rdex-client

Claude Heiland-Allen

2008–2019

Contents

1 ARCHITECTURE.dot

```
digraph G {
  // textures
  node [shape=box,style=filled,fillcolor=lightgrey];
  balls;
  balls1024;
  balls1024h;
  balls512;
  balls512h;
  balls256;
  balls256h;
  balls128;
  balls128h;
  balls64;
  balls64h;
  balls32;
  glowingBalls;
  // shaders
  node [shape=parallelogram,style=filled,fillcolor=lightblue];
  highpass;
  blurH1;
  blurH2;
  blurH3;
  blurH4;
  blurH5;
  blurV1;
  blurV2;
  blurV3;
  blurV4;
```

```
    blurV5;
    wsum;
    // edges
    balls -> highpass ->
    balls1024 -> blurH1 -> balls1024h -> blurV1 ->
    balls512  -> blurH2 -> balls512h  -> blurV2 ->
    balls256  -> blurH3 -> balls256h  -> blurV3 ->
    balls128  -> blurH4 -> balls128h  -> blurV4 ->
    balls64   -> blurH5 -> balls64h   -> blurV5 ->
    balls32;
    balls1024 -> wsum;
    balls512  -> wsum;
    balls256  -> wsum;
    balls128  -> wsum;
    balls64   -> wsum;
    balls32   -> wsum;
    wsum -> glowingBalls;
};
```

2 COPYING

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

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>
>
> 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/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author>
> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

```
  You should also get your employer (if you work as a programmer) or school,
665 if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
670 into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

3 dvd.xml

```
<dvdauthor>
<vmgm/>
<titleset><menus/><titles>
<pgc>
5 <vob file="hour.mpeg" chapters⇙
    ↳ ="00:00,05:00,10:00,15:00,20:00,25:00,30:00,35:00,40:00,45:00,50:00,55:00,1:00:00"⇙
    ↳ />
<post>jump chapter 1;</post>
</pgc>
</titles></titleset>
</dvdauthor>
```

4 .gitignore

```
*~
*.o
*.frag.c
*.vert.c
5 session
src/cursor.bin
src/cursor.c
src/interleave31
src/interleave31_1024
10 src/rdex
*.mpeg
*.ogv
*.m2v
*.mp2
15 *.mkv
*.mp4
*.raw
*.wav
-/
```

5 RDEX.desktop

```
#!/usr/bin/env xdg-open

[Desktop Entry]
Encoding=UTF-8
5 Version=1.0
Type=Application
```

```
Exec=/home/claude/code/rdex/client/start.sh
Name=RDEX
Comment=Reaction Diffusion Explorer
Icon=/home/claude/code/rdex/client/RDEX.png
```

6 rdex.jr

(application/gzip; charset=binary)

7 RDEX.png

8 README

```
rdex -- reaction-diffusion explorer
==========================================

What Is This All About?
-----------------------

A kind of continuous non-linear cellular automaton, based on solution of
partial differential equations representing chemistry of two reagents
involving reaction and diffusion:

    dU/dt = ru * L(U) - U*V*V + f * (1 - U)
    dV/dt = rv * L(V) + U*V*V - (f + k) * V

where:

    L(x) = Laplacian operator
    ru   = diffusion parameter for U
    rv   = diffusion parameter for V
    f    = some coupling parameter
    l    = another coupling parameter
```

```
This gives a 4D parameter space:

    p = (ru, rv, f, k)

What rdex does is to explore this parameter space to try and find the
"interesting" values.  It labels each point (in 4D space) as follows:

    uniform => the reaction decays into uninteresting flatness
    stable  => the reaction decays into a static pattern
    dynamic => the reaction seems to have an ever changing pattern
    erratic => the numerical process blew up

rdex explores the space completely at random, but spends more time
on more "interesting" points in the parameter space.


Viewing 4D
----------

In 3D homogeneous coordinates:

    M_proj = | 2n/(r-l)   0       (r+l)/(r-l)       0       |
             |   0      2n/(t-b) (t+b)/(t-b)       0       |
             |   0        0      (n+f)/(n-f) (2nf)/(n-f)  |
             |   0        0          -1            0       |
        where
            l = left     r = right
            t = top      b = bottom
            n = near     f = far

    P_obj  = ((original point in object space).xyz, 1)

    P_clip = M_proj * M_view * M_model * P_obj

    P_view = P_clip.xyz / P_clip.w

    draw when -1 < P_view.c < 1 for all c in xyz

In 4D homogeneous coordinates:

    M_proj = | 2n/(r-l)   0         0       (r+l)/(r-l)       0       |
             |   0      2n/(t-b)    0       (t+b)/(t-b)       0       |
             |   0        0       2n/(i-j) (i+j)/(i-j)       0       |
             |   0        0         0      (n+f)/(n-f) (2nf)/(n-f)  |
             |   0        0         0          -1            0       |
        where
            l = left     r = right
            t = top      b = bottom
            i = above    j = below
            n = near     f = far

    P_obj  = ((original point in object space).xyzw, 1)

    P_clip = M_proj * M_view * M_model * P_obj

    P_view = P_clip.xyzw / P_clip.h
```

```
    draw when -1 < P_view.c < 1 for all c in xyzw
80
Hardware Requirements
---------------------

85  * Faster GPU preferred
  * GLSL fragment shader support


Software Requirements
90  ---------------------

  * Linux (maybe works on other OS's too)
  * C compiler (gcc), bash shell, sed
  * OpenGL development files: GL/GLU/GLUT/GLEW
95  * X11 development files: Xpm
  * CURL development files
  * Accelerated 3D drivers


100 Usage
-----

Warning: rdex visual output can consist of rapidly strobing bright
colour patterns, if you are sensitive to them then be careful or don't
105 run rdex at all.

  $ make -C src                                          # build the program
  $ export RDEX_SESSION="/path/to/folder"                # where to store data
  $ export RDEX_UPLOAD="http://host/to/upload/to" # if upload desired
110 $ unset RDEX_UPLOAD                                  # if no upload desired
  $ ./src/rdex                                           # run the program

See rdex-server documentation for the correct value for RDEX_UPLOAD.

115
Tested Machines
---------------

Machine #1
120   * Intel(R) Pentium(R) M processor 1.60GHz
  * ATI Technologies Inc RV350 [Mobility Radeon 9600 M10]
  * GNU/Linux | Debian/Lenny | puredyne
  * "evilblob" fglrx driver supporting OpenGL 2.1.7769
  * gcc (Debian 4.3.2-1) 4.3.2
125
Machine #2
  * Intel(R) Core(TM)2 Duo CPU P7550 2.26GHz
  * nVidia Corporation G98M [GeForce G 105M] (rev a1)
  * GNU/Linux | Ubuntu/Karmic prerelease
130 * "evilblob" nvidia driver supporting OpenGL 3.2.0
  * gcc version 4.4.1 (Ubuntu 4.4.1-4ubuntu8)

Test reports (whether succesful or not) are very welcome!
```

```
Benchmarks
----------

rdex-client-0.beta on machine #2

--8<--
    360172 seconds elapsed
    17145469 frames rendered (47.603559 fps)
    74388 species analyzed (0.206535 sps)
         88.78% uniform
          2.80% stable
          0.87% dynamic
          7.55% erratic
--8<--
real    6002m53.508s
user    795m50.752s
sys     66m12.448s
--8<--
Note: timing may be off by an hour due to daylight savings time change

rdex-client-0.svn1543 on machine #2

--8<--
     14912 seconds elapsed (+/- 1)
    882935 frames rendered (59.209697 fps)
     11785 frames dropped (+/- 60)
      2996 species analyzed (0.200912 sps)
         89.72% uniform
          1.84% stable
          1.90% dynamic
          6.54% erratic
--8<--


Legal
-----

rdex -- reaction-diffusion explorer
Copyright (C) 2008-2017  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/>.


rdex contains portions of code derived from FIRE:
```

```
FIRE -- Flexible Image Retrieval System

FIRE 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.

FIRE 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 FIRE; if not, write to the Free Software Foundation, Inc.,
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA


Credits
-------

Thanks to Laruft for finding two bugs and testing!


References
----------

Pedro F. Felzenszwalb, Daniel P. Huttenlocher.
Efficient Graph-Based Image Segmentation.
International Journal of Computer Vision, Volume 59, Number 2, pp. 167-181, ↙
    ↳ September 2004, Springer.
http://www.cs.wisc.edu/~dyer/ai-qual/felzenszwalb-ijcv04.pdf

Greg Pass, Justin Miller, Ramin Zabih.
Comparing Images Using Color Coherence Vectors.
MULTIMEDIA '96: Proceedings of the fourth ACM international conference on ↙
    ↳ Multimedia, pp. 65-73, 1996, ACM
http://www.labiem.cpgei.cefetpr.br/Members/humberto/disciplinas/pdi/papers/↙
    ↳ comparingimagesusingcolor.pdf

Deselaers, T., Keysers, D., Ney, H.
Features for Image Retrieval: An Experimental Comparison.
Information Retrieval, vol. 11, issue 2, The Netherlands, Springer, pp. 77-107, ↙
    ↳ 03/2008.
http://thomas.deselaers.de/system/files/cbir.pdf

Jörg Walter, Daniel Wessling, Kai Essig, Helge Ritter.
Interactive hyperbolic image browsing - towards an integrated multimedia ↙
    ↳ navigator.
ACM MDM/KDD Multimedia Data Mining and Conf Knowledge Discovery and Data Mining,↙
    ↳  Philadelphia, USA, August 2006.
https://www.techfak.uni-bielefeld.de/~walter/pub/WalterWessEssRit-mdm06kdd.pdf
```

9 setup.sh

```
#!/bin/bash
chgrp audio /dev/hpet
```

```
for i in 0 1
do
  echo "performance" > "/sys/devices/system/cpu/cpu${i}/cpufreq/scaling_governor"
done
cat "/data/cm/rdex/session/"* > /dev/null
```

10 src/arithmeticmean.c

```
/* ================================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------------
Arithmetic Mean Shader
================================================================================ */

#include <math.h>
#include "util.h"
#include "arithmeticmean.h"
#include "arithmeticmean.frag.c"

//==============================================================================
// arithmetic mean shader initialization
struct arithmeticmean *arithmeticmean_init(
  struct arithmeticmean *arithmeticmean
) {
  if (! arithmeticmean) { return 0; }
  if (! shader_init(
          &arithmeticmean->shader, 0, arithmeticmean_frag
     )) {
    return 0;
  }
  shader_uniform(arithmeticmean, texture);
  shader_uniform(arithmeticmean, dx);
  shader_uniform(arithmeticmean, dy);
  arithmeticmean->value.texture = 0;
  arithmeticmean->value.dx = 0;
  arithmeticmean->value.dy = 0;
  glGenTextures(arithmeticmean_tex_max, arithmeticmean->textures);
  glEnable(GL_TEXTURE_2D);
  for (int i = 0; i < arithmeticmean_tex_max; ++i) {
    glBindTexture(GL_TEXTURE_2D, arithmeticmean->textures[i]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  }
  glDisable(GL_TEXTURE_2D);
  arithmeticmean->result[0] = 0;
  arithmeticmean->result[1] = 0;
  arithmeticmean->result[2] = 0;
  arithmeticmean->result[3] = 0;
  arithmeticmean->stddev = 0;
  return arithmeticmean;
}

//==============================================================================
// arithmetic mean shader reshape callback
void arithmeticmean_reshape(
```

```
  struct arithmeticmean *arithmeticmean, int w, int h
) {
  int w2 = roundtwo(w);
  int h2 = roundtwo(h);
  int d = w2>h2 ? w2 : h2;
  arithmeticmean->count = logtwo(d);
  glEnable(GL_TEXTURE_2D);
  for (int i = 0; i < arithmeticmean->count; ++i) {
    glBindTexture(GL_TEXTURE_2D, arithmeticmean->textures[i]);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
      1<<i, 1<<i, 0, GL_RGBA, GL_FLOAT, 0
    );
  }
  glDisable(GL_TEXTURE_2D);
}

//=============================================================
// arithmetic mean display callback
void arithmeticmean_display(
  struct arithmeticmean *arithmeticmean, GLuint fbo, GLuint texture
) {
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, texture);
//  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  glUseProgramObjectARB(arithmeticmean->shader.program);
  for (int i = arithmeticmean->count - 1; i >= 0; --i) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, 1<<i, 1<<i);
    glFramebufferTexture2DEXT(
      GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
      arithmeticmean->textures[i], 0
    );
    arithmeticmean->value.dx = 0.5 / (1 << i);
    arithmeticmean->value.dy = 0.5 / (1 << i);
    shader_updatei(arithmeticmean, texture);
    shader_updatef(arithmeticmean, dx);
    shader_updatef(arithmeticmean, dy);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glBindTexture(GL_TEXTURE_2D, arithmeticmean->textures[i]);
  }
  glGetTexImage(
    GL_TEXTURE_2D, 0, GL_RGBA, GL_FLOAT, &arithmeticmean->result
  );
  glUseProgramObjectARB(0);
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  glDisable(GL_TEXTURE_2D);
}

```

```
//=============================================================================
// arithmetic mean idle callback
void arithmeticmean_idle(struct arithmeticmean *arithmeticmean) {
  arithmeticmean->stddev =
    sqrt(arithmeticmean->result[1]
       - arithmeticmean->result[0] * arithmeticmean->result[0]) +
    sqrt(arithmeticmean->result[3]
       - arithmeticmean->result[2] * arithmeticmean->result[2]);
  if (isnan(arithmeticmean->stddev)) { arithmeticmean->stddev = 0; }
}

// EOF
```

11 src/arithmeticmean.frag

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
============================================================================ */

uniform sampler2D texture; // all four channels considered
uniform float dx;          // horizontal distance between source texels
uniform float dy;          // vertical distance between source texels

void main(void) {
  vec2 p = gl_TexCoord[0].st;
  vec4 x00 = texture2D(texture, p);                // box of 4 texels
  vec4 x01 = texture2D(texture, p + vec2(0.0,dy));
  vec4 x10 = texture2D(texture, p + vec2(dx,0.0));
  vec4 x11 = texture2D(texture, p + vec2(dx,dy));
  gl_FragColor = 0.25*(x00+x01+x10+x11);           // arithmetic mean
}
```

12 src/arithmeticmean.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Arithmetic Mean Shader
============================================================================ */

#ifndef ARITHMETIC_MEAN_H
#define ARITHMETIC_MEAN_H 1

#include "shader.h"

//=============================================================================
// maximum number of textures to allocate (16 => 65536x65536 max size)
#define arithmeticmean_tex_max 16

//=============================================================================
// arithmetic mean shader data
struct arithmeticmean { struct shader shader;
  struct { GLint texture; GLint dx; GLint dy; } uniform;
  struct { int   texture; float dx; float dy; } value;
```

```
  int count;
  GLuint textures[arithmeticmean_tex_max];
  GLfloat result[4];
  float stddev;
};

//============================================================================
// prototypes
struct arithmeticmean *arithmeticmean_init(
  struct arithmeticmean *arithmeticmean
);
void arithmeticmean_reshape(
  struct arithmeticmean *arithmeticmean, int w, int h
);
void arithmeticmean_display(
  struct arithmeticmean *arithmeticmean, GLuint fbo, GLuint texture
);
void arithmeticmean_idle(struct arithmeticmean *arithmeticmean);

#endif
```

13 src/audio.c

```
/* ===========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011,2017,2019  Claude Heiland-Allen <claude@mathr.co.uk>
---------------------------------------------------------------------------
Audio output
=========================================================================== */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <GL/glew.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/soundcard.h>
#include "audio.h"
#include "audio.frag.c"

volatile char audio_read  = AUDIO_BUFFERS / 2;
volatile char audio_write = 0;
volatile char audio_reset = 1;

void audio_compute(struct audio *audio, jack_default_audio_sample_t **out, ↙
    ↳ jack_nframes_t nframes) {
  /* copy buffers to JACK */
  for (int i = 0; i < nframes; ++i) {
    int r = audio_read;
    for (int c = 0; c < AUDIO_CHANNELS; ++c) {
      // DC blocker
      switch (audio->state) {
        case audio_state_prepare:
          audio->sample[c] = 0;
          audio->iphase[c] = 0.0;
```

```
            out[c][i] = 0.0;
            break;
          case audio_state_intro: {
            double hz = 1000;
            double sr = audio->sr ? audio->sr : 48000;
            double incr = 6.283185307179586 * hz / sr;
            audio->iphase[c] = fmod(audio->iphase[c] + incr, 6.283185307179586);
            out[c][i] = sin(audio->iphase[c]) / 10;
            audio->sample[c]++;
            break;
          }
          case audio_state_outro: {
            int loop = audio->size - 3.0 * audio->sphase;
            int j = audio->index + audio->sphase * 4.0 * (rand()/((double) RAND_MAX) - 0.5);
            j %= loop;
            if (j < 0) { j += loop; }
            jack_default_audio_sample_t xn = audio->buffer[r][c][j];
            audio->yn1[0][c] = xn - audio->xn1[0][c] + audio->r[0][c] * audio->yn1[0][c];
            audio->xn1[0][c] = xn;
            out[c][i] = audio->yn1[0][c];
            break;
          }
          case audio_state_idle: {
            jack_default_audio_sample_t xn = 0;
            audio->yn1[0][c] = xn - audio->xn1[0][c] + audio->r[0][c] * audio->yn1[0][c];
            audio->xn1[0][c] = xn;
            out[c][i] = 0;
            break;
          }
          case audio_state_rolling: {
            jack_default_audio_sample_t xn = audio->buffer[r][c][audio->index];
            audio->yn1[0][c] = xn - audio->xn1[0][c] + audio->r[0][c] * audio->yn1[0][c];
            audio->xn1[0][c] = xn;
            jack_default_audio_sample_t g = fmax(0, 1 - 1 / (1 + (audio->sphase - 2) / 10));
            out[c][i] = g * audio->yn1[0][c];
            break;
          }
        }
      }
      /* increment/wrap read counter, and swap buffers */
      ++audio->index;
      while (audio->index >= audio->size) {
        audio->index -= audio->size;
        int audio_read2 = (r + 1) % AUDIO_BUFFERS;
        // don't overtake write pointer
        if (audio_read2 == audio_write) {
          audio_read2 = (audio_read2 - 1 + AUDIO_BUFFERS) % AUDIO_BUFFERS;
        }
        audio_read = audio_read2;
      }
    }
    // update state
```

```
  audio->sphase += nframes / (double) audio->sr;
  switch (audio->state) {
    case audio_state_prepare:
      if (! (audio->sphase < 2)) {
        audio->state = audio_state_intro;
        audio->sphase = 0;
      }
      break;
    case audio_state_intro:
#if 0
      if (! (audio->sphase < 5)) {
        audio->state = audio_state_rolling;
        audio->sphase = 0;
      }
#endif
      audio->sphase = 0;
      break;
    case audio_state_outro:
      if (! (audio->sphase < 60)) {
        audio->state = audio_state_idle;
        audio->sphase = 0;
/*
        if (! audio->nrt) {
          jack_transport_stop(audio->client);
        }
*/
      }
      break;
    default:
      break;
  }
}

/* JACK audio processing callback */
static int audio_process(jack_nframes_t nframes, void *arg) {
  struct audio *audio = arg;
  /* get JACK buffers */
  jack_default_audio_sample_t *out[2];
  out[0] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(audio->port[0], nframes);
  out[1] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(audio->port[1], nframes);
  audio_compute(audio, out, nframes);
  return 0;
}

static void *audio_process_oss(void *arg) {
  struct audio *audio = arg;
  const char *device = "/dev/dsp1";
  int audio_fd;
  if ((audio_fd = open(device, O_WRONLY, 0)) == -1) {
    perror(device);
    return 0;
  }
  // buffering
  int buffering = (AUDIO_BUFFERS << 16) | 8; // fragment size 2^8 = 256
  if (ioctl(audio_fd, SNDCTL_DSP_SETFRAGMENT, &buffering)) {
```

```
    perror("SNDCTL_DSP_SETFRAGMENT");
    return 0;
  }
  // int16_t
  int format = AFMT_S16_NE;
  if (ioctl(audio_fd, SNDCTL_DSP_SETFMT, &format) == -1) {
    perror("SNDCTL_DSP_SETFMT");
    return 0;
  }
  if (format != AFMT_S16_LE) {
    fprintf(stderr, "BAD OSS FORMAT %d != %d\n", format, AFMT_S16_NE);
    return 0;
  }
  // stereo
  int stereo = 1;
  if (ioctl(audio_fd, SNDCTL_DSP_STEREO, &stereo) == -1) {
    perror("SNDCTL_DSP_STEREO");
    return 0;
  }
  if (stereo != 1) {
    fprintf(stderr, "BAD OSS STEREO %d != 1\n", stereo);
    return 0;
  }
  // sample rate
  int speed = audio->sr;
  if (ioctl(audio_fd, SNDCTL_DSP_SPEED, &speed) == -1) {
    perror("SNDCTL_DSP_SPEED");
    return 0;
  }
  if (speed != audio->sr) {
    fprintf(stderr, "BAD OSS SPEED %d != %d\n", speed, (int) audio->sr);
    return 0;
  }
  // loop
  jack_default_audio_sample_t *out[2] = { &audio->oss_jack_buffer[0][0], &audio
      ↪ ->oss_jack_buffer[1][0] };
  const int nframes = audio->size / 2;
  while (audio->running) {
    audio_compute(audio, out, nframes);
    for (int i = 0; i < nframes; ++i) {
      for (int c = 0; c < 2; ++c) {
        audio->oss_buffer[i][c] = 32767 * fmin(fmax(audio->oss_jack_buffer[c][i
            ↪ ], -1), 1);
      }
    }
    if (write(audio_fd, &audio->oss_buffer[0][0], nframes * 2 * sizeof(int16_t))
        ↪ == -1) {
      perror("audio write");
      //exit(1);
    }
  }
  return 0;
}

void audio_start_oss(struct audio *audio) {
  pthread_create(&audio->oss_thread, 0, audio_process_oss, audio);
}
```

```
/* JACK sample rate callback */
static int audio_srate(jack_nframes_t sr, void *arg) {
  struct audio *audio = arg;
  audio->sr = sr;
  audio->size = 240; // FIXME hardcoded
  fprintf(stderr, "JACK sample rate: %d\n", sr);
  return 0;
}

/* JACK error callback */
static void audio_error(const char *desc) {
  fprintf(stderr, "JACK error: %s\n", desc);
}

/* JACK shutdown callback */
static void audio_shutdown(void *arg) {
  fprintf(stderr, "JACK shutdown\n");
}

/* JACK timebase callback */
static void audio_timebase(jack_transport_state_t state, jack_nframes_t nframes,↙
    ↪ jack_position_t *pos, int new_pos, void *arg) {
  //struct audio *audio = arg;
  if (new_pos || audio_reset) {
    pos->valid = JackPositionBBT;
    pos->beats_per_bar = 4;
    pos->beat_type = 0.25;
    pos->ticks_per_beat = 1920;
    pos->beats_per_minute = 120;
    audio_reset = 0;
    pos->bar = 1;
    pos->beat = 1;
    pos->tick = 0;
    pos->bar_start_tick = (pos->bar - 1) * pos->beats_per_bar * pos->↙
        ↪ ticks_per_beat;
  } else {
    pos->tick +=nframes * pos->ticks_per_beat * pos->beats_per_minute / (pos->↙
        ↪ frame_rate * 60);
    while (pos->tick >= pos->ticks_per_beat) {
      pos->tick -= pos->ticks_per_beat;
      if (++pos->beat > pos->beats_per_bar) {
        pos->beat = 1;
        ++pos->bar;
        pos->bar_start_tick += pos->beats_per_bar * pos->ticks_per_beat;
      }
    }
  }
}

/* exit callback */
void audio_atexit(struct audio *audio) {
  if (audio->nrt) {
    free(audio->nrtbuffers[0]);
    free(audio->nrtbuffers[1]);
    free(audio->nrtbuffers_interleaved);
    sf_close(audio->nrtf);
```

```
  } else {
    audio->running = 0;
    if (0)
      jack_client_close(audio->client);
    if (1)
      pthread_join(audio->oss_thread, 0);
  }
}

//=========================================================
// ...
struct audio *audio_init(struct audio *audio, int nrt) {
  if (! audio) { return 0; }
  if (! shader_init(&audio->shader, 0, audio_frag)) {
    return 0;
  }
  shader_uniform(audio, texture);
  shader_uniform(audio, coords);
  audio->value.texture = 0;
  audio->value.coords = 1;
  // pitch
  audio->pitch = 1;
  // scan point
  audio->x = 0;
  audio->y = 0;
  audio->a = 0;
  audio->curve = 40;
  audio->speed = 128;
  audio->dcurve = 0;
  audio->dspeed = 0;
  // time passing
  audio->samples = 0;

  // pixel buffers
  for (int c = 0; c < AUDIO_CHANNELS; ++c) {
    glGenBuffersARB(AUDIO_COUNT, &audio->pbos[c][0]);
    for (int i = 0; i < AUDIO_COUNT; ++i) {
      glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, audio->pbos[c][i]);
      glBufferDataARB(GL_PIXEL_PACK_BUFFER_ARB, audio_tex_size * audio_tex_size ↙
          ↳ * sizeof(GLfloat), 0, GL_STREAM_READ_ARB); // 1 channel
    }
  }
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);

  /* default sample rate */
  audio->sr = 48000;
  audio->size = 240; // FIXME hardcoded...
  //audio->value.size = audio->size;
  glGenTextures(2, &audio->textures[0]);
  glEnable(GL_TEXTURE_2D);
  // output
  glBindTexture(GL_TEXTURE_2D, audio->textures[0]);
  glTexImage2D(
    GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, audio_tex_size, audio_tex_size, 0, GL_RGBA↙
        ↳ ,
    GL_FLOAT, 0
  );
```

```
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  // coordinates
  glBindTexture(GL_TEXTURE_2D, audio->textures[1]);
  glTexImage2D(
    GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, audio_tex_size, audio_tex_size, 0, GL_RGBA,
    GL_FLOAT, 0 // audio->coords
  );
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glDisable(GL_TEXTURE_2D);
  audio->index = 0;
  /* initialize DC blocking filters */
  for (int c = 0; c < AUDIO_CHANNELS; ++c) {
    for (int d = 0; d < 2; ++d) {
      audio->r[d][c] = 0.995;
      audio->xn1[d][c] = 0;
      audio->yn1[d][c] = 0;
    }
    // initialize compressor/limiter
    audio->gain[c] = 1.0;
  }
  audio->nrt = nrt;
  audio->frame = 0;
  if (audio->nrt) {
//    audio->size = audio->sr / 25;
    audio->nrtbuffers[0] = calloc(1, 8 * audio->size * sizeof(jack_default_audio_sample_t));// FIXME hardcoded
    audio->nrtbuffers[1] = calloc(1, 8 * audio->size * sizeof(jack_default_audio_sample_t));// FIXME hardcoded
    audio->nrtbuffers_interleaved = calloc(1, 2 * 8 * audio->size * sizeof(float));// FIXME hardcoded
    SF_INFO info = { 0, audio->sr, 2, SF_FORMAT_WAV | SF_FORMAT_FLOAT, 0, 0 };
    audio->nrtf = sf_open("audio.wav", SFM_WRITE, &info);
  } else if (1) {
    audio->state = audio_state_idle;
    audio->sphase = 0;
    audio->running = 1;
    audio_start_oss(audio);
  } else {
    /* set up JACK */
    audio->state = audio_state_idle;
    audio->sphase = 0;
    jack_set_error_function(audio_error);
    if (!(audio->client = jack_client_open("rdex", JackNoStartServer | JackUseExactName, 0))) {
      fprintf (stderr, "JACK server not running?\n");
    } else {
      jack_set_process_callback(audio->client, audio_process, audio);
      jack_set_sample_rate_callback(audio->client, audio_srate, audio);
      jack_on_shutdown(audio->client, audio_shutdown, audio);
      /* create ports */
      audio->port[0] = jack_port_register(
        audio->client, "output_1", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
```

```
      );
      audio->port[1] = jack_port_register(
360     audio->client, "output_2", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
      );
      /* activate audio */
      if (jack_activate(audio->client)) {
        fprintf (stderr, "cannot activate JACK client\n");
365   } else {
#if 1
        /* must be activated before connecting JACK ports */
        const char **ports;
        if ((ports = jack_get_ports(
370       audio->client, NULL, NULL, JackPortIsPhysical | JackPortIsInput
        ))) {
          /* connect up to two physical playback ports */
          int i = 0;
          while (ports[i] && i < 2) {
375         if (jack_connect(
              audio->client, jack_port_name(audio->port[i]), ports[i]
            )) {
              fprintf(stderr, "cannot connect JACK output port\n");
            }
380         i++;
          }
          free(ports);
        }
#endif
385     /* become timebase master */
        if (jack_set_timebase_callback(audio->client, 0, audio_timebase, audio))↙
            ↳ {
          fprintf(stderr, "cannot become JACK master\n");
        }
      }
390 }
  }
  return audio;
}

395 //===========================================================================
// ...
void audio_display1(struct audio *audio, GLuint fbo, GLuint texture, int iter) {
  { // compute coordinates
    memset(audio->coords, 0, 4 * audio_tex_size * audio_tex_size * sizeof(float)↙
        ↳ );
400 float x  = audio->x;
    float y  = audio->y;
    float a  = audio->a;
    float pi2 = 2.0 * 3.1415926;
    float ds =       audio->speed / 256.0 / 256.0;
405 for (int i = 0; i < audio->size; ++i) {
      audio->curve = audio->pitch + 1.0 * rand() / (double) RAND_MAX;
      float da = pi2 * audio->curve / audio->sr;
      float dx = ds * cosf(a);
      float dy = ds * sinf(a);
410   x += dx;
      y += dy;
      a += da;
```

```
      audio->coords[2 * i + 0] = x;
      audio->coords[2 * i + 1] = y;
    }
    audio->x  = x - floorf(x);
    audio->y  = y - floorf(y);
    audio->a  = a - floorf(a/pi2) * pi2;
    audio->samples += audio->size;
  }
  // upload coordinates
  glEnable(GL_TEXTURE_2D);
  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_2D, audio->textures[1]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, audio_tex_size, audio_tex_size,↙
      ↳ 0, GL_RG, GL_FLOAT, audio->coords);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  // image
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  // extract
  glUseProgramObjectARB(audio->shader.program);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0, 1, 0, 1);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glViewport(0, 0, audio_tex_size, audio_tex_size);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,↙
      ↳ GL_TEXTURE_2D, audio->textures[0], 0);
  shader_updatei (audio, texture);
  shader_updatei (audio, coords);
  glBegin(GL_QUADS); { glColor4f(1,1,1,1);
    glTexCoord2f(0, 0); glVertex2f(0, 0);
    glTexCoord2f(1, 0); glVertex2f(1, 0);
    glTexCoord2f(1, 1); glVertex2f(1, 1);
    glTexCoord2f(0, 1); glVertex2f(0, 1);
  } glEnd();
  glActiveTexture(GL_TEXTURE1); glBindTexture(GL_TEXTURE_2D, 0);
  glActiveTexture(GL_TEXTURE0); glBindTexture(GL_TEXTURE_2D, 0);
  glUseProgramObjectARB(0);
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  // copy texture to PBO
  glBindTexture(GL_TEXTURE_2D, audio->textures[0]);
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, audio->pbos[0][iter]);
  glGetTexImage(GL_TEXTURE_2D, 0, GL_RED,   GL_FLOAT, 0); // async
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, audio->pbos[1][iter]);
  glGetTexImage(GL_TEXTURE_2D, 0, GL_GREEN, GL_FLOAT, 0); // async
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
  glBindTexture(GL_TEXTURE_2D, 0);
}

void audio_display2(struct audio *audio) {
```

```
  // copy pixel buffers
  for (int i = 0; i < AUDIO_COUNT; ++i) { // FIXME hardcoded
    int w = audio_write;
    for (int c = 0; c < AUDIO_CHANNELS; ++c) {
      glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, audio->pbos[c][i]);
      float *p = glMapBufferARB(GL_PIXEL_PACK_BUFFER_ARB, GL_READ_ONLY_ARB);
      if (p) {
        memcpy(&audio->buffer[w][c][0], p, audio->size * sizeof(float));
        if (! glUnmapBufferARB(GL_PIXEL_PACK_BUFFER_ARB)) {
          fprintf(stderr, "unmap buffer error\n");
        }
      } else {
        memset(&audio->buffer[w][c][0], 0, audio->size * sizeof(float));
      }
    }
    // don't overtake read pointer
    int audio_write2 = (w + 1) % AUDIO_BUFFERS;
    if (audio_write2 == audio_read) {
      audio_write2 = (audio_write2 - 1 + AUDIO_BUFFERS) % AUDIO_BUFFERS;
    }
    audio_write = audio_write2;
  }
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
  // non-realtime fun
#if 1
  if (audio->nrt) {
    int rate = 1;
    int fps = 25;
    int frame = audio->frame;
    int period = 60 * 60 * fps * rate; // 60 minutes
    int interval = 5 * 60 * fps * rate; // 5 minutes
    if ((frame % rate) == 0 && (frame % period) < interval) {
      audio_compute(audio, audio->nrtbuffers, 8 * audio->size); // FIXME ↙
          ↳ hardcoded
      for (int i = 0; i < 8 * audio->size; ++i) { // FIXME hardcoded
        for (int c = 0; c < 2; ++c) {
          audio->nrtbuffers_interleaved[i*2+c] = audio->nrtbuffers[c][i];
        }
      }
      sf_write_float(audio->nrtf, &audio->nrtbuffers_interleaved[0], 2 * 8 * ↙
          ↳ audio->size);// FIXME hardcoded
    } else if (frame % period == interval) {
      sf_write_sync(audio->nrtf);
    }
  }
#endif
  audio->frame++;
}

void audio_start(struct audio *audio) {
  switch (audio->state) {
    case audio_state_idle:
      audio->state = audio_state_intro;
      audio->sphase = 0;
      if (0)
        jack_transport_start(audio->client);
      break;
```

```
    default:
      break;
  }
}

void audio_stop(struct audio *audio) {
  switch (audio->state) {
    case audio_state_rolling:
      audio->state = audio_state_outro;
      audio->sphase = 0;
      break;
    default:
      break;
  }
}

// EOF
```

14 src/audio.frag

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Extract from input texture using a coordinate texture
================================================================ */

uniform sampler2D texture;
uniform sampler2D coords;

/*
uniform vec2 texsize;
uniform vec2 center;
uniform float radius;
uniform float size;
*/

void main(void) {
  gl_FragColor = texture2D(texture, texture2D(coords, gl_TexCoord[0].st).rg);
/*
  vec2 p = gl_TexCoord[0].st;
  float k = texsize.x * texsize.y / size;
  float da = 2.0 * 3.1415926 / size;
  float a = (p.y + p.x / texsize.x) * 2.0 * 3.1415926 * k;
  vec4 c = vec4(0.0, 0.0, 0.0, 0.0);
  vec2 q;
  q = center + radius * vec2(cos(a), sin(a));
  c += texture2D(texture, q);
  a += da;
  q = center + radius * vec2(cos(a), sin(a));
  c += texture2D(texture, q);
  gl_FragColor = c * 0.5;
*/
}
```

15 src/audio.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2019  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Audio output
============================================================================ */

#ifndef AUDIO_H
#define AUDIO_H 1

#include <stdio.h>
#include <pthread.h>
#include <jack/jack.h>
#include <sndfile.h>
#include "shader.h"

//============================================================================
// audio state

#define AUDIO_BUFFERS 64
#define AUDIO_CHANNELS 2
#define audio_tex_size 16
#define AUDIO_BUFSIZE (audio_tex_size * audio_tex_size)
#define AUDIO_COUNT 8

// synchronise between JACK dsp callback and GLUT timer callback
// JACK is the master clock
extern volatile char audio_read;
extern volatile char audio_write;

struct audio { struct shader shader;
  struct { GLint texture, coords; /*texsize, center, radius, size;*/ } uniform;
  struct { GLint texture, coords; /*GLfloat texsize[2], center[2], radius, size
      ↳ ;*/ } value;
  GLuint textures[2];
  float coords[4 * audio_tex_size * audio_tex_size];
  double pitch; // frequency in Hz
  double x, y, a; // scan point
  double curve, speed, dcurve, dspeed; // scan movement
  double samples; // total samples scanned
  GLuint pbos[AUDIO_CHANNELS][AUDIO_COUNT]; // pixel buffers for reading back
      ↳ from GPU (FIXME hardcoded count)

