zoom

Claude Heiland-Allen

2019

Contents

1 COPYING.md

```
### GNU AFFERO GENERAL PUBLIC LICENSE

    Version 3, 19 November 2007

 5  Copyright (C) 2007 Free Software Foundation, Inc.
    <https://fsf.org/>

    Everyone is permitted to copy and distribute verbatim copies of this
    license document, but changing it is not allowed.
10
    ### Preamble

    The GNU Affero General Public License is a free, copyleft license for
    software and other kinds of works, specifically designed to ensure
15  cooperation with the community in the case of network server software.

    The licenses for most software and other practical works are designed
    to take away your freedom to share and change the works. By contrast,
    our General Public Licenses are intended to guarantee your freedom to
20  share and change all versions of a program--to make sure it remains
    free software for all its users.

    When we speak of free software, we are referring to freedom, not
    price. Our General Public Licenses are designed to make sure that you
25  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.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

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 Affero 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. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the

Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

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 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 work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero 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 Affero 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 Affero 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 Affero 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 Affero 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 Affero General Public License for more details.
>
> You should have received a copy of the GNU Affero General Public License
> along with this program. If not, see <https://www.gnu.org/licenses/>.

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

If your software can interact with users remotely through a computer
network, you should also make sure that it provides a way for users to
get its source. For example, if your program is a web application, its
interface could display a "Source" link that leads users to an archive

```
of the code. There are many ways you could offer source, and different
solutions will be better for different programs; see section 13 for
the specific requirements.

You should also get your employer (if you work as a programmer) or
school, 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 AGPL, see <https://www.gnu.org/licenses/>.
```

2 .gitignore

```
zoom-interpolate
zoom-interpolate-gl4
zoom_interpolate_gl4_frag.glsl.c
zoom_interpolate_gl4_vert.glsl.c
*.exe
```

3 Makefile

```
WINPREFIX := $(HOME)/win64
WFLAGS := -Wall -Wextra -pedantic -Wno-deprecated -O3 -I$(WINPREFIX)/include -I$(WINPREFIX)/include/OpenEXR -D_FILE_OFFSET_BITS=64
WCOMPILE_FLAGS := -std=c++14 $(WFLAGS)
WLINK_FLAGS := -static-libgcc -static-libstdc++ -Wl,--stack,67108864 -Wl,-subsystem,windows -L$(WINPREFIX)/lib

all: zoom-interpolate zoom-interpolate-gl4

zoom-interpolate: zoom-interpolate.cc
	g++ -std=c++14 -Wall -Wextra -pedantic -Wno-deprecated -O3 -fopenmp -o zoom-interpolate zoom-interpolate.cc `pkg-config --cflags --libs OpenEXR`

zoom-interpolate-gl4: zoom-interpolate-gl4.cc zoom_interpolate_gl4_vert.glsl.c zoom_interpolate_gl4_frag.glsl.c
	g++ -std=c++14 -Wall -Wextra -pedantic -Wno-deprecated -O3 -fopenmp -o zoom-interpolate-gl4 zoom-interpolate-gl4.cc `pkg-config --cflags --libs glew glfw3 OpenEXR`

zoom-interpolate.exe: zoom-interpolate.cc
	x86_64-w64-mingw32-g++ $(WCOMPILE_FLAGS) $(WLINK_FLAGS) -o zoom-interpolate.exe zoom-interpolate.cc -lIlmImf-2_4 -lImath-2_4 -lHalf-2_4 -lIex-2_4 -lIexMath-2_4 -lIlmThread-2_4 -lz

%.glsl.c: %.glsl s2c.sh
	bash s2c.sh $* < $< > $@

.SUFFIXES:
.PHONY: all clean
.SECONDARY: zoom_interpolate_gl4_vert.glsl.c zoom_interpolate_gl4_frag.glsl.c
```

4 README.md

```
# zoom

Tools for manipulating zoom animations.
```

```
## About

Zoom videos are a genre of 2D fractal animation. The rendering of the
final video frames can be accelerated by computing only keyframes at
regularly spaced zoom levels and interpolating between them. Rendering
the source keyframes is out of the scope of this project, use dedicated
software for that, but once you have them you can use the tools here to
assemble the final zoom video.

## Dependencies

This project currently works with EXR files, using the OpenEXR library.

## Build

    make

## Usage

Suppose you have files like this, ordered from most zoomed in to most
zoomed out as output by Kalles Fraktaler 2 +, at zoom levels 2x apart:

    keyframe-0000.exr
    keyframe-0001.exr
    keyframe-0002.exr
    ...
    keyframe-0120.exr

These 121 keyframes can be interpolated into a 3 minute (180 seconds)
zoom in video at 60 frames per second (10800 frames total) like this:

    zoom-interpolate --kf --reverse keyframe- 10800 input output

`input` is a user-supplied program that will be invoked for each input
keyframe sequentially (in the order that they are needed), with the
keyframe number (starting from 0, not necessarily the same order as the
filenames) as the first argument, the name of the source keyframe file
as the second argument, and the name of a temporary EXR file that should
be written with the processed keyframe as the third argument. This
scheme is used to allow operations like colouring raw iteration data to
be done per keyframe. `zoom-interpolate` gives up if the `input`
program fails (non-zero exit code). `zoom-input-copy` can be used as
the program if no special processing is needed.

`output` is a user-supplied program that will be invoked for each output
frame sequentially, with the frame number as its first argument and the
name of a temporary EXR file containing the interpolated frame as its
second argument. This scheme is used to avoid needing huge amounts of
temporary disk space for the whole uncompressed video, but also allows
operations like colouring raw iteration data per output frame, needed
for cases like colour cycling. `zoom-interpolate` never writes to
standard output, so (for example) PPM or Y4M data can be output there by
the `output` program and piped to `ffmpeg` or other video encoder.
`zoom-interpolate` gives up if the `output` program fails (non-zero exit
code). `zoom-output-ppm-8` can be used as the program, to convert the
R G B channels from Rec.2020 linear to 8-bit sRGB PPM data on standard
```

```
output.  Similarly `zoom-output-ppm-16` for 16-bit Rec.2020.

The `--kf` argument tells the interpolator to assume the EXR channel
semantics of Kalles Fraktaler 2 +, so that appropriate interpolation
algorithms can be used.

The `--reverse` argument tells the interpolator to start at high-number
files and progress towards zero.

The other arguments are the input keyframe file stem, and total target
frame count.

For a full examples, keep reading.

## Input Filters

Input filters are invoked with 3 arguments:

    filter keyframeindex inputkeyframe.exr outputkeyframe.exr

Input filters are responsible for writing to the `outputkeyframe.exr`
argument.

### `zoom-input-copy`

Copies the input keyframe to the output keyframe.

## Output Filters

Output filters are invoked with 2 arguments:

    filter videoframeindex inputvideoframe.exr

They should read from the `inputvideoframe.exr` argument.

### `zoom-output-copy`

Copies the `inputvideoframe.exr` to `output/########.exr` filled with the
frame number (padded with 0 and starting from 0).  The directory
`output` must exist in the current working directory.

### `zoom-output-ppm-8`

Converts the frame's RGB channels to 8-bit PPM and emits on standard
output.

### `zoom-output-ppm-16`

Converts the frame's RGB channels to 16-bit PPM and emits on standard
output.

### `zoom-output-kfp-8`

Invokes Kalles Fraktaler 2 + (`kf.exe` in the PATH) with the files
`settings.kfs` and `palette.kfp`, as well as the input EXR as a map
file, to colourize the raw iteration data.  The files must exist in the
current working directory.  Output is 8-bit PPM emitted on standard
```

```
output.

