CTR Developer's Guide

Graphics: ULCD and Realism Priority Mode

Version 2010/11/11

The content of this document is highly confidential and should be handled accordingly.

CTR-06-0052-001-A
Released: November 17, 2010

Table of Contents

Figures

1 Introduction

The CTR SDK provides the ULCD Library to support 3D viewing. Functions of the library are accessed through an instance of the StereoCamera class. A StereoCamera object has two modes, called *application priority* and *realism priority*. The chosen mode determines how the View and Perspective matrices are generated for each eye.

This document will provide a clearer look at the mechanics of `CalculateMatricesReal()`. It is intended as a supplement to the documentation found in the SDK. Please read *Description of the ULCD Library* first. It may be found in the install directory under `documents/TechincalNotes/AboutULCDLibrary`.

Given the base camera specified by the application, StereoCamera can generate two new matrices, one for the left eye and one for the right. StereoCamera has two modes of operation, briefly summarized in the following sections.

1.1 Realism Priority Mode

In realism priority mode, new cameras are generated based on the physical dimensions of the LCD screen and the application's camera settings. The field of view, or FOV, will become fixed to a shallow angle and the near and far planes will be adjusted to fit the new frustum. With these new settings, the depth range that maintains parallax within Nintendo's Lotcheck guidelines is fairly small. Therefore this mode may be best suited for scenes that have shallow depth, like dioramas.

Realism mode is selected by generating stereo cameras with `CalculateMatricesReal()`.

1.2 Application Priority Mode

Application mode preserves the application's camera settings, such as FOV and clipping planes. A maximum parallax value can be set by the application and this value will be used for parallax at the far plane. Since this mode preserves the supplied view frustum, it is suited for general 3D scenes and probably the only option for first-person or over-the-shoulder cameras.

The "realistic" part of realism mode is the mathematics that go into calculating the field of view. By selecting the appropriate settings, application mode can mimic realism mode. Therefore, application mode should not be seen as unrealistic.

Application mode is selected by generating stereo cameras with `CalculateMatrices()`. Note that `SetMaxParallax()` only works for application priority mode. It does not affect the results of `CalculateMatricesReal()`.

Note: The guideline values in this document reflect the guidelines available at the time the document was last updated, November 11th, 2010. These values are only meant as a reference since the guidelines may change. Please refer to the actual guidelines documentation for current information

2 About Realism Mode

`CalculateMatricesReal()` attempts to create a realistic scene by matching the expected field of view of a human looking at the LCD screen. *Description of the ULCD Library* presents several real world measurements. Using these measurements it is possible to calculate the approximate vertical angle of vision occupied by the physical LCD screen.

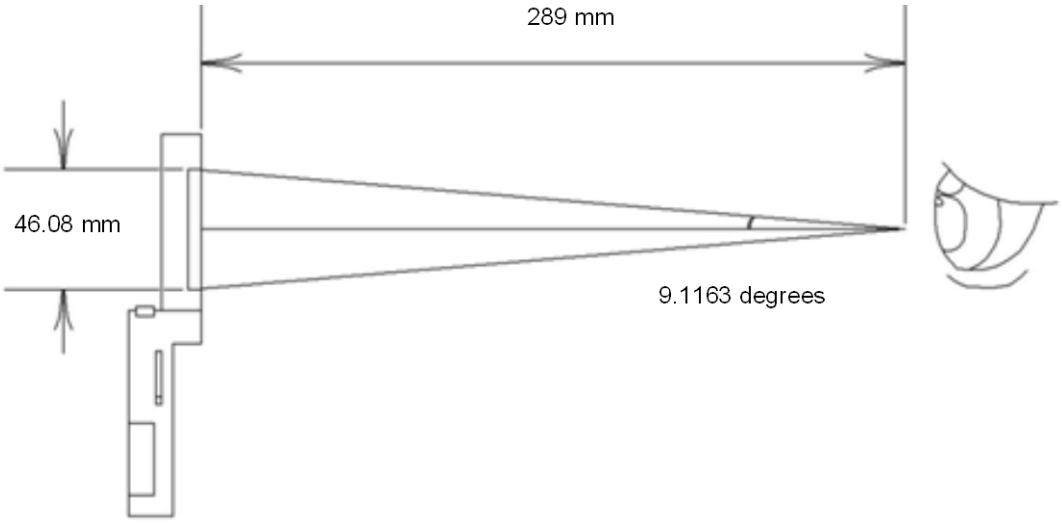
$Dist_{eye}$ = 62 mm, the distance between the player's eyes, ***I*** in *Description of the ULCD Library*