  /* non-realtime mode */
  int nrt;
  int frame;
  jack_default_audio_sample_t *nrtbuffers[2];
  float *nrtbuffers_interleaved;
  SNDFILE *nrtf;
  /* OSS handles */
  int running;
  pthread_t oss_thread;
  int oss_fd;
  int16_t oss_buffer[AUDIO_BUFSIZE][AUDIO_CHANNELS];
  jack_default_audio_sample_t oss_jack_buffer[AUDIO_CHANNELS][AUDIO_BUFSIZE];
  /* JACK handles */
  jack_client_t *client;
```

```
  jack_port_t *port[2];
  enum {
    audio_state_idle,
    audio_state_prepare,
    audio_state_intro,
    audio_state_rolling,
    audio_state_outro
  } state;
  double sphase;
  /* audio buffers */
  unsigned int index;
  jack_default_audio_sample_t buffer[AUDIO_BUFFERS][AUDIO_CHANNELS][AUDIO_BUFSIZE];
  int size;
  int sr;
  /* DC blocker filters */
  jack_default_audio_sample_t r[2][AUDIO_CHANNELS], xn1[2][AUDIO_CHANNELS], yn1[2][AUDIO_CHANNELS], gain[AUDIO_CHANNELS];
  /* intro */
  double iphase[AUDIO_CHANNELS];
  jack_nframes_t sample[AUDIO_CHANNELS];
};

//=============================================================================
// prototypes
struct audio *audio_init(struct audio *audio, int nrt);
void audio_atexit(struct audio *audio);
void audio_display1(struct audio *audio, GLuint fbo, GLuint texture, int iter);
void audio_display2(struct audio *audio);
void audio_start(struct audio *audio);
void audio_stop(struct audio *audio);

#endif
```

16 src/background.c

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Background processing
============================================================================ */

#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include "background.h"
#include "histogram.h"
#include "image.h"
#include "rdex.h"
#include "segment.h"
#include "tamura.h"

//=============================================================================
// upload
#define UPLOAD_DATA_STRLEN 32
#define UPLOAD_DATA_PPMLEN (1024 + rdex_tex_size * rdex_tex_size * 3)
```

```
struct background_upload {
  char ru                [UPLOAD_DATA_STRLEN];
  char rv                [UPLOAD_DATA_STRLEN];
  char f                 [UPLOAD_DATA_STRLEN];
  char k                 [UPLOAD_DATA_STRLEN];
  char behaviour         [UPLOAD_DATA_STRLEN];
  char coarseness_u      [UPLOAD_DATA_STRLEN];
  char contrast_u        [UPLOAD_DATA_STRLEN];
  char directionality_u  [UPLOAD_DATA_STRLEN];
  char coarseness_v      [UPLOAD_DATA_STRLEN];
  char contrast_v        [UPLOAD_DATA_STRLEN];
  char directionality_v  [UPLOAD_DATA_STRLEN];
  char histogram[HISTOGRAM_BUCKETS][HISTOGRAM_BUCKETS][HISTOGRAM_BUCKETS][UPLOAD_DATA_STRLEN];
  char segment[SEGMENT_BUCKETS][UPLOAD_DATA_STRLEN];
  char ppm               [UPLOAD_DATA_PPMLEN];
  int  ppm_bytes;
};

//=============================================================================
// background processing data
struct background_data {
  float ru; float rv; float f; float k; float score; const char *behaviour;
//  char *thumb_ppm; int thumb_ppm_bytes; // must be freed by background thread
  char *full_ppm; int full_ppm_bytes; // must be freed by background thread
  struct image *raw_image; // must be freed by background thread
  struct image *full_image; // must be freed by background thread
};

//=============================================================================
// prototypes
void background_background(struct background *, size_t, struct background_data *);

//=============================================================================
// initialise uploader
struct background *background_init(struct background *background, const char *session) {
  CURLcode c = curl_global_init(CURL_GLOBAL_ALL);
  if (c) { return 0; }
  background->session = session;
  background->url = getenv("RDEX_UPLOAD");
  background->queue = pfifo_create((pfifo_consumer *) background_background, background);
  tamura_init(&background->tamura, rdex_tex_size, rdex_tex_size); //FIXME
  return background;
}

//=============================================================================
// clean up uploader
void background_atexit(struct background *background) {
  curl_global_cleanup();
}

//=============================================================================
// prepare background processing
void background_enqueue(
```

```
75    struct background *background,
      float ru, float rv, float f, float k, float score, const char *behaviour,
//    char *thumb_ppm, int thumb_ppm_bytes, // must be freed by background thread
      char *full_ppm, int full_ppm_bytes,  // must be freed by background thread
      struct image *raw_image, // must be freed by background thread
80    struct image *full_image // must be freed by background thread
    ) {
      struct background_data data;
      data.ru = ru;
      data.rv = rv;
85    data.f = f;
      data.k = k;
      data.score = score;
      data.behaviour = behaviour;
//    data.thumb_ppm = thumb_ppm;
90  //  data.thumb_ppm_bytes = thumb_ppm_bytes;
      data.full_ppm = full_ppm;
      data.full_ppm_bytes = full_ppm_bytes;
      data.raw_image = raw_image;
      data.full_image = full_image;
95    pfifo_enqueue(background->queue, sizeof(struct background_data), &data);
    }

    //=============================================================================
    // do background processing
100 void background_background(
      struct background *background,
      size_t size,
      struct background_data *data
    ) {
105   { // save image
        char filename[1024];
        snprintf(
          filename, 1000,
          "%s/ru_%f__rv_%f__f_%f__k_%f__s_%f.ppm",  // FIXME ensure same stem
110       background->session, data->ru, data->rv, data->f, data->k, data->score
        );
        FILE *imgfile = fopen(filename, "wb");
        if (imgfile) {
          fwrite(data->full_ppm, data->full_ppm_bytes, 1, imgfile);
115       fclose(imgfile);
        }
      } // save thumbnail

      if (background->url) { // upload
120     // image analysis
        struct tamura_features feature[2];
        struct histogram histogram;
        struct segment segment;
        tamura_calculate(&background->tamura, &feature[0], data->raw_image);
125     histogram_calc(&histogram, data->full_image);
        segment_calc(&segment, data->full_image);
        struct background_upload up;
        snprintf(up.ru,                UPLOAD_DATA_STRLEN, "%f", data->ru);
        snprintf(up.rv,                UPLOAD_DATA_STRLEN, "%f", data->rv);
130     snprintf(up.f,                 UPLOAD_DATA_STRLEN, "%f", data->f);
        snprintf(up.k,                 UPLOAD_DATA_STRLEN, "%f", data->k);
```

```
    snprintf(up.behaviour,          UPLOAD_DATA_STRLEN, "%s", data->behaviour);
    snprintf(up.coarseness_u,       UPLOAD_DATA_STRLEN, "%f", feature[0].↙
        ↳ coarseness);
    snprintf(up.contrast_u,         UPLOAD_DATA_STRLEN, "%f", feature[0].contrast)↙
        ↳ ;
    snprintf(up.directionality_u, UPLOAD_DATA_STRLEN, "%f", feature[0].↙
        ↳ directionality);
    snprintf(up.coarseness_v,       UPLOAD_DATA_STRLEN, "%f", feature[1].↙
        ↳ coarseness);
    snprintf(up.contrast_v,         UPLOAD_DATA_STRLEN, "%f", feature[1].contrast)↙
        ↳ ;
    snprintf(up.directionality_v, UPLOAD_DATA_STRLEN, "%f", feature[1].↙
        ↳ directionality);
    for (int r = 0; r < HISTOGRAM_BUCKETS; ++r) {
    for (int g = 0; g < HISTOGRAM_BUCKETS; ++g) {
    for (int b = 0; b < HISTOGRAM_BUCKETS; ++b) {
      snprintf(up.histogram[r][g][b], UPLOAD_DATA_STRLEN, "%f", histogram.bucket↙
          ↳ [r][g][b]);
    }
    }
    }
    for (int s = 0; s < SEGMENT_BUCKETS; ++s) {
      snprintf(up.segment[s], UPLOAD_DATA_STRLEN, "%f", segment.bucket[s]);
    }
    struct curl_httppost *formpost = NULL;
    struct curl_httppost *lastptr = NULL;
#define FORM(s) curl_formadd(&formpost, &lastptr, CURLFORM_COPYNAME, #s, ↙
    ↳ CURLFORM_COPYCONTENTS, up . s, CURLFORM_END)
    FORM(ru);
    FORM(rv);
    FORM(f);
    FORM(k);
    FORM(behaviour);
    FORM(coarseness_u);
    FORM(contrast_u);
    FORM(directionality_u);
    FORM(coarseness_v);
    FORM(contrast_v);
    FORM(directionality_v);
#undef FORM
    for (int r = 0; r < HISTOGRAM_BUCKETS; ++r) {
    for (int g = 0; g < HISTOGRAM_BUCKETS; ++g) {
    for (int b = 0; b < HISTOGRAM_BUCKETS; ++b) {
      char hist[UPLOAD_DATA_STRLEN];
      snprintf(hist, UPLOAD_DATA_STRLEN, "hist_%d_%d_%d", r, g, b);
      curl_formadd(
        &formpost, &lastptr,
        CURLFORM_COPYNAME, hist,
        CURLFORM_COPYCONTENTS, up.histogram[r][g][b],
        CURLFORM_END
      );
    }
    }
    }
    for (int s = 0; s < SEGMENT_BUCKETS; ++s) {
      char seg[UPLOAD_DATA_STRLEN];
      snprintf(seg, UPLOAD_DATA_STRLEN, "seg_%d", s);
```

```
      curl_formadd(
        &formpost, &lastptr,
        CURLFORM_COPYNAME, seg,
        CURLFORM_COPYCONTENTS, up.segment[s],
        CURLFORM_END
      );
    }
    curl_formadd(
      &formpost, &lastptr,
      CURLFORM_COPYNAME, "image",
      CURLFORM_BUFFER, "rdex.ppm",
      CURLFORM_BUFFERPTR, data->full_ppm,
      CURLFORM_BUFFERLENGTH, data->full_ppm_bytes,
      CURLFORM_END
    );
    curl_formadd(
      &formpost, &lastptr,
      CURLFORM_COPYNAME, "submit",
      CURLFORM_COPYCONTENTS, "submit",
      CURLFORM_END
    );
    CURL *curl = curl_easy_init();
    if (curl) {
      curl_easy_setopt(curl, CURLOPT_URL, background->url);
      curl_easy_setopt(curl, CURLOPT_HTTPPOST, formpost);
      curl_easy_setopt(curl, CURLOPT_WRITEDATA, stderr);
      curl_easy_perform(curl);
      curl_easy_cleanup(curl);
      curl_formfree(formpost);
    }
  } // upload

  { // cleanup
//    free(data->thumb_ppm);
    free(data->full_ppm);
    image_free(data->raw_image);
    image_free(data->full_image);
  } // cleanup

}

// EOF
```

17 src/background.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Background processing
============================================================================ */

#ifndef BACKGROUND_H
#define BACKGROUND_H 1

#include "image.h"
#include "histogram.h"
```

```
#include "segment.h"
#include "tamura.h"
#include "pfifo.h"

//=============================================================
// background state
struct background {
  const char *session;
  char *url;
  struct tamura tamura;
  struct pfifo *queue;
};

//=============================================================
// prototypes
struct background *background_init(struct background *background, const char *session);
void background_atexit(struct background *background);
void background_enqueue(
  struct background *background,
  float ru, float rv, float f, float k, float score, const char *behaviour,
//  char *thumb_ppm, int thumb_ppm_bytes, // must be freed by background thread
  char *full_ppm, int full_ppm_bytes,  // must be freed by background thread
  struct image *raw_image, // must be freed by background thread
  struct image *full_image // must be freed by background thread
);

#endif
```

18 src/bloom.c

```
/* =============================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
-------------------------------------------------------------
Bloom/Glow Effect
============================================================= */

#include "util.h"
#include "bloom.h"
#include "bloom.frag.c"

//=============================================================
// initialization
struct bloom *bloom_init(struct bloom *bloom) {
  if (! bloom) { return 0; }
  if (! gblur_init(&bloom->gblur)) { return 0; }
  if (! shader_init(&bloom->shader, 0, bloom_frag)) { return 0; }
  shader_uniform(bloom, tex0);
  shader_uniform(bloom, tex1);
  shader_uniform(bloom, tex2);
  shader_uniform(bloom, tex3);
  shader_uniform(bloom, tex4);
  shader_uniform(bloom, tex5);
  shader_uniform(bloom, tex6);
  shader_uniform(bloom, tex7);
  bloom->value.tex0 = 0;
```

```
  bloom->value.tex1 = 1;
  bloom->value.tex2 = 2;
  bloom->value.tex3 = 3;
  bloom->value.tex4 = 4;
  bloom->value.tex5 = 5;
  bloom->value.tex6 = 6;
  bloom->value.tex7 = 7;
  return bloom;
}

//=========================================================
// reshape callback
void bloom_reshape(struct bloom *bloom, int w, int h) {
  gblur_reshape(&bloom->gblur, w, h);
}

//=========================================================
// display callback
void bloom_display(struct bloom *bloom, GLuint fbo, GLuint texture) {
  // generate blurred copies
  gblur_display(&bloom->gblur, fbo, texture);
  // combine them (back into the input texture)
  glEnable(GL_TEXTURE_2D);
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  glViewport(0, 0, 1<<bloom->gblur.count, 1<<bloom->gblur.count);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0, 1, 0, 1);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
      ↳ GL_TEXTURE_2D, texture, 0);
  glActiveTexture(GL_TEXTURE7); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 7]);
  glActiveTexture(GL_TEXTURE6); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 6]);
  glActiveTexture(GL_TEXTURE5); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 5]);
  glActiveTexture(GL_TEXTURE4); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 4]);
  glActiveTexture(GL_TEXTURE3); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 3]);
  glActiveTexture(GL_TEXTURE2); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 2]);
  glActiveTexture(GL_TEXTURE1); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 1]);
  glActiveTexture(GL_TEXTURE0); glBindTexture(GL_TEXTURE_2D, bloom->gblur.↙
      ↳ textures[0][bloom->gblur.count - 0]);
  glUseProgramObjectARB(bloom->shader.program);
  shader_updatei(bloom, tex0);
  shader_updatei(bloom, tex1);
  shader_updatei(bloom, tex2);
  shader_updatei(bloom, tex3);
  shader_updatei(bloom, tex4);
  shader_updatei(bloom, tex5);
  shader_updatei(bloom, tex6);
  shader_updatei(bloom, tex7);
```

```
75    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
        glTexCoord2f(0, 0); glVertex2f(0, 0);
        glTexCoord2f(1, 0); glVertex2f(1, 0);
        glTexCoord2f(1, 1); glVertex2f(1, 1);
        glTexCoord2f(0, 1); glVertex2f(0, 1);
80    } glEnd();
      // clean up
      glUseProgramObjectARB(0);
      //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
      glActiveTexture(GL_TEXTURE7); glBindTexture(GL_TEXTURE_2D, 0);
85    glActiveTexture(GL_TEXTURE6); glBindTexture(GL_TEXTURE_2D, 0);
      glActiveTexture(GL_TEXTURE5); glBindTexture(GL_TEXTURE_2D, 0);
      glActiveTexture(GL_TEXTURE4); glBindTexture(GL_TEXTURE_2D, 0);
      glActiveTexture(GL_TEXTURE3); glBindTexture(GL_TEXTURE_2D, 0);
      glActiveTexture(GL_TEXTURE2); glBindTexture(GL_TEXTURE_2D, 0);
90    glActiveTexture(GL_TEXTURE1); glBindTexture(GL_TEXTURE_2D, 0);
      glActiveTexture(GL_TEXTURE0); glBindTexture(GL_TEXTURE_2D, 0);
      glDisable(GL_TEXTURE_2D);
    }

95  // EOF
```

19 src/bloom.frag

```
    /* ================================================================
    rdex -- reaction-diffusion explorer
    Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
    ----------------------------------------------------------------
 5  Bloom/Glow effect
    ================================================================ */

    uniform sampler2D tex0;
    uniform sampler2D tex1;
10  uniform sampler2D tex2;
    uniform sampler2D tex3;
    uniform sampler2D tex4;
    uniform sampler2D tex5;
    uniform sampler2D tex6;
15  uniform sampler2D tex7;

    void main(void) {
      //float size;
      //vec2 p0, p1, dp;
20    vec2 p = gl_TexCoord[0].st;
      //vec4 o, o0;
    /* // waaaaaaay too big for most hardware :(
      size = 1024.0; // FIXME hardcoded
      p0 = floor(p * size) / size;
25    p1 = p0 + vec2(1.0, 1.0) / size;
      dp = (p1 - p0) * size;
      o0 = mix(mix(texture2D(tex0, p0), texture2D(tex0, vec2(p1.x, p0.y)), dp.x),
               mix(texture2D(tex0, vec2(p0.x, p1.y)), texture2D(tex0, p1), dp.x), dp
                   ↳ .y);
      o = o0 * 9.0;
30
      size *= 0.5;
      p0 = floor(p * size) / size;
```

```
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex1, p0), texture2D(tex1, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex1, vec2(p0.x, p1.y)), texture2D(tex1, p1), dp.x), dp.y);

    size *= 0.5;
    p0 = floor(p * size) / size;
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex2, p0), texture2D(tex2, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex2, vec2(p0.x, p1.y)), texture2D(tex2, p1), dp.x), dp.y);

    size *= 0.5;
    p0 = floor(p * size) / size;
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex3, p0), texture2D(tex3, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex3, vec2(p0.x, p1.y)), texture2D(tex3, p1), dp.x), dp.y);

    size *= 0.5;
    p0 = floor(p * size) / size;
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex4, p0), texture2D(tex4, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex4, vec2(p0.x, p1.y)), texture2D(tex4, p1), dp.x), dp.y);

    size *= 0.5;
    p0 = floor(p * size) / size;
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex5, p0), texture2D(tex5, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex5, vec2(p0.x, p1.y)), texture2D(tex5, p1), dp.x), dp.y);

    size *= 0.5;
    p0 = floor(p * size) / size;
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex6, p0), texture2D(tex6, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex6, vec2(p0.x, p1.y)), texture2D(tex6, p1), dp.x), dp.y);

    size *= 0.5;
    p0 = floor(p * size) / size;
    p1 = p0 + vec2(1.0, 1.0) / size;
    dp = (p1 - p0) * size;
    o += mix(mix(texture2D(tex7, p0), texture2D(tex7, vec2(p1.x, p0.y)), dp.x),
             mix(texture2D(tex7, vec2(p0.x, p1.y)), texture2D(tex7, p1), dp.x), dp.y);
*/
    vec4 o0 = texture2D(tex0, p);
    vec4 o1 = texture2D(tex1, p);
    vec4 o2 = texture2D(tex2, p);
```

```
  vec4 o3 = texture2D(tex3, p);
  vec4 o4 = texture2D(tex4, p);
  vec4 o5 = texture2D(tex5, p);
  vec4 o6 = texture2D(tex6, p);
  vec4 o7 = texture2D(tex7, p);
  vec4 o;
  float k = 1.1;
  float t = 1.0;
  o  = o7; o *= k; t *= k;
  o += o6; o *= k; t += 1.0; t *= k;
  o += o5; o *= k; t += 1.0; t *= k;
  o += o4; o *= k; t += 1.0; t *= k;
  o += o3; o *= k; t += 1.0; t *= k;
  o += o2; o *= k; t += 1.0; t *= k;
  o += o1; o *= k; t += 1.0; t *= k;
  o += o0; o *= k; t += 1.0; t *= k;
  o /= t;
  float l = length(o.rgb);
  gl_FragColor = vec4(o.rgb * clamp(log(5.0 * log(5.0 * l + 1.0) + 1.0), 0.0, ↙
      ↪ 1.0), 1.0);
}
```

20 src/bloom.h

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------
Bloom/Glow Effect
========================================================================== */

#ifndef BLOOM_H
#define BLOOM_H 1

#include "shader.h"
#include "gblur.h"

//============================================================================
// shader data
struct bloom { struct shader shader;
  struct { GLint tex0, tex1, tex2, tex3, tex4, tex5, tex6, tex7; } uniform;
  struct { int   tex0, tex1, tex2, tex3, tex4, tex5, tex6, tex7; } value;
  struct gblur gblur;
};

//============================================================================
// prototypes
struct bloom *bloom_init(struct bloom *bloom);
void bloom_reshape(struct bloom *bloom, int w, int h);
void bloom_display(struct bloom *bloom, GLuint fbo, GLuint texture);

#endif
```

21 src/blur.c

```
/* ==========================================================================
```

```
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
-------------------------------------------------------------------------------
Blur shader
=============================================================================== */

#include "blur.h"
#include "blur.frag.c"

//=============================================================================
// blur shader initialization
struct blur *blur_init(struct blur *blur) {
  if (! blur) { return 0; }
  if (! shader_init(&blur->shader, 0, blur_frag)) { return 0; }
  shader_uniform(blur, texture);
  shader_uniform(blur, d);
  blur->value.texture = 0;
  blur->value.d[0] = 0;
  blur->value.d[1] = 0;
  return blur;
}

// EOF
```

22 src/blur.frag

```
/* =============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
============================================================================= */

uniform sampler2D texture;
uniform vec2 d;

void main(void) {
  vec2 p = gl_TexCoord[0].st;
  vec4 x00 = texture2D(texture, p);
  vec4 x01 = texture2D(texture, p + vec2(0.0,d.y));
  vec4 x10 = texture2D(texture, p + vec2(d.x,0.0));
  vec4 x11 = texture2D(texture, p + vec2(d.x,d.y));
  vec2 p0 = floor(p / d) * d;
  vec2 dp = (p - p0) / d;
  vec3 tl = texture2D(texture, p0).rgb;
  vec3 tr = texture2D(texture, p0 + vec2(d.x, 0.0)).rgb;
  vec3 bl = texture2D(texture, p0 + vec2(0.0, d.y)).rgb;
  vec3 br = texture2D(texture, p0 + vec2(d.x, d.y)).rgb;
  vec3 t = tl * (1.0 - dp.x) + dp.x * tr;
  vec3 b = bl * (1.0 - dp.x) + dp.x * br;
  vec3 l = tl * (1.0 - dp.y) + dp.y * bl;
  vec3 r = tr * (1.0 - dp.y) + dp.y * br;
  vec3 m1 = t * (1.0 - dp.y) + dp.y * b;
  vec3 m2 = l * (1.0 - dp.x) + dp.x * r;
  vec3 m = 0.5 * (m1 + m2);
  gl_FragColor = vec4(m, 1.0);
}
```

23 src/blur.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Blur shader
================================================================ */

#ifndef BLUR_H
#define BLUR_H 1

#include "shader.h"

//================================================================
// arithmetic mean shader data
struct blur { struct shader shader;
  struct { GLint texture; GLint d; } uniform;
  struct { int   texture; float d[2]; } value;
};

//================================================================
// prototypes
struct blur *blur_init(struct blur *blur);

#endif
```

24 src/borderwindow.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Border/Window Shader
================================================================ */

#include "borderwindow.h"
#include "borderwindow.frag.c"
#include "projection.vert.c"
#include "borderwindow_pick.frag.c"
#include "projection_pick.vert.c"

//================================================================
// borderwindow shader initialization
struct borderwindow *borderwindow_init(struct borderwindow *borderwindow) {
  if (! borderwindow) { return 0; }
  if (! shader_init(&borderwindow->shader, projection_vert, borderwindow_frag)) ↙
      ↳ {
    return 0;
  }
  // vertex shader
  shader_uniform(borderwindow, origin);
  shader_uniform(borderwindow, scale);
  shader_uniform(borderwindow, rotate);
  shader_uniform(borderwindow, translate);
  shader_uniform(borderwindow, near);
```

```
  shader_uniform(borderwindow, far);
  borderwindow->value.origin[0] = 0.15;
  borderwindow->value.origin[1] = 0.15;
  borderwindow->value.origin[2] = 0.05;
  borderwindow->value.origin[3] = 0.05;
  borderwindow->value.scale[0] = 1.0/0.15;
  borderwindow->value.scale[1] = 1.0/0.15;
  borderwindow->value.scale[2] = 1.0/0.05;
  borderwindow->value.scale[3] = 1.0/0.05;
  int k = 0;
  for (int i = 0; i < 4; ++i) {
  for (int j = 0; j < 4; ++j) {
    borderwindow->value.rotate[k++] = i == j;
  }}
  borderwindow->value.translate[0] = 0.0;
  borderwindow->value.translate[1] = 0.0;
  borderwindow->value.translate[2] = 0.0;
  borderwindow->value.translate[3] = 8.0;
  borderwindow->value.near[0] = -1.0;
  borderwindow->value.near[1] = -1.0;
  borderwindow->value.near[2] = -1.0;
  borderwindow->value.near[3] =  4.0;
  borderwindow->value.far[0] =  1.0;
  borderwindow->value.far[1] =  1.0;
  borderwindow->value.far[2] =  1.0;
  borderwindow->value.far[3] = 12.0;
  // fragment shader
  shader_uniform(borderwindow, texture);
  shader_uniform(borderwindow, pack);
  shader_uniform(borderwindow, size);
  borderwindow->value.texture = 0;
  borderwindow->value.pack = 0;
  return borderwindow;
}

//=========================================================================
// borderwindow shader activation
void borderwindow_use(struct borderwindow *borderwindow) {
  if (borderwindow) {
    glUseProgramObjectARB(borderwindow->shader.program);
    borderwindow->attr.delta = glGetAttribLocationARB(borderwindow->shader.↙
        ↳ program, "delta");
    borderwindow->attr.spin  = glGetAttribLocationARB(borderwindow->shader.↙
        ↳ program, "spin");
    borderwindow->attr.pick  = glGetAttribLocationARB(borderwindow->shader.↙
        ↳ program, "pick");
    // vertex shader
    shader_updatef4(borderwindow, origin);
    shader_updatef4(borderwindow, scale);
    shader_updatem4(borderwindow, rotate);
    shader_updatef4(borderwindow, translate);
    shader_updatef4(borderwindow, near);
    shader_updatef4(borderwindow, far);
    // fragment shader
    shader_updatei (borderwindow, texture);
    shader_updatef (borderwindow, pack);
    shader_updatef (borderwindow, size);
```

```
  } else {
    glUseProgramObjectARB(0);
  }
}

//============================================================
// borderwindow picking shader initialization
struct borderwindow_pick *borderwindow_pick_init(struct borderwindow_pick *borderwindow_pick) {
  if (! borderwindow_pick) { return 0; }
  if (! shader_init(&borderwindow_pick->shader, projection_pick_vert, borderwindow_pick_frag)) {
    return 0;
  }
  // vertex shader
  shader_uniform(borderwindow_pick, origin);
  shader_uniform(borderwindow_pick, scale);
  shader_uniform(borderwindow_pick, rotate);
  shader_uniform(borderwindow_pick, translate);
  shader_uniform(borderwindow_pick, near);
  shader_uniform(borderwindow_pick, far);
  borderwindow_pick->value.origin[0] = 0.15;
  borderwindow_pick->value.origin[1] = 0.15;
  borderwindow_pick->value.origin[2] = 0.05;
  borderwindow_pick->value.origin[3] = 0.05;
  borderwindow_pick->value.scale[0] = 1.0/0.15;
  borderwindow_pick->value.scale[1] = 1.0/0.15;
  borderwindow_pick->value.scale[2] = 1.0/0.05;
  borderwindow_pick->value.scale[3] = 1.0/0.05;
  int k = 0;
  for (int i = 0; i < 4; ++i) {
  for (int j = 0; j < 4; ++j) {
    borderwindow_pick->value.rotate[k++] = i == j;
  }}
  borderwindow_pick->value.translate[0] = 0.0;
  borderwindow_pick->value.translate[1] = 0.0;
  borderwindow_pick->value.translate[2] = 0.0;
  borderwindow_pick->value.translate[3] = 8.0;
  borderwindow_pick->value.near[0] = -1.0;
  borderwindow_pick->value.near[1] = -1.0;
  borderwindow_pick->value.near[2] = -1.0;
  borderwindow_pick->value.near[3] =  4.0;
  borderwindow_pick->value.far[0] =  1.0;
  borderwindow_pick->value.far[1] =  1.0;
  borderwindow_pick->value.far[2] =  1.0;
  borderwindow_pick->value.far[3] = 12.0;
  return borderwindow_pick;
}

//============================================================
// borderwindow_pick shader activation
void borderwindow_pick_use(struct borderwindow_pick *borderwindow_pick) {
  if (borderwindow_pick) {
    glUseProgramObjectARB(borderwindow_pick->shader.program);
    borderwindow_pick->attr.delta = glGetAttribLocationARB(borderwindow_pick->shader.program, "delta");
    // vertex shader
```

```
135       shader_updatef4(borderwindow_pick, origin);
          shader_updatef4(borderwindow_pick, scale);
          shader_updatem4(borderwindow_pick, rotate);
          shader_updatef4(borderwindow_pick, translate);
          shader_updatef4(borderwindow_pick, near);
140       shader_updatef4(borderwindow_pick, far);
        } else {
          glUseProgramObjectARB(0);
        }
      }
145
      // EOF
```

25 src/borderwindow.frag

```
      /* ================================================================
      rdex -- reaction-diffusion explorer
      Copyright (C) 2008,2010,2011,2019  Claude Heiland-Allen <claude@mathr.co.uk>
      ----------------------------------------------------------------
  5   Sphere shader
      ================================================================ */

      uniform sampler2DArray tex;

 10   varying vec2 spin2;
      varying float scale2;

      void main(void) {
        float p = gl_TexCoord[0].x;               // float in [0..]
 15     vec2 q0 = gl_TexCoord[0].zw;
        vec2 q1 = 2.0 * (q0 - vec2(0.5, 0.5));
        float d = dot(q1,q1);
        if (d < 1.0) {
          vec3 n = normalize(vec3(q1, sqrt(1.0-d)));
 20       vec2 q = q1 / (1.0 + sqrt(1.0 - d));
          vec2 q2 = scale2 * q + spin2;
          vec2 frac = q2;
          // linear interpolation
          vec3 m = texture(tex, vec3(frac, p)).rgb;
 25       // output
          float a = 1.0;
          float k = gl_FragCoord.z - 0.1 * cos(sqrt(d) * 3.1415926*0.5);
          float c = 0.0625 + pow(dot(vec3(0.0, 0.0, 1.0), n), 2.0);
          vec3 rgb = m * c;
 30       gl_FragDepth = k;
          gl_FragColor = vec4(rgb, a);
        } else {
          discard;
        }
 35   }
```

26 src/borderwindow.h

```
      /* ================================================================
      rdex -- reaction-diffusion explorer
      Copyright (C) 2008,2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
```

```
------------------------------------------------------------------------
Border/Window Shader
============================================================================ */

#ifndef BORDERWINDOW_H
#define BORDERWINDOW_H 1

#include "shader.h"

//============================================================================
// borderwindow shader data
struct borderwindow { struct shader shader;
  struct {
    GLint origin, scale, rotate, translate, near, far; // vertex
    GLint texture, pack, size; // fragment
  } uniform;
  struct {
    float origin[4], scale[4], rotate[16], translate[4], near[4], far[4]; // ↙
        ↳ vertex
    int texture; float pack, size; // fragment
  } value;
  struct {
    GLint delta, spin, pick; // vertex
  } attr;
};

struct borderwindow_pick { struct shader shader;
  struct {
    GLint origin, scale, rotate, translate, near, far; // vertex
  } uniform;
  struct {
    float origin[4], scale[4], rotate[16], translate[4], near[4], far[4]; // ↙
        ↳ vertex
  } value;
  struct {
    GLint delta; // vertex
  } attr;
};

//============================================================================
// prototypes
struct borderwindow *borderwindow_init(struct borderwindow *borderwindow);
void borderwindow_use(struct borderwindow *borderwindow);

struct borderwindow_pick *borderwindow_pick_init(struct borderwindow_pick *↙
    ↳ borderwindow_pick);
void borderwindow_pick_use(struct borderwindow_pick *borderwindow_pick);

#endif
```

27 src/borderwindow_pick.frag

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------------------
Sphere shader for ball picking
```

```
================================================================================ */

varying float ignore;

void main(void) {
  vec2 p = gl_TexCoord[0].xy;                    // float in [0..1]
  vec2 q0 = gl_TexCoord[0].zw;
  vec2 q1 = 2.0 * (q0 - vec2(0.5, 0.5));
  float d = dot(q1,q1);
  if (ignore < 0.5 && d < 1.0) {
    float k = gl_FragCoord.z - 0.1 * cos(sqrt(d) * 3.1415926*0.5);
    gl_FragDepth = k;
    gl_FragColor = vec4(p, 0.0, 1.0);
  } else {
    discard;
  }
}
```

28 src/camera.c

```
/* ================================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------------
Chase camera
================================================================================ */

/*

The camera is modelled by some point masses connected together:

    T  = target
    V* = vertices of camera cage

where:

    massV* << massT

Each V is connected to T by a stretchy spring of appropriate rest length.
Each V is connected to each other V by a stiff spring of appropriate rest
length.

The camera rotation matrix is:

    C = sum(V*) / count(V*)
    M = normalize([ V0, V1, V2, V3 ] - [ C, C, C, C ])

Updating the view is peformed by updating the mass spring damper model.

