NVISION 08

THE WORLD OF VISUAL COMPUTING

What Now? What Next? Integrating with SDI

Thomas J, True
Applied Engineering, NVIDIA

NVIDIA

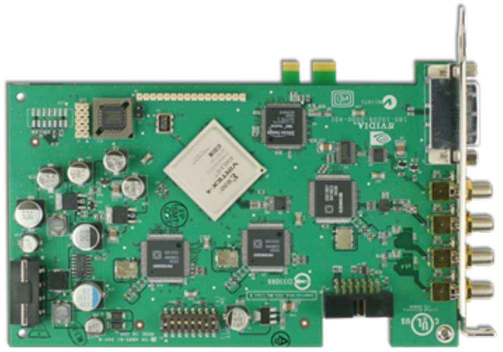
Agenda

- Introduction
- Hardware Architecture
- Software Architecture
- Device Control
- Data Transfer
- Advanced Topics
- More Information
- Questions

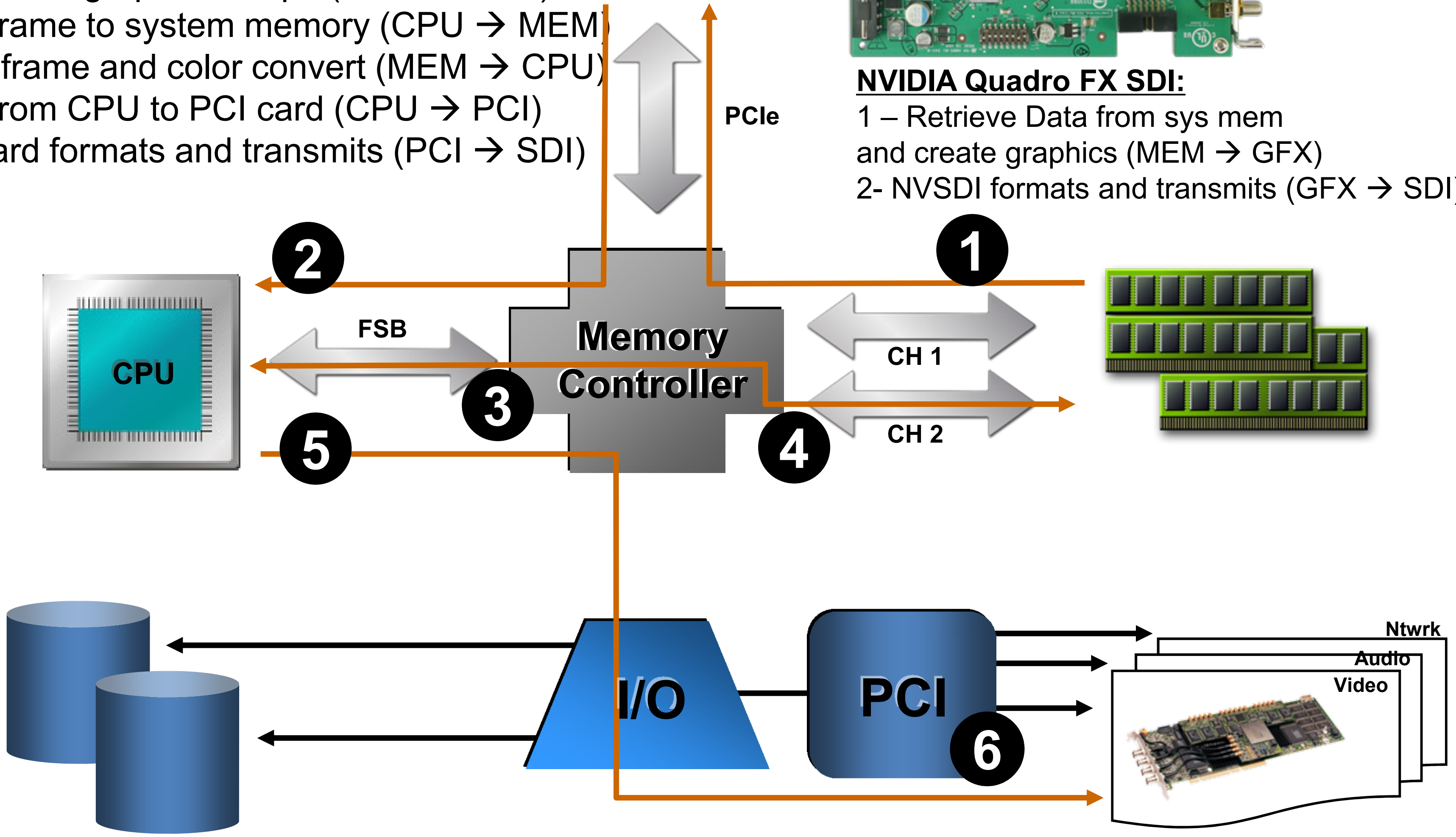
Introduction

Classic PCI SDI boards workflow:

1 – retrieve data from sys mem
And create graphics (MEM → GFX)
2 – read back graphics to cpu (GFX → CPU)
3- write frame to system memory (CPU → MEM)
4 – read frame and color convert (MEM → CPU)
5- write from CPU to PCI card (CPU → PCI)
6- PCI card formats and transmits (PCI → SDI)



NVIDIA Quadro FX SDI:
1 – Retrieve Data from sys mem
and create graphics (MEM → GFX)
2- NVSDI formats and transmits (GFX → SDI)

