Linux Multimedia Programming

Graphics, Audio, Video

Curator: Charles Fox

Licence

Contents

1. Introduction

This is my first attempt at open sourcing some written notes using github and Creative Commons like what my friend Robin Lovelace does. I am collecting all my little text files and code snippits from over the years of working with Linux graphics, sound and video into one place and thought I might as well share them in case they are useful to anyone. As I just finished writing my Springer book "Data Science for Transport" it's easy to reuse the book template, though this is just intended as a loose collection of stuff, not an actual book itself.

The idea is to select only the best of breed of everything and present them together. It's like a Linux distribution in making these selections. But it is a distribution of ideas and choices rather than of actual software. (unless we do a Docker as well). As such, the choice of what to leave out and not cover is also important. The idea is to present a single, best, toolkit, of tools which work and which also work well with one another as in a distro.

It is not a detailed tech manual. The purpose is to present the best tools for media tasks, and to give basic but compilable hello-world examples of code to help get new project started. I use these code snippets all the time when I need to make small new projects based on libraries I might not have used for a while and need to remember how to set them up. After that it's usually easy to consult their big docs on the net to get details for the specific things I need to do.

Maybe one day this might get big enough to make an actual but fully open-source and continually updated commuity book as sometimes printed by `www.oreilly.com/openbook/`. Continual updating would be really important as like a distro all these tool are constantly changing. lyP

The plan is to keep it on github where others can fork it and send back pull requests for incorporation in the main version, both to grow the text and to keep all the software versions up to date. Github also allows readers to edit the text within the github webpage, without having to download to their own machine, this is a really quick and easy way to fix things so please if anyone out there happens to be reading this just go ahead, fork, edit and send a pull request to help keep up to date. If you make useful edits and would like to add your name to the author list then please ask and I will add it.

1.1. Known similar documents

- Pakt 2017 linux sound book, `www.safaribooksonline.com/library/view/linux-sound-programming/9781484224960/` - `en.wikibooks.org/wiki/Configuring_Sound_on_Linux`

1.2. C with CMake

We need to use C so we can see the bits and bytes and understand what's going on. Many of the tools have Python and other wrappers which are great to use later, but only C lets us see the actual data representations which are important in understanding multimedia in detail. It's usually best to learn the C version first then switch to a Python or other language wrapper later if needed.

```
// A simple program that computes the square root of a number
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main (int argc, char *argv[])
{
  if (argc < 2)
    {
    fprintf(stdout,"Usage: %s number\n",argv[0]);
    return 1;
    }
  double inputValue = atof(argv[1]);
  double outputValue = sqrt(inputValue);
  fprintf(stdout,"The square root of %g is %g\n",
          inputValue, outputValue);
  return 0;
}
```

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
add_executable(Tutorial tutorial.cpp)
```

CMake is the best modern build system for C++ on Linux and also other platforms (the C stands for "cross-platform). To use it, first write a program and cmake file as above. Then run,

```
cmake .
make
```

Here, the cmake step searches for all libraries and tools needed for compilation, while make runs compilation of each file and links there results to one another and to the libraries.

(History: make is a lower-level build system. There was once a time when people wrote the "Makefile" generated by CMake by hand. Then there was a time when GNU Autotools were used like CMake is used now.)

CMake lets us specify what libraries, and what versions of them, are used by what object and executable files. The names of stadnard libraries are stored in (/usr/share/cmake-3.5/Modules/*.cmake) with links to their binary object and text header files, for different versions. CMake ships with list of well-known ones and their typical isntall locations to search for mainstream linux distributions. If you need a library not on the list (such as your own) then you need to edit CMake's location lists first.

Part I.

Graphics

2. Graphics architecture

2.1. History

In the old days, (e.g. 1980s), graphics were simple. An area of memory was allocated to represent the array of pixels on the screen. User programs would write to it like any other part of memory. Then a graphics chip would read from it and turn the data into CRT scanning commands to send to the monitor.

In the 2000s, in addition to memory mapping (frame buffers), optional plug-in GPUs sat on the system bus as IO modules and drew graphics in response to commands such as OpenGL or DirectX sent to them via the system bus. Hence one would buy a graphics card labelled as having these functions. Today this is no longer the case. The reason for this was that OpenGL etc rapidly gained many extension commands in later version, and hardware makers struggled to keep up designign new hardware to implement them. They they began to open up "shader languages" to enable these commands to be done in software. Hackers then started using shader languages for non-graphical computing, which led to the present GPU-computing architectures.

2.2. Modern linux graphics stack

Today the situtation is a lot more complex and priobably only a handful of people in the world now understand the whole of the modern graphics stack. APIs inclusing OpenGL, DirectX, and others, are implemented in software libraries, and are translated into lower-level GPU architecture commands. This stack is made especially complicated by the struggle between open-source architecture on which we will focus, and propriatory competitors. As modern graphics cards are made only by propriatory comopanies, they may keep some hardware infromation secret or difficult to obtain, which gives them a competitiative advantage in providing some of these software components. As a result, and together with current interest in GPU computing and reuse and extension of graphics APIs for pure computation, the whole graphics stack is changing very quickly.

Graphics cards sit on the system bus as IO modules. Importantly, they can use DMA (direct memory access). For example, an image can be placed in regular RAM, then a single command given to the GPU to load it from main RAM into the GPU. This copy does not go via the CPU, it goes via DMA, so from the CPU's point of view is almost instant. (It will, however, slow if the bus is needed for other thigns, such as additional DMAs froma webcam into the main RAM.)

2.2.1. Mesa stack

The basic design of all modern graphics software stacks is built around a single kernel module (Direct Renderign Manager, DRM) thourgh which all program-GPU communication is routed. This module receives, buffers and passes commands to the GPU via the system bus. Its main function is to perform security checks on these commands and to buffer them. It registers a single "master" user program (usually the window manager), then all other programs wanting to send commands (e.g. windowed OpenGL programs) must get permission from the master.

The commands are then sent on the system bus as data to addressed to the GPU, which is memory-mapped as an IO module. Unlike 1980s systems, we send commands as data, not raw memory-mapped pixels. Unlike 2000s OpenGL commands, the commands sent on the bus are from the GPU's instruction set architecture (ISA), which for NVida is one of the Fermi/Maxwell/Pascal/Volta series of ISAs. (These are alphabetic ordered. Apart from the original/oldest "Tesla" architecture. NVidia has confusingly now reused the "Tesla" name as a brand for its current line of HPC market products, which currently contain the Pascal architecture; alongside its gamer brand "GeForce", mobile SoC range "Tegra", and its design professional brand "Quadro". AMD's GPU ISAs are called SeaIslands, VolcanicIslands, SouthernIslands and R600 documented at `https://www.phoronix.com/scan.php?page=article&item=amd_r600_700_guide&num=1);KhronosSIPRisaproposedopenGPUISA.` The DRM may be open source or propriatory. The DRM is always coupled tightly to some user-space library which passes commands to the DRM inside the kernel. Both the input to this joint system and the output are dependent on the type of GPU hardware used, they do not present a single standard interface. (This seems quite a messy design - but appears unavoidable because fundamentally we want to expose very low level hardware having different capabilities to very high level programs.) The DRM presents an interface to each GPU as a unix file such as /dev/dri/cardX. The user space library then opens and writes to this file using ioctl commands.

The open-source project which maintains most of this stack is called Mesa. Mesa is not a single system or component but a large family of components. These exist at different levels of the stack, and there are many alternative Mesa implementations for many components which are specific to certain makes of GPU, or which implement higher-level systems in different and sometimes experimental ways. Nouveau is Mesa's DRM for NVidia GPUs. (It is made by reverse engineering nvidiadrm with some help from NVidia staff. nouveaux includes letters NV for NVidia). Mesa also includes many libDRM user-space interfaces to Nouveau and its other kernel modules for other graphics card types. Each libDRM has a different API, as it represents a different card's capabilities. There is no standard here. (Therefore, anything calling libDRM must also exist in different versions for different cards.) (Note that the graphics stack breaks the general rule of API design, that an interface at one level is independent of the implementation at the level below it. This is because different graphics cards have differerent capabilities which must be exposed, quickly and efficiently, to very high level user programs.)

Different user-space graphics libraries then call libDRM. These typically implement a standard, programmer-facing API, including OpenGL, DirectX, the new Vulcan, and also the X windowing commands and GPU computation APIs such as OpenCL and CUDA. Again, these modules are implemented for specific graphics cards, as they send specific commands to the libDRM modules for the specific card. These modules are part of the mesa project and have names like "mesa-opengl-neuveau" – which means Mesa's implementation of the OpenGL API for the neauveau kernem module and its matchign libDRM library. (Some non-card-specific code can be shared between these modules; the new Gallium3D architecture defines an internal separation within them, with standard internal APIs to allow this.)

All of the above process can be implemented using different interfaces between the DRM, libDRM and user-space library implementation. It is hard to find exact details. AFAIK the user-space library implementation (eg mesa-opengl-neuveau) translates the GL call into NVidia Tesla insrtuctins, and passes those instructions to libDRM and to the kernel module. Then libDRM and the kernel module deal with buffering them and putting them on the bus. [TODO prove this?]

(To restate this: the GPU does not itself understand OpenGL, DirectX or CUDA. These are highevel user APIs. The GPU itself understands from a GPU ISA such as Nvidia Pascal. Hence, one does not buy an OpenGL or DirectX card, one buys an NVidia Pascal card, then obtains libraries to translate OpenGL or DirectX to Nvidia Pascal.)

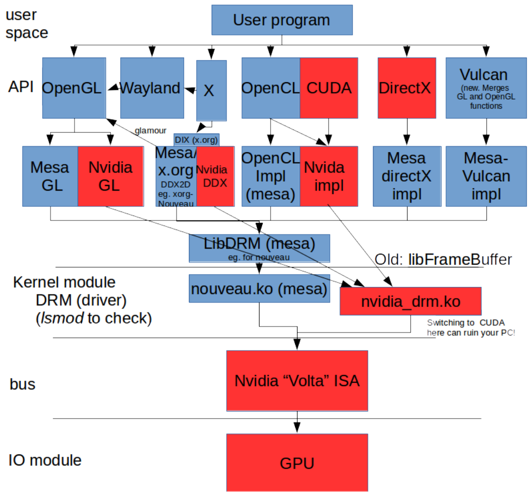
2.2.2. Windowing systems

X is an API, not an implementation. Very old-fashioned X windowing implementations (such as XFree86) used to translaste X calls into memory-mapped pixels. More recent implementations (such as not quite up to date x.org) used to translate them into libDRM commands. Modern impementations (i.e. x.org's latest GLAMOUR) translate them into OpenGL calls then pass them to the OpenGL system. (This is also the case for the new Wayland replacement for X, and for Wayland's X emulation layer.) So today everything goes through mesa-opengl-neuveau, lib-drm-neuvueu, and neuveua then on the bus and into the GPU. Even a full-screen game is likely to run inside a large screen-covering window as part of this system.

In practice, this means that a (compositing) window manager will provide a memory-mapped framebuffer area in main RAM for the user program to draw on, like in the old days. It will then send a small texture command to the GPU telling it to copy this as a texture into GPU memory using fast DMA. Modern window managers can very easily and trivcially run special 3D effects such as rotating the desktop around a 3D cube, because of this GL-based structure. In fact most desktop-users are seriously wasting the capabilities of their GPU by only using it to render 2D desktops. VR / Augmented reality type desktops would be very easy to render with little or no extra overhead, perhaps this will happen soon?

In most modern implementations, a window set up to render OpenGL graphcis will bypass the above and will send GL requests directly, not through the compositing window manager's framebuffer. Such implementations are found in the SDL/GLU/GLX layers.

Figure 2.1.:



2.2.3. Propriatory NVidia stack

Little public information is available on these and in general we try to avoid them as open-source programmers. The one time it is unformtatnely still necessary to use them is if we want to use NVifia's propriatory CUDA language for GPU programming (rather than the better OpenCL alternative). This occurs if we want to do deep learning with NVidia's own easy-to-install DNN tools, if we are not clever enough to install OpenCL based alternative stacks. NVidia's own system is a little different from Mesa's because it lacks the libDRM layer. NVidia instead provides its own binary userspace implementations of the standard APIs – including CUDA but also X and OpenGL – which talk directly to its propriatory binary kernel module. It is not known how them communicate. A downside of this setup is that if we want to run CUDA then we have to replace our entire stack – including switchign to NVidia's priopriatory implementation of X windows and OpenGL at the same time, which may lead to conflicts with other software which needs the Mesa versions. This is an extremely aggressive business move by NVidia, saying you can only use their DNN tools if you agree to replace your whoel desktop windowing suystem with their version of everything, and is a major driver for the push to swap the backends of DNN tools such as Keras and TensorFlow with OpenCL versions.

2.3. Further reading

`https://people.freedesktop.org/~marcheu/linuxgraphicsdrivers.pdf`
`https://blogs.igalia.com/itoral/2014/07/29/a-brief-introduction-to-the-linux-graphics-s`

3. OpenGL (via SDL)

OpenGL is the industry standard 3D graphics command language. It provides an API containing commands which draw triangles and linesin 3D space, to render them under different lighting models, and to position the camera geometry around them.

In theory, the OpenGL API can be implemented in all kinds of ways, including fast graphics card hardware to take these commands and render them at blazing speed directly to a monitor. But also, for example, implementations might render drawings from the same commands onto bitmap images, vector graphics canvases, or even to robic spraycan or oil paintbrush manipulators.

Hence, to use the OpenGL requires an implementation library, such as Mesa, and usually a second library which links it to the screen or to a window in the operating system. We will made use of the SDL (Simple Direct Layer) library as this link here. SDL provides a graphics context which may be full-screen (eg for writing games) or windowed within the operating system.

3.0.1. SDL

As modern systems operate from within a desktop window manager, which often gets implemented through OpenGL itself, some method is needed to prepare part of the screen and/or a window in the windowing system for the programmer's own OpenGL commands to run. This is called a GL context. It is provided by libraries such as SDL. The programmer cannot give OpenGL commands directly because the window manager has already bagged the status of "master" of libDRM, and has instructed it not to take commands from anyone else. SDL is given special permission by the window manager to pass its own commands (via the OpenGL implementations) to libDRM. libDRM will then see that they coem from an allowed source and let them through to the kernel and GPU.

Simple DirectMedia Layer (SDL) is a cross-platform software development library designed to provide a hardware abstraction layer for computer multimedia hardware components including OpenGL graphics cards, and also keyboards and joysticks. It is used in 3D games including 0AD, FreeCiv, Oolite, and in 2D games such as Secret Maryo Chnonicles and many others in Humble Bundles. (Traditionally, a different link library, GLUT, was used. GLUT's programming model requires it to take full control of your program and communicate only through callbacks, while SDL keeps the user in control and assumes they will call its functions regularly. We consider the GLLUT model to be "rude" in takign over control, and this may conflict with other tools which also ask for control, such as ROS. GLUT is considered old and dying. Other altneratives include GLFW and pyglet for Python).