Len_{disp} = 46.08 mm, the length of the shortest side of the upper screen, the height

$Dist_{e2d}$ = 289 mm, the distance from the player's eyes to the surface of the display

FOV_{real} = 9.1163 degrees, this angle becomes the camera's FOV in realism mode

Figure 2-1 Determining the Realistic Field of View



The matrices output by `CalculateMatricesReal` represent a frustum with a 9.1163 degree vertical field of view. Unless the application's camera is using a perspective projection with the same FOV this will result in the virtual camera moving forward or backward in the scene.

2.1 Understanding the Base Plane

The *base plane* corresponds to the surface of the LCD screen. Objects behind the base plane will appear to sink into the LCD. Objects in front of the base plane will appear to pop out of the LCD. Objects at the base plane will appear as if they are just on the surface of the LCD.

Both `CalculateMatrices*()` functions take several parameters: Input matrices describing the application camera, a depth level, and a scaling factor. Depth level is the distance in virtual space from the camera to the base plane, this will be referred to as $Depth_{level}$. The scaling factor reduces the amount of parallax separation. Information presented here assumes that scaling factor is 1.0 (the maximum value).

Though it is called a plane, we are actually concerned with the rectangular slice that intersects the view frustum specified by the application before calling `CalculateMatricesReal()`. This rectangular 2D slice is preserved when `CalculateMatricesReal()` moves the camera. Given the distance to the base plane and the camera's vertical field of view it is possible to find the virtual height of the base plane, H_{base}. The ratio between the base plane's height and the physical height of the screen is used to scale other measurements from real distances to corresponding distances in the virtual scene.

$$H_{base} = 2 * Depth_{level} * \tan(FOV / 2)$$

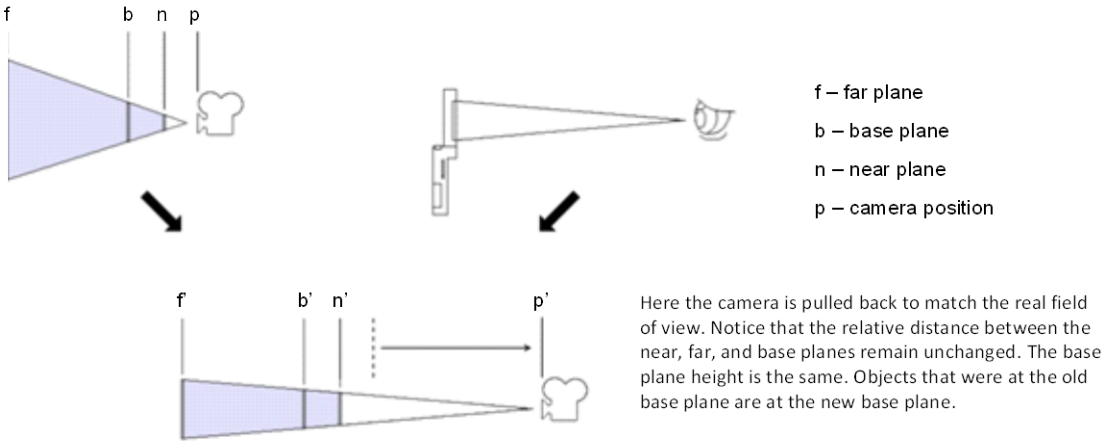
$$Scale_{r2v} = H_{base} / Len_{disp}$$

`CalculateMatricesReal()` uses this scale to find the virtual distance which represents the assumed distance between the LCD surface and a player's eye in real space.

