ds(0), cj(1), cj1(0);
1310            for (N j = 0; j <= p.m_n; ++j)
                {
                  s += p.tab[i][j] * cj;
                  ds += C_lo(j) * p.tab[i][j] * cj1;
                  cj *= c;
1315              cj1 *= c;
                  if (j == 0) cj1 = C_lo(1);
                }
                b[i] = s;
                dbdc[i] = ds;
1320          }
        }

        void eval(C_lo &z) const {
          C_lo zs(0), zi(1);
1325      for (N i = 0; i <= m_m; i += 2)
          {
            zs += b[i] * zi;
            zi *= z * z;
          }
1330      z = zs;
        }

        void eval(C_lo &z, C_lo &dc) const {
          C_lo zs(0), dcs(0), zi(1), zi1(0);
1335      for (N i = 0; i <= m_m; i += 2)
          {
            dcs += C_lo(i) * b[i] * zi1 * dc + dbdc[i] * zi;
            zs += b[i] * zi;
            zi *= z * z;
1340        zi1 *= z * z;
            if (i == 0) zi1 = z;
          }
          z = zs;
          dc = dcs;
1345    }

        void eval_dz(C_lo &z, C_lo &dz) const {
          C_lo zs(0), dzs(0), zi(1), zi1(0);
          for (N i = 0; i <= m_m; i += 2)
1350      {
            dzs += C_lo(i) * b[i] * zi1 * dz;
            zs += b[i] * zi;
            zi *= z * z;
            zi1 *= z * z;
1355        if (i == 0) zi1 = z;
          }
          z = zs;
          dz = dzs;
        }
1360
        void eval(C_lo &z, C_lo &dz, C_lo &dc, C_lo &dzdz, C_lo &dcdz) const {
          C_lo zs(0), dzs(0), dcs(0), dzdzs(0), dcdzs(0), zi(1), zi1(0), zi2(0);
          for (N i = 0; i <= m_m; i += 2)
          {
1365        dcdzs += C_lo(i) * C_lo(i - 1) * b[i] * zi2 * dz * dc + C_lo(i) * b[↙
```

```
                ↪ i] * zi1 * dcdz + C_lo(i) * dbdc[i] * zi1 * dz;
            dzdzs += C_lo(i) * C_lo(i - 1) * b[i] * zi2 * dz * dz + C_lo(i) * b[↙
                ↪ i] * zi1 * dzdz;
            dcs += C_lo(i) * b[i] * zi1 * dc + dbdc[i] * zi;
            dzs += C_lo(i) * b[i] * zi1 * dz;
            zs += b[i] * zi;
            zi *= z * z;
            zi1 *= z * z;
            zi2 *= z * z;
            if (i == 0) zi1 = z;
            if (i == 0) zi2 = C_lo(1);
          }
          z = zs;
          dz = dzs;
          dc = dcs;
          dzdz = dzdzs;
          dcdz = dcdzs;
        }
};


template <typename R_lo>
R_lo iteratePt( COMPLEX<R_lo> d0, biPolyClass<R_lo> &fp, refClass<R_lo> &ref, N ↙
    ↪ period, N maxiter, R_lo Bout, N &si ){
        typedef COMPLEX<R_lo> C_lo;
        C_lo d(0);
        C_lo ld(0);
        N i(0);
        si = N(0);
        if(abs(d0)<Bout){
                while(i<maxiter && norm(d)<Bout){
                        d = fp.eval(d,d0);
                        i+=period;
                        si++;
                }
        }
        if(i>maxiter)
                return R_lo(-1);
        perturbationClass<R_lo> p(d0,d,i);
        p.run(ref,maxiter);
        if(p.haveEscaped())
                return p.getCol();
        else
            return R_lo(-1);
}

template <typename R_lo>
R_lo iteratePtDE( const COMPLEX<R_lo> &d0, const biPolyClass<R_lo> &fp, const ↙
    ↪ refClass<R_lo> &ref, N period, N maxiter, R_lo Bout, N &si, int &iters, ↙
    ↪ float &trans){
        typedef COMPLEX<R_lo> C_lo;
        C_lo d(0);
        C_lo dd(0);
        N i(0);
        si = N(0);
        R_lo d_norm(1.0/0.0);
        if(abs(d0)<Bout){
```

```
        uniPolyClass<R_lo> up(fp, d0);
              while(i<maxiter && norm(d)<Bout){
                      up.eval(d, dd);
                      i+=period;
                      si++;
                      if (norm(d) < d_norm)
                      {
                        d_norm = norm(d);
                        // check interior: newton iterations for w0 = f^(si*p) (w0, d0)
                        C_lo w0(d);
                        C_lo dw;
                        for (int step = 0; step < 16; ++step)
                        {
                          C_lo w(w0);
                          dw = C_lo(1);
                          for (int n = 0; n < si; ++n)
                          {
                            up.eval_dz(w, dw);
                          }
                          C_lo w0_next = w0 - (w - w0) / (dw - C_lo(1));
                          R_lo delta = norm(w0_next - w0);
                          w0 = w0_next;
                          R_lo epsilon2(0); // FIXME
                          if (delta <= epsilon2) break; // converged
                        }
                        if (norm(dw) < R_lo(1))
                        {
                          // is interior, calculate interior DE
                          C_lo z(w0), dz(1), dc(0), dzdz(0), dcdz(0);
                          for (int n = 0; n < si; ++n)
                            up.eval(z, dz, dc, dzdz, dcdz);
                          R_lo de = (R_lo(1) - norm(dz)) / abs(dcdz + (dzdz * dc) / (C_lo(1) - dz));
                          iters = si * period;
                          trans = 0.0f;
                          return -de;
                        }
                      }
              }
      } //else dd = C_lo(1);//knighty: That was a mistake. We begin iterating at 0 so derivative doesn't have to be changed.
      if(i>=maxiter)
      {
        iters = maxiter;
        trans = 0.0f;
        return R_lo(0);
      }
      perturbationClassD<R_lo> p(d0,d,dd,i);
      p.run(ref,maxiter);
      if(p.haveEscaped())
      {
        iters = p.getIters();
        trans = p.getTrans();
        return p.getCol();
      }
      else
```

```
        {
          iters = maxiter;
          trans = 0.0f;
          return R_lo(0);
        }
}

template <typename R_lo>
R_lo iteratePtDE2( const COMPLEX<R_lo> &d0, const biPolyClass<R_lo> &fp, const ↙
    ↪ refClass<R_lo> &ref, COMPLEX<R_lo> nucleusPos, N period, N maxiter, R_lo ↙
    ↪ Bout, N &si, int &iters, float &trans){
      typedef COMPLEX<R_lo> C_lo;
      C_lo d(0);
      C_lo dd(0);
      N i(0);
      si = N(0);
      R_lo d_norm(1.0/0.0);
      if(abs(d0)<Bout){
        uniPolyClass<R_lo> up(fp, d0);
            while(i<maxiter && norm(d)<Bout){
                up.eval(d, dd);
                i+=period;
                si++;
                if (0)//norm(d) < d_norm)
                {
                  d_norm = norm(d);
                  // check interior: newton iterations for w0 = f^(si*p)↙
                      ↪ (w0, d0)
                  C_lo w0(d);
                  C_lo dw;
                  bool converged = false;//------------------
                  for (int step = 0; step < 16; ++step)
                  {
                    C_lo w(w0);
                    dw = C_lo(1);
                    for (int n = 0; n < si; ++n)
                    {
                      up.eval_dz(w, dw);
                    }
                    C_lo w0_next = w0 - (w - w0) / (dw - C_lo(1));
                    //R_lo delta = norm(w0_next - w0);
                        C_lo delta = (w0_next - w0);//-----------------
                    w0 = w0_next;
                    //R_lo epsilon2(0); // FIXME
                    //if (delta <= epsilon2) break; // converged
                        if (abs(delta.re) < abs(w0.re) * R_lo(1e-6) && ↙
                            ↪ abs(delta.im) < abs(w0.im) * R_lo(1e-6)) {↙
                            ↪  converged = true; break↙
                            ↪ ;}//-----------------------
                  }
                  if (converged && norm(dw) < R_lo(1))
                  {
                    // is interior, calculate interior DE
                    C_lo z(w0), dz(1), dc(0), dzdz(0), dcdz(0);
                    for (int n = 0; n < si; ++n)
                      up.eval(z, dz, dc, dzdz, dcdz);
                    R_lo de = (R_lo(1) - norm(dz)) / abs(dcdz + (dzdz * ↙
```

```
                                    ↪ dc) / (C_lo(1) - dz));
                                iters = si * period;
                                trans = 0.0f;
1525                            return -de;
                              }
                            }
                        }
          } //else dd = C_lo(1);//knighty: That was a mistake. We begin iterating ↙
              ↪ at 0 so derivative doesn't have to be changed.
1530      if(i>=maxiter)
          {
            iters = maxiter;
            trans = 0.0f;
            return R_lo(0);
1535      }
          //d0 and d have to be transformed into ref frame.
          C_lo d0_(d0 - nucleusPos);
          d  -= ref[i];
          perturbationClassD<R_lo> p(d0_,d,dd,i);
1540      p.run(ref,maxiter);
          if(p.haveEscaped())
          {
            iters = p.getIters();
            trans = p.getTrans();
1545        return p.getCol();
          }
          else
          {
            iters = maxiter;
1550        trans = 0.0f;
            return R_lo(0);
          }
    }

1555 void plot(image &i,N x, N y, double v, N si)
    {
      (void) si;
      if(v<0){
            i.plot(x, y, 0, 0, 0);
1560        return;
      }
      //v=std::sqrt(v);
      N r(N(255.*(.5*sin(0.1*v)+.5) ));
      N g(N(255.*(.5*sin(0.09*v)+.5) ));
1565  N b(N(255.*(.5*sin(0.12*v)+.5) ));
      //N r((si & 1) * 255);
      i.plot(x, y, r, g, b);
    }

1570 template <typename R_lo>
    void plotDE(image &i,N x, N y, R_lo u, R_lo pixelScale)
    {
      double v = 16 * log(abs(double(u * pixelScale))+1e-10);
      //if(u<0) v*=0.01;
1575  N r(N(255.*(.5*sin(0.1*v)+.5) ));
      N g(N(255.*(.5*sin(0.09*v)+.5) ));
      N b(N(255.*(.5*sin(0.12*v)+.5) ));
```

```
  i.plot(x, y, r, g, b);
}

static uint32_t burtle_hash(uint32_t a)
{
    a = (a+0x7ed55d16) + (a<<12);
    a = (a^0xc761c23c) ^ (a>>19);
    a = (a+0x165667b1) + (a<<5);
    a = (a+0xd3a2646c) ^ (a<<9);
    a = (a+0xfd7046c5) + (a<<3);
    a = (a^0xb55a4f09) ^ (a>>16);
    return a;
}