```
#include <SDL/SDL.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdio.h>
#include <stdlib.h>

static GLboolean should_rotate = GL_TRUE;
static void quit_tutorial( int code )
{
    SDL_Quit( );
    exit( code );
}
static void handle_key_down( SDL_keysym* keysym )
{
    switch( keysym->sym ) {
    case SDLK_ESCAPE:
        quit_tutorial( 0 );
        break;
    case SDLK_SPACE:
        should_rotate = !should_rotate;
        break;
    default:
        break;
    }

}
static void process_events( void )
{
    SDL_Event event;
    while( SDL_PollEvent( &event ) ) {
        switch( event.type ) {
        case SDL_KEYDOWN:
            handle_key_down( &event.key.keysym );
            break;
        case SDL_QUIT:
            quit_tutorial( 0 );
            break;
        }
    }
}
static void draw_screen( void )
{
    static float angle = 0.0f;
    static GLfloat v0[] = { -1.0f, -1.0f,  1.0f };
```

```
static GLfloat v1[] = {  1.0f, -1.0f,  1.0f };
static GLfloat v2[] = {  1.0f,  1.0f,  1.0f };
static GLfloat v3[] = { -1.0f,  1.0f,  1.0f };
static GLfloat v4[] = { -1.0f, -1.0f, -1.0f };
static GLfloat v5[] = {  1.0f, -1.0f, -1.0f };
static GLfloat v6[] = {  1.0f,  1.0f, -1.0f };
static GLfloat v7[] = { -1.0f,  1.0f, -1.0f };
static GLubyte red[]    = { 255,   0,   0, 255 };
static GLubyte green[]  = {   0, 255,   0, 255 };
static GLubyte blue[]   = {   0,   0, 255, 255 };
static GLubyte white[]  = { 255, 255, 255, 255 };
static GLubyte yellow[] = {   0, 255, 255, 255 };
static GLubyte black[]  = {   0,   0,   0, 255 };
static GLubyte orange[] = { 255, 255,   0, 255 };
static GLubyte purple[] = { 255,   0, 255,   0 };

glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
glTranslatef( 0.0, 0.0, -5.0 );
glRotatef( angle, 0.0, 1.0, 0.0 );

if( should_rotate ) {

    if( ++angle > 360.0f ) {
        angle = 0.0f;
    }

}
/* Send our triangle data to the pipeline. */
glBegin( GL_TRIANGLES );

glColor4ubv( red );
glVertex3fv( v0 );
glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( blue );
glVertex3fv( v2 );

glColor4ubv( red );
glVertex3fv( v0 );
glColor4ubv( blue );
glVertex3fv( v2 );
glColor4ubv( white );
```

```
glVertex3fv( v3 );

glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( black );
glVertex3fv( v5 );
glColor4ubv( orange );
glVertex3fv( v6 );

glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( orange );
glVertex3fv( v6 );
glColor4ubv( blue );
glVertex3fv( v2 );

glColor4ubv( black );
glVertex3fv( v5 );
glColor4ubv( yellow );
glVertex3fv( v4 );
glColor4ubv( purple );
glVertex3fv( v7 );

glColor4ubv( black );
glVertex3fv( v5 );
glColor4ubv( purple );
glVertex3fv( v7 );
glColor4ubv( orange );
glVertex3fv( v6 );

glColor4ubv( yellow );
glVertex3fv( v4 );
glColor4ubv( red );
glVertex3fv( v0 );
glColor4ubv( white );
glVertex3fv( v3 );

glColor4ubv( yellow );
glVertex3fv( v4 );
glColor4ubv( white );
glVertex3fv( v3 );
glColor4ubv( purple );
glVertex3fv( v7 );
```

```
glColor4ubv( white );
glVertex3fv( v3 );
glColor4ubv( blue );
glVertex3fv( v2 );
glColor4ubv( orange );
glVertex3fv( v6 );

glColor4ubv( white );
glVertex3fv( v3 );
glColor4ubv( orange );
glVertex3fv( v6 );
glColor4ubv( purple );
glVertex3fv( v7 );

glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( red );
glVertex3fv( v0 );
glColor4ubv( yellow );
glVertex3fv( v4 );

glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( yellow );
glVertex3fv( v4 );
glColor4ubv( black );
glVertex3fv( v5 );

glEnd( );

/*
 * EXERCISE:
 * Draw text telling the user that 'Spc'
 * pauses the rotation and 'Esc' quits.
 * Do it using vetors and textured quads.
 */

/*
 * Swap the buffers. This this tells the driver to
 * render the next frame from the contents of the
 * back-buffer, and to set all rendering operations
 * to occur on what was the front-buffer.
 *
 * Double buffering prevents nasty visual tearing
```

```
     * from the application drawing on areas of the
     * screen that are being updated at the same time.
     */
    SDL_GL_SwapBuffers( );
}

static void setup_opengl( int width, int height )
{
    float ratio = (float) width / (float) height;

    glShadeModel( GL_SMOOTH );
    glCullFace( GL_BACK );
    glFrontFace( GL_CCW );
    glEnable( GL_CULL_FACE );
    glClearColor( 0, 0, 0, 0 );
    glViewport( 0, 0, width, height );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity( );
    gluPerspective( 60.0, ratio, 1.0, 1024.0 );
}

int main( int argc, char* argv[] )
{
    /* Information about the current video settings. */
    const SDL_VideoInfo* info = NULL;
    /* Dimensions of our window. */
    int width = 0;
    int height = 0;
    /* Color depth in bits of our window. */
    int bpp = 0;
    /* Flags we will pass into SDL_SetVideoMode. */
    int flags = 0;

    /* First, initialize SDL's video subsystem. */
    if( SDL_Init( SDL_INIT_VIDEO ) < 0 ) {
        /* Failed, exit. */
        fprintf( stderr, "Video initialization failed: %s\n",
             SDL_GetError( ) );
        quit_tutorial( 1 );
    }
    /* Let's get some video information. */
    info = SDL_GetVideoInfo( );
    if( !info ) {
        /* This should probably never happen. */
```

```
    fprintf( stderr, "Video query failed: %s\n",
         SDL_GetError( ) );
    quit_tutorial( 1 );
}
/*
 * Set our width/height to 640/480 (you would
 * of course let the user decide this in a normal
 * app). We get the bpp we will request from
 * the display. On X11, VidMode can't change
 * resolution, so this is probably being overly
 * safe. Under Win32, ChangeDisplaySettings
 * can change the bpp.
 */
width = 640;
height = 480;
bpp = info->vfmt->BitsPerPixel;

/*
 * Now, we want to setup our requested
 * window attributes for our OpenGL window.
 * We want *at least* 5 bits of red, green
 * and blue. We also want at least a 16-bit
 * depth buffer.
 *
 * The last thing we do is request a double
 * buffered window. '1' turns on double
 * buffering, '0' turns it off.
 *
 * Note that we do not use SDL_DOUBLEBUF in
 * the flags to SDL_SetVideoMode. That does
 * not affect the GL attribute state, only
 * the standard 2D blitting setup.
 */
SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
flags = SDL_OPENGL;

if( SDL_SetVideoMode( width, height, bpp, flags ) == 0 ) {
    fprintf( stderr, "Video mode set failed: %s\n",
         SDL_GetError( ) );
    quit_tutorial( 1 );
```

```
    }
    setup_opengl( width, height );

    while( 1 ) {
        process_events( );
        draw_screen( );
    }
    return 0;
}
```

```
SET(CMAKE_CXX_FLAGS "-std=c++11")
cmake_minimum_required(VERSION 2.8)
project( videoWrite )
find_package( OpenCV REQUIRED )
add_executable( sdlgl sdlgl.cpp )
target_link_libraries( sdlgl GL GLU SDL  )
```

3.0.2. GL/SDL Textures

GL textures are simple bitmap images, ie. arrays of raw data in a specified format. (Such as 32bit RBGA color bytes). SDL wraps these in a lihtweight struct which adds size and format infromation to the raw data. The raw data can still be accessed and passed to GL as an element in the struct (http://sdl.beuc.net/sdl.wiki/SDL$surface$) :