*/

#include <math.h>
#include <string.h>
#include "camera.h"

void camera_init(struct camera *camera) {
  memset(camera, 0, sizeof(*camera));
```

```
  camera->dt = 0.004;
  // camera scale
  float r = 1.0 / 16.0;
  // initialize target point
  camera->T.position.v[3] = r;
  camera->T.mass = 1 << 20;
  // initialize camera points
  camera->V[0].position.v[0] = r;
  camera->V[1].position.v[1] = r;
  camera->V[2].position.v[2] = r;
  camera->V[3].position.v[3] = r;
  camera->V[4].position.v[0] = -r;
  camera->V[5].position.v[1] = -r;
  camera->V[6].position.v[2] = -r;
  camera->V[7].position.v[3] = -r;
  camera->r = r;
  for (int i = 0; i < 8; ++i) {
    camera->V[i].mass = 1.0 / 8.0; // + 4.0 * ((i == 3) || (i == 7));
  }
  // initialize external links
  for (int i = 0; i < 8; ++i) {
    camera->TV[i].p = &camera->T;
    camera->TV[i].q = &camera->V[i];
    camera->TV[i].length = vdistance4(&camera->T.position, &camera->V[i].↙
        ↳ position);
    camera->TV[i].k = 1;
    camera->TV[i].c = 2 * sqrt(camera->TV[i].k); // * camera->V[i].mass);
  }
  // initialize internal links
  int k = 0;
  for (int i = 0; i < 8 - 1; ++i) {
    for (int j = i + 1; j < 8; ++j) {
      camera->VV[k].p = &camera->V[i];
      camera->VV[k].q = &camera->V[j];
      camera->VV[k].length = vdistance4(&camera->V[i].position, &camera->V[j].↙
          ↳ position);
      camera->VV[k].k = 1;
      camera->VV[k].c = 2 * sqrt(camera->VV[k].k * sqrt(camera->V[i].mass * ↙
          ↳ camera->V[j].mass));
      k++;
    }
  }
  // reset position
  for (int i = 0; i < 8; ++i) {
    camera->V[i].position.v[3] -= r;
  }
}

void camera_force(struct link *l) {
  struct vec4 zero = { { 0, 0, 0, 0 } };
  struct vec4 f, dir, ndir, dv;
  float ldir, x, v;
  vcopy4(&f, &zero);
  /* dir  = l->p->position - l->q->position
   * ldir = length(dir)
   * ndir = dir / ldir
   * x    = l->length - ldir
```

```
     * v    = dot(l->p->velocity - l->q->velocity, ndir)
     * f    = - (k x + c v)
     */
    vvsub4(&dir, &l->p->position, &l->q->position);
    ldir = vlength4(&dir);
    if (! (ldir > 0)) {
      return;
    }
    x = ldir - l->length;
    vsdiv4(&ndir, &dir, ldir);
    vvsub4(&dv, &l->p->velocity, &l->q->velocity);
    v = vvdot4(&dv, &ndir);
    vsmul4(&f, &ndir, l->k * x + l->c * v);
    vvsub4(&l->p->force, &l->p->force, &f);
    vvadd4(&l->q->force, &l->q->force, &f);
  }

  void camera_action1(struct mass *m, float dt) {
    struct vec4 a, dv;
    vsdiv4(&a, &m->force, m->mass);
    vsmul4(&dv, &a, dt);
    vvadd4(&m->velocity, &m->velocity, &dv);
  }

  void camera_friction(struct mass *m, struct vec4 *v0, struct vec4 *v0dir) {
    struct vec4 dv;
    vvsub4(&dv, &m->velocity, v0);
    vsmul4(&m->velocity, v0, 0.995);
    vvadd4(&m->velocity, &m->velocity, &dv);
  //  vsmul4(&m->velocity, &m->velocity, 0.99);
  }

  void camera_action2(struct mass *m, float dt) {
    struct vec4 dp;
    vsmul4(&dp, &m->velocity, dt);
    vvadd4(&m->position, &m->position, &dp);
  }

  void camera_update(struct camera *camera, struct vec4 *target) {
    // set target
    struct vec4 zero = { { 0, 0, 0, 0 } };
    vvmul4(&camera->T.position, target, &camera->scale);
    vcopy4(&camera->T.velocity, &zero);
    // reset forces
    vcopy4(&camera->T.force, &zero);
    for (int i = 0; i < 8; ++i) {
      vcopy4(&camera->V[i].force, &zero);
    }
    // accumulate forces from links
    for (int i = 0; i < 8; ++i) {
      camera_force(&camera->TV[i]);
    }
    for (int i = 0; i < 28; ++i) {
      camera_force(&camera->VV[i]);
    }
    // convert forces to movement
    for (int i = 0; i < 8; ++i) {
```

```
    camera_action1(&camera->V[i], camera->dt);
  }
  struct vec4 v0, v0dir;
  vcopy4(&v0, &zero);
  for (int i = 0; i < 8; ++i) {
    vvadd4(&v0, &v0, &camera->V[i].velocity);
  }
  vsdiv4(&v0, &v0, 8.0);
  float vl = vlength4(&v0);
  vl = vl > 0 ? vl : 1.0;
  vsdiv4(&v0dir, &v0, vl);
  for (int i = 0; i < 8; ++i) {
    camera_friction(&camera->V[i], &v0, &v0dir);
  }
  for (int i = 0; i < 8; ++i) {
    camera_action2(&camera->V[i], camera->dt);
  }
  // calculate camera center
  vcopy4(&camera->C, &zero);
  for (int i = 0; i < 8; ++i) {
    vvadd4(&camera->C, &camera->C, &camera->V[i].position);
  }
  vsdiv4(&camera->C, &camera->C, 8);
  // calculate camera matrix
  struct vec4 v[8];
  for (int i = 0; i < 8; ++i) {
    vvsub4(&v[i], &camera->V[i].position, &camera->C);
  }
  int k = 0;
  for (int i = 0; i < 4; ++i) {
    float d = vdistance4(&v[i], &v[i+4]);
    for (int j = 0; j < 4; ++j) {
      camera->M.m[k++] = (v[i].v[j] - v[i+4].v[j]) / d; //(n + m);
    }
  }
  vcopy4(&camera->C, &camera->V[3].position);
  vvdiv4(&camera->C, &camera->C, &camera->scale);
}

// EOF
```

29 src/camera.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Chase camera
================================================================ */

#ifndef CAMERA_H
#define CAMERA_H 1

#include "matrix.h"

struct mass {
  struct vec4 position;
```

```
  struct vec4 velocity;
  struct vec4 force;
  float mass;
};

struct link {
  struct mass *p;
  struct mass *q;
  float length;
  float k;
  float c;
};

struct camera {
  struct mass T;
  struct mass V[8];
  struct link TV[8];
  struct link VV[28];
  struct vec4 C;
  struct mat4 M;
  float dt;
  float r;
  struct vec4 scale;
};

void camera_init(struct camera *camera);
void camera_update(struct camera *camera, struct vec4 *target);

#endif
```

30 src/copysquare.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Copy And Square Shader
================================================================ */

#include "util.h"
#include "copysquare.h"
#include "copysquare.frag.c"

//================================================================
// copysquare shader initialization
struct copysquare *copysquare_init(struct copysquare *copysquare) {
  if (! copysquare) { return 0; }
  if (! shader_init(&copysquare->shader, 0, copysquare_frag)) {
    return 0;
  }
  shader_uniform(copysquare, texture);
  copysquare->value.texture = 0;
  glGenTextures(1, &copysquare->texture);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, copysquare->texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

```
  glDisable(GL_TEXTURE_2D);
  copysquare->width = 0;
  copysquare->height = 0;
  return copysquare;
}

//=============================================================
// copysquare shader reshape callback
void copysquare_reshape(struct copysquare *copysquare, int w, int h) {
  copysquare->width  = roundtwo(w);
  copysquare->height = roundtwo(h);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, copysquare->texture);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
    copysquare->width, copysquare->height, 0, GL_RGBA, GL_FLOAT, 0
  );
  glDisable(GL_TEXTURE_2D);
}

//=============================================================
// copysquare shader display callback
void copysquare_display(
  struct copysquare *copysquare, GLuint fbo, GLuint texture
) {
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, texture);
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  glFramebufferTexture2DEXT(
    GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
    copysquare->texture, 0
  );
  glUseProgramObjectARB(copysquare->shader.program);
  shader_updatei(copysquare ,texture);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0, 1, 0, 1);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glViewport(0, 0, copysquare->width, copysquare->height);
  glBegin(GL_QUADS); { glColor4f(1,1,1,1);
    glTexCoord2f(0, 0); glVertex2f(0, 0);
    glTexCoord2f(1, 0); glVertex2f(1, 0);
    glTexCoord2f(1, 1); glVertex2f(1, 1);
    glTexCoord2f(0, 1); glVertex2f(0, 1);
  } glEnd();
  glUseProgramObjectARB(0);
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  glDisable(GL_TEXTURE_2D);
}

//=============================================================
// copysquare shader idle callback
void copysquare_idle(struct copysquare *copysquare) { }

// EOF
```

31 src/copysquare.frag

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
================================================================ */

uniform sampler2D texture;                          // (U,V,_,_)

void main(void) {
  vec2 p = gl_TexCoord[0].st;
  vec2 c = texture2D(texture, p).rg;
  gl_FragColor = vec4(c.r,c.r*c.r,c.g,c.g*c.g);     // (U,U^2,V,V^2)
}
```

32 src/copysquare.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Copy And Square Shader
================================================================ */

#ifndef COPYSQUARE_H
#define COPYSQUARE_H 1

#include "shader.h"

//================================================================
// copysquare shader data
struct copysquare { struct shader shader;
  struct { GLint texture; } uniform;
  struct { int   texture; } value;
  GLuint texture;
  int width;
  int height;
};

//================================================================
// prototypes
struct copysquare *copysquare_init(struct copysquare *copysquare);
void copysquare_reshape(struct copysquare *copysquare, int w, int h);
void copysquare_display(
  struct copysquare *copysquare, GLuint fbo, GLuint texture
);
void copysquare_idle(struct copysquare *copysquare);

#endif
```

33 src/difference.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
```

```
5   Difference Shader
    ================================================================== */

    #include "util.h"
    #include "difference.h"
10  #include "difference.frag.c"

    //====================================================================
    // difference shader initialization
    struct difference *difference_init(struct difference *difference) {
15    if (! difference) { return 0; }
      if (! shader_init(&difference->shader, 0, difference_frag)) {
        return 0;
      }
      shader_uniform(difference, texture1);
20    shader_uniform(difference, texture2);
      difference->value.texture1 = 0;
      difference->value.texture2 = 1;
      if (! arithmeticmean_init(&difference->arithmeticmean)) { return 0; }
      glGenTextures(2, difference->textures);
25    glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, difference->textures[0]);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glBindTexture(GL_TEXTURE_2D, difference->textures[1]);
30    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glDisable(GL_TEXTURE_2D);
      difference->width = 0;
      difference->height = 0;
35    difference->snap = 0;
      difference->calc = 0;
      difference->mean = 0;
      return difference;
    }
40

    //====================================================================
    // difference shader reshape callback
    void difference_reshape(struct difference *difference, int w, int h) {
      difference->width  = roundtwo(w);
45    difference->height = roundtwo(h);
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, difference->textures[0]);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
        difference->width, difference->height, 0, GL_RGBA, GL_FLOAT, 0
50    );
      glBindTexture(GL_TEXTURE_2D, difference->textures[1]);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
        difference->width, difference->height, 0, GL_RGBA, GL_FLOAT, 0
      );
55    glDisable(GL_TEXTURE_2D);
      arithmeticmean_reshape(&difference->arithmeticmean, w, h);
    }

    //====================================================================
60  // difference shader display callback
    void difference_display(
```

```
  struct difference *difference, GLuint fbo, GLuint texture
) {
  if (difference->snap) {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture);
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
    glFramebufferTexture2DEXT(
      GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
      difference->textures[0], 0
    );
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, difference->width, difference->height);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glDisable(GL_TEXTURE_2D);
    difference->snap = 0;
  } else if (difference->calc) {
    glEnable(GL_TEXTURE_2D);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, texture);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, difference->textures[0]);
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
    glFramebufferTexture2DEXT(
      GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
      difference->textures[1], 0
    );
    glUseProgramObjectARB(difference->shader.program);
    shader_updatei(difference, texture1);
    shader_updatei(difference, texture2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, difference->width, difference->height);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glUseProgramObjectARB(0);
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glDisable(GL_TEXTURE_2D);
    arithmeticmean_display(
      &difference->arithmeticmean, fbo, difference->textures[1]
    );
```

```
    difference->mean = difference->arithmeticmean.result[0];
    difference->calc = 0;
  }
}

//=============================================================================
// difference shader idle callback
void difference_idle(struct difference *difference) {
  arithmeticmean_idle(&difference->arithmeticmean);
}

// EOF
```

34 src/difference.frag

```
/* =============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
============================================================================= */

uniform sampler2D texture1;       // first texture
uniform sampler2D texture2;       // second texture

void main(void) {
  vec2 p = gl_TexCoord[0].st;      // texel coordinates
  vec4 d = texture2D(texture1, p)  // difference between channels
         - texture2D(texture2, p);
  float q = sqrt(dot(d,d));        // length of the difference vector
  gl_FragColor = vec4(q, q, q, 1.0); // output
}
```

35 src/difference.h

```
/* =============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
-----------------------------------------------------------------------------
Difference Shader
============================================================================= */

#ifndef DIFFERENCE_H
#define DIFFERENCE_H 1

#include "shader.h"
#include "arithmeticmean.h"

//=============================================================================
// difference shader data
struct difference { struct shader shader;
  struct { GLint texture1; GLint texture2; } uniform;
  struct { int   texture1; GLint texture2; } value;
  struct arithmeticmean arithmeticmean;
  GLuint textures[2];
  int width;
  int height;
  int snap;
```

```
  int calc;
  float mean;
};

//=============================================================================
// prototypes
struct difference *difference_init(struct difference *difference);
void difference_reshape(struct difference *difference, int w, int h);
void difference_display(
  struct difference *difference, GLuint fbo, GLuint texture
);
void difference_idle(struct difference *difference);

#endif
```

36 src/expcolour.c

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Exponential colour space conversion
============================================================================ */

#include "expcolour.h"
#include "expcolour.frag.c"

//=============================================================================
// shader initialization
struct expcolour *expcolour_init(struct expcolour *expcolour) {
  if (! expcolour) { return 0; }
  if (! shader_init(&expcolour->shader, 0, expcolour_frag)) {
    return 0;
  }
  shader_uniform(expcolour, texture);
  expcolour->value.texture = 0;
  return expcolour;
}

// EOF
```

37 src/expcolour.frag

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Exponential colour space conversion
============================================================================ */

uniform sampler2D texture;

void main(void) {
  vec2 p = gl_TexCoord[0].st;
  vec4 x = texture2D(texture, p);
  float l = dot(vec3(0.3, 0.5, 0.2), x.rgb);
```

```
  float a;
  if (l > 0.5) {
    a = 1.0;
  } else {
    a = 0.0;
  }
  if (l > 0.0) {
    x /= l;
    l *= 2.0;
    l = exp(l) / (l + 1.0) * 0.4;
    x *= l;
  }
  gl_FragColor = vec4(x.rgb, a);
}
```

38 src/expcolour.h

```
/* ===========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
---------------------------------------------------------------------------
Exponential colour space conversion
=========================================================================== */

#ifndef EXPCOLOUR_H
#define EXPCOLOUR_H 1

#include "shader.h"

//============================================================================
// shader data
struct expcolour { struct shader shader;
  struct { GLint texture; } uniform;
  struct { int   texture; } value;
};

//============================================================================
// prototypes
struct expcolour *expcolour_init(struct expcolour *expcolour);

#endif
```

39 src/falsecolour.c

```
/* ===========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
---------------------------------------------------------------------------
False Colour Shader
=========================================================================== */

#include "util.h"
#include "falsecolour.h"
#include "falsecolour.frag.c"

//============================================================================
```

```
// false colour shader initialization
struct falsecolour *falsecolour_init(struct falsecolour *falsecolour) {
  if (! falsecolour) { return 0; }
  if (! shader_init(&falsecolour->shader, 0, falsecolour_frag)) {
    return 0;
  }
  if (! blur_init(&falsecolour->blur)) { return 0; }
  shader_uniform(falsecolour, texture);
  falsecolour->value.texture = 0;
  glGenTextures(1, &falsecolour->texture0);
  glGenTextures(1, &falsecolour->texture);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, falsecolour->texture0);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, falsecolour->texture);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, 0);
  glDisable(GL_TEXTURE_2D);
  falsecolour->width = 0;
  falsecolour->height = 0;
  return falsecolour;
}

//============================================================
// false colour shader reshape callback
void falsecolour_reshape(
  struct falsecolour *falsecolour, int w, int h
) {
  falsecolour->width  = roundtwo(w);
  falsecolour->height = roundtwo(h);
  falsecolour->blur.value.d[0] = 0.5 / falsecolour->width;
  falsecolour->blur.value.d[1] = 0.5 / falsecolour->height;
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, falsecolour->texture0);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
    falsecolour->width * 2, falsecolour->height * 2, 0, GL_RGBA, GL_FLOAT, 0
  );
  glBindTexture(GL_TEXTURE_2D, falsecolour->texture);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
    falsecolour->width, falsecolour->height, 0, GL_RGBA, GL_FLOAT, 0
  );
  glBindTexture(GL_TEXTURE_2D, 0);
  glDisable(GL_TEXTURE_2D);
}

//============================================================
// false colour shader display callback
void falsecolour_display(
  struct falsecolour *falsecolour, GLuint fbo, GLuint texture
) {
  glEnable(GL_TEXTURE_2D);
```

```
    glViewport(0, 0, falsecolour->width * 2, falsecolour->height * 2);
    glBindTexture(GL_TEXTURE_2D, texture);
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
    glFramebufferTexture2DEXT(
      GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
      falsecolour->texture0, 0
    );
    glUseProgramObjectARB(falsecolour->shader.program);
    shader_updatei(falsecolour, texture);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glUseProgramObjectARB(0);
    glBindTexture(GL_TEXTURE_2D, 0);
    // linear interpolation
    glViewport(0, 0, falsecolour->width, falsecolour->height);
    glFramebufferTexture2DEXT(
      GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
      falsecolour->texture, 0
    );
    glUseProgramObjectARB(falsecolour->blur.shader.program);
    shader_updatei(&(falsecolour->blur), texture);
    shader_updatef2(&(falsecolour->blur), d);
    glBindTexture(GL_TEXTURE_2D, falsecolour->texture0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glUseProgramObjectARB(0);
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glBindTexture(GL_TEXTURE_2D, 0);
    glDisable(GL_TEXTURE_2D);
}

//=============================================================================
// false colour shader idle callback
void falsecolour_idle(struct falsecolour *falsecolour) {
}

// EOF
```

40 src/falsecolour.frag

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010  Claude Heiland-Allen <claude@mathr.co.uk>
================================================================ */

uniform sampler2D texture; // U := r, V := g, other channels ignored

void main(void) {
  const mat3 lcc2rgb = mat3(                    // LC1C2 => RGB
     1.0,  1.407,  0.0,                         // colour space
     1.0, -0.677, -0.236,                       // conversion matrix
     1.0,  0.0,    1.848
  );
  vec2 p = gl_TexCoord[0].st;                   // texel coordinates
  vec2 q;
  float u, v, h, l;
  vec3 c;
  // point to colour
  q = texture2D(texture, p).rg;
  u = 1.0 - q.r;
  v = q.g;
  h = 16.0*atan(v, u);
  l = clamp(cos(16.0*(v*v + u*u)), 0.0, 1.0);
  c = vec3(l, 0.5*cos(h), 0.5*sin(h));
  gl_FragColor = vec4(c * lcc2rgb, 1.0);
}
```

41 src/falsecolour.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
False Colour Shader
================================================================ */

#ifndef FALSECOLOUR_H
#define FALSECOLOUR_H 1

#include "shader.h"
#include "blur.h"

//================================================================
// false colour shader data
struct falsecolour { struct shader shader;
  struct { GLint  texture; } uniform;
  struct { int    texture; } value;
  GLuint texture0, texture;
  int width;
  int height;
  struct blur blur;
};

//================================================================
```

```
// protoypes
struct falsecolour *falsecolour_init(struct falsecolour *falsecolour);
void falsecolour_reshape(
  struct falsecolour *falsecolour, int w, int h
);
void falsecolour_display(
  struct falsecolour *falsecolour, GLuint fbo, GLuint texture
);
void falsecolour_idle(struct falsecolour *falsecolour);

#endif
```

42 src/font.png

43 src/fonts.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Text bitmap fonts
================================================================ */

#include <GL/glew.h>
#include "fonts.h"

extern char _binary_font_rgba_start;

struct fonts *fonts_init(struct fonts *fonts) {
  fonts->size = 512;
  fonts->pack = 8;
  glGenTextures(1, &fonts->texture);
  glBindTexture(GL_TEXTURE_2D, fonts->texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

```
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, fonts->size, fonts->size, 0, GL_RGBA, ↙
      ↳ GL_UNSIGNED_BYTE, &_binary_font_rgba_start);
  return fonts;
}
```

44 src/fonts.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Text bitmap fonts
================================================================ */

#ifndef FONTS_H
#define FONTS_H 1

#include <GL/glew.h>

struct fonts {
  GLuint texture;
  int size;
  int pack;
};

struct fonts *fonts_init(struct fonts *fonts);

#endif
```

45 src/gblur.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Multi-scale Gaussian Blur
================================================================ */

#include "util.h"
#include "gblur.h"
#include "gblur.frag.c"

//================================================================
// initialization
struct gblur *gblur_init(struct gblur *gblur) {
  if (!gblur) { return 0; }
  if (!shader_init(&gblur->shader, 0, gblur_frag)) {
    return 0;
  }
  if (!expcolour_init(&gblur->expcolour)) {
    return 0;
  }
  shader_uniform(gblur, texture);
  shader_uniform(gblur, d);
  gblur->value.texture = 0;
```

```
25    gblur->value.d[0] = 0;
      gblur->value.d[1] = 0;
      glGenTextures(gblur_tex_max, &gblur->textures[0][0]);
      glGenTextures(gblur_tex_max, &gblur->textures[1][0]);
      glEnable(GL_TEXTURE_2D);
30    for (int i = 0; i < gblur_tex_max; ++i) {
        for (int c = 0; c < 2; ++c) {
          glBindTexture(GL_TEXTURE_2D, gblur->textures[c][i]);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
35        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
        }
      }
      glDisable(GL_TEXTURE_2D);
40    return gblur;
    }

    //====================================================================
    // reshape callback
45  void gblur_reshape(struct gblur *gblur, int w, int h) {
      // allocate textures
      int w2 = roundtwo(w);
      int h2 = roundtwo(h);
      int d = w2>h2 ? w2 : h2;
50    gblur->count = logtwo(d);
      glEnable(GL_TEXTURE_2D);
      for (int i = 0; i <= gblur->count; ++i) {
        for (int c = 0; c < 2; ++c) {
          glBindTexture(GL_TEXTURE_2D, gblur->textures[c][i]);
55        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 1<<i, 1<<i, 0, GL_RGBA, ↙
              ↳ GL_UNSIGNED_BYTE, 0);
          //glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, 1<<i, 1<<i, 0, GL_RGBA, ↙
              ↳ GL_FLOAT, 0);
        }
      }
      glDisable(GL_TEXTURE_2D);
60  }

    //====================================================================
    // display callback
    void gblur_display(struct gblur *gblur, GLuint fbo, GLuint texture) {
65    // copy texture
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, texture);
      //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
      glMatrixMode(GL_PROJECTION);
70    glLoadIdentity();
      gluOrtho2D(0, 1, 0, 1);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glViewport(0, 0, 1<<gblur->count, 1<<gblur->count);
75    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
          ↳ GL_TEXTURE_2D, gblur->textures[0][gblur->count], 0);
      glUseProgramObjectARB(gblur->expcolour.shader.program);
      shader_updatei(&gblur->expcolour, texture);
      glBegin(GL_QUADS); { glColor4f(1,1,1,1);
```

```
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    // multi-scale blur
    glUseProgramObjectARB(gblur->shader.program);
    for (int i = gblur->count - 1; i >= 0; --i) {
      // blur horizontally
      glBindTexture(GL_TEXTURE_2D, gblur->textures[0][i+1]);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0, 1, 0, 1);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glViewport(0, 0, 1<<(i+1), 1<<(i+1));
      glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
          ↳ GL_TEXTURE_2D, gblur->textures[1][i+1], 0);
      gblur->value.d[0] = 1.0 / (1<<(i+1));
      gblur->value.d[1] = 0.0;
      shader_updatei(gblur, texture);
      shader_updatef2(gblur, d);
      glBegin(GL_QUADS); { glColor4f(1,1,1,1);
        glTexCoord2f(0, 0); glVertex2f(0, 0);
        glTexCoord2f(1, 0); glVertex2f(1, 0);
        glTexCoord2f(1, 1); glVertex2f(1, 1);
        glTexCoord2f(0, 1); glVertex2f(0, 1);
      } glEnd();
      // blur vertically (and downscale)
      glBindTexture(GL_TEXTURE_2D, gblur->textures[1][i+1]);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0, 1, 0, 1);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glViewport(0, 0, 1<<i, 1<<i);
      glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
          ↳ GL_TEXTURE_2D, gblur->textures[0][i], 0);
      gblur->value.d[0] = 0.0;
      gblur->value.d[1] = 1.0 / (1<<(i+1));
      shader_updatei(gblur, texture);
      shader_updatef2(gblur, d);
      glBegin(GL_QUADS); { glColor4f(1,1,1,1);
        glTexCoord2f(0, 0); glVertex2f(0, 0);
        glTexCoord2f(1, 0); glVertex2f(1, 0);
        glTexCoord2f(1, 1); glVertex2f(1, 1);
        glTexCoord2f(0, 1); glVertex2f(0, 1);
      } glEnd();
    }
    // clean up
    glUseProgramObjectARB(0);
    //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glBindTexture(GL_TEXTURE_2D, 0);
    glDisable(GL_TEXTURE_2D);
}

// EOF
```

46 src/gblur.frag

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Separated Gaussian Blur Shader
============================================================================ */

uniform sampler2D texture;
uniform vec2 d;

// weights
const float w0 = 0.402619947, w1 = 0.244201342, w2 = 0.054488685;

void main(void) {
  vec2 p = gl_TexCoord[0].st;
  vec4 o = vec4(0.0, 0.0, 0.0, 0.0);
  o += w2 * texture2D(texture, p - 2.0 * d);
  o += w1 * texture2D(texture, p -       d);
  o += w0 * texture2D(texture, p          );
  o += w1 * texture2D(texture, p +       d);
  o += w2 * texture2D(texture, p + 2.0 * d);
  gl_FragColor = o;
}
```

47 src/gblur.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Multi-scale Gaussian Blur
============================================================================ */

#ifndef GBLUR_H
#define GBLUR_H 1

#include "shader.h"
#include "expcolour.h"

//============================================================================
// maximum number of textures to allocate (16 => 65536x65536 max size)
#define gblur_tex_max 16

//============================================================================
// shader data
struct gblur { struct shader shader;
  struct { GLint texture; GLint d; } uniform;
  struct { int   texture; float d[2]; } value;
  int count;
  // textures[0] is blur output
  // textures[1] is temporary
  GLuint textures[2][gblur_tex_max];
  struct expcolour expcolour;
};
```

```
//============================================================
// prototypes
struct gblur *gblur_init(struct gblur *gblur);
void gblur_reshape(struct gblur *gblur, int w, int h);
void gblur_display(struct gblur *gblur, GLuint fbo, GLuint texture);

#endif
```

48 src/glyph-curve-less.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="128" height="128">
<g fill="none" stroke-linecap="round">
 <g stroke="#000000" stroke-width="8">
  <path d="M 16 64 A 48 48 0 1 1 64 112" />
  <path d="M 16 64 A 24 24 0 1 1 40 88" />
 </g>
 <g stroke="#0000ff" stroke-width="4">
  <path d="M 16 64 A 24 24 0 1 1 40 88" />
 </g>
 <g stroke="#ffff00" stroke-width="4">
  <path d="M 16 64 A 48 48 0 1 1 64 112" />
 </g>
 <g stroke="#000000" stroke-width="8">
  <path d="M 88 64 L 40 64 M 72 80 L 88 64 M 72 48 L 88 64" />
 </g>
 <g stroke="#00ff00" stroke-width="4">
  <path d="M 88 64 L 40 64 M 72 80 L 88 64 M 72 48 L 88 64" />
 </g>
</g>
</svg>
```

49 src/glyph-curve-more.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="128" height="128">
<g fill="none" stroke-linecap="round">
 <g stroke="#000000" stroke-width="8">
  <path d="M 16 64 A 48 48 0 1 1 64 112" />
  <path d="M 16 64 A 24 24 0 1 1 40 88" />
 </g>
 <g stroke="#0000ff" stroke-width="4">
  <path d="M 16 64 A 48 48 0 1 1 64 112" />
 </g>
 <g stroke="#ffff00" stroke-width="4">
  <path d="M 16 64 A 24 24 0 1 1 40 88" />
 </g>
 <g stroke="#000000" stroke-width="8">
  <path d="M 88 64 L 40 64 M 56 80 L 40 64 M 56 48 L 40 64" />
 </g>
 <g stroke="#00ff00" stroke-width="4">
  <path d="M 88 64 L 40 64 M 56 80 L 40 64 M 56 48 L 40 64" />
 </g>
```

```
</g>
</svg>
```

50 src/glyphs.c

```
/* ========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------------------
Sequencer glyphs
======================================================================== */

#include <GL/glew.h>
#include "glyphs.h"

extern char _binary_glyph_curve_less_rgba_start;
extern char _binary_glyph_curve_more_rgba_start;
extern char _binary_glyph_speed_less_rgba_start;
extern char _binary_glyph_speed_more_rgba_start;

struct glyphs *glyphs_init(struct glyphs *glyphs) {
  glGenTextures(4, &glyphs->textures[0]);
  // curve less
  glBindTexture(GL_TEXTURE_2D, glyphs->textures[sequence_curve_less]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 128, 128, 0, GL_RGBA, GL_UNSIGNED_BYTE↙
      ↳ , &_binary_glyph_curve_less_rgba_start);
  // curve more
  glBindTexture(GL_TEXTURE_2D, glyphs->textures[sequence_curve_more]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 128, 128, 0, GL_RGBA, GL_UNSIGNED_BYTE↙
      ↳ , &_binary_glyph_curve_more_rgba_start);
  // speed less
  glBindTexture(GL_TEXTURE_2D, glyphs->textures[sequence_speed_less]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 128, 128, 0, GL_RGBA, GL_UNSIGNED_BYTE↙
      ↳ , &_binary_glyph_speed_less_rgba_start);
  // speed more
  glBindTexture(GL_TEXTURE_2D, glyphs->textures[sequence_speed_more]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 128, 128, 0, GL_RGBA, GL_UNSIGNED_BYTE↙
      ↳ , &_binary_glyph_speed_more_rgba_start);
  return glyphs;
}
```

51 src/glyphs.h

```
/* ========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------------------
Sequencer glyphs
```

```
   ═══════════════════════════════════════════════════════════════ */

#ifndef GLYPHS_H
#define GLYPHS_H 1

#include <GL/glew.h>

struct glyphs {
  GLuint textures[4];
};

struct glyphs *glyphs_init(struct glyphs *glyphs);

#define sequence_curve_less 0
#define sequence_curve_more 1
#define sequence_speed_less 2
#define sequence_speed_more 3

#endif
```

52 src/glyph-speed-less.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/↙
    ↳ xlink" width="128" height="128">
<g fill="none" stroke-linecap="round">
 <g stroke="#000000" stroke-width="8">
  <path d="M 24 24 L 104 24" />
  <path d="M 40 104 L 88 104" />
 </g>
 <g stroke="#0000ff" stroke-width="4">
  <path d="M 24 24 L 104 24" />
 </g>
 <g stroke="#ffff00" stroke-width="4">
  <path d="M 40 104 L 88 104" />
 </g>
 <g stroke="#000000" stroke-width="8">
  <path d="M 64 88 L 64 40 M 80 72 L 64 88 M 48 72 L 64 88" />
 </g>
 <g stroke="#00ff00" stroke-width="4">
  <path d="M 64 88 L 64 40 M 80 72 L 64 88 M 48 72 L 64 88" />
 </g>
</g>
</svg>
```

53 src/glyph-speed-more.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/↙
    ↳ xlink" width="128" height="128">
<g fill="none" stroke-linecap="round">
 <g stroke="#000000" stroke-width="8">
  <path d="M 24 24 L 104 24" />
  <path d="M 40 104 L 88 104" />
 </g>
 <g stroke="#0000ff" stroke-width="4">
```

```
  <path d="M 40 104 L 88 104" />
 </g>
 <g stroke="#ffff00" stroke-width="4">
  <path d="M 24 24 L 104 24" />
 </g>
 <g stroke="#000000" stroke-width="8">
  <path d="M 64 40 L 64 88 M 80 56 L 64 40 M 48 56 L 64 40" />
 </g>
 <g stroke="#00ff00" stroke-width="4">
  <path d="M 64 40 L 64 88 M 80 56 L 64 40 M 48 56 L 64 40" />
 </g>
</g>
</svg>
```

54 src/histogram.c

```
/* ========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2017  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------------------
CPU-Based Image Histograms
======================================================================== */

#include "histogram.h"

void histogram_calc(struct histogram *h, const struct image *i) {
  for (int r = 0; r < HISTOGRAM_BUCKETS; ++r) {
  for (int g = 0; g < HISTOGRAM_BUCKETS; ++g) {
  for (int b = 0; b < HISTOGRAM_BUCKETS; ++b) {
    h->bucket[r][g][b] = 0;
  }
  }
  }
  for (int y = 0; y < i->height; ++y) {
  for (int x = 0; x < i->width; ++x) {
    int r = HISTOGRAM_BUCKETS * I(i,x,y,0);
    int g = HISTOGRAM_BUCKETS * I(i,x,y,1);
    int b = HISTOGRAM_BUCKETS * I(i,x,y,2);
    if (r < 0) r = 0;
    if (r >= HISTOGRAM_BUCKETS) r = HISTOGRAM_BUCKETS - 1;
    if (g < 0) g = 0;
    if (g >= HISTOGRAM_BUCKETS) g = HISTOGRAM_BUCKETS - 1;
    if (b < 0) b = 0;
    if (b >= HISTOGRAM_BUCKETS) b = HISTOGRAM_BUCKETS - 1;
    h->bucket[r][g][b] += 1;
  }
  }
  float pixels = i->width * i->height;
  for (int r = 0; r < HISTOGRAM_BUCKETS; ++r) {
  for (int g = 0; g < HISTOGRAM_BUCKETS; ++g) {
  for (int b = 0; b < HISTOGRAM_BUCKETS; ++b) {
    h->bucket[r][g][b] /= pixels;
  }
  }
  }
}
```

55 src/histogram.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
CPU-Based Image Histograms
============================================================================ */

#ifndef HISTOGRAM_H
#define HISTOGRAM_H 1

#include "image.h"

#define HISTOGRAM_BUCKETS 8
struct histogram {
  float bucket[HISTOGRAM_BUCKETS][HISTOGRAM_BUCKETS][HISTOGRAM_BUCKETS];
};

void histogram_calc(struct histogram * h, const struct image *i);

#endif
```

56 src/image.c

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
CPU-Based Images
============================================================================ */

#include <stdlib.h>
#include <GL/glew.h>
#include "image.h"

struct image *image_alloc(int width, int height, int channels) {
  struct image *image =
    malloc(sizeof(struct image) + width*height*channels*sizeof(float));
  if (! image) { return 0; }
  image->width = width;
  image->height = height;
  image->channels = channels;
  image->data = (float *) (image+1);
  return image;
}

void image_free(struct image *image) {
  free(image);
}

void image_copytexture(struct image *image, GLuint texture) {
  glBindTexture(GL_TEXTURE_2D, texture);
  glGetTexImage(GL_TEXTURE_2D, 0, GL_RGB, GL_FLOAT, image->data); // FIXME check
    ↳ data is big enough for the image!!!!
}
```

57 src/image.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
CPU-Based Images
============================================================================ */

#ifndef IMAGE_H
#define IMAGE_H 1

#include <GL/glew.h>

struct image {
  int width;
  int height;
  int channels;
  float *data;
};

#define I(i,x,y,z) ((i)->data[(i)->channels*((i)->width*(y)+(x))+(z)])

struct image *image_alloc(int width, int height, int channels);
void image_free(struct image *image);
void image_copytexture(struct image *image, GLuint texture);

#endif
```

58 src/interleave31.c

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

int main(int argc, char **argv) {
  if (argc != 4) {
    return 1;
  }
  int d = atoi(argv[3]);
  FILE *in3 = fopen(argv[1], "rb");
  if (in3) {
    FILE *in1 = fopen(argv[2], "rb");
    if (in1) {
      for (int i = 0; i < d*d; ++i) {
        putchar(getc(in3));
        putchar(getc(in3));
        putchar(getc(in3));
        putchar(getc(in1));
      }
      fclose(in1);
    } else {
      return 1;
    }
    fclose(in3);
  } else {
    return 1;
```

```
  }
  return 0;
}
```

59 src/intro.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Intro module
================================================================ */

#include <math.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "intro.h"

extern char _binary_intro_image_rgba_start;

struct intro *intro_init(struct intro *intro) {
  glGenTextures(1, &intro->tex);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, intro->tex);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, INTRO_IMAGE_SIZE, INTRO_IMAGE_SIZE, 0,↙
      ↳ GL_RGBA, GL_UNSIGNED_BYTE, &_binary_intro_image_rgba_start);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glBindTexture(GL_TEXTURE_2D, 0);
  glDisable(GL_TEXTURE_2D);
  intro->state = intro_idle;
  intro->phase = 0;
  return intro;
}

void intro_reshape(struct intro *intro, int w, int h) {
  intro->width = w;
  intro->height = h;
}

void intro_display(struct intro *intro, double dt) { /* FIXME aspect ratio? */
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  switch (intro->state) {
    case intro_idle:
    case intro_active:
      intro->rx = 0;//32.0 * exp(-32.0 * (intro->phase * intro->phase) / (↙
          ↳ INTRO_TIME * INTRO_TIME));
      intro->ry = 0;//32.0 * exp(-16.0 * (intro->phase * intro->phase) / (↙
          ↳ INTRO_TIME * INTRO_TIME));
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glViewport(0, 0, intro->width, intro->height);
      gluOrtho2D(0, 1, 0, 1);
      glMatrixMode(GL_MODELVIEW);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, intro->tex);
```