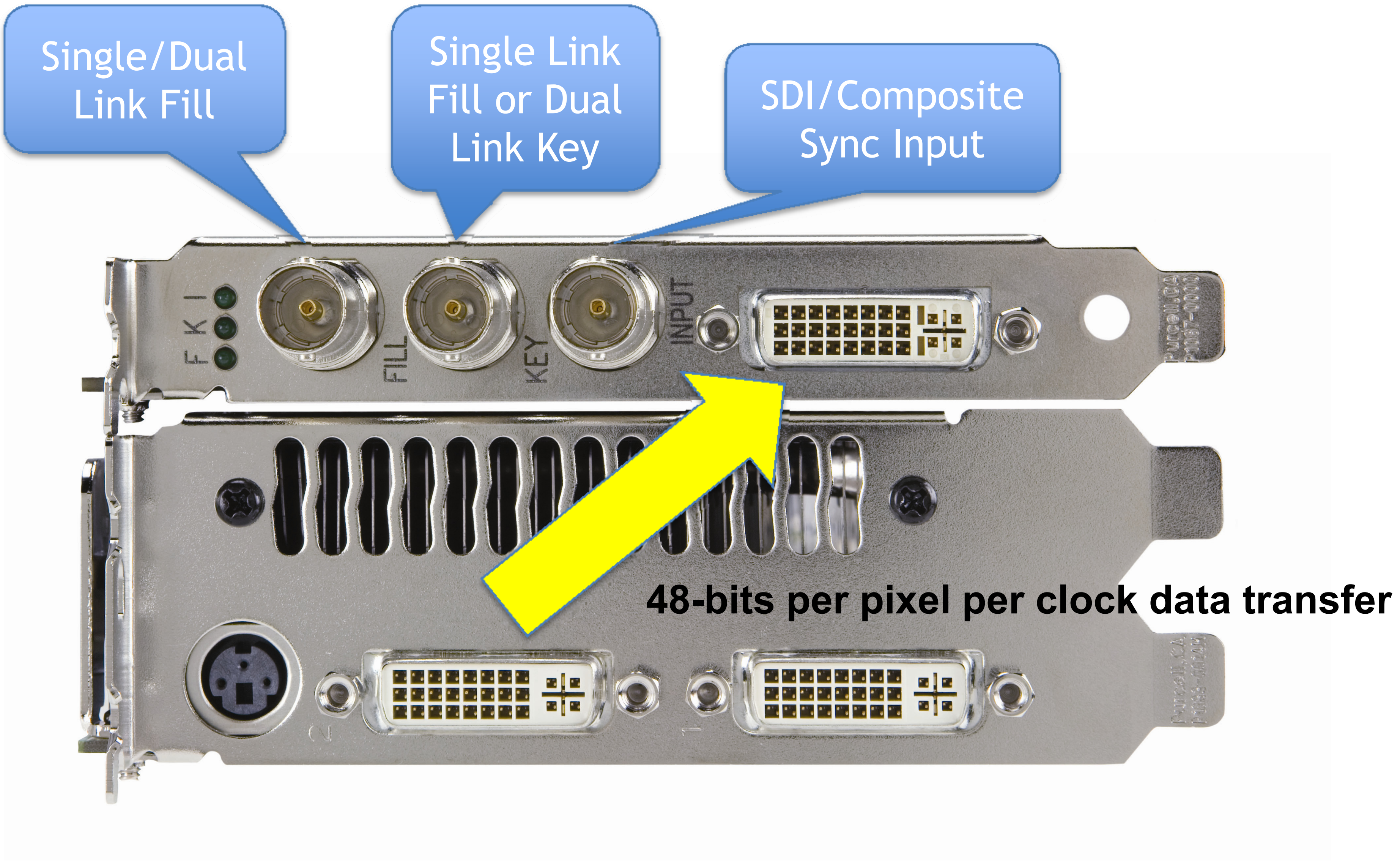


Hardware Architecture

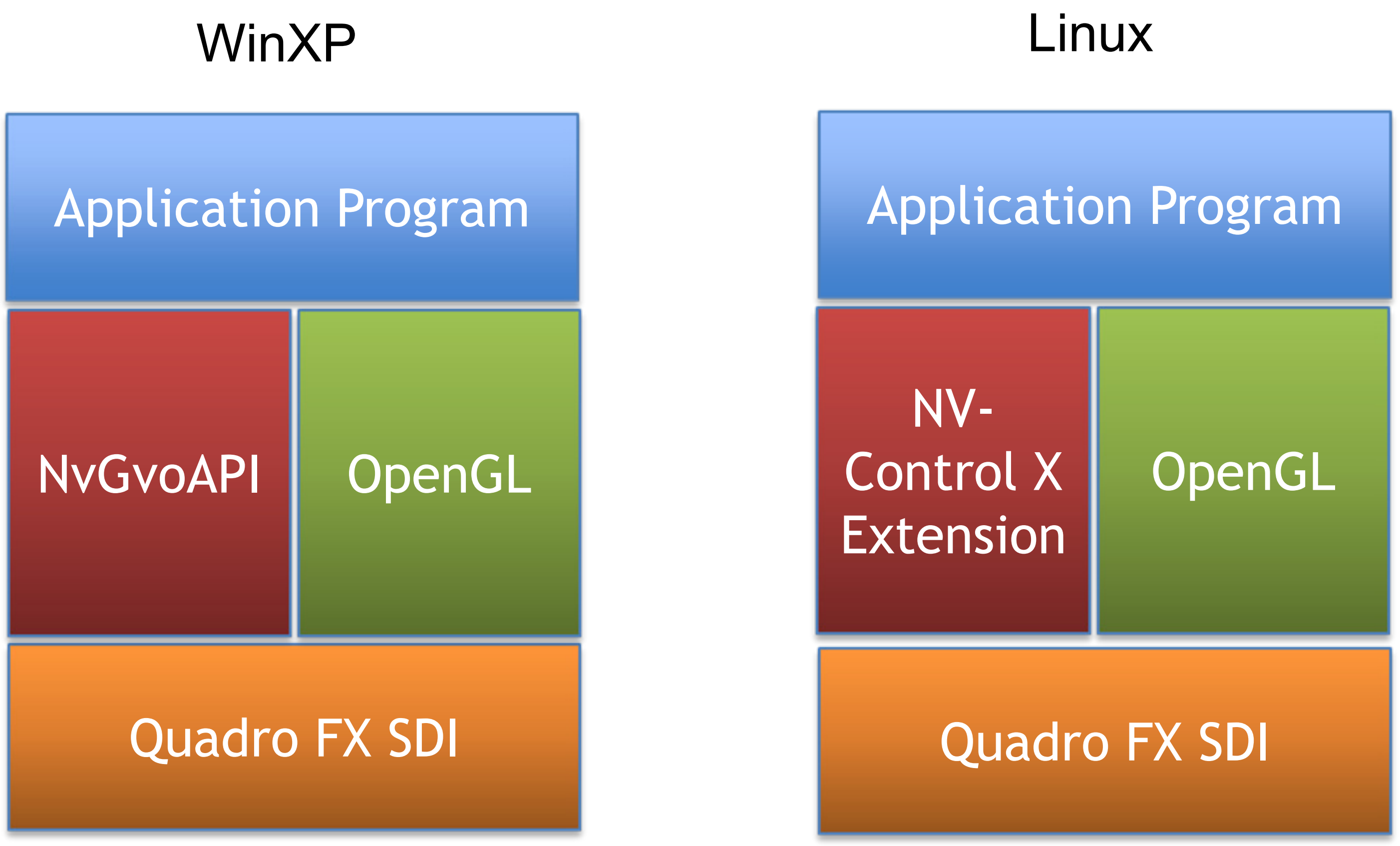
Daughter board for Quadro FX 4600/5600

On-board FPGA implements standard ITU Rec. 601 or 709 or custom CSC

Hardware Architecture


Single/Dual Link Fill
Single Link Fill or Dual Link Key
SDI/Composite Sync Input
F K I
FILL
KEY
INPUT
48-bits per pixel per clock data transfer

Software Architecture



🟥	Device Control
🟩	Data Transfer

Controls

- Signal Format (NTSC, PAL, 720p, 1080i, 1080p, 1080PsF, etc.)
- Data Format (YCrCb/RGB, 422, 444, etc.)
- Sync Type (SDI or Composite)
- Genlock or Framelock
- Horizontal / Vertical Sync Delay
- CSC
- Gama
- Buffer Queue Size

NvGvo API

Header: nvgvoapi.h
Lib: NVCPL.LIB

```
NVRESULT NvGvoCaps( UINT        nAdapterNumber IN,
                    UINT        nReserved      IN,
                    NVGVOCAPS* pAdapterCaps   OUT );

NVRESULT NvGvoOpen( UINT         nAdapterNumber IN,
                    UINT         nReserved      IN,
                    DWORD        dwClass        IN,
                    DWORD        dwAccessRights IN,
                    NVGVOHANDLE* phGvoHandle    OUT );


NVRESULT NvGvoClose( NVGVOHANDLE hGvoHandle IN );


NVRESULT NvGvoDesktopOpen( UINT  nAdapterNumber IN,
                           UINT  nReserved      IN,
                           DWORD dwClass        IN,
                           DWORD dwAccessRights IN,
                           NVGVOHANDLE* phGvoHandle    OUT );

NVRESULT NvGvoDesktopClose( NVGVOHANDLE hGvoHandle IN,
                            BOOL        bRelease   IN );
```

NvGvo API

```
NVRESULT NvGvoStatus( NVGVOHANDLE  hGvoHandle IN,
                                NVGVOSTATUS* pStatus    OUT );

NVRESULT NvGvoSyncFormatDetect( NVGVOHANDLE hGvoHandle IN,
                                          DWORD*      pdwWait    OUT );

NVRESULT NvGvoConfigSet( NVGVOHANDLE  hGvoHandle IN,
                                   NVGVOCONFIG* pConfig    IN );

NVRESULT NvGvoIsRunning( NVGVOHANDLE hGvoHandle IN );

NVRESULT NvGvoStart( NVGVOHANDLE hGvoHandle IN );

NVRESULT NvGvoStop( NVGVOHANDLE hGvoHandle IN );

NVRESULT NvGvoEnumSignalFormats( NVGVOHANDLE  hGvoHandle          IN,
                                 int          nEnumIndex          IN,
                                 BOOL         bByEnum             IN,
                                 NVGVOSIGNALFORMATDETAIL* pSignalFormatDetail OUT );

NVRESULT NvGvoIsFrameLockModeCompatible( NVGVOHANDLE  hGvoHandle          IN,
                                         int          nSrcEnumIndex       IN,
                                         int          nDestEnumIndex      IN,
                                         BOOL*        pbCompatible        OUT );

NVRESULT NvGvoEnumDataFormats( NVGVOHANDLE  hGvoHandle        IN,
                               int          nEnumIndex        IN,
                               BOOL         bByEnum           IN,
                               NVGVODATAFORMATDETAIL* pDataFormatDetail OUT );
```

NV-Control X Extension

Header: NvCtrl.h, NvCtrlLib.h
Lib: libXNVCtrl.a

```
void XNVCTRLSetAttribute (
    Display *dpy,
    int screen,
    unsigned int display_mask,
    unsigned int attribute,
    int value
);


Bool XNVCTRLQueryAttribute (
    Display *dpy,
    int screen,
    unsigned int display_mask,
    unsigned int attribute,
    int *value
);
```

