Warping Library

Warping Overview

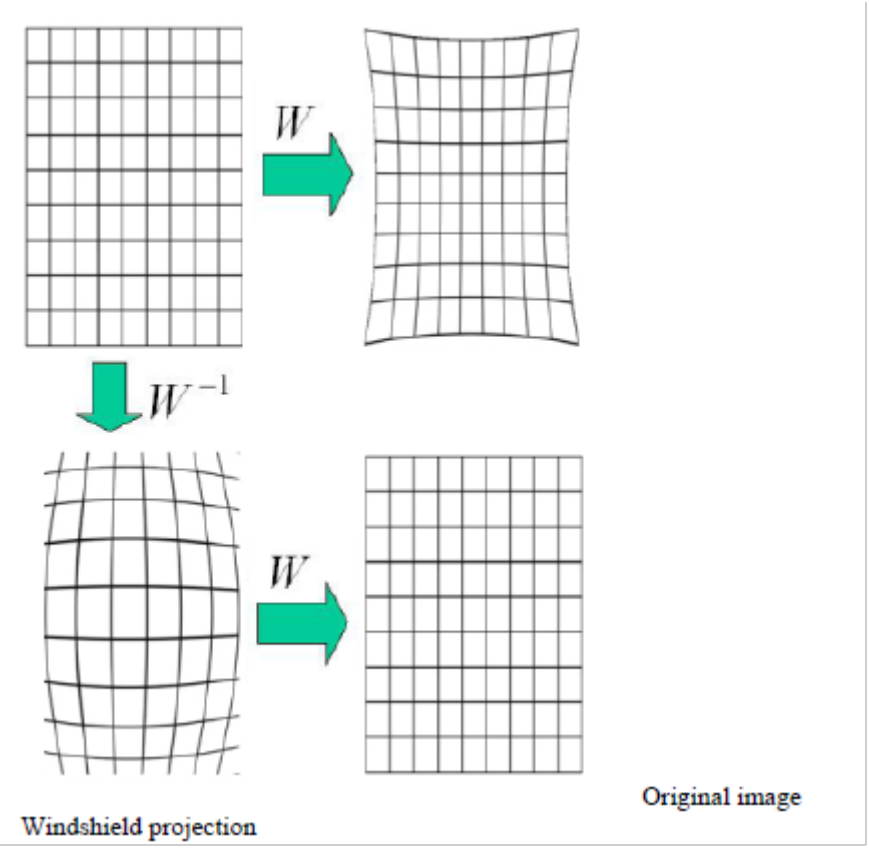
This section explains the implementation of the Warping Library. It is used to adjust the image to be displayed so that it can be projected to a windscreen (which in the end shall result in a non-distorted image presented to the user).

Additionally, the Warping Library can apply graphic transformations like flipping and rotation to the image during the warping process.

Introduction

The Head-up display (HUD) projects information onto the windscreen glass. An image projected to the windscreen is distorted because the glass is not flat. For correcting this image distortion, warping process shall be used and the original image shall be prepared so that the driver can observe it on the windscreen linearly.

The basic principle of the Warping Library is to map a texture onto a grid, which's form compensates for the deformation of the windscreen when the information is projected on the HUD. The texture that is used is the readily rendered content for the HUD.

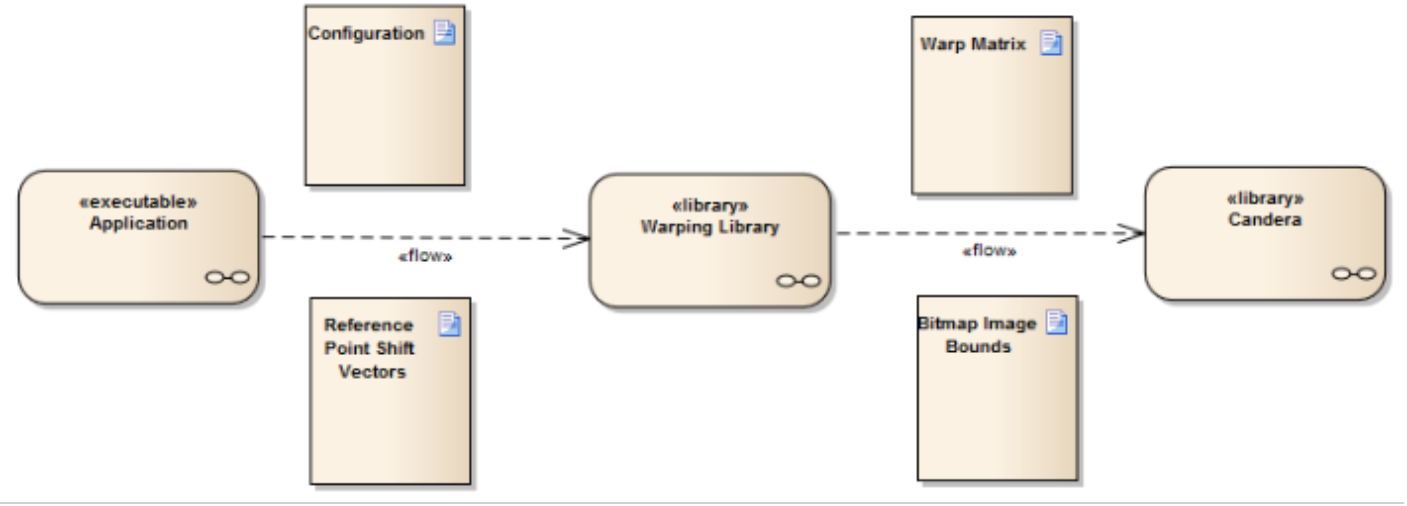


CGI Studio Warping is intended to be used for correcting such image distortions. The following two components are involved in the process:

- **Warping Library**: used for computing of the warping matrix
- **Candera**: used to map a displays image (or part of it) as texture to a mesh created from the warping matrix information and displays the correspondingly distorted result.

CGI Studio Controller Integration Tool

The presented figure shows an overview of the information flow when using the CGI Studio Warping components.

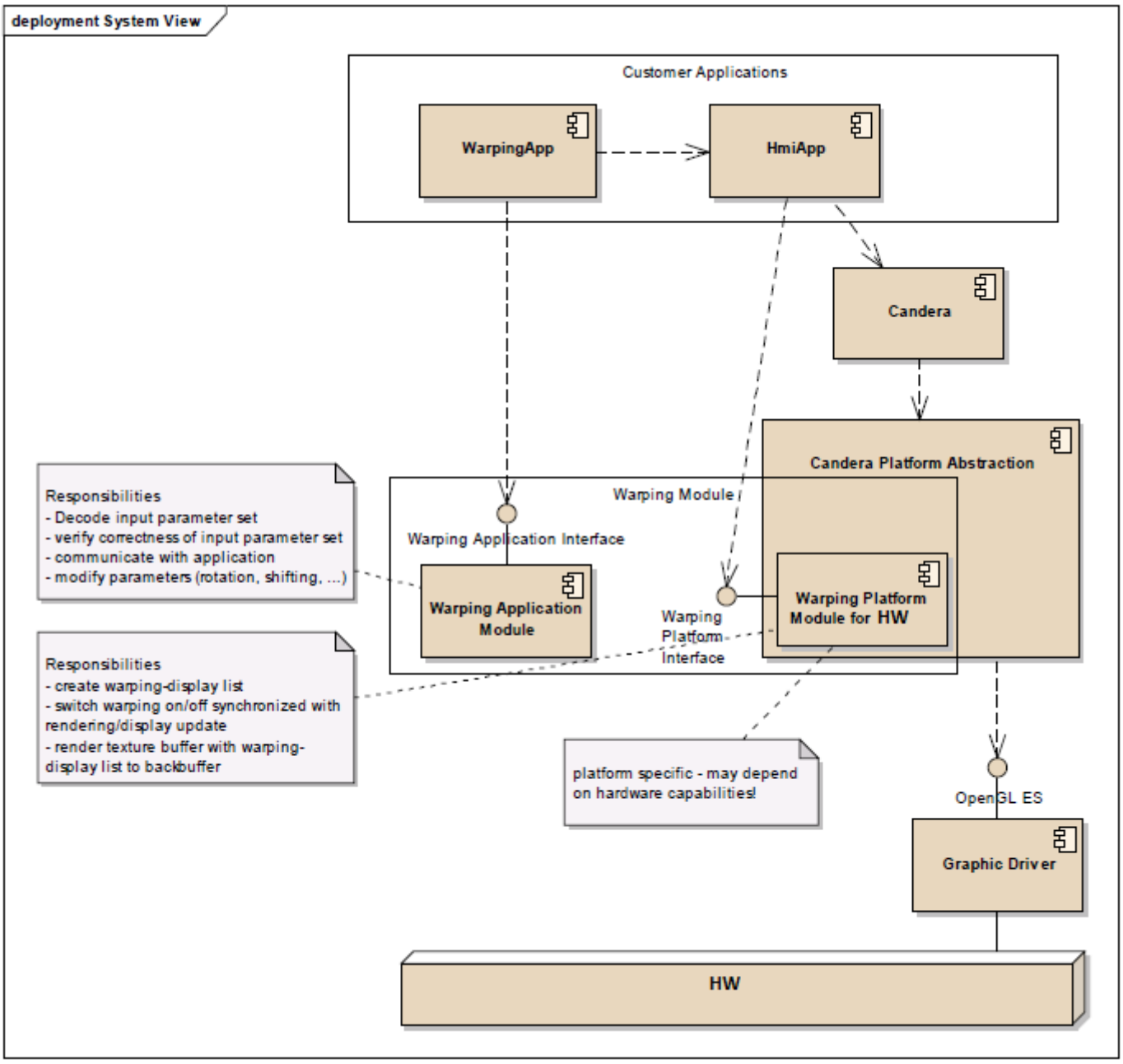


Three components can be identified, that together, implement the warping:

- **Application**: Controls execution, prepares and passes information between the other two components
- **Warping Library**: Computes the warping matrix and image bounds of the are to be warped
- **Candera**: Applies the warping matrix and image bounds to the image rendered to a display, thus warping the desired portion of the overall display content.

System View

Below figure shows the location of the warping library in the system and which other components use it. Globally, the warping library consists of two parts, the warping application module and a warping platform module. The platform module is platform dependent - a version for Windows host platform and a version for target hardware are available (although several platforms share the same implementation). Responsibility for each part is described in the system view.


deployment System View
Customer Applications
WarpingApp
HmiApp
Candera
Candera Platform Abstraction
Warping Module
Warping Application Interface
Warping Application Module
Warping Platform Interface
Warping Platform Module for HW
Responsibilities
- Decode input parameter set
- verify correctness of input parameter set
- communicate with application
- modify parameters (rotation, shifting, ...)
Responsibilities
- create warping-display list
- switch warping on/off synchronized with rendering/display update
- render texture buffer with warping-display list to backbuffer
platform specific - may depend on hardware capabilities!
OpenGL ES
Graphic Driver
HW

Warping Functionality

The Warping Library supports the following functionality:

- Warp matrix computation
- Warp matrix adjustment
- Reference Point Shift Vector encoding and retrieval
- Warp image bounds computation

Warp Matrix Computation

Input:

- Shift Vectors
- Configuration information (please refer to Configuration of warping parameters)

Output:

- Warp Matrix

Algorithm:

- Decode and cache the reference point shift vectors.
- Computation of warp matrix from reference points and corresponding shift vectors. The reference points are distributed over the whole bitmap area according to the specifications in the parameter list.
- Rotation of warp matrix. The rotation of the warped reference points take place aroung the center point of the warped bitmap. (please refer to Rotation and Scale)
- Scaling of warp matrix to illuminated area, if outside of illuminated area. (please refer to Rotation and Scale)
- Normalize of warp matrix. Candera uses normalized coordinates in range [0..1] for rendering, therefore the warp matrix data has to be normalized in this range.

Interface:

CalculateWarpMatrix(const stc_warp_configuration_parameter_t* const pstcWarpConfigParams, const stc_warp_parameter_set_t* const pstcWarpRefPoints, stc_warp_matrix_t* pstcWarpMatrix)

Warp Matrix Adjustment

Input:

- Configuration information.
- Adjustment configuration (please refer to Adjustments).

Output:

- Warp Matrix

Algorithm:

- Adjustment of reference point shift vectors according to the specified adjustment configuration structure using the according mathematical formulas.
- Computation of warp matrix from reference points and corresponding shift vectors.
- Rotation of warp matrix.
- Scaling of warp matrix to illuminated area, if outside of illuminated area.
- Normalize of warp matrix.

Interface:

AdjustWarpMatrix(const stc_warp_configuration_parameter_t* const pstcWarpConfigParams, const stc_warp_adjust_parameter_t* const pstcWarpAdjustParams, en_warp_bool_t* pbRefPointsChanged, stc_warp_matrix_t* pstcWarpMatrix)

Reference Points Encoding and Retrieval

Input:

- Internally cached configuration
- Internally cached shift vectors

Output:

- Encoded bitmap image shift vectors

Algorithm:

- Encode shift vectors.
- Write shift vectors to application supplied memory location.

The current active reference point shift vector set is retrieved from the warping library. Shift vectors are encoded according to the provided configuration.

Interface:

GetRefPoints(stc_warp_parameter_set_t* pstcWarpRefPoints)

Warp Image Bounds Computation

Input:

- Configuration information.

Output:

- Bitmap image area bounds in normalized coordinates

Algorithm:

- Calculate area bounds based on the provided configuration information:
  - x = (bitmapAreaShiftX - bitmapAreaX/2) / displaySizeX
  - y = (bitmapAreaShiftY - bitmapAreaY/2) / displaySizeY
  - width = bitmapAreaX / displaySizeX
  - height = bitmapAreaY / displaySizeY

The image area bounds are normalized as required by Candera, as it uses normalized coordinates in range [0..1] for rendering.

Interface:

ComputeWarpImageBoundsFromConfig(stc_warp_configuration_parameter_t* config, WRP_SFLOAT* x, WRP_SFLOAT* y, WRP_SFLOAT* width, WRP_SFLOAT* height)

Using Warping with Candera

Following are the display properties that required adjustment to use warping with Candera:

- Enable warping - this should be done before the native handle is available
- Set warp matrix
- Set warp image bounds (optional)
- Apply display settings

Candera::Display class uses a transaction based execution model for setting properties. Several changes can be queued. Changes have to be applied explicitly via a call to Display::ApplyChanges() . Warping matrix data from the warping library can be used directly in Candera.

