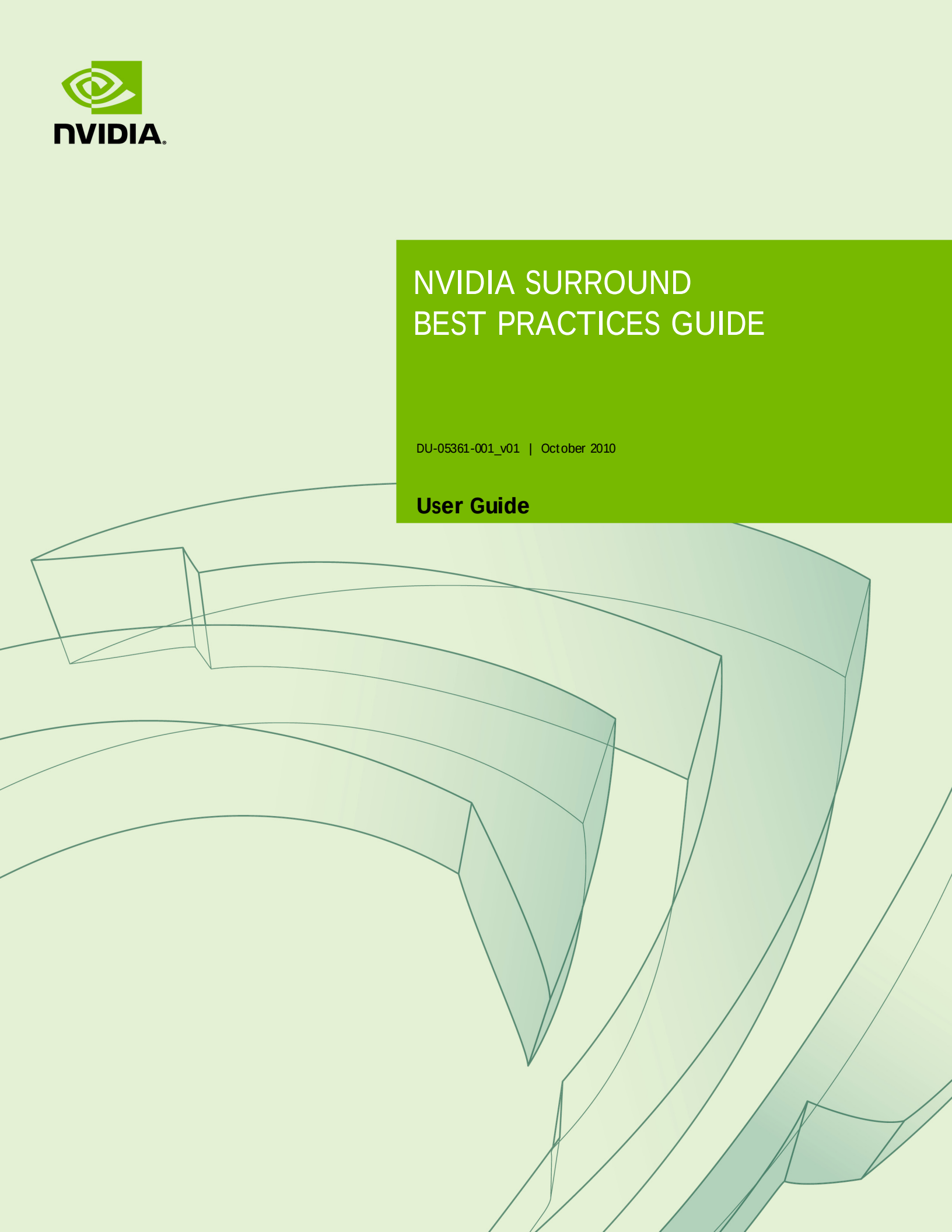

NVIDIA SURROUND BEST PRACTICES GUIDE

DU-05361-001_v01 | October 2010

User Guide

DOCUMENT CHANGE HISTORY

DU-05361-001_v01

Version	Date	Authors	Description of Change
01	June 21, 2010	LN, TS	Initial release
02	August 27, 2010	LN	Revised
03	October 14, 2010	LN	Revised

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

WHAT IS THIS DOCUMENT?