OpenGL

GL_NV_present_video

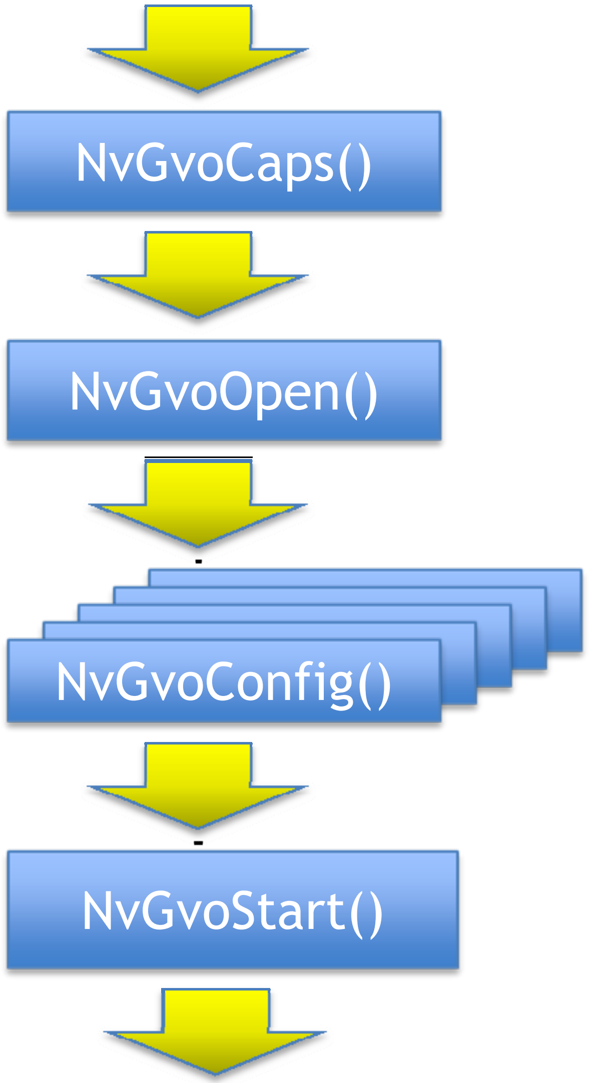
```
void PresentFrameKeyedNV(uint video_slot,
                         uint64EXT minPresentTime, uint beginPresentTimeId,
                         uint presentDurationId,
                         enum type,
                         enum target0, uint fill0, uint key0,
                         enum target1, uint fill1, uint key1);

void PresentFrameDualFillNV(uint video_slot,
                            uint64EXT minPresentTime, uint beginPresentTimeId,
                            uint presentDurationId,
                            enum type,
                            enum target0, uint fill0,
                            enum target1, uint fill1,
                            enum target2, uint fill2,
                            enum target3, uint fill3);


void GetVideoivNV(uint video_slot, enum pname, int *params);

void GetVideouivNV(uint video_slot, enum pname, uint *params);

void GetVideoi64vNV(uint video_slot, enum pname, int64EXT *params);

void GetVideoui64vNV(uint video_slot, enum pname, uint64EXT *params);

void VideoParameterivNV(uint video_slot, enum pname, const int *params);
```

OpenGL

GL_NV_present_video

```
unsigned int *glXEnumerateVideoDevicesNV(Display *dpy,
                                         int screen,
                                         int *nelements);

int glXBindVideoDeviceNV(Display *dpy,
                         unsigned int video_slot,
                         unsigned int video_device,
                         const int *attrib_list);

int wglEnumerateVideoDevicesNV(HDC hDc,
                               HVIDEOOUTPUTDEVICENV *phDeviceList);

BOOL wglBindVideoDeviceNV(HDC hDc,
                          unsigned int uVideoSlot,
                          HVIDEOOUTPUTDEVICENV hVideoDevice,
                          const int *piAttribList);

BOOL wglQueryCurrentContextNV(int iAttribute,
                              int *piValue);
```

Device Initialization - WinXP



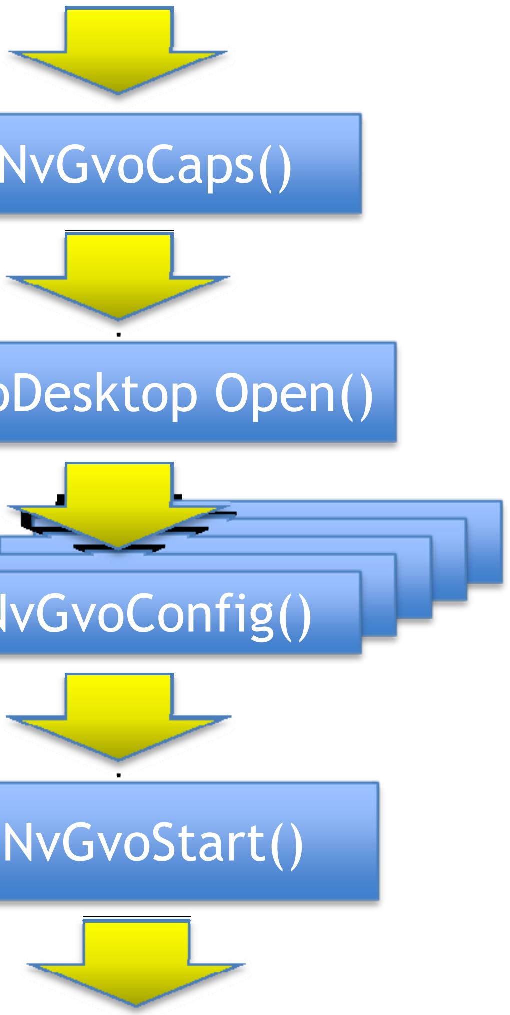
Device Initialization - WinXP

```
const UINT nAdapterNumber = 1;
NVGVOCAPS gvoCaps = {0};
gvoCaps.cbSize = sizeof(gvoCaps);
if (NvGvoCaps(nAdapterNumber, 0, &gvoCaps) == NV_OK)
{
 if (gvoCaps.dwClass & NVGVOCLASS_SDI)
{
 // Quadro FX 4000/4500/5500/4600/5600 SDI Available
}
}
else
{
 printf("Failure: \'%s\'\n", NvGetLastErrorMessage());
 exit(1);
}


NVGVOHANDLE hGVO = INVALID_NVGVOHANDLE;
if (NvGvoOpen(nAdapterNumber, NULL, NVGVOCLASS_SDI,
              NVGVO_O_WRITE_EXCLUSIVE, &hGVO) != NV_OK)
{
     // Handle error.
}
```

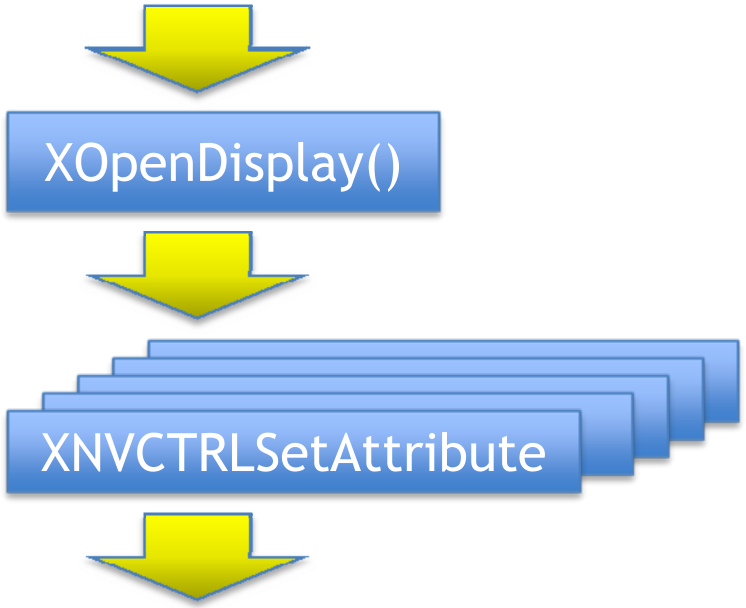
Device Initialization - WinXP

```
// Configure device
NVGVOCONFIG gvoConfig;
gvoConfig.cbSize = sizeof(gvoConfig);
gvoConfig.dwFields = NVGVOCONFIG_SIGNALFORMAT |
                     NVGVOCONFIG_DATAFORMAT |
                     NVGVOCONFIG_SYNCSOURCEENABLE;

gvoConfig.signalFormat = NVGVOSIGNALFORMAT_1080I_5994_SMPTE274;
gvoConfig.dataFormat = NVGVODATAFORMAT_R8B8B8A8_TO_YCRCBA4224;

NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));


// Start video transfers
if ((hGVO) && !(NvGvoIsRunning(hGVO)))
{
     if (NvGvoStart(hGVO) != NV_OK)
     {
     }
}
```