Execution Overview

Warping is executed in the following steps:

- Warping mesh creation
- Rendering of display image to texture
- Texture mapping of rendered image aperture specified by warp images bounds to the warping mesh

Display content is rendered to a texture. This texture is clipped according to the warp image bounds and the resulting texture is mapped to the warp mesh and rendered in 3D using OpenGL and a very simple shader program.

Mesh and Texture Creation

Upon rendering of a displays content, Candera checks, if:

- Warping has been en- or disabled
- Warp matrix has changed
- Image bounds have changed

If any of these conditions holds, the currently used warping mesh is unloaded. A new warping mesh is computed from the warp matrix as well as the image bounds and uploaded. Vertices in the warp mesh directly map to the corresponding values in the warp matrix. Texture coordinates for vertices are computed from warp mesh vertices as well as warp image bounds. The texture is mapped to the warping mesh and distorted corresponding to the position of the of the warping mesh vertices.

Configuration of warping parameters

The configuration information passed to the warping library is used to check constraints and as basis for computations.

Overall configuration parameter structure

The following table details the overall configuration parameter structure and their limits.

Name	Range	Description	Example Value
NumBitsPerParam	1..16	Number of bits used for each parameter (bit length of each parameter, contains pixel fractions)	12
PixResolution	**0**: $^1/_{16}$ px **1**: $^1/_8$ px **2**: $^1/_4$ px **3**: $^1/_2$ px **4**: 1 px **5**: 2 px **6**: 4 px **7**: 8 px	Resolution of warping parameters.	0 ($^1/_{16}$ px)
DisplaySizeX	0..655535	Display size in horizontal direction (in pixel).	480
DisplaySizeY	0..655535	Display size in vertical direction (in pixel).	240
BitmapAreaX	0..655535	Number of Bitmap-pixels in horizontal direction. Reference points are equally distributed over this range in x-direction. **Condition:** BitmapAreaX ≤ DisplaySizeX	480
BitmapAreaY	0..655535	Number of Bitmap-pixels in vertical direction. Reference points are equally distributed over this range in y-direction. **Condition:** BitmapAreaY ≤ DisplaySizeY	240

BitmapAreaShiftX	0..65553 5	Shift of the bitmap area center point in horizontal direction. **Condition:** (BitmapAreaX/2+BitmapAreaShiftX≤DisplaySizeX) AND (BitmapAreaX/2– BitmapAreaShiftX ≥0)	240
BitmapAreaShiftY	0..65553 5	Shift of the bitmap area center point in vertical direction. **Condition:** (BitmapAreaY/2+BitmapAreaShiftY≤DisplaySizeY) AND (BitmapAreaY/2– BitmapAreaShiftY ≥0)	120
IlluminatedDisplayAreaX	0..65553 5	Number of illuminated pixels in horizontal direction **Condition:** IlluminatedDisplayAreaX ≤DisplaySizeY	480
IlluminatedDisplayAreaY	0..65553 5	Number of illuminated pixels in vertical direction **Condition:** IlluminatedDisplayAreaY ≤DisplaySizeY	220
IlluminatedDisplayAreaShiftX	0..65553 5	Shift of the illuminated display area center point in horizontal direction. **Condition:** (IllDisplayAreaX/2+IllDisplayAreaShiftX≤DisplaySizeX) AND (IllDisplayAreaX/2– IllDisplayAreaShiftX ≥0)	480
IlluminatedDisplayAreaShiftY	0..65553 5	Shift of the illuminated display area center point in vertical direction. **Condition:** (IllDisplayAreaY/2+IllDisplayAreaShiftY≤DisplaySizeY) AND (IllDisplayAreaY/2– IllDisplayAreaShiftY ≥0)	220
WarpingAreaX	0..65553 5	Number of pixels used for warping in horizontal direction. **Condition:** WarpingAreaX≤IlluminatedDisplayAreaX > This parameter is present for historical reasons and is not used during computation/adjustment. Its validity is still checked, though.	470

WarpingAreaY	0..65553 5	Number of pixels used for warping in vertical direction. **Condition:** WarpingAreaY≤IlluminatedDisplayAreaY > This parameter is present for historical reasons and is not used during computation/adjustment. Its validity is still checked, though.	120
WarpedAreaShiftX	0..65553 5	Shift of the warped area center point in horizontal direction. > This parameter is present for historical reasons and is not used during computation/adjustment. Its validity is still checked, though.	220
WarpedAreaShiftY	0..65553 5	Shift of the warped area center point in vertical direction. > This parameter is present for historical reasons and is not used during computation/adjustment. Its validity is still checked, though.	110
RotationValue	-18000 .. +18000	Rotation value in grad * 100	-150 // rotate 1.5°
ParamSet	Data	Set of reference point shift vectors row-major ordered(left to right, top to bottom). Each reference point in x/y direction is represented as a signed integer value in range –32,768 .. 32,767 (2 bytes).	(X0, Y0) (X0, Y1) ... (XM, YN) M: #RefPointsX N: #RefPointsY
ParamSetLength	0..65553 5	Length of parameter set in bytes calculated using the following formula: 2 * NumRefPointsX * NumRefPointsY * NumBitsPerParam / 8	825 // 2*25*11*12/8
RotationLimit	0..180	Maximal rotation angle in grad (+/- value)	5 // +/- 5°
TrapezoidLimitX	0..65553 5	Maximum value for trapezoid transformation in horizontal direction (+/- value)	50
TrapezoidLimitY	0..65553 5	Maximum value for trapezoid transformation in vertical direction (+/- value)	50

ParallelogramLimitX	0..65553 5	Maximum value for parallelogram transformation in horizontal direction (+/- value)	50
ParallelogramLimitY	0..65553 5	Maximum value for parallelogram transformation in vertical direction (+/- value)	50
PinbalanceLimitX	0..65553 5	Maximum value for pin balance transformation in horizontal direction (+/- value)	50
PinbalanceLimitY	0..65553 5	Maximum value for pin balance transformation in vetical direction (+/- value)	50
PincushionLimitX	0..65553 5	Maximum value for pincushion transformation in horizontal direction (+/- value)	50
PincushionLimitY	0..65553 5	Maximum value for pincushion transformation in vetical direction (+/- value)	50
WidthLimit	0..65553 5	Maximum value for width transformation in horizontal direction (+/- value)	50
HeightLimit	0..65553 5	Maximum value for height transformation in vetical direction (+/- value)	50
ScaleMode	0 or 1	0 - scale factor is 1. Scaling might occur if image does not fit inside illuminated area 1 - scale factor is calculated based on RotationLimit value, such that, when rotated at maximum, the image will fit inside illuminated area	0

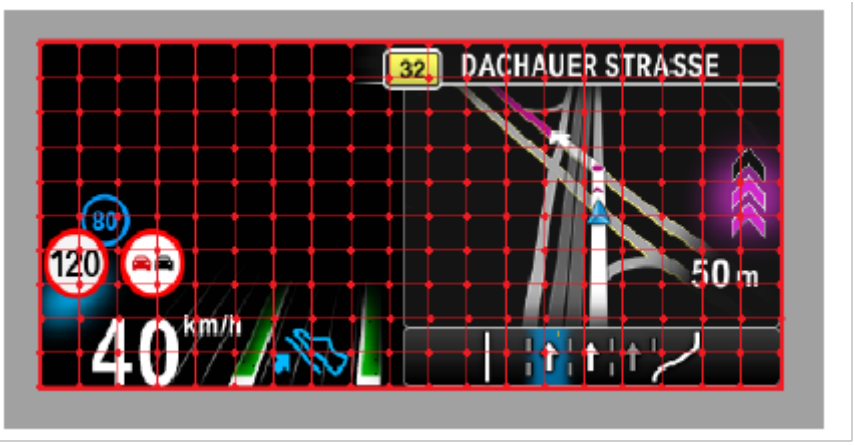
Please refer to the following chapters for details on how the parameters influence the result of the warped image.

Warping Areas

The display area comprises the whole image including the grey border, as depicted in the figure below.

The bitmap image area is the area within the grey border. This is the area of interest and which should be warped. The reference points are evenly distributed over the bitmap image area.

The illuminated display area is the part of the display, which is backlit and projected to the windshield. This is the area where the warped image will be rendered into. If the bitmap image area is larger than the specified illuminated display area, then the warp image will be scaled to fit into the illuminated area boundaries.



The Warping Library needs the following parameters to be configured which are related to the warping areas:

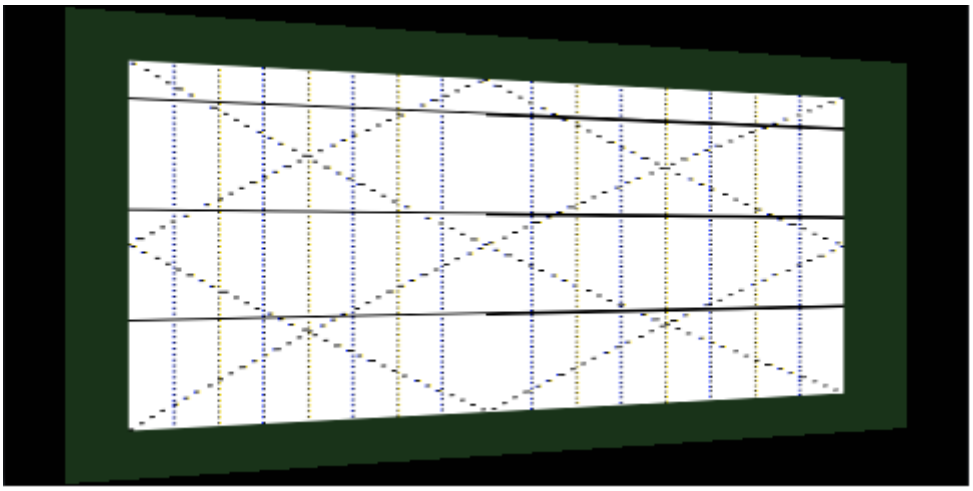
- Display Size X/Y: area of the display size
- Bitmap Area X/Y: area for the warping screen
- Bitmap Shift Area X/Y: shift values of bitmap area center point
- Illuminated Area X/Y: area of the illuminated area
- Illuminated Shift Area X/Y: shift value of illuminated area center point
- Warping Area (parameter is no longer used in the calculation of the warp image, but the values are checked for boundary correctness - use same values as for illumination area).
- Warping Shift Area (parameter is no longer used in the calculation of the warp image, but the values are checked for boundary correctness - use same values as for illumination shift area).

The Bitmap Shift Area describes the connection vector between the origin of the bitmap coordinate system (0,0) and the center point of the bitmap image - the Bitmap Area Center Point. This center point represents the center point of the image area.

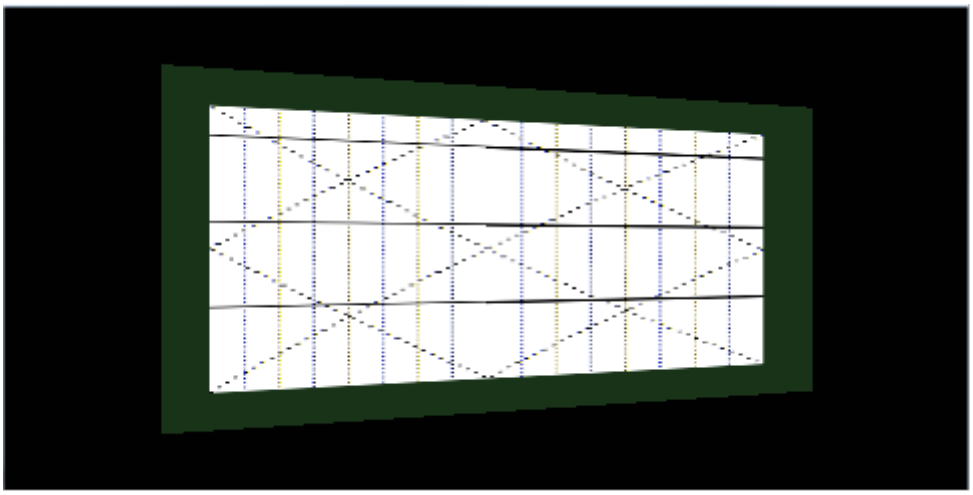
Example

Warp image is influenced by the size of bitmap area and illuminated area.