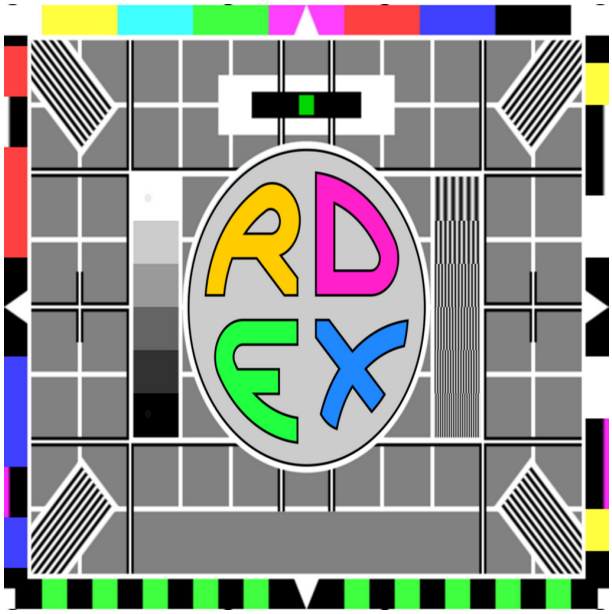
```
      glBegin(GL_QUADS); {
        glColor4f(1,1,1,1);
        glTexCoord2f(0+intro->rx, 1+intro->ry); glVertex2f(0, 0);
        glTexCoord2f(0+intro->rx, 0+intro->ry); glVertex2f(0, 1);
        glTexCoord2f(1+intro->rx, 0+intro->ry); glVertex2f(1, 1);
        glTexCoord2f(1+intro->rx, 1+intro->ry); glVertex2f(1, 0);
      } glEnd();
      glDisable(GL_TEXTURE_2D);
      if (intro->state == intro_active) {
        intro->phase += dt;
        if (! (intro->phase < INTRO_TIME)) {
          intro->state = intro_done;
          intro->phase = 0;
        }
      }
      break;
    default:
      break;
  }
}

void intro_start(struct intro *intro) {
  if (intro->state == intro_idle) {
    intro->state = intro_active;
    intro->phase = 0;
  }
}
```

60 src/intro.h

```
/* ========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------------------
Intro module
======================================================================== */

#ifndef INTRO_H
#define INTRO_H 1

#include <GL/glew.h>
#include <GL/glut.h>

#define INTRO_TIME 5.0
#define INTRO_IMAGE_SIZE 1024

struct intro {
  GLuint tex;
  int width;
  int height;
  enum { intro_idle, intro_active, intro_done } state;
  double phase;
  double rx;
  double ry;
};

struct intro *intro_init(struct intro *intro);
```

```
void intro_reshape(struct intro *intro, int w, int h);
void intro_display(struct intro *intro, double dt);
void intro_start(struct intro *intro);

#endif
```

61 src/intro-image.png



62 src/library.c

```
/* ================================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010,2011,2019  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------------
Library Module
================================================================================ */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wordexp.h>
#include <math.h>
#include "library.h"
#include "background.h"
#include "rdex.h"

static void library_copy_texture_to_array(struct library *library, int layer) {
  glBindTexture(GL_TEXTURE_2D, library->texture);
  glBindTexture(GL_TEXTURE_2D_ARRAY, library->texture_array);
  glBindBuffer(GL_PIXEL_PACK_BUFFER, library->pbo);
  glBindBuffer(GL_PIXEL_UNPACK_BUFFER, library->pbo);
```

```
  for (int level = 0, size = library_tex_size; size > 0; level += 1, size >>= 1)↙
      ↳ {
    glGetTexImage(GL_TEXTURE_2D, level, GL_RGB, GL_UNSIGNED_BYTE, 0);
    glTexSubImage3D(GL_TEXTURE_2D_ARRAY, level, 0, 0, layer, size, size, 1, ↙
        ↳ GL_RGB, GL_UNSIGNED_BYTE, 0);
  }
  glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
  glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
}

// find index to insert at
int library_index(struct library *library, float score) {
  if (library->count == library_tex_layers) {
    float minscore = score;
    int minindex = -1;
    for (int j = 0; j < library_tex_layers; ++j) {
      if (library->array[j].score <= minscore) {
        minscore = library->array[j].score;
        minindex = j;
      }
    }
    return minindex;
  } else {
    return library->count++;
  }
}

// pick a random point weighted by score
int library_pick(struct library *library, float *ru, float *rv, float *f, float ↙
    ↳ *k, float randomness) {
  double p = randomness * library->score * (rand() / (double) RAND_MAX);
  double s = 0.0;
  for (int i = 0; i < library->count; ++i) {
    s += library->array[i].score;
    if (p <= s) {
      *ru = library->array[i].v[0];
      *rv = library->array[i].v[1];
      *f  = library->array[i].v[2];
      *k  = library->array[i].v[3];
      return 0;
    }
  }
  return 1;
}

struct library *library_init(struct library *library, GLuint fbo) {
  if (!library) { return 0; }
  if (! borderwindow_init(&library->borderwindow)) { return 0; }
  if (! borderwindow_pick_init(&library->borderwindow_pick)) { return 0; }
  if (! worldsphere_init(&library->worldsphere)) { return 0; }
  if (! blur_init(&library->blur)) { return 0; }
  //if (! bloom_init(&library->bloom)) { return 0; }
  library->count = 0;
  library->frame = 0;
  camera_init(&library->camera);
  library->width = 1;
  library->height = 1;
```

```
  library->mousex = -1;
  library->mousey = -1;
  library->picked = -1;
  library->hover = -1;
  library->snap = 0;
  library->session = getenv("RDEX_SESSION");
  if (! library->session) { library->session = "."; }
  background_init(&library->background, library->session); //FIXME -- curl global init must be before any threads are created
  { // allocate texture array
    glGenTextures(1, &library->texture_array);
    glBindTexture(GL_TEXTURE_2D_ARRAY, library->texture_array);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    size_t bytes = library_tex_layers * library_tex_size * library_tex_size * 3;
    char *white = malloc(bytes);
    memset(white, 255, bytes);
    glTexImage3D(
      GL_TEXTURE_2D_ARRAY, 0, GL_RGB,
      library_tex_size, library_tex_size, library_tex_layers, 0,
      GL_RGB, GL_UNSIGNED_BYTE, white
    );
    glGenerateMipmap(GL_TEXTURE_2D_ARRAY);
    // allocate pixel buffer
    glGenBuffers(1, &library->pbo);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, library->pbo);
    glBufferData(GL_PIXEL_PACK_BUFFER, library_tex_size * library_tex_size * 4, 0, GL_DYNAMIC_COPY);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
    // allocate texture
    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &library->texture);
    glBindTexture(GL_TEXTURE_2D, library->texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexImage2D(
      GL_TEXTURE_2D, 0, GL_RGB,
      library_tex_size, library_tex_size, 0,
      GL_RGB, GL_UNSIGNED_BYTE, white
    );
    glGenerateMipmap(GL_TEXTURE_2D);
    free(white);
  }
  library->state = library_state_idle;
  library->phase = 0;
  library->loaded = 0;
  return library;
}

void library_insert(
  struct library *library,
  GLfloat ru, GLfloat rv, GLfloat f, GLfloat k, int index, GLfloat score
) {
  assert(0 <= index && index < library->count);
  // remove old element's stats
```

```
130     library->score -= library->array[index].score;
        library->score2 -= library->array[index].score * library->array[index].score;
        library->scorev[0] -= library->array[index].score * library->array[index].v↙
            ↳ [0];
        library->scorev[1] -= library->array[index].score * library->array[index].v↙
            ↳ [1];
        library->scorev[2] -= library->array[index].score * library->array[index].v↙
            ↳ [2];
135     library->scorev[3] -= library->array[index].score * library->array[index].v↙
            ↳ [3];
        library->scorev2[0] -= library->array[index].score * library->array[index].v↙
            ↳ [0] * library->array[index].v[0];
        library->scorev2[1] -= library->array[index].score * library->array[index].v↙
            ↳ [1] * library->array[index].v[1];
        library->scorev2[2] -= library->array[index].score * library->array[index].v↙
            ↳ [2] * library->array[index].v[2];
        library->scorev2[3] -= library->array[index].score * library->array[index].v↙
            ↳ [3] * library->array[index].v[3];
140     // fill element
        library->array[index].score = score;
        library->array[index].v[0] = ru;
        library->array[index].v[1] = rv;
        library->array[index].v[2] = f;
145     library->array[index].v[3] = k;
        library->array[index].spin[0] = 0;
        library->array[index].spin[1] = 0;
        library->array[index].dspin[0] = (rand() / (double) RAND_MAX - 0.5) / 32.0;
        library->array[index].dspin[1] = (rand() / (double) RAND_MAX - 0.5) / 32.0;
150     // add new element's stats
        library->score += library->array[index].score;
        library->score2 += library->array[index].score * library->array[index].score;
        library->scorev[0] += library->array[index].score * library->array[index].v↙
            ↳ [0];
        library->scorev[1] += library->array[index].score * library->array[index].v↙
            ↳ [1];
155     library->scorev[2] += library->array[index].score * library->array[index].v↙
            ↳ [2];
        library->scorev[3] += library->array[index].score * library->array[index].v↙
            ↳ [3];
        library->scorev2[0] += library->array[index].score * library->array[index].v↙
            ↳ [0] * library->array[index].v[0];
        library->scorev2[1] += library->array[index].score * library->array[index].v↙
            ↳ [1] * library->array[index].v[1];
        library->scorev2[2] += library->array[index].score * library->array[index].v↙
            ↳ [2] * library->array[index].v[2];
160     library->scorev2[3] += library->array[index].score * library->array[index].v↙
            ↳ [3] * library->array[index].v[3];
        // compute stats
        library->stddev[0] = sqrt((library->scorev2[0] * library->score - library->↙
            ↳ scorev[0] * library->scorev[0]) / (library->score * library->score - ↙
            ↳ library->score2));
        library->stddev[1] = sqrt((library->scorev2[1] * library->score - library->↙
            ↳ scorev[1] * library->scorev[1]) / (library->score * library->score - ↙
            ↳ library->score2));
        library->stddev[2] = sqrt((library->scorev2[2] * library->score - library->↙
            ↳ scorev[2] * library->scorev[2]) / (library->score * library->score - ↙
            ↳ library->score2));
```

```
165     library->stddev[3] = sqrt((library->scorev2[3] * library->score - library->↙
          ↳ scorev[3] * library->scorev[3]) / (library->score * library->score - ↙
          ↳ library->score2));
    }

    void library_addimage(struct library *library, GLuint fbo, GLuint t, int index) ↙
        ↳ {
      glEnable(GL_TEXTURE_2D);
170   glDisable(GL_DEPTH_TEST);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glViewport(0, 0, library_tex_size, library_tex_size);
      gluOrtho2D(0, 1, 0, 1);
175   glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
      // combine
      glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
          ↳ GL_TEXTURE_2D, library->texture, 0);
180   // packed texture
      glBindTexture(GL_TEXTURE_2D, t);
      glBegin(GL_QUADS); {
        glColor4f(1,1,1,1);
        glTexCoord2f(0, 0); glVertex2f(0, 0);
185     glTexCoord2f(1, 0); glVertex2f(1, 0);
        glTexCoord2f(1, 1); glVertex2f(1, 1);
        glTexCoord2f(0, 1); glVertex2f(0, 1);
      } glEnd();
      glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
190   glBindTexture(GL_TEXTURE_2D, library->texture);
      glGenerateMipmap(GL_TEXTURE_2D);
      library_copy_texture_to_array(library, index);
      glBindTexture(GL_TEXTURE_2D, 0);
    }
195
    int library_load(struct library *library, GLuint fbo) {
      glEnable(GL_TEXTURE_2D);
      /* temporary texture for loading */
      GLuint t;
200   glGenTextures(1, &t);
      glBindTexture(GL_TEXTURE_2D, t);
    //  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    //  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
205   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      wordexp_t p;
      const char *pattern2 = "/ru_*__rv_*__f_*__k_*__s_*.ppm";
      int patternlen = strlen(library->session) + strlen(pattern2) + 1;
      char *pattern = malloc(patternlen);
210   snprintf(pattern, patternlen, "%s%s", library->session, pattern2);
      if (0 == wordexp(pattern, &p, WRDE_NOCMD | WRDE_SHOWERR | WRDE_UNDEF)) {
        char sru[9]; char srv[9]; char sf[9]; char sk[9]; char ss[9];
        for (int i = 0; i < p.we_wordc; ++i) {
          int l = strlen(p.we_wordv[i]);
215       const char *pattern3 = "ru_%8c__rv_%8c__f_%8c__k_%8c__s_%8c.ppm";
          const int expectLen = 3 + 8 + 5 + 8 + 4 + 8 + 4 + 8 + 4 + 8 + 4;
          if (
```

```
        l >= expectLen &&
        5 == sscanf(p.we_wordv[i] + l - expectLen, pattern3, sru, srv, sf, sk, ↙
            ↳ ss)
      ) {
        sru[8] = '\0'; srv[8] = '\0'; sf[8] = '\0'; sk[8] = '\0'; ss[8] = '\0';
        double ru, rv, f, k, s;
        int bsize = rdex_tex_size * rdex_tex_size * 3;
        char buffer[rdex_tex_size * rdex_tex_size * 3];
        if (1 != sscanf(sru, "%lg", &ru)) { break; }
        if (1 != sscanf(srv, "%lg", &rv)) { break; }
        if (1 != sscanf(sf,  "%lg", &f))  { break; }
        if (1 != sscanf(sk,  "%lg", &k))  { break; }
        if (1 != sscanf(ss,  "%lg", &s))  { break; }
        FILE *image = fopen(p.we_wordv[i], "rb");
        if (!image) { break; }
        fseek(image, -bsize, SEEK_END);
        if (fread(buffer, bsize, 1, image) != 1) {
          fprintf(stderr, "warning: failed to read image?\n");
        }
        fclose(image);
        glBindTexture(GL_TEXTURE_2D, t);
        glTexImage2D(
          GL_TEXTURE_2D, 0, GL_RGB,
          rdex_tex_size, rdex_tex_size, 0,
          GL_RGB, GL_UNSIGNED_BYTE, buffer
        );
        int index = library_index(library, s);
        if (index >= 0) {
          library_addimage(library, fbo, t, index);
          library_insert(library, ru, rv, f, k, index, s);
        }
      }
    }
    wordfree(&p);
  }
  free(pattern);
  glDeleteTextures(1, &t);
  glDisable(GL_TEXTURE_2D);
  return 1;
}

void library_display_save(
  struct library *library, GLuint fbo, GLuint texture, GLuint rawtexture,
  double ru, double rv, double f, double k, double score, const char *behaviour
) {
  library->target.v[0] = ru;
  library->target.v[1] = rv;
  library->target.v[2] = f;
  library->target.v[3] = k;
  // capture to library
  if (! library->snap) { return; }
  library->snap = 0;
  int index = library_index(library, score);
  if (index < 0) {
    return;
  }
  // downscale part of the original texture into the new texture
```

```
  glEnable(GL_TEXTURE_2D);
  library_addimage(library, fbo, texture, index);
/*
  // copy thumbnail texture to main memory as 1-byte data
  char *thumb_ppm = malloc(library_tex_size * library_tex_size * 3 + 1024);
  int thumb_ppm_bytes;
  thumb_ppm_bytes = snprintf(thumb_ppm, 1000, "P6\n%d %d 255\n", library_tex_size, library_tex_size);
  glReadPixels((index % library_pack) * library_tex_size, (index / library_pack) * library_tex_size, library_tex_size, library_tex_size, GL_RGB, GL_UNSIGNED_BYTE, thumb_ppm + thumb_ppm_bytes);
  thumb_ppm_bytes += library_tex_size * library_tex_size * 3;
*/
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  // copy original texture to main memory as 1-byte data
  char *full_ppm = malloc(rdex_tex_size * rdex_tex_size * 3 + 1024);
  int full_ppm_bytes;
  full_ppm_bytes = snprintf(full_ppm, 1000, "P6\n%d %d 255\n", rdex_tex_size, rdex_tex_size);
  glBindTexture(GL_TEXTURE_2D, texture);
  glGetTexImage(GL_TEXTURE_2D, 0, GL_RGB, GL_UNSIGNED_BYTE, full_ppm + full_ppm_bytes);
  full_ppm_bytes += rdex_tex_size * rdex_tex_size * 3;
  // copy original raw texture to main memory as float data
  struct image *raw_image = image_alloc(rdex_tex_size, rdex_tex_size, 3);
  image_copytexture(raw_image, rawtexture);
  // copy original texture to main memory as float data
  struct image *full_image = image_alloc(rdex_tex_size, rdex_tex_size, 3);
  image_copytexture(full_image, texture);
  // pass all these images to the background thread, which will free them
  background_enqueue(&library->background, ru, rv, f, k, score, behaviour,
    /*thumb_ppm, thumb_ppm_bytes,*/ full_ppm, full_ppm_bytes, raw_image, full_image);
  // store in the library
  library_insert(library, ru, rv, f, k, index, score);
  glDisable(GL_TEXTURE_2D);
}

void library_reshape(struct library *library, int w, int h) {
  library->width = w;
  library->height = h;
//  bloom_reshape(&library->bloom, 1024, 1024); // FIXME hardcoded
}

void library_display(struct library *library, GLuint fbo, GLuint texture) {
  switch (library->state) {
    case library_state_intro:
      if (! library->loaded) {
        library_load(library, fbo);
        library->loaded = 1;
      } else {
        library->state = library_state_active;
      }
      // fall-through
    case library_state_active: {
      glEnable(GL_TEXTURE_2D);
      glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```

```
325     /*
          glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
          glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
              ↳ GL_TEXTURE_2D, library->balltex, 0);
        */
          glViewport(0, 0, library->width, library->height);
330
          // update camera
          for (int i = 0; i < 4; ++i) {
            library->camera.scale.v[i] = library->stddev[i] > 0 ? 2.0 / library->↙
                ↳ stddev[i] : 1.0;
          }
335       for (int i = 0; i < 10; ++i) camera_update(&library->camera, &library->↙
              ↳ target);
          for (int i = 0; i < 4; ++i) {
            library->borderwindow.value.origin[i] = library->camera.C.v[i];
            library->borderwindow_pick.value.origin[i] = library->camera.C.v[i];
          }
340       for (int i = 0; i < 16; ++i) {
            library->borderwindow.value.rotate[i] = library->camera.M.m[i];
            library->borderwindow_pick.value.rotate[i] = library->camera.M.m[i];
          }
          for (int i = 0; i < 4; ++i) {
345         library->borderwindow.value.scale[i] = library->camera.scale.v[i];
            library->borderwindow_pick.value.scale[i] = library->camera.scale.v[i];
          }

          glClearColor(0,0,0,0);
350       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        //  int width, height;
          glEnable(GL_TEXTURE_2D);
          glBindTexture(GL_TEXTURE_2D, texture);
355       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
          glMatrixMode(GL_PROJECTION);
          glLoadIdentity();
          gluOrtho2D(0, 1, 0, 1);
360       glMatrixMode(GL_MODELVIEW);
          glLoadIdentity();
        //  glViewport(0, 0, library->width, library->height);
        //  glGetTexLevelParameteriv(
        //    GL_TEXTURE_2D, 0, GL_TEXTURE_WIDTH, &width
365     //  );
        //  glGetTexLevelParameteriv(
        //    GL_TEXTURE_2D, 0, GL_TEXTURE_HEIGHT, &height
        //  );
          library->worldsphere.value.size[0] = rdex_tex_size; //1024; //width;
370       library->worldsphere.value.size[1] = rdex_tex_size; //height;
          worldsphere_use(&library->worldsphere);
          glBegin(GL_QUADS); {
            float x = 3.0;
            float y = 3.0 * library->height / library->width;
375         glColor4f(1,1,1,1);
            glTexCoord4f(0.5 - x, 0.5 - y,  x, -y); glVertex2f(0, 0);
            glTexCoord4f(0.5 + x, 0.5 - y, -x, -y); glVertex2f(1, 0);
            glTexCoord4f(0.5 + x, 0.5 + y, -x,  y); glVertex2f(1, 1);
```

```
      glTexCoord4f(0.5 - x, 0.5 + y,   x,   y); glVertex2f(0, 1);
380   } glEnd();
      worldsphere_use(0);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

385   // draw balls
      glEnable(GL_DEPTH_TEST);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      float v = library->width / (float) library->height;
390   glFrustum(-v, v, -1, 1, 4, 12);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glTranslatef(0, 0, -8);
      glScalef(2, 2, 2);
395   borderwindow_use(&library->borderwindow);
      float a = library->height / (float) library->width;
      for (int i = 0; i < library->count; ++i) {
        library->array[i].spin[0] += library->array[i].dspin[0];
        library->array[i].spin[0] -= floor(library->array[i].spin[0]);
400     library->array[i].spin[1] += library->array[i].dspin[1];
        library->array[i].spin[1] -= floor(library->array[i].spin[1]);
        GLfloat z  = sqrt(library->array[i].score) + 1.0/256.0;
        // note: y texcoord flipped for correct display from loaded image files
        int dx[4] = { 0, 1, 1, 0 };
405     int dy[4] = { 1, 1, 0, 0 };
        for (int j = 0; j < 4; ++j) {
          for (int k = 0; k < 4; ++k) {
            library->varray[i][j].vertex[k] = library->array[i].v[k];
          }
410       library->varray[i][j].texcoord[0] = i;
          library->varray[i][j].texcoord[1] = 0;
          library->varray[i][j].texcoord[2] = dx[j];
          library->varray[i][j].texcoord[3] = dy[j];
          library->varray[i][j].delta[0] = (dx[j] * 2 - 1) * z * a;
415       library->varray[i][j].delta[1] = (1 - 2 * dy[j]) * z;
          library->varray[i][j].spin[0] = library->array[i].spin[0];
          library->varray[i][j].spin[1] = library->array[i].spin[1];
        }
      }
420   glEnableClientState(GL_VERTEX_ARRAY);
      glEnableClientState(GL_TEXTURE_COORD_ARRAY);
      glEnableVertexAttribArray(library->borderwindow.attr.delta);
      glEnableVertexAttribArray(library->borderwindow.attr.spin);
      glVertexPointer  (4, GL_FLOAT, sizeof(struct libvert), &library->varray↙
          ↳ [0][0].vertex[0]);
425   glTexCoordPointer(4, GL_FLOAT, sizeof(struct libvert), &library->varray↙
          ↳ [0][0].texcoord[0]);
      glVertexAttribPointer(library->borderwindow.attr.delta, 2, GL_FLOAT, ↙
          ↳ GL_FALSE, sizeof(struct libvert), &library->varray[0][0].delta[0]);
      glVertexAttribPointer(library->borderwindow.attr.spin,  2, GL_FLOAT, ↙
          ↳ GL_FALSE, sizeof(struct libvert), &library->varray[0][0].spin[0]);
      glDrawArrays(GL_QUADS, 0, library->count * 4);
      glDisableClientState(GL_VERTEX_ARRAY);
430   glDisableClientState(GL_TEXTURE_COORD_ARRAY);
      glDisableVertexAttribArray(library->borderwindow.attr.delta);
```

```
        glDisableVertexAttribArray(library->borderwindow.attr.spin);

        // reset texture mode for accurate image save
435     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glBindTexture(GL_TEXTURE_2D, 0);
      //  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
        borderwindow_use(0);
440     glDisable(GL_DEPTH_TEST);

        // apply bloom filter (modifies texture in place)
       // bloom_display(&library->bloom, fbo, library->balltex);
      /*
445     // draw balls
        glViewport(0, 0, 1024, 576);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 1, 0, 1);
450     glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, library->balltex);
        glBegin(GL_QUADS); {
455       float x = 0.5;
          float y = 0.5 * 576.0/1024.0; // FIXME hardcoded
          glColor4f(1,1,1,1);
          glTexCoord2f(0.5 - x, 0.5 - y); glVertex2f(0, 0);
          glTexCoord2f(0.5 + x, 0.5 - y); glVertex2f(1, 0);
460       glTexCoord2f(0.5 + x, 0.5 + y); glVertex2f(1, 1);
          glTexCoord2f(0.5 - x, 0.5 + y); glVertex2f(0, 1);
        } glEnd();
      */
        glDisable(GL_TEXTURE_2D);
465

#if 0
        /* draw cursor */
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
470     gluOrtho2D(0, library->width, 0, library->height);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glBegin(GL_LINES); {
          glColor4f(0,1,0,1);
475       glVertex2f(library->mousex, 0);
          glVertex2f(library->mousex, library->height);
          glVertex2f(0, library->mousey);
          glVertex2f(library->width, library->mousey);
        } glEnd();
480 #endif

        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
        library->frame += 1;
        break;
485   }

#if 0
    case library_state_intro: {
```

```
        if (! library->loaded) {
          library_load(library, fbo);
          library->loaded = 1;
        }
        glEnable(GL_TEXTURE_2D);
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
        glViewport(0, 0, library->width, library->height);
        glClearColor(0,0,1,1);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 1, 0, 1);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        library->worldsphere.value.size[0] = rdex_tex_size;
        library->worldsphere.value.size[1] = rdex_tex_size;
        worldsphere_use(&library->worldsphere);
        glBegin(GL_QUADS); {
          float x0 = 3.0;
          float y0 = x0 * library->height / library->width;
          float x1 = 3.0 / (library->phase * library->phase + 0.001);
          float y1 = x1 * library->height / library->width;
          glColor4f(1,1,1,1);
          glTexCoord4f(0.5 - x0, 0.5 - y0,  x1, -y1); glVertex2f(0, 0);
          glTexCoord4f(0.5 + x0, 0.5 - y0, -x1, -y1); glVertex2f(1, 0);
          glTexCoord4f(0.5 + x0, 0.5 + y0, -x1,  y1); glVertex2f(1, 1);
          glTexCoord4f(0.5 - x0, 0.5 + y0,  x1,  y1); glVertex2f(0, 1);
        } glEnd();
        worldsphere_use(0);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        library->phase += 0.04 / 25.0;
        if (! (library->phase < 1.0)) {
          library->state = library_state_active;
          library->phase = 0;
        }
        glBindTexture(GL_TEXTURE_2D, 0);
        glDisable(GL_TEXTURE_2D);
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
        library->frame += 1;
        break;
      }
#endif
      case library_state_outro: {
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
        glViewport(0, 0, library->width, library->height);
        glClearColor(0,0,1,1);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
        break;
      }
      case library_state_idle:
      case library_state_done: {
```

```
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
        glViewport(0, 0, library->width, library->height);
        glClearColor(0,0,0,1);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
        break;
      }
    }
  }

void library_pmotion(struct library *library, int x, int y) {
  library->mousex = x;
  library->mousey = library->height - y - 1;
}

void library_amotion(struct library *library, int x, int y) {
  library->mousex = x;
  library->mousey = library->height - y - 1;
}

void library_mouse(struct library *library, int button, int state, int x, int y)
    {
  if (state == GLUT_DOWN) {
    library->picked = library->hover;
  }
}

void library_idle(struct library *library) {
  worldsphere_idle(&library->worldsphere);
}

// EOF
```

63 src/library.h

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010,2019  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------
Library Module
========================================================================== */

#ifndef LIBRARY_H
#define LIBRARY_H 1

#include <GL/glew.h>
#include "borderwindow.h"
#include "worldsphere.h"
#include "background.h"
#include "matrix.h"
#include "camera.h"
//#include "bloom.h"
#include "blur.h"

//==========================================================================
// library data structures
```

```
#define library_tex_size 256
#define library_tex_layers 512

struct libelem {
  GLfloat v[4];       // raw parameter coordinates
  float score;        // analysis result
  float spin[2];      // spinning orbs
  float dspin[2];
};

struct libvert {
  GLfloat vertex[4];
  GLfloat texcoord[4];
  GLfloat delta[2];
  GLfloat spin[2];
};

struct library {
  int loaded;
  int count;          // invariant: count <= library_tex_layers
  struct libelem array[library_tex_layers];
  double score, score2, scorev[4], scorev2[4];
  GLfloat stddev[4];
  GLuint texture, texture_array;
  GLuint pbo;
  int which;
  unsigned int frame;
  int width;
  int height;
  int mousex;
  int mousey;
  GLuint picktex;
  int hover;
  int picked;
  int snap;
  char *session;
  struct borderwindow borderwindow;
  struct borderwindow_pick borderwindow_pick;
  struct worldsphere worldsphere;
  struct background background;
  struct camera camera;
  struct vec4 target;
//  struct bloom bloom;
  struct blur blur;
  GLuint blurtex[1]; // FIXME need log2(rdex_tex_size / library_tex_size)-1
//  GLuint balltex;
  enum { library_state_idle,
         library_state_intro,
         library_state_active,
         library_state_outro,
         library_state_done } state;
  double phase;
  // draw arrays stuff
  struct libvert varray[library_tex_layers][4];
};

//============================================================
```

```
80  // prototypes

    struct library *library_init(struct library *library, GLuint fbo);
    int library_load(struct library *library, GLuint fbo);
    void library_display_save(
85    struct library *library, GLuint fbo, GLuint texture, GLuint rawtexture,
      double ru, double rv, double f, double k, double score, const char *behaviour
    );
    void library_reshape(struct library *library, int w, int h);
    void library_display(struct library *library, GLuint fbo, GLuint texture);
90  void library_pmotion(struct library *library, int x, int y);
    void library_amotion(struct library *library, int x, int y);
    void library_mouse(struct library *library, int button, int state, int x, int y)↙
        ↳ ;
    void library_idle(struct library *library);
    int library_pick(struct library *library, float *ru, float *rv, float *f, float ↙
        ↳ *k, float randomness);
95
    #endif
```

64 src/list.c

```
    /* ================================================================
    rdex -- reaction-diffusion explorer
    Copyright (C) 2009  Claude Heiland-Allen <claude@mathr.co.uk>
    ----------------------------------------------------------------
 5  doubly-linked list implementation
    ================================================================ */

    #include <assert.h>
    #include "list.h"
10
    void list_init(struct list *l) {
      assert(l);
      l->head = &(l->headNode);
      l->tail = &(l->tailNode);
15    l->head->pred = 0;
      l->head->next = l->tail;
      l->tail->pred = l->head;
      l->tail->next = 0;
    }
20
    int list_ishead(struct node *n) {
      assert(n);
      return !n->pred;
    }
25
    int list_istail(struct node *n) {
      assert(n);
      return !n->next;
    }
30
    int list_isempty(struct list *l) {
      assert(l && l->head && l->tail);
      return l->head->next == l->tail;
    }
35
```

```
int list_length(struct list *l) {
  struct node *n = l->head->next;
  int i = 0;
  while (n != l->tail) {
    n = n->next;
    i++;
  }
  return i;
}

void list_remove(struct node *n) {
  assert(n && n->pred && n->next);
  n->pred->next = n->next;
  n->next->pred = n->pred;
  n->next = 0;
  n->pred = 0;
}

void list_insertbefore(struct node *n, struct node *beforethis) {
  assert(n && beforethis && beforethis->pred);
  n->next = beforethis;
  n->pred = beforethis->pred;
  beforethis->pred->next = n;
  beforethis->pred = n;
}

void list_insertafter(struct node *n, struct node *afterthis) {
  assert(n && afterthis && afterthis->next);
  n->next = afterthis->next;
  n->pred = afterthis;
  afterthis->next->pred = n;
  afterthis->next = n;
}

void list_inserttail(struct list *l, struct node *n) {
  assert(l);
  list_insertbefore(n, l->tail);
}


struct node *list_removehead(struct list *l) {
  assert(l && ! list_isempty(l));
  struct node *n = l->head->next;
  list_remove(n);
  return n;
}

// EOF
```

65 src/list.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2009,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
doubly-linked list implementation
================================================================ */
```

```
#ifndef LIST_H
#define LIST_H 1

struct node {
  struct node *next;
  struct node *pred;
};

struct list {
  struct node *head;
  struct node *tail;
  // private
  struct node headNode;
  struct node tailNode;
};

void list_init(struct list *l);
int list_ishead(struct node *n);
int list_istail(struct node *n);
int list_isempty(struct list *l);
int list_length(struct list *l);
void list_remove(struct node *n);
void list_insertbefore(struct node *n, struct node *beforethis);
void list_insertafter(struct node *n, struct node *afterthis);
void list_inserttail(struct list *l, struct node *n);
struct node *list_removehead(struct list *l);

#endif
```

66 src/main.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010,2017,2019  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Main Program Entry Point
================================================================ */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <GL/glew.h>
#include <GL/glut.h>

#ifdef RDEX_ICON
#include <GL/glx.h>
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/Shell.h>
#ifdef VMS
#include <X11/shape.h>
#else
#include <X11/extensions/shape.h>
#endif
```

```
25  #include <X11/xpm.h>
    #endif

    #include "rdex.h"

30  #ifdef RDEX_ICON
    #include "rdex-logo-32x32.xpm"

    struct XpmIcon {
      Pixmap pixmap;
35    Pixmap mask;
      XpmAttributes attributes;
    };
    #endif

40  int main(int argc, char **argv) {
      /* initialize GLUT etc */
      fprintf(stderr, "rdex (GPL) 2008-2019 Claude Heiland-Allen <claude@mathr.co.uk↙
          ↳ >\n");
      srand(time(NULL));
      glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
45    glutInitWindowSize(788, 576);
      glutInit(&argc, argv);
      glutCreateWindow("rdex");
      glutFullScreen();
      glutSetCursor(GLUT_CURSOR_NONE);
50    glewInit();
    #ifdef RDEX_ICON
      /* set icon */
      Display *display = glXGetCurrentDisplay();
      if (display) {
55      GLXDrawable drawable = glXGetCurrentDrawable();
        if (drawable) {
          struct XpmIcon icon;
          if (0 == XpmCreatePixmapFromData(display, drawable, rdex_logo_32x32_xpm, &↙
              ↳ icon.pixmap, &icon.mask, NULL)) {
            XWMHints* hints = XAllocWMHints();
60          if (hints) {
              hints->flags = IconPixmapHint;
              hints->icon_pixmap = icon.pixmap;
              if (icon.mask) {
                hints->flags |= IconMaskHint;
65              hints->icon_mask = icon.mask;
              }
              XSetWMHints(display, drawable, hints);
              XFree(hints);
            }
70        }
        }
      }
    #endif
      /* activate callbacks and enter main loop */
75    if (rdex_init()) {
        glutDisplayFunc(rdex_display);
        glutReshapeFunc(rdex_reshape);
        glutTimerFunc(1, rdex_timer, 0);
    //    glutIdleFunc(rdex_idle);
```

```
80        glutPassiveMotionFunc(rdex_pmotion);
          glutMotionFunc(rdex_amotion);
          glutMouseFunc(rdex_mouse);
          glutKeyboardFunc(rdex_keynormal);
          glutSpecialFunc(rdex_keyspecial);
85        glutSetCursor(GLUT_CURSOR_NONE);
          glutMainLoop();
          return 0; // never reached
        } else {
          return 1;
90      }
    }

    // EOF
```

