v4l2-examples

Claude Heiland-Allen

2011–2015

Contents

1 .gitignore

```
dist
```

2 LICENSE

```
Copyright (c)2011, Claude Heiland-Allen

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.

    * Neither the name of Claude Heiland-Allen nor the names of other
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

3 Setup.hs

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

4 v4l2-capture/v4l2-capture.hs

```
module Main (main) where

import Control.Monad (forM_, when)
import Foreign (Ptr)
import System.Environment (getArgs)
import System.Exit (exitFailure)
import System.IO (hPutBuf, hPutStr, hPutStrLn, stdout, stderr)
import System.IO.Error (tryIOError)
import GHC.IO.Exception (IOErrorType(Interrupted), ioe_type)

import Graphics.V4L2

main :: IO ()
main = do
  (devname, verbose) <- checkArgs =<< getArgs
  e <- tryIOError $ withDevice devname $ \d -> do
    f <- setFormat d Capture . (\f->f{ imagePixelFormat = PixelRGB24 }) =<< ↙
        ↳ getFormat d Capture
    checkFormat f
    info $ "frame size: " ++ show (imageWidth f) ++ "x" ++ show (imageHeight f) ↙
        ↳ ++ " pixels (" ++ show (imageSize f) ++ " bytes)"
    forM_ [(0 :: Int) ..] $ \i -> do
      withFrame d f $ \p n -> do
        if n == imageSize f
          then do
            when verbose $ do
              info $ "Frame number " ++ show i
            writePPM (imageWidth f) (imageHeight f) p
          else warn $ "incomplete frame (" ++ show n ++ " bytes, expected " ++ ↙
              ↳ show (imageSize f) ++ " bytes)"
  case e of
    Left f | ioe_type f == Interrupted -> return ()
           | otherwise -> ioError f
    Right () -> return ()

writePPM :: Int -> Int -> Ptr a -> IO ()
writePPM w h p = do
  hPutStr stdout $ "P6\n" ++ show w ++ " " ++ show h ++ " 255\n"
  hPutBuf stdout p (w * h * 3)

checkFormat :: ImageFormat -> IO ()
checkFormat f = do
  when (imagePixelFormat f /= PixelRGB24) $ err "could not set RGB24 pixel ↙
      ↳ format"
  when (imageBytesPerLine f /= imageWidth f * 3) $ err "cannot handle extra ↙
      ↳ padding"
  when (imageSize f /= imageBytesPerLine f * imageHeight f) $ err "cannot handle↙
      ↳  image size"

-- TODO verbosity flag
```

```
checkArgs :: [String] -> IO (String, Bool)
checkArgs [devname] = return (devname, False)
checkArgs _ = err $ "bad arguments; usage: v4l2-capture /dev/video0"

err :: String -> IO a
err msg = (hPutStrLn stderr $ "**ERROR: [v4l2-capture] " ++ msg) >> exitFailure

warn :: String -> IO ()
warn msg = hPutStrLn stderr $ "++ WARN: [v4l2-capture] " ++ msg

info :: String -> IO ()
info msg = hPutStrLn stderr $ "   INFO: [v4l2-capture] " ++ msg
```

5 v4l2-examples.cabal

```
Name:                v4l2-examples
Version:             0.1.0.2
Synopsis:            video for linux two examples

Description:
  This package contains examples using the v4l2 package:
  .
    * v4l2-capture - dumps PPM frames from a video device to stdout:
  .
    @$ v4l2-capture /dev/video0 > out.ppm@
  .
    * v4l2-histogram - shows RGB histogram overlaid on mirror image, using ↙
        ↳ OpenGL/GLUT for display.
  .
    @$ v4l2-histogram /dev/video0@

Homepage:            http://code.mathr.co.uk/v4l2-examples
License:             BSD3
License-file:        LICENSE
Author:              Claude Heiland-Allen
Maintainer:          claude@mathr.co.uk

Category:            Graphics
Build-type:          Simple
Cabal-version:       >=1.2

Executable v4l2-capture
  Main-is: v4l2-capture/v4l2-capture.hs
  Build-depends:
    base >= 3 && < 5,
    v4l2 >= 0.1 && < 0.2
  ghc-options: -Wall