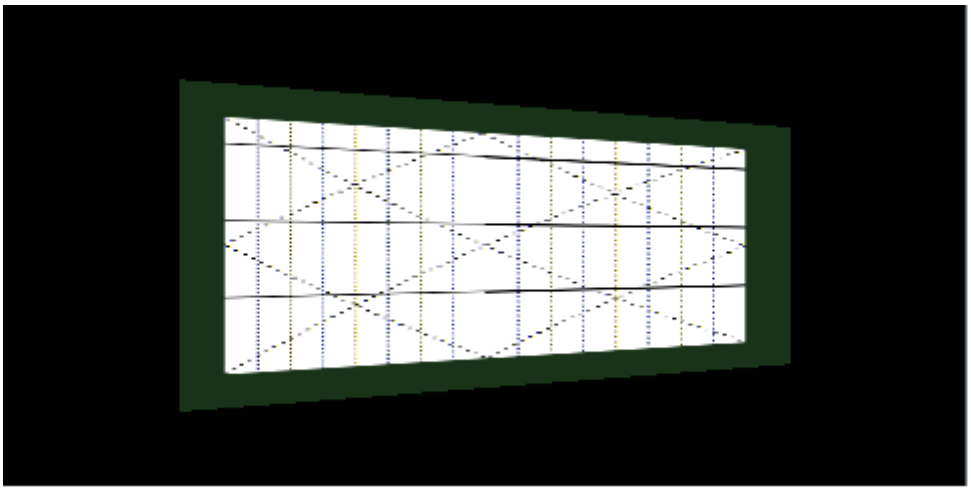
Reference points are distributed over display area
Display Area: 480x240
BitmapArea: 480x240 | BitmapShiftArea: 240x120
IlluminatedArea: 480x240 | IlluminatedShiftArea: 240x120



Content shrinks to fit inside illuminated area
Display Area: 480x240
BitmapArea: 480x240 | BitmapShiftArea: 240x120
IlluminatedArea: 409x190 | IlluminatedShiftArea: 240x120

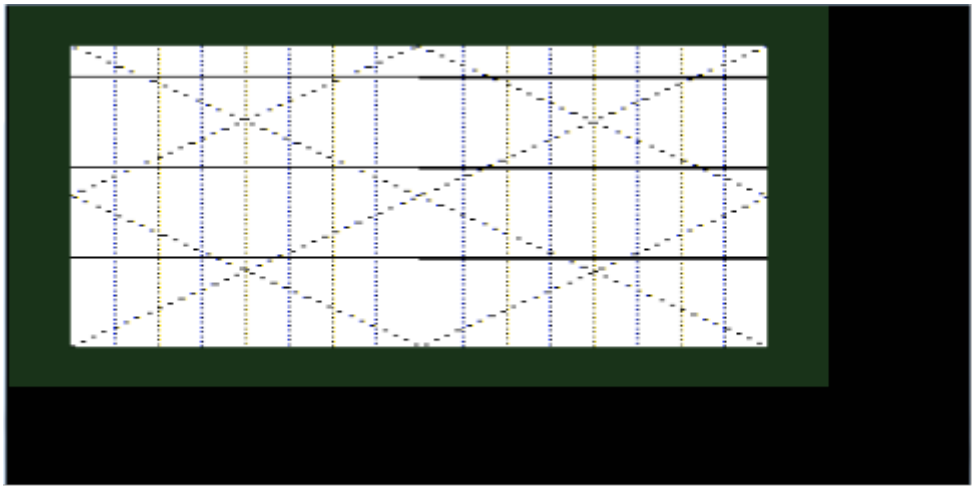


Bitmap area changes, content shrinks even more to fit inside illuminated area
Display Area: 480x240
BitmapArea: 409x190 | BitmapShiftArea: 240x120
IlluminatedArea: 480x240 | IlluminatedShiftArea: 240x120

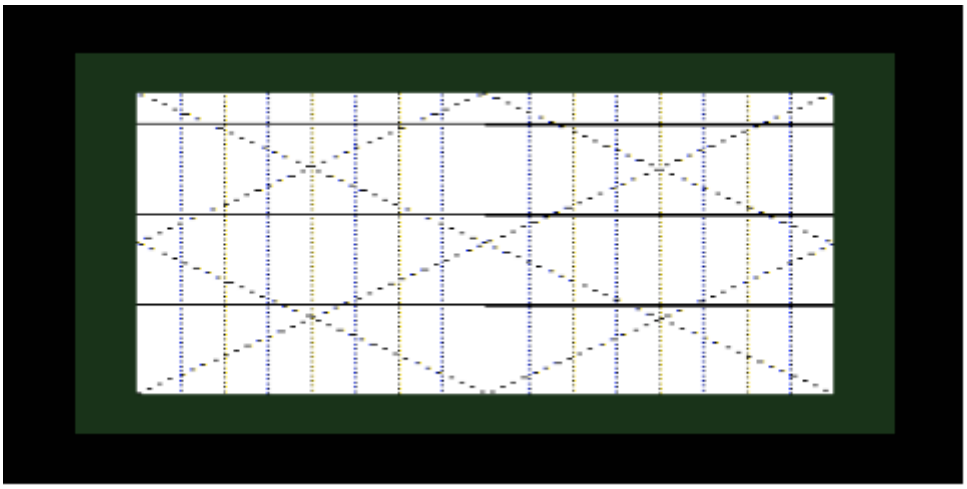


Position of the warp image inside the illuminated area can by influenced by the corresponding BitmapShiftArea parameters. Similarly, the position of the illuminated area is given by the IlluminatedShiftArea. For better visualizing this, no pre-distortion was applied.

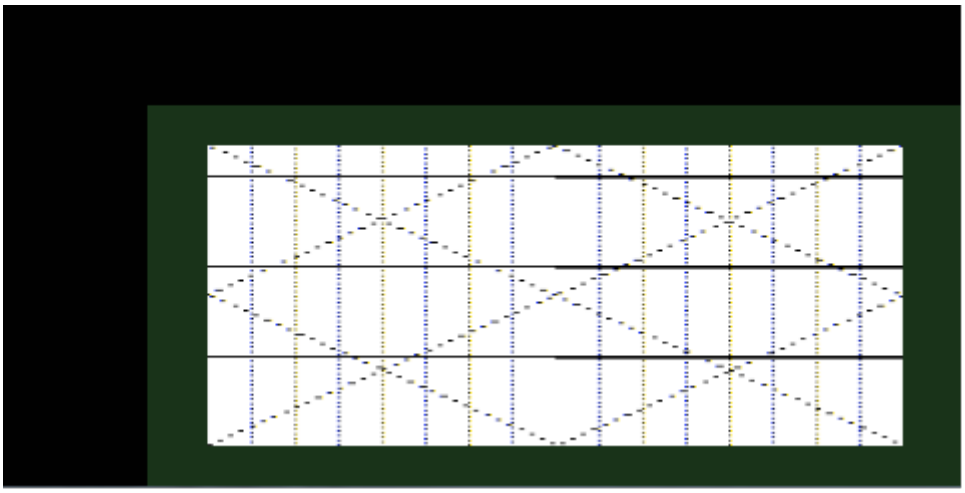
Top left aligned
Display Area: 480x240
BitmapArea: 409x190 | BitmapShiftArea: 205x95
IlluminatedArea: 480x240 | IlluminatedShiftArea: 240x120



Center aligned
Display Area: 480x240
BitmapArea: 409x190 | BitmapShiftArea: 240x120
IlluminatedArea: 480x240 | IlluminatedShiftArea: 240x120



Bottom right aligned
Display Area: 480x240
BitmapArea: 409x190 | BitmapShiftArea: 276x145
IlluminatedArea: 480x240 | IlluminatedShiftArea: 240x120



> For the presented use cases, the warp image bounds were not specified. (see Warp Image Bounds for more details)

Pixel Resolution and Number of Bits per Parameter

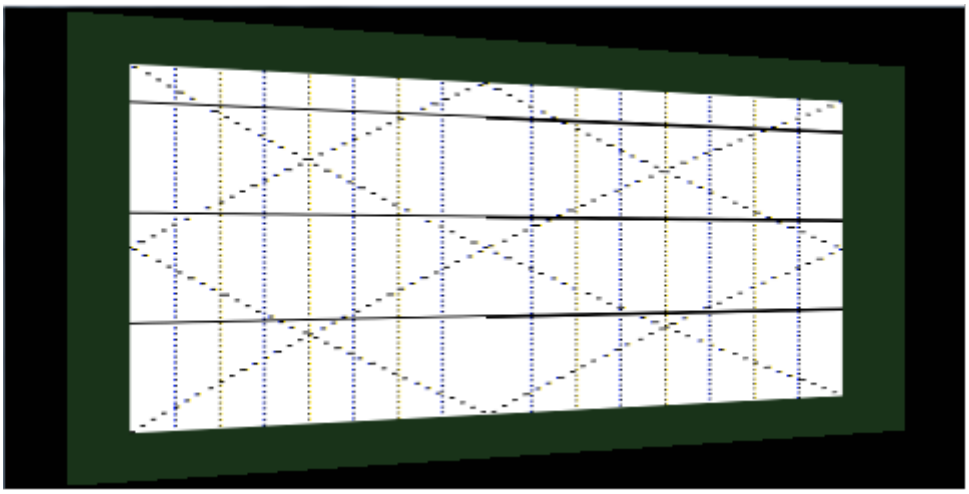
Pixel Resolution

The pixel resolution determines the warping accuracy. The final displacement of a reference point is the product of the warping parameter by parameter resolution. A warping parameter is therefore given by the desired displacement divided by resolution.

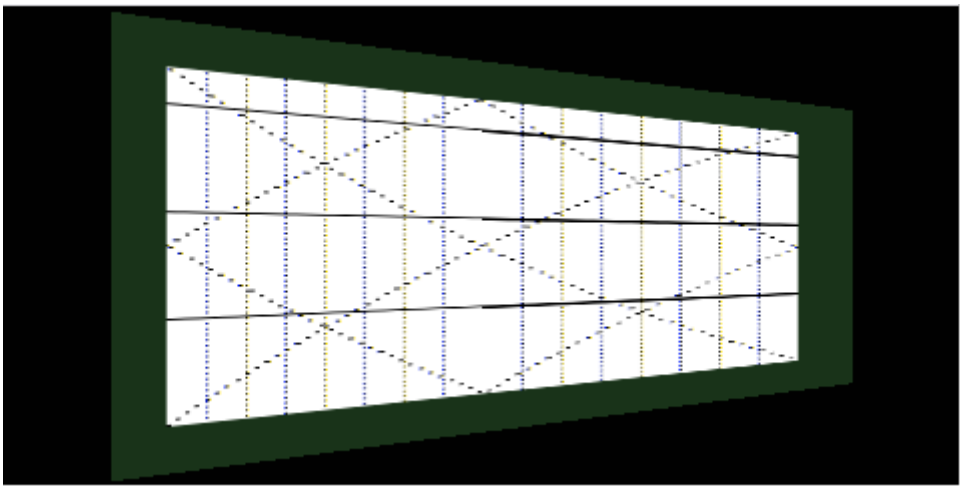
The resolution values supported by the warping library are: **0**: $^1/_{16}$ px, **1**: $^1/_8$ px, **2**: $^1/_4$ px, **3**: $^1/_2$ px, **4**: 1 px, **5**: 2 px, **6**: 4 px, **7**: 8 px.

Example

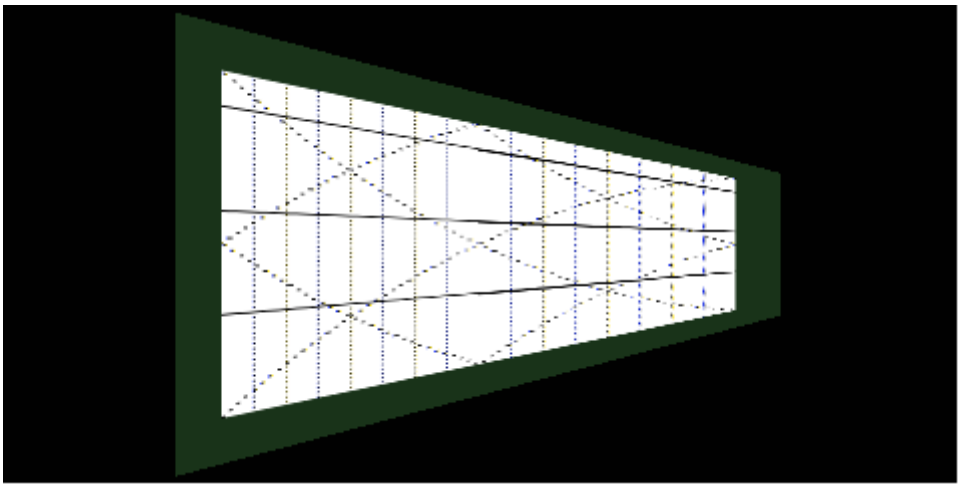
PixResolution: 0
Factor for multiplying: 0.0625f



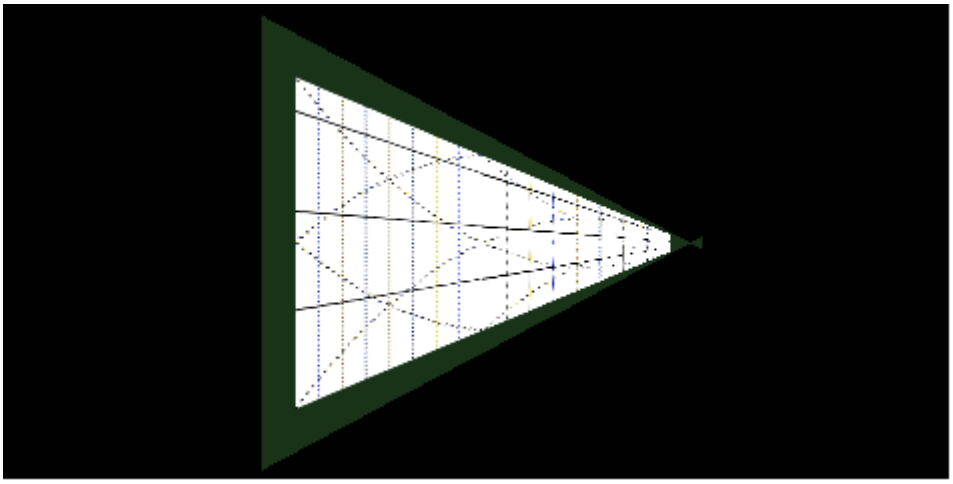
PixResolution: 1
Factor for multiplying: 0.125f



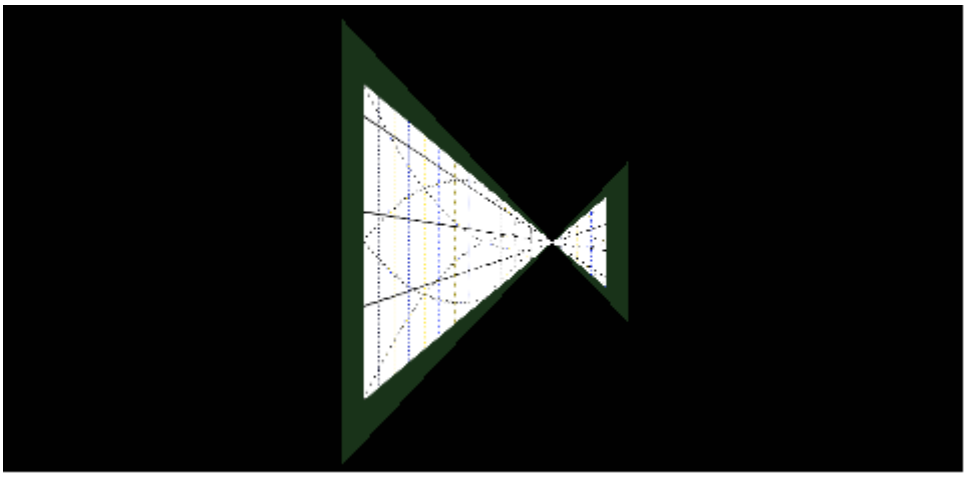
PixResolution: 2
Factor for multiplying: 0.25f



PixResolution: 3
Factor for multiplying: 0.5f



PixResolution: 4
Factor for multiplying: 1.0f



PixResolution > 4
WarpMatrix is no longer computed since delta calculation leads to "out of trapezoid limit" error

Number of Bits per Parameter

The configuration parameter "NumOfBitsPerParam" gives the allowed range for the warping parameter data. Therefore, each parameter is represented as a signed integer value of the given bit width.