67 src/Makefile

```
    #=============================================================================
    # rdex -- reaction-diffusion explorer
    # Copyright (C) 2008,2009,2010,2011,2017,2019  Claude Heiland-Allen <↙
        ↪ claude@mathr.co.uk>
    #-----------------------------------------------------------------------------
 5  # Makefile
    #=============================================================================

    CC = gcc
    CFLAGS = -std=c99 -Wall -pedantic -O3 -pthread `curl-config --cflags` `pkg-↙
        ↪ config --cflags jack` -D"_POSIX_C_SOURCE=199309L"
10  LIBS = -lGL -lGLU -lglut -lGLEW `curl-config --libs` `pkg-config --libs jack` -↙
        ↪ lsndfile -lm
    OBJS = main.o rdex.o shader.o arithmeticmean.o audio.o background.o blur.o ↙
        ↪ borderwindow.o copysquare.o difference.o falsecolour.o histogram.o image.o↙
        ↪  library.o list.o numericerror.o pfifo.o reactiondiffusion.o screenshot.o ↙
        ↪ segment.o tamura.o util.o worldsphere.o matrix.o camera.o gblur.o bloom.o ↙
        ↪ expcolour.o sequence.o glyph-curve-less.o glyph-curve-more.o glyph-speed-↙
        ↪ less.o glyph-speed-more.o glyphs.o intro.o intro-image.o timeline.o font.o↙
        ↪  fonts.o text.o
    GENC = arithmeticmean.frag.c audio.frag.c blur.frag.c borderwindow.frag.c ↙
        ↪ borderwindow_pick.frag.c copysquare.frag.c difference.frag.c falsecolour.↙
        ↪ frag.c numericerror.frag.c reactiondiffusion.frag.c worldsphere.frag.c ↙
        ↪ projection.vert.c projection_pick.vert.c gblur.frag.c bloom.frag.c ↙
        ↪ expcolour.frag.c sequence.frag.c sequence.vert.c text.frag.c text.vert.c

    # optional feature: window icon
15  CFLAGS += -DRDEX_ICON
    LIBS += -lXpm -lX11
    # optional feature: debuggability
    #CFLAGS += -ggdb -pg
    CFLAGS += -march=native -mfpmath=sse -ffast-math
20
    all: rdex

    clean:
            -rm -f $(OBJS) $(GENC)
25
    # no suffix rules, they cause weird issues with the .frag.c files
    .SUFFIXES:
```

```
.PHONY: all clean

# link program
rdex: $(OBJS)
	$(CC) $(CFLAGS) -o rdex $(OBJS) $(LIBS)

# C source to object file
%.o: %.c
	$(CC) $(CFLAGS) -o $@ -c $<

# shader source to C source
%.frag.c: %.frag s2c.sh
	./s2c.sh $*_frag < $< > $@

%.vert.c: %.vert s2c.sh
	./s2c.sh $*_vert < $< > $@

# svg to rasterized RGBA object
%.o: %.svg svg2o.sh interleave31
	./svg2o.sh $*

# png to rasterized RGBA object
%.o: %.png png2o.sh interleave31
	./png2o.sh $*

interleave31: interleave31.c
	$(CC) $(CFLAGS) -s -o interleave31 interleave31.c

interleave31_1024: interleave31_1024.c
	$(CC) $(CFLAGS) -s -o interleave31_1024 interleave31_1024.c

# dependencies
arithmeticmean.o: arithmeticmean.c arithmeticmean.frag.c arithmeticmean.h shader.h util.h
audio.o: audio.c audio.h audio.frag.c shader.h
background.o: background.c background.h pfifo.h image.h histogram.h segment.h
bloom.o: bloom.c bloom.frag.c bloom.h gblur.h shader.h util.h
blur.o: blur.c blur.frag.c blur.h shader.h
borderwindow.o: borderwindow.c borderwindow.frag.c borderwindow_pick.frag.c borderwindow.h shader.h projection.vert.c projection_pick.vert.c
camera.o: camera.c camera.h matrix.h
copysquare.o: copysquare.c copysquare.frag.c copysquare.h shader.h util.h
difference.o: difference.c difference.frag.c difference.h shader.h arithmeticmean.h util.h
expcolour.o: expcolour.c expcolour.frag.c expcolour.h shader.h
falsecolour.o: falsecolour.c falsecolour.frag.c falsecolour.h shader.h util.h blur.h
fonts.o: fonts.c fonts.h
gblur.o: gblur.c gblur.frag.c gblur.h expcolour.h shader.h util.h
glyphs.o: glyphs.c glyphs.h
histogram.o: histogram.c histogram.h image.h
image.o: image.c image.h
intro.o: intro.c intro.h
library.o: library.c library.h background.h borderwindow.h image.h rdex.h shader.h tamura.h matrix.h
list.o: list.c list.h
main.o: main.c rdex.h
```

```
80  matrix.o: matrix.c matrix.h
    numericerror.o: numericerror.c numericerror.frag.c numericerror.h shader.h ↙
        ↳ arithmeticmean.h util.h
    pfifo.o: pfifo.c pfifo.h list.h
    rdex.o: rdex.c rdex.h arithmeticmean.h copysquare.h difference.h falsecolour.h ↙
        ↳ library.h numericerror.h reactiondiffusion.h screenshot.h sequence.h audio↙
        ↳ .h
    reactiondiffusion.o: reactiondiffusion.c reactiondiffusion.frag.c ↙
        ↳ reactiondiffusion.h shader.h util.h
85  screenshot.o: screenshot.c screenshot.h pfifo.h
    segment.o: segment.c segment.h image.h util.h
    sequence.o: sequence.c sequence.h library.h reactiondiffusion.h sequence.frag.c ↙
        ↳ sequence.vert.c glyphs.h
    shader.o: shader.c shader.h
    tamura.o: tamura.c tamura.h image.h
90  text.o: text.c text.h shader.h fonts.h text.frag.c text.vert.c
    timeline.o: timeline.c timeline.h
    util.o: util.c util.h
    worldsphere.o: worldsphere.c worldsphere.frag.c worldsphere.h shader.h

95  # EOF
```

68 src/matrix.c

```
    /* ================================================================
    rdex -- reaction-diffusion explorer
    Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
    ----------------------------------------------------------------
 5  Matrix computations
    ================================================================ */

    #include <math.h>
    #include <stdio.h>
10  #include "matrix.h"

    // vector copy

    void vcopy3(struct vec3 *m, struct vec3 *x) {
15    for (int i = 0; i < 3; ++i) {
        m->v[i] = x->v[i];
      }
    }

20  void vcopy4(struct vec4 *m, struct vec4 *x) {
      for (int i = 0; i < 4; ++i) {
        m->v[i] = x->v[i];
      }
    }
25
    void vcopy5(struct vec5 *m, struct vec5 *x) {
      for (int i = 0; i < 5; ++i) {
        m->v[i] = x->v[i];
      }
30  }

    // matrix copy
```

```
void mcopy3(struct mat3 *m, struct mat3 *x) {
  for (int i = 0; i < 9; ++i) {
    m->m[i] = x->m[i];
  }
}

void mcopy4(struct mat4 *m, struct mat4 *x) {
  for (int i = 0; i < 16; ++i) {
    m->m[i] = x->m[i];
  }
}

void mcopy5(struct mat5 *m, struct mat5 *x) {
  for (int i = 0; i < 25; ++i) {
    m->m[i] = x->m[i];
  }
}

// vector scalar multiplication

void vsmul3(struct vec3 *m, struct vec3 *x, float y) {
  for (int i = 0; i < 3; ++i) {
    m->v[i] = x->v[i] * y;
  }
}

void vsmul4(struct vec4 *m, struct vec4 *x, float y) {
  for (int i = 0; i < 4; ++i) {
    m->v[i] = x->v[i] * y;
  }
}

void vsmul5(struct vec5 *m, struct vec5 *x, float y) {
  for (int i = 0; i < 5; ++i) {
    m->v[i] = x->v[i] * y;
  }
}

// vector scalar division

void vsdiv3(struct vec3 *m, struct vec3 *x, float y) {
  for (int i = 0; i < 3; ++i) {
    m->v[i] = x->v[i] / y;
  }
}

void vsdiv4(struct vec4 *m, struct vec4 *x, float y) {
  for (int i = 0; i < 4; ++i) {
    m->v[i] = x->v[i] / y;
  }
}

void vsdiv5(struct vec5 *m, struct vec5 *x, float y) {
  for (int i = 0; i < 5; ++i) {
    m->v[i] = x->v[i] / y;
  }
}
```

```
// vector vector addition

void vvadd3(struct vec3 *m, struct vec3 *x, struct vec3 *y) {
  for (int i = 0; i < 3; ++i) {
    m->v[i] = x->v[i] + y->v[i];
  }
}

void vvadd4(struct vec4 *m, struct vec4 *x, struct vec4 *y) {
  for (int i = 0; i < 4; ++i) {
    m->v[i] = x->v[i] + y->v[i];
  }
}

void vvadd5(struct vec5 *m, struct vec5 *x, struct vec5 *y) {
  for (int i = 0; i < 5; ++i) {
    m->v[i] = x->v[i] + y->v[i];
  }
}

// vector vector subtraction

void vvsub3(struct vec3 *m, struct vec3 *x, struct vec3 *y) {
  for (int i = 0; i < 3; ++i) {
    m->v[i] = x->v[i] - y->v[i];
  }
}

void vvsub4(struct vec4 *m, struct vec4 *x, struct vec4 *y) {
  for (int i = 0; i < 4; ++i) {
    m->v[i] = x->v[i] - y->v[i];
  }
}

void vvsub5(struct vec5 *m, struct vec5 *x, struct vec5 *y) {
  for (int i = 0; i < 5; ++i) {
    m->v[i] = x->v[i] - y->v[i];
  }
}

// vector vector multiplication (componentwise)

void vvmul4(struct vec4 *m, struct vec4 *x, struct vec4 *y) {
  for (int i = 0; i < 4; ++i) {
    m->v[i] = x->v[i] * y->v[i];
  }
}

// vector vector division (componentwise)

void vvdiv4(struct vec4 *m, struct vec4 *x, struct vec4 *y) {
  for (int i = 0; i < 4; ++i) {
    m->v[i] = x->v[i] / y->v[i];
  }
}
```

```
// matrix vector multiplication

void mvmul3(struct vec3 *v, struct mat3 *x, struct vec3 *y) {
  int k = 0;
  for (int i = 0; i < 3; ++i) {
    v->v[i] = 0;
    for (int j = 0; j < 3; ++j) {
      v->v[i] += x->m[k++] * y->v[j];
    }
  }
}

void mvmul4(struct vec4 *v, struct mat4 *x, struct vec4 *y) {
  int k = 0;
  for (int i = 0; i < 4; ++i) {
    v->v[i] = 0;
    for (int j = 0; j < 4; ++j) {
      v->v[i] += x->m[k++] * y->v[j];
    }
  }
}

void mvmul5(struct vec5 *v, struct mat5 *x, struct vec5 *y) {
  int k = 0;
  for (int i = 0; i < 5; ++i) {
    v->v[i] = 0;
    for (int j = 0; j < 5; ++j) {
      v->v[i] += x->m[k++] * y->v[j];
    }
  }
}

// matrix matrix multiplication

void mmmul3(struct mat3 *m, struct mat3 *x, struct mat3 *y) {
  const int N = 3;
  int kk = 0;
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      m->m[kk] = 0;
      for (int k = 0; k < N; ++k) {
        m->m[kk] += x->m[k * N + j] * y->m[i * N + k];
      }
      kk++;
    }
  }
}

void mmmul4(struct mat4 *m, struct mat4 *x, struct mat4 *y) {
  const int N = 4;
  int kk = 0;
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      m->m[kk] = 0;
      for (int k = 0; k < N; ++k) {
        m->m[kk] += x->m[k * N + j] * y->m[i * N + k];
```

```
205         }
            kk++;
          }
        }
      }
210
      void mmmul5(struct mat5 *m, struct mat5 *x, struct mat5 *y) {
        const int N = 5;
        int kk = 0;
        for (int i = 0; i < N; ++i) {
215       for (int j = 0; j < N; ++j) {
            m->m[kk] = 0;
            for (int k = 0; k < N; ++k) {
              m->m[kk] += x->m[k * N + j] * y->m[i * N + k];
            }
220         kk++;
          }
        }
      }

225   // homogeneous matrix construction

      void identity3(struct mat3 *m) {
        const int N = 3;
        int k = 0;
230     for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
          m->m[k++] = (i == j);
        } }
      }
235
      void identity4(struct mat4 *m) {
        const int N = 4;
        int k = 0;
        for (int i = 0; i < N; ++i) {
240     for (int j = 0; j < N; ++j) {
          m->m[k++] = (i == j);
        } }
      }

245   void identity5(struct mat5 *m) {
        const int N = 5;
        int k = 0;
        for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
250       m->m[k++] = (i == j);
        } }
      }

      void homogen4(struct mat4 *m, struct mat3 *x) {
255     m->m[ 0] = x->m[0]; m->m[ 1] = x->m[1]; m->m[ 2] = x->m[2]; m->m[ 3] = 0;
        m->m[ 4] = x->m[3]; m->m[ 5] = x->m[4]; m->m[ 6] = x->m[5]; m->m[ 7] = 0;
        m->m[ 8] = x->m[6]; m->m[ 9] = x->m[7]; m->m[10] = x->m[8]; m->m[11] = 0;
        m->m[12] = 0;       m->m[13] = 0;       m->m[14] = 0;       m->m[15] = 1;
      }
260
      void homogen5(struct mat5 *m, struct mat4 *x) {
```

```
  m->m[ 0] = x->m[ 0]; m->m[ 1] = x->m[ 1]; m->m[ 2] = x->m[ 2]; m->m[ 3] = x->m[ 3]; m->m[ 4] = 0;
  m->m[ 5] = x->m[ 4]; m->m[ 6] = x->m[ 5]; m->m[ 7] = x->m[ 6]; m->m[ 8] = x->m[ 7]; m->m[ 9] = 0;
  m->m[10] = x->m[ 8]; m->m[11] = x->m[ 9]; m->m[12] = x->m[10]; m->m[13] = x->m[11]; m->m[14] = 0;
  m->m[15] = x->m[12]; m->m[16] = x->m[13]; m->m[17] = x->m[14]; m->m[18] = x->m[15]; m->m[19] = 0;
  m->m[20] = 0;        m->m[21] = 0;        m->m[22] = 0;        m->m[23] = 0;        m->m[24] = 1;
}

void translate4(struct mat4 *m, struct vec3 *v) {
  const int N = 4;
  identity4(m);
  int k = N - 1;
  for (int i = 0; i < N - 1; ++i) {
    m->m[k] = v->v[i];
    k += N;
  }
}

void translate5(struct mat5 *m, struct vec4 *v) {
  const int N = 5;
  identity5(m);
  int k = N - 1;
  for (int i = 0; i < N - 1; ++i) {
    m->m[k] = v->v[i];
    k += N;
  }
}

// rolling ball

void rollingball4(struct mat4 *m, float R, struct vec3 *v) {
  float r = sqrt(v->v[0] * v->v[0] + v->v[1] * v->v[1] + v->v[2] * v->v[2]);
  float D = sqrt(R*R+r*r);
  float c = R / D;
  float s = r / D;
  float x, y, z;
  if (r > 0.00001) {
    x = v->v[0] / r;
    y = v->v[1] / r;
    z = v->v[2] / r;
  } else {
    x = 0;
    y = 0;
    z = 0;
  }
  float t[16] = {
    1-x*x*(1-c), -(1-c)*x*y,  -(1-c)*x*z,  s*x,
    -(1-c)*x*y,  1-y*y*(1-c), -(1-c)*y*z,  s*y,
    -(1-c)*x*z,  -(1-c)*y*z,  1-z*z*(1-c), s*z,
    -s*x,        -s*y,        -s*z,        c
  };
  for (int i = 0; i < 16; ++i) {
    m->m[i] = t[i];
```

```
  }
}

// helper functions

float vlength4(struct vec4 *x) {
  float v = 0;
  for (int i = 0; i < 4; ++i) {
    v += x->v[i] * x->v[i];
  }
  return sqrt(v);
}

float vdistance4(struct vec4 *x, struct vec4 *y) {
  struct vec4 d;
  vvsub4(&d, x, y);
  return vlength4(&d);
}

float vvdot4(struct vec4 *x, struct vec4 *y) {
  float v = 0;
  for (int i = 0; i < 4; ++i) {
    v += x->v[i] * y->v[i];
  }
  return v;
}

// debugging

void vprint4(struct vec4 *m) {
  fprintf(stderr, "%f %f %f %f\n", m->v[0], m->v[1], m->v[2], m->v[3]);
}

void mprint4(struct mat4 *m) {
  fprintf(
    stderr, "%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n",
    m->m[ 0], m->m[ 1], m->m[ 2], m->m[ 3],
    m->m[ 4], m->m[ 5], m->m[ 6], m->m[ 7],
    m->m[ 8], m->m[ 9], m->m[10], m->m[11],
    m->m[12], m->m[13], m->m[14], m->m[15]
  );
}

// EOF
```

69 src/matrix.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Matrix computations
================================================================ */

#ifndef MATRIX_H
#define MATRIX_H 1

```

```
// data structures
struct vec3 { float v[3]; };
struct vec4 { float v[4]; };
struct vec5 { float v[5]; };
struct mat3 { float m[3*3]; };
struct mat4 { float m[4*4]; };
struct mat5 { float m[5*5]; };

// copying
void vcopy3(struct vec3 *m, struct vec3 *x);
void vcopy4(struct vec4 *m, struct vec4 *x);
void vcopy5(struct vec5 *m, struct vec5 *x);
void mcopy3(struct mat3 *m, struct mat3 *x);
void mcopy4(struct mat4 *m, struct mat4 *x);
void mcopy5(struct mat5 *m, struct mat5 *x);

// basic operations
void vsmul3(struct vec3 *m, struct vec3 *x, float y);
void vsmul4(struct vec4 *m, struct vec4 *x, float y);
void vsmul5(struct vec5 *m, struct vec5 *x, float y);
void vsdiv3(struct vec3 *m, struct vec3 *x, float y);
void vsdiv4(struct vec4 *m, struct vec4 *x, float y);
void vsdiv5(struct vec5 *m, struct vec5 *x, float y);

void vvadd3(struct vec3 *m, struct vec3 *x, struct vec3 *y);
void vvadd4(struct vec4 *m, struct vec4 *x, struct vec4 *y);
void vvadd5(struct vec5 *m, struct vec5 *x, struct vec5 *y);
void vvsub3(struct vec3 *m, struct vec3 *x, struct vec3 *y);
void vvsub4(struct vec4 *m, struct vec4 *x, struct vec4 *y);
void vvsub5(struct vec5 *m, struct vec5 *x, struct vec5 *y);

void vvmul4(struct vec4 *m, struct vec4 *x, struct vec4 *y);
void vvdiv4(struct vec4 *m, struct vec4 *x, struct vec4 *y);

void mvmul3(struct vec3 *v, struct mat3 *x, struct vec3 *y);
void mvmul4(struct vec4 *v, struct mat4 *x, struct vec4 *y);
void mvmul5(struct vec5 *v, struct mat5 *x, struct vec5 *y);

void mmmul3(struct mat3 *m, struct mat3 *x, struct mat3 *y);
void mmmul4(struct mat4 *m, struct mat4 *x, struct mat4 *y);
void mmmul5(struct mat5 *m, struct mat5 *x, struct mat5 *y);

// homogeneous matrix construction
void identity3(struct mat3 *m);
void identity4(struct mat4 *m);
void identity5(struct mat5 *m);
void homogen4(struct mat4 *m, struct mat3 *x);
void homogen5(struct mat5 *m, struct mat4 *x);
void translate4(struct mat4 *m, struct vec3 *v);
void translate5(struct mat5 *m, struct vec4 *v);
void rollingball4(struct mat4 *m, float R, struct vec3 *v);

// helper functions
float vlength4(struct vec4 *x);
float vdistance4(struct vec4 *x, struct vec4 *y);
float vvdot4(struct vec4 *x, struct vec4 *y);
```

```
// debugging
void vprint4(struct vec4 *m);
void mprint4(struct mat4 *m);

#endif
```

70 src/numericerror.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Numeric Error Shader
================================================================ */

#include "util.h"
#include "numericerror.h"
#include "numericerror.frag.c"

//==============================================================
// numeric error shader initialization
struct numericerror *numericerror_init(
  struct numericerror *numericerror
) {
  if (! numericerror) { return 0; }
  if (! shader_init(&numericerror->shader, 0, numericerror_frag)) {
    return 0;
  }
  shader_uniform(numericerror, texture);
  shader_uniform(numericerror, dx);
  shader_uniform(numericerror, dy);
  numericerror->value.texture = 0;
  numericerror->value.dx      = 0;
  numericerror->value.dy      = 0;
  if (! arithmeticmean_init(&numericerror->arithmeticmean)) {
    return 0;
  }
  glGenTextures(1, &numericerror->texture);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, numericerror->texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glDisable(GL_TEXTURE_2D);
  numericerror->width = 0;
  numericerror->height = 0;
  numericerror->calc = 0;
  numericerror->mean = 0;
  return numericerror;
}

//==============================================================
// numeric error shader reshape callback
void numericerror_reshape(
  struct numericerror *numericerror, int w, int h
) {
  numericerror->width  = roundtwo(w);
  numericerror->height = roundtwo(h);
```

```
50    numericerror->value.dx = 1.0 / numericerror->width;
      numericerror->value.dy = 1.0 / numericerror->height;
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, numericerror->texture);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB,
55      numericerror->width, numericerror->height, 0, GL_RGBA, GL_FLOAT, 0
      );
      glDisable(GL_TEXTURE_2D);
      arithmeticmean_reshape(&numericerror->arithmeticmean, w, h);
    }
60
    //=============================================================
    // numeric error shader display callback
    void numericerror_display(
      struct numericerror *numericerror, GLuint fbo, GLuint texture
65  ) {
      if (numericerror->calc) {
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture);
        //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
70      glFramebufferTexture2DEXT(
          GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
          numericerror->texture, 0
        );
        glUseProgramObjectARB(numericerror->shader.program);
75      shader_updatei(numericerror, texture);
        shader_updatef(numericerror, dx);
        shader_updatef(numericerror, dy);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
80      gluOrtho2D(0, 1, 0, 1);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glViewport(0, 0, numericerror->width, numericerror->height);
        glBegin(GL_QUADS); { glColor4f(1,1,1,1);
85        glTexCoord2f(0, 0); glVertex2f(0, 0);
          glTexCoord2f(1, 0); glVertex2f(1, 0);
          glTexCoord2f(1, 1); glVertex2f(1, 1);
          glTexCoord2f(0, 1); glVertex2f(0, 1);
        } glEnd();
90      glUseProgramObjectARB(0);
        //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
        glDisable(GL_TEXTURE_2D);
        arithmeticmean_display(
          &numericerror->arithmeticmean, fbo, numericerror->texture
95      );
        numericerror->mean = numericerror->arithmeticmean.result[0];
        numericerror->calc = 0;
      }
    }
100
    //=============================================================
    // numeric error shader idle callback
    void numericerror_idle(struct numericerror *numericerror) {
      arithmeticmean_idle(&numericerror->arithmeticmean);
105 }
```

```
// EOF
```

71 src/numericerror.frag

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
================================================================ */

uniform sampler2D texture; // U := r, V := g, other channels ignored
uniform float dx;          // horizontal distance between texels
uniform float dy;          // vertical distance between texels

void main(void) {
  vec2 p = gl_TexCoord[0].st;                          // coordinates
  vec2 gx = texture2D(texture, p + vec2(-dx,0.0)).rg  // x-wards edge
          + texture2D(texture, p + vec2( dx,0.0)).rg
          - texture2D(texture, p + vec2(0.0,0.0)).rg * 2.0;
  vec2 gy = texture2D(texture, p + vec2(0.0,-dy)).rg  // y-wards edge
          + texture2D(texture, p + vec2(0.0, dy)).rg
          - texture2D(texture, p + vec2(0.0,0.0)).rg * 2.0;
  vec2 xy = texture2D(texture, p + vec2(-dx,-dy)).rg  // xy-wards edge
          + texture2D(texture, p + vec2( dx, dy)).rg
          - texture2D(texture, p + vec2(0.0,0.0)).rg * 2.0;
  vec2 yx = texture2D(texture, p + vec2( dx, dy)).rg  // yx-wards edge
          + texture2D(texture, p + vec2(-dx,-dy)).rg
          - texture2D(texture, p + vec2(0.0,0.0)).rg * 2.0;
  float e = sqrt(dot(gx,gx)+dot(gy,gy)+dot(xy,xy)+dot(yx,yx));
  gl_FragColor = vec4(e, e, e, 1.0);                    // total edges
}
```

72 src/numericerror.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Numeric Error Shader
================================================================ */

#ifndef NUMERICERROR_H
#define NUMERICERROR_H 1

#include "arithmeticmean.h"

//================================================================
// numeric error shader data
struct numericerror { struct shader shader;
  struct { GLint texture; GLint dx; GLint dy; } uniform;
  struct { int   texture; float dx; float dy; } value;
  struct arithmeticmean arithmeticmean;
  GLuint texture;
  int width;
  int height;
  int calc;
  float mean;
```

```
};

//=============================================================
// prototypes
struct numericerror *numericerror_init(
  struct numericerror *numericerror
);
void numericerror_reshape(
  struct numericerror *numericerror, int w, int h
);
void numericerror_display(
  struct numericerror *numericerror, GLuint fbo, GLuint texture
);
void numericerror_idle(struct numericerror *numericerror);

#endif
```

73 src/pfifo.c

```
/* =============================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2009,2010  Claude Heiland-Allen <claude@mathr.co.uk>
-------------------------------------------------------------------
pthread-based fifo
============================================================= */

#include <assert.h>
#include <string.h>
#include "pfifo.h"

//#include <stdio.h>

//=============================================================
// prototypes
void *pfifo_consumerthread(void *);

//=============================================================
// create a new fifo with a consumer thread
struct pfifo *pfifo_create(pfifo_consumer *consumer, void *consumerdata) {
  struct pfifo *p = malloc(sizeof(struct pfifo));
  if (!p) return 0;
  list_init(&(p->list));
  p->consumer = consumer;
  p->consumerdata = consumerdata;
  p->mutex = malloc(sizeof(pthread_mutex_t));
  pthread_mutex_init(p->mutex, 0);
  p->nonempty = malloc(sizeof(pthread_cond_t));
  pthread_cond_init(p->nonempty, 0);
  pthread_create(&(p->thread), 0, pfifo_consumerthread, p);
  return p;
}

//=============================================================
// append to the fifo, copying data
void pfifo_enqueue(struct pfifo *p, size_t length, const void *data) {
  struct pfifo_node *n = malloc(sizeof(struct pfifo_node));
  assert(n);
```

```
  n->length = length;
  n->data = malloc(length);
  assert(n->data);
  memcpy(n->data, data, length);
  pthread_mutex_lock(p->mutex);
  list_inserttail(&(p->list), &(n->node));
  //int i = list_length(&(p->list));
  pthread_mutex_unlock(p->mutex);
  pthread_cond_signal(p->nonempty);
  //fprintf(stderr, "\ndebug: pfifo size %d\n", i);
}

//============================================================
// loop running consumer callback on each fifo item, freeing data
void *pfifo_consumerthread(void *fifo) {
  struct pfifo *p = fifo;
  while (1) {
    pthread_mutex_lock(p->mutex);
    while (list_isempty(&(p->list))) pthread_cond_wait(p->nonempty, p->mutex);
    struct pfifo_node *n = (struct pfifo_node *) list_removehead(&(p->list));
    pthread_mutex_unlock(p->mutex);
    p->consumer(p->consumerdata, n->length, n->data);
    free(n->data);
    free(n);
  }
  return 0;
}


// EOF
```

74 src/pfifo.h

```
/* ============================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2009  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------
pthread-based fifo
============================================================ */

#ifndef PFIFO_H
#define PFIFO_H 1

#include <stdlib.h>
#include <pthread.h>
#include "list.h"

typedef void (pfifo_consumer)(void *, size_t, void *);

struct pfifo_node {
  struct node node;
  size_t length;
  void *data;
};

struct pfifo {
  pfifo_consumer *consumer;
```

```
25    void *consumerdata;
      struct list list;
      pthread_t thread;
      pthread_mutex_t *mutex;
      pthread_cond_t *nonempty;
30  };

    struct pfifo *pfifo_create(pfifo_consumer *consumer, void *consumerdata);
    void pfifo_destroy(struct pfifo *fifo);
    void pfifo_enqueue(struct pfifo *fifo, size_t length, const void *data);
35
    #endif
```

75 src/png2o.sh

```
    #!/bin/bash
    PNGFILE="${1}.png"
    PPMFILE="${1}.ppm"
    PGMFILE="${1}.pgm"
 5  RGBFILE="${1}.rgb"
    AFILE="${1}.a"
    RGBAFILE="${1}.rgba"
    OFILE="${1}.o"
    PNGSIZE="$(identify -format "%w" "${PNGFILE}")"
10  pngtopnm        "${PNGFILE}" > "${PPMFILE}"
    pngtopnm -alpha "${PNGFILE}" > "${PGMFILE}"
    tail -c "$(( 3*${PNGSIZE}*${PNGSIZE} ))" "${PPMFILE}" > "${RGBFILE}"
    tail -c "$(( 1*${PNGSIZE}*${PNGSIZE} ))" "${PGMFILE}" > "${AFILE}"
    ./interleave31 "${RGBFILE}" "${AFILE}" "${PNGSIZE}" > "${RGBAFILE}"
15  #objcopy --input binary --output elf32-i386 --binary-architecture i386 "${↙
        ↘ RGBAFILE}" "${OFILE}"
    objcopy --input binary --output elf64-x86-64 --binary-architecture i386 "${↙
        ↘ RGBAFILE}" "${OFILE}"
    rm -f "${PPMFILE}" "${PGMFILE}" "${RGBFILE}" "${AFILE}" "${RGBAFILE}"
```

76 src/projection_pick.vert

```
    /* ================================================================
    rdex -- reaction-diffusion explorer
    Copyright (C) 2010,2011,2019  Claude Heiland-Allen <claude@mathr.co.uk>
    ----------------------------------------------------------------
 5  4D->3D projection shader for ball picking
    ================================================================ */

    // 4D perspective projection
    uniform vec4 origin;
10  uniform vec4 scale;
    uniform mat4 rotate;
    uniform vec4 translate;
    uniform vec4 near;
    uniform vec4 far;
15
    // billboarding
    attribute vec2 delta;

    varying float ignore;
```

```
20
    void main(void) {
      vec4 p0 = rotate * (0.5 * scale * (gl_Vertex - origin));
      vec4 p = p0 + translate;
      mat4 m = mat4(
25      2.0*near.w/(far.x - near.x), 0.0, 0.0, (far.x + near.x)/(far.x - near.x),
        0.0, 2.0*near.w/(far.y - near.y), 0.0, (far.y + near.y)/(far.y - near.y),
        0.0, 0.0, 2.0*near.w/(far.z - near.z), (far.z + near.z)/(far.z - near.z),
        0.0, 0.0, 0.0,                         (far.w + near.w)/(far.w - near.w)
      );
30    float k = 2.0 * near.w * far.w / (far.w - near.w);
      vec4 q = m * p;
      q.w -= k;
      q /= p.w;
      vec4 v = vec4(10000.0, 10000.0, 10000.0, 1.0); // way off-screen
35    float r2 = dot(p0,p0) * 0.5;
      float s = 0.0;
      if (r2 < 1.0) {
        s = sqrt(1.0 - r2);
        v = gl_ModelViewProjectionMatrix * q;
40      v.xy += delta * s;
        ignore = 0.0;
      } else {
        ignore = 1.0;
      }
45    gl_TexCoord[0] = gl_MultiTexCoord0;
      gl_Position = v;
    }
```

77 src/projection.vert

```
    /* ================================================================
    rdex -- reaction-diffusion explorer
    Copyright (C) 2010,2011,2019  Claude Heiland-Allen <claude@mathr.co.uk>
    ----------------------------------------------------------------------
 5  4D->3D projection shader
    ================================================================ */

    // 4D perspective projection
    uniform vec4 origin;
10  uniform vec4 scale;
    uniform mat4 rotate;
    uniform vec4 translate;
    uniform vec4 near;
    uniform vec4 far;
15
    // billboarding
    attribute vec2 delta;

    // spinning orbs
20  attribute vec2 spin;

    // ball picking
    attribute float pick;

25  varying vec2 spin2;
    varying float scale2;
```

```
varying float pick2;

varying float ignore;

void main(void) {
  vec4 p0 = rotate * (0.5 * scale * (gl_Vertex - origin));
  vec4 p = p0 + translate;
  mat4 m = mat4(
    2.0*near.w/(far.x - near.x), 0.0, 0.0, (far.x + near.x)/(far.x - near.x),
    0.0, 2.0*near.w/(far.y - near.y), 0.0, (far.y + near.y)/(far.y - near.y),
    0.0, 0.0, 2.0*near.w/(far.z - near.z), (far.z + near.z)/(far.z - near.z),
    0.0, 0.0, 0.0,                         (far.w + near.w)/(far.w - near.w)
  );
  float k = 2.0 * near.w * far.w / (far.w - near.w);
  vec4 q = m * p;
  q.w -= k;
  q /= p.w;
  vec4 v = vec4(10000.0, 10000.0, 10000.0, 1.0); // way off-screen
  float r2 = dot(p0,p0) * 0.5;
  float s = 0.0;
  if (r2 < 1.0) {
    s = sqrt(1.0 - r2);
    v = gl_ModelViewProjectionMatrix * q;
    v.xy += delta * s;
    ignore = 0.0;
  } else {
    ignore = 1.0;
  }
  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_Position = v;
  spin2 = spin;
  scale2 = 2.0 * length(delta) * s;
  pick2 = pick;
}
```

78 src/rdex.c

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010,2017,2019  Claude Heiland-Allen <claude@mathr.co.uk>
---------------------------------------------------------------------------
Main Module
========================================================================== */

#define _DEFAULT_SOURCE

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "rdex.h"

//==========================================================================
// main module global mutable state
static struct rdex rdex;
```

```
20  //============================================================
    // main module initialization
    int rdex_init() {
      int nrt = 0 != getenv("RDEX_RENDER");
      memset(&rdex, 0, sizeof(rdex));
25    glGenFramebuffersEXT(1, &rdex.fbo);
      glClampColorARB( GL_CLAMP_VERTEX_COLOR_ARB , GL_FALSE );
      glClampColorARB( GL_CLAMP_FRAGMENT_COLOR_ARB, GL_FALSE );
      glClampColorARB( GL_CLAMP_READ_COLOR_ARB , GL_FALSE );
      if (! reactiondiffusion_init(&rdex.reactiondiffusion)) { return 0; }
30    if (! copysquare_init       (&rdex.copysquare       )) { return 0; }
      if (! arithmeticmean_init   (&rdex.arithmeticmean   )) { return 0; }
      if (! numericerror_init     (&rdex.numericerror     )) { return 0; }
      if (! difference_init       (&rdex.difference       )) { return 0; }
      if (! falsecolour_init      (&rdex.falsecolour      )) { return 0; }
35    if (! library_init          (&rdex.library, rdex.fbo)) { return 0; }
      if (! screenshot_init       (&rdex.screenshot, nrt  )) { return 0; }
      if (! audio_init            (&rdex.audio, nrt       )) { return 0; }
    //  if (! sequence_init         (&rdex.sequence         )) { return 0; }
      if (! intro_init            (&rdex.intro            )) { return 0; }
40    if (! timeline_init         (&rdex.timeline         )) { return 0; }
      if (! text_init             (&rdex.text             )) { return 0; }
      rdex.starttime = 0;
      rdex.fullscreen = 0;
    //  reactiondiffusion_spawn(&rdex.reactiondiffusion);
45    rdex.done = 0;
      rdex.behaviour = "none";
    #if 1
      rdex.mode = mode_random;
    #else
50    rdex.mode = mode_textual;
    #endif
      rdex.nrt = nrt;
      return 1;
    }
55
    //============================================================
    // main module reshape callback
    void rdex_reshape(int w, int h) {
      reactiondiffusion_reshape(&rdex.reactiondiffusion, rdex_tex_size, ↙
          ↳ rdex_tex_size);
60    copysquare_reshape       (&rdex.copysquare,        rdex_tex_size, ↙
          ↳ rdex_tex_size);
      arithmeticmean_reshape   (&rdex.arithmeticmean,    rdex_tex_size, ↙
          ↳ rdex_tex_size);
      numericerror_reshape     (&rdex.numericerror,      rdex_tex_size, ↙
          ↳ rdex_tex_size);
      difference_reshape       (&rdex.difference,        rdex_tex_size, ↙
          ↳ rdex_tex_size);
      falsecolour_reshape      (&rdex.falsecolour,       rdex_tex_size, ↙
          ↳ rdex_tex_size);
65    library_reshape          (&rdex.library,           w, h);
      screenshot_reshape       (&rdex.screenshot,        w, h);
    //  sequence_reshape         (&rdex.sequence,          w, h);
      intro_reshape            (&rdex.intro,             w, h);
      text_reshape             (&rdex.text,              w, h);
70  }
```

