exrtact

Claude Heiland-Allen

2019

Contents

1 exrtact.cpp

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

#include <iostream>

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

int main(int argc, char **argv)
{
  // args
  if (argc != 4)
  {
    std::cerr << "usage: " << argv[0] << " in.exr out.exr channel" << std::endl;
    return 1;
  }
  const char *ifilename = argv[1];
  const char *ofilename = argv[2];
  const char *ichannel = argv[3];
  const char *ochannel = "Y";
  // read
  InputFile ifile(ifilename);
  const Header &h = ifile.header();
  Box2i dw = h.dataWindow();
  size_t width = dw.max.x - dw.min.x + 1;
  size_t height = dw.max.y - dw.min.y + 1;
  float *Y = new float[width * height];
  FrameBuffer ifb;
```

```
    ifb.insert(ichannel, Slice(IMF::FLOAT, (char *) (&Y[0] - dw.min.x - dw.min.y * width), sizeof(Y[0]) * 1, sizeof(Y[0]) * width, 1, 1, 0));
    ifile.setFrameBuffer(ifb);
    ifile.readPixels(dw.min.y, dw.max.y);
    // write
    Header oh(width, height);
    oh.channels().insert(ochannel, Channel(IMF::FLOAT));
    OutputFile ofile(ofilename, oh);
    FrameBuffer ofb;
    ofb.insert(ochannel, Slice(IMF::FLOAT, (char *) (&Y[0]), sizeof(Y[0]) * 1, sizeof(Y[0]) * width));
    ofile.setFrameBuffer(ofb);
    ofile.writePixels(height);
    delete[] Y;
    return 0;
}
```

2 exrtactile.cpp

```
#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 <algorithm>
#include <iostream>
#include <cstring>

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

template <typename T>
std::set<T> intersect(std::set<T> &s1, std::set<T> &s2)
{
  std::set<T> intersection;
  std::set_intersection
    ( s1.begin(), s1.end()
    , s2.begin(), s2.end()
    , std::inserter(intersection, intersection.begin())
    );
  return intersection;
}

int main(int argc, char **argv)
{
  // args
  if (! (argc >= 5))
  {
    std::cerr << "usage: " << argv[0] << " stem factor stratify out.exr [channel [channel [...]]]" << std::endl;
    return 1;
  }
```

```
  const char *istem = argv[1];
  const int factor = atoi(argv[2]);
  const bool stratify = atoi(argv[3]);
  const char *ofilename = argv[4];
  std::set<std::string> keep_channels;
  for (int i = 5; i < argc; ++i)
  {
    keep_channels.insert(argv[i]);
  }

  std::set<std::string> uint_channels, half_channels, float_channels;
  uint32_t *uint_data = 0;
  half *half_data = 0;
  float *float_data = 0;
  size_t width = 0;
  size_t height = 0;

  for (int y = 0; y < factor; ++y)
  {
    for (int x = 0; x < factor; ++x)
    {
      char ifilename[1000];
      std::snprintf(ifilename, 999, "%s-%04d-%04d.exr", istem, y, x);
      std::fprintf(stderr, "reading %s\n", ifilename);
      InputFile ifile(ifilename);
      const Header &h = ifile.header();
      Box2i dw = h.dataWindow();
      size_t fwidth = dw.max.x - dw.min.x + 1;
      size_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();
          for (ChannelList::ConstIterator j = cl.begin(); j != cl.end(); ++j)
          {
            if (j.channel().xSampling != 1)
            {
              std::cerr << "ERROR: xSampling != 1" << std::endl;
              return 1;
            }
            if (j.channel().ySampling != 1)
            {
              std::cerr << "ERROR: ySampling != 1" << std::endl;
              return 1;
            }
            switch (j.channel().type)
            {
              case UINT:
                fuint_channels.insert(j.name());
                break;
              case HALF:
                fhalf_channels.insert(j.name());
```

```
                break;
95            case FLOAT:
                ffloat_channels.insert(j.name());
                break;
              default:
                std::cerr << "ERROR: unknown channel type " << j.channel().type ↙
                    ↳ << std::endl;
100             return 1;
            }
          }
          fuint_channels = keep_channels.size() > 0 ? intersect(keep_channels, ↙
              ↳ fuint_channels) : fuint_channels;
          fhalf_channels = keep_channels.size() > 0 ? intersect(keep_channels, ↙
              ↳ fhalf_channels) : fhalf_channels;
105       ffloat_channels = keep_channels.size() > 0 ? intersect(keep_channels, ↙
              ↳ ffloat_channels) : ffloat_channels;