Executable v4l2-histogram
  Main-is: v4l2-histogram/v4l2-histogram.hs
  Build-depends:
    base >= 3 && < 5,
    GLUT >= 2.1 && < 2.4,
    v4l2 >= 0.1 && < 0.2
  ghc-options: -Wall
```

6 v4l2-histogram/v4l2-histogram.hs

```
module Main (main) where

import Prelude hiding (map, sum)

import Control.Monad (when)
import Data.Word (Word8, Word32)
import Foreign (Ptr, nullPtr, allocaArray, peekElemOff, pokeElemOff)
import System.Exit (exitFailure, exitSuccess)
import System.IO (hPutStrLn, stderr)

import Graphics.UI.GLUT hiding (PixelFormat, histogram, imageHeight)
import Graphics.V4L2

pixel :: PixelFormat
pixel = PixelRGB24

main :: IO ()
main = do
  initialWindowSize $= Size 640 480
  initialDisplayMode $= [DoubleBuffered]
  devname <- checkArgs . snd =<< getArgsAndInitialize
  withDevice devname $ \d -> do
    f <- setFormat d Capture . (\f->f{ imagePixelFormat = pixel }) =<< getFormat d Capture
    checkFormat f
    info $ "frame size: " ++ show (imageWidth f) ++ "x" ++ show (imageHeight f) ++ " pixels (" ++ show (imageSize f) ++ " bytes)"
    let (_, texSize:_) = break (>= (imageWidth f `max` imageHeight f)) $ iterate (2*) 1
    _ <- createWindow "v4l2-histogram"
    depthFunc $= Nothing
    texture Texture2D $= Enabled
    [th, ti] <- genObjectNames 2
    textureBinding Texture2D $= Just th
    texImage2D Nothing NoProxy 0 RGBA' (TextureSize2D 256 256) 0 (PixelData RGBA UnsignedByte nullPtr)
    textureFilter Texture2D $= ((Nearest, Nothing), Nearest)
    textureWrapMode Texture2D S $= (Repeated, ClampToEdge)
    textureWrapMode Texture2D T $= (Repeated, ClampToEdge)
    textureBinding Texture2D $= Just ti
    texImage2D Nothing NoProxy 0 RGBA' (TextureSize2D (fromIntegral texSize) (fromIntegral texSize)) 0 (PixelData RGBA UnsignedByte nullPtr)
    textureFilter Texture2D $= ((Nearest, Nothing), Nearest)
    textureWrapMode Texture2D S $= (Repeated, ClampToEdge)
    textureWrapMode Texture2D T $= (Repeated, ClampToEdge)
    matrixMode $= Modelview 0
    loadIdentity
    matrixMode $= Projection
    loadIdentity
    ortho2D 0 1 1 0
    idleCallback $= Just (idle d f texSize ti th)
    displayCallback $= display d f texSize ti th
    keyboardMouseCallback $= Just (\_ _ _ _ -> exitSuccess)
    mainLoop

```

```
idle :: Device -> ImageFormat -> Int -> TextureObject -> TextureObject -> IO ()
idle d f _ ti th = withFrame d f $ \p n -> do
  if n == imageSize f
    then allocaArray (256 * 3) $ \h -> allocaArray (256 * 256 * 4) $ \q -> do
      histogram (imageWidth f * imageHeight f * 3) p h
      expand (fromIntegral $ imageWidth f * imageHeight f) h q
      textureBinding Texture2D $= Just ti
      texSubImage2D Nothing 0 (TexturePosition2D 0 0) (TextureSize2D (↙
          ↳ fromIntegral $ imageWidth f) (fromIntegral $ imageHeight f)) (↙
          ↳ PixelData RGB UnsignedByte p)
      textureBinding Texture2D $= Just th
      texSubImage2D Nothing 0 (TexturePosition2D 0 0) (TextureSize2D 256 256) (↙
          ↳ PixelData RGBA UnsignedByte q)
      postRedisplay Nothing
    else warn $ "incomplete frame (" ++ show n ++ " bytes, expected " ++ show (↙
        ↳ imageSize f) ++ " bytes)"