```
//============================================================================
// main module display callback
void rdex_display() {
  if (rdex.timeline.state1 != rdex.timeline.state) {
    switch (rdex.timeline.state) {
      case timeline_intro1:
        intro_start(&rdex.intro);
        audio_start(&rdex.audio);
        break;
      case timeline_intro2:
        rdex.library.phase = 0;
        rdex.library.state = library_state_intro;
        break;
      case timeline_main:
        rdex.library.phase = 0;
        rdex.library.state = library_state_active;
        break;
      case timeline_outro:
        rdex.library.phase = 0;
        rdex.library.state = library_state_outro;
        audio_stop(&rdex.audio);
        break;
      case timeline_done:
        rdex.library.phase = 0;
        rdex.library.state = library_state_done;
        break;
      case timeline_idle:
        break;
    }
  }
  timeline_advance(&rdex.timeline);
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, rdex.fbo);
  screenshot_display1       (&rdex.screenshot);
  for (int g = 0; g < rdex_overdrive; ++g) {
    reactiondiffusion_display(&rdex.reactiondiffusion, rdex.fbo);
    if (g % 2 == 0) {
      audio_display1        (&rdex.audio,              rdex.fbo, rdex.↙
          ↪ reactiondiffusion.texture, g / 2);
    }
  }
  copysquare_display        (&rdex.copysquare,         rdex.fbo, rdex.↙
      ↪ reactiondiffusion.texture);
  arithmeticmean_display    (&rdex.arithmeticmean,     rdex.fbo, rdex.copysquare ↙
      ↪       .texture);
  numericerror_display      (&rdex.numericerror,       rdex.fbo, rdex.↙
      ↪ reactiondiffusion.texture);
  difference_display        (&rdex.difference,         rdex.fbo, rdex.↙
      ↪ reactiondiffusion.texture);
  falsecolour_display       (&rdex.falsecolour,        rdex.fbo, rdex.↙
      ↪ reactiondiffusion.texture);
  library_display_save(
    &rdex.library,
    rdex.fbo,
    rdex.falsecolour.texture,
    rdex.reactiondiffusion.texture,
    rdex.reactiondiffusion.value.ru,
```

```
    rdex.reactiondiffusion.value.rv,
    rdex.reactiondiffusion.value.f,
    rdex.reactiondiffusion.value.k,
    (rdex.reactiondiffusion.frame / (double) (1 << 14)) *
    (rdex.reactiondiffusion.frame / (double) (1 << 14)),
    rdex.behaviour
  );
  library_display          (&rdex.library,          rdex.fbo, rdex.falsecolour.texture);
  if (rdex.library.picked != -1) {
    rdex.reactiondiffusion.value.ru = rdex.library.array[rdex.library.picked].v[0];
    rdex.reactiondiffusion.value.rv = rdex.library.array[rdex.library.picked].v[1];
    rdex.reactiondiffusion.value.f  = rdex.library.array[rdex.library.picked].v[2];
    rdex.reactiondiffusion.value.k  = rdex.library.array[rdex.library.picked].v[3];
    rdex.library.target.v[0] = rdex.library.array[rdex.library.picked].v[0];
    rdex.library.target.v[1] = rdex.library.array[rdex.library.picked].v[1];
    rdex.library.target.v[2] = rdex.library.array[rdex.library.picked].v[2];
    rdex.library.target.v[3] = rdex.library.array[rdex.library.picked].v[3];
    rdex.library.picked = -1;
  }

//  sequence_display         (&rdex.sequence, &rdex.library, &rdex.reactiondiffusion, &rdex.audio);
  text_display             (&rdex.text);
  intro_display            (&rdex.intro, 0.04); /* FIXME hardcoded frame rate */
  glutSwapBuffers();
  audio_display2           (&rdex.audio);
  screenshot_display2      (&rdex.screenshot);
  rdex.frame += 1;
  int reseed = rdex.done;
  if (rdex.done) {
    float ru = 0, rv = 0, f = 0, k = 0, ru2, rv2, f2, k2;
    switch (rdex.mode) {
    case mode_random: {
      int i = 0;
      float randomness = 1e6; //16.0;
      while (! library_pick(&rdex.library, &ru2, &rv2, &f2, &k2, randomness)) {
        ru += ru2;
        rv += rv2;
        f  += f2;
        k  += k2;
        i  += 1;
//      randomness = 2.0;
      }
      if (i == 0) {
        reactiondiffusion_randomize(&rdex.reactiondiffusion);
      } else {
        ru /= i;
        rv /= i;
        f  /= i;
        k  /= i;
        reactiondiffusion_near(&rdex.reactiondiffusion, ru, rv, f, k, 0.01);
      }
```

```
      rdex.audio.sphase = 0;
    } break;
    case mode_textual: {
      if (reseed) { rdex.reactiondiffusion.frame = 0; }
    } break;
    case mode_sequence: {
      if (reseed) { rdex.reactiondiffusion.frame = 0; }
    /*
      if (reseed) {
      float randomness = 1.1;
      if (! library_pick(&rdex.library, &ru, &rv, &f, &k, randomness)) {
        rdex.loop[rdex.loopindex].v[0] = ru;
        rdex.loop[rdex.loopindex].v[1] = rv;
        rdex.loop[rdex.loopindex].v[2] = f;
        rdex.loop[rdex.loopindex].v[3] = k;
      } else { //if (randomness * rand() / (double) RAND_MAX < 1.0) {
        rdex.loop[rdex.loopindex].v[0] = 0.3 * rand() / (double) RAND_MAX;
        rdex.loop[rdex.loopindex].v[1] = 0.3 * rand() / (double) RAND_MAX;
        rdex.loop[rdex.loopindex].v[2] = 0.1 * rand() / (double) RAND_MAX;
        rdex.loop[rdex.loopindex].v[3] = 0.1 * rand() / (double) RAND_MAX;
      }
    }
      ru = rdex.loop[rdex.loopindex].v[0];
      rv = rdex.loop[rdex.loopindex].v[1];
      f  = rdex.loop[rdex.loopindex].v[2];
      k  = rdex.loop[rdex.loopindex].v[3];
      if (reseed) {
        reactiondiffusion_near(&rdex.reactiondiffusion, ru, rv, f, k, 0.001);
      } else {
        rdex.reactiondiffusion.value.ru = ru;
        rdex.reactiondiffusion.value.rv = rv;
        rdex.reactiondiffusion.value.f  = f;
        rdex.reactiondiffusion.value.k  = k;
      }
      fprintf(stderr, "L\t");
    */
    } break;
    }
    rdex.library.worldsphere.delta[0] = (rand() / (double) RAND_MAX - 0.5) / ↙
        ↪ 64.0;
    rdex.library.worldsphere.delta[1] = (rand() / (double) RAND_MAX - 0.5) / ↙
        ↪ 64.0;
    rdex.done = 0;
  }
  glutReportErrors();
  if (rdex.library.state == library_state_active)
    rdex.audio.state = audio_state_rolling;
  if (rdex.timeline.state == timeline_idle)
    timeline_start(&rdex.timeline);
}

//=============================================================================
// main module exit callback
void rdex_atexit(void) {
// /* not safe to call at exit it seems... */
//   if (glutGameModeGet(GLUT_GAME_MODE_ACTIVE)) {
//     glutLeaveGameMode();
```

```
//  }
  audio_atexit(&rdex.audio);
  double timeelapsed = (double) time(NULL) - (double) rdex.starttime;
  fprintf(stderr, "\n\n↙
      ↪ ------------------------------------------------------------------------\↙
      ↪ n");
  fprintf(stderr, "------------------------------ statistics ↙
      ↪ -------------------------------\n");
  fprintf(stderr, ↙
      ↪ "------------------------------------------------------------------------\↙
      ↪ n");
  fprintf(stderr, "%10d seconds elapsed (+/- 1)\n", (int) timeelapsed);
  fprintf(stderr, "%10d frames rendered (%f fps)\n", rdex.frame, rdex.frame / ↙
      ↪ timeelapsed);
  fprintf(stderr, "%10d frames dropped\n", (int) (25 * timeelapsed - rdex.frame)↙
      ↪ );
  fprintf(stderr, "%10d species analyzed (%f sps)\n", rdex.species, rdex.species↙
      ↪  / timeelapsed);
  fprintf(stderr, "        %6.2f%% uniform (%d)\n", 100.0 * rdex.uniform / (↙
      ↪ double) rdex.species, rdex.uniform);
  fprintf(stderr, "        %6.2f%% stable  (%d)\n", 100.0 * rdex.stable  / (↙
      ↪ double) rdex.species, rdex.stable);
  fprintf(stderr, "        %6.2f%% dynamic (%d)\n", 100.0 * rdex.dynamic / (↙
      ↪ double) rdex.species, rdex.dynamic);
  fprintf(stderr, "        %6.2f%% erratic (%d)\n", 100.0 * rdex.erratic / (↙
      ↪ double) rdex.species, rdex.erratic);
  fprintf(stderr, ↙
      ↪ "------------------------------------------------------------------------\↙
      ↪ n");
  fprintf(stderr, "average frame time: %f seconds\n", rdex.sumdt / rdex.sum1);
  fprintf(stderr, "standard deviation: %f\n", sqrt( rdex.sumdt2 / rdex.sum1 - ↙
      ↪ pow(rdex.sumdt / rdex.sum1, 2) ));
  fprintf(stderr, ↙
      ↪ "------------------------------------------------------------------------\↙
      ↪ n");
}

//=============================================================
// main module timer callback
//void rdex_idle() {
void rdex_timer(int v) {

  // initialize
  if (rdex.starttime == 0) {
    rdex.starttime = time(NULL);
    rdex.frame = 0;
    rdex.sum1 = 0;
    rdex.sumdt = 0;
    rdex.sumdt2 = 0;
    atexit(rdex_atexit);
    clock_gettime(CLOCK_REALTIME, &rdex.clock0);
  }

  // check if there are enough free buffers to render a frame
  int r = audio_read;
  int w = audio_write;
  if (rdex.nrt || (w < r && w + AUDIO_COUNT < r) || ( r < w && w + AUDIO_COUNT <↙
```

```
      ↪  r + AUDIO_BUFFERS)) {

  for (int g = 0; g < rdex_overdrive; ++g) {
    reactiondiffusion_idle(&rdex.reactiondiffusion);
  }
  copysquare_idle       (&rdex.copysquare);
  arithmeticmean_idle   (&rdex.arithmeticmean);
  numericerror_idle     (&rdex.numericerror);
  difference_idle       (&rdex.difference);
  falsecolour_idle      (&rdex.falsecolour);
  library_idle          (&rdex.library);
  screenshot_idle       (&rdex.screenshot);
  if (rdex.reactiondiffusion.frame > (rdex_count*1) && rdex.arithmeticmean.↙
      ↪ stddev < 0.01) {
//    if (rdex.species % 32 == 0) {
//      if (rdex.species) { fputc('|', stderr); }
//      fprintf(stderr, "\n%08x|", rdex.species);
//    }
    rdex.done = 1; //(rdex.mode == mode_random);
    rdex.species += 1;
    rdex.uniform += 1;
    rdex.behaviour = "uniform";
//    rdex.library.snap = 1;
    //fprintf(stderr, "_\t%d", rdex.reactiondiffusion.frame);
//    reactiondiffusion_randomize(&rdex.reactiondiffusion);
  } else if (rdex.reactiondiffusion.frame % (rdex_count*2) == (rdex_count*1)) {
    rdex.numericerror.calc = 1;
  } else if (rdex.reactiondiffusion.frame % (rdex_count*2) == (rdex_count*1) + ↙
      ↪ rdex_overdrive) {
    if (rdex.numericerror.mean > 0.2) {
//      if (rdex.species % 32 == 0) {
//        if (rdex.species) { fputc('|', stderr); }
//        fprintf(stderr, "\n%08x|", rdex.species);
//      }
      rdex.done = 1;
      rdex.species += 1;
      rdex.erratic += 1;
      //fprintf(stderr, ".\t%d", rdex.reactiondiffusion.frame);
      //reactiondiffusion_randomize(&rdex.reactiondiffusion);
    }
  } else if (rdex.reactiondiffusion.frame % (rdex_count*3) == (rdex_count*1)) {
    rdex.difference.snap = 1;
  } else if (rdex.reactiondiffusion.frame % (rdex_count*3) == (rdex_count*2)) {
    rdex.difference.calc = 1;
  } else if (rdex.reactiondiffusion.frame % (rdex_count*3) == (rdex_count*2) + ↙
      ↪ rdex_overdrive) {
    if (rdex.difference.mean < 0.011) {
      rdex.done = (rdex.mode == mode_random);
      rdex.species += 1;
      rdex.stable += 1;
      rdex.behaviour = "stable";
      rdex.library.snap = (rdex.mode == mode_random);
      //fprintf(stderr, "o\t%d", rdex.reactiondiffusion.frame);
    } else if (rdex.reactiondiffusion.frame > 16000 && rdex.mode == mode_random)↙
        ↪ {
      rdex.done = 1;
      rdex.species += 1;
```

```
320         rdex.dynamic += 1;
            rdex.behaviour = "dynamic";
            rdex.library.snap = (rdex.mode == mode_random);
            //fprintf(stderr, "*\t%d", rdex.reactiondiffusion.frame);
          }
325       }
      /* else if (rdex.reactiondiffusion.frame == 14400) {
          if (rdex.species % 32 == 0) {
            if (rdex.species) { fputc('|', stderr); }
            fprintf(stderr, "\n%08x|", rdex.species);
330         }
          rdex.species += 1;
          int which = rdex.difference.mean < 0.01;
          if (which) { rdex.stable += 1; } else { rdex.dynamic += 1; }
          fputc(which ? 'o' : '*', stderr);
335       //reactiondiffusion_perturb(&rdex.reactiondiffusion);
          reactiondiffusion_randomize(&rdex.reactiondiffusion);
        }
      */
        glutPostRedisplay();
340

        // gather timing statistics
        clock_gettime(CLOCK_REALTIME, &rdex.clock1);
        double dt = (rdex.clock1.tv_sec - rdex.clock0.tv_sec) + 1.0e-9 * (rdex.clock1.↙
            ↳ tv_nsec - rdex.clock0.tv_nsec);
345     rdex.sum1 += 1;
        rdex.sumdt += dt;
        rdex.sumdt2 += dt*dt;
        clock_gettime(CLOCK_REALTIME, &rdex.clock0);

350     }

        // sleep for 1ms and check again if jack is ready
        glutTimerFunc(1, rdex_timer, v + 1);

355   }

      void rdex_pmotion(int x, int y) {
        library_pmotion(&rdex.library, x, y);
      }
360
      void rdex_amotion(int x, int y) {
        library_amotion(&rdex.library, x, y);
      }

365   void rdex_mouse(int button, int state, int x, int y) {
        library_mouse(&rdex.library, button, state, x, y);
      }

      void rdex_keynormal(unsigned char key, int x, int y) {
370     return;
        if (text_keynormal(&rdex.text, key, x, y)) {
          if (key == ' ') {
            struct libelem *e = &rdex.library.array[rand() % (rdex.library.count - 1)↙
                ↳ ];
            rdex.reactiondiffusion.value.ru = e->v[0];
```

```
        rdex.reactiondiffusion.value.rv = e->v[1];
        rdex.reactiondiffusion.value.f  = e->v[2];
        rdex.reactiondiffusion.value.k  = e->v[3];
          }
      reactiondiffusion_perturb(&rdex.reactiondiffusion);
    } else {
//  if (! sequence_keynormal(&rdex.sequence, &rdex.library, key, x, y)) {
          switch (key) {
                  case 27: // escape
                                  exit(0);
                    break;
                  case 9: // tab
        timeline_start(&rdex.timeline);
//                  audio_start(&rdex.audio);
//        intro_start(&rdex.intro);
                    break;
                  case 127: // delete
        timeline_stop(&rdex.timeline);
//                  audio_stop(&rdex.audio);
        //outro_start(&rdex.outro);
                    break;
                  default:
                    break;
          }
  }
}

void rdex_keyspecial(int key, int x, int y) {
return;
//  if (! sequence_keyspecial(&rdex.sequence, &rdex.library, key, x, y)) {
    if (! text_keyspecial(&rdex.text, key, x, y)) {
    switch (key) {
            default:
              break;
    }
  }
}

// EOF
```

79 src/rdex.ds

```
(begin

(if (is (application_name) "rdex")
  (begin
    (pin)
    (undecorate)
    (above)
    (geometry "1024x768+0+0")
  )
)
(if (is (application_name) "meterbridge")
  (begin
    (geometry "74x210+770+30")
  )
)
```

```
(if (contains (window_name) "JACK Audio Connection Kit [(default)]")
  (begin
    (geometry "500x110+860+30")
  )
)
(if (is (window_name) "Connections - JACK Audio Connection Kit")
  (begin
    (geometry "500x210+860+160")
  )
)
(if (is (window_name) "JACK Rack (rdex) - rdex.jr")
  (begin
    (geometry "500x160+860+400")
  )
)
(if (is (window_name) "top")
  (begin
    (geometry "500x160+860+590")
  )
)

)
```

80 src/rdex.h

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------
Main Module
========================================================================== */

#ifndef RDEX_H
#define RDEX_H 1

#include <time.h>
#include <GL/glew.h>
#include <GL/glut.h>

#include "reactiondiffusion.h"
#include "copysquare.h"
#include "arithmeticmean.h"
#include "numericerror.h"
#include "difference.h"
#include "falsecolour.h"
#include "library.h"
#include "screenshot.h"
#include "audio.h"
#include "sequence.h"
#include "intro.h"
#include "timeline.h"
#include "text.h"

//==========================================================================
// configuration
#define rdex_tex_size 256
// FIXME: breakage will occur when rdex_count % rdex_overdrive != 0
```

```
// FIXME: check audio when audioHz % videoHz != 0
// FIXME: check audio when overdrive % (audioHz / videoHz) != 0
#define rdex_count 96
// FIXME optimal overdrive depends on sample rate!
//   14   * 25 * 126 = 44100
//   16   * 25 * 120 = 48000
#define rdex_overdrive (AUDIO_COUNT * 2)
//#define rdex_videoHz 33

enum rdex_mode {
  mode_random = 100, mode_textual, mode_sequence
};

//======================================================================
// main module data
struct rdex {
  struct reactiondiffusion reactiondiffusion;
  struct copysquare        copysquare;
  struct arithmeticmean    arithmeticmean;
  struct numericerror      numericerror;
  struct difference        difference;
  struct falsecolour       falsecolour;
  struct library           library;
  struct screenshot        screenshot;
  struct audio             audio;
//  struct sequence          sequence;
  struct intro             intro;
  struct timeline          timeline;
  struct text              text;
  GLuint fbo;
  time_t starttime;
  struct timespec clock0, clock1;
  unsigned int frame;
  unsigned int framedrops;
  unsigned int species;
  unsigned int uniform;
  unsigned int stable;
  unsigned int dynamic;
  unsigned int erratic;
  int done;
  int fullscreen;
  char *behaviour;
  enum rdex_mode mode;
  int nrt;
  // timing statistics
  double sum1, sumdt, sumdt2;
};

//======================================================================
// prototypes
int rdex_init(void);
void rdex_reshape(int w, int h);
void rdex_display(void);
void rdex_atexit(void);
void rdex_timer(int v);
//void rdex_idle();
void rdex_pmotion(int x, int y);
```

```
90  void rdex_amotion(int x, int y);
    void rdex_mouse(int button, int state, int x, int y);
    void rdex_keynormal(unsigned char key, int x, int y);
    void rdex_keyspecial(int key, int x, int y);

95  #endif
```

81 src/rdex-logo-128x128.ppm

82 src/rdex-logo-32x32.xpm

```
    /* XPM */
    static char * rdex_logo_32x32_xpm[] = {
    "32 32 103 2",
    "       c None",
 5  ".      c #352D0C",
    "+      c #765F04",
    "@      c #A38300",
    "#      c #B69200",
    "$      c #A48300",
10  "%      c #776005",
    "&      c #372F0D",
    "*      c #221E0D",
    "=      c #927501",
    "-      c #F8C700",
15  ";      c #FFCC00",
    ">      c #F9C700",
    ",      c #947701",
    "'      c #262314",
    ")      c #5C4B06",
20  "!      c #E3B600",
    "~      c #CBA200",
    "{      c #2B2610",
    "]      c #252114",
```

```
"^      c #9E7E01",
"/      c #FECB00",
"(      c #B49000",
"_      c #362D0B",
":      c #CFA500",
"<      c #604E05",
"[      c #44390D",
"}      c #E5B700",
"|      c #C49D00",
"1      c #6B5706",
"2      c #68550A",
"3      c #AC8900",
"4      c #BB9600",
"5      c #44380B",
"6      c #EBBC00",
"7      c #EEBF00",
"8      c #6D5804",
"9      c #977902",
"0      c #F5C400",
"a      c #1B180D",
"b      c #352B07",
"c      c #E6B800",
"d      c #C8A000",
"e      c #372E0B",
"f      c #3A310E",
"g      c #372F0E",
"h      c #26210D",
"i      c #D4AA00",
"j      c #A68400",
"k      c #211E13",
"l      c #201B07",
"m      c #30290B",
"n      c #A98701",
"o      c #6B5604",
"p      c #EDBE00",
"q      c #1B1A13",
"r      c #6A5605",
"s      c #A18100",
"t      c #0B0A05",
"u      c #171408",
"v      c #675406",
"w      c #F1C000",
"x      c #A98800",
"y      c #2D260A",
"z      c #ECBD00",
"A      c #F4C300",
"B      c #F1C100",
"C      c #F6C500",
"D      c #4E410C",
"E      c #9C7D01",
"F      c #927502",
"G      c #3E3308",
"H      c #FCC900",
"I      c #A58402",
"J      c #A38301",
"K      c #DFB300",
"L      c #6E5905",
```

```
"M  c #342C0C",
"N  c #F8C600",
"O  c #C7A000",
"P  c #15130B",
"Q  c #806704",
"R  c #A78600",
"S  c #362C03",
"T  c #E4B600",
"U  c #A28200",
"V  c #D3A900",
"W  c #4F410A",
"X  c #372C02",
"Y  c #3D330B",
"Z  c #352B02",
"`  c #745E06",
" . c #866B02",
".. c #51430A",
"+. c #6E5A09",
"@. c #D6AB00",
"#. c #1A1811",
"$. c #7C650A",
"%. c #1D1A0E",
"&. c #C9A100",
"*. c #24200D",
"=. c #6D5700",
"-. c #53440A",
"                                                                ",
"                                                                ",
"                                                                ",
"                                                                ",
"                                    . + @ # $ % &               ",
"                                * = - ; ; ; ; ; > , '           ",
"                              ) ! ; ; ; ; ; ; ; ; ; ~ {         ",
"                          ] ^ / ; ; ; ; ; ; ; ; ; ; ; (         ",
"                        _ : ; ; ; ; ; ; ; ; ; ; ; ; ; ; <       ",
"                      [ } ; ; ; ; ; ; | 1 2 3 ; ; ; ; ; 4       ",
"                    5 6 ; ; ; ; ; 7 8         9 ; ; ; ; 0 a     ",
"                  b c ; ; ; ; ; d e           f ; ; ; ; ; g     ",
"                h i ; ; ; ; ; j k             l ; ; ; ; ; m     ",
"                n ; ; ; ; ; 9                 o ; ; ; ; p q     ",
"              r ; ; ; ; ; s t             u v w ; ; ; ; x       ",
"            y z ; ; ; ; ; A B B B B B B B C ; ; ; ; ; / D       ",
"            E ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; F         ",
"          G H ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; I           ",
"          J ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; K L             ",
"        M N ; ; ; H | | | | | | | 6 ; ; ; ; ; O P               ",
"        Q ; ; ; ; R               S T ; ; ; ; ; U               ",
"        V ; ; ; ; W                 X T ; ; ; ; ; U             ",
"      Y ; ; ; ; i                     Z T ; ; ; ; ; U           ",
"      ` ; ; ; ;  .                      X T ; ; ; ; ; U         ",
"      $ ; ; ; ; ..                        Z T ; ; ; ; ; +.      ",
"      @.; ; ; C #.                          X T ; ; ; ; $.      ",
"    %.C ; ; ; &.                              Z T ; ; ; $.      ",
"    *.$ $ $ $ =.                                X s $ $ -.      ",
"                                                                ",
"                                                                ",
"                                                                ",
```

```
"                                                                    "};
```

83 src/reactiondiffusion.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009,2010,2019  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Reaction Diffusion Shader
================================================================ */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "util.h"
#include "reactiondiffusion.h"
#include "reactiondiffusion.frag.c"

//================================================================
// reaction diffusion shader parameter randomization
void reactiondiffusion_randomize(
  struct reactiondiffusion *reactiondiffusion
) {
  reactiondiffusion->value.ru = 0.3*rand() / (double) RAND_MAX + 0.001;
  reactiondiffusion->value.rv = 0.3*rand() / (double) RAND_MAX + 0.001;
  reactiondiffusion->value.f  = 0.1*rand() / (double) RAND_MAX + 0.001;
  reactiondiffusion->value.k  = 0.1*rand() / (double) RAND_MAX + 0.001;
  reactiondiffusion->frame = 0;
#if 0
  fprintf(stderr, "\n%f\t%f\t%f\t%f\t",
    reactiondiffusion->value.ru,
    reactiondiffusion->value.rv,
    reactiondiffusion->value.f,
    reactiondiffusion->value.k
  );
#endif
}
/*
  reactiondiffusion->spawnangle += 1.0 / 32.0;
  double a = reactiondiffusion->spawnangle;
  reactiondiffusion->value.ru = (cos(sqrt(2) * a) * 0.5 + 0.5) * 0.01 + 0.15;
  reactiondiffusion->value.rv = (cos(sqrt(3) * a) * 0.5 + 0.5) * 0.01 + 0.15;
  reactiondiffusion->value.f  = (sin(sqrt(5) * a) * 0.5 + 0.5) * 0.01 + 0.001;
  reactiondiffusion->value.k  = (sin(sqrt(7) * a) * 0.5 + 0.5) * 0.01 + 0.001;
*/

void reactiondiffusion_near(
  struct reactiondiffusion *reactiondiffusion, float ru, float rv, float f, ↙
    ↳ float k, float d
) {
  reactiondiffusion->value.ru = ru + d * 0.3*(rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->value.rv = rv + d * 0.3*(rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->value.f  = f  + d * 0.1*(rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->value.k  = k  + d * 0.1*(rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->frame = 0;
  fprintf(stderr, "\n%f\t%f\t%f\t%f\t",
  reactiondiffusion->value.ru,
```

```
    reactiondiffusion->value.rv,
    reactiondiffusion->value.f,
    reactiondiffusion->value.k
  );
}

//=============================================================================
// reaction diffusion shader parameter perturbation
void reactiondiffusion_perturb(
  struct reactiondiffusion *reactiondiffusion
) {
  reactiondiffusion->value.ru += 0.003 * (rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->value.rv += 0.003 * (rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->value.f  += 0.003 * (rand() / (double) RAND_MAX - 0.5);
  reactiondiffusion->value.k  += 0.003 * (rand() / (double) RAND_MAX - 0.5);
#define clip(x,m,M) ((x)=(((x)<(m))?(m):(((x)>(M))?(M):(x))))
  clip(reactiondiffusion->value.ru, 0.0, 0.3);
  clip(reactiondiffusion->value.rv, 0.0, 0.3);
  clip(reactiondiffusion->value.f,  0.0, 0.1);
  clip(reactiondiffusion->value.k,  0.0, 0.1);
#undef clip
//  reactiondiffusion->frame = 0;
}

//=============================================================================
// reaction diffusion shader initialization
struct reactiondiffusion *reactiondiffusion_init(
  struct reactiondiffusion *reactiondiffusion
) {
  if (! reactiondiffusion) { return 0; }
  if (! shader_init(
          &reactiondiffusion->shader, 0, reactiondiffusion_frag
    )) {
    return 0;
  }
  shader_uniform(reactiondiffusion, texture);
  shader_uniform(reactiondiffusion, dx);
  shader_uniform(reactiondiffusion, dy);
  shader_uniform(reactiondiffusion, ru);
  shader_uniform(reactiondiffusion, rv);
  shader_uniform(reactiondiffusion, f);
  shader_uniform(reactiondiffusion, k);
  reactiondiffusion->value.texture = 0;
  reactiondiffusion->value.dx      = 0;
  reactiondiffusion->value.dy      = 0;
  reactiondiffusion->spawnangle = 0.0;
  reactiondiffusion->frame = 0;
  reactiondiffusion_randomize(reactiondiffusion);
  glGenTextures(2, reactiondiffusion->textures);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, reactiondiffusion->textures[0]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, reactiondiffusion->textures[1]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glDisable(GL_TEXTURE_2D);
```

```
  reactiondiffusion->buffer = 0;
  reactiondiffusion->width  = 0;
  reactiondiffusion->height = 0;
  reactiondiffusion->seed   = 60;
  return reactiondiffusion;
}

//=============================================================
// reaction diffusion shader reshape callback
void reactiondiffusion_reshape(
  struct reactiondiffusion *reactiondiffusion, int w, int h
) {
  reactiondiffusion->width  = roundtwo(w);
  reactiondiffusion->height = roundtwo(h);
  reactiondiffusion->value.dx = 1.0 / reactiondiffusion->width;
  reactiondiffusion->value.dy = 1.0 / reactiondiffusion->height;
  float *zero = calloc(1, reactiondiffusion->width * reactiondiffusion->height *↙
      ↳ 2 * sizeof(float));
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, reactiondiffusion->textures[0]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RG32F,
    reactiondiffusion->width, reactiondiffusion->height,
    0, GL_RG, GL_FLOAT, zero
  );
  glBindTexture(GL_TEXTURE_2D, reactiondiffusion->textures[1]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RG32F,
    reactiondiffusion->width, reactiondiffusion->height,
    0, GL_RG, GL_FLOAT, zero
  );
  free(zero);
  glDisable(GL_TEXTURE_2D);
}

//=============================================================
// reaction diffusion shader display callback
void reactiondiffusion_display(
  struct reactiondiffusion *reactiondiffusion, GLuint fbo
) {
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D,
    reactiondiffusion->textures[reactiondiffusion->buffer]
  );
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  glFramebufferTexture2DEXT(
    GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D,
    reactiondiffusion->textures[1 - reactiondiffusion->buffer], 0
  );
  glUseProgramObjectARB(reactiondiffusion->shader.program);
  shader_updatei(reactiondiffusion, texture);
  shader_updatef(reactiondiffusion, dx);
  shader_updatef(reactiondiffusion, dy);
  shader_updatef(reactiondiffusion, ru);
  shader_updatef(reactiondiffusion, rv);
  shader_updatef(reactiondiffusion, f);
  shader_updatef(reactiondiffusion, k);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

```
  gluOrtho2D(0, 1, 0, 1);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glViewport(0, 0, reactiondiffusion->width, reactiondiffusion->height);
  glBegin(GL_QUADS); { glColor4f(1,1,1,1);
    glTexCoord2f(0, 0); glVertex2f(0, 0);
    glTexCoord2f(1, 0); glVertex2f(1, 0);
    glTexCoord2f(1, 1); glVertex2f(1, 1);
    glTexCoord2f(0, 1); glVertex2f(0, 1);
  } glEnd();
  glUseProgramObjectARB(0);
  glDisable(GL_TEXTURE_2D);
  if (reactiondiffusion->frame < reactiondiffusion->seed) {
    if (reactiondiffusion->frame == 0) {
      glBegin(GL_QUADS); {
        glColor4f(0.5,0.5,0.5,1);
        glVertex2f(0, 0);
        glVertex2f(1, 0);
        glVertex2f(1, 1);
        glVertex2f(0, 1);
      } glEnd();
    }
    glEnable(GL_POLYGON_SMOOTH);
    glBegin(GL_QUADS); { for (int q = 0; q < reactiondiffusion->seed - ↙
        ↳ reactiondiffusion->frame; ++q) {
      float u = (double) rand() / (double) RAND_MAX;
      float v = (double) rand() / (double) RAND_MAX;
      glColor4f(u,v,0,1); //(0,1,0,1);
      float x = (double) rand() / (double) RAND_MAX;
      float y = (double) rand() / (double) RAND_MAX;
      float w = ((double) rand() / (double) RAND_MAX - 0.5) / 16.0;
      float h = ((double) rand() / (double) RAND_MAX - 0.5) / 16.0;
      float a = ((double) rand() / (double) RAND_MAX) * 2.0 * 3.1415926;
      float c = cos(a);
      float s = sin(a);
      for (int dx = -1; dx <= 1; dx += 1) {
        for (int dy = -1; dy <= 1; dy += 1) {
          glVertex2f(dx + x + w * c + h *  s, dy + y + w * -s + h * c);
          glVertex2f(dx + x + w * c + h * -s, dy + y + w * -s + h *-c);
          glVertex2f(dx + x + w *-c + h * -s, dy + y + w *  s + h *-c);
          glVertex2f(dx + x + w *-c + h *  s, dy + y + w *  s + h * c);
        }
      }
    } } glEnd();
    glDisable(GL_POLYGON_SMOOTH);
  }
  //glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  glDisable(GL_TEXTURE_2D);
  reactiondiffusion->buffer = 1 - reactiondiffusion->buffer;
  reactiondiffusion->texture =
    reactiondiffusion->textures[reactiondiffusion->buffer];
}

//=============================================================================
// reaction diffusion shader idle callback
void reactiondiffusion_idle(
  struct reactiondiffusion *reactiondiffusion
```

```
) {
  reactiondiffusion->frame += 1;
}

// EOF
```

84 src/reactiondiffusion.frag

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2009  Claude Heiland-Allen <claude@mathr.co.uk>
============================================================================ */

uniform sampler2D texture; // U := r, V := g, other channels ignored
uniform float dx;          // horizontal distance between texels
uniform float dy;          // vertical distance between texels
uniform float ru;          // rate of diffusion of U
uniform float rv;          // rate of diffusion of V
uniform float f;           // some coupling parameter
uniform float k;           // another coupling parameter

void main(void) {
  float center = -(4.0+4.0/sqrt(2.0));  // -1 * other weights
  float diag   = 1.0/sqrt(2.0);        // weight for diagonals
  vec2 p = gl_TexCoord[0].st;              // center coordinates
  vec2 c = texture2D(texture, p).rg;       // center value
  vec2 l                                   // compute Laplacian
    =(texture2D(texture, p + vec2(-dx,-dy)).rg
    + texture2D(texture, p + vec2( dx,-dy)).rg
    + texture2D(texture, p + vec2(-dx, dy)).rg
    + texture2D(texture, p + vec2( dx, dy)).rg) * diag
    + texture2D(texture, p + vec2(-dx, 0.0)).rg
    + texture2D(texture, p + vec2( dx, 0.0)).rg
    + texture2D(texture, p + vec2(0.0,-dy)).rg
    + texture2D(texture, p + vec2(0.0, dy)).rg
    + c.rg * center;
  float u = c.r;                           // compute some temporary
  float v = c.g;                           // values which might save
  float lu = l.r;                          // a few GPU cycles
  float lv = l.g;
  float uvv = u * v * v;
  //============================================================
  float du = ru * lu - uvv + f * (1.0 - u); // Gray-Scott equation
  float dv = rv * lv + uvv - (f + k) * v; // diffusion+-reaction
  //============================================================
  u += 0.6*du; // semi-arbitrary scaling (reduces blow-ups?)
  v += 0.6*dv;
  gl_FragColor = vec4(clamp(u, 0.0, 1.0), clamp(v, 0.0, 1.0), 0.0, 1.0);
      ↪ // output new (U,V)
}
```

85 src/reactiondiffusion.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
```

```
    ------------------------------------------------------------------------
5   Reaction Diffusion Shader
    ======================================================================== */

    #ifndef REACTIONDIFFUSION_H
    #define REACTIONDIFFUSION_H 1
10
    #include "shader.h"

    //========================================================================
    // reaction diffusion shader data
15  struct reactiondiffusion { struct shader shader;
      struct { GLint texture; GLint dx, dy, ru, rv, f, k; } uniform;
      struct { int   texture; float dx, dy, ru, rv, f, k; } value;
      GLuint textures[2];
      GLuint texture;
20    int buffer;
      int width;
      int height;
      int seed;
      int frame;
25    double spawnangle;
    };

    //========================================================================
    // prototypes
30  void reactiondiffusion_randomize(
      struct reactiondiffusion *reactiondiffusion
    );
    void reactiondiffusion_near(
      struct reactiondiffusion *reactiondiffusion, float ru, float rv, float f, ↙
          ↳ float k, float d
35  );
    void reactiondiffusion_perturb(
      struct reactiondiffusion *reactiondiffusion
    );
    struct reactiondiffusion *reactiondiffusion_init(
40    struct reactiondiffusion *reactiondiffusion
    );
    void reactiondiffusion_reshape(
      struct reactiondiffusion *reactiondiffusion, int w, int h
    );
45  void reactiondiffusion_display(
      struct reactiondiffusion *reactiondiffusion, GLuint fbo
    );
    void reactiondiffusion_idle(
      struct reactiondiffusion *reactiondiffusion
50  );

    #endif
```