          if (x == 0 && y == 0)
          {
            width = fwidth;
110         height = fheight;
            uint_channels = fuint_channels;
            half_channels = fhalf_channels;
            float_channels = ffloat_channels;
            if (uint_channels.size() > 0)
115         {
              uint_data = new uint32_t[uint_channels.size() * factor * width * ↙
                  ↳ factor * height];
            }
            if (half_channels.size() > 0)
            {
120           half_data = new half[half_channels.size() * factor * width * ↙
                  ↳ factor * height];
            }
            if (float_channels.size() > 0)
            {
              float_data = new float[float_channels.size() * factor * width * ↙
                  ↳ factor * height];
125         }
          }
          else
          {
            if (width != fwidth || height != fheight || uint_channels != ↙
                ↳ fuint_channels || half_channels != fhalf_channels || ↙
                ↳ float_channels != ffloat_channels)
130         {
              std::cerr << "incompatible files" << std::endl;
              return 1;
            }
          }
135     }
        }

        // read image
        FrameBuffer ifb;
140     if (stratify)
        {
```

```
        int k = 0;
        for (auto name : uint_channels)
        {
145       ifb.insert
            ( name.c_str()
            , Slice
             ( IMF::UINT
              , (char *) (&uint_data[0] + k - dw.min.x * factor - dw.min.y * ↙
                  ↳ factor * width + x * uint_channels.size() + y * ↙
                  ↳ uint_channels.size() * width * factor)
150           , sizeof(uint_data[0]) * factor * uint_channels.size()
              , sizeof(uint_data[0]) * factor * factor * width * uint_channels.↙
                  ↳ size()
              , 1, 1
              , 0
              )
155         );
          ++k;
        }
        k = 0;
        for (auto name : half_channels)
160     {
          ifb.insert
            ( name.c_str()
            , Slice
              ( IMF::HALF
165           , (char *) (&half_data[0] + k - dw.min.x * factor - dw.min.y * ↙
                  ↳ factor * width + x * half_channels.size() + y * ↙
                  ↳ half_channels.size() * width * factor)
              , sizeof(half_data[0]) * factor * half_channels.size()
              , sizeof(half_data[0]) * factor * factor * width * half_channels.↙
                  ↳ size()
              , 1, 1
              , 0
170           )
            );
          ++k;
        }
        k = 0;
175     for (auto name : float_channels)
        {
          ifb.insert
            ( name.c_str()
            , Slice
180           ( IMF::FLOAT
              , (char *) (&float_data[0] + k - dw.min.x * factor - dw.min.y * ↙
                  ↳ factor * width + x * float_channels.size() + y * ↙
                  ↳ float_channels.size() * width * factor)
              , sizeof(float_data[0]) * factor * float_channels.size()
              , sizeof(float_data[0]) * factor * factor * width * float_channels↙
                  ↳ .size()
              , 1, 1
185           , 0
              )
            );
          ++k;
        }
```

```
190         }
            else
            {
              int k = 0;
              for (auto name : uint_channels)
195           {
                ifb.insert
                  ( name.c_str()
                  , Slice
                    ( IMF::UINT
                    , (char *) (&uint_data[0] + k - dw.min.x - dw.min.y * width + x * ↙
                        ↳ uint_channels.size() * width + y * uint_channels.size() * ↙
                        ↳ factor * width * height)
                    , sizeof(uint_data[0]) * uint_channels.size()
                    , sizeof(uint_data[0]) * factor * width * uint_channels.size()
                    , 1, 1
                    , 0
205                 )
                 );
                ++k;
              }
              k = 0;
              for (auto name : half_channels)
              {
                ifb.insert
                  ( name.c_str()
                  , Slice
215                 ( IMF::HALF
                    , (char *) (&half_data[0] + k - dw.min.x - dw.min.y * width + x * ↙
                        ↳ half_channels.size() * width + y * half_channels.size() * ↙
                        ↳ factor * width * height)
                    , sizeof(half_data[0]) * half_channels.size()
                    , sizeof(half_data[0]) * factor * width * half_channels.size()
                    , 1, 1
220                 , 0
                    )
                  );
                ++k;
              }
225           k = 0;
              for (auto name : float_channels)
              {
                ifb.insert
                  ( name.c_str()
230               , Slice
                    ( IMF::FLOAT
                    , (char *) (&float_data[0] + k - dw.min.x - dw.min.y * width + x *↙
                        ↳  float_channels.size() * width + y * float_channels.size() *↙
                        ↳  factor * width * height)
                    , sizeof(float_data[0]) * float_channels.size()
                    , sizeof(float_data[0]) * factor * width * float_channels.size()
235                 , 1, 1
                    , 0
                    )
                 );
                ++k;
240           }
```