If **n** is the number of bits, then the allowed range is: $-(2^{n-1})$ .. + $(2^{n-1}- 1)$.

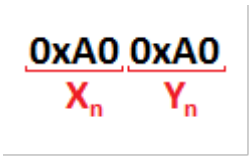
Maximum possible value for "NumOfBitsPerParam" is 16. Considering this, the maximum allowed range for the warping parameter data is: −32 768 .. 32 767.

The length of the parameter data is given by the following formula: 2 * NumOfRefPointsX * NumOfRefPointsY * NumOfBitsPerParam / 8.

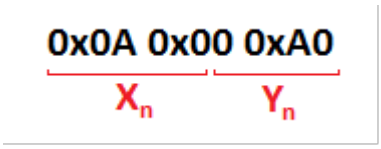
Example

- if "NumOfBitsPerParam" is 8, the warping parameter corresponding to point X_nY_n is represented as: 0xA0 0xA0
- if "NumOfBitsPerParam" is 12, the warping parameter corresponding to point X_nY_n is represented as: 0x0A 0x00 0xA0

NumOfBitsPerParam: 8



NumOfBitsPerParam: 12

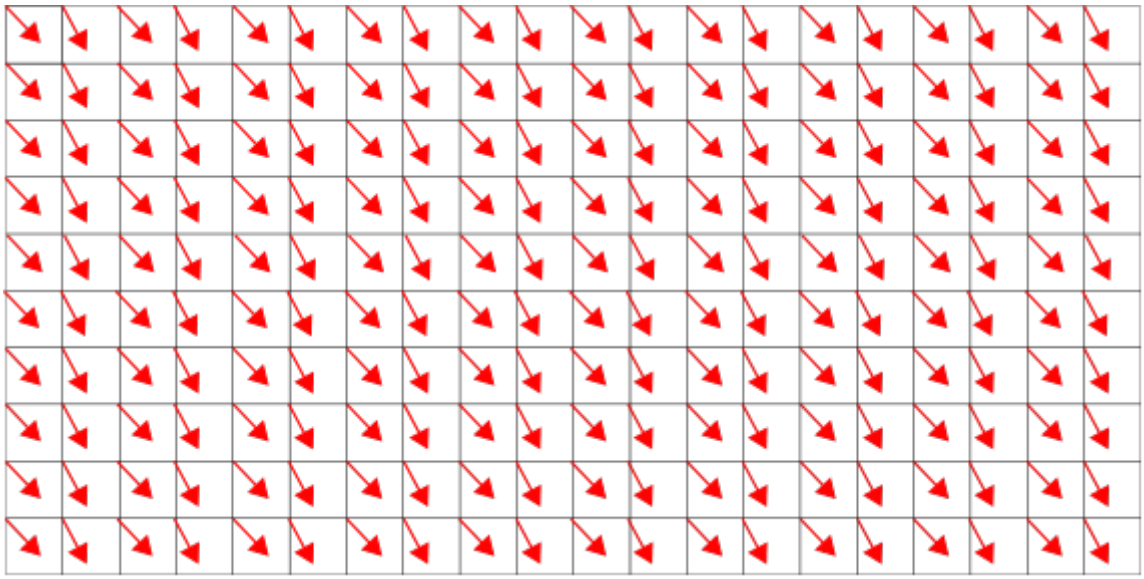

0x0A 0x00 0xA0
Xn
Yn

Reference Points and the Warp Matrix

The warping parameters are based on reference points. These points are mapped by a 2D array containing coordinates inside the bitmap area to be warped.

For each reference point, a shift vector is stored by the application. Every shift vector determines the amount of distortion to be applied to the corresponding point in x- and y-direction. The warping parameter correspond to the shift vectors (in pixels) of warping interpolation point in warped images, in comparison to reference points in non-warped images (divided by the specified pixel resolution). To save memory space and computing time, the absolute coordinates are not saved as warping parameters, but rather only the distance from the reference points (without rotation and shifting).

The figure below shows a reference point grid with corresponding shift vectors.



CGI Studio Warping Library used row-major data ordering for area shift vector sets.

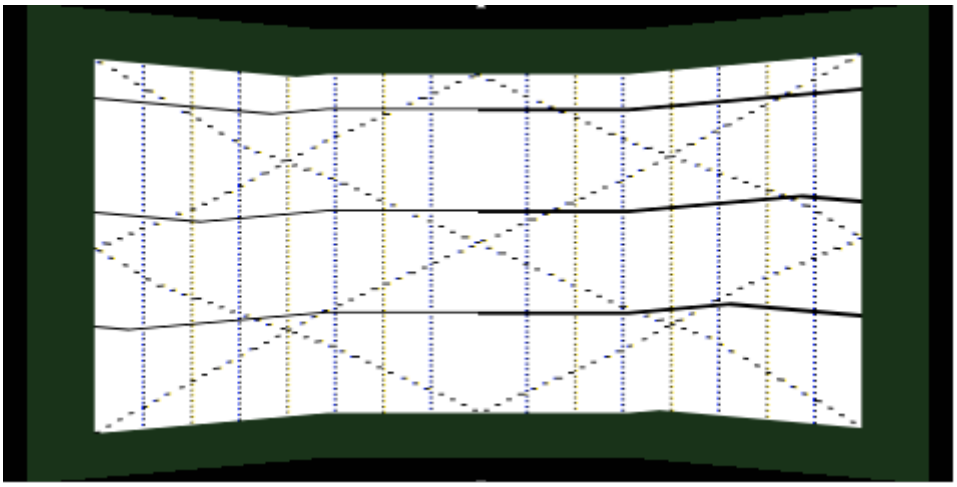
The warp matrix is a two-dimensional matrix computed by the warping library. Each point corresponds to a bitmap image reference point that has been shifted. The value of each point is the vector-addition of the reference point and the corresponding shift vector with regards to pixel resolution. All values of the warp matrix are normalized to the range [0..1] at the end of warp matrix computation or adjustment in order to be used by Candera.

> The Warping Library does not provide an interface that supports the usage of "final shifted" points instead of the shift vectors. In such case, the library cannot be used. These "final shifted" points can serve as entry point in calculating the Candera::WarpMatrix data.

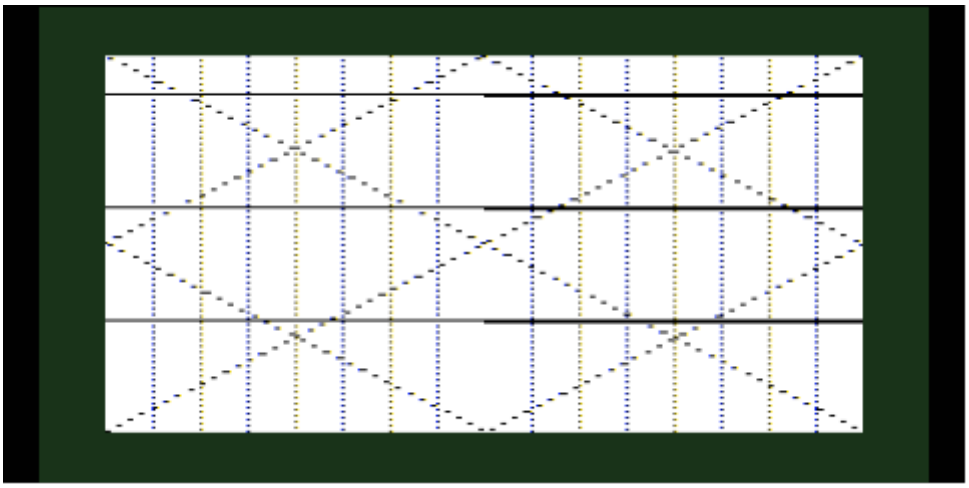
Example

The number of reference points influence the output of the warp matrix. The points should be chosen, proportionally, with regards to the size of the bitmap area. The larger the value, the more accurate and smooth the result of the warp image.

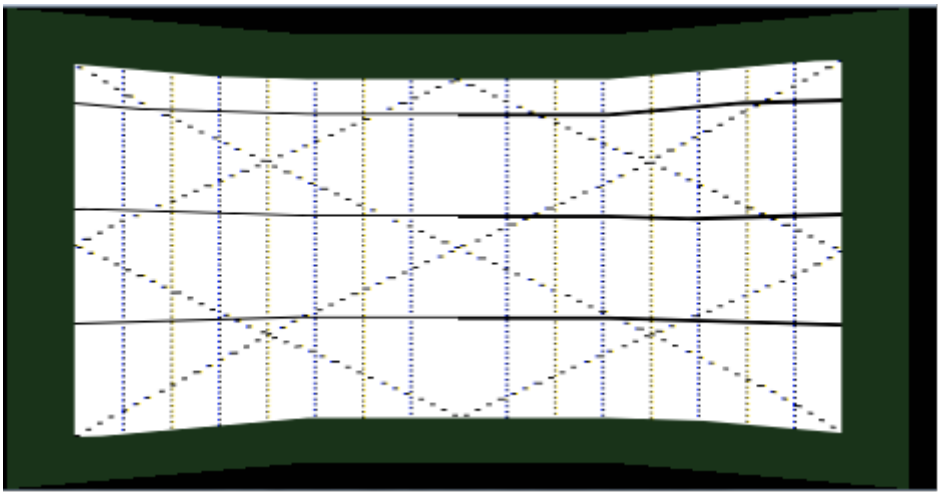
NumOfRefPointsX: 4 | NumOfRefPointsY: 2



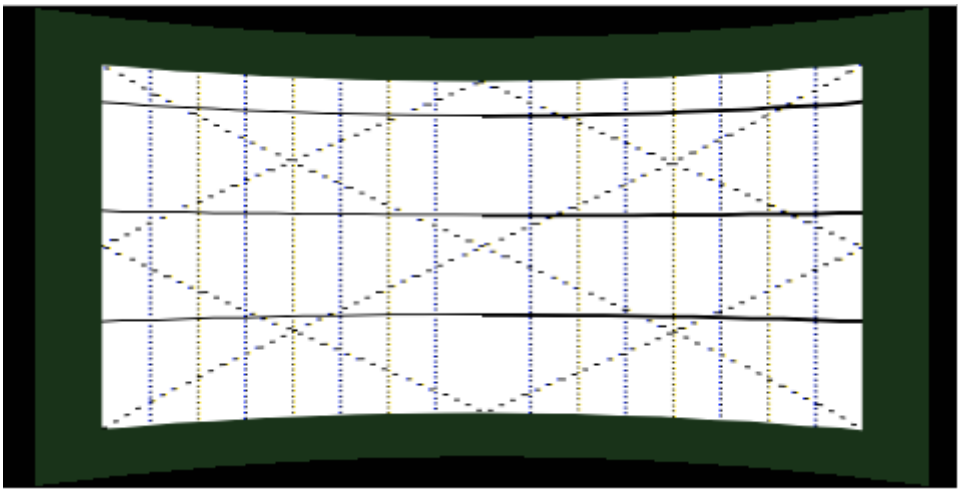
NumOfRefPointsX: 2 | NumOfRefPointsY: 4



NumOfRefPointsX: 4 | NumOfRefPointsY: 4



NumOfRefPointsX: 11 | NumOfRefPointsY: 5



Adjustments

An adjustment is specified by a **mode**, a **direction** and a **delta**. For each reference point and corresponding shift vector, the adjustment is computed and applied. The resulting shift vector is again cached in the warping library. This way of implementation allows incremental adjustments.

Mode defines the way of the distortion. Following modes are available:

- Parallelogram
- Trapezoid
- Pin balance
- Pincushion
- Shift
- Size
- Rotation (around bitmap image center point)

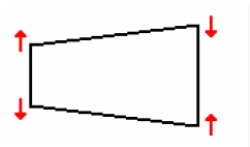
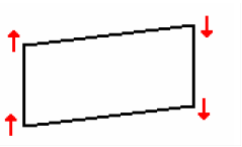
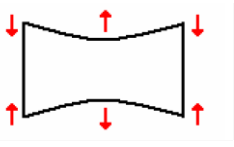
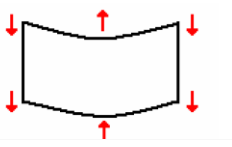
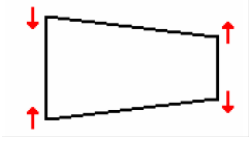
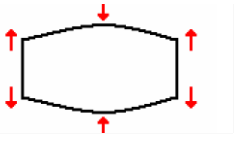
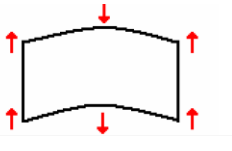
Direction specifies the direction in which the distortion is applied. Following directions are available:

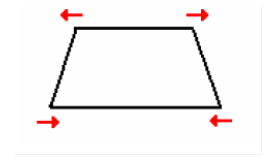
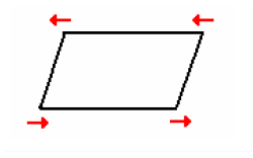
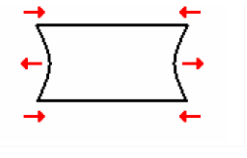
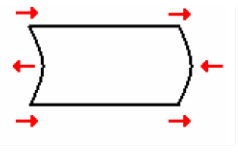
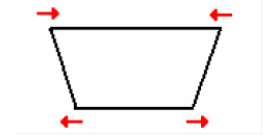
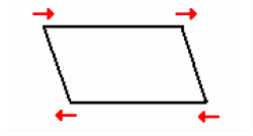
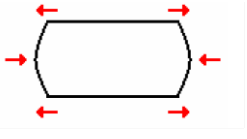
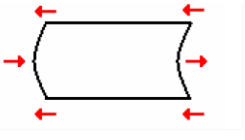
- Up
- Down
- Left
- Right

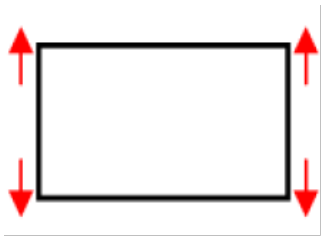
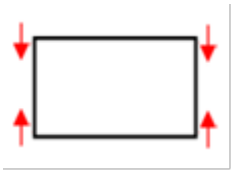
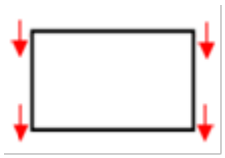
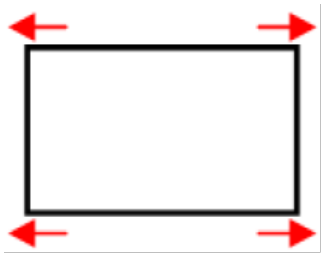
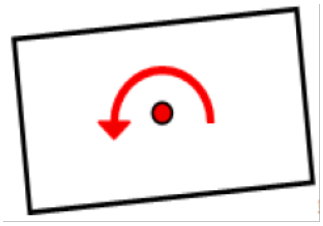
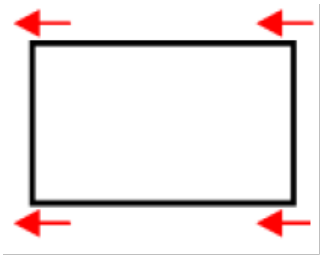
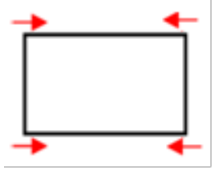
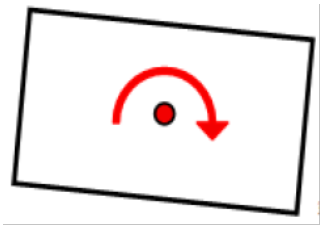
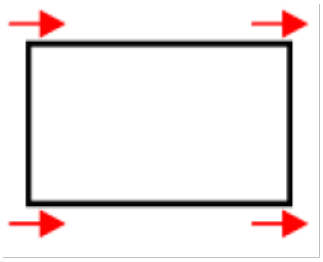
Delta specifies the amount of distortion that is applied. Delta is an unsigned integer value in range 0..255 (1 byte). The delta is added to the existing offset of the corresponding reference points.