// uniform in [0,1)
static double jitter(uint32_t x, uint32_t y, uint32_t c)
{
  return burtle_hash(x + burtle_hash(y + burtle_hash(c))) / (double) (0↙
      ↳ x100000000LL);
}

struct kfb
{
  int w;
  int h;
  int n;
  int **iters;
  float **trans;
  float **de;
  kfb(int _w, int _h, int _n) : w(_w), h(_h), n(_n)
  {
    iters = new int  *[w];
    trans = new float*[w];
    de    = new float*[w];
    for (int x = 0; x < w; ++x)
    {
      iters[x] = new int  [h];
      trans[x] = new float[h];
      de   [x] = new float[h];
    }
  }
  ~kfb()
  {
    for (int x = 0; x < w; ++x)
    {
      delete[] iters[x];
      delete[] trans[x];
      delete[] de   [x];
    }
    delete[] iters;
    delete[] trans;
    delete[] de;
  }
  void plot(int x, int y, int _iters, float _trans, float _de)
  {
    iters[x][y] = _iters;
    trans[x][y] = _trans;
```

```
    de    [x][y] = _de;
  }
  void save(const char *filename)
  {
    FILE *f = fopen(filename, "wb");
    fwrite("KFB", 3, 1, f);
    fwrite(&w, sizeof(w), 1, f);
    fwrite(&h, sizeof(h), 1, f);
    for (int x = 0; x < w; ++x) fwrite(iters[x], sizeof(iters[x][0]) * h, 1, f);
    int iterdiv = 1;
    fwrite(&iterdiv, sizeof(iterdiv), 1, f);
    int colours = 2;
    fwrite(&colours, sizeof(colours), 1, f);
    unsigned char keys[] = { 0, 0, 0, 255, 255, 255 };
    fwrite(keys, sizeof(keys), 1, f);
    fwrite(&n, sizeof(n), 1, f);
    for (int x = 0; x < w; ++x) fwrite(trans[x], sizeof(trans[x][0]) * h, 1, f);
    for (int x = 0; x < w; ++x) fwrite(de    [x], sizeof(de    [x][0]) * h, 1, f);
    fclose(f);
  }
};

//-------------------------------------------------------------------------------------------------
// entry point
template <typename R_lo>
void main1(const C_hi &c, N bm, N bn, N period, N maxiters, N width, N height, float *buffer, const R_lo &tmax)
{
  typedef COMPLEX<R_lo> C_lo;
  //Compute bivariate polynomial approximating period iterations ----------------------------
  biPolyClass<R_lo> fp(bm,bn);
  refClass<R_lo> ref;
  C_hi z(0);
  C_lo zlo(0);
  int64_t lastpc = -1;
  for(N i = 0; i < period; i++){
        ref.adde(zlo);
        z = z*z + c;
        zlo = c_lo(z, tmax);
        fp.cstep(zlo);
        int64_t pc = ((i + 1) * 100) / period;
        if (pc > lastpc)
        {
          cerr << "\rreference " << pc << "%";
          lastpc = pc;
        }
  }
  cerr << "__" << c_lo(z, tmax) << endl;
  fp.print();
  cerr << "R == " << fp.getRadius() << endl;
  R_lo Bout = fp.getRadius();

  //return 1;

  //-------------------------------------------------------------------------------------------
```

```
    ↪
  //In case the location is not "exactly" at a nucleus, we need to "correct" the↙
    ↪  perturbation reference.
  //This is because the reference orbit is computed for only one period. If the ↙
    ↪ reference c is outside the mbset, it will eventually escape or be likely↙
    ↪  off 0.
  //Find the nucleus of the minibrot.
  tmpPolyClass<R_lo> tp(fp);
  C_lo nucleusPos = tp.getRoot();
  //cout << "----------------" << endl << "nucleus rel pos: " << nucleusPos << ↙
    ↪ endl;
  //Rebase the reference orbit to the nucleus
  refClass<R_lo> ref1;
  C_lo zlo1(0);
  for(N i = 0; i < period; i++){
      zlo = ref[i];
      ref1.adde(zlo+zlo1);
      zlo1 = zlo1 * (zlo1 + R_lo(2) * zlo) + nucleusPos;
  }
  //At this point zlo+zlo1 should be very close to 0
  //zlo = c_lo(z, tmax);
  //cout << "----------------" << endl << "at iteration: " << period << ", zlo1 ↙
    ↪ = " << zlo1 << endl;
  //cout << "zlo1+zlo = " << zlo1+zlo << endl;
  //-------------------------------------------------------------------------------↙
    ↪
  N progress = 0;
  #pragma omp parallel for schedule(dynamic)
  for (N y = 0; y < height; ++y)
  {
    for (N x = 0; x < width; ++x)
    {
      R_lo t0 = (x + jitter(x, y, 0)) / width * 2 * pi;
      R_lo c0 = cosl(t0);
      R_lo s0 = sinl(t0);
      R_lo t1 = (x + jitter(x, y, 0) + 1) / width * 2 * pi;
      R_lo c1 = cosl(t1);
      R_lo s1 = sinl(t1);
      R_lo l0 = (y + jitter(x, y, 1)) / width * 2 * pi;
      R_lo r0 = expl(8 - l0);
      C_lo dc(r0 * c0, r0 * s0);
      C_lo dc1(r0 * c1, r0 * s1);
      R_lo pixel_spacing = sqrtl(norm(dc1 - dc));
          N si(0);
          int iters = 0;
          float trans = 0.0f;
          //we need to send the nucleus position wrt the center of the picture.
          R_lo v = iteratePtDE2( dc, fp, ref1, nucleusPos, period, maxiters, ↙
              ↪ Bout, si, iters, trans);
          //R_lo v = iteratePtDE( dc, fp, ref, period, maxiters, Bout, si, iters↙
              ↪ , trans);

    float dwell = v <= 0 ? -1 : 16 + iters + trans;
    float distance = fabsl(v / pixel_spacing);
    float angle = 0;
    float isperiod = 0;
    float isnewton = 0;
```

```
        float *result = buffer + size_t(y) * width * CHANNELS + x * CHANNELS;
        result[0] = dwell;
        result[1] = distance;
        result[2] = angle;
        result[3] = isperiod;
        result[4] = isnewton;

        }
        #pragma omp critical
        cerr << "\rimage scanline " << ++progress << " / " << height;
      }
      cerr << endl;
    }
    #endif

    int main(int argc, char **argv) {
      if (argc != 8) exit(1);
      FILE *out = fopen(argv[1], "wb");
      N w = atoi(argv[2]);
      long double r0 = strtold(argv[3], 0);
      N bits = fmaxl(53, ceill(53 - log2l(r0)));
      mp_real cx(argv[4], bits);
      mp_real cy(argv[5], bits);
      N period = atoi(argv[6]);
      N maxsi = atoi(argv[7]);
      N maxiters = period * maxsi;
      C_hi c(cx, cy);
      N bm = 16;
      N bn = 8;
      N h = roundl(w * (16 - logl(r0)) / (2 * pi));
      size_t bytes = sizeof(float) * w * h * CHANNELS;
      float *buffer = (float *) malloc(bytes);
      main1(c, bm, bn, period, maxsi, maxiters, w, h, buffer, r0);
      fwrite(buffer, bytes, 1, out);
      fflush(out);
      fclose(out);
      return 0;
    }
```

10 emndl_colourize.c

```
    /*
        emndl -- exponentially transformed Mandelbrot Set renderer
        Copyright (C) 2011,2012  Claude Heiland-Allen <claude@mathr.co.uk>

    5   This program is free software: you can redistribute it and/or modify
        it under the terms of the GNU General Public License as published by
        the Free Software Foundation, either version 3 of the License, or
        (at your option) any later version.

    10  This program is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU General Public License for more details.

    15  You should have received a copy of the GNU General Public License
        along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

```
*/

#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <math.h>

float lin_interp(float x, float domain_low, float domain_hi, float range_low, float range_hi) {
  if ((x >= domain_low) && (x <= domain_hi)) {
    x = (x - domain_low) / (domain_hi - domain_low);
    x = range_low + x * (range_hi - range_low);
  }
  return x;
}

float pvp_adjust_3(float x) {
  // red
  x = lin_interp(x, 0.00, 0.125, -0.050, 0.090);
  // orange
  x = lin_interp(x, 0.125, 0.25,  0.090, 0.167);
  // yellow
  x = lin_interp(x, 0.25, 0.375,  0.167, 0.253);
  // chartreuse
  x = lin_interp(x, 0.375, 0.50,  0.253, 0.383);
  // green
  x = lin_interp(x, 0.50, 0.625,  0.383, 0.500);
  // teal
  x = lin_interp(x, 0.625, 0.75,  0.500, 0.667);
  // blue
  x = lin_interp(x, 0.75, 0.875,  0.667, 0.800);
  // purple
  x = lin_interp(x, 0.875, 1.00,  0.800, 0.950);
  return(x);
}

void hsv2rgb(float h, float s, float v, float *rp, float *gp, float *bp)
{
  float i, f, p, q, t, r, g, b;
  int ii;
  if (s == 0.0) {
    // Ignore hue
    r = v;
    g = v;
    b = v;
  } else {
    /* Apply physiological mapping: red, yellow, green and blue should
       be equidistant. */
    h = pvp_adjust_3(h);
    h = h - floor(h);
    h = h * 6.0;
    i = floor(h); ii = (int) i;
    f = h - i;
    p = v*(1.0 - s);  // The "low-flat" curve
    q = v*(1.0 - (s*f));  // The "falling" curve
    t = v*(1.0 - (s*(1.0 - f)));  // The "rising" curve
```

```
    switch(ii) {
      case 0:  // red point
        r = v; g = t; b = p; // RED max, GRN rising, BLU min
        break;
      case 1:  // yellow point
        r = q; g = v; b = p; // RED falling, GRN max, BLU min
        break;
      case 2:  // green point
        r = p; g = v; b = t; // RED min, GRN max, BLU rising
        break;
      case 3:  // cyan point
        r = p; g = q; b = v; // RED min, GRN falling, BLU max
        break;
      case 4:  // blue point
        r = t; g = p; b = v; // RED rising, GRN min, BLU max
        break;
      case 5:  // magenta point
      default:
        r = v; g = p; b = q; // RED max, GRN min, BLU falling
        break;
    }
  }
  *rp = r;
  *gp = g;
  *bp = b;
}

int main(int argc, char **argv) {

  if (argc != 7) exit(1);

  FILE *fddwell    = fopen(argv[1], "rb");
  FILE *fddistance = fopen(argv[2], "rb");
  FILE *fdangle    = fopen(argv[3], "rb");
  FILE *fdperiod   = fopen(argv[4], "rb");
  FILE *fdnewton   = fopen(argv[5], "rb");
  struct stat s;
  if (stat(argv[1], &s)) exit(1);
  int n = s.st_size / sizeof(float);

  FILE *fdo = fopen(argv[6], "wb");

  for (int i = 0; i < n; ++i) {
    float dwell, distance, angle, period, newton;
    if (1 != fread(&dwell,    sizeof(float), 1, fddwell   )) exit(1);
    if (1 != fread(&distance, sizeof(float), 1, fddistance)) exit(1);
    if (1 != fread(&angle,    sizeof(float), 1, fdangle   )) exit(1);
    if (1 != fread(&period,   sizeof(float), 1, fdperiod  )) exit(1);
    if (1 != fread(&newton,   sizeof(float), 1, fdnewton  )) exit(1);
    float h = dwell;
    float s = 0.5;
    float v = 0.25 + log2f(distance) / 4;
    v = fminf(fmaxf(v, 0), 1);
    if (h <= 0) {
      s = 0;
      v = 1;
    }
```

```
130       float r, g, b;
          hsv2rgb(h, s, v, &r, &g, &b);
          unsigned char rgb[3];
          rgb[0] = fminf(fmaxf(255 * r, 0), 255);
          rgb[1] = fminf(fmaxf(255 * g, 0), 255);
135       rgb[2] = fminf(fmaxf(255 * b, 0), 255);
          if (1 != fwrite(rgb, 3, 1, fdo)) exit(1);
        }

        fclose(fdo);
140     fclose(fddwell);
        fclose(fddistance);
        fclose(fdangle);
        fclose(fdperiod);
        fclose(fdnewton);
145
        return 0;
      }
```

