# libvisiongl

1.0

Generated by Doxygen 1.7.6.1

Fri Jul 5 2013 07:45:15

# Contents

# Chapter 1

# Directory Hierarchy

## 1.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

# Chapter 3

# Directory Documentation

## 3.1 src/ Directory Reference

### Files

- file **glsl2cpp_BG.h**
- file **glsl2cpp_shaders.h**
- file **glsl2cpp_Stereo.h**
- file **vglImage.cpp**

# Chapter 4

# File Documentation

## 4.1 src/glsl2cpp_BG.h File Reference

```
#include "vglImage.h"
```

### Functions

- void **vglDetectFGSimpleBGModel** (VglImage ∗img_in, VglImage ∗average, VglImage ∗variance, VglImage ∗foreground, float std_thresh)
- void **vglTrainSimpleBGModel** (VglImage ∗img_in, VglImage ∗average, VglImage ∗variance, float weight)
- void **vglUpdatePartialSimpleBGModel** (VglImage ∗img_in, VglImage ∗foregorundClose, VglImage ∗average, VglImage ∗variance, float weight)

### 4.1.1 Function Documentation

#### 4.1.1.1 void vglDetectFGSimpleBGModel ( VglImage ∗ *img_in,* VglImage ∗ *average,* VglImage ∗ *variance,* VglImage ∗ *foreground,* float *std_thresh* )

Detects foreground pixels.

#### 4.1.1.2 void vglTrainSimpleBGModel ( VglImage ∗ *img_in,* VglImage ∗ *average,* VglImage ∗ *variance,* float *weight* )

Updates average and variance of background model.

#### 4.1.1.3 void vglUpdatePartialSimpleBGModel ( VglImage * *img_in,* VglImage * *foregorundClose,* VglImage * *average,* VglImage * *variance,* float *weight* )

Updates average and variance of background model only in pixels that are classified as background.

## 4.2 src/glsl2cpp_shaders.h File Reference

```
#include "vglImage.h"
```

### Functions

- void **shader_15_1** (VglImage *src, VglImage *dst)
- void **vgl1to3Channels** (VglImage *src, VglImage *dst)
- void **vglAbsDiff** (VglImage *src0, VglImage *src1, VglImage *dst)
- void **vglAnd** (VglImage *src0, VglImage *src1, VglImage *dst)
- void **vglBaricenterInit** (VglImage *src, VglImage *dst)
- void **vglBlurSq3** (VglImage *src, VglImage *dst)
- void **vglClear2** (VglImage *src_dst, float r, float g, float b, float a=0.0)
- void **vglContrast** (VglImage *src, VglImage *dst, float factor)
- void **vglCoordToColor** (VglImage *dst)
- void **vglCopy** (VglImage *src, VglImage *dst)
- void **vglCrossingNumber** (VglImage *src, VglImage *dst)
- void **vglDeleteSkeletonCorners** (VglImage *src, VglImage *dst, int step)
- void **vglDeleteSkeletonWarts** (VglImage *src, VglImage *dst)
- void **vglDeleteSkeletonWarts2** (VglImage *src, VglImage *dst)
- void **vglDiff** (VglImage *src0, VglImage *src1, VglImage *dst)
- void **vglDilateCross3** (VglImage *src, VglImage *dst)
- void **vglDilateSq3** (VglImage *src, VglImage *dst)
- void **vglErodeCross3** (VglImage *src, VglImage *dst)
- void **vglErodeHL3** (VglImage *src, VglImage *dst)
- void **vglErodeHL5** (VglImage *src, VglImage *dst)
- void **vglErodeHL7** (VglImage *src, VglImage *dst)
- void **vglErodeSq3** (VglImage *src, VglImage *dst)
- void **vglErodeSq3off** (VglImage *src, VglImage *dst)
- void **vglErodeSq5** (VglImage *src, VglImage *dst)
- void **vglErodeSq5off** (VglImage *src, VglImage *dst)
- void **vglErodeSq7** (VglImage *src, VglImage *dst)
- void **vglErodeSqSide** (VglImage *src, VglImage *dst, int side)
- void **vglErodeVL3** (VglImage *src, VglImage *dst)
- void **vglErodeVL5** (VglImage *src, VglImage *dst)
- void **vglErodeVL7** (VglImage *src, VglImage *dst)
- void **vglFeaturePoints** (VglImage *src, VglImage *dst, int type)
- void **vglGaussianBlurSq3** (VglImage *src, VglImage *dst)
- void **vglGray** (VglImage *src, VglImage *dst)