## Other useful software

### `ffmpeg`

A video encoder.  Example for web-browser compatible h264 encoding:

    zoom-interpolate --kf --reverse keyframe- 10800 \
      zoom-input-copy zoom-output-ppm-8 |
    ffmpeg -framerate 60 -i - -i soundtrack.wav -s 1920x1080 \
      -pix_fmt yuv420p -profile:v high -level:v 4.1 -crf:v 20 \
      -b:a 512k -movflags +faststart \
      output.mp4

### `pnmsplit`

Splits a PPM stream into individual files.  Example to preview 10 frames
from the output, to check that the KF palette is good enough before
rendering the full thing:

    zoom-interpolate --kf --reverse keyframe- 10 \
      zoom-input-copy zoom-output-kfp-8 |
    pnmsplit - output-%d.ppm
```

5 s2c.sh

```
#!/bin/bash
## zoom-tools -- zoom video tools
## Copyright (C) 2019 Claude Heiland-Allen
## License: GNU AGPLv3+
echo "const char *$1 ="
# strip multiline /* .. */ comments, followed by // .. EOL comments, very hacky
tr '\n' '@' |
sed 's|/\*[^*]*\*/|\n|g' |
sed 's|//[^@]*|\n|g' |
tr '@' '\n' |
sed 's/^ *//g' |
tr -s '\n ' |
sed 's/ = /=/g' |
sed 's|\\|\\\\|g' |
sed 's|"|\\"|g' |
sed 's|^\(\#.*\)$|\\n\1\\n|' |
sed 's|^|"|' |
sed 's|$|"|'
echo '"\n"'
echo ";"
```

6 TODO.md

```
# zoom-interpolate

- read phase stream from stdin (to allow varispeed)
- read additional values from stdin (pass through to filter commands)

# zoom-interpolate-gl4
```

```
- read phase stream from stdin (to allow varispeed)
- read additional values from stdin (named shader uniforms, float only)
- yuv420p colour space conversion with chroma subsampling
- multi-stage pipeline (using mapped PBO) for host<->device transfers
- motion blur by sampling within subframe range
  - problems when range crosses keyframe boundaries?
  - keep nearest 3 keyframes in memory?
    - one more deeper, for end of subframe range
  - need to take care with boundaries and interpolation
  - uniforms to specify phase of keyframes, and subframe phase range
- sampler2D array?  indexing?
- pack all data in one vec4 texture instead of multiple float textures?
- adaptive supersampling based on RGB statistics
  - minimum sample count
  - maximum sample count
  - stopping condition (how many % of Y/UV is expected to change?)
- support >32bit iteration counts
- support OpenGL < 4
  - as far back as OpenGL 2.1?
- make fragment shader source code file a command line argument

# zoom-control

- command to generate single speed phase stream from:
  - number of input keyframes
  - number of output video frames
```

7 zoom-input-copy

```
#!/bin/sh
cp "${2}" "${3}"
```

8 zoom-interpolate.cc

```
#define _DEFAULT_SOURCE

#include <ImfNamespace.h>
#include <ImfInputFile.h>
#include <ImfOutputFile.h>
#include <ImfHeader.h>
#include <ImfChannelList.h>
#include <ImfChannelListAttribute.h>
#include <ImfFrameBuffer.h>
#include <ImathBox.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#include <algorithm>
#include <csignal>
#include <cstdlib>
#include <cstring>
#include <iomanip>
#include <iostream>
#include <vector>
```

```
namespace IMF = OPENEXR_IMF_NAMESPACE;
using namespace IMF;
using namespace IMATH_NAMESPACE;

struct Frame
{
  std::set<std::string> uint_channels, half_channels, float_channels;
  ssize_t width;
  ssize_t height;
  uint32_t *uint_data;
  half *half_data;
  float *float_data;
  ~Frame()
  {
    delete[] uint_data;
    delete[] half_data;
    delete[] float_data;
  };
};

bool are_compatible(const Frame *a, const Frame *b)
{
  return
    a && b &&
    a->uint_channels == b->uint_channels &&
    a->half_channels == b->half_channels &&
    a->float_channels == b->float_channels &&
    a->width == b->width &&
    a->height == b->height;
}

Frame *read_frame(Frame *old_buffer, const std::string &ifilename, const std::string &argv0)
{
  Frame *new_buffer = nullptr;
  if (! old_buffer)
  {
    new_buffer = new Frame();
  }
  else
  {
    new_buffer = old_buffer;
  }
  // read image header
  InputFile ifile(ifilename.c_str());
  const Header &h = ifile.header();
  Box2i dw = h.dataWindow();
  ssize_t fwidth = dw.max.x - dw.min.x + 1;
  ssize_t fheight = dw.max.y - dw.min.y + 1;
  std::set<std::string> fuint_channels, fhalf_channels, ffloat_channels;
  for (Header::ConstIterator i = h.begin(); i != h.end(); ++i)
  {
    const Attribute *a = &i.attribute();
    const ChannelListAttribute *ta = dynamic_cast<const ChannelListAttribute *>(a);
    if (ta)
    {
```

```
        const ChannelList &cl = ta->value();
80      for (ChannelList::ConstIterator j = cl.begin(); j != cl.end(); ++j)
        {
          if (j.channel().xSampling != 1)
          {
            std::cerr << argv0 << ": error: xSampling != 1\n";
85          delete new_buffer;
            return nullptr;
          }
          if (j.channel().ySampling != 1)
          {
90          std::cerr << argv0 << ": error: ySampling != 1\n";
            delete new_buffer;
            return nullptr;
          }
          switch (j.channel().type)
95        {
            case UINT:
              fuint_channels.insert(j.name());
              break;
            case HALF:
100           fhalf_channels.insert(j.name());
              break;
            case FLOAT:
              ffloat_channels.insert(j.name());
              break;
105         default:
              std::cerr << argv0 << ": error: unknown channel type " << j.channel↙
                  ↳ ().type << "\n";
              delete new_buffer;
              return nullptr;
          }
110     }
        // allocate data
        if (! old_buffer)
        {
          new_buffer->width = fwidth;
115       new_buffer->height = fheight;
          new_buffer->uint_channels = fuint_channels;
          new_buffer->half_channels = fhalf_channels;
          new_buffer->float_channels = ffloat_channels;
          if (new_buffer->uint_channels.size() > 0)
120       {
            new_buffer->uint_data = new uint32_t[new_buffer->uint_channels.size() ↙
                ↳ * new_buffer->width * new_buffer->height];
          }
          if (new_buffer->half_channels.size() > 0)
          {
125         new_buffer->half_data = new half[new_buffer->half_channels.size() * ↙
                ↳ new_buffer->width * new_buffer->height];
          }
          if (new_buffer->float_channels.size() > 0)
          {
            new_buffer->float_data = new float[new_buffer->float_channels.size() *↙
                ↳  new_buffer->width * new_buffer->height];
130       }
        }
```

```
      else
      {
        if (old_buffer->width != fwidth ||
            old_buffer->height != fheight ||
            old_buffer->uint_channels != fuint_channels ||
            old_buffer->half_channels != fhalf_channels ||
            old_buffer->float_channels != ffloat_channels
           )
        {
          std::cerr << argv0 << " : error: incompatible files\n";
          delete new_buffer;
          return nullptr;
        }
      }
    }
    // read image data
    FrameBuffer ifb;
    int k = 0;
    for (auto name : new_buffer->uint_channels)
    {
      ifb.insert
        ( name.c_str()
        , Slice
          ( IMF::UINT
          , (char *) (&new_buffer->uint_data[0] + k - dw.min.x - dw.min.y * ↙
              ↳ new_buffer->width)
          , sizeof(new_buffer->uint_data[0]) * new_buffer->uint_channels.size()
          , sizeof(new_buffer->uint_data[0]) * new_buffer->width * new_buffer->↙
              ↳ uint_channels.size()
          , 1, 1
          , 0
          )
       );
      ++k;
    }
    k = 0;
    for (auto name : new_buffer->half_channels)
    {
      ifb.insert
        ( name.c_str()
        , Slice
          ( IMF::HALF
          , (char *) (&new_buffer->half_data[0] + k - dw.min.x - dw.min.y * ↙
              ↳ new_buffer->width)
          , sizeof(new_buffer->half_data[0]) * new_buffer->half_channels.size()
          , sizeof(new_buffer->half_data[0]) * new_buffer->width * new_buffer->↙
              ↳ half_channels.size()
          , 1, 1
          , 0
          )
        );
      ++k;
    }
    k = 0;
    for (auto name : new_buffer->float_channels)
    {
```

