Linux Multimedia Programming

Graphics, Audio, Video

Curator: Charles Fox

Licence

Contents

1. Introduction

idea: select the best of breed of everything and present together. Not a detailed tech manual. Can refer reader to web APIs etc.

hence what we choose to miss out is aslo improtant. eg dont cover OSS. Only latest, one of everything. Find tools that work toghehter. One view of how to do it.

http://www.oreilly.com/openbook/

Similar books: theres a pakt 2017 linux sound book here, https://www.safaribooksonline.com/library/view sound-programming/9781484224960/

beginlstlisting place your source code here

talk about IOX, this was written during it.

We need to use C so we can see the bits and bytes and understand what's going on.

1.1. CMake

```
// 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 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 Tesla. Hence, one does not buy an OpenGL or DirectX card, one buys a Tesla card, then obtains libraries to translate OpenGL or DirectX to Tesla.)

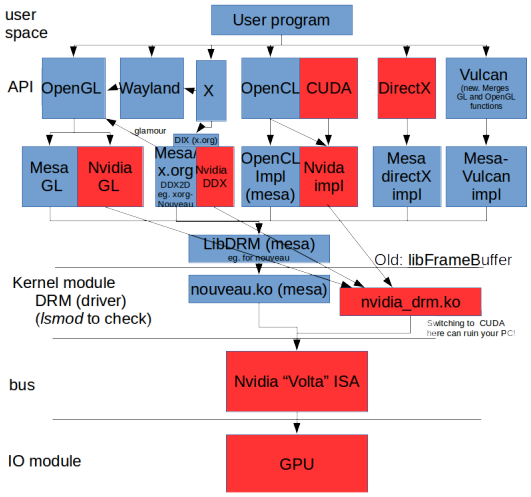
2.2.2. Windowing systems

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).

```
#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  )
```

GL textures 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

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.

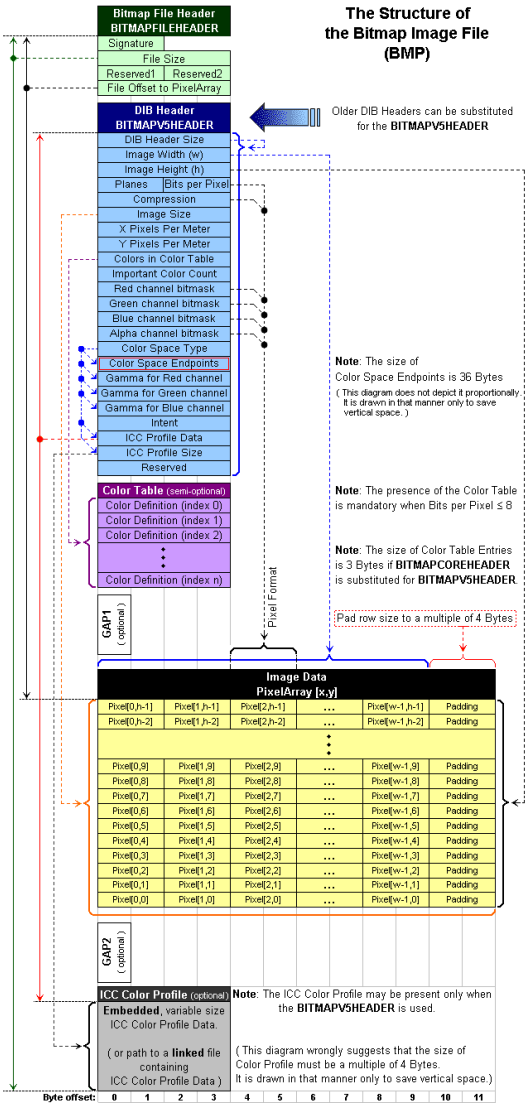
9.5. Portable Document Fomat (pdf)

To extract pages from an existign pdf:

```
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
```

9.6. Fonts

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]
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

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/xh

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. ALSA (kernel module)

13.1. input

13.2. output

13.3. MIDI

live
  files

14. Spatial audio: OpenAL

15. JACK

16. LADSPA

17. File formats

17.1. wav

header + raw bytes. NB 4Gb limit? still dont know how to fix this/ repair from header.

17.2. vorbis (ogg)

18. Tools

18.1. 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

        #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
```

19. Music synthesis

20. Speech synthesis

festival

21. Speech recognitions

kaldi?

22. Applications

list best LADSPA plugins?
- LMMS
- Ardour
- ZynAddSubFX
- musichastie?

Part III.

Video

23. 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.

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

24. 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

24.1. 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.

enable kernel module.

create loopback.

25. GStreamer

linux system for media streams, based on PIPELINEs of modules. (Competitor to ffmpeg in some ways). Requires modules to go in the pipelines (eg good/bad/ugly sets, own code). Like "ROS for video" ? 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).

includes 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.

can give commands to jump to points in streams

it would be a good habit to get used to using gstreamer instead of ffmpeg for genreal conversion use. it is more powerful for realtime use and worth knowing.