- void **vglHorizontalFlip** (VglImage ∗src, VglImage ∗dst)
- void **vglInOut** (VglImage ∗src, VglImage ∗dst)
- void **vglJulia** (VglImage ∗dst, float ox=0.0, float oy=0.0, float half_win=1.0, float c_real=-1.36, float c_imag=.11)
- void **vglLaplaceSq3** (VglImage ∗src, VglImage ∗dst)
- void **vglMandel** (VglImage ∗dst, float ox=0.0, float oy=0.0, float half_win=1.0)
- void **vglMipmap** (VglImage ∗src, VglImage ∗dst, float lod)
- void **vglMulScalar** (VglImage ∗src, VglImage ∗dst, float factor)
- void **vglMultiInput** (VglImage ∗src0, VglImage ∗src1, VglImage ∗dst, float weight=.5)
- void **vglMultiOutput** (VglImage ∗src, VglImage ∗dst, VglImage ∗dst1)
- void **vglNoise** (VglImage ∗src, VglImage ∗dst)
- void **vglNot** (VglImage ∗src, VglImage ∗dst)
- void **vglOr** (VglImage ∗src0, VglImage ∗src1, VglImage ∗dst)
- void **vglRescale** (VglImage ∗src, VglImage ∗dst, float x0, float y0, float x1, float y1)
- void **vglRgbToBgr** (VglImage ∗src, VglImage ∗dst)
- void **vglRgbToHsl** (VglImage ∗src, VglImage ∗dst)
- void **vglRgbToHsv** (VglImage ∗src, VglImage ∗dst)
- void **vglRgbToXyz** (VglImage ∗src, VglImage ∗dst)
- void **vglRobertsGradient** (VglImage ∗src, VglImage ∗dst)
- void **vglSelfSum22** (VglImage ∗src, VglImage ∗dst)
- void **vglSelfSum3v** (VglImage ∗src, VglImage ∗dst)
- void **vglSelfSum4h** (VglImage ∗src, VglImage ∗dst)
- void **vglSelfSum5h** (VglImage ∗src, VglImage ∗dst)
- void **vglSelfSum5v** (VglImage ∗src, VglImage ∗dst)
- void **vglSharpenSq3** (VglImage ∗src, VglImage ∗dst)
- void **vglSobelGradient** (VglImage ∗src, VglImage ∗dst)
- void **vglSobelXSq3** (VglImage ∗src, VglImage ∗dst)
- void **vglSobelYSq3** (VglImage ∗src, VglImage ∗dst)
- void **vglSum** (VglImage ∗src0, VglImage ∗src1, VglImage ∗dst)
- void **vglSumWeighted** (VglImage ∗src0, VglImage ∗src1, VglImage ∗dst, float weight=.5)
- void **vglSwapRGB** (VglImage ∗src, VglImage ∗dst)
- void **vglTestInOut** (VglImage ∗src_dst, float r, float g, float b, float a=0.0)
- void **vglTestInOut2** (VglImage ∗src_dst, VglImage ∗dst)
- void **vglTestMultiInput** (VglImage ∗src0, VglImage ∗src1, VglImage ∗dst, float weight=.5)
- void **vglTestMultiOutput** (VglImage ∗src, VglImage ∗dst, VglImage ∗dst1)
- void **vglTeste** (VglImage ∗src, VglImage ∗dst)
- void **vglThinBernardAux** (VglImage ∗src, VglImage ∗eroded, VglImage ∗dst)
- void **vglThinChinAux** (VglImage ∗src, VglImage ∗dst)
- void **vglThresh** (VglImage ∗src, VglImage ∗dst, float thresh, float top=1.0)
- void **vglThreshLevelSet** (VglImage ∗src, VglImage ∗dst, float thresh, float top=1.0)
- void **vglVerticalFlip** (VglImage ∗src, VglImage ∗dst)
- void **vglWhiteRohrerEdge** (VglImage ∗src, VglImage ∗dst, float radius)
- void **vglXGY** (VglImage ∗src, VglImage ∗dst)
- void **vglZoom** (VglImage ∗src, VglImage ∗dst, float factor)

### 4.2.1 Function Documentation

#### 4.2.1.1 void shader_15_1 ( VglImage * *src,* VglImage * *dst* )

#### 4.2.1.2 void vgl1to3Channels ( VglImage * *src,* VglImage * *dst* )

Convert grayscale image to RGB

#### 4.2.1.3 void vglAbsDiff ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst* )

Absolute difference between two images.