```
185       ifb.insert
            ( name.c_str()
            , Slice
              ( IMF::FLOAT
              , (char *) (&new_buffer->float_data[0] + k - dw.min.x - dw.min.y * ↙
                  ↳ new_buffer->width)
190           , sizeof(new_buffer->float_data[0]) * new_buffer->float_channels.size()
              , sizeof(new_buffer->float_data[0]) * new_buffer->width * new_buffer->↙
                  ↳ float_channels.size()
              , 1, 1
              , 0
              )
195         );
          ++k;
        }
        ifile.setFrameBuffer(ifb);
        ifile.readPixels(dw.min.y, dw.max.y);
200     return new_buffer;
      }

      bool write_frame(const Frame *buffer, const std::string &ofilename)
      {
205     Header oh(buffer->width, buffer->height, 1, V2f(0, 0), 1, INCREASING_Y, ↙
            ↳ NO_COMPRESSION);
        for (auto name : buffer->uint_channels)
        {
          oh.channels().insert(name.c_str(), Channel(IMF::UINT));
        }
210     for (auto name : buffer->half_channels)
        {
          oh.channels().insert(name.c_str(), Channel(IMF::HALF));
        }
        for (auto name : buffer->float_channels)
215     {
          oh.channels().insert(name.c_str(), Channel(IMF::FLOAT));
        }
        OutputFile ofile(ofilename.c_str(), oh);
        FrameBuffer ofb;
220     int k = 0;
        for (auto name : buffer->uint_channels)
        {
          ofb.insert(name, Slice(IMF::UINT, (char *) (&buffer->uint_data[0] + k), ↙
              ↳ sizeof(buffer->uint_data[0]) * buffer->uint_channels.size(), sizeof(↙
              ↳ buffer->uint_data[0]) * buffer->width * buffer->uint_channels.size()))↙
              ↳ ;
          ++k;
225     }
        k = 0;
        for (auto name : buffer->half_channels)
        {
          ofb.insert(name, Slice(IMF::HALF, (char *) (&buffer->half_data[0] + k), ↙
              ↳ sizeof(buffer->half_data[0]) * buffer->half_channels.size(), sizeof(↙
              ↳ buffer->half_data[0]) * buffer->width * buffer->half_channels.size()))↙
              ↳ ;
230       ++k;
        }
        k = 0;
```

```
    for (auto name : buffer->float_channels)
    {
      ofb.insert(name, Slice(IMF::FLOAT, (char *) (&buffer->float_data[0] + k), sizeof(buffer->float_data[0]) * buffer->float_channels.size(), sizeof(buffer->float_data[0]) * buffer->width * buffer->float_channels.size()));
      ++k;
    }
    ofile.setFrameBuffer(ofb);
    ofile.writePixels(buffer->height);
    return true;
  }

  static inline double lookup1_uint(const Frame *input, int k, int i, int j)
  {
    if (i < 0) i = 0;
    if (i >= input->width) i = input->width - 1;
    if (j < 0) j = 0;
    if (j >= input->height) j = input->height - 1;
    return input->uint_data[(j * input->width + i) * input->uint_channels.size() + k];
  }

  static inline double lookup1_half(const Frame *input, int k, int i, int j)
  {
    if (i < 0) i = 0;
    if (i >= input->width) i = input->width - 1;
    if (j < 0) j = 0;
    if (j >= input->height) j = input->height - 1;
    double h = input->half_data[(j * input->width + i) * input->half_channels.size() + k];
  #if 0
    double minimum_positive_normal_half_value = pow(2, -14);
    double maximum_representable_half_value = 65504;
    if (! (h > minimum_positive_normal_half_value)) h = minimum_positive_normal_half_value;
    if (! (h < maximum_representable_half_value)) h = maximum_representable_half_value;
  #endif
    return h;
  }

  static inline double lookup1_float(const Frame *input, int k, int i, int j)
  {
    if (i < 0) i = 0;
    if (i >= input->width) i = input->width - 1;
    if (j < 0) j = 0;
    if (j >= input->height) j = input->height - 1;
    return input->float_data[(j * input->width + i) * input->float_channels.size() + k];
  }

  static inline double lookup1_N_NF(const Frame *input, int k_n, int k_nf, int i, int j)
  {
    if (i < 0) i = 0;
    if (i >= input->width) i = input->width - 1;
```

```
  if (j < 0) j = 0;
  if (j >= input->height) j = input->height - 1;
  return
    input->uint_data[(j * input->width + i) * input->uint_channels.size() + k_n] +
    input->float_data[(j * input->width + i) * input->float_channels.size() + k_nf];
}

static inline double lookup_uint(const Frame *input, int k, int x, int y, double factor)
{
  ssize_t w = input->width;
  ssize_t h = input->height;
  double i = factor * (x + 0.5 - 0.5 * w) + 0.5 * w - 0.5;
  double j = factor * (y + 0.5 - 0.5 * h) + 0.5 * h - 0.5;
  ssize_t i0 = floor(i);
  ssize_t j0 = floor(j);
  ssize_t i1 = i0 + 1;
  ssize_t j1 = j0 + 1;
  double ix = i - i0;
  double jy = j - j0;
  double f00 = lookup1_uint(input, k, i0, j0);
  double f10 = lookup1_uint(input, k, i1, j0);
  double f01 = lookup1_uint(input, k, i0, j1);
  double f11 = lookup1_uint(input, k, i1, j1);
  double f0 = f00 + ix * (f10 - f00);
  double f1 = f01 + ix * (f11 - f01);
  double f = f0 + jy * (f1 - f0);
  return f;
}

static inline double lookup_half(const Frame *input, int k, int x, int y, double factor)
{
  ssize_t w = input->width;
  ssize_t h = input->height;
  double i = factor * (x + 0.5 - 0.5 * w) + 0.5 * w - 0.5;
  double j = factor * (y + 0.5 - 0.5 * h) + 0.5 * h - 0.5;
  ssize_t i0 = floor(i);
  ssize_t j0 = floor(j);
  ssize_t i1 = i0 + 1;
  ssize_t j1 = j0 + 1;
  double ix = i - i0;
  double jy = j - j0;
  double f00 = lookup1_half(input, k, i0, j0);
  double f10 = lookup1_half(input, k, i1, j0);
  double f01 = lookup1_half(input, k, i0, j1);
  double f11 = lookup1_half(input, k, i1, j1);
  double f0 = f00 + ix * (f10 - f00);
  double f1 = f01 + ix * (f11 - f01);
  double f = f0 + jy * (f1 - f0);
  return f;
}

static inline double lookup_float(const Frame *input, int k, int x, int y, double factor)
```

```
{
  ssize_t w = input->width;
  ssize_t h = input->height;
  double i = factor * (x + 0.5 - 0.5 * w) + 0.5 * w - 0.5;
  double j = factor * (y + 0.5 - 0.5 * h) + 0.5 * h - 0.5;
  ssize_t i0 = floor(i);
  ssize_t j0 = floor(j);
  ssize_t i1 = i0 + 1;
  ssize_t j1 = j0 + 1;
  double ix = i - i0;
  double jy = j - j0;
  double f00 = lookup1_float(input, k, i0, j0);
  double f10 = lookup1_float(input, k, i1, j0);
  double f01 = lookup1_float(input, k, i0, j1);
  double f11 = lookup1_float(input, k, i1, j1);
  double f0 = f00 + ix * (f10 - f00);
  double f1 = f01 + ix * (f11 - f01);
  double f = f0 + jy * (f1 - f0);
  return f;
}

static inline double lookup_N_NF(const Frame *input, int k_n, int k_nf, int x, ↙
    ↳ int y, double factor)
{
  ssize_t w = input->width;
  ssize_t h = input->height;
  double i = factor * (x + 0.5 - 0.5 * w) + 0.5 * w - 0.5;
  double j = factor * (y + 0.5 - 0.5 * h) + 0.5 * h - 0.5;
  ssize_t i0 = floor(i);
  ssize_t j0 = floor(j);
  ssize_t i1 = i0 + 1;
  ssize_t j1 = j0 + 1;
  double ix = i - i0;
  double jy = j - j0;
  double f00 = lookup1_N_NF(input, k_n, k_nf, i0, j0);
  double f10 = lookup1_N_NF(input, k_n, k_nf, i1, j0);
  double f01 = lookup1_N_NF(input, k_n, k_nf, i0, j1);
  double f11 = lookup1_N_NF(input, k_n, k_nf, i1, j1);
  double f0 = f00 + ix * (f10 - f00);
  double f1 = f01 + ix * (f11 - f01);
  double f = f0 + jy * (f1 - f0);
  return f;
}

bool run_input_filter
  ( Frame * &next_keyframe
  , int keyframe_index
  , int keyframe_count
  , const std::string &keyframe_stem
  , bool reverse_keyframes
  , const std::string &argv0
  , const std::string &input_filter
  , const std::string &temporary_file
  )
{
  std::ostringstream keyframe_file;
  keyframe_file << keyframe_stem << std::setfill('0') << std::setw(4) <<
```