11 emndl_comb~.pd

```
#N canvas 0 0 450 300 10;
#N canvas 0 0 450 300 \$0-early 0;
#X obj 40 40 inlet~;
#X obj 40 61 delwrite~ \$0-del 100;
 5  #X obj 40 82 outlet~;
#X connect 0 0 1 0;
#X restore 36 82 pd \$0-early;
#X obj 16 21 inlet~;
#N canvas 0 0 450 300 \$0-late 0;
10  #X obj 18 20 inlet~;
#X obj 18 62 outlet~;
#X obj 18 41 delread~ \$0-del 20;
#X connect 2 0 1 0;
#X restore 36 103 pd \$0-late;
15  #X obj 15 161 -~;
#X obj 15 182 outlet~;
#X obj 36 124 *~ 0.9;
#X obj 16 46 lop~ 8000;
#X connect 0 0 2 0;
20  #X connect 1 0 6 0;
#X connect 2 0 5 0;
#X connect 3 0 4 0;
#X connect 5 0 3 1;
#X connect 6 0 3 0;
25  #X connect 6 0 0 0;
```

12 emndl_compress~.pd

```
#N canvas 275 79 452 587 10;
#X obj 28 28 inlet~;
#X obj 176 20 inlet~;
#X obj 27 543 outlet~;
 5  #X obj 177 544 outlet~;
#X obj 58 71 hip~ 5;
#X obj 121 71 hip~ 5;
#X obj 58 97 *~;
```

```
#X obj 120 98 *~;
#X obj 87 158 +~;
#X obj 281 25 inlet;
#X obj 87 281 rmstodb~;
#X obj 86 353 dbtorms~;
#X obj 27 474 *~;
#X obj 177 474 *~;
#X obj 120 123 lop~ 10;
#X obj 57 124 lop~ 10;
#X obj 87 221 lop~ 25;
#X obj 25 272 /~;
#X obj 178 270 /~;
#X obj 201 390 dbtorms;
#X obj 86 417 /~ 0;
#X obj 87 179 sqrt~;
#X obj 86 442 *~ 0.25;
#X obj 310 55 loadbang;
#X obj 310 79 f \$1;
#X obj 27 514 expr~ tanh($v1);
#X obj 177 514 expr~ tanh($v1);
#X obj 86 314 expr~ if($v1>$f2 \, $f2+($v1-$f2)/8 \, $f2);
#X obj 201 366 expr (100-$f1)/8+$f1;
#X obj 88 202 +~ 0.01;
#X connect 0 0 4 0;
#X connect 0 0 17 0;
#X connect 1 0 5 0;
#X connect 1 0 18 0;
#X connect 4 0 6 0;
#X connect 4 0 6 1;
#X connect 5 0 7 0;
#X connect 5 0 7 1;
#X connect 6 0 15 0;
#X connect 7 0 14 0;
#X connect 8 0 21 0;
#X connect 9 0 27 1;
#X connect 9 0 28 0;
#X connect 10 0 27 0;
#X connect 11 0 20 0;
#X connect 12 0 25 0;
#X connect 13 0 26 0;
#X connect 14 0 8 1;
#X connect 15 0 8 0;
#X connect 16 0 17 1;
#X connect 16 0 18 1;
#X connect 16 0 10 0;
#X connect 17 0 12 0;
#X connect 18 0 13 0;
#X connect 19 0 20 1;
#X connect 20 0 22 0;
#X connect 21 0 29 0;
#X connect 22 0 12 1;
#X connect 22 0 13 1;
#X connect 23 0 24 0;
#X connect 24 0 27 1;
#X connect 24 0 28 0;
#X connect 25 0 2 0;
#X connect 26 0 3 0;
```

```
#X connect 27 0 11 0;
#X connect 28 0 19 0;
#X connect 29 0 16 0;
```

13 emndl_downscale.c

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011,2012  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv) {
  if (argc != 3) exit(1);
  int m = -1;
  if (strcmp("dwell",    argv[1]) == 0) m = 0; else
  if (strcmp("distance", argv[1]) == 0) m = 1; else
  if (strcmp("angle",    argv[1]) == 0) m = 0; else
  if (strcmp("period",   argv[1]) == 0) m = 0; else
  if (strcmp("newton",   argv[1]) == 0) m = 0;
  if (m == -1) exit(1);
  int iwidth = atoi(argv[2]);
  int owidth = iwidth >> 1;
  float *iline[2];
  float *oline;
  int ibytes = iwidth * sizeof(float);
  int obytes = owidth * sizeof(float);
  iline[0] = (float *) malloc(ibytes);
  iline[1] = (float *) malloc(ibytes);
  oline    = (float *) malloc(obytes);
  while ((1 == fread(iline[0], ibytes, 1, stdin)) &&
         (1 == fread(iline[1], ibytes, 1, stdin))) {
    for (int ox = 0; ox < owidth; ++ox) {
      int i = ox << 1;
      int j = i + 1;
      int k = 0;
      float o;
      if (m == 0) {
        o = 0.0;
        if (iline[0][i] > 0) { k += 1; o += iline[0][i]; }
```

```
        if (iline[0][j] > 0) { k += 1; o += iline[0][j]; }
        if (iline[1][i] > 0) { k += 1; o += iline[1][i]; }
        if (iline[1][j] > 0) { k += 1; o += iline[1][j]; }
        if (k > 0) {
          o /= k;
        }
      } else {
        o = 1.0;
        if (iline[0][i] > 0) { k += 1; o *= iline[0][i]; }
        if (iline[0][j] > 0) { k += 1; o *= iline[0][j]; }
        if (iline[1][i] > 0) { k += 1; o *= iline[1][i]; }
        if (iline[1][j] > 0) { k += 1; o *= iline[1][j]; }
        if (k > 1) {
          o = 0.5 * powf(o, 1.0 / k);
        } else {
          o = 0.0;
        }
      }
      oline[ox] = o;
    }
    if (fwrite(oline, obytes, 1, stdout) != 1) exit(1);
  }
  free(iline[0]);
  free(iline[1]);
  free(oline);
  return 0;
}
```

14 emndl_equalize.c

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>

struct pixel { float x; int i; };

int pixel_cmp_i(const void *a, const void *b) {
  const struct pixel *i = a;
```

```
  const struct pixel *j = b;
  if (i->i < j->i) return -1;
  if (i->i > j->i) return  1;
  return 0;
}

int pixel_cmp_x(const void *a, const void *b) {
  const struct pixel *i = a;
  const struct pixel *j = b;
  if (i->x < j->x) return -1;
  if (i->x > j->x) return  1;
  return 0;
}

int main(int argc, char **argv) {

  if (argc != 3) exit(1);

  FILE *fdi = fopen(argv[1], "rb");
  struct stat s;
  if (stat(argv[1], &s)) exit(1);
  int n = s.st_size / sizeof(float);
  struct pixel *p = calloc(n, sizeof(struct pixel));
  if (! p) exit(1);
  for (int i = 0; i < n; ++i) {
    float x;
    if (1 != fread(&x, sizeof(float), 1, fdi)) exit(1);
    p[i].x = x;
    p[i].i = i;
  }
  fclose(fdi);

  qsort(p, n, sizeof(struct pixel), pixel_cmp_x);
  int k = 0;
  while (p[k].x <= 0 && k < n) ++k;
  float f = 1.0 / (n - k);
  for (int i = k; i < n; ++i) {
    p[i].x = f * (1 + i - k);
  }
  qsort(p, n, sizeof(struct pixel), pixel_cmp_i);

  FILE *fdo = fopen(argv[2], "wb");
  for (int i = 0; i < n; ++i) {
    if (1 != fwrite(&p[i].x, sizeof(float), 1, fdo)) exit(1);
  }
  fclose(fdo);
  free(p);

  return 0;
}
```

15 emndl_filter.pd

```
#N canvas 0 0 450 300 10;
#X obj 63 117 vcf~;
#X obj 157 118 vcf~;
#X obj 239 73 loadbang;
```

```
#X obj 239 94 f \$1;
#X obj 239 167 /;
#X obj 50 267 throw~ \$2-o-l;
#X obj 154 265 throw~ \$2-o-r;
#X obj 61 21 receive~ \$2-i-l;
#X obj 158 25 receive~ \$2-i-r;
#X obj 203 46 receive~ \$2-f;
#X obj 204 81 *~ 1;
#X obj 63 139 *~ 1;
#X obj 157 141 *~ 1;
#X obj 239 120 sqrt;
#X obj 239 188 sqrt;
#X obj 284 93 bng 15 250 50 0 empty empty empty 17 7 0 10 -262144 -1
-1;
#X obj 50 217 emndl_compress~ 48;
#X obj 154 240 /~;
#X obj 50 238 /~;
#X obj 21 168 expr~ sin(tanh($v1)) \; sin(tanh($v2));
#X obj 239 146 swap 100;
#X connect 0 0 11 0;
#X connect 1 0 12 0;
#X connect 2 0 3 0;
#X connect 3 0 10 1;
#X connect 3 0 13 0;
#X connect 4 0 1 2;
#X connect 4 0 0 2;
#X connect 4 0 14 0;
#X connect 7 0 0 0;
#X connect 8 0 1 0;
#X connect 9 0 10 0;
#X connect 10 0 1 1;
#X connect 10 0 0 1;
#X connect 11 0 19 0;
#X connect 12 0 19 1;
#X connect 13 0 20 0;
#X connect 13 0 17 1;
#X connect 13 0 18 1;
#X connect 14 0 12 1;
#X connect 14 0 11 1;
#X connect 15 0 3 0;
#X connect 16 0 18 0;
#X connect 16 1 17 0;
#X connect 17 0 6 0;
#X connect 18 0 5 0;
#X connect 19 0 16 0;
#X connect 19 1 16 1;
#X connect 20 0 4 0;
#X connect 20 1 4 1;
```

16 emndl_finalize.cc

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011,2012  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
```

```
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv) {
  if (argc != 7) exit(1);
  FILE *fdi = fopen(argv[1], "rb");
  FILE *fddwell = fopen(argv[2], "wb");
  FILE *fddistance = fopen(argv[3], "wb");
  FILE *fdangle = fopen(argv[4], "wb");
  FILE *fdperiod = fopen(argv[5], "wb");
  FILE *fdnewton = fopen(argv[6], "wb");
  float dda[5];
  while (1 == fread(dda, 5 * sizeof(float), 1, fdi)) {
    if (1 != fwrite(&dda[0], sizeof(float), 1, fddwell)) exit(1);
    if (1 != fwrite(&dda[1], sizeof(float), 1, fddistance)) exit(1);
    if (1 != fwrite(&dda[2], sizeof(float), 1, fdangle)) exit(1);
    if (1 != fwrite(&dda[3], sizeof(float), 1, fdperiod)) exit(1);
    if (1 != fwrite(&dda[4], sizeof(float), 1, fdnewton)) exit(1);
  }
  fclose(fdi);
  fclose(fddwell);
  fclose(fddistance);
  fclose(fdangle);
  fclose(fdperiod);
  fclose(fdnewton);
  return 0;
}
```

17 emndl_loader.pd