("!" 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=~/data/qb/NorwichLeeds1280.mp4 ! filesink location=~/out.mp4

#play mp3 gst-launch-1.0 filesrc location=~/test.mp3 ! decodebin ! autoaudiosink

#play mp4 video gst-launch-1.0 filesrc location=~/test.mp4 ! decodebin ! autovideosink

#extract a segment of a file, recode, and send to another file (not working) gst-launch-0.10 gnlfilesource location=$PWD/source.mp3 start=0 duration=5000000000 media-start=10000000000 media-duration=5000000000 ! audioconvert ! vorbisenc ! oggmux ! filesink location=destination.ogg

#decode a file and stream raw video to a file, gst-launch-1.0 filesrc location=~/test.mp4 ! decodebin ! filesink location=~/out.mp4

#stream to RTP over UDP port gst-launch-1.0 filesrc location=~/test.mp4 ! decodebin ! x264enc ! rtph264pay ! udpsink host=127.0.0.1 port=9001

#stream from Ibex robot 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' ! videoconvert ! x264enc pass=qual quantizer=20 tune=zerolatency ! rtph264pay ! udpsink host=192.168.0.220 port=1234

#streaming to a virtual divce is called loopback and it needs to be enabled: sudo apt-get install v4l2loopback-dkms sudo modprobe v4l2loopback #(creates /sys/class/video4linux/video1 and /dev/video1)

In Ibex, use gstreamer loopback to read an incoming VLC video stream from the robot eg in RTP format, and create a gstreamer stream here into V4L. Then python/CV doesn't care what the source was, it just looks like another webcam. (ffmpeg can also do this?)

We can also do this to stream a video file into gstreamer/loopback(?). eg. to enable multiple processes to read the same stream and process it in parallel. gst-launch-

0.10 filesrc location=~/Documents/my_video.ogv ! decodebin2 ! ffmpegcolorspace ! videoscale ! ffmpegcolorspace ! v4l2sink device=/dev/video1 (maybe is a bug in gstreamer here)

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

#this works to stream a file over UDP, and read and view it on client: gst-launch-1.0 filesrc location=~/data/qb/NorwichLeeds1280.mp4 ! decodebin ! x264enc pass=qual quantizer=20 tune=zerolatency ! rtph264pay ! udpsink host=127.0.0.1 port=9001

gst-launch-1.0 udpsrc port=9001 ! "application/x-rtp, payload=127" ! rtph264depay ! avdec_h264 ! videoconvert ! xvimagesink

#to read from a V4L port on the same client: (eg then view in VLC) gst-launch-1.0 udpsrc port=9001 ! "application/x-rtp, payload=127" ! rtph264depay ! avdec_h264 ! videoconvert ! videorate ! video/x-raw, framerate=25/5 ! v4l2sink device=/dev/video1

*this works to stream a file into a V4L device: (after activiating loopback)

sudo modprobe v4l2loopback gst-launch-1.0 filesrc location=~/data/qb/NorwichLeeds1280.mp4 ! decodebin ! v4l2sink device=/dev/video1

or with some conversions:

gst-launch-1.0 filesrc location=~/data/qb/NorwichLeeds1280.mp4 ! decodebin ! videoconvert ! videorate ! video/x-raw, framerate=25/5 ! v4l2sink device=/dev/video1

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

26. ffmpeg

get video file info ffprobe -show_streams -i ~/data/qb/NorwichLeeds1280.mp4

speedup playback of video ffmpeg -r:v "480/1" -i sprayingVideo.mp4 -an -r:v "12/1" output.mp4

split avi into frames: ffmpeg -i corinthian_raw_images.avi -f image2 frames/frame-%3d.png

extract section: ffmpeg -ss 00:00:05.123 -i in.mp4 -t 00:01:00.00 -c copy out.mp4 #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 resolution: ffmpeg -i in.avi -c:a copy -c:v libx264 -crf 23 -s:v 640x360 output.mp4

```
#trimmign with ffmpeg:
ffmpeg -i 140712-g1_s1_10-09-14_c2_45min.avi  -vcodec copy -acodec copy -ss 00

#downsample HD video
ffmpeg -i hd.mts -r 30 -s 960x540 out.mp4
```

27. File formats

27.1. theora (ogg)

27.2. AVI

audio video interleave. File format containing variously coded audio and video. (vs: ogg, mp4-part14(container), asf(AdvancedSystemFormat,Microsfot)) can also store other interleaved data, eg subtitles, ubisense data

27.3. MP4

27.4. MTS

MTS is found on portable digital cameras and phones as a video format.
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. ROS

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

29. (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=n –loop –ttl

30. Video edit applications

30.1. 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)

30.2. Zonemaster

For CCTV multi IP-camera control.

30.3. Desktop recording

recordmydesktop tool

Part IV.

Multimedia

31. Containers and streams

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.

32. rosbag as a container

33. H264 (skype, DVDs, mp4s, CCTV cams)

h264 (aka MP4-part10 (vs just MP4)) is newer format than MP4, used by our IP cameras used on Bluray they are formats (not suites)

34. 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.

35. Real-time transport protocol (RTP) streaming

RTP is a dedicated http-like protocol, running over UDP (timeliness over reliability), for real time multi media (a media stream format is bit like a container file) usually audio and video.

Application layer (like HTTP, FTP) used with RTCP control protocol (quality, syncronisation, monitoring/QoS) (RTSP is a higher-level session control protol sometimes used together with RTP, e.g. to give rewind/fast-forward type commands.) 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://:85

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

35.1. Session Initial Protocol (SIP)

to set up telcon calls

36. 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.)

37. 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)

38. DSP microprocessors (Texas instruments)

39. FPGA DSP (verilog, Chisel)

40. 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 :)