86 src/s2c.sh

```
#!/bin/bash
echo "/* machine-generated file, do not edit */"
echo "static const char $1[] = \"#extension GL_EXT_gpu_shader4 : enable\n\""
sed 's|//.*||' |
```

```
sed 's|^|"|' |
sed 's|$|"|'
echo ";"
```

87 src/screenshot.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010,2019  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Screenshot Module
================================================================ */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <GL/glew.h>
#include "screenshot.h"

//================================================================
// screenshot data
struct screenshot_data {
  int width;
  int height;
  unsigned char *data;
};

// background consumer
void screenshot_output(struct screenshot *screenshot, size_t s, struct screenshot_data *sd) {
  if (sd && sd->data) {
    fprintf(stdout, "P6\n%d %d 255\n", sd->width, sd->height);
    for (int y = sd->height - 1; y >= 0; --y) {
      fwrite(sd->data + y * sd->width * 3, sd->width * 3, 1, stdout);
    }
    fflush(stdout);
    free(sd->data);
  }
}

//================================================================
// screenshot module initialization
struct screenshot *screenshot_init(struct screenshot *screenshot, int nrt) {
  if (! screenshot) { return 0; }
  screenshot->nrt = nrt;
  screenshot->rate = 1; // FIXME use environment variable for this
  screenshot->frame = 0;
  screenshot->width = 0;
  screenshot->height = 0;
  screenshot->buffer = 0;
  // fbo
//  glGenFramebuffersEXT(1, &screenshot->fbo);
  // pbo
  glGenBuffersARB(1, &screenshot->pbo);
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, screenshot->pbo);
  glBufferDataARB(GL_PIXEL_PACK_BUFFER_ARB, 1024 * 768 * 3, 0, GL_STREAM_READ_ARB); // FIXME hardcoded
```

```
50    glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
      screenshot->ready = 0;
      // background output
    #if 0
      screenshot->queue = pfifo_create((pfifo_consumer *) screenshot_output, ↙
          ↳ screenshot);
55  #endif
      return screenshot;
    }

    //============================================================
60  // screenshot module display callbacks


    // save PBO data
    void screenshot_display1(struct screenshot *screenshot) {
65    if (! screenshot->ready) { return; }
      glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, screenshot->pbo);
      unsigned char *p = glMapBufferARB(GL_PIXEL_PACK_BUFFER_ARB, GL_READ_ONLY_ARB);
      if (p) {
        struct screenshot_data sd;
70      sd.width = screenshot->width;
        sd.height = screenshot->height;
        sd.data = malloc(sd.width * sd.height * 3);
        memcpy(sd.data, p, sd.width * sd.height * 3);
    #if 0
75      pfifo_enqueue(screenshot->queue, sizeof(struct screenshot_data), &sd);
    #endif
        if (! glUnmapBufferARB(GL_PIXEL_PACK_BUFFER_ARB)) {
          fprintf(stderr, "unmap buffer error\n");
        }
80  #if 1
        screenshot_output(screenshot, sizeof(struct screenshot_data), &sd);
    #endif
      }
      glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
85  }

    // copy frame to PBO and start download
    void screenshot_display2(struct screenshot *screenshot) {
      int fps = 25;
90    int rate = screenshot->rate;
      int frame = screenshot->frame;
      int period = 60 * 60 * fps * rate; // 60 minutes
      int interval = 5 * 60 * fps * rate; // 5 minutes
      if (screenshot->nrt && (frame % rate) == 0 && (frame % period) < interval) {
95      // copy framebuffer to PBO
        glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, screenshot->pbo);
        glReadPixels(0, 0, screenshot->width, screenshot->height, GL_RGB, ↙
            ↳ GL_UNSIGNED_BYTE, 0);
        glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
    /*
100     glReadPixels(0, 0, screenshot->width, screenshot->height, GL_RGB, ↙
            ↳ GL_UNSIGNED_BYTE, screenshot->buffer);
        fprintf(stdout, "P6\n%d %d 255\n", screenshot->width, screenshot->height);
        for (int y = screenshot->height - 1; y >= 0; --y) {
          fwrite(screenshot->buffer + y * screenshot->width * 3, screenshot->width *↙
```

```
        ↪ 3, 1, stdout);
    }
*/
    screenshot->ready = 1;
  } else {
    screenshot->ready = 0;
  }
  screenshot->frame += 1;
}

//=========================================================
// screenshot module reshape callback
void screenshot_reshape(struct screenshot *screenshot, int w, int h) {
  screenshot->width = w;
  screenshot->height = h;
  if (screenshot->buffer) free(screenshot->buffer);
  screenshot->buffer = malloc(w * h * 3);
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, screenshot->pbo);
  glBufferDataARB(GL_PIXEL_PACK_BUFFER_ARB, (w + 4) * h * 3, 0, ↙
      ↪ GL_STREAM_READ_ARB);
  glBindBufferARB(GL_PIXEL_PACK_BUFFER_ARB, 0);
}

//=========================================================
// screenshot module idle callback
void screenshot_idle(struct screenshot *screenshot) {
}

// EOF
```

88 src/screenshot.h

```
/* =========================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
-------------------------------------------------------------
Screenshot Module
========================================================= */

#ifndef SCREENSHOT_H
#define SCREENSHOT_H 1

#include "pfifo.h"

//=========================================================
// screenshot module data
struct screenshot {
  int nrt;
  int rate;
  int frame;
  int width;
  int height;
  char *buffer;
  GLuint pbo;
  int ready;
  struct pfifo *queue;
};
```

```
//=============================================================================
// prototypes
struct screenshot *screenshot_init(struct screenshot *screenshot, int nrt);
void screenshot_display1(struct screenshot *screenshot);
void screenshot_display2(struct screenshot *screenshot);
void screenshot_reshape(struct screenshot *screenshot, int w, int h);
void screenshot_idle(struct screenshot *screenshot);

#endif
```

89 src/segment.c

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
CPU-Based Image Segmentation
============================================================================ */

#include <math.h>
#include <stdlib.h>
#include "segment.h"
#include "util.h"

#define SEGMENT_LEVELS 4

struct segment_edge {
  int   fx,fy,tx,ty;
  float w;
};

static inline void segment_weight(struct segment_edge *e, const struct image *i)↙
    ↳ {
  e->w = fabs(I(i,e->fx,e->fy,0)-I(i,e->tx,e->ty,0))
       + fabs(I(i,e->fx,e->fy,1)-I(i,e->tx,e->ty,1))
       + fabs(I(i,e->fx,e->fy,2)-I(i,e->tx,e->ty,2));
}

static int segment_compare(const void *x, const void *y) {
  const struct segment_edge *e1 = x;
  const struct segment_edge *e2 = y;
  if (e1->w < e2->w) return -1;
  if (e1->w > e2->w) return 1;
  return 0;
}

struct segment_component {
  int deleted;
  int merged;
  int size;
  float diff;
};

void segment_calc(struct segment *s, const struct image *i) {
  float threshold = 0.1;
  for (int b = 0; b < SEGMENT_BUCKETS; ++b) {
```

```
    s->bucket[b] = 0;
  }
  int n = i->width * i->height;
  int m = n * 4;
  struct segment_edge *edges = malloc(m * sizeof(struct segment_edge));
  int e = 0;                                    //  ...
  for (int y = 0; y < i->height; ++y) {  //  .ox
  for (int x = 0; x < i->width; ++x) {   //  xxx
    edges[e].fx = x; edges[e].fy = y; edges[e].tx = (x + 1) % i->width; edges[e]
        ].ty = y; segment_weight(&edges[e], i); e++;
    edges[e].fx = x; edges[e].fy = y; edges[e].tx = x; edges[e].ty = (y + 1) % i
        ->height; segment_weight(&edges[e], i); e++;
    edges[e].fx = x; edges[e].fy = y; edges[e].tx = (x + 1) % i->width; edges[e]
        ].ty = (y + 1) % i->height; segment_weight(&edges[e], i); e++;
    edges[e].fx = x; edges[e].fy = y; edges[e].tx = (x - 1 + i->width) % i->
        width; edges[e].ty = (y + 1) % i->height; segment_weight(&edges[e], i)
        ; e++;
  }
  }
  qsort(edges, m, sizeof(struct segment_edge), segment_compare);
  struct image *S = image_alloc(i->width, i->height, 1);
  int k = 0;
  for (int y = 0; y < i->height; ++y) {
  for (int x = 0; x < i->width; ++x) {
    I(S,x,y,0) = k++;
  }
  }
  struct segment_component *components = malloc(n * sizeof(struct
      segment_component));
  for (k = 0; k < n; ++k) {
    components[k].deleted = 0;
    components[k].merged = -1;
    components[k].size = 1;
    components[k].diff = 0;
  }
  for (int q = 0; q < m; ++q) {
    int ci = I(S, edges[q].fx, edges[q].fy, 0);
    int cj = I(S, edges[q].tx, edges[q].ty, 0);
    while (components[ci].deleted) { ci = components[ci].merged; }
    while (components[cj].deleted) { cj = components[cj].merged; }
    if (ci != cj) {
      if (ci > cj) { int t = cj; cj = ci; ci = t; }
      float d = fminf(components[ci].diff + threshold / components[ci].size,
                      components[cj].diff + threshold / components[cj].size);
      if (edges[q].w <= d) {
        components[ci].size += components[cj].size;
        components[ci].diff = fmaxf(fmaxf(components[ci].diff, components[cj].
            diff), edges[q].w);
        components[cj].deleted = 1;
        components[cj].merged = ci;
        I(S, edges[q].fx, edges[q].fy, 0) = ci;
        I(S, edges[q].tx, edges[q].ty, 0) = ci;
      }
    }
  }
  image_free(S); S = NULL;
  free(edges); edges = NULL;
```

```
  for (int c = 0; c < n; ++c) {
    if (!components[c].deleted) {
      int b = logtwo(components[c].size);
      s->bucket[b] += components[c].size;
    }
  }
  free(components); components = NULL;
  for (int b = 0; b < SEGMENT_BUCKETS; ++b) {
    s->bucket[b] /= n;
  }
}
```

90 src/segment.h

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------
CPU-Based Image Segmentation
========================================================================== */

#ifndef SEGMENT_H
#define SEGMENT_H 1

#include "image.h"

#define SEGMENT_BUCKETS 32
struct segment {
  float bucket[SEGMENT_BUCKETS];
};

void segment_calc(struct segment * s, const struct image *i);

#endif
```

91 src/sequence.c

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2019  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------
Sequencer module
========================================================================== */

#include <math.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "sequence.h"
#include "sequence.vert.c"
#include "sequence.frag.c"
#if 0
struct sequence *sequence_init(struct sequence *sequence) {
  if (! shader_init(&sequence->shader, sequence_vert, sequence_frag)) {
    return 0;
  }
  shader_uniform(sequence, texture);
```

```
20    shader_uniform(sequence, pack);
      shader_uniform(sequence, size);
      glyphs_init(&sequence->glyphs);
      for (int i = 0; i < SEQUENCE_LENGTH; ++i) {
        sequence->seq1[i].active = 0;
25      sequence->seq1[i].index = -(i == 0);
        sequence->seq2[i].active = 0;
        sequence->seq2[i].index = -1;
        // pitch sequence
        sequence->pitch[i].active = 0;
30      sequence->pitch[i].pitch = 50;
      }
      sequence->page = seqpage_library;
      sequence->step = 0;
      sequence->cursor = 0;
35    sequence->frame = 0;
      return sequence;
    }

    void sequence_reshape(struct sequence *sequence, int w, int h) {
40    sequence->width = w;
      sequence->height = h;
    }

    void sequence_display(struct sequence *sequence, struct library *library, struct↙
        ↪ reactiondiffusion *react, struct audio *audio) {
45    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

      // reaction sequence
      if (sequence->seq1[sequence->step].active) {
        struct libelem *e = &library->array[sequence->seq1[sequence->step].index];
50      react->value.ru = e->v[0];
        react->value.rv = e->v[1];
        react->value.f  = e->v[2];
        react->value.k  = e->v[3];
      }
55
      // pitch sequence
      if (sequence->pitch[sequence->step].active) {
        audio->pitch = sequence->pitch[sequence->step].pitch;
      }
60
    #if 0
      // curve/speed sequence
      if (sequence->frame == 0 && sequence->seq2[sequence->step].active) {
        switch (sequence->seq2[sequence->step].index) {
65      case sequence_curve_less: audio->dcurve = fmax(audio->dcurve - 1.0, -1.0); ↙
            ↪ break;
        case sequence_curve_more: audio->dcurve = fmin(audio->dcurve + 1.0,  1.0); ↙
            ↪ break;
        case sequence_speed_less: audio->dspeed = fmax(audio->dspeed - 1.0, -1.0); ↙
            ↪ break;
        case sequence_speed_more: audio->dspeed = fmin(audio->dspeed + 1.0,  1.0); ↙
            ↪ break;
        default: break;
70      }
      }
```

```
    if (sequence->frame == 0) {
      audio->speed += audio->dspeed;
      audio->curve += audio->dcurve;
    }
#endif

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, sequence->width, sequence->height);
    gluOrtho2D(0, sequence->width, 0, sequence->height);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // sequence 1: library balls
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, library->tex[library->which]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glUseProgramObjectARB(sequence->shader.program);
    sequence->attr.spin   = glGetAttribLocationARB(sequence->shader.program, "↙
        ↪ myspin0");
//  sequence->attr.step   = glGetAttribLocationARB(sequence->shader.program, "↙
    ↪ mystep0");
    sequence->attr.cursor = glGetAttribLocationARB(sequence->shader.program, "↙
        ↪ mycursor0");
    sequence->attr.active = glGetAttribLocationARB(sequence->shader.program, "↙
        ↪ myactive0");
    sequence->attr.pitch  = glGetAttribLocationARB(sequence->shader.program, "↙
        ↪ mypitch0");
    sequence->value.texture = 0;
    sequence->value.pack    = library_pack;
    sequence->value.size    = library_packed_size;
    shader_updatei(sequence, texture);
    shader_updatef(sequence, pack);
    shader_updatef(sequence, size);
    glBegin(GL_QUADS); {
      for (int j = 0; j < SEQUENCE_LENGTH; ++j) {
        float t = sequence->step + sequence->frame / 12.0;
        float x0 = library->width * (j + 0.5) / SEQUENCE_LENGTH;
        float y0 = library->width * 0.5 / SEQUENCE_LENGTH;
        float d = 1.0 / (0.3 + pow(fmin(fmin(fabs(t - j), fabs(t - j + ↙
            ↪ SEQUENCE_LENGTH)), fabs(t - j - SEQUENCE_LENGTH))/(SEQUENCE_LENGTH↙
            ↪ /2.0), 0.5));
        float dx = (0.75 + 0.25 * d) * y0;
        float dy = (0.75 + 0.25 * d) * y0;
          int i = sequence->seq1[j].index;
//          glVertexAttrib1f(sequence->attr.step,   j == sequence->step);
          glVertexAttrib1f(sequence->attr.cursor, j == sequence->cursor);
          glVertexAttrib1f(sequence->attr.active, sequence->seq1[j].active);
          glVertexAttrib1f(sequence->attr.pitch,  sequence->pitch[j].active ? ↙
              ↪ sequence->pitch[j].pitch / 5 : 0);
        if (sequence->seq1[j].active) {
          glVertexAttrib2f(sequence->attr.spin, library->array[i].spin[0], library↙
              ↪ ->array[i].spin[1]);
          glTexCoord4f( (i % library_pack)       / (float) library_pack, ((i / ↙
              ↪ library_pack) + 1) / (float) library_pack, 0, 1);
        } else {
```

```
          glVertexAttrib2f(sequence->attr.spin, 0, 0);
120       glTexCoord4f(-1,-1,0,1);
        }
          glVertex2f(x0 - dx, y0 + dy);
//        glVertexAttrib1f(sequence->attr.step,   j == sequence->step);
          glVertexAttrib1f(sequence->attr.cursor, j == sequence->cursor);
125       glVertexAttrib1f(sequence->attr.active, sequence->seq1[j].active);
          glVertexAttrib1f(sequence->attr.pitch,  sequence->pitch[j].active ? ↙
              ↳ sequence->pitch[j].pitch / 5 : 0);
        if (sequence->seq1[j].active) {
          glVertexAttrib2f(sequence->attr.spin, library->array[i].spin[0], library↙
              ↳ ->array[i].spin[1]);
          glTexCoord4f(((i % library_pack) + 1) / (float) library_pack, ((i / ↙
              ↳ library_pack) + 1) / (float) library_pack, 1, 1);
130       } else {
          glVertexAttrib2f(sequence->attr.spin, 0, 0);
          glTexCoord4f(-1,-1,1,1);
        }
          glVertex2f(x0 + dx, y0 + dy);
135 //        glVertexAttrib1f(sequence->attr.step,   j == sequence->step);
          glVertexAttrib1f(sequence->attr.cursor, j == sequence->cursor);
          glVertexAttrib1f(sequence->attr.active, sequence->seq1[j].active);
          glVertexAttrib1f(sequence->attr.pitch,  sequence->pitch[j].active ? ↙
              ↳ sequence->pitch[j].pitch / 5 : 0);
        if (sequence->seq1[j].active) {
140       glVertexAttrib2f(sequence->attr.spin, library->array[i].spin[0], library↙
              ↳ ->array[i].spin[1]);
          glTexCoord4f(((i % library_pack) + 1) / (float) library_pack,  (i / ↙
              ↳ library_pack)      / (float) library_pack, 1, 0);
        } else {
          glVertexAttrib2f(sequence->attr.spin, 0, 0);
          glTexCoord4f(-1,-1,1,0);
145     }
          glVertex2f(x0 + dx, y0 - dy);
//        glVertexAttrib1f(sequence->attr.step,   j == sequence->step);
          glVertexAttrib1f(sequence->attr.cursor, j == sequence->cursor);
          glVertexAttrib1f(sequence->attr.active, sequence->seq1[j].active);
150       glVertexAttrib1f(sequence->attr.pitch,  sequence->pitch[j].active ? ↙
              ↳ sequence->pitch[j].pitch / 5 : 0);
        if (sequence->seq1[j].active) {
          glVertexAttrib2f(sequence->attr.spin, library->array[i].spin[0], library↙
              ↳ ->array[i].spin[1]);
          glTexCoord4f( (i % library_pack)      / (float) library_pack,  (i / ↙
              ↳ library_pack)      / (float) library_pack, 0, 0);
        } else {
155       glVertexAttrib2f(sequence->attr.spin, 0, 0);
          glTexCoord4f(-1,-1,0,0);
        }
          glVertex2f(x0 - dx, y0 - dy);
      }
160   } glEnd();
      glUseProgramObjectARB(0);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

165 #if 0
      // sequence 2: speed/curve controls
```

```
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  for (int j = 0; j < SEQUENCE_LENGTH; ++j) {
    if (sequence->seq2[j].active) {
      glBindTexture(GL_TEXTURE_2D, sequence->glyphs.textures[sequence->seq2[j].index]);
      glBegin(GL_QUADS); {
        float t = sequence->step + sequence->frame / 12.0;
        float x0 = library->width * (j + 0.5) / SEQUENCE_LENGTH;
        float y0 = 32;
        float d = 1.0 / (0.3 + pow(fmin(fmin(fabs(t - j), fabs(t - j + SEQUENCE_LENGTH)), fabs(t - j - SEQUENCE_LENGTH))/(SEQUENCE_LENGTH/2.0), 0.5));
        float dx = 24 + 8 * d;
        float dy = 24 + 8 * d;
        glColor4f(1,1,1,1);
        glTexCoord2f(0, 0); glVertex2f(x0 + dx, y0 + dy);
        glTexCoord2f(0, 1); glVertex2f(x0 + dx, y0 - dy);
        glTexCoord2f(1, 1); glVertex2f(x0 - dx, y0 - dy);
        glTexCoord2f(1, 0); glVertex2f(x0 - dx, y0 + dy);
      } glEnd();
    }
  }
  glDisable(GL_BLEND);
#endif

  glBindTexture(GL_TEXTURE_2D, 0);
  glDisable(GL_TEXTURE_2D);
  if (++sequence->frame == 12) {
    sequence->frame = 0;
    if (++sequence->step == SEQUENCE_LENGTH) {
      sequence->step = 0;
    }
  }

}

int sequence_keynormal(struct sequence *sequence, struct library *library, int key, int x, int y) {
  switch (key) {
/*
    case '[':
      switch (sequence->page) {
        case seqpage_curves:
          sequence->seq2[sequence->cursor].index = sequence_curve_less;
          sequence->seq2[sequence->cursor].active = 1;
          return 1;
        default:
          return 0;
      }
    case ']':
      switch (sequence->page) {
        case seqpage_curves:
          sequence->seq2[sequence->cursor].index = sequence_curve_more;
          sequence->seq2[sequence->cursor].active = 1;
          return 1;
        default:
```

```
220             return 0;
        }
*/
      case ' ':
        switch (sequence->page) {
225       case seqpage_library:
            sequence->seq1[sequence->cursor].active = 0;
            sequence->seq1[sequence->cursor].index = -1;
            return 1;
          case seqpage_curves:
230         sequence->seq2[sequence->cursor].active = 0;
            sequence->seq2[sequence->cursor].index = -1;
            return 1;
        }

235     // pitch sequence
#define P(c,p) \
      case c: sequence->pitch[sequence->cursor].pitch = p; sequence->pitch[↙
          ↳ sequence->cursor].active = 1; return 1
        P('7', 80); P('8', 85); P('9', 90); P('0', 95); P('-', 100);
        P('u', 60); P('i', 65); P('o', 70); P('p', 75);
240     P('j', 40); P('k', 45); P('l', 50); P(';', 55);
        P('m', 20); P(',', 25); P('.', 30); P('/', 35);
#undef P
      case 'n': sequence->pitch[sequence->cursor].active = 0; return 1;

245     default:
        return 0;
    }
}

250 int sequence_keyspecial(struct sequence *sequence, struct library *library, int ↙
      ↳ key, int x, int y) {
    switch (key) {
      case GLUT_KEY_LEFT:
        sequence->cursor = (sequence->cursor + SEQUENCE_LENGTH - 1) % ↙
            ↳ SEQUENCE_LENGTH;
        return 1;
255   case GLUT_KEY_RIGHT:
        sequence->cursor = (sequence->cursor + 1) % SEQUENCE_LENGTH;
        return 1;
      case GLUT_KEY_UP:
        switch (sequence->page) {
260     case seqpage_library:
          sequence->seq1[sequence->cursor].index = rand() % library->count;
          sequence->seq1[sequence->cursor].active = 1;
          break;
        case seqpage_curves:
265       sequence->seq2[sequence->cursor].index = sequence_speed_more;
          sequence->seq2[sequence->cursor].active = 1;
          break;
        }
        return 1;
270   case GLUT_KEY_DOWN:
        switch (sequence->page) {
        case seqpage_library:
          sequence->seq1[sequence->cursor].active = 0;
```

```
        sequence->seq1[sequence->cursor].index = -1;
        break;
      case seqpage_curves:
        sequence->seq2[sequence->cursor].index = sequence_speed_less;
        sequence->seq2[sequence->cursor].active = 1;
        break;
      }
      return 1;
    case GLUT_KEY_PAGE_UP:
      sequence->page = seqpage_library;
      return 1;
    case GLUT_KEY_PAGE_DOWN:
      sequence->page = seqpage_curves;
      return 1;
    default:
      return 0;
  }
}
#endif
```

92 src/sequence.frag

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Sphere shader for sequencer
================================================================ */

uniform sampler2D texture;
uniform float pack;
uniform float size;

varying vec2 myspin2;
varying float mycursor2;
//varying float mystep2;
varying float myactive2;
varying float mypitch2;

void main(void) {
  vec2 p = gl_TexCoord[0].xy;                // float in [0..1]
  vec2 q0 = gl_TexCoord[0].zw;
  vec2 q1 = 3.0 * (q0 - vec2(0.5, 0.5));
  float d = dot(q1,q1);
  if (d < 1.0 && 0.0 < myactive2) {
    vec3 n = normalize(vec3(q1, sqrt(1.0-d)));
    vec2 q = q1 / (1.0 + sqrt(1.0 - d));
    vec2 q2 = 0.5 * q + myspin2;
    vec2 cell = floor(p * pack);             // int   in [0..(pack-1)]
    //vec2 frac = p * pack - cell;             // float in [0..1]
    //vec2 frac = vec2(0.0, 0.0);
    vec2 frac = q2;
    frac -= floor(frac);
    // FIXME check the ratio - maybe (size/pack) / (size/pack + 2) instead?
    frac -= vec2(0.5, 0.5);
    frac *= (size / pack - 2.0) / (size / pack);
    frac += vec2(0.5, 0.5);
```

```
      vec2 px0 = floor(  frac * size / pack ); // pixels
      vec2 px1 = px0 + vec2(1.0, 1.0);          // pixels
      // coordinates to interpolate
      vec2 p0 = cell / pack + px0 / size;
      vec2 p1 = cell / pack + px1 / size;
      vec2 dp = frac;
      // linear interpolation
      vec3 tl = texture2D(texture, p0).rgb;
      vec3 tr = texture2D(texture, vec2(p1.x, p0.y)).rgb;
      vec3 bl = texture2D(texture, vec2(p0.x, p1.y)).rgb;
      vec3 br = texture2D(texture, p1).rgb;
      vec3 t = tl * (1.0 - dp.x) + dp.x * tr;
      vec3 b = bl * (1.0 - dp.x) + dp.x * br;
      vec3 l = tl * (1.0 - dp.y) + dp.y * bl;
      vec3 r = tr * (1.0 - dp.y) + dp.y * br;
      vec3 m1 = t * (1.0 - dp.y) + dp.y * b;
      vec3 m2 = l * (1.0 - dp.x) + dp.x * r;
      vec3 m = 0.5 * (m1 + m2);
      // output
      float a = 1.0;
      float k = gl_FragCoord.z - 0.1 * cos(sqrt(d) * 3.1415926*0.5);
      float c = 0.0625 + pow(dot(vec3(0.0, 0.0, 1.0), n), 2.0);
      gl_FragDepth = k;
      gl_FragColor = vec4(m * c, a);
    } else if (abs(d - 1.3) < 0.3 && 0.0 < mycursor2) {
      float c = 1.0 - abs(d - 1.3) / 0.3;
      float a = fract((atan(q1.y, q1.x) / (3.1415926 * 2.0) + 0.5) * mypitch2) < ↙
          ↳ 0.5 ? 1.0 : 0.0;
      gl_FragDepth = gl_FragCoord.z;
      gl_FragColor = vec4(0.0, a * c*c, 0.0, c*c);
    } else {
      discard;
    }
  }
```

93 src/sequence.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Sequencer module
============================================================================ */

#ifndef SEQUENCE_H
#define SEQUENCE_H 1

#include "shader.h"
#include "library.h"
#include "reactiondiffusion.h"
#include "audio.h"
#include "glyphs.h"

#define SEQUENCE_LENGTH 16

struct seqelem {
  int active; // is this step going to change things
```

```
  int index;  // index for this step
};

enum sequence_page { seqpage_library, seqpage_curves };

struct sequence { struct shader shader;
  struct {
    GLint texture, pack, size; // fragment
  } uniform;
  struct {
    int texture; float pack, size; // fragment
  } value;
  struct {
    GLint spin, cursor, step, active, pitch; // vertex
  } attr;
  struct seqelem seq1[SEQUENCE_LENGTH]; // library index

  struct { int active; float pitch; } pitch[SEQUENCE_LENGTH]; // pitch sequence

  struct glyphs glyphs;
  struct seqelem seq2[SEQUENCE_LENGTH]; // curve/speed controls
  enum sequence_page page;
  int step;
  int cursor;
  int frame;
  int width;
  int height;
};

struct sequence *sequence_init(struct sequence *sequence);
void sequence_reshape(struct sequence *sequence, int w, int h);
void sequence_display(struct sequence *sequence, struct library *library, struct reactiondiffusion *react, struct audio *audio);
int sequence_keynormal(struct sequence *sequence, struct library *library, int key, int x, int y);
int sequence_keyspecial(struct sequence *sequence, struct library *library, int key, int x, int y);

#endif
```

94 src/sequence.vert

```
/* ==========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
--------------------------------------------------------------------------
Sequence vertex shader
========================================================================== */

attribute vec2 myspin0;
//attribute float mystep0;
attribute float mycursor0;
attribute float myactive0;
attribute float mypitch0;

varying vec2 myspin2;
//varying float mystep2;
```

```
varying float mycursor2;
varying float myactive2;
varying float mypitch2;

void main(void) {
  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
  myspin2 = myspin0;
//  mystep2 = mystep0;
  mycursor2 = mycursor0;
  myactive2 = myactive0;
  mypitch2 = mypitch0;
}
```

95 src/shader.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Generic Shader
================================================================ */

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

//================================================================
// print a shader object's debug log
void shader_debug(GLhandleARB obj) {
//  return; // FIXME: only dump logs when shader compile/link failed
  int infologLength = 0;
  int maxLength;
  if (glIsShader(obj)) {
    glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
  } else {
    glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
  }
  char *infoLog = malloc(maxLength);
  if (!infoLog) {
    return;
  }
  if (glIsShader(obj)) {
    glGetShaderInfoLog(obj, maxLength, &infologLength, infoLog);
  } else {
    glGetProgramInfoLog(obj, maxLength, &infologLength, infoLog);
  }
  if (infologLength > 0) {
    fprintf(stderr, "%s\n", infoLog);
  }
  free(infoLog);
}

//================================================================
// generic shader uniform location access macro
#define shader_uniform(self,name) \
  (self)->uniform.name = \
```

```
    glGetUniformLocationARB((self)->shader.program, #name)

//============================================================================
// generic shader uniform update access macro (integer)
#define shader_updatei(self,name) \
  glUniform1iARB((self)->uniform.name, (self)->value.name)

//============================================================================
// generic shader uniform update access macro (float)
#define shader_updatef(self,name) \
  glUniform1fARB((self)->uniform.name, (self)->value.name)

//============================================================================
// generic shader initialization
struct shader *shader_init(
  struct shader *shader, const char *vert, const char *frag
) {
  if (!shader) { return 0; }
  shader->linkStatus     = 0;
  shader->vertexSource   = vert;
  shader->fragmentSource = frag;
  if (shader->vertexSource || shader->fragmentSource) {
    shader->program = glCreateProgramObjectARB();
    if (shader->vertexSource) {
      shader->vertex =
        glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
      glShaderSourceARB(shader->vertex,
        1, (const GLcharARB **) &shader->vertexSource, 0
      );
      glCompileShaderARB(shader->vertex);
      shader_debug(shader->vertex);
      glAttachObjectARB(shader->program, shader->vertex);
    }
    if (shader->fragmentSource) {
      shader->fragment =
        glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
      glShaderSourceARB(shader->fragment,
        1, (const GLcharARB **) &shader->fragmentSource, 0
      );
      glCompileShaderARB(shader->fragment);
      shader_debug(shader->fragment);
      glAttachObjectARB(shader->program, shader->fragment);
    }
    glLinkProgramARB(shader->program);
    shader_debug(shader->program);
    glGetObjectParameterivARB(shader->program,
      GL_OBJECT_LINK_STATUS_ARB, &shader->linkStatus
    );
    if (!shader->linkStatus) { return 0; }
  } else { return 0; }
  return shader;
}
```

96 src/shader.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
```

```
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
-------------------------------------------------------------------------
Generic Shader
=========================================================================*/

#ifndef SHADER_H
#define SHADER_H 1

#include <GL/glew.h>

//=========================================================================
// generic shader data
struct shader {
  GLint linkStatus;
  GLhandleARB program;
  GLhandleARB fragment;
  GLhandleARB vertex;
  const GLcharARB *fragmentSource;
  const GLcharARB *vertexSource;
};

//=========================================================================
// generic shader uniform location access macro
#define shader_uniform(self,name) \
  (self)->uniform.name = \
    glGetUniformLocationARB((self)->shader.program, #name)

//=========================================================================
// generic shader uniform update access macro (integer)
#define shader_updatei(self,name) \
  glUniform1iARB((self)->uniform.name, (self)->value.name)

//=========================================================================
// generic shader uniform update access macro (float)
#define shader_updatef(self,name) \
  glUniform1fARB((self)->uniform.name, (self)->value.name)

//=========================================================================
// generic shader uniform update access macro (float2)
#define shader_updatef2(self,name) \
  glUniform2fARB((self)->uniform.name, (self)->value.name[0], (self)->value.name
    ↳ [1])

//=========================================================================
// generic shader uniform update access macro (float4)
#define shader_updatef4(self,name) \
  glUniform4fARB((self)->uniform.name, (self)->value.name[0], (self)->value.name
    ↳ [1], (self)->value.name[2], (self)->value.name[3])

//=========================================================================
// generic shader uniform update access macro (matrix4)
#define shader_updatem4(self,name) \
  glUniformMatrix4fvARB((self)->uniform.name, 1, 0, &((self)->value.name[0]))

//=========================================================================
// generic shader initialization
struct shader *shader_init(
```

```
  struct shader *shader, const char *vert, const char *frag
);

#endif
```

97 src/svg2o.sh

```
#!/bin/bash
SVGFILE="${1}.svg"
PNGFILE="${1}.png"
PPMFILE="${1}.ppm"
PGMFILE="${1}.pgm"
RGBFILE="${1}.rgb"
AFILE="${1}.a"
RGBAFILE="${1}.rgba"
OFILE="${1}.o"
rsvg-convert "${SVGFILE}" > "${PNGFILE}"
pngtopnm          "${PNGFILE}" > "${PPMFILE}"
pngtopnm -alpha "${PNGFILE}" > "${PGMFILE}"
tail -c 49152 "${PPMFILE}" > "${RGBFILE}"
tail -c 16384 "${PGMFILE}" > "${AFILE}"
./interleave31 "${RGBFILE}" "${AFILE}" 128 > "${RGBAFILE}"
#objcopy --input binary --output elf32-i386 --binary-architecture i386 "${
    ↪ RGBAFILE}" "${OFILE}"
objcopy --input binary --output elf64-x86-64 --binary-architecture i386 "${
    ↪ RGBAFILE}" "${OFILE}"
rm -f "${PNGFILE}" "${PPMFILE}" "${PGMFILE}" "${RGBFILE}" "${AFILE}" "${RGBAFILE
    ↪ }"
```

98 src/tamura.c

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2017 Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
Based on: fire-2.2/FeatureExtractors/tamurafeature.cpp
Homepage: http://www-i6.informatik.rwth-aachen.de/~deselaers/fire/
----------------------------------------------------------------------------
Tumara Feature Extraction
============================================================================ */

#define _DEFAULT_SOURCE 1

#include <assert.h>
#include <math.h>
#include "tamura.h"

float tamura_coarseness(
  struct tamura *tamura, struct image *image, int channel
);
float tamura_directionality(
  struct tamura *tamura, struct image *image, int channel
);
float tamura_contrast(
  struct tamura *tamura, struct image *image, int channel
);
```

```
float tamura_localmean(struct tamura *tamura, int x, int y, int k);
void tamura_runningsum(
  struct tamura *tamura, struct image *image, int channel
);

struct tamura *tamura_init(
  struct tamura *tamura, int width, int height
) {
  assert(tamura);
  assert(width > 0);
  assert(height > 0);
  tamura->width    = width;
  tamura->height   = height;
  tamura->Rsum  = image_alloc(width, height, 1);  assert(tamura->Rsum);
  tamura->Ak    = image_alloc(width, height, 5);  assert(tamura->Ak);
  tamura->Ekh   = image_alloc(width, height, 5);  assert(tamura->Ekh);
  tamura->Ekv   = image_alloc(width, height, 5);  assert(tamura->Ekv);
  tamura->Sbest = image_alloc(width, height, 1);  assert(tamura->Sbest);
  return tamura;
}

void tamura_calculate(
  struct tamura *tamura,
  struct tamura_features *features,
  struct image *image
) {
  assert(tamura);
  assert(features);
  assert(image);
  assert(tamura->width  == image->width);
  assert(tamura->height == image->height);
  assert(2              <= image->channels);
  for (int c = 0; c < 2; ++c) {
    features[c].coarseness     = tamura_coarseness    (tamura, image, c);
    features[c].contrast       = tamura_contrast      (tamura, image, c);
    features[c].directionality = tamura_directionality(tamura, image, c);
  }
}

void tamura_runningsum(
  struct tamura *tamura, struct image *image, int channel
) {
  struct image *r = tamura->Rsum;
  float l, u, ul;
  for (int y = 0; y < tamura->height; ++y) {
    for (int x = 0; x < tamura->width; ++x) {
      l  = x      ? I(r,x-1,y,  0) : 0;
      u  = y      ? I(r,x,  y-1,0) : 0;
      ul = x && y ? I(r,x-1,y-1,0) : 0;
      I(r,x,y,0) = I(image,x,y,channel) + l + u + ul;
    }
  }
}

float tamura_localmean(struct tamura *tamura, int x, int y, int k) { // FIXME ↙
    ↳ handle wrap-around more excitingly
  struct image *r = tamura->Rsum;
```