$$Depth_{level\_new} = Dist_{e2d} * Scale_{r2v}$$

The base camera is then adjusted to be $Depth_{level\_new}$ units away from the base plane. A new viewing frustum is created from the camera's new position through the fixed base plane. This new frustum will have the same field of view, about 9.1163 degrees, as the angle occupied by the real LCD screen at $Dist_{e2d}$. `CalculateMatricesReal()` attempts to preserve the relative distance between the base plane and the near and far clip planes. It is useful to think of these locations as fixed in space with only the camera position moving. If the camera ends up in an odd location, such as behind the near plane, values for the near and far clipping planes will be adjusted. See Equations 3-9, 3-10, 3-11, and 3-12 in *Description of the ULCD Library* for more information.

Figure 2-2 Camera Moves to Emulate Field of View



If the camera's field of view is already 9.1163 degrees, `CalculateMatricesReal()` does not change the view frustum or move the base camera. If the camera's FOV is greater than 9.1163 degrees, the new camera position will be moved away from the base plane as the FOV is shrunk. If the camera's FOV is smaller than 9.1163 degrees, the new camera position will be moved closer to the base plane as the FOV is expanded. If the distance between the far plane and the base plane is large relative to $Depth_{level}$, this could have a great impact on the area of the scene within the new view frustum.

2.2 Dealing with Maximum Parallax Guidelines in Realism Mode

Parallax refers to the separation between the screen position of an object rendered for the left eye and its position when rendered for the right eye. Current guidelines recommend a maximum parallax of 10 mm going into the screen and 5mm popping out for viewing comfort.

These real distances can be scaled to a virtual distance. Given the display height of 46.08 mm and the vertical resolution of 240 pixels, a single pixel is 0.192 mm in width and height. Furthermore, the StereoCamera instance provides a member function to retrieve the parallax offset at a given distance using the results of the most recent call to `CalculateMatrices*()`. `GetParallax()` returns a ratio of the parallax distance to screen width. This value is the separation between the central position and an offset to the either side. So the full distance between left and right images is twice the result of `GetParallax()`.

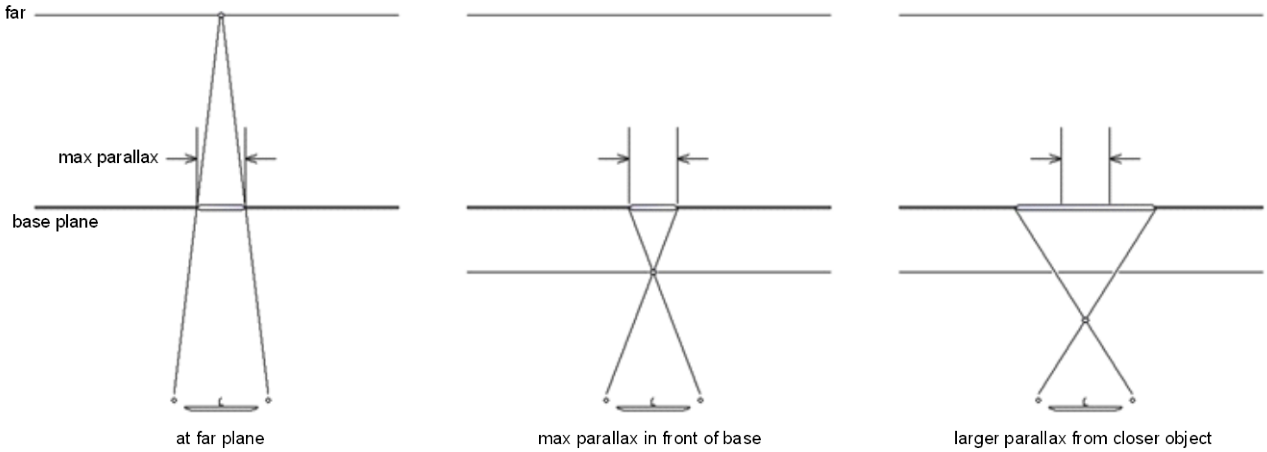
Recommended max in parallax = 10.0 mm = 52 pixels = 0.130 * Width

Recommended max out parallax = 5.0 mm = 26 pixels = 0.065 * Width

As objects behind the base plane approach the far plane, the parallax offset increases and is at its maximum value at the far plane. In realism mode, one may restrict the maximum parallax for objects behind the base plane by setting the far plane such that the parallax for objects at the far plane is the desired maximum parallax.