Referenced by vglGetLevelDistTransform5().

#### 4.2.1.4 void vglAnd ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst* )

Logical AND between two images

#### 4.2.1.5 void vglBaricenterInit ( VglImage * *src,* VglImage * *dst* )

Initialize image to be used in baricenter calculation. The initialization is done by storing the values (1, x, y) in each output pixel so that the summation over th whole image gives the three moments of the image.

R = f(x, y)

G = x * f(x, y)

B = y * f(x, y)

Referenced by vglBaricenterVga().

#### 4.2.1.6 void vglBlurSq3 ( VglImage * *src,* VglImage * *dst* )

vglBlurSq3

Blur image by 3x3 square structuring element.

#### 4.2.1.7 void vglClear2 ( VglImage * *src_dst,* float *r,* float *g,* float *b,* float *a* = 0.0 )

Clear image with given color.

#### 4.2.1.8 void vglContrast ( VglImage * *src,* VglImage * *dst,* float *factor* )

Changes contrast of image by given factor.

#### 4.2.1.9 void vglCoordToColor ( VglImage * *dst* )

Shows coordinates of pixels as colors. Red is horizontal and green is vertical. - Coordinates and colors are defined by OpenGL, that is, between 0 and 1.

#### 4.2.1.10 void vglCopy ( VglImage * *src,* VglImage * *dst* )

Direct copy from src to dst.

Referenced by vglCopyCreateImage(), vglDistTransform5(), vglGetLevelDistTransform5(), vglThinBernard(), and vglThinChin().

#### 4.2.1.11 void vglCrossingNumber ( VglImage * *src,* VglImage * *dst* )

Crossing number is defined as the number of ocurrences of the pattern 01 in the neihborhood of a pixel.

Neighborhood of pixel P is indexed as follows:

$$\begin{matrix} P3 & P2 & P1 \\ P4 & P & P0/8 \\ P5 & P6 & P7 \end{matrix}$$

References:

M. Couprie, Note on fifteen 2D parallel thinning algorithms, 2006

T. M. Bernard and A. Manzanera, Improved low complexity fully parallel thinning algorithms, 1999

#### 4.2.1.12 void vglDeleteSkeletonCorners ( VglImage * *src,* VglImage * *dst,* int *step* )

Deletes corner from skeleton.

Receive as input the image with the skeleton to be thinned. Receives also the step. must be called once with step 1 and once with step 2.

Neighborhood pixels is indexed as follows:

$$\begin{matrix} P3 & P2 & P1 \\ P4 & P8 & P0 \\ P5 & P6 & P7 \end{matrix}$$

Pixels deleted are the ones that mach the pattern and its rotations by 90deg.

$$\begin{matrix} 0 & 0 & x \\ 0 & 1 & 1 \\ x & 1 & 0 \end{matrix}$$

References:

M. Couprie, Note on fifteen 2D parallel thinning algorithms, 2006

T. M. Bernard and A. Manzanera, Improved low complexity fully parallel thinning algorithms, 1999

#### 4.2.1.13 void vglDeleteSkeletonWarts ( VglImage * *src,* VglImage * *dst* )

Deletes warts from skeleton. Receive as input the image with the skeleton to be thinned. Neighborhood pixels are indexed as follows:

$$\begin{array}{ccc} P3 & P2 & P1 \\ P4 & P & P0/8 \\ P5 & P6 & P7 \end{array}$$

Pixels deleted are the ones that mach the pattern and its rotations by 45deg.

$$\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{array}$$

That is the same as delete the pixels with crossing number = 1 and neighbor number = 3

References:

Ke Liu et al., Identification of fork points on the skeletons of handwritten chinese characters

#### 4.2.1.14 void vglDeleteSkeletonWarts2 ( VglImage * *src,* VglImage * *dst* )

Deletes warts from skeleton. Receive as input the image with the skeleton to be thinned. Neighborhood pixels are indexed as follows:

P3 P2 P1

P4 P P0/8

P5 P6 P7

Pixels deleted are the ones that mach the pattern and its rotations by 45deg.

1 0 0 1 1 0 1 0 0

1 1 0 1 1 0 1 0 0

1 1 0 1 1 0 1 1 0

1 1 1 1 1 0 1 1 0

1 1 1 1 1 0 1 1 1

That is the same as delete the pixels with crossing number = 1 and neighbor number >=3

References:

Ke Liu et al., Identification of fork points on the skeletons of handwritten chinese characters

#### 4.2.1.15 void vglDiff ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst* )

Image src0 minus src1.