Predefined Adjustments

Based on mode and direction, the following transformations are possible:

Mode Direction	Trapezoid	Parallelogram	Pincushion	Pin balance
Up				
Down				

Left				
Right				

Mode Direction	Size	Rotation	Shift
Up			
Down			
Left			
Right			

Formulas for adjustments

The following formulas describe the mathematically necessary adjustments of the reference points. If the requested adjustment exceeds the limit, no calculation shall be done and the previous status will be kept.

Trapezoid mode

***mode* = trapezoid transformation; *direction* = up**
condition: $TrapezoidY - delta \geq -TrapezoidLimitY$

$$y[i][j] = y[i][j] - delta \cdot \left(1 - \frac{2\,i}{NumOfRefPointsX - 1}\right) \cdot \left(1 - \frac{2\,j}{NumOfRefPointsY - 1}\right)$$

***mode* = trapezoid transformation; *direction* = down**
condition: $TrapezoidY + delta \leq TrapezoidLimitY$

$$y[i][j] = y[i][j] + delta \cdot \left(1 - \frac{2\,i}{NumOfRefPointsX - 1}\right) \cdot \left(1 - \frac{2\,j}{NumOfRefPointsY - 1}\right)$$

***mode* = trapezoid transformation; *direction* = left**
condition: $TrapezoidX - delta \geq -TrapezoidLimitX$

$$x[i][j] = x[i][j] - delta \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1}\right) \cdot \left(1 - \frac{2\,j}{NumOfRefPointsY - 1}\right)$$

***mode* = trapezoid transformation; *direction* = right**
condition: $Trapezoidx + delta \leq TrapezoidLimitX$

$$x[i][j] = x[i][j] + delta \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1}\right) \cdot \left(1 - \frac{2\,j}{NumOfRefPointsY - 1}\right)$$

Parallelogram mode

***mode* = parallelogram transformation; *direction* = up**
condition: $ParallelogramY - delta \geq -ParallelogramLimitY$

$$y[i][j] = y[i][j] - delta \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1}\right)$$

***mode* = parallelogram transformation; *direction* = down**
condition: $ParallelogramY + delta \leq ParallelogramLimitY$

$$y[i][j] = y[i][j] + delta \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1}\right)$$

***mode* = parallelogram transformation; *direction* = left**
condition: $ParallelogramX - delta \geq -ParallelogramLimitX$

$$x[i][j] = x[i][j] - delta \cdot \left(1 - \frac{2\,j}{NumOfRefPointsY - 1}\right)$$

***mode* = parallelogram transformation; *direction* = right**
condition: $ParallelogramX + delta \leq ParallelogramLimitX$

$$x[i][j] = x[i][j] + delta \cdot \left(1 - \frac{2\,j}{NumOfRefPointsY - 1}\right)$$

Pinbalance mode

***mode* = pin balance transformation; *direction* = up**
condition: $PinbalanceY - delta \geq -PinbalanceLimitY$

$$y[i][j] = y[i][j] - delta \cdot \left(\frac{-4i^2}{(NumOfRefPointsX - 1)^2} + \frac{4i}{NumOfRefPointsX - 1} - 0.5 \right)$$

***mode* = pin balance transformation; *direction* = down**
condition: $PinbalanceY + delta \leq PinbalanceLimitY$

$$y[i][j] = y[i][j] + delta \cdot \left(\frac{-4i^2}{(NumOfRefPointsX - 1)^2} + \frac{4i}{NumOfRefPointsX - 1} - 0.5 \right)$$

***mode* = pin balance transformation; *direction* = left**
condition: $PinbalanceX - delta \geq -PinbalanceLimitX$

$$x[i][j] = x[i][j] - delta \cdot \left(\frac{-4j^2}{(NumOfRefPointsY - 1)^2} + \frac{4j}{NumOfRefPointsY - 1} - 0.5 \right)$$

***mode* = pin balance transformation; *direction* = right**
condition: $PinbalanceX + delta \leq PinbalanceLimitX$

$$x[i][j] = x[i][j] + delta \cdot \left(\frac{-4j^2}{(NumOfRefPointsY - 1)^2} + \frac{4j}{NumOfRefPointsY - 1} - 0.5 \right)$$

Pincushion mode

***mode* = pincushion transformation; *direction* = up**
condition: $PincushionY - delta \geq -PincushionLimitY$

$$y[i][j] = y[i][j] - delta \cdot \left(\frac{-4i^2}{(NumOfRefPointsX - 1)^2} + \frac{4i}{NumOfRefPointsX - 1} - 0.5 \right) \cdot \left(1 - \frac{2j}{NumOfRefPointsY - 1} \right)$$

***mode* = pincushion transformation; *direction* = down**
condition: $PincushionY + delta \leq PincushionLimitY$

$$y[i][j] = y[i][j] + delta \cdot \left(\frac{-4i^2}{(NumOfRefPointsX - 1)^2} + \frac{4i}{NumOfRefPointsX - 1} - 0.5 \right) \cdot \left(1 - \frac{2j}{NumOfRefPointsY - 1} \right)$$

***mode* = pincushion transformation; *direction* = left**
condition: $PincushionX - delta \geq -PincushionLimitX$

$$x[i][j] = x[i][j] - delta \cdot \left(\frac{-4j^2}{(NumOfRefPointsY - 1)^2} + \frac{4j}{NumOfRefPointsY - 1} - 0.5 \right) \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1} \right)$$

***mode* = pincushion transformation; *direction* = right**
condition: $PincushionX + delta \leq PincushionLimitX$

$$x[i][j] = x[i][j] + delta \cdot \left(\frac{-4j^2}{(NumOfRefPointsY - 1)^2} + \frac{4j}{NumOfRefPointsY - 1} - 0.5 \right) \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1} \right)$$

Size mode

mode = **size**, ***direction*** = **up**
condition: *Height* + *delta* <= *HeightLimit*

$$y[i][j] = y[i][j] + delta \cdot \left(1 - \frac{2j}{NumOfRefPointsY - 1}\right)$$

mode = **size**, ***direction*** = **down**
condition: *Height* - *delta* >= - *HeightLimit*

$$y[i][j] = y[i][j] - delta \cdot \left(1 - \frac{2j}{NumOfRefPointsY - 1}\right)$$

mode = **size**, ***direction*** = **left**
condition: *Width* + *delta* <= *WidthLimit*

$$x[i][j] = x[i][j] + delta \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1}\right)$$

mode = **size**, ***direction*** = **right**
condition: *Width* - *delta* <= - *WidthLimit*

$$x[i][j] = x[i][j] - delta \cdot \left(1 - \frac{2i}{NumOfRefPointsX - 1}\right)$$

Rotation mode

mode = **rotation**, ***direction*** = **left**
condition: all warped reference points are inside the Illuminated TFT Area

$$x[i][j] = \cos(-delta) \cdot (x[i][j] - x\_center) + \sin(-delta) \cdot (y[i][j] - y\_center) + x\_center$$

$$y[i][j] = -\sin(-delta) \cdot (x[i][j] - x\_center) + \cos(-delta) \cdot (y[i][j] - y\_center) + y\_center$$

mode = **rotation**, ***direction*** = **right**
condition: all warped reference points are inside the Illuminated TFT Area

$$x[i][j] = \cos(delta) \cdot (x[i][j] - x\_center) + \sin(delta) \cdot (y[i][j] - y\_center) + x\_center$$

$$y[i][j] = -\sin(delta) \cdot (x[i][j] - x\_center) + \cos(delta) \cdot (y[i][j] - y\_center) + y\_center$$

Shift mode

***mode* = shift, *direction* = up**
condition: all warped reference points are inside the Illuminated TFT Area
$y[i][j] = y[i][j] - delta$

***mode* = shift, *direction* = down**
condition: all warped reference points are inside the Illuminated TFT Area
$y[i][j] = y[i][j] - delta$

***mode* = shift, *direction* = left**
condition: all warped reference points are inside the Illuminated TFT Area
$x[i][j] = x[i][j] + delta$

***mode* = shift, *direction* = right**
condition: all warped reference points are inside the Illuminated TFT Area
$x[i][j] = x[i][j] + delta$

> NumOfRefPointsX - number of reference points in horizontal direction
> NumOfRefPointsY - number of reference points in vertical direction
> x[i][j] - x-coordinate of reference point, a 2-dimensional array of floating point variables
> y[i][j] - y-coordinate of reference point, a 2-dimensional array of floating point variables

Rotation and Scale

Rotation

The rotation takes place around the warped center point of the bitmap area.

If an uneven number of reference points in x/y direction is specified, the warped center point coincides with the middle reference grid point and the coordinates of the rotation axis are present. If an even number of reference grid points is use, the coordinates of the warped center point are calculated.

An initial RotationValue and the RotationLimit are specified through the configuration parameter structure:

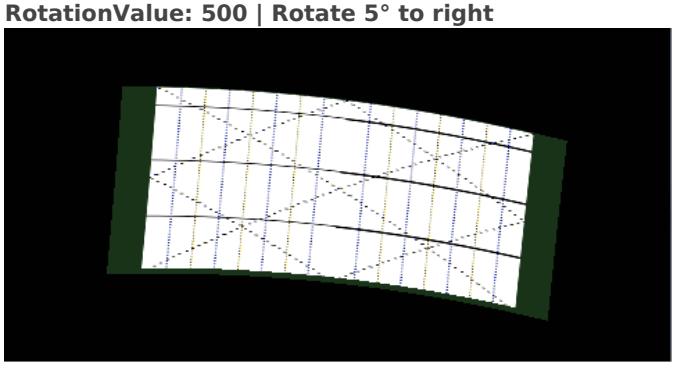
- Initial Rotation value is represented in **grad*100**. This is the default rotation value that is applied to the warp image prior to rendering.
- RotationLimit represents the maximum rotation angle in grad that the warped image will be rotated to. Both positive and negative values are accepted.

Warp image rotation at run-time is achieved through the adjustments supported by the Warping Library. Subsequent calls of warp matrix adjustments result in incremental rotation of the warp image with an incremental step equal to the specified delta value.

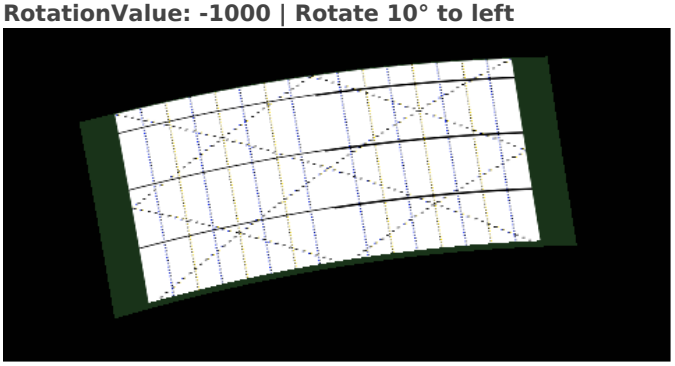
> If the initial RotationValue exceeds the specified RotationLimit a WarpResultErrorConfigInvalidValue will be thrown and the calculation of the warp matrix will be aborted.
> Scaling of the warp image occurs when the RotationLimit value is chosen such that at maximum rotation point, the image will extend beyond the illuminated area. More details related to image scaling is presented in the next section.