```
#N canvas 0 0 450 300 10;
#X obj 23 19 r emndl;
#X obj 23 111 list;
#X msg 100 56 \; pd dsp 1;
#X obj 23 90 delay 2000;
#X obj 100 91 delay 1000;
#N canvas 0 0 450 300 \$0-loader 0;
#X restore 50 180 pd \$0-loader;
#X obj 23 40 spigot 1;
#X msg 100 36 0;
#X obj 100 153 s pd-\$0-loader;
#X obj 23 132 s emndl;
#X msg 143 178 clear;
#X obj 23 69 t b a b;
#X msg 100 132 obj 10 10 emndl_audio \, loadbang;
```

```
#X obj 100 112 t b b b;
#X msg 160 56 \; pd dsp 0;
#X connect 0 0 6 0;
#X connect 1 0 9 0;
#X connect 3 0 1 0;
#X connect 4 0 13 0;
#X connect 6 0 11 0;
#X connect 7 0 6 1;
#X connect 10 0 8 0;
#X connect 11 0 3 0;
#X connect 11 1 1 1;
#X connect 11 2 2 0;
#X connect 11 2 4 0;
#X connect 11 2 7 0;
#X connect 12 0 8 0;
#X connect 13 0 2 0;
#X connect 13 1 12 0;
#X connect 13 2 14 0;
```

18 emndl_parse.cc

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdio.h>
#include <qd/qd_real.h>
#include <qd/fpu.h>

int main(int argc, char **argv) {
  fpu_fix_start(0);
  if (argc != 3) return 1;
  qd_real re(argv[1]);
  qd_real im(argv[2]);
  fwrite(&re, sizeof(qd_real), 1, stdout);
  fwrite(&im, sizeof(qd_real), 1, stdout);
  return 0;
}
```

19 emndl_ppmtoy4m.c

```
/*
emndl_ppmtoy4m -- faster PPM to YUV4MPEG2 conversion
```

```
    Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>

 5  This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

10  This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

15  You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.


    Portions of this code are derived from mjpegtools:
20  "mjpeg/mjpeg_play/lavtools/colorspace.c"

    colorspace.c: Routines to perform colorspace conversions.
    Copyright (C) 2001 Matthew J. Marjanovic <maddog@mir.com>

25  This program is free software; you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation; either version 2
    of the License, or (at your option) any later version.

30  This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

35  You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
    */

40  #include <stdint.h>
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>

45  #define FP_BITS 18

    static inline int myround(double n) {
      if (n >= 0) return (int)(n + 0.5);
      else        return (int)(n - 0.5);
50  }

    #define Y 0
    #define U 1
    #define V 2
55  #define R 0
    #define G 1
    #define B 2

    int main(int argc, char **argv) {
```

```
60    if (argc != 3) { return 1; }
      char *widths = argv[1];
      char *heights = argv[2];
      char ppmtemplate[1024];
      snprintf(ppmtemplate, 1000, "P6\n%s %s 255\n", widths, heights);
65    int width;
      int height;
      if (sscanf(ppmtemplate, "P6\n%d %d 255\n", &width, &height) != 2) { fprintf(↙
          ↳ stderr, "ppm template parse\n"); return 1; }
      int n = width * height;
      unsigned char *rgbs = malloc(n * 3);
70    unsigned char *ys = malloc(n);
      unsigned char *us = malloc(n);
      unsigned char *vs = malloc(n);
      char y4m[1024];
      snprintf(y4m, 1000, "YUV4MPEG2 W%d H%d F60:1 Ip A1:1 C444\n", width, height);
75    if (fwrite(y4m, strlen(y4m), 1, stdout) != 1) { fprintf(stderr, "y4m header ↙
          ↳ write\n"); return 1; }
      int32_t cc[3][3][256];
      for (int i = 0; i < 256; ++i) {
        cc[Y][R][i] = myround(0.299 * (double)i * 219.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS));
        cc[Y][G][i] = myround(0.587 * (double)i * 219.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS));
80      cc[Y][B][i] = myround((0.114 * (double)i * 219.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS)) + (double)(1<<(FP_BITS-1)) + (16.0 * (double)(1<<FP_BITS)));
        cc[U][R][i] = myround(-0.168736 * (double)i * 224.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS));
        cc[U][G][i] = myround(-0.331264 * (double)i * 224.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS));
        cc[U][B][i] = myround((0.500 * (double)i * 224.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS)) + (double)(1<<(FP_BITS-1)) + (128.0 * (double)(1<<FP_BITS)))↙
            ↳ ;
        cc[V][R][i] = myround(0.500 * (double)i * 224.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS));
85      cc[V][G][i] = myround(-0.418688 * (double)i * 224.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS));
        cc[V][B][i] = myround((-0.081312 * (double)i * 224.0 / 255.0 * (double)(1<<↙
            ↳ FP_BITS)) + (double)(1<<(FP_BITS-1)) + (128.0 * (double)(1<<FP_BITS)))↙
            ↳ ;
      }
      int ppml = strlen(ppmtemplate);
      char ppm[1024];
90    while (fread(ppm, ppml, 1, stdin) == 1) {
        ppm[ppml] = '\0';
        if (strncmp(ppmtemplate, ppm, ppml)) { fprintf(stderr, "ppm template ↙
            ↳ mismatch\n%s%s", ppmtemplate, ppm); return 1; }
        if (fread(rgbs, n * 3, 1, stdin) != 1) { fprintf(stderr, "ppm read mismatch\↙
            ↳ n"); return 1; }
        unsigned char *rgb = rgbs;
95      unsigned char *y = ys;
        unsigned char *u = us;
        unsigned char *v = vs;
        for (int i = 0; i < n; ++i) {
          int r = *rgb++;
100       int g = *rgb++;
          int b = *rgb++;
```

```
      *y++ = (cc[Y][R][r] + cc[Y][G][g] + cc[Y][B][b]) >> FP_BITS;
      *u++ = (cc[U][R][r] + cc[U][G][g] + cc[U][B][b]) >> FP_BITS;
      *v++ = (cc[V][R][r] + cc[V][G][g] + cc[V][B][b]) >> FP_BITS;
    }
    if (fwrite("FRAME\n", 6, 1, stdout) != 1) { fprintf(stderr, "y4m frame write↙
        ↳ \n"); return 1; }
    if (fwrite(ys, n, 1, stdout) != 1) { fprintf(stderr, "y4m y write\n"); ↙
        ↳ return 1; }
    if (fwrite(us, n, 1, stdout) != 1) { fprintf(stderr, "y4m u write\n"); ↙
        ↳ return 1; }
    if (fwrite(vs, n, 1, stdout) != 1) { fprintf(stderr, "y4m v write\n"); ↙
        ↳ return 1; }
  }
  return 0;
}
```

20 emndl_pretty.cc

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <qd/qd_real.h>
#include <qd/fpu.h>

int main(int argc, char **argv) {
  fpu_fix_start(0);
  qd_real q[2];
  std::cout << std::setprecision(64);
  while (2 == fread(q, sizeof(qd_real), 2, stdin)) {
    std::cout << q[0] << " " << q[1] << std::endl;
  }
  return 0;
}
```

21 emndl_randomize.cc

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>
```

```
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <qd/qd_real.h>

#define likely(x)   __builtin_expect((x),1)
#define unlikely(x) __builtin_expect((x),0)

qd_real exterior_distance(const qd_real &cx, const qd_real &cy, int *n, int ↙
    ↳ count, const qd_real &r2) {
  qd_real px = 0;
  qd_real py = 0;
  qd_real dx = 1;
  qd_real dy = 0;
  int escape = 0;
  while (likely(count--)) {
    qd_real px2 = px * px;
    qd_real py2 = py * py;
    qd_real d2 = px2 + py2;
    if (d2 > r2) { escape = 1; break; }
    qd_real pxy = px * py;
    qd_real px1 = px2 - py2 + cx;
    qd_real py1 = 2 * pxy + cy;
    qd_real dx1 = 2 * (px * dx - py * dy) + 1;
    qd_real dy1 = 2 * (dx * py + dy * px);
    px = px1; py = py1; dx = dx1; dy = dy1;
    ++(*n);
  }
  if (escape) {
    qd_real p = sqrt(px * px + py * py);
    qd_real d = sqrt(dx * dx + dy * dy);
    return 2 * p * log(p) / d;
  } else {
    return -1;
  }
}

void random_interesting(qd_real &re, qd_real &im, qd_real &d) {
  int interesting = 0;
  qd_real x;
```

```
60    qd_real y;
      while (likely(! interesting)) {
        x = re + d * 2.0 * (qdrand() - 0.5);
        y = im + d * 2.0 * (qdrand() - 0.5);
        int n = 0;
65      qd_real e = exterior_distance(x, y, &n, 1024, 65536);
        if (0 < e && e < d) {
          re = x;
          im = y;
          d = e;
70        std::cerr << std::setprecision(64) << re << " " << im << " " << std::↙
              ↪ setprecision(8) << d << "\r";
        }
        interesting = 0 < e && e < pow(qd_real(2.0), int(-180));
      }
    }
75
    int main(int argc, char **argv) {
      fpu_fix_start(0);
      srand(time(0));
      qd_real re(0.0);
80    qd_real im(0.0);
      qd_real d(2.0);
      random_interesting(re, im, d);
      std::cerr << std::endl;
      fwrite(&re, sizeof(qd_real), 1, stdout);
85    fwrite(&im, sizeof(qd_real), 1, stdout);
      return 0;
    }
```