Device Initialization - WinXP



Device Initialization - Linux

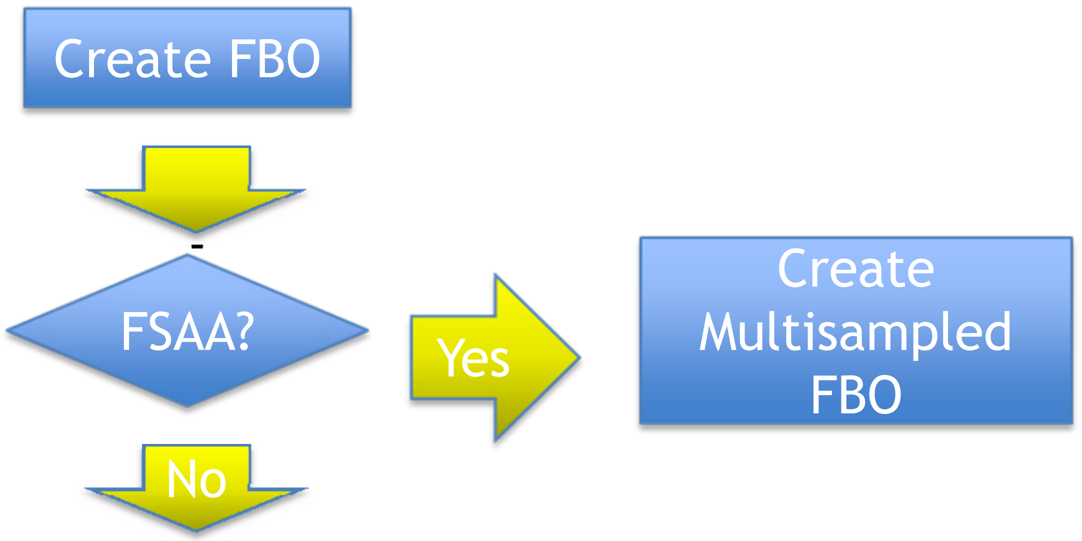


Device Initialization - Linux

```
// Query NV_CTRL_GVO support.
if (!XNVCTRLQueryAttribute(dpy, screen, 0,
                           NV_CTRL_GVO_SUPPORTED, &value)) {
     return FALSE;
} else {
     return TRUE;
}

     screen = DefaultScreen(dpy);

     // Set video signal format
     XNVCTRLSetAttribute(dpy, screen, 0,
NV_CTRL_GVO_OUTPUT_VIDEO_FORMAT,

     NV_CTRL_GVO_VIDEO_FORMAT_1080I_59_94_SMPTE274);

     // Set video data format
     XNVCTRLSetAttribute(dpy, screen, 0,
NV_CTRL_GVO_DATA_FORMAT,

     NV_CTRL_GVO_DATA_FORMAT_R8G8B8A8_TO_YCRCBA4224);
```

FBO Initialization

Per-Channel OpenGL Render Target Initialization

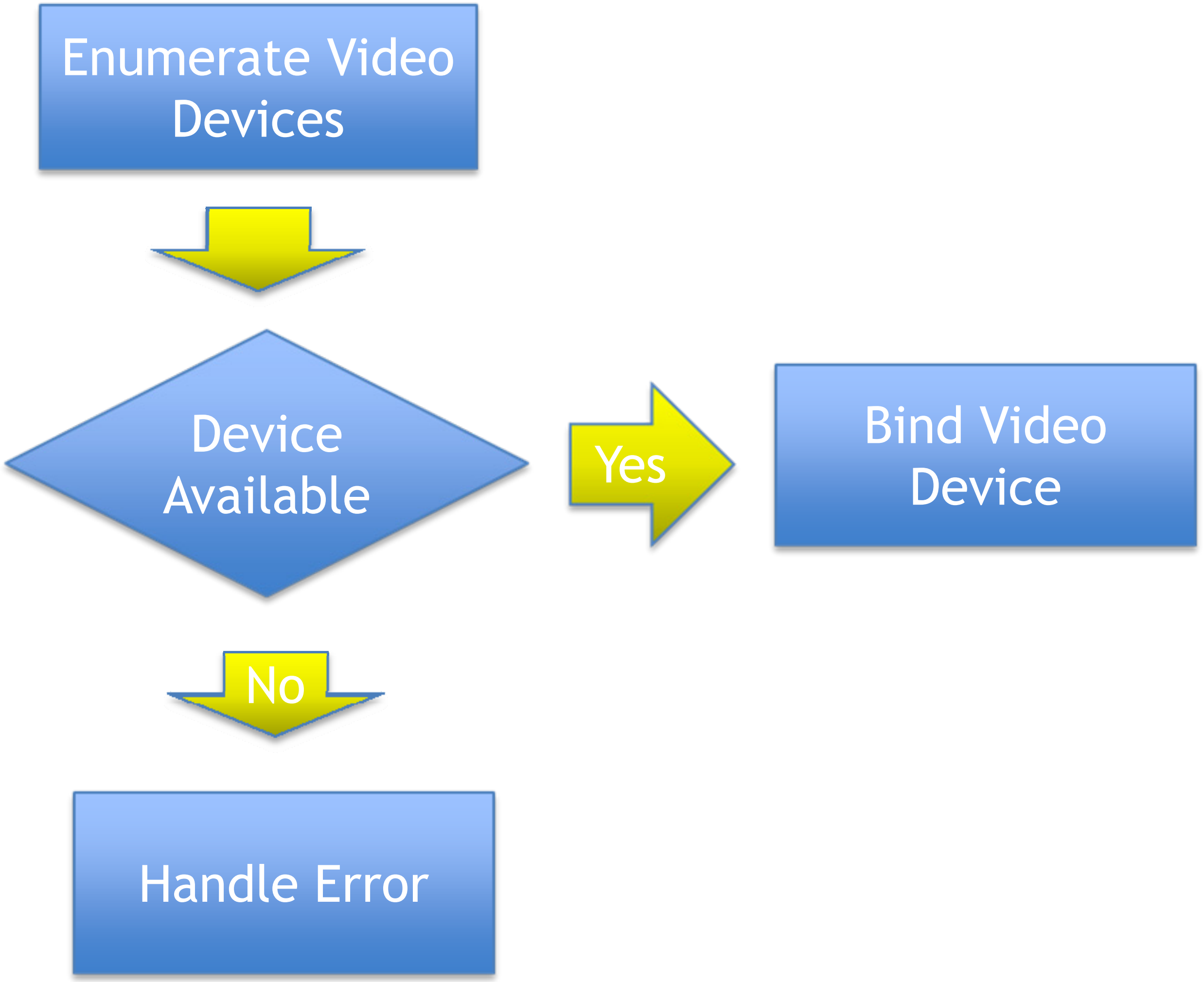


FBO Initialization

```
GLuint fboId;
GLuit textureObject;
GLuint renderbufferIds[2];
glGenRenderbuffersEXT(numRenderbuffers, renderbufferIds);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, renderbufferIds[0]);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_RGBA8, width, height);

if (depth) {
    glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, renderbufferIds[1]);
    glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, width,
                                     height);
}

glGenFramebuffersEXT(1, &fboId);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fboId);
if (!textureObject) {
       glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                                    GL_RENDERBUFFER_EXT, renderbufferIds[0]);
} else {
       glBindTexture(GL_TEXTURE_RECTANGLE_NV, textureObject);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                                GL_TEXTURE_RECTANGLE_NV, textureObject, 0 );
}

if (depth) {
    glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
                                        GL_RENDERBUFFER_EXT, renderbufferIds[1]);
}
```