```
    ( reverse_keyframes
390 ? keyframe_count - 1 - keyframe_index
    : keyframe_index
    ) << ".exr";
  std::ostringstream input_command;
  input_command
395 << input_filter << " "
  << keyframe_index << " "
  << keyframe_file.str() << " "
  << temporary_file;
  int ret = system(input_command.str().c_str());
400 if (WIFSIGNALED(ret) && (WTERMSIG(ret) == SIGINT || WTERMSIG(ret) == SIGQUIT))
  {
    std::cerr << argv0 << ": interrupt\n";
    return false;
  }
405 if (! (WIFEXITED(ret) && WEXITSTATUS(ret) == 0))
  {
    std::cerr << argv0 << ": error: input filter failed:\n";
    std::cerr << argv0 << ": error: " << input_command.str() << "\n";
    return false;
410 }
  next_keyframe = read_frame(next_keyframe, temporary_file, argv0);
  if (! next_keyframe)
  {
    std::cerr << argv0 << ": error: reading output of input filter failed\n";
415 return false;
  }
  return true;
}

420 Frame *interpolate_frames(Frame *output, const Frame *prev, const Frame *next, ↙
    ↳ bool kf_channel_semantics, double phase)
{
  if (! (are_compatible(output, prev) && are_compatible(output, next)))
  {
    delete output;
425 output = nullptr;
  }
  if (! output)
  {
    output = new Frame();
430 output->uint_channels = prev->uint_channels;
    output->half_channels = prev->half_channels;
    output->float_channels = prev->float_channels;
    output->width = prev->width;
    output->height = prev->height;
435 if (output->uint_channels.size() > 0)
    {
      output->uint_data = new uint32_t[output->uint_channels.size() * output->↙
          ↳ width * output->height];
    }
    if (output->half_channels.size() > 0)
440 {
      output->half_data = new half[output->half_channels.size() * output->width ↙
          ↳ * output->height];
    }
```

```
    if (output->float_channels.size() > 0)
    {
      output->float_data = new float[output->float_channels.size() * output->width * output->height];
    }
  }
  double phase1 = phase + 1;
  double factor = pow(0.5, phase);
  double factor1 = pow(0.5, phase1);
  double mixer = phase;//(1 - pow(4, -phase)) / (pow(4, 1 - phase) - pow(4, -phase));
  int k = 0;
  for (auto name : output->uint_channels)
  {
    if (kf_channel_semantics && name == "N" && output->float_channels.find("NF") != output->float_channels.end())
    {
      int k_NF = 0;
      for (auto name : output->float_channels)
      {
        if (name == "NF")
        {
          break;
        }
        k_NF++;
      }
      #pragma omp parallel for schedule(static)
      for (int y = 0; y < output->height; ++y)
      {
        for (int x = 0; x < output->width; ++x)
        {
          double N_NF0 = lookup_N_NF(prev, k, k_NF, x, y, factor1);
          double N_NF1 = lookup_N_NF(next, k, k_NF, x, y, factor);
          double N_NF  = N_NF0 + mixer * (N_NF1 - N_NF0);
          uint32_t N = floor(N_NF);
          float NF = N_NF - N;
          output->uint_data[(y * output->width + x) * output->uint_channels.size() + k] = N;
          output->float_data[(y * output->width + x) * output->float_channels.size() + k_NF] = NF;
        }
      }
    }
    else
    {
      #pragma omp parallel for schedule(static)
      for (int y = 0; y < output->height; ++y)
      {
        for (int x = 0; x < output->width; ++x)
        {
          double N0 = lookup_uint(prev, k, x, y, factor1);
          double N1 = lookup_uint(next, k, x, y, factor);
          double N  = N0 + mixer * (N1 - N0);
          output->uint_data[(y * output->width + x) * output->uint_channels.size() + k] = round(N);
        }
      }
```

```
      }
      ++k;
    }
    k = 0;
    for (auto name : output->half_channels)
    {
      ssize_t error_count = 0;
      #pragma omp parallel for schedule(static)
      for (int y = 0; y < output->height; ++y)
      {
        for (int x = 0; x < output->width; ++x)
        {
          double h0 = lookup_half(prev, k, x, y, factor1);
          double h1 = lookup_half(next, k, x, y, factor);
          double h  = h0 + mixer * (h1 - h0);
          output->half_data[(y * output->width + x) * output->half_channels.size()↙
              ↳  + k] = h;
        }
      }
      if (error_count > 0)
      {
        std::cerr << "WARNING: channel " << name << " has " << (error_count * ↙
            ↳ 100.0 / (output->height * output->width)) << "% hot pixels\n";
      }
      ++k;
    }
    k = 0;
    for (auto name : output->float_channels)
    {
      if (kf_channel_semantics && name == "NF" && output->uint_channels.find("N") ↙
          ↳ != output->uint_channels.end())
      {
        // already done above
      }
      else
      {
        #pragma omp parallel for schedule(static)
        for (int y = 0; y < output->height; ++y)
        {
          for (int x = 0; x < output->width; ++x)
          {
            double f0 = lookup_float(prev, k, x, y, factor1);
            double f1 = lookup_float(next, k, x, y, factor);
            double f  = f0 + mixer * (f1 - f0);
            output->float_data[(y * output->width + x) * output->float_channels.↙
                ↳ size() + k] = f;
          }
        }
      }
      ++k;
    }
    return output;
  }

  int main(int argc, char **argv)
  {
    // parse arguments
```

```
  bool kf_channel_semantics = false;
  bool reverse_keyframes = false;
  std::vector<std::string> positional_arguments;
  for (int i = 1; i < argc; ++i)
  {
    std::string argument(argv[i]);
    if (argument == "-?" || argument == "-h" || argument == "--help")
    {
      std::cout <<
        "usage:\n    " << argv[0] << " [options] stem count input output\n"
        "options:\n"
        "    -?, -h, --help       print this message\n"
        "    -v, -V, --version    print version string\n"
        "    --[no-]kf            (dis)enable KF EXR channel semantics\n"
        "    --[no-]reverse       (dis)enable keyframe order reversal\n"
        "arguments:\n"
        "    stem                 keyframes are in files stem####.exr\n"
        "    count                number of output video frames\n"
        "    input                input filter, e.g. zoom-input-copy\n"
        "    output               output filter, e.g. zoom-output-ppm-8\n"
        "output:\n"
        "    output filters may write to standard output, for piping to\n"
        "    a video encoder such as ffmpeg\n"
        ;
      return 0;
    }
    else if (argument == "-v" || argument == "-V" || argument == "--version")
    {
      std::cout << "0.1\n";
      return 0;
    }
    else if (argument == "--kf")
    {
      kf_channel_semantics = true;
    }
    else if (argument == "--no-kf")
    {
      kf_channel_semantics = false;
    }
    else if (argument == "--reverse")
    {
      reverse_keyframes = true;
    }
    else if (argument == "--no-reverse")
    {
      reverse_keyframes = false;
    }
    else
    {
      positional_arguments.push_back(argument);
    }
  }
  if (positional_arguments.size() != 4)
  {
    std::cerr << argv[0] << ": error: expected 4 positional arguments\n";
    return 1;
  }
```

```
  std::string keyframe_stem = positional_arguments[0];
  int output_frame_count = atoi(positional_arguments[1].c_str());
  if (output_frame_count <= 0)
  {
    std::cerr << argv[0] << ": error: bad output frame count <= 0\n";
    return 1;
  }
  std::string input_filter = positional_arguments[2];
  std::string output_filter = positional_arguments[3];
  // count keyframes
  struct stat sb;
  int keyframe_count = 0;
  std::string filename_buf;
  do
  {
    std::ostringstream filename;
    filename << keyframe_stem << std::setfill('0') << std::setw(4) << ↙
        ↳ keyframe_count << ".exr";
    filename_buf = filename.str();
    keyframe_count++;
  } while (! stat(filename_buf.c_str(), &sb));
  keyframe_count--;
  if (keyframe_count <= 1)
  {
    std::cerr << argv[0] << ": error: bad input keyframe count <= 1\n";
    return 1;
  }
  // open temporary file
  char pattern[] = "zoom-interpolate-XXXXXX.exr";
  int temporary_file_fd = mkstemps(pattern, 4);
  if (temporary_file_fd == -1)
  {
    std::cerr << argv[0] << ": error: could not create temporary file\n";
    return 1;
  }
  std::string temporary_file(pattern);
  // compute speed
  /*
     first output frame is at first input keyframe
     last input keyframe is at last-but-one output frame
     keyframes      |       |       |       | (4)
     output frames  |  |  |  |  |  |  |  |  : (8)
     so, speed = (keyframes - 1) / (output frames)
     then offset output frames by incr/2 to avoid rounding issues at eof
  */
  double phase_increment = (keyframe_count - 1) / double(output_frame_count);
  // main loop
  double phase = phase_increment / 2;
  Frame *previous_keyframe = nullptr,
        *next_keyframe = nullptr,
        *output_frame = nullptr;
  int previous_keyframe_index = -1;
  int next_keyframe_index = -1;
  for ( int output_frame_index = 0
      ; output_frame_index < output_frame_count
      ; ++output_frame_index
      )
```