22 emndl.sh

```
    #!/bin/bash
    #
    #    emndl -- exponentially transformed Mandelbrot Set renderer
    #    Copyright (C) 2011,2012,2018  Claude Heiland-Allen <claude@mathr.co.uk>
5   #
    #    This program is free software: you can redistribute it and/or modify
    #    it under the terms of the GNU General Public License as published by
    #    the Free Software Foundation, either version 3 of the License, or
    #    (at your option) any later version.
10  #
    #    This program is distributed in the hope that it will be useful,
    #    but WITHOUT ANY WARRANTY; without even the implied warranty of
    #    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    #    GNU General Public License for more details.
15  #
    #    You should have received a copy of the GNU General Public License
    #    along with this program.  If not, see <http://www.gnu.org/licenses/>.
    #

20  TIME0="$(date --iso=s)"
    E="$(dirname "$(readlink -f "$0")")"
    OUT="emndl-${TIME0}"

    version()
25  {
```

```
    cat << EOF
    emndl.sh git (GPL) 2011,2012,2018  Claude Heiland-Allen <claude@mathr.co.uk>
    EOF
    }

    usage()
    {
    cat << EOF
    usage options:
        --help -h -?    show this message and exit
        --version       show version and exit

    input options:
        --auto bits     find an interesting point with precision 'bits'
        --re real       choose coordinates
        --im imag
        --period p
        --maxiters n
        --size sz

    render options:
        --quality q     horizontal size, (5 - 12)

    image options:
        --png           generate full size PNG
        --jpeg          generate thumbnail JPEG

    video options:
        --aspect w:h    output aspect (4:3, 16:9)
        --speed s       zoom speed (default 1)
        --dvd           generate PAL DVD (SD)
        --mkv           generate Matroska (HD)
        --ogv           generate Theora (LD)

    other options:
        --quiet         less output
        --verbose       more output
    EOF
    }

    if [ "x$*" == "x" ]
    then
      version
      cat << EOF
    Try \`emndl.sh --help' for usage information.
    EOF
      exit 0
    fi

    TEMP=`getopt -o '' --long help,version,auto:,re:,im:,period:,maxiters:,size:,↙
        ↪ quality:,png,jpeg,aspect:,speed:,dvd,mkv,ogv,quiet,verbose -n 'emndl.sh' ↙
        ↪ -- "$@"`
    if [ $? != 0 ]
    then
      echo "error 1 in getopt" >&2
      exit 1
    fi
```

```
eval set -- "$TEMP"

AUTO=0
AUTOBITS="128"
REAL=0
REALCOORD="0.0"
IMAG=0
IMAGCOORD="0.0"
PER=0
PERIOD=1
MAXITERS=1000
SZ=0
SIZE=1
QUALITY="8"
PNG=0
JPEG=0
ASPECT="16:9"
SPEED="1"
DVD=0
MKV=0
OGV=0
VERBOSE=1

while true
do
  case "$1" in
    --help|-h|-\?) usage ; exit 0 ;;
    --version) version ; exit 0 ;;
    --auto) AUTO=1 ; AUTOBITS="$2" ; shift 2 ;;
    --re) REAL=1 ; REALCOORD="$2" ; shift 2 ;;
    --im) IMAG=1 ; IMAGCOORD="$2" ; shift 2 ;;
    --period) PER=1; PERIOD="$2" ; shift 2 ;;
    --maxiters) MIT=1; MAXITERS="$2" ; shift 2 ;;
    --size) SZ=1; SIZE="$2" ; shift 2 ;;
    --quality) QUALITY="$2" ; shift 2 ;;
    --png) PNG=1 ; shift ;;
    --jpeg) JPEG=1 ; shift ;;
    --aspect) ASPECT="$2" ; shift 2 ;;
    --speed) SPEED="$2" ; shift 2 ;;
    --dvd) DVD=1 ; shift ;;
    --mkv) MKV=1 ; shift ;;
    --ogv) OGV=1 ; shift ;;
    --quiet) VERBOSE=0 ; shift ;;
    --verbose) VERBOSE=2 ; shift ;;
    --) shift ; break ;;
    *) echo "error 2 in getopt" >&2 ; exit 1 ;;
  esac
done

(

case "$ASPECT" in
  4:3) WIDE=0 ;;
  16:9) WIDE=1 ;;
  *) echo "aspect $ASPECT not recognized" >&2 ; exit 1 ;;
esac
```

```
case "$QUALITY" in
  5) WIDTH=32 ; VWIDTH=32 ;;
  6) WIDTH=64 ; VWIDTH=64 ;;
  7) WIDTH=128 ; VWIDTH=128 ;;
  8) WIDTH=256 ; VWIDTH=256 ;;
  9) WIDTH=512 ; VWIDTH=512 ;;
  10) WIDTH=1024 ; VWIDTH=960 ;;
  11) WIDTH=2048 ; VWIDTH=1920 ;;
  12) WIDTH=4096 ; VWIDTH=3840 ;;
  *) echo "quality $QUALITY not recognized" >&2 ; exit 1 ;;
esac

if [ $WIDE -eq 1 ]
then
  VHEIGHT=$((VWIDTH / 16 * 9))
else
  VHEIGHT=$((VWIDTH / 4 * 3))
fi

if [ $AUTO -eq 1 ]
then
  # FIXME
  TEMP="$("${E}/emndl_automu" "${AUTOBITS}" "${PERIOD}" "${REALCOORD}" "${IMAGCOORD}" 2>/dev/null | tee /dev/stderr | tail -n 1 | ( read b p r i s ; printf "%s\n" "${r} ${i} ${p}") )"
  REALCOORD="$(echo "${TEMP}" | sed 's|^\([^ ]*\) .*$|\1|')"
  IMAGCOORD="$(echo "${TEMP}" | sed 's|^[^ ]* \([^ ]*\) .*$|\1|')"
  PERIOD="$(echo "${TEMP}" | sed 's|^[^ ]* [^ ]* \([^ ]*\)$|\1|')"
  SIZE="$(m-size "${AUTOBITS}" "${REALCOORD}" "${IMAGCOORD}" "${PERIOD}" | cut -d\  -f 1)"
fi

if [ $VERBOSE -gt 0 ]
then
  cat << EOF
AUTO=$AUTO
AUTOBITS=$AUTOBITS
REAL=$REAL
REALCOORD=$REALCOORD
IMAG=$IMAG
IMAGCOORD=$IMAGCOORD
PER=$PER
PERIOD=$PERIOD
MAXITERS=$MAXITERS
SZ=$SZ
SIZE=$SIZE
QUALITY=$QUALITY
PNG=$PNG
JPEG=$JPEG
ASPECT=$ASPECT
SPEED=$SPEED
DVD=$DVD
MKV=$MKV
OGV=$OGV
VERBOSE=$VERBOSE
```

```
WIDE=$WIDE
WIDTH=$WIDTH
VWIDTH=$VWIDTH
VHEIGHT=$VHEIGHT
E=$E
OUT=$OUT
EOF
fi

mkdir "${OUT}"
cd "${OUT}"

if [ $VERBOSE -gt 0 ] ; then echo "calculating.." ; fi
"${E}/emndl_calculate" "out.emndl" "${WIDTH}" "${SIZE}" "${REALCOORD}" "${IMAGCOORD}" "${PERIOD}" "${MAXITERS}"

if [ $VERBOSE -gt 0 ] ; then echo "finalizing.." ; fi
"${E}/emndl_finalize" "out.emndl" "dwell-neq.f" "distance.f" "angle.f" "period.f" " "newton.f"

if [ $VERBOSE -gt 0 ] ; then echo "equalizing.." ; fi
"${E}/emndl_equalize" "dwell-neq.f" "dwell.f"

if [ $VERBOSE -gt 0 ] ; then echo "computing depth.." ; fi
DEPTH="$(( $(stat -c '%s' "dwell.f") / (WIDTH * WIDTH * 4) ))"
if [ $VERBOSE -gt 0 ] ; then echo "DEPTH=${DEPTH}" ; fi

if [ $VERBOSE -gt 0 ] ; then echo "downscaling.." ; fi
for p in dwell distance angle period newton
do
  cat "${p}.f" "/dev/zero" | head -c "$(( WIDTH * WIDTH * (DEPTH + 1) * 4 ))" > "${p}-00.f"
done
for q in $(seq "0" "$((QUALITY - 1))")
do
  for p in dwell distance angle period newton
  do
    "${E}/emndl_downscale" "${p}" "$(( 1 << (QUALITY - q) ))" < "${p}-$(printf "%02d" "$((q))").f" > "${p}-$(printf "%02d" "$((q + 1))").f"
  done
done

if [ $VERBOSE -gt 0 ] ; then echo "colourizing.." ; fi
for q in $(seq 0 "${QUALITY}")
do
  f="$(printf "%02d" "${q}").f"
  r="${f%f}raw"
  "${E}/emndl_colourize" "dwell-${f}" "distance-${f}" "angle-${f}" "period-${f}" "newton-${f}" "${r}"
  w="$(( 1 << (QUALITY - q) ))"
  h="$(( w * (DEPTH + 1) ))"
  ( echo -e "P6\n${w} ${h} 255" && cat "${r}" ) > "emndl.${f%f}ppm"
done

if [ $VERBOSE -gt 0 ] ; then echo "making images.." ; fi
if [ $PNG  -eq 1 ] ; then convert "emndl.00.ppm" "out.png" ; fi
if [ $JPEG -eq 1 ] ; then convert "emndl.00.ppm" -thumbnail 64x "out.jpeg" ; fi
```

```
245 if [ $VERBOSE -gt 0 ] ; then echo "making videos.." ; fi

    if [ $MKV -eq 1 ]
    then
      (
250     "${E}/emndl_unwarp" emndl "${VWIDTH}" "${VHEIGHT}" "${SPEED}" 2>audio.mkv.raw |
        "${E}/emndl_ppmtoy4m" "${VWIDTH}" "${VHEIGHT}" |
        ffmpeg -f yuv4mpegpipe -i - -pix_fmt yuv420p -profile:v high -level:v 4.1 -crf:v 20 video.mkv >/dev/null 2>&1
      ) &
    fi
255
    if [ $DVD -eq 1 ]
    then
      ( "${E}/emndl_unwarp" emndl 1048 576 "${SPEED}" 2>audio.dvd.raw |
        "${E}/emndl_ppmtoy4m" 1048 576 |
260     y4mscaler -v 0 -O preset=dvd_wide -O yscale=1/1 |
        mpeg2enc -v 0 -f 8 -q 3 -b 8000 -B 768 -D 9 -g 9 -G 15 -P -R 2 -o "video.m2v"
      ) &
    fi

265 if [ $OGV -eq 1 ]
    then
      ( "${E}/emndl_unwarp" emndl 512 288 "${SPEED}" 2>audio.ogv.raw |
        "${E}/emndl_ppmtoy4m" 512 288 |
        ffmpeg2theora -o video.ogv - >/dev/null 2>&1
270   ) &
    fi

    wait

275 if [ $VERBOSE -gt 0 ] ; then echo "making soundtrack.." ; fi
    if [ $((DVD + MKV + OGV)) -gt 0 ]
    then
      audio_raw="$(ls -1 audio.???.raw | head -n 1)"
      ecasound -q -f:f32_le,2,48000,i -i:"${audio_raw}" -f:s16_le,2,48000,i -o:audio-raw.wav
280   if [ "x$(pwd | tr -cd "[:space:]\\{}")" == "x" ]
      then
        pd -nrt -noaudio -noprefs -path . -r 48000 -batch -open "${E}/../share/emndl/emndl_loader.pd" -send "emndl audio-raw.wav $(pwd)/audio.wav"
      else
        echo "warning: dir \`$(pwd)' contains bad chars, aborting"
285     exit
      fi
    fi

    if [ $DVD -eq 1 ]
290 then
      (
        twolame --quiet -b 224 audio.wav audio.mp2
        mplex -v 0 -f 8 -V -o out.mpeg video.m2v audio.mp2
      ) &
295 fi
```

```
if [ $MKV -eq 1 ]
then
  (
    flac --best --verify audio.wav -o audio.flac
    mkvmerge -o out.mkv video.mkv audio.flac
  ) &
fi
if [ $OGV -eq 1 ]
then
  (
    oggenc --quiet -b 192 -o audio.ogg audio.wav
    oggz-merge -o out.ogv video.ogv audio.ogg
  ) &
fi
wait

cd ..
TIME1="$(date --iso=s)"

if [ $VERBOSE -gt 0 ]
then
  echo "${TIME0}  begin"
  echo "${TIME1}  end"
fi

) 2>&1 | tee "${OUT}.log"

exit 0
```