#### 4.2.1.16 void vglDilateCross3 ( VglImage * *src,* VglImage * *dst* )

Dilation of image by 3x3 cross structuring element.

#### 4.2.1.17 void vglDilateSq3 ( VglImage * *src,* VglImage * *dst* )

Dilation of image by 3x3 square structuring element.

Referenced by vglCloseSq3(), and vglOpenSq3().

#### 4.2.1.18 void vglErodeCross3 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by 3x3 cross structuring element.

Referenced by vglCErodeCross3(), vglDistTransform5(), vglDistTransformCross3(), vglGetLevelDistTransform5(), and vglThinBernard().

#### 4.2.1.19 void vglErodeHL3 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by horizontal line with 3 pixels.

Referenced by vglErodeSq3Sep().

#### 4.2.1.20 void vglErodeHL5 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by horizontal line with 5 pixels.

Referenced by vglErodeSq5Sep().

#### 4.2.1.21 void vglErodeHL7 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by horizontal line with 7 pixels.

#### 4.2.1.22 void vglErodeSq3 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by 3x3 square structuring element.

Referenced by vglCloseSq3(), vglDistTransform5(), vglDistTransformSq3(), vglGetLevelDistTransform5(), and vglOpenSq3().

#### 4.2.1.23 void vglErodeSq3off ( VglImage * *src,* VglImage * *dst* )

Erosion of image by 3x3 square structuring element. Uses an offset array with 9 elements. Slower than vglErodeSq3.

#### 4.2.1.24 void vglErodeSq5 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by 5x5 square structuring element.

#### 4.2.1.25 void vglErodeSq5off ( VglImage * *src,* VglImage * *dst* )

Erosion of image by 3x3 square structuring element. Uses an offset array with 25 elements. Slower than vglErodeSq5.

#### 4.2.1.26 void vglErodeSq7 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by 7x7 square structuring element.

#### 4.2.1.27 void vglErodeSqSide ( VglImage * *src,* VglImage * *dst,* int *side* )

Erosion of image by square structuring element. The parameter "side" is the dimension of the square side in pixels.

#### 4.2.1.28 void vglErodeVL3 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by vertical line with 3 pixels.

Referenced by vglErodeSq3Sep().

#### 4.2.1.29 void vglErodeVL5 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by vertical line with 5 pixels.

Referenced by vglErodeSq5Sep().

#### 4.2.1.30 void vglErodeVL7 ( VglImage * *src,* VglImage * *dst* )

Erosion of image by vertical line with 7 pixels.

#### 4.2.1.31 void vglFeaturePoints ( VglImage * *src,* VglImage * *dst,* int *type* )

Feature Points are defined as function of the crossing number and number of neighbors of a pixel.

The number of neighbors is indicated as Nb. Crossing number is defined as

Nc = number of occurrences of the pattern 01 in the neighborhood of P

Neighborhood pixels are indexed as follows:

P3 P2 P1

P4 P P0

P5 P6 P7

All the ending points are feature points. Are defined as Se = { P | Nc(P) = 1 }

Feature points type 1, denoted as S1, are defined as S1 = { P | Nc(P) >= 3}

Feature points type 2, denoted as S2, are defined as S1 = { P | Nb(P) >= 3}

Feature points type 3, denoted as S3, are defined as S3 = { P | Nc(P) >= 3 or Nb(P) >= 4 }

References:

Ke Liu et al., Identification of fork points on the skeletons of handwritten chinese characters

#### 4.2.1.32 void vglGaussianBlurSq3 ( VglImage * *src,* VglImage * *dst* )

Blurs image by 3x3 square gaussian structuring element.

#### 4.2.1.33 void vglGray ( VglImage * *src,* VglImage * *dst* )

Convert image to grayscale by calculating the scalar product of (r, g, b) and (.2125, .7154, .0721).

#### 4.2.1.34 void vglHorizontalFlip ( VglImage * *src,* VglImage * *dst* )

Flip image horizontally i.e. left becomes right.

Image flip done by shader.