```
660   {
        if (floor(phase) != previous_keyframe_index || ceil(phase) != ↙
            ↳ next_keyframe_index)
        {
          // need to read a frame
          if (floor(phase) == next_keyframe_index || ceil(phase) == ↙
              ↳ previous_keyframe_index)
665       {
            // we advanced to next or previous frame, can reuse the frame
            std::swap(previous_keyframe, next_keyframe);
            std::swap(previous_keyframe_index, next_keyframe_index);
          }
670       if (floor(phase) != previous_keyframe_index)
          {
            previous_keyframe_index = floor(phase);
            if (! run_input_filter
                  ( previous_keyframe
675               , previous_keyframe_index
                  , keyframe_count
                  , keyframe_stem
                  , reverse_keyframes
                  , argv[0]
680               , input_filter
                  , temporary_file
                  ))
            {
              return 1;
685         }
          }
          if (ceil(phase) != next_keyframe_index)
          {
            next_keyframe_index = ceil(phase);
690         if (! run_input_filter
                  ( next_keyframe
                  , next_keyframe_index
                  , keyframe_count
                  , keyframe_stem
695               , reverse_keyframes
                  , argv[0]
                  , input_filter
                  , temporary_file
                  ))
700         {
              return 1;
            }
          }
        }
705     if (are_compatible(previous_keyframe, next_keyframe))
        {
          output_frame = interpolate_frames
            ( output_frame
            , previous_keyframe
710         , next_keyframe
            , kf_channel_semantics
            , phase - floor(phase)
            );
          if (! output_frame)
```

```
715         {
              std::cerr << argv[0] << ": error: interpolating keyframes failed\n";
              return 1;
            }
            if (! write_frame(output_frame, temporary_file))
720         {
              std::cerr << argv[0] << ": error: writing input of output filter failed\↙
                  ↳ n";
              return 1;
            }
            std::ostringstream output_command;
725         output_command
              << output_filter << " "
              << output_frame_index << " "
              << temporary_file;
            int ret = system(output_command.str().c_str());
730         if (WIFSIGNALED(ret) && (WTERMSIG(ret) == SIGINT || WTERMSIG(ret) == ↙
                ↳ SIGQUIT))
            {
              std::cerr << argv[0] << ": interrupt\n";
              return 1;
            }
735         if (! (WIFEXITED(ret) && WEXITSTATUS(ret) == 0))
            {
              std::cerr << argv[0] << ": error: output filter failed:\n";
              std::cerr << argv[0] << ": error: " << output_command.str() << "\n";
              return 1;
740         }
            phase += phase_increment;
          }
          else
          {
745         std::cerr << argv[0] << ": error: incompatible keyframes\n";
            return 1;
          }
        }
        return 0;
750   }
```

9 zoom-interpolate-gl4.cc