Example

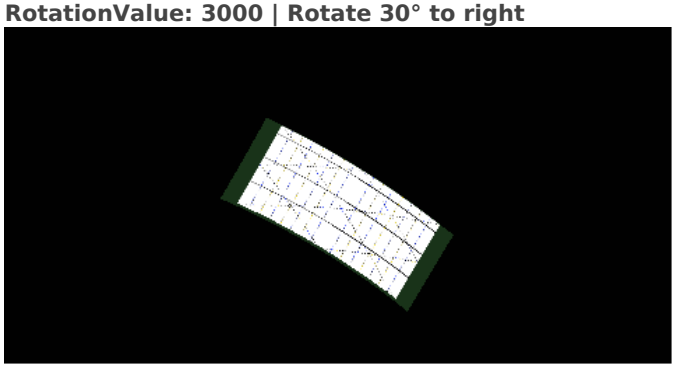
RotationValue: 500 | Rotate 5° to right



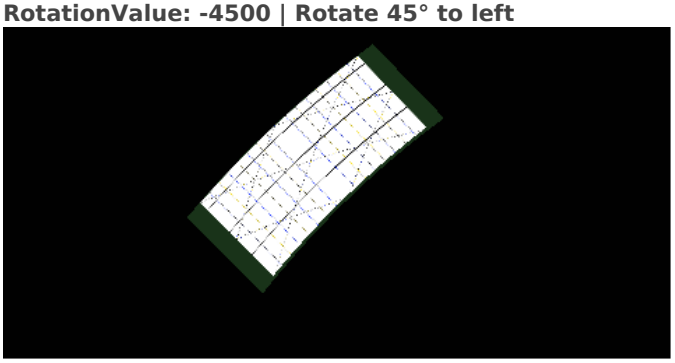
RotationValue: -1000 | Rotate 10° to left



RotationValue: 3000 | Rotate 30° to right



RotationValue: -4500 | Rotate 45° to left



Scaling

The warping library performs automatic rescaling of the resulting warping matrix whenever it overlaps or exceeds the boundaries of the illuminated area. It is possible to identify two cases in which rescaling will be done:

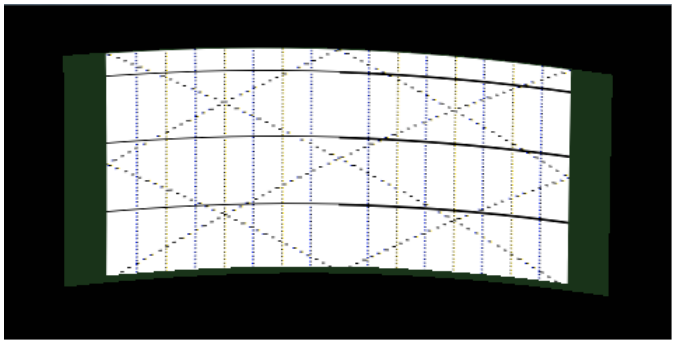
- If the configured illuminated area is smaller than the bitmap area, the resulting warp matrix will be rescaled to the specified illuminated area.
- If the RotationLimit value is chosen such that at maxim rotation point, the warp image will extend beyond illuminated area boundaries.

The ScaleMode parameter influences the behavior of scaling by modifying the internal scale factor:

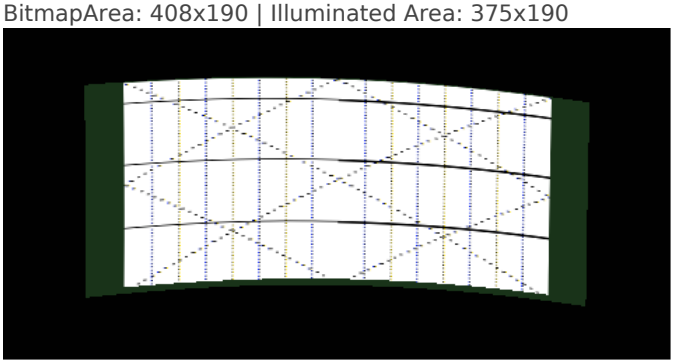
- 0 - scale factor is 1. Scaling occurs only if the image does not fit inside illuminated area. Rotating the warp image at run-time will rescale the resulted rotated image if it falls outside illuminated area boundaries.
- 1 - scale factor is calculated based on RotationLimit value. The warp image will be scaled down to the size that fits inside the illuminated area when a rotation equal to RotationLimit is applied.

Example

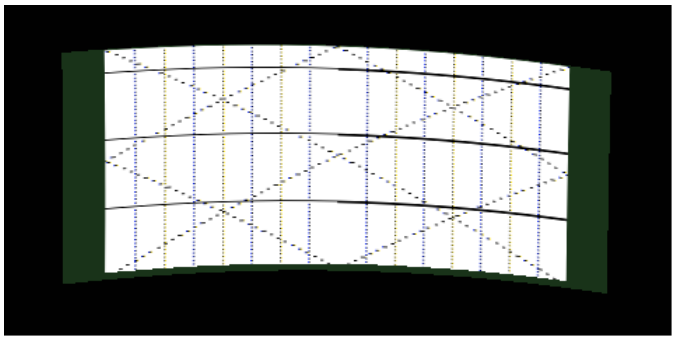
No Scaling occurs
BitmapArea: 408x190 | Illuminated Area: 480x190



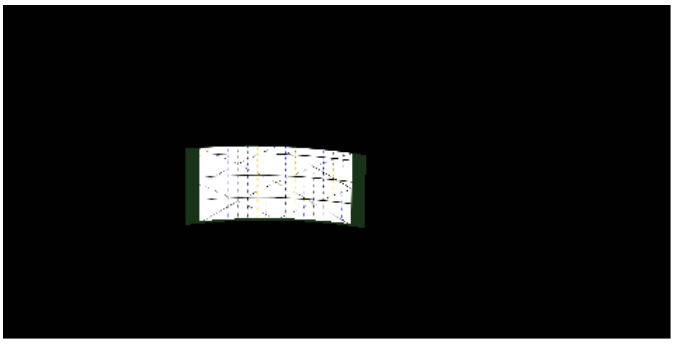
Scaling occurs because bitmap area exceeds illuminated area
BitmapArea: 408x190 | Illuminated Area: 375x190



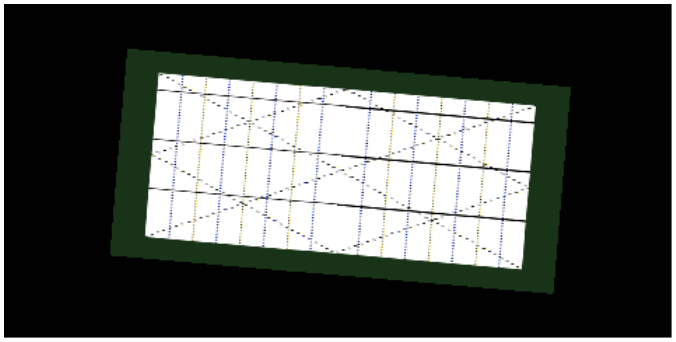
No Scaling occurs since warp image does not exceed illuminated area
BitmapArea: 408x190 | Illuminated Area: 480x190
RotationValue: 0° | RotationLimit: 45°
ScaleMode: 0



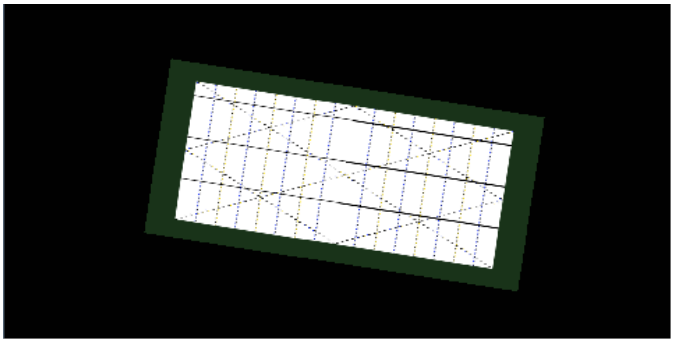
Scaling occurs because of specified RotationLimit and ScaleMode
BitmapArea: 408x190 | Illuminated Area: 480x190
RotationValue: 0° | RotationLimit: 45°
ScaleMode: 1



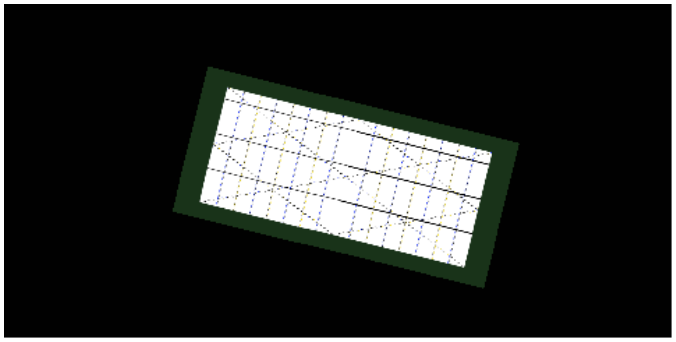
Scaling down during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 0



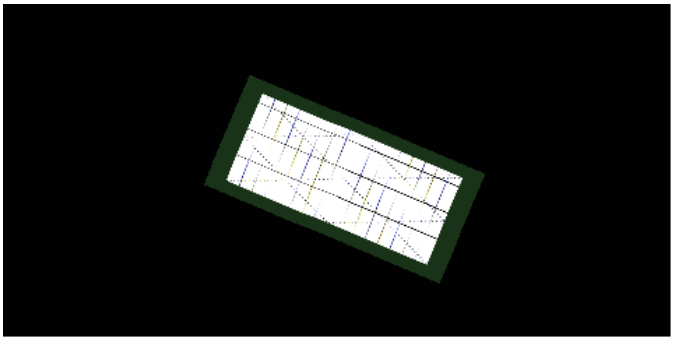
Scaling down during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 0



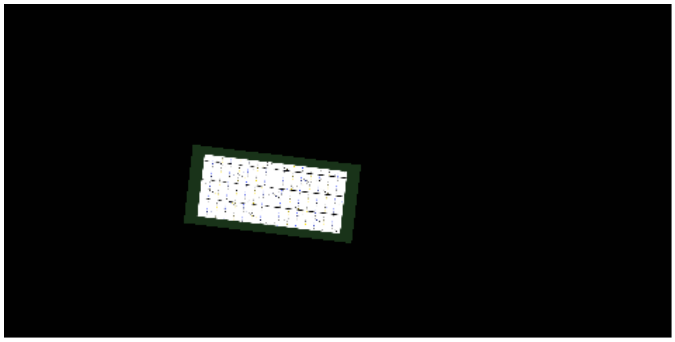
Scaling down during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 0



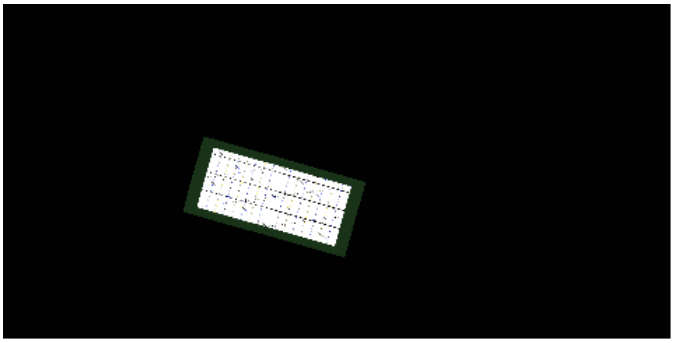
Scaling down during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 0



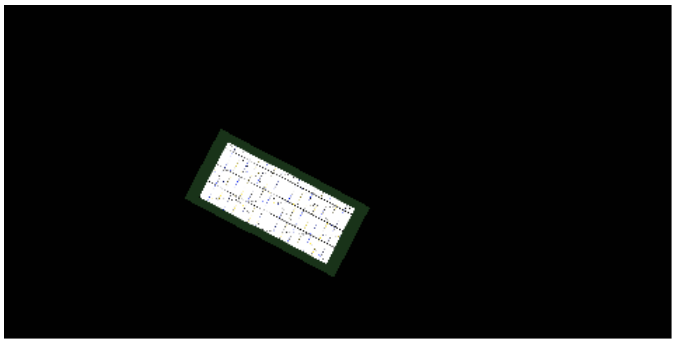
No scaling during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 1



No scaling during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 1



No scaling during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 1



No scaling during incremental rotation
BitmapArea: 408x190 | Illuminated Area: 480x190
ScaleMode: 1

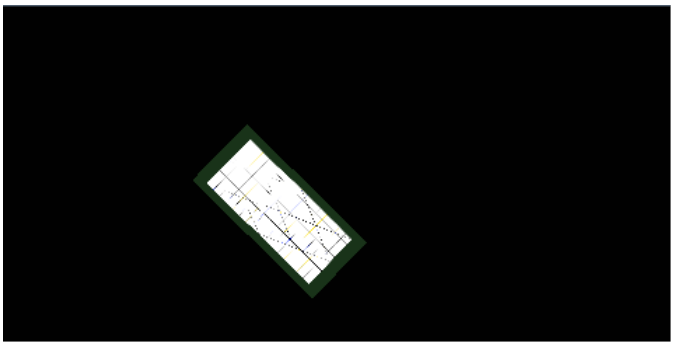


Image Flipping

Flipping transforms the content prior to warping the image on screen. Flipping is supported only when warping is enabled. The following orientations are supported:

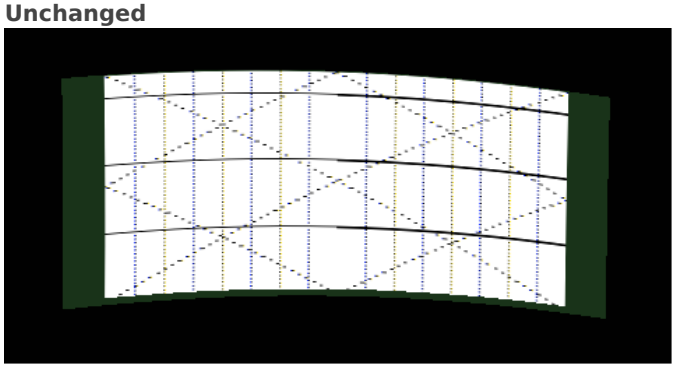
- Candera::DisplayOrientation::Unchanged
- Candera::DisplayOrientation::HorizontallyFlipped
- Candera::DisplayOrientation::VerticallyFlipped
- Candera::DisplayOrientation::VerticallyAndHorizontallyFlipped

Changing the image orientation is achieved through the public interface of Candera::Display class using Candera::Display::SetWarpOrientation, like in the following snippet.