```
        }
        ifile.setFrameBuffer(ifb);
        ifile.readPixels(dw.min.y, dw.max.y);
      }
    }

    // write
    std::fprintf(stderr, "writing %s\n", ofilename);
    Header oh(width * factor, height * factor);
    for (auto name : uint_channels)
    {
      oh.channels().insert(name.c_str(), Channel(IMF::UINT));
    }
    for (auto name : half_channels)
    {
      oh.channels().insert(name.c_str(), Channel(IMF::HALF));
    }
    for (auto name : float_channels)
    {
      oh.channels().insert(name.c_str(), Channel(IMF::FLOAT));
    }
    OutputFile ofile(ofilename, oh);
    FrameBuffer ofb;
    int k = 0;
    for (auto name : uint_channels)
    {
      ofb.insert(name, Slice(IMF::UINT, (char *) (&uint_data[0] + k), sizeof(↙
          ↪ uint_data[0]) * uint_channels.size(), sizeof(uint_data[0]) * factor * ↙
          ↪ width * uint_channels.size()));
      ++k;
    }
    k = 0;
    for (auto name : half_channels)
    {
      ofb.insert(name, Slice(IMF::HALF, (char *) (&half_data[0] + k), sizeof(↙
          ↪ half_data[0]) * half_channels.size(), sizeof(half_data[0]) * factor * ↙
          ↪ width * half_channels.size()));
      ++k;
    }
    k = 0;
    for (auto name : float_channels)
    {
      ofb.insert(name, Slice(IMF::FLOAT, (char *) (&float_data[0] + k), sizeof(↙
          ↪ float_data[0]) * float_channels.size(), sizeof(float_data[0]) * factor↙
          ↪  * width * float_channels.size()));
      ++k;
    }
    ofile.setFrameBuffer(ofb);
    ofile.writePixels(factor * height);
    return 0;
}
```

3 exrtact-preview.cpp

```
#include <ImfNamespace.h>
#include <ImfInputFile.h>
#include <ImfHeader.h>
```

```
#include <ImfPreviewImage.h>

#include <zlib.h>
#include <png.h>

#include <cstdio>

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

int main(int argc, char **argv)
{
  // args
  if (argc != 3)
  {
    fprintf(stderr, "usage: %s in.exr out.png\n", argv[0]);
    return 1;
  }
  const char *ifilename = argv[1];
  const char *ofilename = argv[2];
  // read
  InputFile ifile(ifilename);
  auto i = ifile.header().previewImage();
  auto w = i.width();
  auto h = i.height();
  auto p = i.pixels();
  auto rows = new unsigned char *[h];
  for (unsigned int y = 0; y < h; ++y)
    rows[y] = (unsigned char *) (p + y * w);
  // write
  FILE *file = fopen(ofilename, "wb");
  if (! file)
    return 1;
  jmp_buf jmpbuf;
  png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, &jmpbuf, ↙
      ↳ nullptr, nullptr);
  if (! png)
    return 1;
  png_infop info = png_create_info_struct(png);
  if (! info)
  {
    png_destroy_write_struct(&png, 0);
    fclose(file);
    return 1;
  }
  if (setjmp(jmpbuf))
  {
    png_destroy_write_struct(&png, &info);
    fclose(file);
    return 1;
  }
  png_init_io(png, file);
  png_set_compression_level(png, Z_BEST_COMPRESSION);
  png_set_IHDR(png, info, w, h, 8, PNG_COLOR_TYPE_RGBA, PNG_INTERLACE_ADAM7, ↙
      ↳ PNG_COMPRESSION_TYPE_DEFAULT, PNG_FILTER_TYPE_DEFAULT);
  png_set_gAMA(png, info, 2.2);
```