This *Best Practices Guide* should be used to help developers obtain the best compatibility with NVIDIA® Surround and NVIDIA 3D Vision™ Surround technologies, allowing for an incredibly immersive gaming experience.

WHO SHOULD READ THIS GUIDE?

This guide is intended for developers who are interested in supporting multi-display NVIDIA Surround configurations when designing and implementing systems in their game or engine. This guide is meant to assist developers in avoiding common pitfalls that have been observed when running games in Surround configurations.

MODE SUPPORT

ENUMERATE AVAILABLE DISPLAY MODES

In order to properly support NVIDIA 3D Vision Surround and NVIDIA Surround configurations, it is important to enumerate all supported modes exposed by the graphics boards present in the system. This is important because NVIDIA Surround technology exposes many resolutions that may not have been considered *standard resolutions* previously, and assuming (or hard-coding) support for particular resolutions or display modes can often lead to problems; This is not only because the adapter simply may not work at the modes a game has been hard-coded (without proper enumeration) to offer to users, but also because it prevents newer widescreen and bezel-corrected display modes that are exposed by the driver from being offered to users in a game.

SUPPORT LANDSCAPE AND PORTRAIT MODES

Another important consideration for supporting NVIDIA Surround technology is handling both landscape and portrait modes for device creation and rendering. *Landscape mode* is when the monitors are placed in standard orientation on a desk (Figure 1).

Figure 1. Three Displays in Landscape Mode

A user can rotate a 1920x1080 display (physically and in the NVIDIA control panel) by 90 degrees, resulting in an orientation known as *portrait mode*, with a resolution of 1080x1920. NVIDIA Surround allows you to run three displays in a portrait mode configuration (Figure 2). As an example, three 1920x1080 displays in portrait mode results in a resolution of 3240x1920, so a game should not assume that a user has a particular display orientation. Proper enumeration of modes will always provide the modes available for the current display orientation.

Figure 2. Three Displays in Portrait Mode

Finally, game developers may want to know the monitor orientation for proper HUD placement. The implications of display orientation for in-game HUD elements are discussed later, in the *UI, Menu, and Video Considerations* section.

SUPPORT ALL ASPECT RATIOS

As a result of offering the many new display modes available when using NVIDIA Surround, "non-standard" aspect-ratio configurations will likely become more common among users' systems. While many games currently support 5:4, 4:3, 16:10, and 16:9 aspect ratios, NVIDIA Surround configurations in landscape orientation may have aspect ratios such as 15:4, 12:3, 48:10, and 48:9.

Additionally, in cases where Surround displays are arranged in portrait orientation, configurations may have aspect ratios of 27:16, 15:8, 9:4, and 12:5. This means that for proper Surround support, it is important for games to avoid assuming or forcing the use of certain aspect ratios. Where possible, developers should allow for field of view adjustments in game to account for these new aspect ratios. *Table 1* provides a list of common aspect ratios and their related 1x3 (1 display high x 3 displays wide) landscape and portrait Surround aspect ratios:

Table 1. Common Aspect Ratios

Standard Aspect Ratios		NVIDIA 1x3 Surround Aspect Ratios			
		Landscape Mode		Portrait Mode	
5:4	1.25	**15:4**	3.75	**12:5**	2.4
4:3	1.3333	**12:3**	4.0	**9:4**	2.25
16:9	1.7777	**48:9**	5.3333	**27:16**	1.6875
16:10	1.6	**48:10**	4.8	**15:8**	1.875

UI, MENU, AND VIDEO CONSIDERATIONS

USER INTERFACE/MENU/PRE-RENDERED VIDEO PLACEMENT

In NVIDIA Surround mode there can be components of games which are undesirable to have on outside displays (landscape orientation), or split between multiple displays. Examples of these components include game menus, pre-rendered videos, and user interface elements such as heads-up displays (HUDs).