#### 4.2.1.35 void vglInOut ( VglImage * *src,* VglImage * *dst* )

vglInOut

Test and model for IN_OUT semantics

#### 4.2.1.36 void vglJulia ( VglImage * *dst,* float *ox* = `0.0`, float *oy* = `0.0`, float *half_win* = `1.0`, float *c_real* = `-1.36`, float *c_imag* = `.11` )

Calculate Julia set

#### 4.2.1.37 void vglLaplaceSq3 ( VglImage * *src,* VglImage * *dst* )

Laplacian of image by 3x3 square structuring element.

#### 4.2.1.38 void vglMandel ( VglImage * *dst*, float *ox* = 0.0, float *oy* = 0.0, float *half_win* = 1.0 )

Calculate Mandelbrot set

#### 4.2.1.39 void vglMipmap ( VglImage * *src*, VglImage * *dst*, float *lod* )

Get specified level of detail.

#### 4.2.1.40 void vglMulScalar ( VglImage * *src*, VglImage * *dst*, float *factor* )

Multiply image by scalar.

#### 4.2.1.41 void vglMultiInput ( VglImage * *src0*, VglImage * *src1*, VglImage * *dst*, float *weight* = .5 )

VglAdd

Sum of two images.

#### 4.2.1.42 void vglMultiOutput ( VglImage * *src*, VglImage * *dst*, VglImage * *dst1* )

vglGray

Convert image to grayscale

#### 4.2.1.43 void vglNoise ( VglImage * *src*, VglImage * *dst* )

Add gaussian noise to image

#### 4.2.1.44 void vglNot ( VglImage * *src*, VglImage * *dst* )

Inverts image.

#### 4.2.1.45 void vglOr ( VglImage * *src0*, VglImage * *src1*, VglImage * *dst* )

Logical OR between two images

Referenced by vglCErodeCross3().

#### 4.2.1.46 void vglRescale ( VglImage * *src*, VglImage * *dst*, float *x0*, float *y0*, float *x1*, float *y1* )

Rescales corners of image to given corners

#### 4.2.1.47 void vglRgbToBgr ( VglImage * *src,* VglImage * *dst* )

Converts image RGB to BGR color space

#### 4.2.1.48 void vglRgbToHsl ( VglImage * *src,* VglImage * *dst* )

Converts image RGB to HSL color space

#### 4.2.1.49 void vglRgbToHsv ( VglImage * *src,* VglImage * *dst* )

Converts image RGB to HSV color space

#### 4.2.1.50 void vglRgbToXyz ( VglImage * *src,* VglImage * *dst* )

Converts image RGB to XYZ color space.

#### 4.2.1.51 void vglRobertsGradient ( VglImage * *src,* VglImage * *dst* )

Roberts gradient of image

#### 4.2.1.52 void vglSelfSum22 ( VglImage * *src,* VglImage * *dst* )

Stores in output pixel the sum of 4 adjacent pixels of the input image. The width and height of the output image must be half of the input image.

Referenced by vglBaricenterVga().

#### 4.2.1.53 void vglSelfSum3v ( VglImage * *src,* VglImage * *dst* )

Stores in output pixel the sum of 3 adjacent pixels of the input image. The height of the output image must be 1/3th of the input image.

Referenced by vglBaricenterVga().

#### 4.2.1.54 void vglSelfSum4h ( VglImage * *src,* VglImage * *dst* )

Stores in output pixel the sum of 4 adjacent pixels of the input image. The width of the output image must be 1/4th of the input image.

Referenced by vglBaricenterVga().

#### 4.2.1.55 void vglSelfSum5h ( VglImage * *src,* VglImage * *dst* )

Stores in output pixel the sum of 5 adjacent pixels of the input image. The width of the output image must be 1/5th of the input image.

Referenced by vglBaricenterVga().

#### 4.2.1.56 void vglSelfSum5v ( VglImage * *src,* VglImage * *dst* )

Stores in output pixel the sum of 5 adjacent pixels of the input image. The height of the output image must be 1/5th of the input image.

Referenced by vglBaricenterVga().

#### 4.2.1.57 void vglSharpenSq3 ( VglImage * *src,* VglImage * *dst* )

Sharpens image using 3x3 square window.

#### 4.2.1.58 void vglSobelGradient ( VglImage * *src,* VglImage * *dst* )

Sobel gradient of image

#### 4.2.1.59 void vglSobelXSq3 ( VglImage * *src,* VglImage * *dst* )

Sobel edge filtering in X direction.

#### 4.2.1.60 void vglSobelYSq3 ( VglImage * *src,* VglImage * *dst* )

Sobel edge filtering in Y direction.

#### 4.2.1.61 void vglSum ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst* )

Sum of two images.