For objects in front of the base plane, there is no easy solution. As objects approach the camera the parallax offset approaches infinity. Setting the near plane to be the point where objects reach maximum parallax does not work. Objects would be clipping the near plane. It will take some work on the application side to arrange the scene so objects do not regularly cross this boundary. One way to avoid the problem is to make the near clipping plane the base plane, so all 3D is into the screen. Alternately, the camera can be positioned some distance away from objects in the scene.

Figure 2-3 Maximum Parallax on Each Side of the Base Plane



2.3 Equations for Finding the Distance That Will Give a Particular Parallax

The distance to the base plane after a call to `CalculateMatricesReal()` can be found without actually calculating the matrices. Using this value, one can find the parallax at a given distance, or the distance which provides a desired parallax. With the equations below, find the distance to maximum parallax in front of and behind the base plane. Keep the scene within these distances to meet the parallax separation guidelines.

$Depth_{level\_new} = 289 * Scale_{r2v}$, where 289 mm is the assumed distance from eye to LCD surface

$Dist_{eye}$ = 62 mm, the assumed separation between a player's eyes

$Parallax_{real}$ = On screen parallax in mm

$Far_{max\_parallax}$ = Far distance with parallax = $Parallax_{real}$, $Far_{max\_parallax} > Depth_{level\_new}$

$Near_{max\_parallax}$ = Near distance with parallax = $Parallax_{real}$, $Near_{max\_parallax} < Depth_{level\_new}$

$$Far_{max\_parallax} = Depth_{level\_new} * (1 / (1 – (Parallax_{real} / Dist_{eye})))$$

$$Near_{max\_parallax} = Depth_{level\_new} * (Dist_{eye} / (Dist_{eye} + Parallax_{real}))$$

For example, using the guideline value of 5.0 mm and 10.0 mm, two distances are found.

$Far_{max\_parallax} = Depth_{level\_new} * (1 / (1 – (10.0/62)) = Depth_{level\_new} * 1.192$

$Near_{max\_parallax} = Depth_{level\_new} * (62 / (62 + 5.0)) = Depth_{level\_new} * 0.925$

Objects between these two distances will meet parallax recommended by the guidelines. It can be seen from the scale values that there is little room for deviation from the base plane. This is the motivation for recommending realism mode for diorama-like scenes.

Since the distance between the base plane and the far clip plane remains the same, one can use $Far_{max\_parallax}$ to set the initial far clip plane to end up at $Far_{max\_parallax}$ after `CalculateMatricesReal()`. One can also use $Near_{max\_parallax}$ to find a reference plane to keep objects within maximum parallax. Note this does not directly correspond to the near clip plane. Use this to create the initial frustum.

$Far_{clip} = Depth_{level} + (Far_{max\_parallax} – Depth_{level\_new})$

$Near_{reference} = Depth_{level} – (Depth_{level\_new} – Near_{max\_parallax})$

3 Summary

In this document, usage of the ULCD library for stereoscopic display has been discussed. Focus was placed on the mechanics of realism mode. You should now have a better understanding of what happens to the view frustum after a call to `CalculateMatricesReal()` and an understanding of how to meet CTR guidelines for parallax limits when using realism mode. With this knowledge you can make an educated choice between stereo camera modes and develop applications that meet specific parallax guidelines.

Questions? Comments? Please send us any feedback you have by emailing support@noa.com and using a subject line beginning with: [CDG]

We are happy to take your suggestions for improving our documentation.

Sincerely,
Software Development Support Group
Nintendo of America

Revision History

Revision Date	Description
2010/11/17	Initial version.