Some games are not designed with systems' display configuration in mind. As a result, they can end up stretching menus, subtitles, or pre-rendered in-game videos—like cut scenes—across the multiple displays. This causes difficulty reading and using the menus, and distortion of the videos, which are typically designed for a particular aspect ratio.

Figure 3. In-game Video Scaled Up with Undesirable Stretching

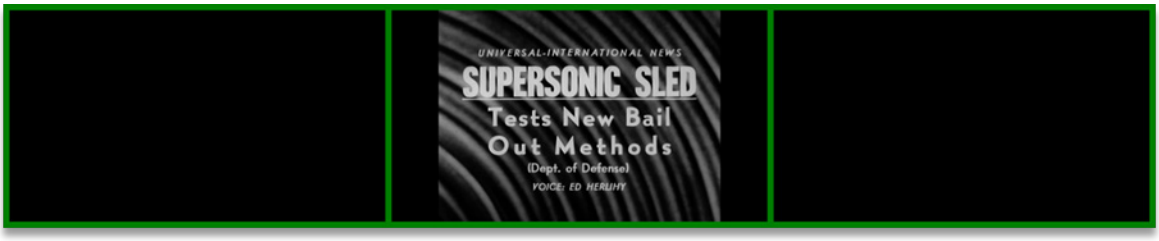

Figure 4. In-game Video Scaled Up Until it Fits Properly into One Display without Stretching

Sometimes when running in landscape orientation Surround modes, games place parts of the user interface on displays that are in the peripheral regions of a user's eyesight, negatively impacting the experience. This is especially undesirable if a user typically wants to use or read these interface components often, as is the case with a game's map, health/damage indicators, or controls (Figures 5 and 6).

Figure 5. HUD Components Inconveniently Placed on Outside Displays

Figure 6. HUD Components Conveniently Placed on Center Display

In contrast, when a user is running in portrait orientation mode, it may actually be preferable to allow interface elements onto the outside displays since they are closer together than in landscape mode, and portrait mode is very close to the 16:9 or 16:10 aspect ratios that many single displays use. As a result, the outside edges of the outside displays are often within a comfortably viewable area (Figures 7 and 8).