```
/*
zoom-tools -- zoom video tools
Copyright (C) 2019 Claude Heiland-Allen

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

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

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

```
*/

#define _DEFAULT_SOURCE

#include <ImfNamespace.h>
#include <ImfInputFile.h>
#include <ImfOutputFile.h>
#include <ImfHeader.h>
#include <ImfChannelList.h>
#include <ImfChannelListAttribute.h>
#include <ImfFrameBuffer.h>
#include <ImathBox.h>

#include <GL/glew.h>
#include <GLFW/glfw3.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#include <algorithm>
#include <csignal>
#include <cstdlib>
#include <cstring>
#include <iomanip>
#include <iostream>
#include <vector>

#include "zoom_interpolate_gl4_vert.glsl.c"
#include "zoom_interpolate_gl4_frag.glsl.c"

namespace IMF = OPENEXR_IMF_NAMESPACE;
using namespace IMF;
using namespace IMATH_NAMESPACE;

struct Frame
{
  std::set<std::string> uint_channels, half_channels, float_channels;
  ssize_t width;
  ssize_t height;
  uint32_t *uint_data;
  half *half_data;
  float *float_data;
  GLuint Nt, NFt, DEXt, DEYt; // texture objects
  GLint Nu, NFu, DEXu, DEYu; // texture units
  ~Frame()
  {
    delete[] uint_data;
    delete[] half_data;
    delete[] float_data;
  };
};

bool are_compatible(const Frame *a, const Frame *b)
{
  return
    a && b &&
```

```
    a->uint_channels == b->uint_channels &&
    a->half_channels == b->half_channels &&
    a->float_channels == b->float_channels &&
    a->width == b->width &&
    a->height == b->height;
}

static int next_texture_unit = 0;

Frame *read_frame(Frame *old_buffer, const std::string &ifilename, const std::string &argv0)
{
  Frame *new_buffer = nullptr;
  if (! old_buffer)
  {
    new_buffer = new Frame();
  }
  else
  {
    new_buffer = old_buffer;
  }
  // read image header
  InputFile ifile(ifilename.c_str());
  const Header &head = ifile.header();
  Box2i dw = head.dataWindow();
  ssize_t fwidth = dw.max.x - dw.min.x + 1;
  ssize_t fheight = dw.max.y - dw.min.y + 1;
  std::set<std::string> fuint_channels, fhalf_channels, ffloat_channels;
  for (Header::ConstIterator i = head.begin(); i != head.end(); ++i)
  {
    const Attribute *a = &i.attribute();
    const ChannelListAttribute *ta = dynamic_cast<const ChannelListAttribute *>(a);
    if (ta)
    {
      const ChannelList &cl = ta->value();
      for (ChannelList::ConstIterator j = cl.begin(); j != cl.end(); ++j)
      {
        if (j.channel().xSampling != 1)
        {
          std::cerr << argv0 << ": error: xSampling != 1\n";
          delete new_buffer;
          return nullptr;
        }
        if (j.channel().ySampling != 1)
        {
          std::cerr << argv0 << ": error: ySampling != 1\n";
          delete new_buffer;
          return nullptr;
        }
        switch (j.channel().type)
        {
          case UINT:
            fuint_channels.insert(j.name());
            break;
          case HALF:
            fhalf_channels.insert(j.name());
```

```
            break;
130         case FLOAT:
            ffloat_channels.insert(j.name());
            break;
          default:
            std::cerr << argv0 << ": error: unknown channel type " << j.channel↙
                ↳ ().type << "\n";
135         delete new_buffer;
            return nullptr;
        }
      }
      // allocate data
140   if (! old_buffer)
      {
        new_buffer->width = fwidth;
        new_buffer->height = fheight;
        new_buffer->uint_channels = fuint_channels;
145     new_buffer->half_channels = fhalf_channels;
        new_buffer->float_channels = ffloat_channels;
        if (new_buffer->uint_channels.size() > 0)
        {
          new_buffer->uint_data = new uint32_t[new_buffer->uint_channels.size() ↙
              ↳ * new_buffer->width * new_buffer->height];
150     }
        if (new_buffer->half_channels.size() > 0)
        {
          new_buffer->half_data = new half[new_buffer->half_channels.size() * ↙
              ↳ new_buffer->width * new_buffer->height];
        }
155     if (new_buffer->float_channels.size() > 0)
        {
          new_buffer->float_data = new float[new_buffer->float_channels.size() *↙
              ↳  new_buffer->width * new_buffer->height];
        }
      }
160   else
      {
        if (old_buffer->width != fwidth ||
            old_buffer->height != fheight ||
            old_buffer->uint_channels != fuint_channels ||
165         old_buffer->half_channels != fhalf_channels ||
            old_buffer->float_channels != ffloat_channels
           )
        {
          std::cerr << argv0 << " : error: incompatible files\n";
170       delete new_buffer;
          return nullptr;
        }
      }
    }
175 }
    // read image data
    FrameBuffer ifb;
    int k = 0;
    for (auto name : new_buffer->uint_channels)
180 {
      ifb.insert
```

```
        ( name.c_str()
        , Slice
          ( IMF::UINT
          , (char *) (&new_buffer->uint_data[0] + k * new_buffer->width * ↙
              ↳ new_buffer->height - dw.min.x - dw.min.y * new_buffer->width)
          , sizeof(new_buffer->uint_data[0])
          , sizeof(new_buffer->uint_data[0]) * new_buffer->width
          , 1, 1
          , 0
          )
      );
    ++k;
  }
  k = 0;
  for (auto name : new_buffer->half_channels)
  {
    ifb.insert
      ( name.c_str()
      , Slice
        ( IMF::HALF
        , (char *) (&new_buffer->half_data[0] + k * new_buffer->width * ↙
            ↳ new_buffer->height - dw.min.x - dw.min.y * new_buffer->width)
        , sizeof(new_buffer->half_data[0])
        , sizeof(new_buffer->half_data[0]) * new_buffer->width
        , 1, 1
        , 0
        )
      );
    ++k;
  }
  k = 0;
  for (auto name : new_buffer->float_channels)
  {
    ifb.insert
      ( name.c_str()
      , Slice
        ( IMF::FLOAT
        , (char *) (&new_buffer->float_data[0] + k * new_buffer->width * ↙
            ↳ new_buffer->height - dw.min.x - dw.min.y * new_buffer->width)
        , sizeof(new_buffer->float_data[0])
        , sizeof(new_buffer->float_data[0]) * new_buffer->width
        , 1, 1
        , 0
        )
      );
    ++k;
  }
  ifile.setFrameBuffer(ifb);
  ifile.readPixels(dw.min.y, dw.max.y);
  // upload to OpenGL
  ssize_t w = new_buffer->width;
  ssize_t h = new_buffer->height;
  if (! old_buffer)
  {
    // allocate textures
    GLuint tex[4];
    glGenTextures(4, &tex[0]);
```

```
      new_buffer->Nt = tex[0];
      new_buffer->NFt = tex[1];
      new_buffer->DEXt = tex[2];
      new_buffer->DEYt = tex[3];
240   new_buffer->Nu = next_texture_unit++;
      new_buffer->NFu = next_texture_unit++;
      new_buffer->DEXu = next_texture_unit++;
      new_buffer->DEYu = next_texture_unit++;
      glActiveTexture(GL_TEXTURE0 + new_buffer->Nu);
245   glBindTexture(GL_TEXTURE_2D, new_buffer->Nt);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_R32UI, w, h, 0, GL_RED_INTEGER, ↙
          ↳ GL_UNSIGNED_INT, 0);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
250   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
      glActiveTexture(GL_TEXTURE0 + new_buffer->NFu);
      glBindTexture(GL_TEXTURE_2D, new_buffer->NFt);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_R32F, w, h, 0, GL_RED, GL_FLOAT, 0);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
255   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
      glActiveTexture(GL_TEXTURE0 + new_buffer->DEXu);
      glBindTexture(GL_TEXTURE_2D, new_buffer->DEXt);
260   glTexImage2D(GL_TEXTURE_2D, 0, GL_R32F, w, h, 0, GL_RED, GL_FLOAT, 0);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
265   glActiveTexture(GL_TEXTURE0 + new_buffer->DEYu);
      glBindTexture(GL_TEXTURE_2D, new_buffer->DEYt);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_R32F, w, h, 0, GL_RED, GL_FLOAT, 0);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
270   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    }
    // upload textures
    k = 0;
275 for (auto name : new_buffer->uint_channels)
    {
      if (name == "N")
      {
        glActiveTexture(GL_TEXTURE0 + new_buffer->Nu);
280     glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, w, h, GL_RED_INTEGER, ↙
            ↳ GL_UNSIGNED_INT, new_buffer->uint_data + k * w * h);
      }
      ++k;
    }
    k = 0;
285 for (auto name : new_buffer->float_channels)
    {
      if (name == "NF")
      {
        glActiveTexture(GL_TEXTURE0 + new_buffer->NFu);
290     glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, w, h, GL_RED, GL_FLOAT, new_buffer↙
```

```
        ↳ ->float_data + k * w * h);
    }
    else if (name == "DEX")
    {
      glActiveTexture(GL_TEXTURE0 + new_buffer->DEXu);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, w, h, GL_RED, GL_FLOAT, new_buffer↙
        ↳ ->float_data + k * w * h);
    }
    else if (name == "DEY")
    {
      glActiveTexture(GL_TEXTURE0 + new_buffer->DEYu);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, w, h, GL_RED, GL_FLOAT, new_buffer↙
        ↳ ->float_data + k * w * h);
    }
    ++k;
  }
  return new_buffer;
}

static void debug_program(GLuint program) {
  GLint status = 0;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
  GLint length = 0;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = (char *)malloc(length + 1);
    info[0] = 0;
    glGetProgramInfoLog(program, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
  }
  if (info) {
    free(info);
  }
}

static void debug_shader(GLuint shader, GLenum type, const char *source) {
  GLint status = 0;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
  GLint length = 0;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = (char *)malloc(length + 1);
    info[0] = 0;
    glGetShaderInfoLog(shader, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    const char *type_str = "unknown";
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
      case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
    }
```

```
345       fprintf
            ( stderr
            , "%s shader compile info:\n%s\nshader source:\n%s"
            , type_str
            , info ? info : "(no info log)"
350         , source ? source : "(no source)"
            );
        }
        if (info) {
          free(info);
355     }
      }

      // shader
      static const GLint u_N_0 = 0;
360   static const GLint u_NF_0 = 1;
      static const GLint u_DEX_0 = 2;
      static const GLint u_DEY_0 = 3;
      static const GLint u_N_1 = 4;
      static const GLint u_NF_1 = 5;
365   static const GLint u_DEX_1 = 6;
      static const GLint u_DEY_1 = 7;
      static const GLint u_phase = 8;
      static const GLint u_time = 9;
      static const GLint u_samples = 10;
370   static const GLint u_palette = 11;

      void interpolate_frames
        ( const Frame *prev
        , const Frame *next
375     , double phase
        , double time
        , int samples_per_pixel
        , GLuint palette_u
        )
380   {
        glUniform1i(u_N_0, prev->Nu);
        glUniform1i(u_NF_0, prev->NFu);
        glUniform1i(u_DEX_0, prev->DEXu);
        glUniform1i(u_DEY_0, prev->DEYu);
385     glUniform1i(u_N_1, next->Nu);
        glUniform1i(u_NF_1, next->NFu);
        glUniform1i(u_DEX_1, next->DEXu);
        glUniform1i(u_DEY_1, next->DEYu);
        glUniform1f(u_phase, phase);
390     glUniform1f(u_time, time);
        glUniform1i(u_samples, samples_per_pixel);
        glUniform1i(u_palette, palette_u);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
      }
395
      static GLuint vertex_fragment_shader(const char *vert, const char *frag) {
        GLuint program = glCreateProgram();
        {
          GLuint shader = glCreateShader(GL_VERTEX_SHADER);
400       glShaderSource(shader, 1, &vert, 0);
          glCompileShader(shader);
```

```
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  glLinkProgram(program);
  debug_program(program);
  return program;
}

int main(int argc, char **argv)
{
  // parse arguments
  bool reverse_keyframes = false;
  std::vector<std::string> positional_arguments;
  for (int i = 1; i < argc; ++i)
  {
    std::string argument(argv[i]);
    if (argument == "-?" || argument == "-h" || argument == "--help")
    {
      std::cout <<
        "usage:\n    " << argv[0] << " [options] stem count width height samples↙
            ↳  fps > stream.ppm\n"
        "options:\n"
        "    -?, -h, --help       print this message\n"
        "    -v, -V, --version    print version string\n"
        "    --[no-]reverse       (dis)enable keyframe order reversal\n"
        "arguments:\n"
        "    stem                 keyframes are in files stem####.exr\n"
        "    count                number of output video frames\n"
        "    width                output video frame width\n"
        "    height               output video frame height\n"
        "    samples              output video samples per pixel\n"
        "    fps                  output video frames per second\n"
        ;
      return 0;
    }
    else if (argument == "-v" || argument == "-V" || argument == "--version")
    {
      std::cout << "0.1\n";
      return 0;
    }
    else if (argument == "--reverse")
    {
      reverse_keyframes = true;
    }
    else if (argument == "--no-reverse")
    {
      reverse_keyframes = false;
    }
```

```
    else
    {
      positional_arguments.push_back(argument);
    }
  }
  if (positional_arguments.size() != 6)
  {
    std::cerr << argv[0] << ": error: expected 6 positional arguments\n";
    return 1;
  }
  std::string keyframe_stem = positional_arguments[0];
  int output_frame_count = atoi(positional_arguments[1].c_str());
  if (output_frame_count <= 0)
  {
    std::cerr << argv[0] << ": error: bad output frame count <= 0\n";
    return 1;
  }
  int output_frame_width = atoi(positional_arguments[2].c_str());
  if (output_frame_width <= 0)
  {
    std::cerr << argv[0] << ": error: bad output frame width <= 0\n";
    return 1;
  }
  int output_frame_height = atoi(positional_arguments[3].c_str());
  if (output_frame_height <= 0)
  {
    std::cerr << argv[0] << ": error: bad output frame height <= 0\n";
    return 1;
  }
  int samples_per_pixel = atoi(positional_arguments[4].c_str());
  if (samples_per_pixel <= 0)
  {
    std::cerr << argv[0] << ": error: bad samples per pixel <= 0\n";
    return 1;
  }
  double fps = atof(positional_arguments[5].c_str());
  if (! (fps > 0))
  {
    std::cerr << argv[0] << ": error: bad frames per second <= 0\n";
    return 1;
  }
  // count keyframes
  struct stat sb;
  int keyframe_count = 0;
  std::string filename_buf;
  do
  {
    std::ostringstream filename;
    filename << keyframe_stem << std::setfill('0') << std::setw(4) << ↙
        ↳ keyframe_count << ".exr";
    filename_buf = filename.str();
    keyframe_count++;
  } while (! stat(filename_buf.c_str(), &sb));
  keyframe_count--;
  if (keyframe_count <= 1)
  {
    std::cerr << argv[0] << ": error: bad input keyframe count <= 1\n";
```

```
    return 1;
  }
  // initialize OpenGL
  glfwInit();
  glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
  glfwWindowHint(GLFW_DECORATED, GL_FALSE);
  GLFWwindow *window = glfwCreateWindow(output_frame_width, output_frame_height,
      argv[0], 0, 0);
  glfwMakeContextCurrent(window);
  glewExperimental = GL_TRUE;
  glewInit();
  glGetError(); // discard common error from glew
  // set up vertex data
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_BLEND);
  glClearColor(0, 0, 0, 1);
  GLuint vao;
  glGenVertexArrays(1, &vao);
  glBindVertexArray(vao);
  GLuint vbo;
  glGenBuffers(1, &vbo);
  glBindBuffer(GL_ARRAY_BUFFER, vbo);
  float vertices[] = { 0, 0, 1, 0, 0, 1, 1, 1 };
  glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
  glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(0);
  // set up palette data
  GLuint palette_t;
  glGenTextures(1, &palette_t);
  GLuint palette_u = next_texture_unit++;
  glActiveTexture(GL_TEXTURE0 + palette_u);
  glBindTexture(GL_TEXTURE_1D, palette_t);
  unsigned char palette[] =
    { 255,255,255
    , 128,0,64
    , 160,0,0
    , 192,128,0
    , 64,128,0
    , 0,255,255
    , 64,128,255
    , 0,0,255
    };
  glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 8, 0, GL_RGB, GL_UNSIGNED_BYTE, palette
      );
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  // compile shaders
  GLuint program = vertex_fragment_shader
    ( zoom_interpolate_gl4_vert
    , zoom_interpolate_gl4_frag
    );
```

```
    glUseProgram(program);
    // compute speed
    /*
       first output frame is at first input keyframe
       last input keyframe is at last-but-one output frame
       keyframes      |       |       |       | (4)
       output frames  | | | | | | | | | : (8)
       so, speed = (keyframes - 1) / (output frames)
       then offset output frames by incr/2 to avoid rounding issues at eof
    */
    double phase_increment = (keyframe_count - 1) / double(output_frame_count);
    // main loop
    double phase = phase_increment / 2;
    Frame *previous_keyframe = nullptr,
          *next_keyframe = nullptr;
    unsigned char *output = new unsigned char[ssize_t(3) * output_frame_width * ↙
        ↳ output_frame_height];
    int previous_keyframe_index = -1;
    int next_keyframe_index = -1;
    for ( int output_frame_index = 0
        ; output_frame_index < output_frame_count
        ; ++output_frame_index
        )
    {
      if (floor(phase) != previous_keyframe_index || ceil(phase) != ↙
          ↳ next_keyframe_index)
      {
        // need to read a frame
        if (floor(phase) == next_keyframe_index || ceil(phase) == ↙
            ↳ previous_keyframe_index)
        {
          // we advanced to next or previous frame, can reuse the frame
          std::swap(previous_keyframe, next_keyframe);
          std::swap(previous_keyframe_index, next_keyframe_index);
        }
        if (floor(phase) != previous_keyframe_index)
        {
          previous_keyframe_index = floor(phase);
          std::ostringstream keyframe_file;
          keyframe_file << keyframe_stem << std::setfill('0') << std::setw(4) <<
            ( reverse_keyframes
            ? keyframe_count - 1 - previous_keyframe_index
            : previous_keyframe_index
            ) << ".exr";
          if (! (previous_keyframe = read_frame(previous_keyframe, keyframe_file.↙
              ↳ str(), argv[0])))
            return 1;
        }
        if (ceil(phase) != next_keyframe_index)
        {
          next_keyframe_index = ceil(phase);
          std::ostringstream keyframe_file;
          keyframe_file << keyframe_stem << std::setfill('0') << std::setw(4) <<
            ( reverse_keyframes
            ? keyframe_count - 1 - next_keyframe_index
            : next_keyframe_index
            ) << ".exr";
```

```
        if (! (next_keyframe = read_frame(next_keyframe, keyframe_file.str(), ↙
            ↳ argv[0])))
          return 1;
      }
    }
    if (are_compatible(previous_keyframe, next_keyframe))
    {
      glClear(GL_COLOR_BUFFER_BIT);
      double time = output_frame_index / fps;
      interpolate_frames(previous_keyframe, next_keyframe, phase - floor(phase),↙
          ↳ time, samples_per_pixel, palette_u);
      glReadPixels(0, 0, output_frame_width, output_frame_height, GL_RGB, ↙
          ↳ GL_UNSIGNED_BYTE, output);
      glfwSwapBuffers(window);
      if (! output)
      {
        std::cerr << argv[0] << ": error: interpolating keyframes failed\n";
        return 1;
      }
      fprintf(stdout, "P6\n%d %d\n255\n", output_frame_width, ↙
          ↳ output_frame_height);
      fwrite(output, size_t(3) * output_frame_width * output_frame_height, 1, ↙
          ↳ stdout);
      int e;
      while ((e = glGetError()))
        fprintf(stderr, "GLERROR(%d)\n", e);
    }
    else
    {
      std::cerr << argv[0] << ": error: incompatible keyframes\n";
      return 1;
    }
    phase += phase_increment;
  }
  return 0;
}

#if 0
// motion blur
static double shutter = 0.0;
static int motion_count = 1;
  for (motion = 0; motion < motion_count; ++motion) {
    double motion_alpha = 1.0 / (motion + 1.0);
    if (motion <= shutter * motion_count) {
      // scale and blend
      glUniform1iARB(combine_tex0, which);
      glUniform1iARB(combine_tex1, 1 - which);
      glUniform1fARB(combine_phase, phase);
      glUniform1fARB(combine_alpha, motion_alpha);
      double x = 0.5*pow(0.5, phase);
      double y = 0.5*pow(0.5, phase);
      glBegin(GL_QUADS); {
        glTexCoord2f(0.5 - x, 0.5 + y); glVertex2f(0, 1);
        glTexCoord2f(0.5 + x, 0.5 + y); glVertex2f(1, 1);
        glTexCoord2f(0.5 + x, 0.5 - y); glVertex2f(1, 0);
        glTexCoord2f(0.5 - x, 0.5 - y); glVertex2f(0, 0);
      } glEnd();
```

```
      }
      // advance
      phase += increment / motion_count;
    }

    /*
    zoom0 = 0.5 ** phase
    zoom1 = 0.5 ** (phase + increment / motion_count)
    texcoord = (width/2 + width/2 * zoom , height/2 + height/2 * zoom)
    assume phase = 0
    texcoord0 = (width, height)
    texcoord1 = (width, height) * (1 + 0.5 ** (increment / motion_count)) / 2
    texcoord1 - texcoord2 = (width, height) * [(1 + 0.5 ** (increment / ↙
        ↳ motion_count)) / 2 - 1]
    | tex1 - tex2 | = sqrt(w^2+h^2) * [(1 + 0.5 ** (inc / mot)) / 2 - 1]
    | tex1 - tex2 | == 1  for smooth motion blur
    2*[1/sqrt(w^2+h^2) + 1] - 1 = 0.5 ** (inc / mot)
    2/sqrt(w^2 + h^2) + 1 = 0.5**(inc/mot)
    log (2 / sqrt (w^2 + h^2) + 1) = (inc / mot) log 0.5
    mot = inc * log 0.5 / log (2 / sqrt(w^2 + h^2) + 1)
    */
    motion_count = fmax(1.0, ceil(fabs(increment * log(0.5) / log(2.0 / sqrt(↙
        ↳ OWIDTH * OWIDTH + OHEIGHT * OHEIGHT) + 1.0))));
    fprintf(stderr, "zoom: motion_count = %d (%d)\n", motion_count, (int) ceil(↙
        ↳ shutter * motion_count));
#endif
```

