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);
```

Revision #3
Created 2023-02-21 06:53:03 UTC by Kanai Tomoaki
Updated 2023-06-19 01:40:46 UTC by Tsuyoshi.Kato