```
//sudo apt-get install libsdl-image1.2-dev
//g++ test.cpp  -L/usr/lib/x86_64-linux-gnu  -lGLU -lGL -lSDL -lSDL_image

#include "SDL/SDL.h"
   #include "SDL/SDL_opengl.h"
   #include "SDL/SDL_image.h"

    const int SCREEN_WIDTH = 640;
    const int SCREEN_HEIGHT = 480;
    const int SCREEN_BPP = 32;

    int tex;

    int loadTexture(char* fileName){

        SDL_Surface *image=IMG_Load(fileName);
        SDL_DisplayFormatAlpha(image);

        GLuint object;
        glGenTextures(1,&object);
```

```
    glBindTexture(GL_TEXTURE_2D, object);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP_TO_EDGE);
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,image->w,image->h,0,GL_RGBA,GL_UN

    SDL_FreeSurface(image);

    return object;
}
void init(){
    glClearColor(0.0,0.0,0.0,0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0,800,600,1.0,-1.0,1.0);
    glEnable(GL_BLEND);
    glEnable(GL_TEXTURE_2D);
    glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
    tex = loadTexture("hi.png");
}
void draw(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBindTexture(GL_TEXTURE_2D,tex);
    glBegin(GL_QUADS);
        glTexCoord2f(0,0);
        glVertex2f(0,0);
        glTexCoord2f(1,0);
        glVertex2f(500,0);
        glTexCoord2f(1,1);
        glVertex2f(500,500);
        glTexCoord2f(0,1);
        glVertex2f(0,500);
    glEnd();
    glFlush();
}
int main(int argc,char** argv){
    SDL_Init(SDL_INIT_EVERYTHING);
    SDL_Surface* screen=SDL_SetVideoMode(800,600,32,SDL_SWSURFACE|SDL_OPEN
    bool running=true;
    Uint32 start;
    SDL_Event event;
    init();
    while(running){
```

```
            start=SDL_GetTicks();
            draw();
            while(SDL_PollEvent(&event)){
                switch(event.type){
                    case SDL_QUIT:
                        running=false;
                        break;
                }
            }
            SDL_GL_SwapBuffers();
            if(1000/60>(SDL_GetTicks()-start))
                SDL_Delay(1000/60-(SDL_GetTicks()-start));
        }
        SDL_Quit();
        return 0;
    }
```

3.0.3. Video textures with CV/GL/SDL

Suppose we are making a 3D video game set in a city and want to render a video of the Joker's threat to Gotham City on a giant screen on the side of a skyscraper. Here we read frames usign OpenCV and push them into GL textures in real time. This method is also useful fo AR if we want to just render a flat video image in GL – it is done in exactly the saem way. It is fast because the texture transfer is done via DMA on the GL command, so doesn't take up CPU time.

```
//render an openCV webcam stream into a 3d openGL object texture

//g++ -std=c++11 -I/opt/ros/kinetic/include/opencv-3.3.1/   test.cpp
/opt/ros/kinetic/lib/libopencv_*.so -L/usr/lib/x86_64-linux-gnu
-lGLU -lGL -lSDL -lSDL_image

#include "opencv2/opencv.hpp"
#include <iostream>
#include <string>
#include "SDL/SDL.h"
#include "SDL/SDL_opengl.h"
#include "SDL/SDL_image.h"

using namespace std;
using namespace cv;

const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
```

```
const int SCREEN_BPP = 32;

int tex;
VideoCapture cap(0);
Mat frame;

void init(){
                                        glClearColor(0.0,0.0,0.0,0.0);
                                        glMatrixMode(GL_PROJECTION);
                                        glLoadIdentity();
                                        glOrtho(0.0,800,600,1.0,-1.0,1.0);
                                        glEnable(GL_BLEND);
                                        glEnable(GL_TEXTURE_2D);
                                        glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPH
}

void draw(){
                    bool b = cap.read(frame);
                    imshow("foo2", frame);
                    waitKey(30);

                    GLuint object;
                    glGenTextures(2,&object);
                    glBindTexture(GL_TEXTURE_2D, object);
                    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST
                    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST
                    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_ED
                    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP_TO_ED

                    glTexImage2D(GL_TEXTURE_2D,0,3,frame.cols,frame.rows,0,GL_BGR,

                    glClear(GL_COLOR_BUFFER_BIT);
                    glBindTexture(GL_TEXTURE_2D, object);

                    glBegin(GL_QUADS);
                    glTexCoord2f(0,0);
                    glVertex2f(0,0);
                    glTexCoord2f(1,0);
                    glVertex2f(500,0);
                    glTexCoord2f(1,1);
                    glVertex2f(500,500);
                    glTexCoord2f(0,1);
                    glVertex2f(0,500);
                    glEnd();
```

```
        glFlush();
}
int main(int argc,char** argv){
        SDL_Init(SDL_INIT_EVERYTHING);
        SDL_Surface* screen=SDL_SetVideoMode(800,600,32,SDL_SWSURFACE|
        bool running=true;
        Uint32 start;
        SDL_Event event;
        init();
        if(!cap.isOpened())  // check if we succeeded
        {
                cout << "outch" << endl;
                return -1;
        }
        namedWindow("edges",1);
        while(running)
        {
                start=SDL_GetTicks();
                draw();
                while(SDL_PollEvent(&event)){
                        switch(event.type){
                                case SDL_QUIT:


                        }
                }
                SDL_GL_SwapBuffers();
                if(1000/60>(SDL_GetTicks()-start))
                        SDL_Delay(1000/60-(SDL_GetTick
        }
        SDL_Quit();
        return 0;
}
```

3.0.4. GL animation

Once you have a context set up, you can apply any GL commands. The classic tutorial on pure GL programming is NeHe's website. The classic reference is the OpenGL Red Book.

How OpenGL works intenrnall: graphics pipeline: https://fgiesen.wordpress.com/2011/07/01/a-trip-through-the-graphics-pipeline-2011-part-1/

4. SDL 2D graphics and input

eg for 2d platform games we are using SDL1.2 (there is newer 2.0 now)

lazyfoo.net

Here is how to blit an image,

4.1. Combining 2D and 3D graphics in SDL

eg for Augmented Reality overlay! blit + GL done by blitting to texture ?

5. Game Engines

5.1. 2D

pygame - built on SDL and OpenAL. 2D scene graph. Collision detection. Partial android version.

5.2. 3D

Panda3D - unity-like - developed by Disney? Blender game engine

6. Cairo vector graphics

also Pango font rendering ?

7. CAD

FreeCAD Blender

7.1. Collada (dae) format

7.2. OpenSceneGraph (uses dae)

B+trees (from GIS book?) - for collision detection

7.3. ODE physics engine?

8. OpenCV

8.1. Reading and writing

```
#include <iostream> // for standard I/O
#include <string>   // for strings
#include <opencv2/core/core.hpp>        // Basic OpenCV structures (cv::Mat)
#include <opencv2/highgui/highgui.hpp>  // Video write
#include "opencv2/opencv.hpp"
using namespace std;
using namespace cv;

int main()
{
        VideoWriter outputVideo; // For writing the video
        int width = 640; // Declare width here
        int height = 480; // Declare height here
        Size S = Size(width, height); // Declare Size structure
        const string filename = "bar.avi"; // Declare name of file here
        int fourcc = CV_FOURCC('M','J','P','G');
        int fps = 10;
        outputVideo.open(filename, fourcc, fps, S);

  //if ogg bug here, do
  //    mencoder foo.mp4 -ovc lavc -oac mp3lame -o foo.avi
   //and try again in avi -- working
  //VideoCapture cap("/home/charles/foo.avi");
        VideoCapture cap(0);    // dev/video0 webcam; or use a regular filename
        if(!cap.isOpened())  // check if we succeeded
        {
        cout << "couldnt open video input" << endl;
        return -1;
        }
        cout << "loaded video" << endl;

        namedWindow("edges",1);
        for(;;)
        {
```

```
            Mat frame;
        bool b;
        b = cap.read(frame);  //this also advances the frame
            imshow("videoWrite", frame);
            if(waitKey(30) >= 0)
            break;
    }
        return 0;
}
```

```
SET(CMAKE_CXX_FLAGS "-std=c++11")

cmake_minimum_required(VERSION 2.8)
project( videoWrite )
find_package( OpenCV REQUIRED )
add_executable( videoWrite videoWrite.cpp )
target_link_libraries( videoWrite ${OpenCV_LIBS} )
```

8.2. Basic manipulations

9. Files and formats

8 bit colouri = 256 color pallette. Usually with the palette defined in 24 bit in a header. (+ Old VGA has a fixed standard palette) Standard 24 bit color = 1 byte for each of R,G,B. 32bit adds alpha byte too. Nice to see and work with, human-readable in hex.

BGR (and ABRG) format for historical reasons. Used by GPU hardware, so libraries like CV follow it for speed.

Various color depths.

"convert" command - very versatile. eg. ps to png.

9.1. Data formats

9.1.1. Bitmap (BMP)

Windows standard. Header then raw RGB array data.

9.2. ROS Image message

header includes timestamps etc as well as img size and depth.

9.3. Portable Network Graphics (PNG)

compressed, like JPG. File made of chunks of labelled types.

9.4. Postscript vector graphics

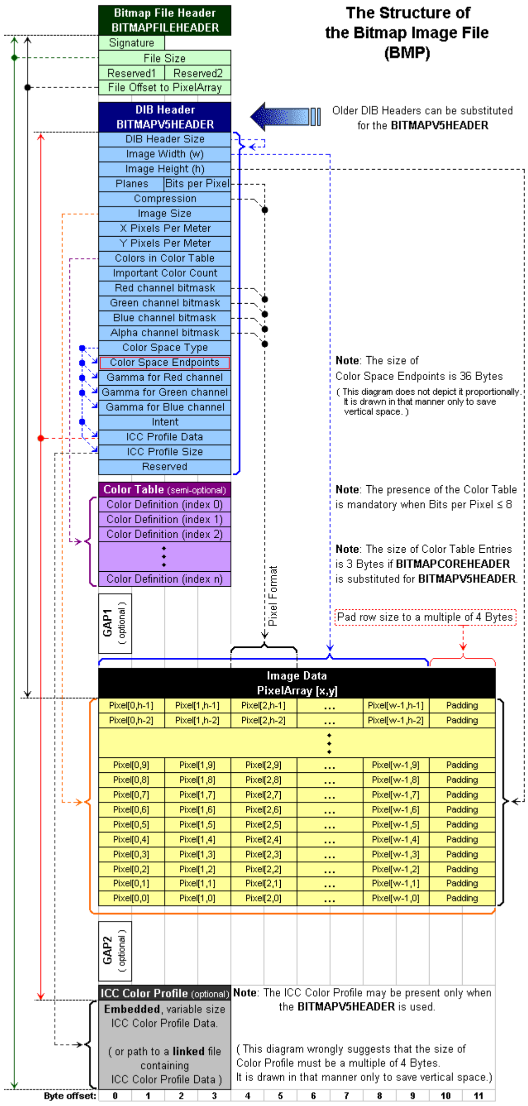
as programming language. as used in IOX.

9.5. Portable Document Fomat (pdf)

```
extract pdf pages:
 pdftk A=etc.pdf cat A1 A2    A6 A7 A8  A9  A11 A12 A13  A14 A15
A19-25 A26 A27 A29 A28 A30 A31 A33    output out.pdf