10 zoom_interpolate_gl4_frag.glsl

```
/*
zoom-tools -- zoom video tools
Copyright (C) 2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero 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 Affero General Public License for more details.

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

#version 400 core
#extension GL_ARB_explicit_uniform_location : require

layout(location = 0) uniform usampler2D N_0;
layout(location = 1) uniform sampler2D NF_0;
layout(location = 2) uniform sampler2D DEX_0;
layout(location = 3) uniform sampler2D DEY_0;

layout(location = 4) uniform usampler2D N_1;
layout(location = 5) uniform sampler2D NF_1;
```

```
layout(location = 6) uniform sampler2D DEX_1;
layout(location = 7) uniform sampler2D DEY_1;

layout(location = 8) uniform float phase;
layout(location = 9) uniform float time;
layout(location = 10) uniform int samples;

layout(location = 11) uniform sampler1D palette;

in vec2 coord_0;
in vec2 coord_1;

layout(location = 0) out vec4 colour;

// http://lolengine.net/blog/2013/07/27/rgb-to-hsv-in-glsl
vec3 hsv2rgb(vec3 c)
{
  vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
  vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);
  return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);
}

uint hash(uint a) // burtle's 9th hash
{
  a = (a+0x7ed55d16u) + (a<<12u);
  a = (a^0xc761c23cu) ^ (a>>19u);
  a = (a+0x165667b1u) + (a<<5u);
  a = (a+0xd3a2646cu) ^ (a<<9u);
  a = (a+0xfd7046c5u) + (a<<3u);
  a = (a^0xb55a4f09u) ^ (a>>16u);
  return a;
}

uint hash(int a) { return hash(uint(a)); }
uint hash(float a) { return hash(floatBitsToUint(a)); }
uint hash(uvec2 a) { return hash(a.x ^ hash(a.y)); }
uint hash(uvec3 a) { return hash(a.x ^ hash(a.yz)); }
uint hash(uvec4 a) { return hash(a.x ^ hash(a.yzw)); }
uint hash(ivec2 a) { return hash(uint(a.x) ^ hash(a.y)); }
uint hash(ivec3 a) { return hash(uint(a.x) ^ hash(a.yz)); }
uint hash(ivec4 a) { return hash(uint(a.x) ^ hash(a.yzw)); }
uint hash(vec2 a) { return hash(floatBitsToUint(a.x) ^ hash(a.y)); }
uint hash(vec3 a) { return hash(floatBitsToUint(a.x) ^ hash(a.yz)); }
uint hash(vec4 a) { return hash(floatBitsToUint(a.x) ^ hash(a.yzw)); }
float uniform01(uint a) { return float(a) / 4294967296.0; }

vec2 random2(vec4 seed)
{
  return vec2
    ( uniform01(hash(uvec2(hash(seed), 1u)))
    , uniform01(hash(uvec2(hash(seed), 2u)))
    );
}

double NNF_0(vec2 d)
{
  return double(texture(N_0, coord_0 + d).x) + double(texture(NF_0, coord_0 + d)↙
```

```
      ↪ .x);
}

double NNF_1(vec2 d)
{
  return double(texture(N_1, coord_1 + d).x) + double(texture(NF_1, coord_1 + d)↙
      ↪ .x);
}

vec2 DE_0(vec2 d)
{
  return vec2(texture(DEX_0, coord_0 + d).x, texture(DEY_0, coord_0 + d).x);
}

vec2 DE_1(vec2 d)
{
  return vec2(texture(DEX_1, coord_1 + d).x, texture(DEY_1, coord_1 + d).x);
}

float wave(float p)
{
  return 0.5 + 0.5 * cos(6.283185307179586 * p);
}

vec4 colour1(double N, vec2 DE)
{
  // interior
  if (N >= double(0xffffFFFFu))
    return vec4(vec3(0.0), 1.0);
  // colour cycling
  float C0 = mod(time * 48000.0 / 20197.0 / 128.0, 1.0);
  float C1 = mod(time * 48000.0 / 20197.0 / 512.0, 1.0);
  float C2 = mod(time * 48000.0 / 20197.0 / 256.0, 1.0);
  float C3 = mod(time * 48000.0 / 20197.0 /   4.0, 1.0);
  // repeating cycles
  float N0 = float(mod(floor(N - 1024.0 + C0 * 257.0) / 257.0, 1.0));
  float N1 = float(mod(floor(N - 1024.0 + C1 * 126.0) / 126.0, 1.0));
  float N2 = float(mod(floor(N - 1024.0 + C2 *  26.0) /  26.0, 1.0));
  float N3 = float(mod(floor(N - 1024.0 + C3 *   5.0) /   5.0, 1.0));
  // base colour
  vec3 b = texture(palette, N0).bgr;
  // infinite waves colour
  vec3 w = hsv2rgb(vec3
    ( wave(N1)
    , wave(N2)
    , wave(N3)
    ));
  // blend
  vec3 h = mix(b, w, 0.5);
  // slopes
  float slope = dot(normalize(DE), vec2(1.0, 1.0));
  float strength = abs(slope) / (1.0 + length(DE));
  if (slope < 0.0) h = mix(h, vec3(0.0), vec3(clamp(strength, 0.0, 1.0)));
  if (slope > 0.0) h = mix(h, vec3(1.0), vec3(clamp(strength, 0.0, 1.0)));
  return vec4(h, 1.0);
}
```

```
void main(void)
{
  vec4 o = vec4(0.0);
  vec2 d0 = vec2(dFdx(coord_0).x, dFdy(coord_0).y);
  vec2 d1 = vec2(dFdx(coord_1).x, dFdy(coord_1).y);
  for (int i = 0; i < samples; ++i)
  {
    vec2 d = random2(vec4(gl_FragCoord.xy, time, float(i)));
    o += mix
      ( colour1(NNF_0(d * d0), DE_0(d * d0))
      , colour1(NNF_1(d * d1), DE_1(d * d1))
      , phase
      );
  }
  colour = o / float(samples);
}
```