```
  int k2 = k/2;
  int x0 = x - k2 - 1;
  int x1 = x + k2 - 1;
  int y0 = y - k2 - 1;
  int y1 = y + k2 - 1;
  if (x0 < 0) x0 = 0;
  if (y0 < 0) y0 = 0;
  if (x1 >= tamura->width ) x1 = tamura->width  - 1;
  if (y1 >= tamura->height) y1 = tamura->height - 1;
  int count = (x1 - x0 + 1) * (y1 - y0 + 1);
  float l  = x0 > 0           ? I(r,x0-1,y1,  0) : 0;
  float t  = y0 > 0           ? I(r,x1,  y0-1,0) : 0;
  float tl = x0 > 0 && y0 > 0 ? I(r,x0-1,y0-1,0) : 0;
  float b  =                    I(r,x1,  y1,  0);
  return (b - l - t + tl) / count;
}

float tamura_contrast(
  struct tamura *tamura, struct image *image, int channel
) {
  float size = 121.0;
  float contrast = 0;
  for (int y = 0; y < tamura->height; ++y) {
    for (int x = 0; x < tamura->width; ++x) {
      float mean = 0;
      float sigma = 0;
      float kurtosis = 0;
      for (int j = y - 5; j <= y + 5; ++j) {
        int jj = j;
        while (jj <  0             ) { jj += tamura->height; }
        while (jj >= tamura->height) { jj -= tamura->height; }
        for (int i = x - 5; i <= x + 5; ++i) {
          int ii = i;
          while (ii <  0            ) { ii += tamura->width; }
          while (ii >= tamura->width) { ii -= tamura->width; }
          float t = I(image,ii,jj,channel);
          mean += t;
          sigma += t*t;
        }
      }
      mean /= size;
      sigma /= size;
      sigma -= mean * mean;
      for (int j = y - 5; j <= y + 5; ++j) {
        int jj = j;
        while (jj <  0             ) { jj += tamura->height; }
        while (jj >= tamura->height) { jj -= tamura->height; }
        for (int i = x - 5; i <= x + 5; ++i) {
          int ii = i;
          while (ii <  0            ) { ii += tamura->width; }
          while (ii >= tamura->width) { ii -= tamura->width; }
          float t = I(image,ii,jj,channel) - mean;
          t *= t;
          t *= t;
          kurtosis += t;
        }
      }
```

```
      kurtosis /= size;
      if (kurtosis > 0.000001) {
        contrast += sigma / sqrtf(sqrtf(kurtosis));
      }
    }
  }
  return contrast / (tamura->width * tamura->height);
}

float tamura_directionality(
  struct tamura *tamura, struct image *image, int channel
) {
  const float mh[3][3] = { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 } };
  const float mv[3][3] = { { 1, 0, -1 }, { 2, 0, -2 }, { 1, 0, -1 } };
  float phi = 0;
  for (int y = 0; y < tamura->height; ++y) {
    for (int x = 0; x < tamura->width; ++x) {
      float v = 0, h = 0;
      for (int j = 0; j < 3; ++j) {
        int jj = y + j - 1;
        while (jj <  0             ) { jj += tamura->height; }
        while (jj >= tamura->height) { jj -= tamura->height; }
        for (int i = 0; i < 3; ++i) {
          int ii = x + i - 1;
          while (ii <  0            ) { ii += tamura->width; }
          while (ii >= tamura->width) { ii -= tamura->width; }
          float t = I(image,ii,jj,channel);
          h += mh[i][j] * t;
          v += mv[i][j] * t;
        }
      }
      phi += sqrtf(0.5f*(h*h+v*v));
      //if (h != 0) { phi += atanf(v/h) + M_PI/2.0 + 0.001; } // FIXME ?
    }
  }
  return phi / (tamura->width * tamura->height);
}

float tamura_coarseness(
  struct tamura *tamura, struct image *image, int channel
) {
  tamura_runningsum(tamura, image, channel);
  //step 1
  int lenOfk=1;
  for(int k=1;k<=5;++k) {
    lenOfk*=2;
    for(int y=0;y<tamura->height;++y) {
      for(int x=0;x<tamura->width;++x) {
        I(tamura->Ak,x,y,k-1) = tamura_localmean(tamura,x,y,lenOfk);
      }
    }
  }
  //step 2
  lenOfk=1;
  for(int k=1;k<=5;++k) {
    int k2=lenOfk;
    lenOfk*=2;
```

```
    for(int y=0;y<tamura->height;++y) {
      for(int x=0;x<tamura->width;++x) {
        int posx1=x+k2;
        int posx2=x-k2;
        int posy1=y+k2;
        int posy2=y-k2;
        if(posx1<tamura->width && posx2>=0) {
          I(tamura->Ekh,x,y,k-1)=fabsf(I(tamura->Ak,posx1,y,k-1)-I(tamura->Ak,
              posx2,y,k-1));
        } else {
          I(tamura->Ekh,x,y,k-1)=0;
        }
        if(posy1<tamura->height && posy2>=0) {
          I(tamura->Ekv,x,y,k-1)=fabsf(I(tamura->Ak,x,posy1,k-1)-I(tamura->Ak,x,
              posy2,k-1));
        } else {
          I(tamura->Ekv,x,y,k-1)=0;
        }
      }
    }
  }
  float coarseness=0.0;
  //step3
  for(int y=0;y<tamura->height;++y) {
    for(int x=0;x<tamura->width;++x) {
      float maxE=0;
      int maxk=0;
      for(int k=1;k<=5;++k) {
        if(I(tamura->Ekh,x,y,k-1)>maxE) {
          maxE=I(tamura->Ekh,x,y,k-1);
          maxk=k;
        }
        if(I(tamura->Ekv,x,y,k-1)>maxE) {
          maxE=I(tamura->Ekv,x,y,k-1);
          maxk=k;
        }
      }
//      I(tamura->Sbest,x,y,0)=maxk;
      coarseness+=maxk;
    }
  }
  return coarseness / ((tamura->width - 32) * (tamura->height - 32));
}

/*
vector<uint> histogramImageBinImage(const ImageFeature &image) {
  vector<uint> result(image.xsize()*image.ysize());
  vector<double> t(3,0);

  vector<uint> steps(3,8);
  HistogramFeature tmp(steps);
  tmp.min()[0]=0;    tmp.max()[0]=1.0;
  tmp.min()[1]=0;    tmp.max()[1]=1.0;
  tmp.min()[2]=0;    tmp.max()[2]=1.0;

  tmp.initStepsize();

```

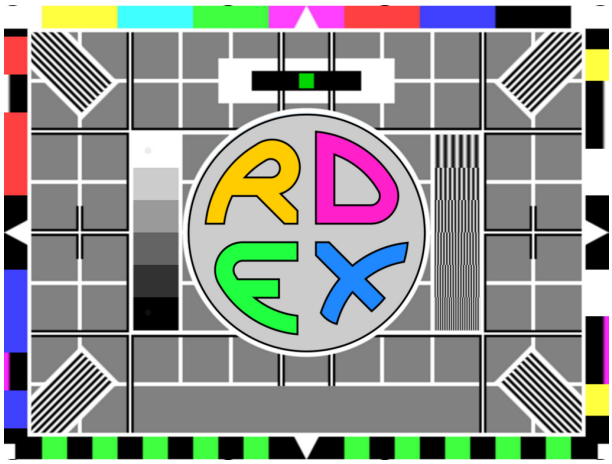
```
  for(uint x=0;x<image.xsize();++x) {
    for(uint y=0;y<image.ysize();++y) {
      for(uint z=0;z<3;++z) {t[z]=image(x,y,z);}
      result[y*image.xsize()+x]=tmp.posToBin(tmp.pointToPos(t));
    }
  }

  return result;
}

HistogramFeature histogramize(const ImageFeature image,
                              const vector<uint> &bins,
                              const uint left,
                              const uint top,
                              const uint right,
                              const uint bottom) {
  vector<uint> steps(3,8);
  HistogramFeature result(steps);
  result.min()[0]=0;    result.max()[0]=1.0;
  result.min()[1]=0;    result.max()[1]=1.0;
  result.min()[2]=0;    result.max()[2]=1.0;
  result.initStepsize();

  uint xsize=image.xsize();

  for(uint x=left;x<right;++x) {
    for(uint y=top;y<bottom;++y) {
      //      DBG(10) << y << " " << x << " "<< y*xsize+x << " " << bins[y*xsize+x] << endl;
      result.feedbin(bins[y*xsize+x]);
    }
  }

  return result;
}


HistogramFeature histogramize(const ImageFeature &image) {
  // min -> max: 0,0,0 -> pi,5,128
  // steps: 8,8,8

  vector<uint> steps(3,8);
  HistogramFeature result(steps);
  result.min()[0]=0;    result.max()[0]=1.0;
  result.min()[1]=0;    result.max()[1]=1.0;
  result.min()[2]=0;    result.max()[2]=1.0;
  result.initStepsize();

  vector<double> tmp(3,0.0);

  for(uint x=0;x<image.xsize();++x) {
    for(uint y=0;y<image.ysize();++y) {
      for(uint z=0;z<3;++z) {tmp[z]=image(x,y,z);}
      result.feed(tmp);
    }
  }
```

```
  return result;
}
*/
```

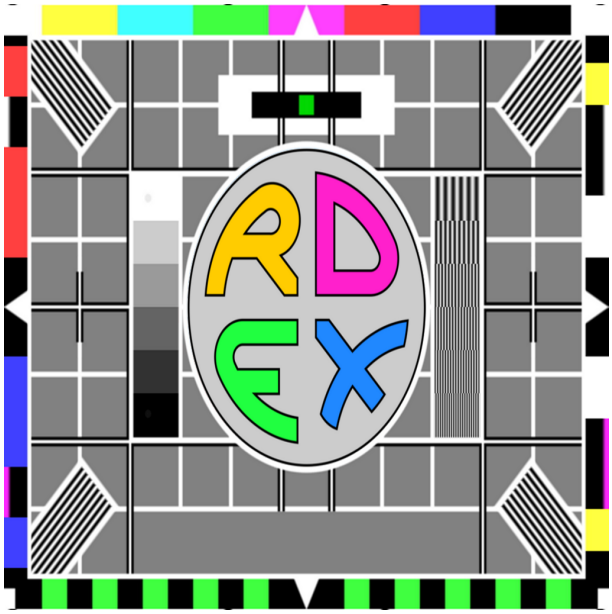
99 src/tamura.h

```
/* ========================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
------------------------------------------------------------------------
Based on: fire-2.2/FeatureExtractors/tamurafeature.cpp
Homepage: http://www-i6.informatik.rwth-aachen.de/~deselaers/fire/
------------------------------------------------------------------------
Tumara Feature Extraction
======================================================================== */

#ifndef TAMURA_H
#define TAMURA_H

#include "image.h"

struct tamura_features {
  float coarseness;
  float contrast;
  float directionality;
};

struct tamura {
  int width;
  int height;
  struct image *Rsum;  // w*h*1
  struct image *Ak;    // w*h*5
  struct image *Ekh;   // w*h*5
  struct image *Ekv;   // w*h*5
  struct image *Sbest; // w*h*1
};

struct tamura *tamura_init(
  struct tamura *tamura, int width, int height
);

void tamura_calculate(
  struct tamura *tamura,
  struct tamura_features *features,
  struct image *image
);

#endif
```

100 src/test-card-intro.png



101 src/test-card-intro.ppm



102 src/test_gl.sh

```
#!/bin/bash
grep -h gl *.c | sed "s|.*gl\(.*\)(.*|gl\1|g" | grep -v "=" | grep -v "^#" |
grep -v "^/" | sed "s|(.*||g" | grep -v "^gls_" | grep -v "^global_" |
grep -v "^glXGetCurrent" | grep -v " " | sort | uniq |
(echo "
#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GL/glut.h>
static void test_display(void);
```

```
int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(640, 480);
  glutCreateWindow(\"test_gl\");
  glewInit();
  glutDisplayFunc(test_display);
  glutMainLoop();
  return 0;
}
static void test_display(void) {
" &&
 cat | sed 's|\(.*\)|printf("\1\\t\\t\\t%p\\n", \1);|g' && echo "
  exit(0);
}
") | gcc -xc - -o test_gl -lGLEW -lGL -lGLU -lglut && ./test_gl | grep "(nil)"
```

103 src/text.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Text module
================================================================ */

#include <math.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "text.h"
#include "text.vert.c"
#include "text.frag.c"

struct text *text_init(struct text *text) {
  if (! shader_init(&text->shader, text_vert, text_frag)) {
    return 0;
  }
  shader_uniform(text, texture);
  fonts_init(&text->fonts);
  for (int i = 0; i < TEXT_LENGTH; ++i) {
    text->textstr[i] = ' ';
  }
  return text;
}

void text_reshape(struct text *text, int w, int h) {
  text->width = w;
  text->height = h;
}

void text_display(struct text *text) {
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glViewport(0, 0, text->width, text->height);
  gluOrtho2D(0, text->width, 0, text->height);
```

```
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  // text
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, text->fonts.texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glUseProgramObjectARB(text->shader.program);
  text->value.texture = 0;
  shader_updatei(text, texture);
  glBegin(GL_QUADS); {
    for (int j = 0; j < TEXT_LENGTH; ++j) {
      float x0 = text->width * (j + 0.5) / TEXT_LENGTH;
      float y0 = text->height * 0.75 - text->width * 0.5 / TEXT_LENGTH;
      float dx = text->width * 0.5 / TEXT_LENGTH;
      float dy = text->width * 0.5 / TEXT_LENGTH;
      int i = text->textstr[j];
      if ('a' <= i && i <= 'z') {
        i -= 'a';
      } else if ('A' <= i && i <= 'Z') {
        i -= 'A';
      } else {
        i = 255;
      }
      glTexCoord4f( (i % text->fonts.pack)      / (float) text->fonts.pack, ((i ↙
          ↳ / text->fonts.pack) + 1) / (float) text->fonts.pack, 0, 1);
      glVertex2f(x0 - dx, y0 - dy);
      glTexCoord4f(((i % text->fonts.pack) + 1) / (float) text->fonts.pack, ((i ↙
          ↳ / text->fonts.pack) + 1) / (float) text->fonts.pack, 1, 1);
      glVertex2f(x0 + dx, y0 - dy);
      glTexCoord4f(((i % text->fonts.pack) + 1) / (float) text->fonts.pack,  (i ↙
          ↳ / text->fonts.pack)      / (float) text->fonts.pack, 1, 0);
      glVertex2f(x0 + dx, y0 + dy);
      glTexCoord4f( (i % text->fonts.pack)      / (float) text->fonts.pack,  (i ↙
          ↳ / text->fonts.pack)      / (float) text->fonts.pack, 0, 0);
      glVertex2f(x0 - dx, y0 + dy);
    }
  } glEnd();
  glUseProgramObjectARB(0);
  glBindTexture(GL_TEXTURE_2D, 0);
  glDisable(GL_TEXTURE_2D);
}

int text_keynormal(struct text *text, int key, int x, int y) {
  if (('A' <= key && key <= 'Z') || ('a' <= key && key <= 'z') || (key == ' ')) ↙
      ↳ {
    for (int i = 1; i < TEXT_LENGTH; ++i) {
      text->textstr[i-1] = text->textstr[i];
    }
    text->textstr[TEXT_LENGTH-1] = key;
    return 1;
  } else if (('\r' == key) || ('\n' == key)) {
    for (int i = 0; i < TEXT_LENGTH; ++i) {
      text->textstr[i] = ' ';
    }
    return 1;
```

```
  } else if ('\b' == key) {
    for (int i = TEXT_LENGTH - 1; i >= 1; --i) {
      text->textstr[i] = text->textstr[i-1];
    }
    text->textstr[0] = ' ';
    return 1;
  } else {
    return 0;
  }
}

int text_keyspecial(struct text *text, int key, int x, int y) {
  return 0;
}
```

104 src/text.frag

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Shader for text
================================================================ */

uniform sampler2D texture;

void main(void) {
  vec2 p = gl_TexCoord[0].xy;
  vec4 c = vec4(0.0, 1.0, 0.0, 1.0) * texture2D(texture, p).rgba;
  if (c.a < 0.1) {
    discard;
  } else {
    gl_FragColor = c;
  }
}
```

105 src/text.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Text module
================================================================ */

#ifndef TEXT_H
#define TEXT_H 1

#include "shader.h"
#include "fonts.h"

#define TEXT_LENGTH 32

struct text { struct shader shader;
  struct {
    GLint texture; // fragment
```

```
  } uniform;
  struct {
    int texture; // fragment
  } value;
  struct fonts fonts;
  char textstr[TEXT_LENGTH]; // characters to display
  int width;
  int height;
};

struct text *text_init(struct text *text);
void text_reshape(struct text *text, int w, int h);
void text_display(struct text *text);
int text_keynormal(struct text *text, int key, int x, int y);
int text_keyspecial(struct text *text, int key, int x, int y);

#endif
```

106 src/text.vert

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Text vertex shader
================================================================ */

void main(void) {
  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

107 src/timeline.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Timeline module
================================================================ */

#include "timeline.h"

struct timeline *timeline_init(struct timeline *timeline) {
  timeline->state1 = timeline_idle;
  timeline->state = timeline_idle;
  timeline->phase = 0;
  return timeline;
}

void timeline_advance(struct timeline *timeline) {
  timeline->state1 = timeline->state;
  timeline->phase += 0.04;
  switch (timeline->state) {
    case timeline_done: break;
    case timeline_idle: break;
```

```
    case timeline_main: break;
    case timeline_intro1:
      if (! (timeline->phase < TIMELINE_INTRO1_LENGTH)) {
        timeline->phase = 0;
        timeline->state = timeline_intro2;
      }
      break;
    case timeline_intro2:
      if (! (timeline->phase < TIMELINE_INTRO2_LENGTH)) {
        timeline->phase = 0;
        timeline->state = timeline_main;
      }
      break;
    case timeline_outro:
      if (! (timeline->phase < TIMELINE_OUTRO_LENGTH)) {
        timeline->phase = 0;
        timeline->state = timeline_done;
      }
      break;
  }
}

void timeline_start(struct timeline *timeline) {
  timeline->phase = 0;
  timeline->state = timeline_intro1;
}

void timeline_stop(struct timeline *timeline) {
  timeline->phase = 0;
  timeline->state = timeline_outro;
}

/* EOF */
```

108 src/timeline.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2010,2011  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Timeline module
================================================================ */

#ifndef TIMELINE_H
#define TIMELINE_H 1

#define TIMELINE_INTRO1_LENGTH 5
#define TIMELINE_INTRO2_LENGTH 25
#define TIMELINE_OUTRO_LENGTH 30

enum timeline_state {
  timeline_idle,
  timeline_intro1,
  timeline_intro2,
  timeline_main,
  timeline_outro,
  timeline_done
```

```
};

struct timeline {
  enum timeline_state state1, state;
  double phase;
};

struct timeline *timeline_init(struct timeline *timeline);
void timeline_advance(struct timeline *timeline);
void timeline_start(struct timeline *timeline);
void timeline_stop(struct timeline *timeline);

#endif
```

109 src/util.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Utility Functions
================================================================ */

#include <assert.h>
#include "util.h"

//================================================================
// round 'x' up to the nearest power of two (which may be 'x' itself)
unsigned int roundtwo(unsigned int x) {
  assert(x <= 1 << 31); // termination condition
  unsigned int y = 1;
  while (y < x) y <<= 1;
  return y;
}

//================================================================
// find log2 of the nearest power of two above 'x'
unsigned int logtwo(unsigned int x) {
  assert(x <= 1 << 31); // termination condition
  unsigned int y = 1, z = 0;
  while (y < x) { y <<= 1; z += 1; };
  return z;
}

// EOF
```

110 src/util.h

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
Utility Functions
================================================================ */

#ifndef UTIL_H
```

```
#define UTIL_H 1

unsigned int roundtwo(unsigned int x);
unsigned int logtwo(unsigned int x);

#endif
```

111 src/worldsphere.c

```
/* ================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------
World sphere shader
================================================================ */

#include "worldsphere.h"
#include "worldsphere.frag.c"

//================================================================
// worldsphere shader initialization
struct worldsphere *worldsphere_init(struct worldsphere *worldsphere) {
  if (! worldsphere) { return 0; }
  if (! shader_init(&worldsphere->shader, 0, worldsphere_frag)) {
    return 0;
  }
  shader_uniform(worldsphere, texture);
  shader_uniform(worldsphere, rot);
  shader_uniform(worldsphere, size);
  worldsphere->value.texture = 0;
  worldsphere->value.rot[0] = 0;
  worldsphere->value.rot[1] = 0;
  return worldsphere;
}

//================================================================
// worldsphere shader activation
void worldsphere_use(struct worldsphere *worldsphere) {
  if (worldsphere) {
    glUseProgramObjectARB(worldsphere->shader.program);
    shader_updatei(worldsphere, texture);
    shader_updatef2(worldsphere, rot);
    shader_updatef2(worldsphere, size);
  } else {
    glUseProgramObjectARB(0);
  }
}

void worldsphere_idle(struct worldsphere *worldsphere) {
  worldsphere->value.rot[0] += worldsphere->delta[0];
  worldsphere->value.rot[1] += worldsphere->delta[1];
  while (worldsphere->value.rot[0] <  0) worldsphere->value.rot[0] += 1;
  while (worldsphere->value.rot[1] <  0) worldsphere->value.rot[1] += 1;
  while (worldsphere->value.rot[0] >= 1) worldsphere->value.rot[0] -= 1;
  while (worldsphere->value.rot[1] >= 1) worldsphere->value.rot[1] -= 1;
}
```

```
// EOF
```

112 src/worldsphere.frag

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
----------------------------------------------------------------------------
World sphere shader
============================================================================ */

uniform sampler2D texture;
uniform vec2 rot;
uniform vec2 size;

void main(void) {
  vec2 p = gl_TexCoord[0].xy;
  vec2 s = sqrt(0.5) * gl_TexCoord[0].zw;
  float d = dot(s,s);
  vec3 n;
  float a;
  float c;
  if (d < 1.0) {
    c = 1.0;
    a = 1.0;
    n = normalize(vec3(s, sqrt(1.0 - d)));
    p = sqrt(2.0) * s / (1.0 + sqrt(1.0 - d));
  } else {
    c = 0.0;
    a = 0.0;
    n = vec3(0.0, 0.0, 0.0);
  }
  c = c * 0.0625 + pow(dot(vec3(0.0, 0.0, 1.0), n), 2.0);
  p += rot;
  p -= floor(p);
  vec2 p0 = floor(p * size) / size;
  vec2 dp = (p - p0) * size;
  vec3 tl = texture2D(texture, p0).rgb;
  vec3 tr = texture2D(texture, p0 + vec2(1.0/size.x, 0.0       )).rgb;
  vec3 bl = texture2D(texture, p0 + vec2(0.0,        1.0/size.y)).rgb;
  vec3 br = texture2D(texture, p0 + vec2(1.0/size.x, 1.0/size.y)).rgb;
  vec3 t = tl * (1.0 - dp.x) + dp.x * tr;
  vec3 b = bl * (1.0 - dp.x) + dp.x * br;
  vec3 l = tl * (1.0 - dp.y) + dp.y * bl;
  vec3 r = tr * (1.0 - dp.y) + dp.y * br;
  vec3 m1 = t * (1.0 - dp.y) + dp.y * b;
  vec3 m2 = l * (1.0 - dp.x) + dp.x * r;
  vec3 m = 0.5 * (m1 + m2);
  gl_FragColor = vec4(m * c, a);
}
```

113 src/worldsphere.h

```
/* ============================================================================
rdex -- reaction-diffusion explorer
Copyright (C) 2008,2010  Claude Heiland-Allen <claude@mathr.co.uk>
```

```
-------------------------------------------------------------------
World sphere shader
=================================================================== */

#ifndef WORLDSPHERE_H
#define WORLDSPHERE_H 1

#include "shader.h"

//=====================================================================
// borderwindow shader data
struct worldsphere { struct shader shader;
  struct { GLint texture; GLint rot;    GLint size;    } uniform;
  struct { int   texture; float rot[2]; float size[2]; } value;
  float delta[2];
};

//=====================================================================
// prototypes
struct worldsphere *worldsphere_init(struct worldsphere *worldsphere);
void worldsphere_use(struct worldsphere *worldsphere);
void worldsphere_idle(struct worldsphere *worldsphere);

#endif
```

114 start.sh

```
#!/bin/bash
DATE="$(date --iso=s)"
(
  killall light-locker
  xset -dpms
  xset s off
  xrandr --output LVDS-0 --same-as HDMI-0 --scale-from 1920x1080
  while true
  do
    RDEX_UPLOAD="http://rdex/sounding-diy-iii-vm-ml/upload/new" RDEX_SESSION="${↙
        ↳ HOME}/opt/var/rdex/session" "${HOME}/opt/bin/rdex"
    sleep 1
  done &
) > "${HOME}/opt/var/rdex/log/${DATE}.log" 2>&1
```

115 stop.sh

```
#!/bin/bash
killall devilspie
killall jack-rack
killall -HUP ecasound
killall -HUP meterbridge
killall -HUP jackd
killall -HUP qjackctl
killall -HUP top
```

116 TODO

```
rdex-client tickets
```

```
URGENT
------

  * bash any possible NaN/inf to 0 in audio
  * intro sequence
  * outro sequence
  * use vertex arrays instead of immediate mode for ball drawing

TODO
----

  * check audio sample rate / video sample rate interdependence
  * game controller input
  * 8-way nearest neighbour navigation for sequence editing
  * set CPU affinity and performance governors if necessary
  * interpolate textures for audio if not everything
  * attempt to use async PBOs for realtime video recording
  * restore non-realtime rendering
  * fix audio at other than JACK/48000Hz
  * merge from CyPi for loading screen/progress bar
  * incremental library/session loading with progress bar
  * sequencer controls for audio radius/spin/travel/speed/etc
  * keyboard controls for reseed stuff
  * audio waveform / spirograph display overlay
  * fix Makefile dependencies to avoid memory corruption and OS crash
  * use multiple texture tile-sheets to boost ball count limit
  * fill hyperspace with fog/dust/ether
  * alpha blend balls with background for smoother appearance
  * video script option handling for 4:3 vs 16:9 aspect ratio
  * full screen anti-aliasing (render at 2x size, downscale smoothly)
  * interestingness boost by seeing if larger r-d grid works fast enough
  * authentication for upload to server
  * maintain README to be in sync with implementation
  * webpage texts

ONGOING
-------

  * make some videos at various stages of development

LATER
-----

  * motion blur
  * make balls nicely glowing
  * ripples when interesting species found
  * catch libcurl std(out|err) output and use it somehow
    http://curl.haxx.se/libcurl/c/curl_easy_setopt.html#CURLOPTWRITEFUNCTION
  * statistics display in-program
  * use GLEW more sensibly/correctly/etc
  * use OpenGL 2.x variants instead of a mishmash of ARB/EXT stuff
  * investigate OpenGL 3.x/4.x API changes and start updating things
  * Debian/Ubuntu/etc packages
  * select point from library for rendering
  * rdex-remote to handle rdex:// urls from browser for above
```

```
  * interactive camera controls (back to rolling ball?)
  * make camera target have an orientation somehow
  * make camera target spin
  * 4D lighting of worldsphere and balls
  * paint into the reactions
  * add more automation of mode switching ("attract mode")
  * investigate USB game controller support
  * distributed rendering!?
  * rdex.rc for options (eg, session dir and initial fullscreenness)
  * add command line arguments to override environment variables
  * runtime setting of NRT render frame rate (both audio and video)
  * runtime setting of NRT capture intervals (both period and interval)
  * runtime setting of NRT quit when rendering complete
  * runtime setting of display aspect ratio and size
  * lock display size when in NRT mode to avoid accidental mess-ups
  * hyperbolic layout by image similarity
  * download library from server, again with threads
  * download library using bittorrent or other p2p

DONE
----

  * worldsphere size variation over time (2010-09-27)
  * disable screensaver and DPMS etc (2010-09-25)
  * send JACK transport commands on start / stop commands (2010-09-25)
  * investigate sync to vblank (2010-09-25)
  * hide mouse cursor (2010-09-25)
  * fix explore/random mode and generate a new pristine session dir (2010-05-17)
  * CPU to generate coordinate texture for audio.frag lookup (2010-05-14)
  * audio coordinates to vary smoothly over time to reduce glitches (2010-05-14)
    ie: spirograph-style wandering instead of discrete circles
  * audio coordinates to be integrated with overdrive to reduce glitches ↙
    ↳ (2010-05-14)
    ie: grab part of waveform from each overdriven frame
  * quality boost by antialiased downscaling to 64x64 or so for the thumbs ↙
    ↳ (2010-05-14)
    ie: upscale -> false colours -> downscale smoothly
  * improve audio tone smoothness (better interpolation/filtering) [ATi BUG] ↙
    ↳ (2010-05-12)
  * check that the linear interpolation implementation is correct [ATi BUG] ↙
    ↳ (2010-05-12)
  * audio dynamic range compression [using jack-rack] (2010-05-12)
  * window size/aspect appearance independence (seems fixed?) (2010-05-12)
  * worldsphere aspect ratio checkup (is it really circular) (2010-05-12)
  * test 4:3 to see if it looks better than 16:9 [handle both] (2010-05-12)
  * video script should clean up files when complete (2010-05-08)
  * video script should generate an ogv from the mpeg (2010-05-08)
  * space scaling for camera physics as well as ball display (2010-05-06)
  * improve audio change smoothness (more often than display frame rate) ↙
    ↳ (2010-05-05)
  * audio sample rate handling logic (sound should be rate independent) ↙
    ↳ (2010-05-05)
  * speed up by using vertex shaders for projection from 4D (billboarding) ↙
    ↳ (2010-05-04)
  * immersive hyperspace experience (flying around to new worlds) (2010-05-04)
  * speed up by using arrays with large texture containing many small images ↙
    ↳ (2010-04-??)
```

```
  * move image analysis stuff into uploader thread (avoid long pause) ↙
    ↳ (2010-??-??)
  * backport style from current rdex-server (2009-10-27)
  * use threads to upload to server in background (2009-10-18)
  * test on other machines (2009-10-17)
  * rdex server (Python/WSGI) with rdex client and browser clients (2009-04-04)
  * make cute icon! (2009-04-01)
  * random vs search vs what? research, decide! (perturb) (2009-04-01)
  * investigate glut "game mode" instead of "fullscreen" (sucks) (2009-04-01)
  * debug segfault on startup (latest fglrx is fine) (2009-04-01)
  * image texture feature extraction (ripped from FIRE) (2008-11-07)
  * health warning (2008-11-03)
  * hide mouse pointer (2008-11-03)
  * fullscreen mode support (2008-11-03)
  * handle perspective frustrum properly (2008-11-03)
  * debug rolling ball implementation (2008-11-03)
  * rationalize Makefile (2008-11-03)
  * depth sorting of points for library display (2008-11-02)
  * save library to session folder while running (2008-11-02)
  * load library from session folder at startup (2008-11-02)
  * split the behemoth (2008-11-01)

WONT
----

  * interpolate between adjacent frames for even smoother audio
  * audio silence detection -> position randomization
  * space scaling by local ball density (statistics of nearest N balls)
```

117 video.sh

```
#!/bin/bash
MKV="rdex-client-$(date --iso=s | tr ':' '-').mkv"
RDEX_RENDER="1" RDEX_SESSION="./session/" ./src/rdex 2>"${MKV}.log" > "${HOME}/↙
    ↳ out/rec.ppm"
ecasound -f:f32,1,48000 -a 1 -i:audio.l.raw -chmove 1,1 -ea 200 -a 2 -i:audio.r.↙
    ↳ raw -chmove 1,2 -ea 200 -a 1,2 -f:s16_le,2,48000 -o audio.wav &&
ffmpeg -r 25 -f image2pipe -i "${HOME}/out/rec.ppm" -b 5000000 -i audio.wav -ab ↙
    ↳ 192000 "${MKV}"
#ppmtoy4m -S 444 -F 25:1 |
#y4mscaler -I sar=1/1 -O preset=dvd -O yscale=1/1 |
#mpeg2enc -f 8 -q 3 -b 8000 -B 768 -D 9 -g 9 -G 15 -P -R 2 -o video.m2v &&
#ecasound -f:f32,1,48000 -a 1 -i:audio.l.raw -chmove 1,1 -ea 200 -a 2 -i:audio.r↙
    ↳ .raw -chmove 1,2 -ea 200 -a 1,2 -f:s16_le,2,48000 -o audio.wav &&
#twolame -b 224 audio.wav &&
#mplex -f 8 -V -o "${MPEG}" video.m2v audio.mp2 &&
#rm -f video.m2v audio.mp2 audio.wav audio.l.raw audio.r.raw &&
#ffmpeg2theora -p preview "${MKV}"
```

118 xorg.conf

```
# nvidia-settings: X configuration file generated by nvidia-settings
# nvidia-settings:  version 1.0  (buildd@biber)  Tue May 18 10:36:08 UTC 2010

Section "ServerLayout"
    Identifier     "Layout0"
    Screen      0  "Screen0" 0 0
```

```
    Screen      1  "Screen1" RightOf "Screen0"
    InputDevice    "Keyboard0" "CoreKeyboard"
    InputDevice    "Mouse0" "CorePointer"
    Option         "Xinerama" "0"
EndSection

Section "Files"
EndSection

Section "InputDevice"
    # generated from default
    Identifier     "Mouse0"
    Driver         "mouse"
    Option         "Protocol" "auto"
    Option         "Device" "/dev/psaux"
    Option         "Emulate3Buttons" "no"
    Option         "ZAxisMapping" "4 5"
EndSection

Section "InputDevice"
    # generated from default
    Identifier     "Keyboard0"
    Driver         "kbd"
EndSection

Section "Monitor"
    # HorizSync source: edid, VertRefresh source: edid
    Identifier     "Monitor0"
    VendorName     "Unknown"
    ModelName      "Chi Mei Optoelectronics corp."
    HorizSync       30.0 - 75.0
    VertRefresh     60.0
    Option         "DPMS"
EndSection

Section "Monitor"
    # HorizSync source: edid, VertRefresh source: edid
    Identifier     "Monitor1"
    VendorName     "Unknown"
    ModelName      "Idek Iiyama HM903DB/DTB"
    HorizSync       30.0 - 132.0
    VertRefresh     50.0 - 200.0
    Option         "DPMS"
EndSection

Section "Device"
    Identifier     "Device0"
    Driver         "nvidia"
    VendorName     "NVIDIA Corporation"
    BoardName      "GeForce G 105M"
    BusID          "PCI:1:0:0"
    Screen          0
EndSection

Section "Device"
    Identifier     "Device1"
    Driver         "nvidia"
```

```
    VendorName     "NVIDIA Corporation"
    BoardName      "GeForce G 105M"
    BusID          "PCI:1:0:0"
    Screen          1
EndSection

Section "Screen"
    Identifier     "Screen0"
    Device         "Device0"
    Monitor        "Monitor0"
    DefaultDepth    24
    Option         "TwinView" "0"
    Option         "metamodes" "DFP: nvidia-auto-select +0+0"
    SubSection     "Display"
        Depth       24
    EndSubSection
EndSection

Section "Screen"
    Identifier     "Screen1"
    Device         "Device1"
    Monitor        "Monitor1"
    DefaultDepth    24
    Option         "TwinView" "0"
    Option         "TwinViewXineramaInfoOrder" "CRT-0"
    Option         "metamodes" "CRT: 1024x768_85 +0+0; CRT: 1024x768_75 +0+0; ↙
        ↳ CRT: 1024x768 +0+0"
    SubSection     "Display"
        Depth       24
    EndSubSection
EndSection
```