23 emndl_unwarp.c

```
/*
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct image {
  size_t width, height, channels, levels;
  unsigned char **data;
```

```
};

int ilog2(int i) {
  int l = 0;
  while (i > 0) {
    i >>= 1;
    l += 1;
  }
  return l - 1;
}

struct image *loadppms(char *filestem) {
  struct image *img = calloc(sizeof(struct image), 1);
  img->levels = 1;
  for (int i = 0; i < img->levels; ++i) {
    char filename[1024];
    snprintf(filename, 1000, "%s.%02d.ppm", filestem, i);
    FILE *f = fopen(filename, "rb");
    if (!f) exit(6);
    int w, h;
    if (2 != fscanf(f, "P6\n%d %d 255", &w, &h)) exit(4);
    fgetc(f);
    if (i == 0) {
      img->width = w;
      img->height = h;
      img->channels = 3;
      img->levels = ilog2(w) + 1;
      img->data = calloc(sizeof(unsigned char *), img->levels);
      if (!img->data) exit(7);
    } else {
      if (w != img->width  >> i) exit(8);
      if (h != img->height >> i) exit(9);
    }
    img->data[i] = malloc(((size_t)w) * h * 3);
    if (!img->data[i]) exit(11);
    if (1 != fread(img->data[i], ((size_t)w) * h * 3, 1, f)) exit(5);
    fclose(f);
  }
  return img;
}

inline float pixel(const struct image *img, int l, int y, int x, int c) {
  if (0 <= l && l < img->levels
   && 0 <= y && y < img->height >> l
   && 0 <= x && x < img->width  >> l
   && 0 <= c && c < img->channels) {
    return img->data[l][img->channels * ((img->width >> l) * y + x) + c]/255.0f;
  } else {
    exit(42);
  }
}

inline float clamp(float x, float lo, float hi) {
  if (!(x > lo)) return lo;
  if (!(x < hi)) return hi;
  return x;
}
```

```
inline float wrap(float x, float lo, float hi) {
  if (x <  lo) return wrap(x + (hi - lo), lo, hi);
  if (x >= hi) return wrap(x - (hi - lo), lo, hi);
  return x;
}

inline float linear(float u, float v, float t) {
  return u * (1 - t) + t * v;
}

inline float pixel2ly(const struct image *img, int l, int y, float x, int c) {
  int x0 = floor(x);
  float xt = x - x0;
  int x1 = x0 + 1;
  x0 = wrap(x0, 0, img->width >> l);
  x1 = wrap(x1, 0, img->width >> l);
  return linear(pixel(img, l, y, x0, c), pixel(img, l, y, x1, c), xt);
}

inline float pixel2l(const struct image *img, int l, float y, float x, int c) {
  int y0 = floor(y);
  float yt = y - y0;
  int y1 = y0 + 1;
  y0 = clamp(y0, 0, (img->height >> l) - 1);
  y1 = clamp(y1, 0, (img->height >> l) - 1);
  return linear(pixel2ly(img, l, y0, x, c), pixel2ly(img, l, y1, x, c), yt);
}

inline float pixel2(const struct image *img, float l, float y, float x, int c) {
  int l0 = floor(l);
  float lt = l - l0;
  int l1 = l0 + 1;
  l0 = clamp(l0, 0, img->levels - 1);
  l1 = clamp(l1, 0, img->levels - 1);
  return linear( pixel2l(img, l0, y / (1 << l0), x / (1 << l0), c)
               , pixel2l(img, l1, y / (1 << l1), x / (1 << l1), c), lt);
}

int main(int argc, char **argv) {
  if (argc != 5) exit(1);
  char *stem = argv[1];
  int ow = atoi(argv[2]);
  int oh = atoi(argv[3]);
  float speed = atof(argv[4]);
  struct image *img = loadppms(stem);
  unsigned char *out = malloc(ow * oh * 3);
  float SR = 48000;
  float fps = 60;
  float *au = malloc((int)roundf(SR/fps) * 2 * sizeof(float));
  char header[1024];
  snprintf(header, 1000, "P6\n%d %d 255\n", ow, oh);
  float h = 1.0 / sqrt(ow * ow + oh * oh);
  float k = img->width / 8.0;
  float dy = speed * k / fps;

  int owoh = ow * oh;
```

```
  float *ds = malloc(owoh * sizeof(*ds));
  float *xs = malloc(owoh * sizeof(*xs));
  float *ls = malloc(owoh * sizeof(*ls));
  for (int j = 0; j < oh; ++j) {
    for (int i = 0; i < ow; ++i) {
      int k = j * ow + i;
      float p = (i - (ow >> 1) + 0.5f) * h;
      float q = ((oh >> 1) - j - 0.5f) * h;
      ds[k] = (logf(sqrtf(p * p + q * q)) * 0.15915494309189535f) * img->width;
      xs[k] = (atan2f(-q, -p) * 0.15915494309189535f + 0.5f) * img->width;
      ls[k] = - log2f(sqrtf(p * p + q * q)) - 2;
    }
  }

  for (float y0 = 0; y0 < img->height - img->width; y0 += dy)
  {
    #pragma omp parallel for
    for (int k = 0; k < owoh; ++k)
    {
      for (int c = 0; c < 3; ++c)
      {
        out[k * 3 + c] = clamp(pixel2(img, ls[k], y0 - ds[k], xs[k], c) * 255, ↙
            ↳ 0, 255);
      }
    }
    if (1 != fwrite(header, strlen(header), 1, stdout)) exit(2);
    if (1 != fwrite(out, ow * oh * 3, 1, stdout)) exit(3);
    {
      float l = -log2(img->width / roundf(SR / fps / 2));
      for (int s = 0; s < SR/fps; ++s) {
        for (int c = 0; c < 2; ++c) {
          float x = fmod(s / roundf(SR / fps / 2) + 0.5 * c, 1) * img->width;
          float y = y0 + dy * (s / roundf(SR / fps));
          float a = pixel2(img, l, y, x, 0) + pixel2(img, l, y, x, 1) + pixel2(↙
              ↳ img, l, y, x, 2);
          if (a == 3) a = 0;
          au[2 * s + c] = a / 3.0 - 0.5;
        }
      }
    }
    if (1 != fwrite(au, (int)roundf(SR/fps) * 2 * sizeof(float), 1, stderr)) ↙
        ↳ exit(6);
  }
  return 0;
}
```

24 .gitignore

```
emndl_automu
emndl_autotune
emndl_calculate
emndl_colourize
emndl_downscale
emndl_equalize
emndl_finalize
emndl_parse
emndl_ppmtoy4m
```

```
emndl_pretty
emndl_randomize
emndl_unwarp
*.hi
*.o
*.log
*/
```

25 GridScan.hs

```
{-
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
-}

module GridScan (gridEdge, gridShow, gridConverge, gridStep, gridScan, gridSpace
    ↪ ) where

import Control.Parallel.Strategies (parMap, rseq)
import Data.Array.IArray (Array, listArray, IArray(bounds), inRange, assocs, ↙
    ↪ elems, indices, (!))
import Data.Maybe (catMaybes, isJust, isNothing)
import Data.Number.MPFR (set, Precision, RoundMode(Near), sqr, getPrec, mulw)

import Number (I, R, C)
import Complex (Complex ((:+)), magnitude2)

data GridScan = GridScan I (Array (I,I) (Maybe C)) (I -> I -> C)

gridSize :: I
gridSize = 64

gridRadius :: R
gridRadius = 2

gridOffset :: I
gridOffset = gridSize `div` 2

gridScan :: C -> R -> GridScan
gridScan c0 r0 = GridScan 0 (listArray ((0,0),(gridSize - 1, gridSize - 1)) (↙
    ↪ replicate (gridSize * gridSize) (Just z))) f
  where
    s = gridSpace r0
    p = ceiling $ 5 - logBase 2 s
```

```
45      z = set Near p 0 :+ set Near p 0
        f i j =
          let x = s * (fromIntegral (i - gridOffset))
              y = s * (fromIntegral (j - gridOffset))
              re :+ im = c0 + (x :+ y)
50        in  set Near p re :+ set Near p im

    gridSpace :: R -> R
    gridSpace r0 = r0 * gridRadius / fromIntegral gridOffset

55  gridStep :: GridScan -> GridScan
    gridStep (GridScan n pixels coords) =
      let m = (5 * n `div` 4) `max` 1000
          d = m - n
          g _ _ o@Nothing = o
60        g k0 c@(r:+_) (Just z0) = go k0 z0
            where
              p = getPrec r
              go 0 z = z `seq` Just z
              go k z = z `seq` let z' = sqr' p z + c in if magnitude2 z' > 4 then ↙
                  ↳ Nothing else go (k - 1) z'
65        ps = parMap rseq (\((i,j),e) -> g d (coords i j) e) . assocs $ pixels
          pixels' = listArray (bounds pixels) ps
      in  GridScan m pixels' coords

    sqr' :: Precision -> C -> C
70  sqr' p (r:+i) =
      let r2 = sqr Near p r
          i2 = sqr Near p i
          ri = mulw Near p (r * i) 2
      in  (r2 - i2) :+ ri
75
    gridConverge :: [GridScan] -> GridScan
    gridConverge (GridScan _ p _ : gs@(g@(GridScan _ q _) : _))
      | fp == m = gridConverge gs
      | fp == f q = g
80    | otherwise = gridConverge gs
      where
        fp = f p
        f = length . catMaybes . elems
        m = gridSize * gridSize
85  gridConverge _ = error "GridScan.gridConverge: list too short"

    gridEdge :: GridScan -> [C]
    gridEdge (GridScan _ pixels coords) =
      let b = bounds pixels
90        edge (i, j) = isNothing (pixels ! (i, j)) && any isJust [pixels ! (i',j') ↙
              ↳ | i' <- [i-2 .. i+2], j' <- [j-2 .. j+2], inRange b (i',j')]
      in  map (uncurry coords) . filter edge . indices $ pixels

    gridShow :: GridScan -> String
    gridShow (GridScan _ a _) = unlines [ [ block (isJust (a ! (i,j))) (isJust (a ! ↙
      ↳ (i, j - 1))) | i <- [i0 .. i1] ] | j <- [j1, j1 - 2 .. j0] ]
95    where
        ((i0,j0),(i1,j1)) = bounds a
        block False False = ' '
        block True False = '\x2580'
```

```
    block False True = '\x2584'
    block True True = '\x2588'
```