Convert pdf to multiple png pages:
pdftoppm -rx 80 -ry 80  -png CAV3\ Consortium\ Agreement\ -\ 17.4.18\ Clean\ \
```

Figure 9.1.:


The Structure of the Bitmap Image File (BMP)
Bitmap File Header BITMAPFILEHEADER
Signature
File Size
Reserved1
Reserved2
File Offset to PixelArray
DIB Header BITMAPV5HEADER
Older DIB Headers can be substituted for the BITMAPV5HEADER
DIB Header Size
Image Width (w)
Image Height (h)
Planes
Bits per Pixel
Compression
Image Size
X Pixels Per Meter
Y Pixels Per Meter
Colors in Color Table
Important Color Count
Red channel bitmask
Green channel bitmask
Blue channel bitmask
Alpha channel bitmask
Color Space Type
Color Space Endpoints
Gamma for Red channel
Gamma for Green channel
Gamma for Blue channel
Intent
ICC Profile Data
ICC Profile Size
Reserved
Note: The size of Color Space Endpoints is 36 Bytes
(This diagram does not depict it proportionally. It is drawn in that manner only to save vertical space.)
Color Table (semi-optional)
Color Definition (index 0)
Color Definition (index 1)
Color Definition (index 2)
Color Definition (index n)
Note: The presence of the Color Table is mandatory when Bits per Pixel ≤ 8
Note: The size of Color Table Entries is 3 Bytes if BITMAPCOREHEADER is substituted for BITMAPV5HEADER
GAP1 (optional)
Pixel Format
Pad row size to a multiple of 4 Bytes
Image Data PixelArray [x,y]
Pixel[0,h-1]
Pixel[1,h-1]
Pixel[2,h-1]
Pixel[w-1,h-1]
Padding
Pixel[0,0]
Pixel[1,0]
Pixel[2,0]
Pixel[w-1,0]
GAP2 (optional)
ICC Color Profile (optional)
Embedded, variable size ICC Color Profile Data
(or path to a linked file containing ICC Color Profile Data)
Note: The ICC Color Profile may be present only when the BITMAPV5HEADER is used.
(This diagram wrongly suggests that the size of Color Profile must be a multiple of 4 Bytes. It is drawn in that manner only to save vertical space.)
Byte offset: 0 1 2 3 4 5 6 7 8 9 10 11

```
Create a pdf from multiple png pages:
convert *.png out.pdf

Merge pdf files:
pdfunite innput1.pdf input2.pdf input3.pdf out.pdf
```

9.6. Fonts

Copy the .ttf file and paste it inside /.fonts folder, ie /home/username/.fonts folder. Create one if you dont already have one.

Now run

sudo fc-cache -fv

10. GPU OpenCL as graphics programming?

10.1. Architecture

10.2. Low level GPU ISA programming

It is very rare to program GPUs directly in their own ISA. Probably the only people who do this are staff at NVidia and staff and volunteers at Mesa who write the implementations for OpenGL and OpenCL etc.

An example program (for an AMD card):

```
00 ALU: ADDR(32) CNT(4) KCACHE0(CB0:0-15)
0 x: MUL R0.x, KC0[0].x, KC0[1].x
y: MUL R0.y, KC0[0].y, KC0[1].y
1 z: MUL R0.z, KC0[0].z, KC0[1].z
w: MUL R0.w, KC0[0].w, KC0[1].w
01 EXP_DONE: PIX0, R0
END_OF_PROGRAM
```

from: https://stackoverflow.com/questions/27733704/how-is-webgl-or-cuda-code-actually-translated-into-gpu-instructions

to pass GPU binary via a CL function: https://www.khronos.org/registry/OpenCL/sdk/1.0/docs/man/xl

TODO how to call libDRM directly with the commands ?

10.3. OpenCL setup

We can choose the Mesa stack or a propriatory (eg. NVidia or Intel) stack, for the particular GPU type in our computer.

The Ubuntu nvidia stack can be installed via the Restricted repository, eg. nvidia-opencl-dev gets everything, and replaces the Mesa stack. Then sudo apt install nvidia-cuda-toolkit to get CUDA and OpenCL APIs, cublas, cudnn, etc.

The mesa stack is harder ... maybe easier in 16.04 ?

```
/usr/lib/x86_64-linux-gnu/libOpenCL.so.1
```

ICD (Installable Client Driver) - Khronos tool to allow multiple implemenations of CL to coexist. mesa-opencv-icd apt package ?

10.4. OpenCL programming

Hello world in OpenCL:

```
#include <stdio.h>
#include <string.h>
#include <CL/cl.h>

//the CLC program as a string (this is not C. It is CLC).
const char rot13_cl[] = "                                   \
__kernel void rot13                                                 \
    (   __global    const   char*   in          \
    ,   __global            char*   out         \
    )
{
    const uint index = get_global_id(0);        \
    char c=in[index];                                   \
    if (c<'A' || c>'z' || (c>'Z' && c<'a')) \
        {
        out[index] = in[index];                     \
    } else
        {
        if (c>'m' || (c>'M' && c<'a'))          \
                {
                out[index] = in[index]-13;          \
                } else
                {
                    out[index] = in[index]+13;      \
                }
    }
}
";

void rot13 (char *buf)
{
    int index=0;
    char c=buf[index];
    while (c!=0) {
        if (c<'A' || c>'z' || (c>'Z' && c<'a')) {
            buf[index] = buf[index];
        } else {
            if (c>'m' || (c>'M' && c<'a')) {
                buf[index] = buf[index]-13;
            } else {
```

```
                buf[index] = buf[index]+13;
            }
        }
        c=buf[++index];
    }
}

int main() {
        char buf[]="Hello, World!";
        size_t srcsize, worksize=strlen(buf);

        cl_int error;
        cl_platform_id platform;
        cl_device_id device;
        cl_uint platforms, devices;

        error=clGetPlatformIDs(1, &platform, &platforms);
        error=clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, &device, &device
        cl_context_properties properties[]={
                CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
                0};
        cl_context context=clCreateContext(properties, 1, &device, NULL, NULL,
        cl_command_queue cq = clCreateCommandQueue(context, device, 0, &error)

        rot13(buf);     // scramble using the CPU
        puts(buf);      // Just to demonstrate the plaintext is destroyed

        const char *src=rot13_cl;
        srcsize=strlen(rot13_cl);

        const char *srcptr[]={src};
        cl_program prog=clCreateProgramWithSource(context,
                1, srcptr, &srcsize, &error);
        error=clBuildProgram(prog, 0, NULL, "", NULL, NULL);  //compile

        cl_mem mem1, mem2;  // Allocate memory for the kernel to work with
        mem1=clCreateBuffer(context, CL_MEM_READ_ONLY, worksize, NULL, &error)
        mem2=clCreateBuffer(context, CL_MEM_WRITE_ONLY, worksize, NULL, &error

        // get a handle and map parameters for the kernel
        cl_kernel k_rot13=clCreateKernel(prog, "rot13", &error);
        clSetKernelArg(k_rot13, 0, sizeof(mem1), &mem1);
        clSetKernelArg(k_rot13, 1, sizeof(mem2), &mem2);
```

```
        // Target buffer just so we show we got the data from OpenCL
        char buf2[sizeof buf];
        buf2[0]='?';
        buf2[worksize]=0;

        // Send input data to OpenCL (async, don't alter the buffer!)
        error=clEnqueueWriteBuffer(cq, mem1, CL_FALSE, 0, worksize, buf, 0, NU
        // Perform the operation
        error=clEnqueueNDRangeKernel(cq, k_rot13, 1, NULL, &worksize, &worksiz
        // Read the result back into buf2
        error=clEnqueueReadBuffer(cq, mem2, CL_FALSE, 0, worksize, buf2, 0, NU
        error=clFinish(cq); //wait for completion
        puts(buf2); //output the result
}
```

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)


#defines OPENCL_LIBRARIES etc if sucessful
find_package( OpenCL REQUIRED )


add_executable(go go.cpp)

target_link_libraries(go  ${OpenCL_LIBRARIES} )
```

(Cmake requres a manual link of libopencl.so.1 to libopencl.so before finding it).

11. Applications

GIMP
FreeCAD

Part II.

Audio

12. How sound cards work

firewire system (is a bus, chained). FFADO driver (IML setup). USB system (p2p)

13. Linux audio systems

Fig. 13.1 outlines the current Linux audio system and highlights the parts used in our stack. The Linux audio architecture has grown quite complex in recent years, with multiple ways of doing things competing and improving.

Historically, the OSS system was developed for Linux in the early 1990s, focused initially on the Creative SoundBlaster cards then extending to others. It was a locking system which allowed only one program at a time to access the sound card, and lacked support for modern features such as surround sound. It allowed low level access to the card, for example by `cataudio.wav > /dev/dsp0`. The ALSA system was designed as a modern replacement for OSS, and is used on most current distributions including our Ubuntu Studio 11.10. PortAudio is an API with backends that abstract both OSS and ALSA, as well as sound systems of non-free platforms such as Win32 sound and Mac CoreAudio, created to allow portable audio programs to be written. Several software mixer systems were built to resolve the locking problem for consumer-grade applications, including PulseAudio, ESD and aRts. Some of these mixers grew to take advantage of and to control hardware mixing provided by sounds cards, and provided additional features such as network streaming. They provided their own APIs as well as emulation layers for older (or mixer-agnostic) OSS and ALSA applications. (To complicate matters further, recent versions of OSS4 and ALSA have now begun to provide their own software mixers, as well as emulation layers for each other.) Many current Linux distributions including Ubuntu 11.10 deploy PulseAudio running on ALSA, and also include an ALSA emulation layer on Pulse to allow multiple ALSA and Pulse applications to run together through the mixer. Common media libraries such as GStreamer (which powers consumer applications such as VLC, Skype and Flash) and libcanberra (the GNOME desktop sound system) have been developed closely with PulseAudio, increasing its popularity. However, Pulse is not designed for pro-audio work which relies on very low latencies and minimal drop-outs.

The JACK system is an alternative software mixer which fills this need. Like the other soft mixers, JACK runs on many lower level platforms – usually ALSA on modern Linux machines. The bulk of pro-audio applications such as Ardour, zynAddSubFx and qSynth run on JACK. JACK also provides network streaming, and emulations/interfaces for other audio APIs including ALSA, OSS and PulseAudio. (Pulse-on-JACK is useful when using pro and consumer applications at the same time, such as when watching a YouTube tutorial about how to use a pro application. This re-configuration happens automatically when JACK is launched on a modern Pulse machine such as Ubuntu 11.10.)