Data Transfer Initialization

Per-Channel OpenGL Video Device Initialization

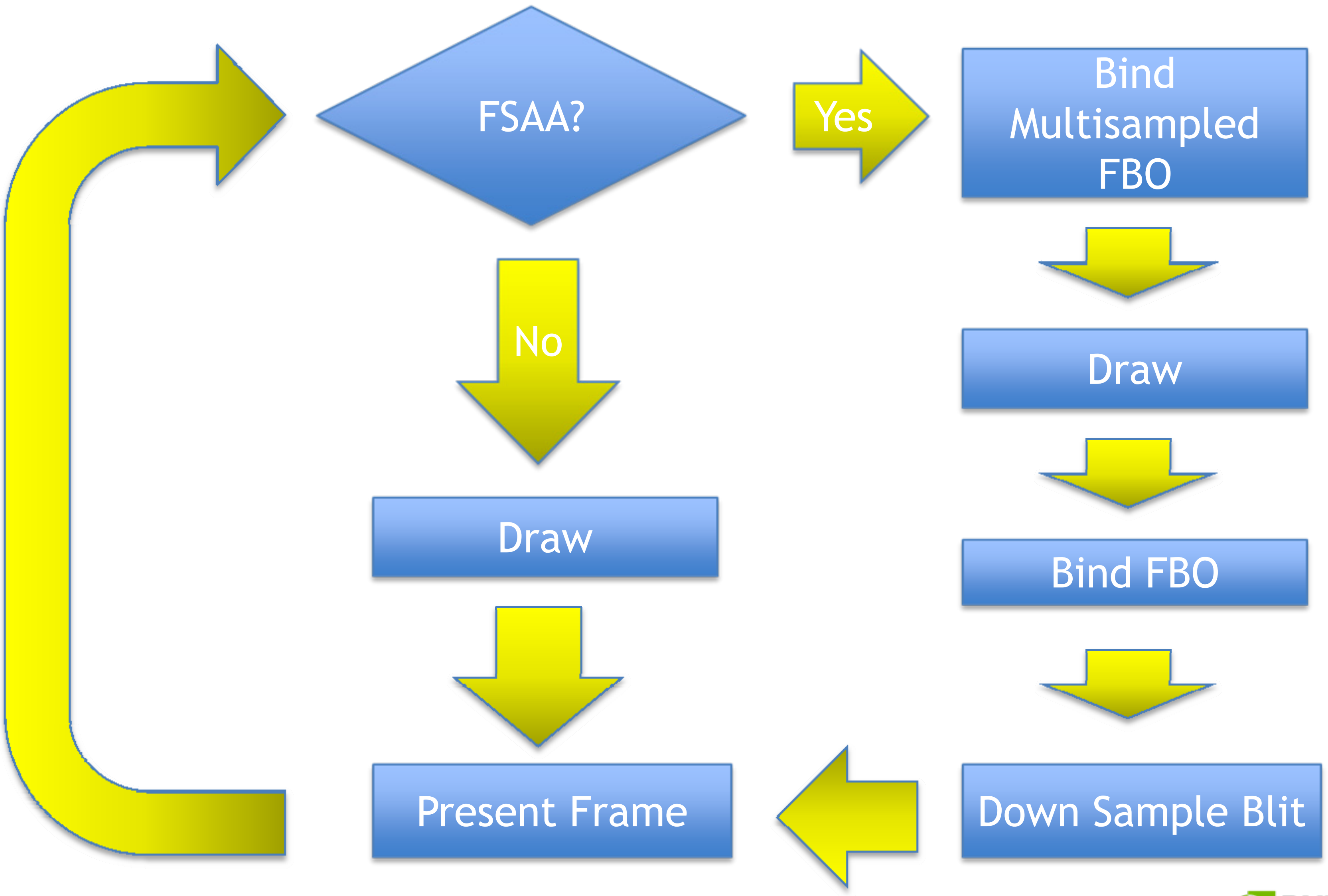


Data Transfer Initialization

```
// Enumerate the available video devices and
// bind to the first one found
HVIDEOOUTPUTDEVICENV *videoDevices;

// Get list of available video devices.
int numDevices = wglEnumerateVideoDevicesNV(ghWinDC, NULL);
if (numDevices <= 0) {
    // Handle error
}

videoDevices = (HVIDEOOUTPUTDEVICENV *)malloc(numDevices *
                                              sizeof(HVIDEOOUTPUTDEVICENV));
if (!videoDevices) {
    fprintf(stderr, "malloc failed.  OOM?");
    exit(1);
}
if (numDevices != wglEnumerateVideoDevicesNV(ghWinDC, videoDevices)) {
    // Handle error
}

//Bind the first device found.
if (!wglBindVideoDeviceNV(ghWinDC, 1, videoDevices[0], NULL)) {
    // Handle error
}

// Free list of available video devices, don't need it anymore.
free(videoDevices);
```

FBO Data Transfer

Draw Loop



FBO Data Transfer

```
// Update Texture
// Unbind texture object and bind FBO
glBindTexture(GL_TEXTURE_RECTANGLE_NV, 0);
if (options.fsaa == 1) {
     gFBO.bind(gWidth, gHeight);
} else {
     gFBOMultiSampled.bind(gWidth, gHeight);
     glEnable(GL_MULTISAMPLE);
}



// Draw Frame Content to FBO
.
.
.
```

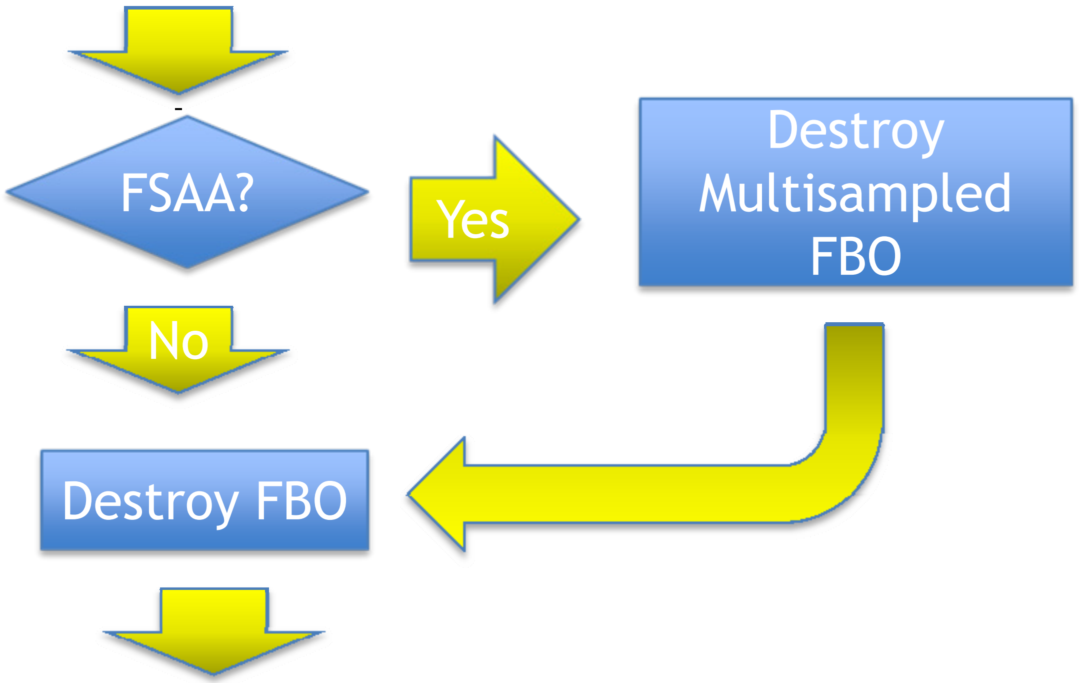
FBO Data Transfer

```
// Do FSAA blit
if (options.fsaa == 1){

     // Unbind FBO
     gFBO.unbind();

} else {

     // If using multisample render buffer, then blit to downsample and filter
     gFBOMultiSampled.bindRead(gWidth, gHeight);
     gFBO.bindDraw(gWidth, gHeight);

     glBlitFramebufferEXT(0, 0, gWidth, gHeight, 0, 0, gWidth, gHeight,
                         GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
                         GL_STENCIL_BUFFER_BIT, GL_NEAREST);

     glDisable(GL_MULTISAMPLE);

     // Unbind FBOs
     gFBOMultiSampled.unbind();
     gFBO.unbind();
}
```

FBO Data Transfer

```
// Bind texture object
glBindTexture(GL_TEXTURE_RECTANGLE_NV, gTO);
glEnable(GL_TEXTURE_RECTANGLE_NV);

glPresentFrameKeyedNV(1, 0, 0, 0,
                    GL_FRAME_NV, GL_TEXTURE_RECTANGLE_NV, gTO, 0,
                    GL_NONE, 0, 0);
```