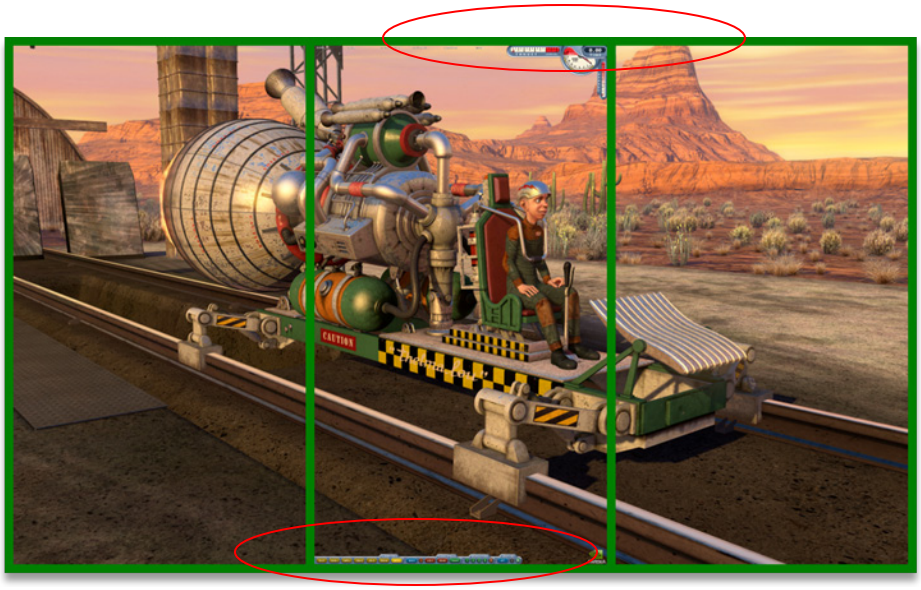

Figure 7. HUD Inconveniently Restricted to Center Portrait-Oriented Display

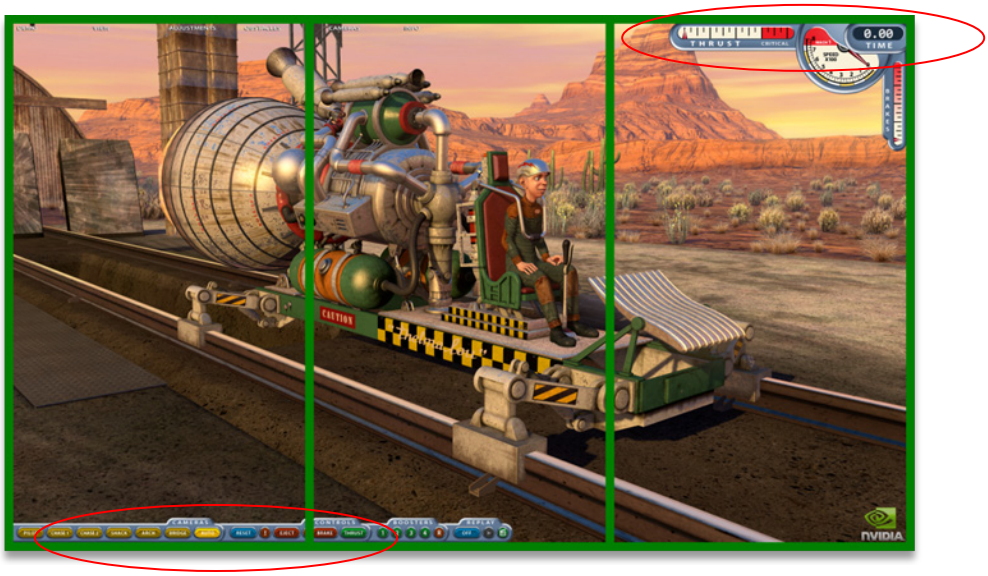

Figure 8. HUD Conveniently Spans Three Portrait-Oriented Displays

In either landscape or portrait orientation, it is usually not desirable to have individual parts of the interface spanning multiple displays, not only because they will be split between displays in normal Surround modes, but also because they may end up partially or wholly occluded in bezel-corrected display modes (Figure 9).

Figure 9. UI Components at Top and Bottom of Screen are Occluded by Bezel in Bezel-Corrected Display Mode

NVIDIA Surround technology presents multiple displays as one large display to the game and it can be difficult for a developer to know where the bezels of the actual physical displays reside with respect to the final rendered image. It can also be difficult to know if the current display mode is *bezel-corrected*, which results in some pixels being virtually hidden behind the bezel for the sake of having rendered objects appear to line up across the multiple displays. Bezel-correction is designed to create the perception of looking through a window with panes, rather than just seeing the normal display separated by display bezels. In this way, better immersion is afforded at the expense of visibility of some of the rendered pixels (Figures 10 and 11).

Figure 10. Lack of Bezel-Corrected Mode Shows Split/Elongated View Between Left and Center Surround Displays

Figure 11. Bezel-Corrected Mode Shows Continuous View Between Left and Center Surround Displays but Occludes Some Pixels

The recommended way to avoid issues with menus and video rendering in Surround modes is to restrict them to the center display, so that they are not stretched in any way. Alternatively, you could scale these elements until they maximize either the width or height of the active display mode, thus enlarging the display area covered while retaining correct proportions. Similarly, the recommended way to avoid the user interface placement issues when in landscape mode is to restrict user interfaces to the center screen, by making sure that none of the rendered interface elements are placed beyond certain boundaries where a display bezel may be. This way interface elements are not rendered on the outside displays, or split between displays.

In Portrait orientation modes, it is generally preferred to allow interface elements to be placed on the outside displays, since the aspect ratio is close to that of 16:9 or 16:10 modes. However, it is still ideal to avoid having them split (or occluded in the case of bezel-corrected modes) between displays (Figure 12).