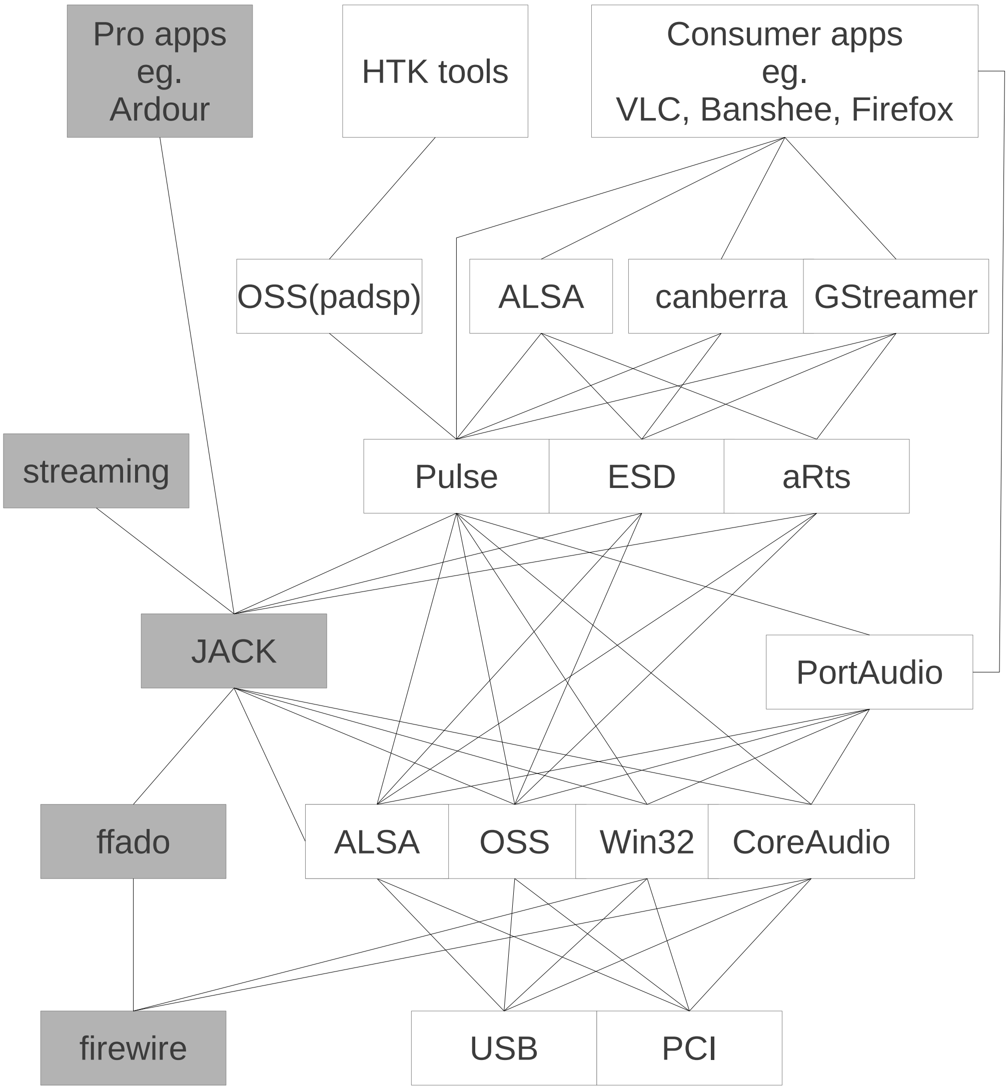


Figure 13.1.: Audio system for recording.

13.1. Example setup

Our DAW is a relatively low-power Intel E8400 (Wolfdale) duo-core, 3GHz, 4Gb Ubuntu Studio 11.10-64-bit machine. Ubuntu studio was installed directly from CD – not added as packages to an existing Ubuntu installation – this gives a more minimalist installation than the latter approach. In particular the window manager defaults to the low-power XFCE, and CPU-hogging programs such as Gnome-Network-Monitor (which periodically searches for new wifi networks in the background) are not installed.

The standard ALSA and OSS provide interfaces to USB and PCI devices below, and to JACK above. However for firewire devices such as our Pre8, the `ffado` driver provides a *direct* interface to JACK from the hardware, bypassing ALSA or OSS. (Though the latest/development version provides an ALSA output layer as well.) Our DAW uses `ffado` with JACK2 (Ubuntu packages: `jack2d, jack2d-firewire, libffado, jackd, laditools`. JACK1 is the older but perhaps more stable single-processor implementation of the JACK API) and fig. 13.2 shows our JACK settings, in the `qjackctl` tool. Note that the firewire backend driver (ie. `ffado`) is selected rather than ALSA.

It is important to unlock memory for good JACK performance. As well as ticking the unlock memory option, the user must also be allowed to use it, eg. `adduser charles audio`. Also the file `/etc/security/limits.d/audio.conf` was edited (followed by a reboot) to include

```
@audio - rtprio 95
@audio - memlock unlimited
```

These settings can be checked by

`ulimit -r -l`.

The JACK sample rate was set to 48kHz, matching the Pre8s. (This is a good sample rate for speech research work as it is similar to CD quality but allows simple sub-sampling to power-of-two frequencies used in analysis.)

Fig. 13.3 shows the JACK connections (again in `qjackctl`) for our meeting room studio setup. The eight channels from the converter-mode Pre8 appear as ADAT optical inputs, and the eight channels from the interface-mode Pre8 appear as 'Analog' inputs, all within the firewire device. Ardour was used with two tracks of eight channel audio to record as shown in fig. 13.4.

13.2. Example: MAUDIO USB card

inputs: signal dials: -40db control. Pad: extra -20db. flow model: all inputs have direct JACK outputs the box also has a line out and a phones out A/B button-out :patches line out to phones when OUT(A); (spdif to phones when button-in, B) so ALWAYS button-OUT if not using any SPDIFs Mix: Left: live inputs to phone. Right: PC to phones should be all-right Output: volume of line out Level: volume of phones gtr: inst, vol at 45deg north east

NB sampled audio goes out of the MAUDIO through hw:1,1. That's nonstanard, defaults are usually hw1,0. So need to set to listen to hw:1,1.

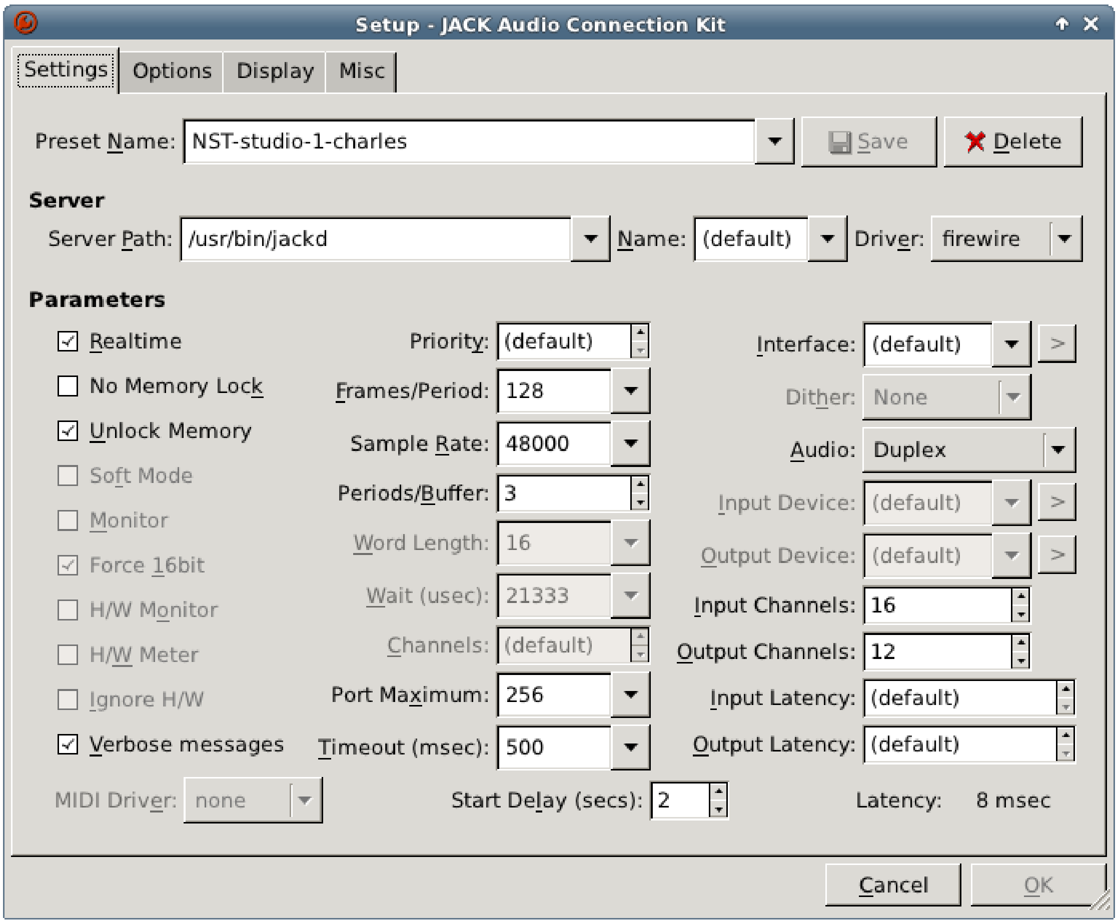


Figure 13.2.: 16 channel recording JACK settings.

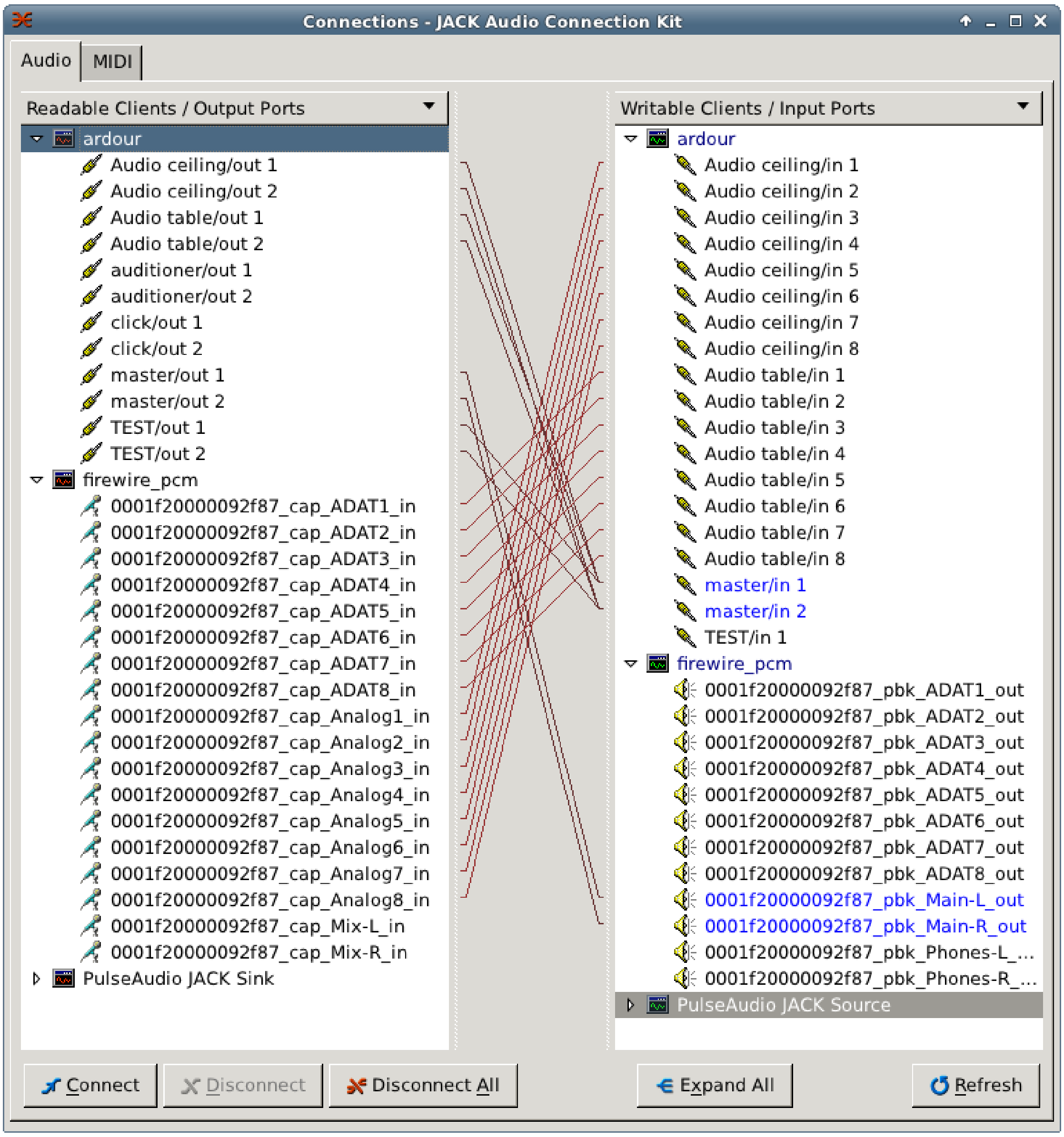


Figure 13.3.: 16 channel recording JACK connections.