```
  png_write_info(png, info);
  png_write_image(png, rows);
  png_write_end(png, 0);
  // cleanup
  png_destroy_write_struct(&png, &info);
  fclose(file);
  delete [] rows;
  return 0;
}
```

4 exrtoohot.cpp

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

#include <iostream>

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

int main(int argc, char **argv)
{
  // args
  if (argc != 3)
  {
    std::cerr << "usage: " << argv[0] << " in.exr channel" << std::endl;
    return 1;
  }
  const char *ifilename = argv[1];
  const char *ichannel = argv[2];
  // read
  InputFile ifile(ifilename);
  const Header &h = ifile.header();
  Box2i dw = h.dataWindow();
  ssize_t width = dw.max.x - dw.min.x + 1;
  ssize_t height = dw.max.y - dw.min.y + 1;
  half *Y = new half[width * height];
  FrameBuffer ifb;
  ifb.insert(ichannel, Slice(IMF::HALF, (char *) (&Y[0] - dw.min.x - dw.min.y * ↙
      ↳ width), sizeof(Y[0]) * 1, sizeof(Y[0]) * width, 1, 1, 0));
  ifile.setFrameBuffer(ifb);
  ifile.readPixels(dw.min.y, dw.max.y);
  // analyse
  ssize_t n_nan = 0, n_pinf = 0, n_ninf = 0, n_zero = 0, n_neg = 0, n_sub = 0, ↙
      ↳ n_norm = 0;
  for (ssize_t k = 0; k < width * height; ++k)
  {
    double y = Y[k];
    if (isnan(y)) n_nan++;
    else if (isinf(y) && y > 0) n_pinf++;
    else if (isinf(y) && y < 0) n_ninf++;
```

```
      else if (y == 0) n_zero++;
      else if (y < 0) n_neg++;
      else if (y < pow(2, -14)) n_sub++;
      else n_norm++;
    }
    delete[] Y;
    std::cout
      << "nan\t" << n_nan << "\n"
      << "+inf\t" << n_pinf << "\n"
      << "-inf\t" << n_ninf << "\n"
      << "0.0\t" << n_zero << "\n"
      << "<0\t" << n_neg << "\n"
      << "sub\t" << n_sub << "\n"
      << "ok\t" << n_norm << "\n"
      ;
    return 0;
}
```

5 .gitignore

```
exrtact
exrtact-preview
exrtactile
exrtoohot
*.exe*
```

6 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: exrtact exrtact-preview exrtactile exrtoohot

exrtact: exrtact.cpp
	g++ -std=c++14 -Wall -Wextra -pedantic -Wno-deprecated -O3 -o exrtact exrtact.cpp `pkg-config --cflags --libs OpenEXR`

exrtact-preview: exrtact-preview.cpp
	g++ -std=c++14 -Wall -Wextra -pedantic -Wno-deprecated -O3 -o exrtact-preview exrtact-preview.cpp `pkg-config --cflags --libs OpenEXR libpng`

exrtactile: exrtactile.cpp
	g++ -std=c++14 -Wall -Wextra -pedantic -Wno-deprecated -O3 -o exrtactile  exrtactile.cpp `pkg-config --cflags --libs OpenEXR`

exrtoohot: exrtoohot.cpp
	g++ -std=c++14 -Wall -Wextra -pedantic -Wno-deprecated -O3 -o exrtoohot exrtoohot.cpp `pkg-config --cflags --libs OpenEXR`

exrtactile.exe: exrtactile.cpp
	x86_64-w64-mingw32-g++ $(WCOMPILE_FLAGS) $(WLINK_FLAGS) -o exrtactile.exe exrtactile.cpp -lIlmImf-2_4 -lImath-2_4 -lHalf-2_4 -lIex-2_4 -
```

```
    ↳ lIexMath-2_4 -lIlmThread-2_4 -lz
```

7 README.md

```
# exrtact

extract channels of exr files
<https://mathr.co.uk/exrtact>
<https://code.mathr.co.uk/exrtact>

## get

    sudo apt install git make g++ pkg-config libopenexr-dev libpng-dev
    git clone https://code.mathr.co.uk/exrtact.git
    cd exrtact
    make

## usage

### channel extraction

    exrtact in.exr out.exr chan

- `in.exr` is the input file, it should have a channel named `chan`
  which will be extracted
- `out.exr` is the output file which will be created, it will have
  a channel named `Y` which contains the extracted channel

### preview extraction

    exrtact-preview in.exr out.png

- `in.exr` is the input file, it should have a preview image
  which will be extracted
- `out.png` is the output file which will be created, it will have
  RGBA8 channels containing the extracted preview image

### image tiling

    exrtactile stem factor stratify out.exr [channel [channel [...]]

- combines `stem-YYYY-XXXX.exr` into `out.exr`
  where each `XXXX`, `YYYY` is zero-padded
  4 decimal digits from `0` to `factor - 1`.
  `X` increases left to right.
  `Y` increases top to bottom.
- if `stratify` is `0`, does regular tiling
  (the images are placed side by side)
- if `stratify` is `1`, does "stratified" tiling
  (the pixels get interleaved)
- if channel list is specified, only those channels are used,
  otherwise all channels are used

## bugs

- no license
```

```
### channel extraction

- only single part scanline flat images are supported
- only float32 channels are supported
  (OpenEXR may cast other types on read, potentially losing data
  or expanding space usage for no gain)

### preview extraction

- only single part images are supported

### image tiling

- preview images are not processed
- only single part scanline flat images are supported
- only one factor for both dimensions
```