NVIDIA does not recommend developers look for a display number since the NVIDIA Control Panel allows users to arrange the displays in any order they desire. Instead, it is recommended that you use NVAPI for the most robust programmatic placement of menu, video, and interface elements for Surround configurations. Here is an example solution where we are querying for "visible rects" and then restricting the UI to the visible part of the center display:

```
NvAPI_Status retVal;
NvU32 displayID = 0;

//We assume that displayArgs, a struct of device properties such as
//resolution and fullscreen state, has already been initialized with
//valid data. Also, we assume that Rect is some previously defined
//struct type which contains member variables to describe location and
//size of an arbitrary display rect.
Rect safeUIRegion;

safeUIRegion.x = 0;
safeUIRegion.y = 0;
safeUIRegion.width = displayArgs.width;
safeUIRegion.height = displayArgs.height;

retVal = NvAPI_DISP_GetGDIPrimaryDisplayId(&displayID);

if (retVal == NVAPI_OK) {
    NV_RECT viewports[NV_MOSAIC_MAX_DISPLAYS];
    NvU8 isBezelCorrected;

    retVal = NvAPI_Mosaic_GetDisplayViewportsByResolution(displayID,
        displayArgs.width, displayArgs.height, viewports,
```

```
            &isBezelCorrected);

    if (retVal == NVAPI_OK && displayArgs.isFullscreen) {
//NVIDIA Surround is enabled.

        print("NVIDIA Surround topology found.\n");

        NvU32 displayCount = 0;

        while (viewports[displayCount].top !=
            viewports[displayCount].bottom)   displayCount++;

//Ensure there is more than one viewport rect and that there is an odd
//number of displays in the Surround configuration if we want to
//restrict the UI to the center display.
        if ((displayCount > 1) && (displayCount & 1)) {
            bool isSingleDisplayRow = true;

            for (i = 1; i < displayCount; i++) {
                if ((viewports[0].top != viewports[i].top) ||
                    (viewports[0].bottom != viewports[i].bottom)) {
                    isSingleDisplayRow = false;
                    break;
                }
            }

//Check to make sure that the displays are in a single row together.
            if (isSingleDisplayRow) {
                NvU32 centerDisplayIdx = displayCount / 2;

//Check which rect dimension of the center display is larger. If the
//height is larger than the width, we assume the displays are arranged
//in portrait mode and we leave the UI spanning multiple displays (and
//possibly occlude parts of the UI in a bezel-corrected configuration).
//If the width is larger than the height, we store that center
//display's viewport coordinates and size so we can restrict the UI to
//that region for the user's convenience and because we know the UI was
//probably designed with this display orientation/aspect ratio in mind.
//Alternatively, we could choose to just use the rect regions returned
//by NvAPI_Mosaic_GetDisplayViewportsByResolution to place UI elements
//on more than one display, while still restricting them to the visible
//regions.
                NvU32 centerWidth = viewports[centerDisplayIdx].right
                    - viewports[centerDisplayIdx].left + 1;
                NvU32 centerHeight = viewports[centerDisplayIdx].bottom
                    - viewports[centerDisplayIdx].top + 1);

                if (centerWidth > centerHeight) {
                    print("Restricting UI to center display.\n");
                    safeUIRegion.x = viewports[centerDisplayIdx].left;
                    safeUIRegion.y = viewports[centerDisplayIdx].top;
                    safeUIRegion.width = centerWidth;
                    safeUIRegion.height = centerHeight;
```

```
                }
            }
        }
    }
    else if (retVal == NVAPI_MOSAIC_NOT_ACTIVE) {
        print("NVIDIA Surround is not enabled.\n");
    }
    else {
        print("NvAPI_Mosaic_GetDisplayViewportsByResolution call
            failed.\n");
    }
}
else {
  print("NvAPI_DISP_GetGDIPrimaryDisplayId call failed.\n");
}

DrawUI(safeUIRegion);
```

