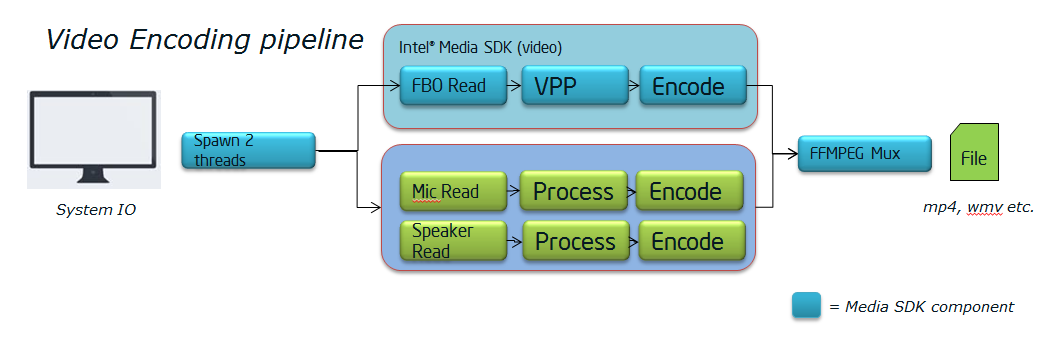
Code Documentation for OpenGL Recorder

## Overview

OpenGL Recorder is a demonstration C++/Java library. It has been integrated into the Minecraft source code as well as the light weight java game library (LWJGL) using JNI. This Manual is written by Farzon Lotfi. If you are unable to get this demo working please feel free to email [farzon@gatech.edu](mailto:farzon@gatech.edu) after 8/9/13 or [farzon.lotfi@intel.com](mailto:farzon.lotfi@intel.com) before.

## Software Architecture

1. Main Pipeline

The pipeline splits into two paths, the video stream and the audio stream that are asynchronous of each other. The video path setups a Frame buffer object and has OpenGL point and write to this off screen surface. Then this off-screen surface splits down two more paths. First it is given to the Video Preprocessing for color conversion and second it is passed to the back buffer so that the frame just seen will be displayed to the User on the next buffer swap[<https://en.wikipedia.org/wiki/Multiple_buffering>].

**OpenGL*: achieving Asynchronous read-back of the screen***

The solution was to redirect all rendering to an off screen Frame Buffer Object (FBO) as mentioned above. This only gets us half way to the solution. You need to also create a double buffered Pixel Buffer Objects (an array of two PBOs). These PBOs are alternated to perform an asynchronous read-back of pixel data to system memory via DMA transfers. This approach is faster than glReadPixel because the DMA transfers are asynchronous.

Sources:

<http://www.codesampler.com/2011/01/21/asynchronous-read-back-of-pixel-data-using-a-frame-buffer-object-and-two-pixel-buffer-objects/>

<http://www.comp.nus.edu.sg/~ashwinna/docs/FBO_Readback_using_PBO.pdf>

<http://www.songho.ca/opengl/gl_pbo.html>

**OpenGL: Bottlenecks to this approach**

You will notice that the DMA transfers are to system memory. Screen Grabs could be faster if the transfer to system memory was not needed. Other potential Bottlenecks, the FBO allocated is the same size as the default frame buffer of the App. This could be done faster with smaller buffers and scaling back to the default frame buffer of the App.

**OpenGL Work around**

OpenGL has a few differences from DX that need to be noted. To make this clear you must think of Frames as Matrices. In OpenGL the first row of a Matrix corresponds to the last row in DX. A row reversal algorithm was developed that takes advantage of SSE instructions was used to reverse the frames. See custom\_reverse\_algo()in FBOReadBack.hpp. The second work around has not been developed but involves color conversion. This is described in detail below.

**OpenGL Win32 Bugs**

The Windows Version of OpenGL does color conversions between GL\_RGBA and GL\_BGRA inefficiently. This conversion cuts the frame rate of this Screen cap library in half. As a result an unsupported color format has to be used which means the VPP (Video Preprocessing) portion of the code needs to be replaced with IPP (Intel performance primitives) [<http://software.intel.com/sites/default/files/m/d/4/1/d/8/UnspColorConv-MediaSDK.pdf>].

**Audio Speaker capture**

This portion of the code is completed but has not been added to the Muxer. I used the following blog post: <http://blogs.msdn.com/b/matthew_van_eerde/archive/2008/12/16/sample-wasapi-loopback-capture-record-what-you-hear.aspx> to learn how to do this properly using the windows audio api. The article was last updated in 2009. The major difference from our two approaches is mine is Object Oriented. The following oddities mentioned in the blog post are mention here for the readers convenience.

There are a couple of oddities for WASAPI loopback capture.  One is that "event mode" doesn't work for loopback capture; you can call pAudioClient->Initialize(... AUDCLNT\_STREAMFLAGS\_LOOPBACK | AUDCLNT\_STREAMFLAGS\_EVENTCALLBACK, ... ), you can call pAudioClient->SetEventHandle(...), and everything will succeed... but the "data is ready" event will never fire.  So this app creates its own waitable timer.

Another oddity is that WASAPI will only push data down to the render endpoint when there are active streams.  When nothing is playing, there is nothing to capture.

For example, play a song, and then run loopback-capture.  While loopback-capture is running, stop the song, and then start it again.  When the song stopped, no more data was available to capture.  Eventually the song started up again, and WASAPI dutifully reported that there was a glitch detected. This library will ignore this glitch for now.

**Audio Microphone**

**FFMPEG Muxing**