display :: Device -> ImageFormat -> Int -> TextureObject -> TextureObject -> IO ↙
    ↳ ()
display _ f texSize ti th = do
  textureBinding Texture2D $= Just ti
  renderPrimitive Quads (u 0 0 >> u 0 1 >> u 1 1 >> u 1 0)
  blend $= Enabled
  blendFunc $= (SrcAlpha, OneMinusSrcAlpha)
  textureBinding Texture2D $= Just th
  renderPrimitive Quads (v 0 0 >> v 0 1 >> v 1 1 >> v 1 0)
  blend $= Disabled
  swapBuffers
  where
    u :: GLfloat -> GLfloat -> IO ()
    u x y = texCoord (TexCoord2 (x * fromIntegral (imageWidth f) / fromIntegral ↙
        ↳ texSize) (y * fromIntegral (imageHeight f) / fromIntegral texSize)) >>↙
        ↳  vertex (Vertex2 (1 - x) y)
    v :: GLfloat -> GLfloat -> IO ()
    v x y = texCoord (TexCoord2 x y) >> vertex (Vertex2 x y)

histogram :: Int -> Ptr Word8 -> Ptr Word32 -> IO ()
histogram m p q = c 0 >> h 0 >> h 1 >> h 2
  where
    c i | i >= 256 * 3 = return ()
        | otherwise = do
            pokeElemOff q i 0
            c (i + 1)
    h i0 = h' i0
      where
        h' i  | i >= m = return ()
              | otherwise = do
                  j <- peekElemOff p i
                  let j' = fromIntegral j * 3 + i0
                  t <- peekElemOff q j'
                  pokeElemOff q j' (t + 1)
                  h' (i + 3)

expand :: Float -> Ptr Word32 -> Ptr Word8 -> IO ()
expand m p q = e 0 >> a 0
  where
    e i | i >= 256 = return ()
```

```
          | otherwise = e' 0 >> e' 1 >> e' 2 >> e (i + 1)
        where
          e' c = do
            s <- peekElemOff p (3 * i + c)
            let t | s == 0 = 255
                  | otherwise = round . max 0 . min 255 $ 256 * log (m / ↙
                    ↳ fromIntegral s) / log 256
            f t
            where
              f t = g 0
                where
                  g j | j == 256 = return ()
                      | otherwise = do
                          pokeElemOff q ((j * 256 + i) * 4 + c) $ if j > t then ↙
                              ↳ 255 else 0
                          g (j + 1)
      a i | i >= 256 * 256 * 4 = return ()
          | otherwise = do
              r <- peekElemOff q i
              if r > 0 then pokeElemOff q (i + 3) 255 else do
                g <- peekElemOff q (i + 1)
                if g > 0 then pokeElemOff q (i + 3) 255 else do
                  b <- peekElemOff q (i + 2)
                  if b > 0 then pokeElemOff q (i + 3) 255 else do
                    pokeElemOff q (i + 3) 0
              a (i + 4)

checkFormat :: ImageFormat -> IO ()
checkFormat f = do
  when (imagePixelFormat f /= pixel) $ err ("could not set pixel format " ++ ↙
      ↳ show pixel)
  when (imageBytesPerLine f /= imageWidth f * 3) $ err "cannot handle extra ↙
      ↳ padding"
  when (imageSize f /= imageBytesPerLine f * imageHeight f) $ err "cannot handle↙
      ↳  image size"

checkArgs :: [String] -> IO String
checkArgs [devname] = return devname
checkArgs _ = err $ "bad arguments; usage: v4l2-histogram /dev/video0"

err :: String -> IO a
err msg = (hPutStrLn stderr $ "**ERROR: [v4l2-histogram] " ++ msg) >> ↙
    ↳ exitFailure

warn :: String -> IO ()
warn msg = hPutStrLn stderr $ "++ WARN: [v4l2-histogram] " ++ msg

info :: String -> IO ()
info msg = hPutStrLn stderr $ "   INFO: [v4l2-histogram] " ++ msg
```