Another way of doing this which does not require the use of NVAPI, but is less robust, is to assume the user has a configuration that is three displays wide based on the aspect ratio of the selected resolution, or let the user select an option in-game to enable this mode. This example defines rough bezel-correction-safe horizontal boundaries for programmatic placement of menu, video, and interface elements:

```
SingleDisplayWidth = TotalWidth / 3;

//Covers cases for Surround Landscape mode comprised of 5:4 displays
//and up.
if AspectRatio >= 3.75 {
//Define minimum interface x value as the left boundary of the center
//display + 10% of a single display's width (for bezel-corrected
//safety).
    HUD.minX = SingleDisplayWidth + (SingleDisplayWidth / 10);

//Define maximum interface x value as the right boundary of the center
//display + 10% of a single display's width (for bezel-corrected
//safety).
    HUD.maxX = (2 * SingleDisplayWidth) – (SingleDisplayWidth / 10);
}

else {
//Assume portrait mode, allowing interface elements on all 3 displays,
//but make sure there is no interface element within 10% of 1/3 and 2/3
//of the total display width.
    HUD.maxDisp1 = (SingleDisplayWidth) – (SingleDisplayWidth / 10);
    HUD.minDisp2 = (SingleDisplayWidth) + (SingleDisplayWidth / 10);
    HUD.maxDisp2 = (2 * SingleDisplayWidth)
                        – (SingleDisplayWidth / 10);
    HUD.minDisp3 = (2 * SingleDisplayWidth)
                        + (SingleDisplayWidth / 10);
```

```
//Bezels are located between HUD.maxDisp1 and HUD.minDisp2 as well as
//between HUD.maxDisp2 and HUD.minDisp3
}
```

Figure 12. Bezel-Correction-Safe Areas to Place UI Elements so They are not Occluded by Bezels for Common Resolutions and Bezel Offsets

Yet another solution for these placement issues is to allow the user to have some element of control over the interface so they can offset interface elements by some amount or control their exact location. This could function by means of in-game controls or by exposing interface object positions in a text configuration file (which, for example, could be edited by third party tools or even manually by users).

MOUSE INTERACTION WITH USER INTERFACE

In certain Surround modes, some games have problems with mouse cursor interaction with the user interface. Typically, these problems are caused by the visual representation of the mouse cursor rendering in a different location than the game believes it to be, or hit-targets for the UI elements failing to scale/move with their corresponding visual components.

Because the mouse cursor being rendered in the incorrect location will likely break the ability to play a game, it is important to verify that this works correctly in Surround resolutions, for both landscape and portrait Surround configurations.

In order to make sure that user interface hit-targets work correctly as resolution scales upwards, it is recommended that you either restrict your UI and menu elements to the center display using the methods mentioned in the *User Interface/Menu/Pre-rendered Video Placement* section, or scale up the hit-targets in the same way that their corresponding visual components are scaled, for all resolutions.

DISPLAYING ENUMERATED RESOLUTION MODE LIST IN MENUS

It is important to display a full list of available modes in a game's graphics options menu because NVIDIA Surround technology adds many new resolution modes (such as bezel-corrected Surround resolutions) to the modes traditionally supported by the NVIDIA driver. In some games, the resolution mode list is designed as a drop down box that does not scroll, and as a result, many resolutions in the longer mode list end up above the top edge or below the bottom edge of the display. This prevents a user from being able to select those resolutions. NVIDIA recommends offering users the option to select all enumerated resolutions from scrollable lists or other compatible control types.

Additionally, to ensure all modes are able to be offered to users, it is important to ensure that a game does not have some pre-defined maximum number of resolution mode list entries in the menu interface, which truncates the list of enumerated modes that is offered to users.

Lastly, it's important to make sure that enumerated resolutions are not filtered in some way, like by only displaying resolutions that are a multiple of some number. This is because Surround users are able to specify arbitrary amounts of bezel correction, and as a result, they can create odd value resolutions (like 6011x1080) that a developer cannot predict.

ASPECT RATIO GROUPING IN MENUS

Some games try to make resolution choice easier for users by enumerating the available resolutions, and then sorting them into pre-defined aspect ratio groups in an in-game menu's user interface. For games that do this, it is recommended that one of the following solutions is used, in light of all the new aspect ratio configurations that are possible:

- Add new groups for common Surround aspect ratios (for both the Landscape and Portrait orientation cases, as seen in *Table 1*) to these pre-defined aspect ratio groups.

or

- Place resolutions with non-standard/unrecognized aspect-ratios all together into one general additional aspect ratio group, so that users can still find all possible surround resolutions even if they do not fit into groups defined by the game.