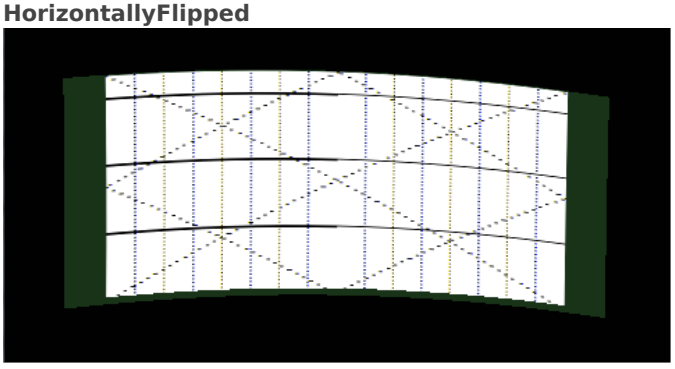
```
    // flip the image vertically and horizontally
    display->SetWarpOrientation(Candera::DisplayOrientation::VerticallyAndHorizontallyFlipped);
```

Example

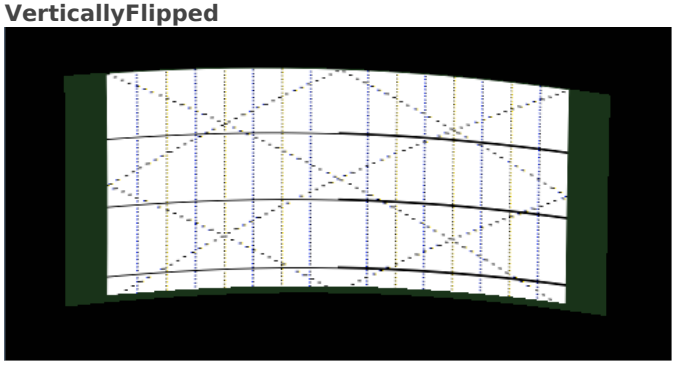
Unchanged



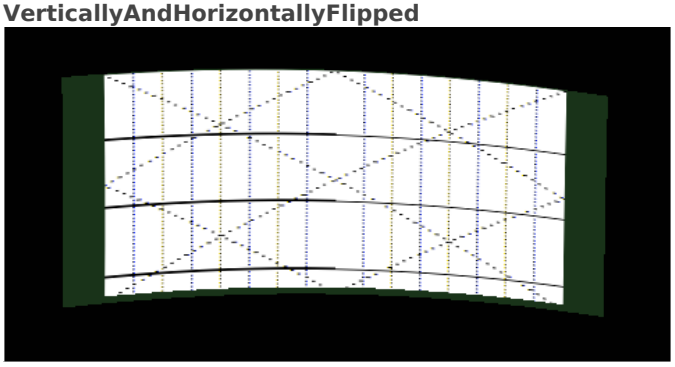
HorizontallyFlipped



VerticallyFlipped



VerticallyAndHorizontallyFlipped



Warp Image Bounds

Warp image bounds comprise upper left corner point of the bitmap image area, its width, and height (normalized to the range [0..1]).

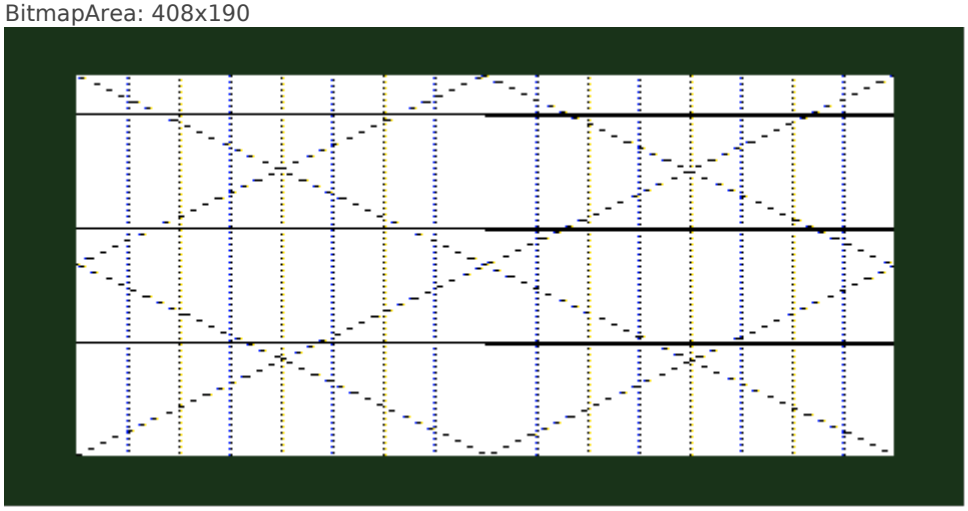
They are used to specify to Candera which part of the rendered display image shall be mapped to the warping mesh computed from the warp matrix. If not specified, then the complete display image will be mapped.

The following code snippet shows how to retrieve the warp image bounds from the Warping library. These can be further passed to Candera through the public interface of Candera::Display class using Candera::Display::SetWarpImageBounds.

```
    WRP_SFLOAT textureOriginX;
    WRP_SFLOAT textureOriginY;
    WRP_SFLOAT textureWidth;
    WRP_SFLOAT textureHeight;

    // retrieve warpTextureBounds
    ComputeWarpImageBoundsFromConfig(&stcWarpConfigParams, &textureOriginX, &textureOriginY, &te
    Candera::Rectangle
warpTextureBounds(textureOriginX, textureOriginY, textureWidth, textureHeight);

    // set the bounds of the warping texture
    display->SetWarpImageBounds(warpTextureBounds);
```

Example

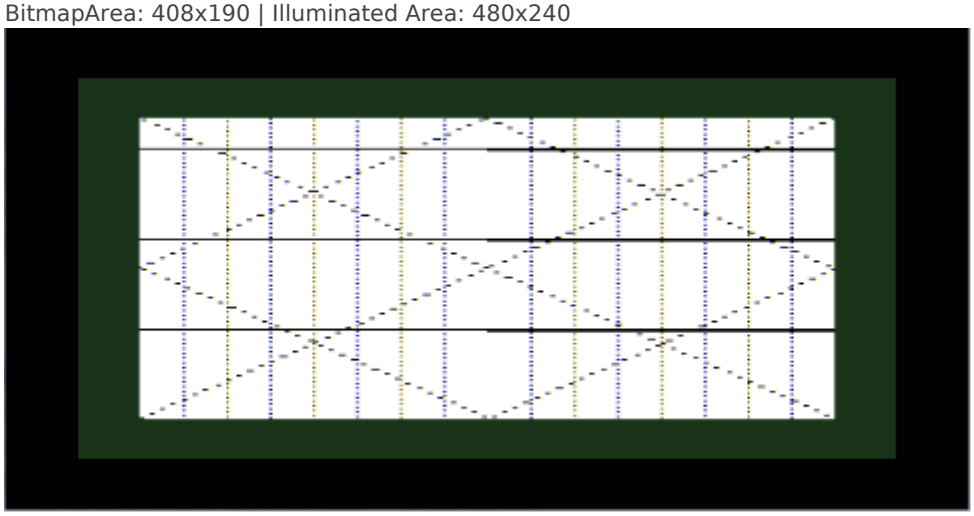
Warping is disabled
Display Area: 480x240
BitmapArea: 408x190



Warping is enabled, image bounds are not specified

Display Area: 480x240

BitmapArea: 408x190 | Illuminated Area: 480x240



Appendix A: Sample code

Please refer to the following code snippets for information on how to the Warping Library and Candera are used in order to calculate the warped output that will be rendered on display.

Create Warp Matrix

```
    en_warp_result_t enResult = WarpResultSuccess;          // Enumeration for possible return v
    stc_warp_configuration_parameter_t stcWarpConfigParams; // Warping configuration parameters
    stc_warp_parameter_set_t stcWarpParamSet;               // Structure of the encoded input pa
    stc_warp_matrix_t stcWarpMatrix;                        // Structure for the calculated matr

    ... // Specify the configuration parameters and the input parameter set according the projec

    // generate the warp matrix based on provided input configuration data
    enResult = CalculateWarpMatrix(&stcWarpConfigParams, &stcWarpParamSet, &stcWarpMatrix);

    // check return error of previous call
    if (enResult == WarpResultSuccess) {
        FEATSTD_LOG_DEBUG("Warp matrix succsessfully created.\n");
    } else {
        FEATSTD_LOG_DEBUG("Failed to create warp matrix.\n");
    }
```

Warp Matrix Adjustments

```
    en_warp_result_t enResult = WarpResultSuccess;          // Enumeration for possible return v
    stc_warp_configuration_parameter_t stcWarpConfigParams; // Warping configuration parameters
    stc_warp_adjust_parameter_t stcWarpAdjustParams;        // Structure with all parameters for
    stc_warp_matrix_t stcWarpMatrix;                        // Structure for the calculated matr
    en_warp_bool_t bSuccess;

    ... // Specify the configuration parameters

    // Specify parameters for delta calculation (example)
    stcWarpAdjustParams.WarpAdjustMode = WarpAdjustModeTrapezoid;
    stcWarpAdjustParams.WarpAdjustDirection = WarpAdjustDirectionDown;
    stcWarpAdjustParams.WarpDelta = 100;

    enResult = AdjustWarpMatrix(&stcWarpConfigParams, &stcWarpAdjustParams, &bSuccess, &stcWarpM

    if (enResult == WarpResultSuccess) {
        FEATSTD_LOG_DEBUG("Adjusted warp matrix.\n");
    } else {
        FEATSTD_LOG_DEBUG("Failed to adjust warp matrix.\n");
```

```
    }
```

Create the Candera WarpMatrix from the previously calculated warp matrix

```
      // Create warp matrix. This is a proof of the disposer concept.
      // The matrix data doesn't need to be copied, since it is static and can
      // be simply set into the warp matrix without a disposer.
      Candera::Float* matrixData = CANDERA_NEW_ARRAY(Candera::Float, 2 * stcWarpMatrix.usNumRefF
      if (matrixData == 0) {
        FEATSTD_LOG_ERROR("failed to allocate matrix data");
        return;
      }
      Candera::MemoryPlatform::Copy(matrixData, FeatStd::Internal::PointerToPointer<void*>(stcWa
      typedef
Candera::MemoryManagement::AdaptedArrayDisposer<Candera::WarpMatrix::ConstData, const Candera::F
 MatrixDataDisposer;
      Candera::WarpMatrix warpMatrix(stcWarpMatrix.usNumRefPointsX, stcWarpMatrix.usNumRefPoints
```

Calculate Warp Texture Bounds

```
      WRP_SFLOAT textureOriginX;
      WRP_SFLOAT textureOriginY;
      WRP_SFLOAT textureWidth;
      WRP_SFLOAT textureHeight;

      ComputeWarpImageBoundsFromConfig(&stcWarpConfigParams, &textureOriginX, &textureOriginY, &
      Candera::Rectangle
 warpTextureBounds(textureOriginX, textureOriginY, textureWidth, textureHeight);
```

Render Warped Image to Display

```
      // define display parameters
      Candera::Display::CommonSettings params;
      Candera::MemoryPlatform::Set(&params, 0, sizeof(params));

      params.width =          800;
      params.hps =            832;
      params.hpe =            976;
      params.hpt =            1008;

      params.height =         600;
      params.vps =            612;
      params.vpe =            618;
      params.vpt =            631;

      params.refreshRate =    60;
      params.colorBits =      32;
```

```
// create display
Candera::Display *display = Candera::DevicePackageInterface::CreateDisplay(0);
if (display == 0) {
    FEATSTD_LOG_ERROR("Failed to create display.");
    return false;
}

//enable warping
display->SetWarpingEnabled(true);

// pass previously created Candera warpMatrix to display
display->SetWarpMatrix(warpMatrix);

// optionally, set the warp texture bounds
display->SetWarpImageBounds(warpTextureBounds);

// ApplyChanges() has to be called for the changes to be taken into account
display->ApplyChanges();

//upload display
if (!display->Upload(params)) {
    FEATSTD_LOG_ERROR("Failed to upload display.");
    return false;
}
```

> Warping should be enabled before the NativeHandle is attached to the display (attached either by AttachNativeHandle or created at display upload).

Create a Default WarpMatrix Data

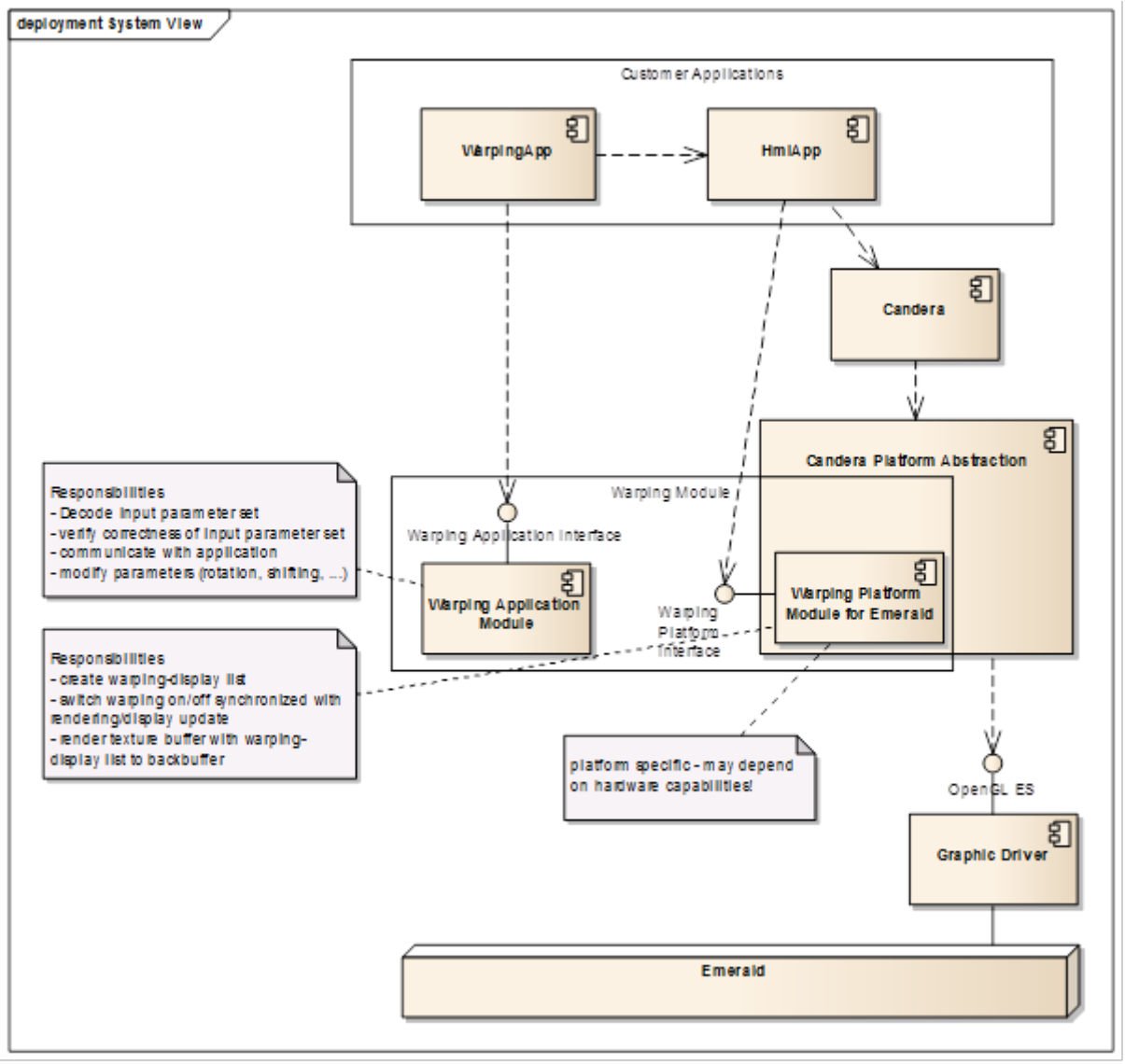
```
// retrieve display
Candera::Display *display = Candera::DevicePackageInterface::GetDisplay(0);

if (display == 0) {
  FEATSTD_LOG_ERROR("Failed to retrieve display.");
  return false;
}

// default warpMatrix data
static const Candera::Float data[] = {
          0.0F, 0.0F,
          1.0F, 0.0F,
          0.0F, 1.0F,
          1.0F, 1.0F,
      };

Candera::WarpMatrix warpMatrix(2, 2, data);
display->SetWarpMatrix(warpMatrix);

Candera::Rectangle defaultImageBounds(0.F, 0.F, 1.F, 1.F);
display->SetWarpImageBounds(defaultImageBounds);
```

Warping Library User Guide

This guide gives an overview on development and build folder as well as of the warping library's source code structure.
The goal is to guide users from build environment preparation through generation of warping component sample applications to the execution of these applications.