26 Makefile

```
#
#    emndl -- exponentially transformed Mandelbrot Set renderer
#    Copyright (C) 2011,2018  Claude Heiland-Allen <claude@mathr.co.uk>
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>.
#

# standard paths/tools
prefix ?= $(HOME)/opt
exec_prefix = $(prefix)
data_prefix = $(prefix)
bindir = $(exec_prefix)/bin
datadir = $(data_prefix)/share/emndl
INSTALL = install
INSTALL_PROGRAM = $(INSTALL)
INSTALL_DATA = $(INSTALL) -m 644

# tools
CC = gcc
CXX = g++
GHC = ghc

# flags
OFLAGS = -O3 -march=native -Wall -pedantic -Wextra -Wno-unused-parameter -↙
    ↳ D_FILE_OFFSET_BITS=64 -pthread -fopenmp
CFLAGS = -std=c99
CXXFLAGS =
GHCFLAGS = -O3 -Wall -threaded -rtsopts -fspec-constr-count=50
LIBS = -lm -lqd -lmpfr -lgmp
EXES = emndl_automu emndl_colourize emndl_downscale emndl_equalize ↙
    ↳ emndl_finalize emndl_calculate emndl_parse emndl_ppmtoy4m emndl_pretty ↙
    ↳ emndl_randomize emndl_unwarp
OBJS = emndl_autotune emndl_autotune.hi emndl_autotune.o Complex.hi Complex.o ↙
    ↳ GridScan.hi GridScan.o MuAtom.hi MuAtom.o Number.hi Number.o Roots.hi ↙
    ↳ Roots.o
DATAS = emndl_audio.pd emndl_comb~.pd emndl_compress~.pd emndl_filter.pd ↙
    ↳ emndl_loader.pd

# build
all: $(EXES)
```

```
# clean
clean:
	-rm -f $(EXES) $(OBJS)

# install
install:
	$(INSTALL) -d $(DESTDIR)$(bindir)
	$(MAKE) install-bin
	$(MAKE) install-sh
	$(MAKE) install-data

install-strip:
	$(INSTALL) -d $(DESTDIR)$(bindir)
	$(MAKE) INSTALL_PROGRAM='$(INSTALL_PROGRAM) -s' install-bin
	$(MAKE) install-sh
	$(MAKE) install-data

install-bin:
	$(INSTALL_PROGRAM) $(EXES) $(DESTDIR)$(bindir)

install-sh:
	$(INSTALL_PROGRAM) emndl.sh $(DESTDIR)$(bindir)

install-data:
	$(INSTALL) -d $(DESTDIR)$(datadir)
	$(INSTALL_DATA) $(DATAS) $(DESTDIR)$(datadir)

# Haskell executables
emndl_autotune: emndl_autotune.hs Complex.hs GridScan.hs MuAtom.hs Number.hs ↙
    ↳ Roots.hs
	$(GHC) $(GHCFLAGS) --make emndl_autotune.hs

# C executables
%: %.c
	$(CC) $(CFLAGS) $(OFLAGS) -o $@ $< $(LIBS)

# C++ executables
%: %.cc
	$(CXX) $(CXXFLAGS) $(OFLAGS) -o $@ $< $(LIBS)

# dependencies
emndl_calculate: emndl_calculate.cc mp_real.h

# meta
.SUFFIXES:
.PHONY: all clean install install-strip install-bin install-sh
```

27 mp_real.h

```
#ifndef MP_REAL_H
#define MP_REAL_H 1

#include <algorithm>
#include <cmath>
#include <qd/dd_real.h>
#include <qd/qd_real.h>
#include <mpfr.h>
```

```
#define MPFR_RNDN GMP_RNDN

using namespace std;

class mp_real;
inline mp_real operator-(mp_real const &x, double const &y);

class mp_real {
public:
  mpfr_t m;
  mp_real() {
    mpfr_init2(m, 53);
    mpfr_set_d(m, 0, MPFR_RNDN);
  }
  mp_real(const double &s) {
    mpfr_init2(m, 53);
    mpfr_set_d(m, s, MPFR_RNDN);
  }
  mp_real(const int32_t &s) {
    mpfr_init2(m, 53);
    mpfr_set_si(m, s, MPFR_RNDN);
  }
  mp_real(const mp_real &s) {
    mpfr_init2(m, mpfr_get_prec(s.m));
    mpfr_set(m, s.m, MPFR_RNDN);
  }
  mp_real(mpfr_prec_t p) {
    mpfr_init2(m, p);
  }
  mp_real(mpfr_t s, mpfr_prec_t p) {
    mpfr_init2(m, p);
    mpfr_set(m, s, MPFR_RNDN);
  }
  mp_real(const char *s, mpfr_prec_t p) {
    mpfr_init2(m, p);
    mpfr_set_str(m, s, 10, MPFR_RNDN);
  }
  mp_real(double s, mpfr_prec_t p) {
    mpfr_init2(m, p);
    mpfr_set_d(m, s, MPFR_RNDN);
  }
  mp_real(const mp_real &s, mpfr_prec_t p) {
    mpfr_init2(m, p);
    mpfr_set(m, s.m, MPFR_RNDN);
  }
  mp_real(dd_real s) {
    mp_real a(s.x[0], 108);
    mp_real b(s.x[1], 108);
    mpfr_init2(m, 108);
    mpfr_add(m, a.m, b.m, MPFR_RNDN);
  }
  mp_real(qd_real s) {
    mp_real a(s[0], 216);
    mp_real b(s[1], 216);
    mp_real c(s[2], 216);
    mp_real d(s[3], 216);
    mpfr_init2(m, 216);
```

```
    mpfr_add(m, a.m, b.m, MPFR_RNDN);
    mpfr_add(m,   m, c.m, MPFR_RNDN);
    mpfr_add(m,   m, d.m, MPFR_RNDN);
  }

  ~mp_real() {
    mpfr_clear(m);
  }
  operator float() const {
    return mpfr_get_d(m, MPFR_RNDN);
  }
  operator double() const {
    return mpfr_get_d(m, MPFR_RNDN);
  }
  operator long double() const {
    return mpfr_get_ld(m, MPFR_RNDN);
  }
  operator dd_real() const {
    double  a = *this;
    mp_real n = *this - a;
    double  b = n;
    return  dd_real(a, b);
  }
  operator qd_real() const {
    double  a = *this;
    mp_real n = *this - a;
    double  b = n;
    mp_real o = n - b;
    double  c = o;
    mp_real p = o - c;
    double  d = p;
    return  qd_real(a, b, c, d);
  }
#define maxprec(x, y) max(mpfr_get_prec(x.m), mpfr_get_prec(y.m))
  inline mp_real& operator=(mp_real const &y) {
    mpfr_set_prec(m, maxprec((*this), y));
    mpfr_set(m, y.m, MPFR_RNDN);
    return *this;
  }
};

inline bool operator<(mp_real const &x, mp_real const &y) {
  return mpfr_less_p(x.m, y.m);
}

inline mp_real operator+(mp_real const &x, mp_real const &y) {
  mp_real r(maxprec(x, y));
  mpfr_add(r.m, x.m, y.m, MPFR_RNDN);
  return r;
}

inline mp_real operator+(mp_real const &x, double const &y) {
  mp_real r(max(mpfr_get_prec(x.m), mpfr_prec_t(53)));
  mpfr_add_d(r.m, x.m, y, MPFR_RNDN);
  return r;
}
```

```
inline mp_real operator-(mp_real const &x, mp_real const &y) {
  mp_real r(maxprec(x, y));
  mpfr_sub(r.m, x.m, y.m, MPFR_RNDN);
  return r;
}

inline mp_real operator-(mp_real const &x, double const &y) {
  mp_real r(max(mpfr_get_prec(x.m), mpfr_prec_t(53)));
  mpfr_sub_d(r.m, x.m, y, MPFR_RNDN);
  return r;
}

inline mp_real operator*(mp_real const &x, mp_real const &y) {
  mp_real r(maxprec(x, y));
  mpfr_mul(r.m, x.m, y.m, MPFR_RNDN);
  return r;
}

inline mp_real operator*(double const &x, mp_real const &y) {
  mp_real r(max(mpfr_prec_t(53), mpfr_get_prec(y.m)));
  mpfr_mul_d(r.m, y.m, x, MPFR_RNDN);
  return r;
}

inline mp_real operator*(mp_real const &x, double const &y) {
  mp_real r(max(mpfr_get_prec(x.m), mpfr_prec_t(53)));
  mpfr_mul_d(r.m, x.m, y, MPFR_RNDN);
  return r;
}

inline mp_real operator/(mp_real const &x, mp_real const &y) {
  mp_real r(maxprec(x, y));
  mpfr_div(r.m, x.m, y.m, MPFR_RNDN);
  return r;
}

inline mp_real sqr(mp_real const &x) {
  mp_real r(mpfr_get_prec(x.m));
  mpfr_sqr(r.m, x.m, MPFR_RNDN);
  return r;
}

inline mp_real sqrt(mp_real const &x) {
  mp_real r(mpfr_get_prec(x.m));
  mpfr_sqrt(r.m, x.m, MPFR_RNDN);
  return r;
}

inline mp_real log(mp_real const &x) {
  mp_real r(mpfr_get_prec(x.m));
  mpfr_log(r.m, x.m, MPFR_RNDN);
  return r;
}

inline mp_real atan2(mp_real const &y, mp_real const &x) {
  mp_real r(maxprec(x, y));
  mpfr_atan2(r.m, y.m, x.m, MPFR_RNDN);
```

```
  return r;
}

inline mp_real pow(mp_real const &y, mp_real const &x) {
  mp_real r(maxprec(x, y));
  mpfr_pow(r.m, y.m, x.m, MPFR_RNDN);
  return r;
}

inline float to_float(mp_real const &x) {
  return mpfr_get_d(x.m, MPFR_RNDN);
}

#undef maxprec

#endif
```

28 MuAtom.hs

```
{-# LANGUAGE BangPatterns, RecordWildCards, Rank2Types, FlexibleContexts #-}

{-
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011,2018  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
-}

module MuAtom (refineNucleus) where

import Data.List (genericIndex, genericSplitAt)
import Data.Vec (NearZero())

import Complex (Complex((:+)), cis, magnitude)

import Roots (root2, root4, auto, F)

-- finding bond points

fdf :: (Integral i, Num c) => i -> c -> c -> (c, c)
fdf !n !z !c = let (fzs, fz:_) = genericSplitAt n $ iterate (\w -> w * w + c) z
               in  (fz, 2 ^ n * product fzs)  -- [ f i z c | i <- [0 .. n - 1] ]

bondIter :: (Ord r, Floating r) => Integer -> Complex r -> F r
bondIter !n !(br:+bi) [x0, x1, x2, x3] =
  let !z = x0:+x1
```

```
      !c = x2:+x3
      !b = auto br :+ auto bi
      (!fz, !dfz) = fdf n z c
      !(y0 :+ y1) =  fz - z -- f n z c - z
      !(y2 :+ y3) = dfz - b -- df n z c - (lift br :+ lift bi)
  in  [y0, y1, y2, y3]
bondIter _ _ _ = error "MuAtom.bondIter: internal error"

-- finding nucleus

l :: (Integral i, Num c) => i -> c -> c
l !n !c = (`genericIndex` n) . iterate (\z -> z * z + c) $ 0

nucleusIter :: Floating r => Integer -> F r
nucleusIter !n [x0, x1] =
  let !c = x0 :+ x1
      !(y0 :+ y1) = l n c
  in  [y0, y1]
nucleusIter _ _ = error "MuAtom.nucleusIter: internal error"

refineNucleus :: (NearZero r, Floating r, Ord r) => Integer -> Complex r -> (r, r, r)
refineNucleus p (gr :+ gi) =
  let [cr, ci] = root2 (nucleusIter p) [gr, gi]
      [_, _, b0r, b0i] = root4 (bondIter p (cis 0)) [cr, ci, cr, ci]
      [_, _, b1r, b1i] = root4 (bondIter p (cis pi)) [cr, ci, cr, ci]
      bond0 = b0r :+ b0i
      bond1 = b1r :+ b1i
      r = magnitude (bond1 - bond0)
  in  (cr, ci, r)
```