GENERAL RENDERING CONSIDERATIONS

SUPPORT AN EXPANDED VIEW WITH SURROUND

For optimal support of Surround resolutions in landscape mode, it's important to expand the field of view to allow for more of the rendered scene to be displayed. When this is done correctly, the result is known as *Horizontal+*, which means that Surround resolutions result in an expanded horizontal viewport into the rendered scene, with more of the scene visible than was previously visible without Surround modes (Figures 13 and 14).

Figure 13. Standard 16:10 Resolution

Figure 14. Surround Resolution, Game Renders Horizontal+

However, when a game does not increase the field of view based on the increased aspect ratio of Surround landscape modes (despite even having correctly enumerated and created a device at a Surround resolution) the game usually compensates by either zooming in on or stretching the rendered image, both of which are undesirable and should ideally be avoided.

Games which zoom in on the scene to compensate for lack of field of view when filling in the expanded resolution are known as a *Vertical-* (Figure 15). In lieu of adding horizontal visibility, these games sacrifice vertical visibility in order to display the scene at the correct aspect ratio for the selected Surround resolution. In a 1x3 Surround landscape configuration, the result of Vertical- rendering is a zoomed-in perspective, where the top and bottom of the rendered scene have effectively been cropped by the edges of the monitor. In this case, not only is there no new horizontal visibility, there is also less vertical visibility than in standard single display configurations.

Figure 15. Surround Resolution, Game Renders Vertical-

Games that render *stretched* in Surround landscape modes tend to render at a lower resolution than the device is created at, and then stretch the image in the horizontal Surround-expanded dimension. The result is an image that is forced to cover all pixels of all of the displays, but is also distorted, in some cases greatly (Figure 16).

Figure 16. Surround Resolution, Game Renders Stretched

To avoid rendering Vertical- or stretching, utilizing either of the following solutions is recommended:

- The field of view for each dimension should be calculated as a function of the aspect ratio of the display configuration. Refer to *Table 1* for a list of recommended aspect ratios.
- Expose field of view controls to the user, ideally through the game's user interface. If field of view controls cannot be exposed through the user interface, it is recommended that they be exposed through game configuration files in a format that is text-editable. This allows users and 3rd-party tools to manipulate the field of view values for Surround configurations.

SUPPORTING SCREEN-SPACE RENDERING EFFECTS

In some games, screen-space rendering is used for rendering effects on top of the 3D scene. It is important to make sure that these effects work properly in Surround configurations by making sure that they are not limited to the primary display.

For example, sometimes when games render screen-space effects such as glows or bloom over the scene in Surround configurations, they will not correctly expand the effects onto the outside displays, leading to cases where certain objects are rendered on the outside displays, but their accompanying glow or bloom effects are still rendered somewhere within the center display.

Similarly, screen-space effects are often used for situations where the user's view into a scene is intentionally occluded in some way. A common use of this kind of effect is seen when zooming into a sniper/scoped mode with a weapon in a game, simulating the limited view one sees when looking through a real weapon scope. For this effect, usually only part of the scene is visible on a particular part of the display. However in Surround configurations, sometimes games will only occlude the proper parts of the center display, and not occlude the outside displays properly. This both negatively impacts immersion and gives an unfair advantage to some users.

To avoid these problems, it is important to make sure that when screen-space effects are used, they are correctly expanded across all Surround displays by making sure that the effect is taking the active resolution size into account, and not artificially limiting the effect to some pre-defined size or space.

Additionally it is important to make sure that screen space rendering effects are not linked to assumptions about resolutions. For example, if a pixel shader relies on doing integer math with an even value resolution (like 6010x1080) and a user has an odd value resolution (like 6011x1080), the effect that the pixel shader is used for may break.

Notice

HDMI

ROVI Compliance Statement

OpenCL

Trademarks

Copyright