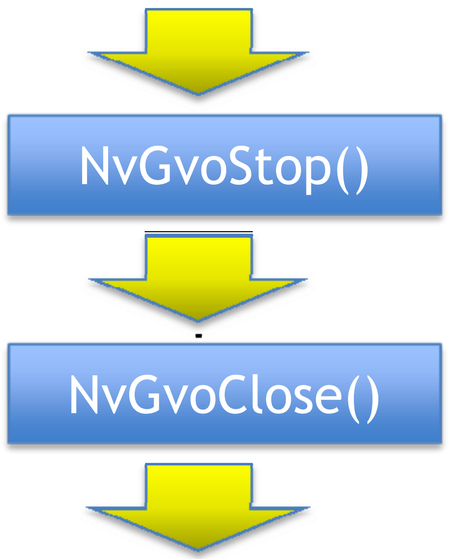
Data Transfer Teardown



Data Transfer Teardown

```
// Unbind video device.
wglBindVideoDeviceNV(ghWinDC, 1, NULL, NULL)

// Destroy objects
glDeleteFramebuffersEXT(1, &fboId);
glDeleteRenderbuffersEXT(numRenderbuffers, renderbufferIds);
glDeleteTextures(1, &gTO);
```

Device Teardown - WinXP



```
//
// Cleanup video
//
BOOL
cleanupVideo(GLvoid)
{
        if ( g_hGVO ) {

                if ( NvGvoIsRunning(g_hGVO) ) {
                        NvGvoStop(g_hGVO);
                }

                if ( NvGvoClose(g_hGVO) != NV_OK ) {
                        return E_FAIL;
                }
        }

        return S_OK;
}
```

Device Teardown - Linux

Colorspace Conversion

ITU Rec. 601 or 709 at 12-bit Precision

$$Y = offset_Y + scale_Y * (r_y*R + g_y*G + b_y*B)$$
$$Cb = offset_{cb} + scale_{cb} * (r_{cb}*R + g_{cb}*G + b_{cb} *B)$$
$$Cr = ofset_{cr} + scale_{cr} * (r_{cr}*R + g_{cr}*G + b_{cr}*B)$$

Coefficients

SD ITU 601 Coefficients

	r	g	b
Y	0.2989	0.5865	0.1150
C_b	-0.1684	-0.3310	0.5000
C_r	0.5000	-0.4181	-0.08095

SD ITU 709 Coefficients

	r	g	b
Y	0.2130	0.7156	0.0723
C_b	-0.1146	-0.38450	0.5000
C_r	0.5000	-0.4535	-0.0455

Scale

10-bit Video Range

$scale_y = (940 - 64) / 1024 = 0.85547$
$scale_{cr} = (960 - 64) / 1024 = 0.875$
$scale_{cb} = (960 - 64) / 1024 = 0.875$

8-bit Video Range

$scale_y = (235 - 16) / 256 = 0.85547$
$scale_{cr} = (240 - 16) / 256 = 0.875$
$scale_{cb} = (240 - 16) / 256 = 0.875$

10-bit Full Range

$scale_y = (1019 - 4) / 1024 = 0.992$
$scale_{cr} = (1019 - 4) / 1024 = 0.992$
$scale_{cb} = (1019 - 4) / 1024 = 0.992$

Offset

Video Range

$offset_y = (64 / 1024) = 0.0625$
$offset_{cr} = offset_{cb} = ((64 + 960) / 2) / 1024 = 0.5$

Full Range

$offset_y = (4 / 1019) = 0.003925417$
$offset_{cr} = offset_{cb} = ((4 + 1019) / 2) / 1024 = 0.5$

Full-Scene Antialiasing

Use GL_EXT_framebuffer_multisample

```
if (num_samples > 1) {
     glRenderbufferStorageMultisampleEXT(GL_RENDERBUFFER_EXT,
                                         num_samples, texFormat,
                                         width, height);
} else {
     glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, texFormat, width, height);
}
```

Full-Scene Antialiasing

```
// Bind buffer object
if (options.fsaa == 1)
     glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, gFBO);
else
     glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, gFBOMultiSampled);
     glEnable(GL_MULTISAMPLE);
 // Draw frame content here
.
.
.
if (options.fsaa == 1){
     // Unbind FBO
     glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
} else {
     // If using multisample render buffer, blit to downsample and filter
     glBindFramebufferEXT(GL_READ_FRAMEBUFFER_EXT, gFBOMultiSampled);
     glBindFramebufferEXT(GL_DRAW_FRAMEBUFFER_EXT, gFBO);
     glBlitFramebufferEXT(0, 0, gWidth, gHeight,
                          0, 0, gWidth, gHeight,
                          GL_COLOR_BUFFER_BIT |
                          GL_DEPTH_BUFFER_BIT |
                          GL_STENCIL_BUFFER_BIT,
                          GL_NEAREST);
     glDisable(GL_MULTISAMPLE);
     // Unbind FBOs
      glBindFramebufferEXT(GL_READ_FRAMEBUFFER_EXT, 0);
     glBindFramebufferEXT(GL_DRAW_FRAMEBUFFER_EXT, 0);
}
```

Use GL_EXT_framebuffer_multisample

External Sync - WinXP

1) Set Sync Source

```
// Set sync source if specified.
// before a valid sync can be detected.
if (options->syncEnable) {
     l_gvoConfig.syncEnable = options->syncEnable;
     l_gvoConfig.syncSource = options->syncSource;

     switch(options->syncSource) {
          case NVGVOSYNCSOURCE_SDISYNC:
               l_gvoConfig.dwFields |= NVGVOCONFIG_SYNCSOURCEENABLE;
               break;
          case NVGVOSYNCSOURCE_COMPSYNC:
               l_gvoConfig.compositeSyncType = NVGVOCOMPSYNCTYPE_AUTO;
               l_gvoConfig.dwFields |= (NVGVOCONFIG_SYNCSOURCEENABLE |
                                          NVGVOCONFIG_COMPOSITESYNCTYPE);
               break;
     } // switch
}

NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));
```

External Sync - WinXP

2) Detect Sync and Confirm Compatability

```
// Trigger redetection of sync format
l_nvResult = NvGvoSyncFormatDetect(g_hGVO, &l_dwWait);

// Wait for sync detection to complete
Sleep(l_dwWait);

// Get sync signal format
l_gvoStatus.cbSize = sizeof(NVGVOSTATUS);
l_nvResult = NvGvoStatus(g_hGVO, &l_gvoStatus);

// Verify that incoming sync signal is compatible outgoing video signal
if (!options->frameLock) {
     if (l_gvoStatus.syncFormat != l_gvoConfig.signalFormat) {
          // Incompatible - handle error
     }
     l_gvoConfig.frameLockEnable = FALSE;
     l_gvoConfig.dwFields |= NVGVOCONFIG_FRAMELOCKENABLE;
} else {  // Framelock Case
     BOOL l_bCompatible;
     l_nvResult = NvGvoIsFrameLockModeCompatible(g_hGVO, l_gvoStatus.syncFormat,
                                                  l_gvoConfig.signalFormat, &l_bCompatible);
     if (l_bCompatible) {
          l_gvoConfig.frameLockEnable = TRUE;
          l_gvoConfig.dwFields |= NVGVOCONFIG_FRAMELOCKENABLE;
     } else {
          // Incompatible - handle error
     }
}
```

External Sync - WinXP

3) Set Sync Delay (optional)

```
// Sync delay
NVGVOSYNCDELAY l_gvoSyncDelay;
memset(&l_gvoSyncDelay, 0, sizeof(l_gvoSyncDelay));
l_gvoSyncDelay.wHorizontalDelay = options->hDelay;
l_gvoSyncDelay.wVerticalDelay = options->vDelay;
l_gvoConfig.dwFields |= NVGVOCONFIG_SYNCDELAY;
l_gvoConfig.syncDelay = l_gvoSyncDelay;

// Setup external sync
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));
```

External Sync - Linux

```
// Enable genlock
XNVCTRLSetAttribute(dpy, screen, 0, NV_CTRL_GVO_SYNC_MODE,
                     NV_CTRL_GVO_SYNC_MODE_GENLOCK);

// Set sync type to composite.
XNVCTRLSetAttribute(dpy, screen, 0, NV_CTRL_GVO_SYNC_SOURCE,
                    NV_CTRL_GVO_SYNC_SOURCE_COMPOSITE);
XFlush(dpy);

// Sleep to allow time for sync detection
sleep(2);

// Detect input sync.
XNVCTRLQueryAttribute(dpy, screen, 0,
                      NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED,
                      &val);

// If valid sync detected, query input video format.
if (val) {
     XNVCTRLQueryAttribute(dpy, screen
                            NV_CTRL_GVO_INPUT_VIDEO_FORMAT, &val);
}
```