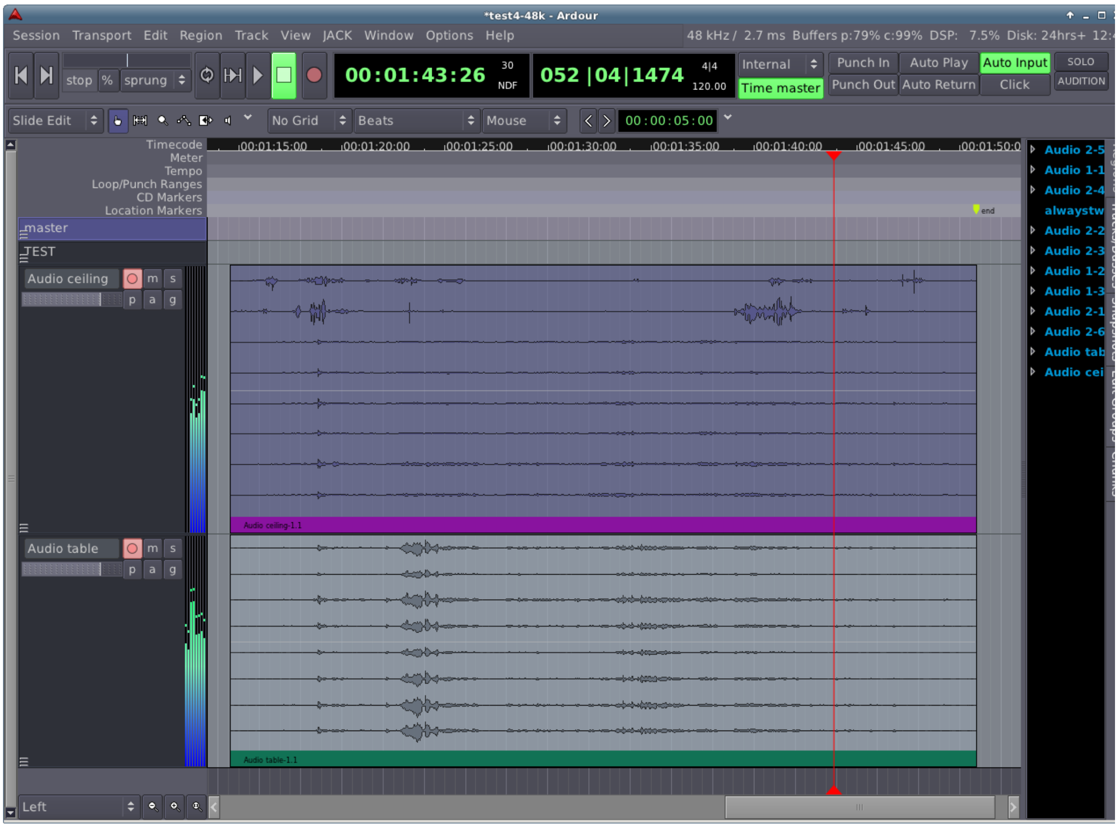


Figure 13.4.: 16 channel meeting room recording in Ardour, using two 8-channel tracks.

2015: on XPS machine – need hw2,0 OUTPUT and hw2,1 INPUT (different!) mich ch 1

14. ALSA (kernel module)

14.1. input

14.2. output

15. Spatial audio: OpenAL

16. JACK

16.1. setup

http://jackaudio.org/faq/linux_rt_config.html

```
    jackd -d alsa - r48000 -p128 -n2  -P hw:1,0 -C hw:1,1     #works for mic in

        jackd -dalsa -dhw:0 -r48000 -p128 -n2  #INTERNAL soundcard (not -p64 k

        killall -9 jackd
```

```
qjackctl: (not needed if use JACK cmd above)
        interface (default) (greyed out)
        in dev: hw:1,1
        out dev: hw:1,0
        chs 2 in 2 out
        128 frames.
```

16.2. JACK tools

NB we need to record form the nonstandard hw1,1 , not the default hw1,0:

```
 /usr/bin/jackd -P65 -u -dalsa -r48000 -p128 -n2 -Xseq -D -Chw:1,1 -Phw:1,0 -i
#works (and in qjackctl)

killall -9 jackd                          #to kill
ecasound -f:16,2,48000 -i jack,system -o rec.wav
#record -- not working** (16bit,2ch,48kHz)
ecasound -i output.wav -o jack,system
#playback

meterbridge 1   #to open a test app - use qjackctl to connect
```

17. LADSPA

17.1. MIDI

live
files

18. Music notation

18.1. MusicML

Is a (not very) human read-writable XML standard backed by many large companies and projects, including beign the standard for music notation on Wikipedia.

(Unliek Lilypond) it would not be nice to write by hand. It is better to use a graphic free editor such as MuseScore.

Can be converted to lilypond for engraving with musicxml2ly tool (ships with Lilypond). It is not usually possible to go the other way. Hence editing should be done in MusicXML and only exported one-way to Lilypond for engraving. Don't write in Lilypond as its not transferrable out. (until someone writes a converter).

Due to its horrific XML verbosity (several lines of text to describe each note) there is a zipped version included in the specification. As with other XMLs, the philosophy is to be human readable but verbose then worry about compression separately.

Ontology of MusicXML: harmony changes and section names are discrete events put into measures at points. Lyrical sylablus are tagged to individual notes. (which make it hard to write lyrics separetly from the music; but forces the music to have the correct maps to the notes. As usual, MusicXML is not for composing in, it's a low-level description of the notes. Might want to use some other ABC like notation for lyrics and compile to here?)

18.2. MuseScore

Is standard opensource graphical MusicXML editor. Has its own format too.

Has semantic guitar chords (appear as 'harmony' objects in XML). Transposable (Notes->Transpose). Can play back as MIDI. To split music into sections eg. verse, chorus, use a 'section break', show by 'new system' flag in the XML, plus add>text>rehearsal mark. (The mark names a point not a segment; but we can interpret as namign the segment up to the next mark or end.)

To enter: press N for note entry mode. Press number 1-8 for duration. Type note letter name C,D,E etc or 0 for rest. They assume the current octave. Cursor keys to move back and forward in time, and to move notes up and down. Also mouse entry.

MuseScore.com has a large library of standard music to download, eg. Beatles, Mozart, in own and MusicXML formats.

18.3. ABC notation

A common standard for folk music especially. Words are paired with *lines* of music, usign hyphens and ties to link them to the notes.

Can be converted to (and partially from?) MusicXML.

Lilypond format is based on it but extends it a lot.

Maybe a nice format for MusicHastie-like systems to all standardize upon.

18.4. Lilypond engraving

Lilypond is like Latex for music scores. It names both a file format and a program. It creates extremely high-end, professional "engravings" of scores, including fine details equivilent to Latex's kerning and ligatures. Such as tweaking the exact size of note heads and stems, setting stem directions, and hundreds of other small rules passed down from centuries of beautiful score engraving tradition. The Lilyond program is not an editor and works from a latex-like source file specifying the musical, but not graphical, objects, such as bars, notes, and time signatures. These files can be written by hand or by an editor program such as Denemo (see below). These files do not contain any high level semantic information and they do not show relationships between simultenoys staves (e.g. that a note in one staff is occuring at the saem time as in another) or enable transpoition – they only represent the low-level musical note content needed for engraving. So you probably don't want to cide directly in LilyPond files. You want to code in some higher level representation then use another program to export it to Lilypond format.

LilyPond can however do basic transposition from within its code – usign a transpose command and environment. This is sufficient for quick conversions eg. for Bb jazz isntruments. But not going to be able to do hierarachical stuff.

eg. to write a Schenkerian composition program: use its won format and notation at high level; export to MusicXML; convert to LilyPond; engrave.

Sample Liypond source:

```
\include "english.ly"
\header {
  title = \markup { "Exit Music" }
  composer = "Radiohead"
}

\markup  {VERSE1 VERSE2 CHORUS VERSE3 ENDING}
\markup {|}
\markup  {VERSE}
\score {


<<
  %chords with durations
```

```
  \chords { a1:m e1:/gsharp  c1:/g  d2:9/fsharp d2:m/f  a1:m e a:sus4 a }

  \relative c' {
                e4 r4 r4 f8 e8 |
                e4 r4 r4 r8 a,8 |
                e'4.( a,8) a4 e'4 |
                e2( d4.) d8 |

                c4 r4 r4 c8 c8 |
                b4 r4 r4 b8 b8( |
                b2) a4 r4 |
                r1 |

  }
  %one word per melody note
  \addlyrics { Wake from your sleep the drying of your tears, to- day we es- c
>>


}

\markup {CHORUS}
\score {
<<
         \chords{g:m11 g:m11 d:7 a:sus2 a:m e:sus4 e}

         \relative c' {
                 \repeat volta 2 {

                                  bflat'2( a2 |
                                  g2.) g4 |
                                  a8 g8 fsharp2. |
                                  e2  e2 |
                                  e2 f4 e4 |
                                  e1 |
                                  r1 |
                 }
         }

         \addlyrics{Breathe keep breathing don't loose your nerve}
>>
}
```

```
\markup {ENDING}
\score {
<<

        \chords{a:m b e f f bflat e e a:m e c d2 d2:m a1:m e a:sus4 a}

        \relative c' {
                c'2. e,4 |
                eflat2. b'4 |
                b2 d,2 |
                d8 c8 r2 r8 f8 |

                \break

                a2 c,2 |
                d2 f2 |
                e2 aflat2 |
                d2 e4. e8 |

                \break

                e2. f8 e8 |
                e2. e4 |
                e4 d4 c4 e4 |
                e2 d4. d8 |

                \break
                \repeat volta 2 {
                                c4 r4 r4 b8 c8 |
                                b4 r4 r4 b8 c8 |
                                b2 a2 |
                                r2 r4 r8 c8^"x3" |
                }
        }

        \addlyrics{You can laugh a spineless laugh}

>>
}
```

To isntall and compile to pdf:

```
sudo apt-get install lilypond
lilypond test.ly
```

Notations:

```
c4 d4 e4 f4  : crochets. 4 means quarter note.
r1 r2 r4 r8  : rests
a4. dotted quarter-note
ds4 : d sharp
a,4 a'4 : octave registers CHANGES. assumes all followign notes in same regist
<c4 e4> : notes together
(c4 e4) : slur notes
\repeat volta 2 { c1 | e1}  : repeat section
c4^"super" e4_"sub" : text above and below note
```

Dynamics are considers as additional over staves. (LilyPond's name is a pun on an old MIDI editor, RoseGarden).

18.4.1. Denemo score editor GUI

Is designed specifically as a front end to LilyPond, and is to LilyPond as LyX is to Latex. A what you see if what you mean GUI with its own quick, cheap and chearful interactive graphical screen display, which gets swapped for the advanced engraver when we are done.

Includes human input + assign rhythms to the hummed pitches.

Curosr updown pitch, or type letter name (register nearest to cursor pitch). Int number keys (use keypad) for durations.

Generally we don't want to compose or edit in Denemo because then we are stuck in Lilypond world and can't get back to MusicXML. We might thus use Denemo to add really fine grained engraving detail to a score done in MuseScore, just for priting..

(Does not yet export MusicXML but may do soon? CHECK?)