29 NEWS

```
2011-06-03  v0.2  soon superceded
2011-04-22  v0.1  first release
2011-02-02        it begins
```

30 Number.hs

```
{-
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
-}
```

```
module Number (I, N, Q, R, C) where

import Data.Vec (NearZero(..))
import Data.Number.MPFR (MPFR, getPrec, int2w, RoundMode(Up))
import Data.Number.MPFR.Instances.Near ()
{-
import System.Random (Random(..))
import Data.Ratio ((%))
-}

import Complex (Complex())

type I = Int
type N = Integer
type Q = Rational
type R = MPFR
type C = Complex R

instance NearZero MPFR where
  nearZero x = let p = getPrec x in not (abs x > int2w Up p 1 (4 - fromIntegral p))

{-
instance Random MPFR where
  random g0 = let k = 2 ^ (300 :: Int) :: Integer
                  (m, g1) = randomR (0, k) g0
              in  (fromRational (m % k), g1)
  randomR (lo, hi) g0 = let (r, g1) = random g0
                        in  (lo + (hi - lo) * r, g1)
-}
```

31 README

```
emndl git (2018-07-18)                    ------
 ------------    ---    --------  --------/  /  /
/   ---_-/    ,,,    \/    ---    /    ---     /  /-,
\------/---/---/---/---/  /---\---------/------)

about
=====

emndl renders the Mandelbrot Set transformed through an [1] exponential
coordinate transform.

[1] http://mrob.com/pub/muency/exponentialmap.html

quick start
===========

prerequisites
-------------

This is not a complete list, but if I missed something let me know.

gcc, g++, libqd-dev       -- for building most programs
```

```
ghc, cabal                -- for building emndl_autotune
bash, getopt (GNU), bc    -- for running
imagemagick               -- for image output
y4mscaler, mpeg2enc       -- for dvd output
ffmpeg                    -- for mkv outpout
ffmpeg2theora             -- for ogv output
ecasound, puredata        -- for video soundtracks


compiling
---------

'make' will build everything apart from 'emndl_autotune'.

'make emndl_autotune' or 'cabal install' will build 'emndl_autotune',
but see below for further information.


installing
----------

After compiling, 'make install prefix=~/opt' will install the programs
(excepting 'emndl_autotune') and the front end script into '~/opt/bin',
with the puredata sonification patches in '~/opt/share/emndl'.  Make
sure to install 'emndl_autotune' in the same prefix as the rest of emndl
so that the script can find it, or create a symlink to where 'cabal'
installed it.


running
-------

A directory is created for output, as well as a neighbouring log file,
both in the current working directory.  So, 'cd' somewhere you don't
mind putting these files, which may be relatively large.

For more information:
$ emndl.sh --help

A simple example:
$ emndl.sh --quality 7 --png --jpeg --depth 6 \
  --re -1.241733127596417466318604 --im -0.16989658410283839228793 27

If you have emndl_autotune compiled:
$ emndl.sh --auto 96 --png --jpeg

If you have some days to spare:
$ emndl.sh --quality 10 --depth 21 --png --jpeg --dvd --ogv \
  --re -0.59885296584604451137005579991698735726381067664671185174 77 \
  --im -0.66278734189819746839198562872793650424236039843380940070 74


emndl programs
==============

emndl consists of a number of programs, plus a more user-friendly script
to drive the process from start to finish (described above).
```

```
emndl_autotune
--------------

emndl_autotune attempts to find interesting points in the Mandelbrot Set
that might look nice. It takes one optional argument, being a positive
integer specifying how many bits of precision to use (the default is 128
bits). Progress (including Unicode block graphics) is printed on stderr
with the final result on stdout for use by emndl.sh (or other programs).

Use it like 'emndl_autotune +RTS -N -RTS 80'.

emndl_autotune is implemented in Haskell, so far requiring a GHC built
with integer-simple (see the [2] hmpfr documentation on Hackage for the
rationale behind this awkwardness). There is a .cabal file which can be
used by the cabal-install tool to automatically install the required
dependencies and compile emndl_autotune itself, run: 'cabal install'.
If you have the dependencies already, run: 'make emndl_autotune'.

[2] http://hackage.haskell.org/package/hmpfr


emndl_calculate
---------------

emndl_calculate does the fractal iteration calculations. Its arguments
are output raw data file name, iteration state raw data file name (for
future resumation of interrupted calculations), image width, outer
radius, and finally number of threads. The coordinates of the center
point are passed through standard input in raw 'quad-double' format.

Use it like 'emndl_calculate out.emndl state.emndl 256 1024 2 < coords.qd'.


emndl_colourize
---------------

emndl_colourize takes a raw 'float' data file containing the relative
distance estimate for each pixel (the distance estimate divided by the
pixel spacing at that point). It outputs a raw 'rgb' data file with the
colour for each pixel.

Use it like 'emndl_colourize in.f out.raw'.


emndl_downscale
---------------

emndl_downscale reduces the size of a raw 'float' image by a factor of 2
using a scaled geometric mean to preserve relative distance estimates.
It takes one argument, the width of the input image (which must be even,
and so must be its height).

Use it like 'emndl_downscale 128 < in.f > out.f'.
```

```
emndl_equalize
---------------

emndl_equalize is currently unused and undocumented.  FIXME


emndl_finalize
---------------

emndl_finalize extracts the raw 'float' image data from an iteration
file.

Use it like 'emndl_finalize in.emndl out.f'.


emndl_parse
-----------

emndl_parse takes two arguments specifying the real and imaginary parts
of a complex number.  These are output on stdout in raw 'quad-double'
format.

Use it like 'emndl_parse -1.75 0.001 > coords.qd'.


emndl_ppmtoy4m
---------------

emndl_ppmtoy4m takes a PPM stream on stdin and outputs a Y4M stream on
stdout.  It requires two arguments, being the width and height of the
PPM stream.  Error checking is minimal, but it's significantly faster
than 'ppmtoy4m' from 'mjpegtools'.

Use it like 'emndl_ppmtoy4m 640 480 < in.ppm > out.y4m'.


emndl_pretty
------------

emndl_pretty takes raw 'quad-double' on stdin and outputs human-readable
numbers.

Use it like 'emndl_pretty < coords.qd'.


emndl_randomize
---------------

emndl_randomize is currently unused and undocumented.  FIXME


emndl_unwarp
------------

emndl_unwarp takes a series of downscaled and colourized exponential
strip images, and reverses the coordinate transform to turn them into
zooming videos.  It takes a file name stem as first argument, which has
```

```
'.00.ppm' appended to get the highest resolution strip image file name,
its dimensions (width must be a power of two, height must be a multiple
of width) determine the number of images that will be loaded.  The next
two arguments determine the size of the output PPM stream, and the final
fourth argument is the speed in pixels per frame.  PPM stream data is
sent to stdout - you will want to redirect it or (better, as it is
likely to be quite large) pipe it to an encoder, likewise raw stereo
32bit float audio data is sent to stderr.

Use it like 'emndl_unwarp emndl 320 200 3.21 > out.ppm 2> out.raw'.


author
======

Claude Heiland-Allen
claude@mathr.co.uk
https://mathr.co.uk

bureaucracy
===========

emndl -- exponentially transformed Mandelbrot Set renderer
Copyright (C) 2011,2018  Claude Heiland-Allen <claude@mathr.co.uk>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

32 Roots.hs

```
{-# LANGUAGE BangPatterns, Rank2Types, ScopedTypeVariables, FlexibleContexts #-}

{-
    emndl -- exponentially transformed Mandelbrot Set renderer
    Copyright (C) 2011,2018  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
```

```
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
-}

module Roots (root2, root4, F, auto) where

import Prelude hiding (zipWith)
import Data.Functor ((<$>))
import Data.Vec (toList, solve, fromList, matFromLists, zipWith, Vec2, Mat22, ↙
    ↳ Vec4, Mat44, NearZero(nearZero))
import Numeric.AD (jacobian', auto)
import Numeric.AD.Mode.Reverse (Reverse)
import Numeric.AD.Internal.Reverse (Tape)
import Data.Reflection (Reifies)

type F r = forall s . Reifies s Tape => [Reverse s r] -> [Reverse s r]

root2' :: forall r . (Fractional r, NearZero r, Ord r) => F r -> Vec2 r -> Vec2 ↙
    ↳ r
root2' f !x = go x
  where
    jf = jacobian' f :: [r] -> [(r, [r])]
    go x0 =
      let (ys, js) = unzip $ jf (toList x0)
          y = fromList (negate <$> ys) :: Vec2 r
          j = matFromLists js :: Mat22 r
          mdx = solve j y
      in  if all nearZero ys
            then  x0
            else  case mdx of
              Nothing ->  x0
              Just dx ->
                let x1 = zipWith (+) x0 dx
                in  if all nearZero (toList dx)
                      then x1
                      else go x1

root2 :: (Fractional r, NearZero r, Ord r) => F r -> [r] -> [r]
root2 f = toList . root2' f . fromList

root4' :: forall r . (Fractional r, NearZero r, Ord r) => F r -> Vec4 r -> Vec4 ↙
    ↳ r
root4' f !x = go x
  where
    jf = jacobian' f
    go x0 =
      let (ys, js) = unzip $ jf (toList x0)
          y = fromList (negate <$> ys) :: Vec4 r
          j = matFromLists js :: Mat44 r
          mdx = solve j y
      in  if all nearZero ys
            then  x0
            else  case mdx of
              Nothing ->  x0
              Just dx ->
                let x1 = zipWith (+) x0 dx
                in  if all nearZero (toList dx)
```

```
                    then x1
                    else go x1

root4 :: (Fractional r, NearZero r, Ord r) => F r -> [r] -> [r]
root4 f = toList . root4' f . fromList
```

33 Setup.hs

```
import Distribution.Simple
main = defaultMain
```

34 THANKS

Thanks are owed to:

Robert Munafo
Wolf Jung
James Morris
Roman Haefeli
Mathieu Bouchard
knighty
gerrit
pauldelbrot

35 TODO

OpenGL/GLSL unwarp

```
// unwarping (todo: cubic interpolation)
const char *unwarp_frag_src =
"uniform sampler2D tex;\n"
"uniform float radius;\n"
"uniform float offset;\n"
"\n"
"void main() {\n"
"  vec2 q = gl_TexCoord[0].xy;\n"
"  float a = atan(q.y, q.x) * 0.15915494309189535;\n"
"  float d = max(sqrt(dot(q,q)), 0.000000001);\n"
"  float r = log(d) * radius;\n"
"  vec2 p = vec2(frac(a), clamp(r + offset, 0.0, 1.0));\n"
"  float l = -log2(d);\n"
"  gl_FragColor = texture2DLod(tex, p, l);\n"
"}\n";
// main program
logradius = 1.0 / log(pow(0.5, 8.0 / IWIDTH));
```

Y4M tidbits

```
y4m = "YUV4MPEG2 W720 H576 F25:1 Ip A118:81 C420mpeg2\n"; -- PAL 16:9
y4m = "YUV4MPEG2 W720 H576 F25:1 Ip A59:54 C420mpeg2\n";  -- PAL  4:3
ffmpeg -f yuv4mpegpipe -b 5625000 -r 25 -i - out.mkv # 1920x1080
```

DVD Author

```
dvdauthor -o dvd/ -x dvd.xml
k3b
growisofs -dvd-compat -Z/dev/dvd=dvd.iso
```

Miscellany

```
emndl_unwarp_gl for speed when GLSL is available
emndl.sh option to use unwarp_gl (or autodetect?)
emndl_colourize speedup
emndl_colourize configuration
```