Overview

The system view shows the location of the warping library in the system as well as connections to other components.



Globally, the warping library consists of two parts, the warping application module and a warping platform module. The platform module is platform dependent – a version for Windows host platform and a version for iMX6 target hardware are available. Responsibility of each part is described in the system view model.

Directory Overview

The directory structure of the delivery package consist of the code *code* and the document *doc* sub directory.
The *code* directory structure in detail:

- *bin*: Output directory for the host and target "WarpingLibrary" library file (sub directory *WarpingLibrary*) and the executable file for the "WarpSample" application (sub directory *WarpSample*).
- *build*: This sub directory contains the build scripts for all supported host and target variants for "WarpingLibrary" (sub directory *WarpingLibrary*) and the "WarpSample" application (sub directory *WarpSample*).
- *sample*: Storage of the "WarpSample" application source code files.
- *src*: Contains matrix calculation source code needed for the warping library (sub directory \3rdParty) and the source code for the warping library (sub directory *WarpingLibrary*).
  The doc directory structure in detail:
- *UserDocumentation*: Location of this user's guide.
- *ReferenceMaterial* (optional): This sub directory contains the product requirement document (versions) being the base for the software requirements.

Integration Overview

Integration of the Candera Warping Library is performed in the following steps:

1. **Apply Delivery:** Copy the delivery contents to the installation directory of the CGI Studio base release package on your hard disk (for further reference, we call the path to this directory ***warp_base_path***). The ***warp_base_path*** should reside on the same level as the folder ***cgi_studio_candera***, containing the Candera source code, to make CMake auto-detecting of related Candera source code work. Besides Candera, also *cgi_studio_3psw* and *featstd* folders are needed. The build scripts delivered expect the Candera sources in these directories – in case on your machine these locations are different, please adapt the paths in the build scripts accordingly.
2. **Create Warping Library:** Building the Candera independent Warping Library needs to be done in a first step as this is not integrated into the Candera build scripts. For detailed instruction about this step please refer to chapter 2. Note that in case of binary delivery of Warping Library, there are pre-built libraries provided for all supported host and target variants.
3. **Build and Run Sample:** To build the Candera Warping sample application, follow the instructions described in 3. For running the sample application in the host environment, please make sure that your machine satisfies Candera's hardware and software requirements (especially concerning graphic hardware and graphic drivers).

Creation

Library for Host Environment

The source code for building the Warping Application Module (aka Warping library) is located in the *src/WarpingLibrary* sub directory.

In a console window, run one of the batch files referring to Visual Studio (example: *run_cmake_VS2010_build.bat*) to generate the Microsoft Visual Studio 2010 or 2013 solution and project files for starting the build environment. A successful CMake run will generate these files into directory *build/WarpingLibrary/<device_platform>Win32*.

To build the warping library, open *WarpingLibrary.sln* with MSVS2010/MSVS2013 and build the library either in Debug or Release mode. After successful build the library file *WarpingLibrary.lib* will be stored in a sub directory of *bin/WarpingLibrary/<device_platform>Win32*, depending on the build mode selected in MSVS.

Library for Integrity Target Environment

Open a MS Windows console window and run the batch file *run_cmake_<device_platform>_integrity_build.bat* (located in directory *src/WarpingLibrary*) to generate the device-specific Integrity solution and project files for starting the build environment. A successful CMake run will generate these files into directory *build/WarpingLibrary/<device_platform>Integrity*. The build process is started automatically. If necessary, the path to the MinGW compiler has to be adapted in the batch file.

After successful build the library file *libWarpingLibrary.a* will be stored in directory *bin/WarpingLibrary/<device_platform>Integrity*.

Library for Linux Target Environment

Use a native Linux based operating system (e.g. Ubuntu) or a version in a virtual environment (e.g. Oracle VirtualBox) to build the Warping Library for the device-specific Linux target environment.

In a Linux terminal, run the shell script “run_cmake_<device_platform_linux_build” (located in directory *src/WarpingLibrary*) to generate the project files for starting the build environment (these files will be generated by CMake into directory *build/WarpingLibrary/<device_platform>Linux*).

After successful build the library file *libWarpingLibrary.a* will be stored in directory *bin/WarpingLibrary/<device_platform>Linux*.

Build and Run Sample Applications

This page gives an overview and explanations on how to build and execute sample applications for different host or target systems.

Host Environment

Build Application WarpingSample1 for Windows Host

The source code of the warping sample application *WarpSample1* is stored in directory apps/WarpSample.

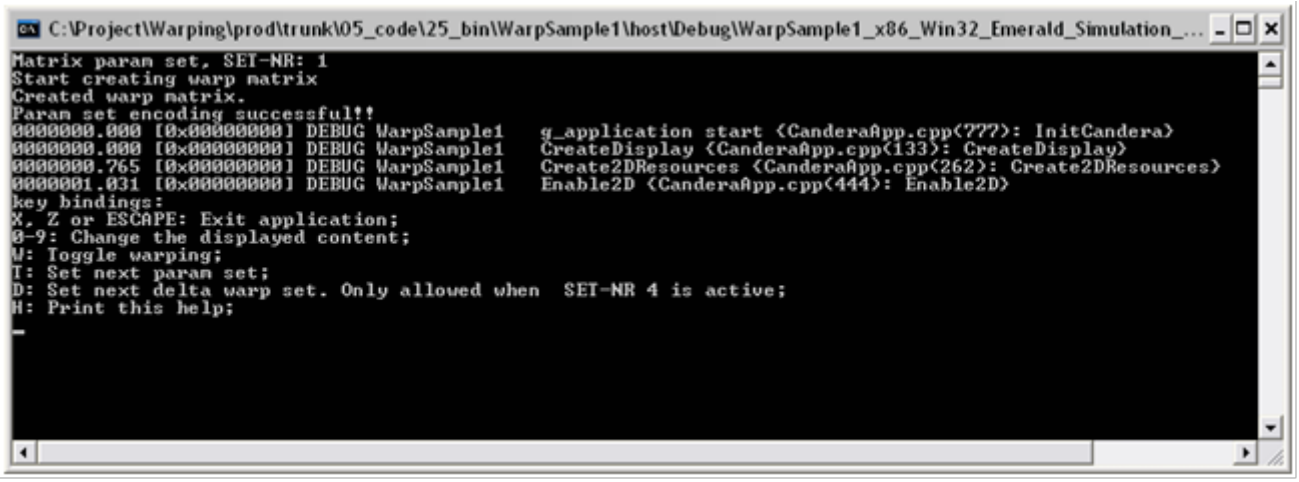
Depending on your needs, run the batch file *run_cmake_<device-platform>_<MSVC-version>_build.bat* in a windows console window to make CMake generate the project files and build environment for the WarpSample application into directory *build/WarpSample1/<device-platform>Win32*.

For starting the build environment, open the project file *WarpSample1.sln*. Building the project will – if necessary – create the required *Candera* libraries, link in the Warping Library and for a successful build create the executable *WarpSample_x86_Win32_<device-platform>_Simulation_MSVC.exe* into the directory *bin/WarpSample/<device-platform>Win32/Debug*.

Run Application WarpingSample on Windows Host

> Before you continue, please make sure that the computer satisfies the hardware requirements (e.g. concerning graphics card) and software requirements (e.g. needed graphic driver version) of CGI Studio/Candera Framework.

For starting the WarpingSample1 application, open a MS Windows command shell window and change to the directory of the executable (*bin/WarpSample/*<device-platform>*Win32/Debug*). Start the WarpingSample application by calling *WarpSample_x86_Win32_*<device-platform>*_Simulation_MSVC.exe* from the command line. Another option to start the WarpingSample application is to start a debugging session directly from the MSVS2010/MSVS2013 environment.

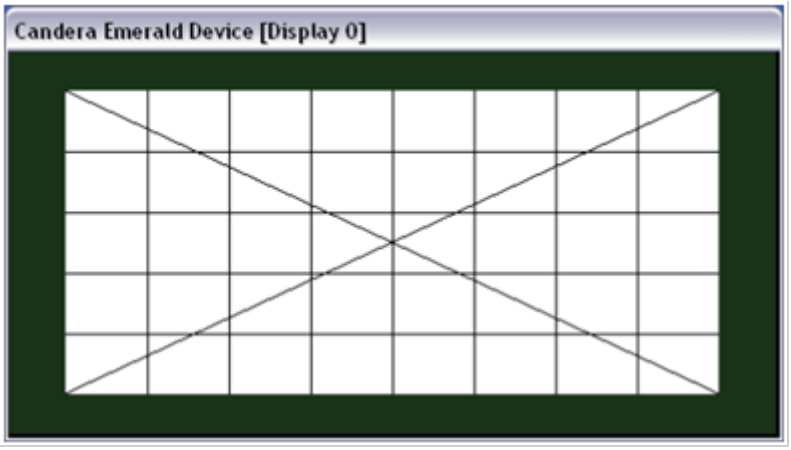


The last lines show the key bindings for handling the WarpingSample1 test software:

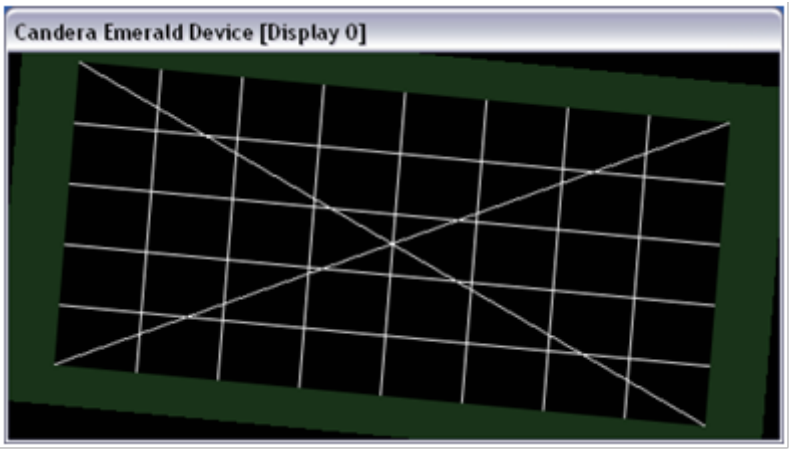
- *X, Z or ESCAPE: Exit application:* These keys have to be used to terminate the test software.
- *0-9: Change the displayed content:* The number keys are used to switch between different test scenes. Key 0 activates a scene implemented with Candera 2D rendering a predefined text, keys 1 and 2 activate scenes implemented in Candera 3D displaying fixed bitmaps. Keys 3 to 9 are reserved for future use.
- *W: Toggle warping:* This key switches warping on/off.
- *T: Set next param set:* Pressing this key will cycle the screen through 4 different used warping parameter sets.
  - SET-NR 1: Input parameter set without rotation and distortion.
  - SET-NR 2: Input parameter set with rotation and without distortion.
  - SET-NR 3: Input parameter set without rotation and with distortion.
  - SET-NR 4: Same parameter set as SET-NR 1, this is the base parameter set for the delta adjusting.
- *D: Set next delta warp set:* Pressing the D key will cycle the screen through all different delta adjusting modes in combination with each possible direction. This possibility is only enabled when the parameter set *SET-NR 4* is active.
- *H: Print this help:* The H key prints the overview of selectable keys on the command line.

Example Scenes

The following figures show example scenes with applied parameter sets:



Scene #1 with Parameter Set 1



Scene #2 with Parameter Set 2

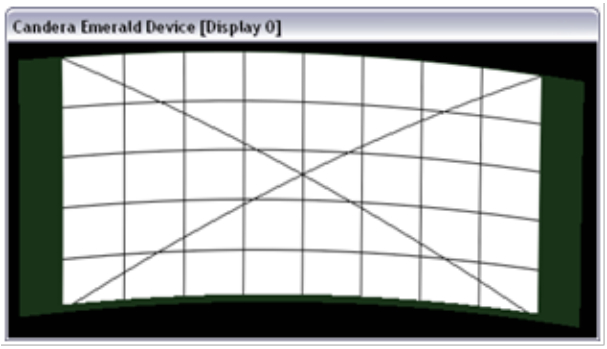
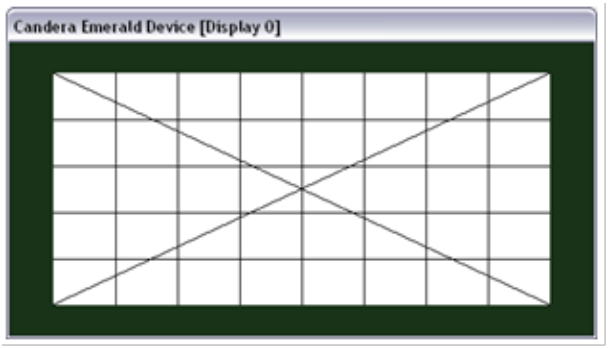
Scene #2 with Parameter Set 2

Further, the following screenshots show the effect of warping in comparison to the original image for the 3 scenes already used in the examples above when warping parameter set *SET-NR 3* is applied:

- **Scene #0**

warped	original
	

- **Scene #1**

warped	original
	

- **Scene #2**

warped	original