Frescobaldi FrescoBaldi - a Lilypond editor which does have MusicXML output? Tis one is not pointy clicky like LyX, is like TexPad, editing the raw lilypond code but with tools to autocompile and display etc.

19. Audio file formats

19.1. wav

header + raw bytes. (raw = Pulse Code Modulation) eg. 32 bit audio = 4 bytes per sample, from linearly spaced possible values. (32 bit here is the 'bit depth'). NB 4Gb limit? still dont know how to fix this/ repair from header.

Since recently - wav also supports floating point representations (which avoid clipping but require more computation).

19.2. vorbis

used in ogg containers.

19.3. AAC

Audio format often used in mp4 containers along with videos.

FLAC Lossles.

20. Tools

20.1. Sox and soxi

Sox is the swiss army knife of the audio command line. Its friend soxi gives information about audio files.

```
        sox k.wav -n stat   #see length(seconds)
        sox in.wav out.wav trim start_time dur
        sox in.wav out.wav pad start_pad_sec end_pad_sec

        sox ts.wav -b 16 output.wav rate -s -a 48000 dither -s
#upsample
        sox mono.wav -c 2 stereo.wav
#make stereo fm mono (JACK cares a lot about right types here)

        #delay and weighed mix.  -p is used in place of output fn, to pipe out
        sox -m -v 0.9 a.wav   -v 0.5 '|sox a.wav -p pad 1' out.wav
#in seconds
        sox -m -v 0.9 a.wav   -v 0.5 '|sox a.wav -p pad 1s' out.wav
#in samples

        #lima
        sox -D -m -v 1 -t sox "|sox a.wav -p pad 0s trim 0 5s"
-v 0.5 -t sox "|sox a.wav -p pad 1s trim 0 5s"  out.wav

        #equiv to (as "-p" subs to "-t sox -" and "-" is the pipe out)
        sox  -m -v 1 -t sox "|sox a.wav -t sox - pad 0s trim 0 5s"
-v 0.5 -t sox "|sox a.wav -t sox - pad 1s trim 0 5s"   out.wav

        #piping out of sox
        sox -m -v 0.9 a.wav   -v 0.5 '|sox a.wav -p pad 1s' -t wav - | less
#WARNING about wav header though

        #pad and trim
        sox Array1-01.wav -p pad 0s trim 13437440s 104800s  o2.wav

 soxi  /share/spandh.ami1/asr/dev/mtg/ac/exp.mdm/exp/lima/pylima/out/testRun/A
```

```
Input File     : '/share/spandh.ami1/asr/dev/mtg/ac/exp.mdm/exp/lima/pylima/ou
Channels       : 1
Sample Rate    : 16000
Precision      : 32-bit
Duration       : 00:00:06.55 = 104800 samples ~ 491.25 CDDA sectors
File Size      : 419k
Bit Rate       : 512k
Sample Encoding: 32-bit Signed Integer PCM
```

20.2. ALSA command line tools

```
aplay -D hw:0,0 stereo.wav    #PC
aplay -D hw:0 stereo.wav      #PC
aplay -D hw:1,0 stereo.wav      #USB
aplay -D hw:1,1 stereo.wav      #silent (spdif?)

arecord -f cd -D hw:1,1 g.wav              #works**
```

21. Music synthesis

22. Speech synthesis

festival

23. Speech recognitions

kaldi?

24. Applications

list best LADSPA plugins?
LMMS

24.1. Ardour 3

just create new audio track, should be already to record. (right click in track space, add track) (dont need to add audio bus patch) for second and more tracks: might need to set to hardware in 1 vs 2 use icons in top right under "Audio2"

flow model: jack,system is conencted auto to the Ardour Master the master is connected to any track with record enabled.

region = one segment of wav or midi

SPACE: play/stop SHIFT-SPACE: play and record SHIFT-R: en/diable record (while playing) (CTL-SPACE: stop and forget record)

PUNCH markers are set by ctrl-drag in marker field. Enable puch IN OUT at top right of screen. (under DSP) Press [to set markers + drag them Start record (not just play) NB this records NEW WAV FILE doesn't destory old one

L: en/disable loop play works similar to punch, using] instead of [each version is separate WAV, keep old ones

cursors: move to start/end of regions shift-cursors: cue (use spa ce to stop) ctl-cursor move to marks (CF set, was keypad)

HOME/END goto start/end

6: Auto-Return enable: back to play start location (after play or rec) 7: clicks

to0 SPLIT: select region, position playhead, press S

ARDOUR editing S split region at edit pount use trim bar (region name area) to crop left and right ends

ARDOUR CONFIGS: preferences - default dir

24.2. ZynAddSubFX

musichastie?

Part III.

Video

25. Video architecture

Multimedia comes in various kinds of streams. Streams may contain video or audio or other things. These may be compressed with codecs, eg h264,theora. Combining streams eg audio+video is multiplexing. Demupliplexing is seprate from decoding. Streams can be passed over realtime protocols such as RTP or stored in container files such as mp4,ogg. problem with compressing video is that it introduces lateny, some compressions require knowledge of future frames, eg MP4. Others are designed for live use eg H264.

Rough rule for video CPU usage: on a 2017 laptop, coding or decoding one video does not start the CPU fan; doing two or more does. (Possibly they are designed to this, as watchign a single video as a consumer is much more common that doing anything fancier.) Consequences: if we want to run two programs than run on the same video stream, the fan is needed. They must either share an encoded stream and each run decoders, so we have two decode processes; or decode one but then stream high-bandwidth raw images around, which also starts the fan! Having the fan on for long periods has previously melted by CPU glue and required sending my laptop to Germany to have it replaced so is not recommended.

eg. playing 2 hi-res youtubes in cinema mode at once takes about 200% CPU and starts the fan. (NB youtube does many clever things such as ceasing to download video when the window is not visible; need to open the windows for a full test.)

IP cameras - streaming. vs. save to container files.

26. Video4Linux (V4L)

is a standard API for video devices such as webcams, plus some drivers implementing it.

API devices appear as files at ls /sys/class/video4linux/ which can be accessed eg. via pythonCV: cv2.videoCapture(3)

v4l2-ctl: gives low level info and setting for USB camera hardware typically GUI tools are calling into it to change camera settings

has many useful help options, see –help

get current settings: v4l2-ctl –device=/dev/video0 –get-fmt-video

eg to set image size and codec: v4l2-ctl –device=/dev/video0 –set-fmt-video=width=800,height=600,pixel

logitech C920 has onboard codecs: YUYV – splits up luma (Y) and croma (Cr and Cb red and blue diffs) and downsamples chroma a bit (YUV 4:4:2) MPEG H264

ask for available frame rates: v4l2-ctl –device=/dev/video0 –list-frameintervals=width=640,height=480,pi

26.1. Loopback

Loopback is a system which allows you to create virtual video devices in V4L, so that other applications may access them just as if they were real devices. Like real devices they appears in /dev/videoX.

Loopback module must be installed then enabled with:

```
sudo apt-get install v4l2loopback-dkms
sudo modprobe v4l2loopback
```

This creates virtual devices /sys/class/video4linux/video1 and /dev/video1 which behave like local webcams. For example they can be opened in OpenCV:

27. Video formats (codecs)

27.1. raw images

Individual frames may be stored or transported, either as uncompressed images themselves (bmp) or as compressed images (png/jpeg). However much better compression comes form considered the frames in sequence and compressing accordingly. eg. often the camera does not move and large areas of background do not change over time.

Images have pixel formats. Traditionally, still images are stored in RGB (or similar, BGR, or RGBA) as three (or four) bytes per color channel. However it is more common for video images to use different pixel formats. This could be HSV (hue saturation value) or more commonly, YUV format, which is a similar idea to HSV. Like RGB, which can be various bit-depths (eg. 24bit, 8bit), YUV also comes in different depths and flavours. YUV also has nice compression properties which make it popular now. Y=luma (overall brightness); U and V are 2D chromas. (Historically: its was nice for sharing streams for black-and-white and color TVs, such as in USA PAL analog broadcasts, because the luma channel looks good by itself in B+W.)

27.2. theora

Open format video codec used in ogg containers.

27.3. h264 (aka. MPEG4-part 10; MPEG4-AVC

Used in Freeview and Freesat Digital TV; Skype, Bluray, Youtube, inside mp4 containers, CCTV cams. heavily patenteted but allowed for gratis use in some cases. Uses image subblocks; motion prediction http://iphome.hhi.de/wiegand/assets/pdfs/h264-AVC-Standard.pdf

27.4. MPEG2 video (MPEG2-Part2; h.262)

used in most DVDs. (don't confuse with MPEG2 container which also has various audio formats (MPEG2-Part3) and system control protocol MPEG2-Part1/h.222).

27.5. MTS

MTS is a video codec found on portable digital cameras and phones. Here is a conversion method using mencoder:

```
sudo apt-get install build-essential subversion zlib1g-dev
        svn checkout svn://svn.mplayerhq.hu/mplayer/trunk mplayer
        cd mplayer
        ./configure
        make
        sudo make install
        mencoder 00001.MTS -o 1.avi -oac copy -ovc lavc -lavcopts vcodec=mpeg4
```

28. ffmpeg

ffmeg is a simple command line tool for performing quick, small operations on media files.

transcode to open source everything (ogg+vorbis+theora; very slow):

```
ffmpeg -i in.mp4 -acodec libvorbis out.ogg
```

extract section:

```
ffmpeg -ss 00:00:05.123 -i in.mp4 -t 00:01:00.00 -c copy out.mp4
```

extract frame as image,

```
ffmpeg -ss 00:12:58 -i in.mp4  -vframes 1 -q:v 2 out.jpg
```

Get video file info,

```
ffprobe -show\_streams -i \textasciitilde{}/data/qb/NorwichLeeds1280.mp4
```

Speedup playback of video

```
ffmpeg -r:v \textquotedbl{}480/1\textquotedbl{}
-i in.mp4 -an -r:v \textquotedbl{}12/1\textquotedbl{} out.mp4
\begin{lstlisting}

Split video into image frame files:
\begin{lstlisting}
ffmpeg -i corinthian\_raw\_images.avi -f image2 frames/frame-\%3d.png
```

(start time and duration args; can be 00:00:00.000 format, or seconds as 00.000. NB only splits to nearest keyframes, unless omit copy to transcode)

downsample,

```
ffmpeg -i in.mts -r 30 -s 960x540 out.mp4
```

downsample resolution:

```
ffmpeg -i in.avi -c:a copy -c:v libx264 -crf 23 -s:v 640x360 output.mp4
```

trim:

```
ffmpeg -i in.avi -vcodec copy -acodec copy -ss 00:00:00 -t 00:00:04 out.avi
```

29. GStreamer

GStreamer a Linux system for media streaming, within and between computers. It is based on small modules (processes) which are 'piped' together similarly to UNIX pipes. Like ROS this allows everything to run as separate processes. Unlike ROS, GStreamer is targetted at efficiency which includes use of containers and codecs for everything. In ROS we pass messages of raw image data in video. In GStreamer we can stream coded compressed video for speed and efficiency. This means we need to consider container and codec modules in the pipelines.

29.1. as command line tool

GStreamer is sometimes used as a competitor to ffmpeg to perform small command line media editing tasks such as cutting and splicing video and audio streams. It is more powerful and harder to use than ffmpeg because of its pipeline abilities. It would be a good habit to get used to using GStreamer instead of ffmpeg even for small tasks, so that it becomes regular and easy to use for large ones.

GStreamer requires modules (plugins) to go in the pipelines. The main ones are distributed in packages: Base: solid stable simple ones used by many other tools; Good: solid stable more specific ones, eg. theora codecs. Ugly: solid stable but with patent issues, eg. reverse engineering of propriatory codecs. Bad: In-development, unstable, bleeding-edge, for use at own risk.