External Sync - Linux

```
// Get sync status
while ((!bSync) && (inc < MAX_WAIT_TIME)) {

     sleep(1);

     // Query the lock status
     XNVCTRLQueryAttribute(dpy, DefaultScreen(dpy), 0,
                               NV_CTRL_GVO_SYNC_LOCK_STATUS, &val);

     if (val == NV_CTRL_GVO_SYNC_LOCK_STATUS_LOCKED) {
          bSync = TRUE;
     }
}
```

Performance Monitoring

```
static int cur_query = 0;
static bool queryTime = GL_FALSE;
GLuint64EXT presentTime, durationTime;
static GLuint64EXT lastPresentTime = 0;
static GLuint64EXT sendTime[NUM_QUERIES];
GLuint presentTimeID = gPresentID[cur_query];
GLuint presentDurationID = gDurationID[cur_query];

cur_query++;

// Query video present time and duration.  Only do this once
// we have been through the query loop once to ensure that
// results are available.
if (queryTime) {
      glGetQueryObjectui64vEXT(presentTimeID,GL_QUERY_RESULT_ARB, &presentTime);
      glGetQueryObjectuivARB(presentDurationID,GL_QUERY_RESULT_ARB, &durationTime);

      float latency = (presentTime - sendTime[cur_query]) *.000001;
      float presentationInterval = (presentTime –lastPresentTime) * .000001;

      int bufsQueued = (int)(latency / presentationInterval);

      lastPresentTime = presentTime;
}

// Query send time
glGetVideoui64vNV(1, GL_CURRENT_TIME_NV, &sendTime[cur_query]);

// Draw to video
glPresentFrameKeyedNV(1, 0,  presentTimeID, presentDurationID,
                      GL_FRAME_NV,GL_TEXTURE_RECTANGLE_NV, gTO, 0,
                      GL_NONE, 0, 0);
if (cur_query == NUM_QUERIES) {
      cur_query = 0;0
      queryTime = GL_TRUE;
}
```

Query Objects

Video Memory Usage

Framebuffer	**Width:**	1920	**Height:**	1080		
Color : (32-bit RGBA, double buffered)	1920 x 1080 x 4 x 2 / 1024 / 1024 =					15.82 MB
Depth : (32-bit with packed stencil)	1920 x 1080 x 4 / 1024 / 1024 =					7.91 MB
Overlay : (16-bit front + back)	1920 x 1080 x 2 x 2 / 1024 / 1024=					7.91 MB
Total:						31.64 MB
FBO	**Width:**	1920	**Height:**	1080	**Samples Per Pixel:**	4
Color: (32-bit RGBA, double buffered)	1920 x 1080 x 4 x 4 x 2 / 1024 / 1024 =					63.28 MB
Depth: (32-bit with packed stencil)	1920 x 1080 x 4 x 4 / 1024 / 1024 =					31.64 MB
Total:						94.92 MB
SDI Video	**Width:**	1920	**Height:**	1080	**Num Buffers:**	5
	1920 x 1080 x 8 x 5 / 1024 / 1024 =					79.10 MB
Grand Total:						205.66 MB

Multiple SDI Channels

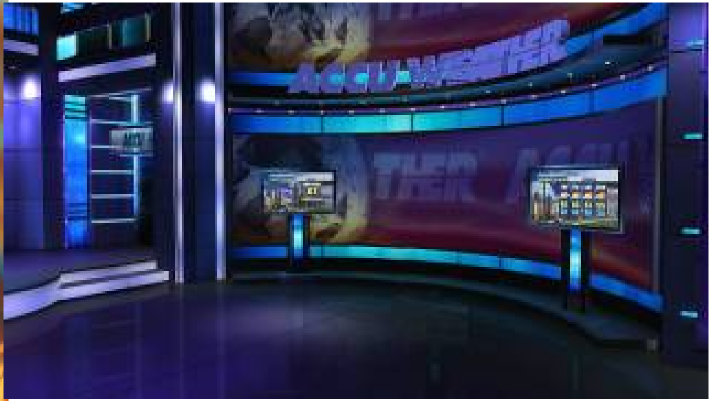
Two Independent 8-bit YCrCb Video Channels from Single Board

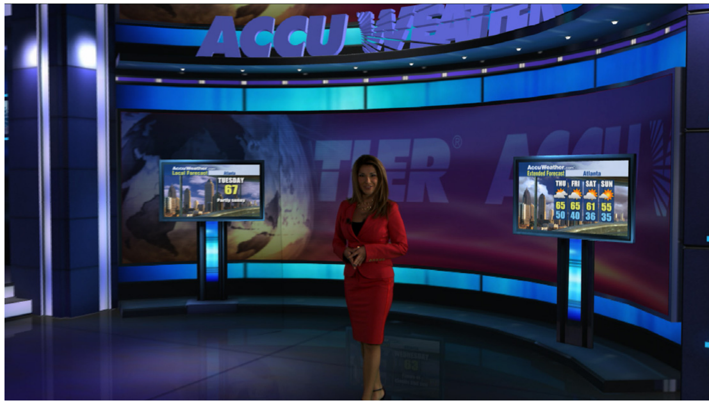
```
// Draw oontents of first channel
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo1Id);
drawPattern1();
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);


// Draw contents of second channel
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo2Id);
drawPattern2();
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);


// Send both channels to the SDI device.
glPresentFrameDualFillNV(1, 0, 0, 0, GL_FRAME_NV,
                       GL_RENDERBUFFER_EXT, renderbuffer1Id,
                       GL_NONE, 0,
                       GL_RENDERBUFFER_EXT, renderbuffer2Id,
                       GL_NONE,0);
```

Multiple SDI Channels

Two Independent Video Channels from a Multi-GPU config

Alpha Compositing



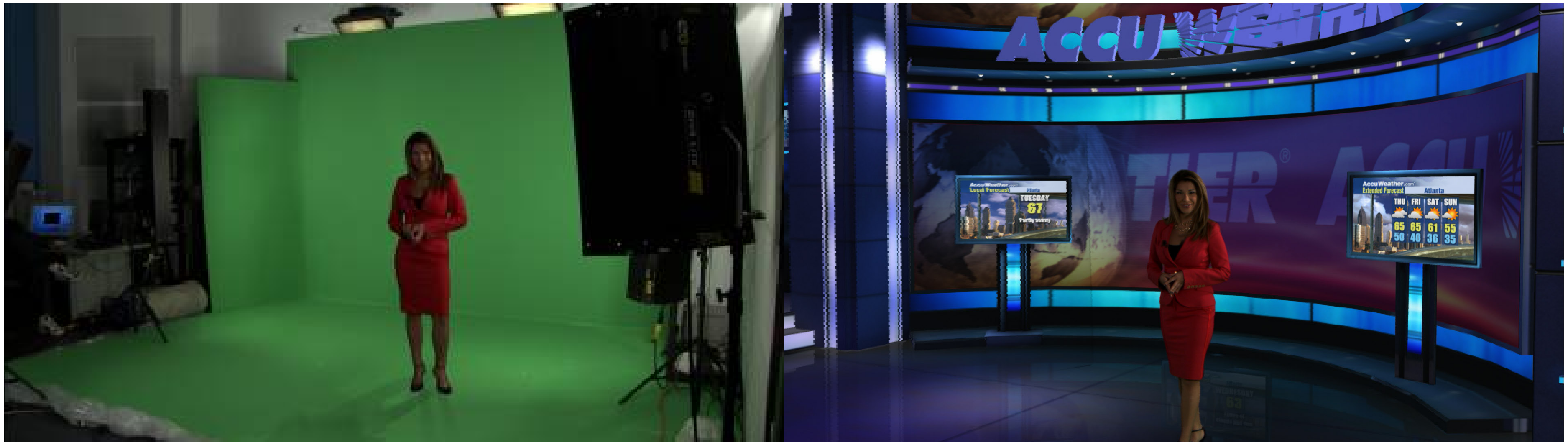

Images courtesy of Accuweather using the Cinema Live HD system

*output color = input video color * (1 - alpha) + input graphics color * (alpha)*

```
l_gvoConfig.dwFields = 0;
l_gvoConfig.dwFields = NVGVOCONFIG_COMPOSITE;
l_gvoConfig.bEnableComposite = TRUE;
l_gvoConfig.dwFields = NVGVOCONFIG_ALPHAKEYCOMPOSITE;
l_gvoConfig.bEnableAlphaKeyComposite |= TRUE;
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));
```