11 zoom_interpolate_gl4_vert.glsl

```
/*
zoom-tools -- zoom video tools
Copyright (C) 2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero 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 Affero General Public License for more details.

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

#version 400 core
#extension GL_ARB_explicit_uniform_location : require

layout(location = 8) uniform float phase;

layout(location = 0) in vec2 coord;

out vec2 coord_0;
out vec2 coord_1;

void main(void)
{
  coord_0 = pow(0.5, phase + 1.0) * (coord - vec2(0.5)) + vec2(0.5);
  coord_1 = pow(0.5, phase      ) * (coord - vec2(0.5)) + vec2(0.5);
  gl_Position = vec4(2.0 * coord - vec2(1.0), 0.0, 1.0);
}
```

12 zoom-output-copy

```
#!/bin/sh
```

```
cp "${2}" "output/$(printf %08d ${1}).exr"
```

13 zoom-output-kfp-8

```
#!/bin/sh
ColorSpeed=10.138613861386139
FrameRateFactor=1
sed -i "s/^ColorOffset:.*$/ColorOffset: $(ghc -fimplicit-import-qualified -e "↙
    ↪ floor (Data.Fixed.mod' (${ColorSpeed} * ${FrameRateFactor} * (${1} + 0.5))↙
    ↪  1024)")/g" palette.kfp
kf.exe -s "settings.kfs" -o "${2}" -c "palette.kfp" -t "${2}.tif" --log error &&
convert "${2}.tif" -depth 8 ppm:- &&
rm -f "${2}.tif"
```

14 zoom-output-ppm-16

```
#!/bin/sh
convert -colorspace RGB "${2}" -colorspace sRGB ppm:-
```

15 zoom-output-ppm-8

```
#!/bin/sh
convert -colorspace RGB "${2}" -colorspace sRGB -depth 8 ppm:-
```