We are using version 1.0 for everything (0.10 also exists) Modules are binary (Cpp) executables, implementing standard API. (Like LADSPA - but not as real time? eg including buffering).

There are modules for reading and writing to/from v4l devices. eg. stream things into new virtual v4l devices for others to read. also UDP data stream sources and sinks. v4l is a minority sport though as gstreamer has its own internal appsrc and appsinks.

We can give transport control commands to jump to points in streams using an additional interface.

29.1.1. Examples - local streams

("!" makes the pipe, like unix "|" but called a "pad" rather than "pipe")

Basic copy a source file to a sink file:

```
gst-launch-1.0 filesrc location=in.mp4 ! filesink location=out.mp4
```

Play an mp3:

```
gst-launch-1.0 filesrc location=in.mp3 ! decodebin ! autoaudiosink
```

Play mp4 video

```
gst-launch-1.0 filesrc location=in.mp4 ! decodebin ! autovideosink
```

(decodebin is a complex plugin which detects the video filetype and automatically constructs a whole series of modules specific to its type to decode it.)

Show TV test card source,

```
gst-launch-1.0 videotestsrc ! autovideosink
```

Extract a segment of a file, recode, and send to another file (not working)

```
gst-launch-0.10 gnlfilesource location=in.mp3 start=0
duration=5000000000 media-start=10000000000 media-duration=5000000000
! audioconvert ! vorbisenc ! oggmux ! filesink location=out.ogg
```

Decode a file and stream raw video to a file,

```
gst-launch-1.0 filesrc location=in.mp4 ! decodebin ! filesink location=out.mp4
```

Read from webcam, downsample, and view:

```
gst-launch-1.0 v4l2src device="/dev/video0" ! video/x-raw,width=640,height=480
```

Split a source into multiple sinks using "tee" (like in Unix "Tee"; named after the shape of the letter "T" where a signal from below splits into left and right copies.),

```
gst-launch filesrc location=test.mp4 ! tee name=tp tp. ! queue ! filesink loca
```

29.1.2. Examples - network streams

Stream to RTP over UDP port (e.g. to broadcast a live video stream on the net):

```
gst-launch-1.0 filesrc location=in.mp4 ! decodebin ! x264enc ! rtph264pay ! ud
```

This works by decoding the local file, encoding it in the desired codec, then containerising (payloading) it as RTP, then sending over a lower-level UDP link.

Version to lowwer quality and reduce latency (?):

```
gst-launch-1.0 filesrc location=in.mp4 ! decodebin ! x264enc pass=qual quantiz
```

To read the stream into GStreamer,

```
gst-launch-1.0 udpsrc port=9001 ! "application/x-rtp, payload=127" ! rtph264de
```

From a remote RTP stream (e.g. an IP camera),

```
gst-launch-1.0 rtspsrc location=rtsp://192.168.0.119:88/videoMain !
rtph264depay ! avdec_h264 ! autovideosink
```

(TODO why no mention of UDP here? Is it just assumed?)

From an IP Camera with a password,

```
gst-launch-1.0 rtspsrc location=rtsp://192.168.0.119:88/videoMain userid=charl
rtph264depay ! avdec_h264 ! autovideosink
```

Stream from my webcam, to h264 over RTP/UDP (the comments tell the other modules about formats which exist at points),

```
gst-launch-1.0 v4l2src ! 'video/x-raw, width=640, height=480, framerate=10/1'
host=192.168.0.220 port=1234
```

29.1.3. Virtual v4l devices - loopback

THIS REQUIRES LOOPBACK TO BE ENABLED AS ABOVE section 26.1.

```
gst-launch-1.0 -v videotestsrc ! tee ! v4l2sink device=/dev/video1
```

We can then open *multiple* OpenCV apps which read the same capture device as,

```
cv2.VideoCapture(1)
```

This works to stream the real webcam into a virtual device and then view in CV:

```
gst-launch-1.0 -v v4l2src device=/dev/video0  ! tee ! v4l2sink device=/dev/vid
```

Send an incoming VLC/RTP format, to local V4L loopback:

```
gst-launch-1.0 udpsrc port=9001 ! "application/x-rtp, payload=127" ! rtph264de
```

Streaming a file into loopback (with conversions, are they needed? Maybe yes, loopback must be raw video not compressed?):

```
gst-launch-1.0 filesrc location=in.mp4 ! decodebin ! videoconvert ! videorate
```

(TODO: This will then open in VLC, but not openCV, why?) (V4L2: Pixel format of incoming image is unsupported by OpenCV. (from v4l info: the file is YU12 pixel format, while a webcam is YUYV. Need to convert?)

29.1.4. TODO

Not working:

```
gst-launch-0.10 filesrc location=in.ogv ! decodebin2 ! ffmpegcolorspace ! vide
```

```
gst-launch-1.0 filesrc location=in.mp4 ! decodebin ! v4l2sink device=/dev/vide
```

(maybe is a bug in gstreamer here?)

29.1.5. OpenCV

OpenCV3 only and must be built with -DWITH$_{G}STREAMER = ON.(IsnotinROS - OpenCVwhichbasicallykillsthisforrobotics.)cv2.VideoCapture(\text{"}autovideosrc!appsink\text{"})$

Hence, use loopback instead.

29.1.6. Python

PYTHON GSTREAMER API https://github.com/rubenrua/GstreamerCodeSnippets good tutorials

29.2. as C API

Application Development Manual: https://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/m
heavily based on GObject/GLib (C, not cpp)

30. ROS

ROS video streams ROS, CV have differerent img formats, use cvbridge node to convert them: http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython

31. (C)VLC

STREAMING FROM VLC is maybe easier than gstreamer! To stream a file as RTP over UDP:

cvlc ~/data/qb/NorwichLeeds1280.mp4 :sout=#transcode{vcodec=h264}:"rtp{dst=127.0.0.1,port=9001

then read from CV with (few seconds delay needed) – but only by one client :-(cap = cv2.VideoCapture("udp://@127.0.0.1:9001")

multicast: cvlc -vvv ~/data/qb/NorwichLeeds1280.mp4 :norm=ntsc :v4l2-width=320 :v4l2-height=240 :v4l2-standard=45056 :channel=1 –no-sout-audio –sout '#transcode{vb="1600",vcodec=m
–loop –ttl

32. Video edit applications

32.1. Openshot

use openshot program import avi , images, make text titles , drag aroudn like cubase to crop: select clip, right click, properties, length export as avi if no clips at start – shows blank screen (for padding)

32.2. Zonemaster

For CCTV multi IP-camera control.

32.3. Desktop recording

recordmydesktop tool

Part IV.

Multimedia

33. Containers

arbitary data streams mixed in. eg robot commands and sensors. Also subtitles, foreign language audio tracks...

Architecturally – media streaming is nice as there are no/few branch hazards. Hence DSPs etc.

33.1. Ogg

Open container format, designed to stream Theora video+Vorbis audio.

33.2. Matroska

Open format container, more general than ogg, designed to store and stream anything. filenames: .mkv, (.mka=audio only; , mks=subititles only; mk3d for 3d video).

33.3. AVI

Audio Video Interleave. Container format containing variously coded audio and video.

34. mp4

mp4 (MPEG4-part 14) is a container format.

35. rosbag as a container

36. H323 (ekiga streams)

(ITU) telcon (signalling) is a recommendation (?) to use a whole stack for AV streams: RTP (application layer) plus 15 more h. protocols for signalling, resitration, control, all over RTP application layer protocols include: eg G.711 speech codec, usees VAD to reduce info in quiet bits G.728, linear pred speech coding G.722 wideband audio codec H.261 an old low res video codec (YouTube, Google Video) (vs MP2,MP4) H.263 1996 video codec T.127 data exchange during conference T.126 image annotation (whiteboard?) also included DVB (http://www.protocols.com/pbook/h323.htm)

vs Skype format, whichis secret/closed.

37. Streaming

Some containers can be used to stream; some streaming protocols don't use containers (?). They are a similar idea, containers usually for files and streams for live streams. When we stream a container we just take the file data and bung it over the network.

37.1. Real-time transport protocol (RTP) streaming

RTP is a dedicated high level protocol (like http,ftp), typically running over UDP (rather than over TCP; timeliness over reliability), for real time multi media (a media stream format is bit like a container file) usually audio and video.

RTP can be used with the even higher-level RTCP session control protocol (rewind, fast forward control; also quality, syncronisation, monitoring/QoS). 2010: other transport layers exist for streams, new, eg SCTP stream control.

via VLC:

cvlc -vvv v4l2:///dev/video0 –sout '#transcode{vcodec=mp2v,vb=800,acodec=none}:rtp{sdp=rtsp://:8

cvlc -vvv v4l2:///dev/video0 –sout '#transcode{vcodec=h264,vb=800,acodec=none}:rtp{sdp=rtsp://:85
https://sandilands.info/sgordon/live-webca-streaming-using-vlc-command-line

to read client:

vlc -vvv –network-caching 200 rtsp://127.0.0.1:8554/

:sout=#transcode{vcodec=h264,acodec=mpga,ab=128,channels=2,samplerate=44100}:rtp{dst=127.0.0.
:sout-keep

multicast option

37.2. Session Initial Protocol (SIP)

to set up telcon calls

38. Augmented reality (GL+CV)

The Graphics Stack architecture above has implications for programs that need to combine 2D and 3D graphics and video, such as head-up displays in 3D games, and augmented reality 3D graphics drawn on top of 2D video camera images.

The best way to do these is to create the image in main RAM in standard (4-byte ABGR, or 3-byte BGR) arrays. Then, like the window manager itself, ask OpenGL to DMA then into GPU texture memory and render them in 3D (using a flat projection matrix). Other 3D graphics can then be drawn over them. Unlike other image copying, this is a FAST operation due to the DMA implementation.

(Note that GL can render to other buffers inside the graphics card, than the one sent the screen. This is done in souble buffering for example. It is also possible to DMA these buffers back into RAM, eg. if we want to get a rendered image as a sprite and save it to an image file.)

39. Parallel programming

ROS image format diff frm cv, cvbridge to convert requires ROS stack overhead ROS kinetic all uses python2, with opencv3 (dont change system pyton to 3 - kills ROS!) serialise/deserialise and pipe implem,entaion : serialise is slow. ROS nodelets:allow several nodes to run as a single process, no msgs.

MPI (network layer 5) wraps all of SYSV/TCPIP/Infiniband python example message passing, not shmem (some shmem in MPI2?) eg. at each frame, map (img,dM) across Pool functions to do stuff.

SYSV-python wrapper shmem - after serialisation

GStreamer / V4L app source and sinks stuck on how to get it into opencv/py

RTP/UDP sockets from GStreamer or vlc via GStreamer (http://stackoverflow.com/questions/13564817/p send-and-receive-rtp-packets)

multiprocessing pipes semaphores shared arrays and map

filelock stuck on a lock

Thrift (over TCP)

40. DSP microprocessors (Texas instruments)

41. FPGA DSP (verilog, Chisel)

42. MISC IDEAS

(DVD uses H.262/MP2 video; MP2/MP1/AAC audio, all encrypted)

DVB-T Suite used by digital TV (T for terrestrial, also S for satellite and others) codecs: video: h.264, AVS ... audio: mp3, mp2, aac ...

camera cvlc command , to record camea is using http to communicate (?) root/88o88o resultion request in cgi URL. Many settings here too. save encode format, currently avi/mpeg, 640x400 480x360, 20fps. as URL, shows CCTV monitor :)