Note: Buffer objects must be GL_RGBA8 or GL_RGBA16F_ARB

Chroma-Keying



Images courtesy of Accuweather using the Cinema Live HD system

Chroma-Keying

```
// Cr composite ranges
l_gvoConfig.dwFields = 0;     // reset fields
l_gvoConfig.dwFields = NVGVOCONFIG_COMPOSITE | NVGVOCONFIG_COMPOSITE_CR;
l_gvoConfig.bEnableComposite = TRUE;
l_gvoConfig.compRange.bEnabled = TRUE;
l_gvoConfig.compRange.dwRange = 0;
l_gvoConfig.compRange.dwMin = options->crCompRange[0];
l_gvoConfig.compRange.dwMax = options->crCompRange[1];
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));

l_gvoConfig.compRange.dwRange = 1;
l_gvoConfig.compRange.dwMin = options->crCompRange[2];
l_gvoConfig.compRange.dwMax = options->crCompRange[3];
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));

// Cb composite ranges
l_gvoConfig.dwFields = 0;     // reset fields
l_gvoConfig.dwFields = NVGVOCONFIG_COMPOSITE | NVGVOCONFIG_COMPOSITE_CB;
l_gvoConfig.bEnableComposite = TRUE;
l_gvoConfig.compRange.bEnabled = TRUE;
l_gvoConfig.compRange.dwRange = 0;
l_gvoConfig.compRange.dwMin = options->cbCompRange[0];
l_gvoConfig.compRange.dwMax = options->cbCompRange[1];
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));

l_gvoConfig.compRange.dwRange = 1;
l_gvoConfig.compRange.dwMin = options->cbCompRange[2];
l_gvoConfig.compRange.dwMax = options->cbCompRange[3];
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));
```

Luma-Keying

```
// Y composite ranges
l_gvoConfig.dwFields = 0;     // reset fields
l_gvoConfig.dwFields = NVGVOCONFIG_COMPOSITE |
NVGVOCONFIG_COMPOSITE_Y;
l_gvoConfig.bEnableComposite = TRUE;
l_gvoConfig.compRange.bEnabled = TRUE;
l_gvoConfig.compRange.dwRange = 0;
l_gvoConfig.compRange.dwMin = options->yCompRange[0];
l_gvoConfig.compRange.dwMax = options->yCompRange[1];
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));

l_gvoConfig.compRange.dwRange = 1;
l_gvoConfig.compRange.dwMin = options->yCompRange[2];
l_gvoConfig.compRange.dwMax = options->yCompRange[3];
NVGVOCHECK(NvGvoConfigSet(g_hGVO, &l_gvoConfig));
```

Ancillary Data

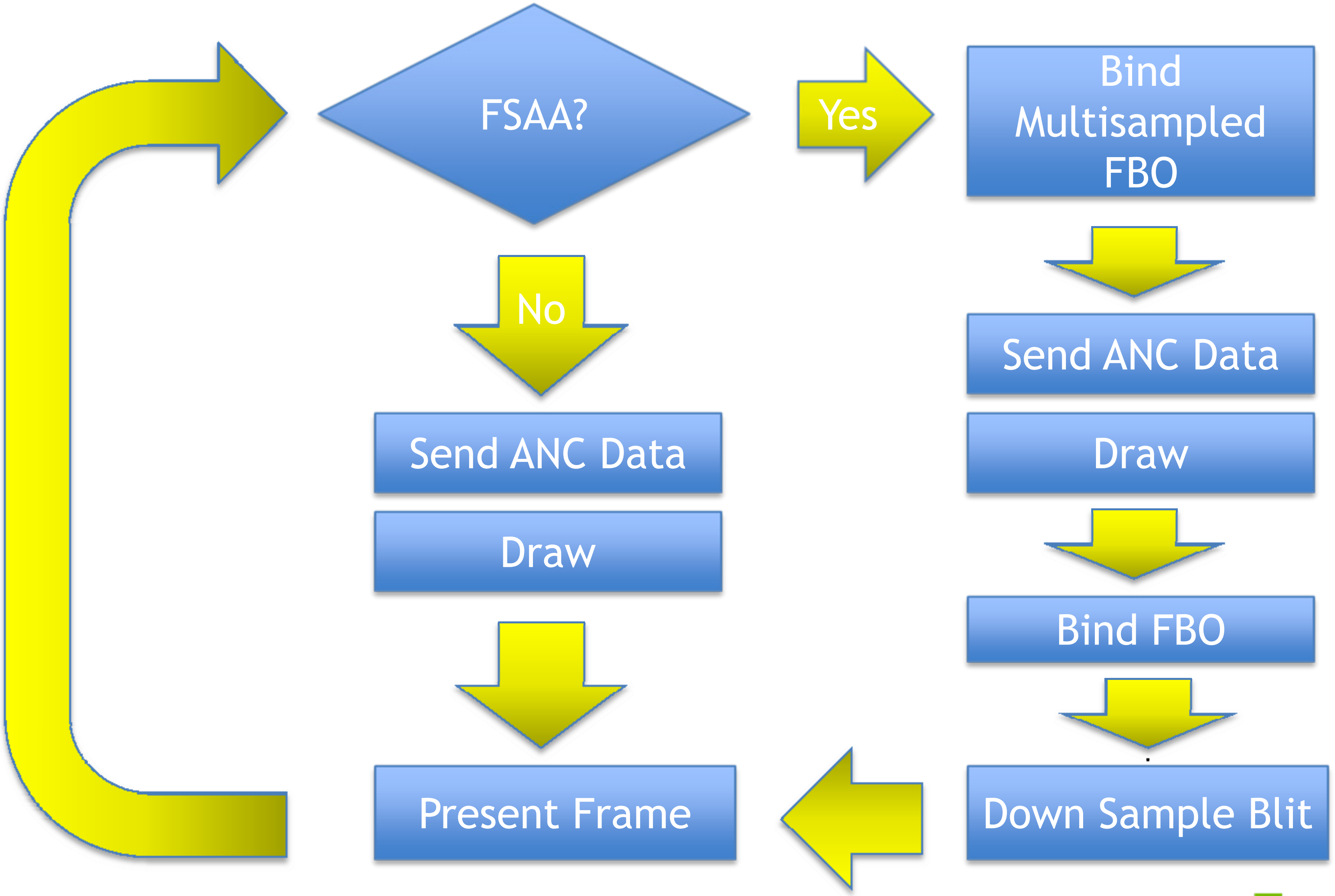
```
// Per Frame
typedef struct tagNVGVOANCDATAFRAME {
    NvU32 version;                // Structure version
    NvU32 fields;                // Field mask
    NvU32  *audioData01to04[4];    // Data pointer for audio channels 1-4
    NVGVOANCAUDIOCNTRL audioCntrl01to04;  // Controls for audio channels 1-4
    NvU32  *audioData05to08[4];  // Data pointer for audio channels 5-8
    NVGVOANCAUDIOCNTRL audioCntrl05to08;  // Controls for audio channels 5-8
    NvU32  *audioData09to12[4];  // Data pointer for audio channels 9-12;
    NVGVOANCAUDIOCNTRL audioCntrl09to12;  // Controls for audio channels 9-12
    NvU32  *audioData13to16[4];  // Data pointer for audio channels 13-16;
    NVGVOANCAUDIOCNTRL audioCntrl13to16; // Controls for audio channels 1316
    NvU32 LTCTimecode;           // RP188
    NvU32 LTCUserBytes;
    NvU32 VITCTimecode;
    NvU32 VITCUserBytes;
    NvU32 FilmTimecode;
    NvU32 FilmUserBytes;
    NvU32 ProductionTimecode;    // RP201
    NvU32 ProductionUserBytes;   // RP201
    NvU32 FrameID;
    NvU32 numCustomPackets;
    NVGVOANCDATAPACKET *CustomPackets;
} NVGVOANCDATAFRAME;
```

Ancillary Data

```
// Initialize ANC API
 NvGVOANCAPI_Initialize((NvGVOHandle)g_hGVO);

 // Send ANC data
 NvGVOANCAPI_SendANCData(NULL, &ancData);
```

Ancillary Data

Draw Loop with Ancillary Data



More Information

- OpenGL Specifications
  - GL_NV_present_video
  - GL_EXT_framebuffer_object
  - GL_EXT_timer_query
- Quadro FX SDI Programmer's Guide

NVGVOSDK Download

Available to registered developers at
 http://nvdeveloper.nvidia.com

Email ttrue@nvidia.com for access.

Questions?