Referenced by vglDistTransform5(), vglDistTransformCross3(), and vglDistTransform-Sq3().

#### 4.2.1.62 void vglSumWeighted ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst,* float *weight* = .5 )

Weighted sum of two images. The first image is multiplied by weight, and the second, by 1 - weight. Default weight is 0.5.

#### 4.2.1.63 void vglSwapRGB ( VglImage * *src,* VglImage * *dst* )

Convert image from RGB to BGR and vice versa.

#### 4.2.1.64 void vglTeste ( VglImage * *src,* VglImage * *dst* )

vglDilate

Dilation of image by 3x3 square structuring element.

#### 4.2.1.65 void vglTestInOut ( VglImage * *src_dst,* float *r,* float *g,* float *b,* float *a* = 0.0 )

Test and model for IN_OUT semantics

#### 4.2.1.66 void vglTestInOut2 ( VglImage * *src_dst,* VglImage * *dst* )

Test and model for IN_OUT semantics, with double output.

#### 4.2.1.67 void vglTestMultiInput ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst,* float *weight* = .5 )

Test and model for multiple input functions.

#### 4.2.1.68 void vglTestMultiOutput ( VglImage * *src,* VglImage * *dst,* VglImage * *dst1* )

Test and model for multiple output functions.

#### 4.2.1.69 void vglThinBernardAux ( VglImage * *src,* VglImage * *eroded,* VglImage * *dst* )

Return one step of thinning. Algorithm by Bernard and Manzanera 1999. Receive as input the image to be thinned and its erosion by a elementary cross structuring element. Neighborhood pixels are indexed as follows:

$P3$ $P2$ $P1$
$P4$ $P8$ $P0$
$P5$ $P6$ $P7$

References:

M. Couprie, Note on fifteen 2D parallel thinning algorithms, 2006

T. M. Bernard and A. Manzanera, Improved low complexity fully parallel thinning algorithms, 1999

Referenced by vglThinBernard().

#### 4.2.1.70 void vglThinChinAux ( VglImage * *src,* VglImage * *dst* )

Return one step of thinning. Algorithm by Chin, Wan Stover and Iverson, 1987. - Receive as input the image to be thinned, buffer image and number of times to iterate. Neighborhood pixels are indexed as follows:

$$\begin{array}{ccccc} x & x & P10 & x & x \\ x & P3 & P2 & P1 & x \\ P11 & P4 & P0 & P8 & P9 \\ x & P5 & P6 & P7 & x \\ x & x & P12 & x & x \end{array}$$

References:

M. Couprie, Note on fifteen 2D parallel thinning algorithms, 2006

R. T. Chin et al., A one-pass thinning algorithm and its parallel implementation, 1987

Referenced by vglThinChin().

#### 4.2.1.71 void vglThresh ( VglImage * *src,* VglImage * *dst,* float *thresh,* float *top* = 1.0 )

Threshold of image. If value is greater than threshold, output is top, else, output is 0. Default top value is 1.

Referenced by vglDistTransform5(), vglDistTransformCross3(), and vglDistTransform-Sq3().

#### 4.2.1.72 void vglThreshLevelSet ( VglImage * *src,* VglImage * *dst,* float *thresh,* float *top* = 1.0 )

Threshold of image. If value is equal to level, output is top, else, output is 0. Default top value is 1. Use after some Distance Transform to get a single distance level set.

#### 4.2.1.73 void vglVerticalFlip ( VglImage * *src,* VglImage * *dst* )

Flip image vertically i.e. top becomes bottom.

Image flip done by shader.

#### 4.2.1.74 void vglWhiteRohrerEdge ( VglImage * *src,* VglImage * *dst,* float *radius* )

Finds edge by using a White-Rohrer mask.

#### 4.2.1.75 void vglXGY ( VglImage * *src,* VglImage * *dst* )

Stores sobel edge filtering in X direction in red channel grayscale in y and sobel edge filtering in Y direction in green channel

#### 4.2.1.76 void vglZoom ( VglImage * *src,* VglImage * *dst,* float *factor* )

Zoom image by factor.

## 4.3 src/glsl2cpp_Stereo.h File Reference

```
#include "vglImage.h"
```

### Functions

- void **vglAbsDiffDisparity** (VglImage ∗img_ref, VglImage ∗img_2, VglImage ∗dst, float disparity)
- void **vglAbsDiffDisparityMipmap** (VglImage ∗img_ref, VglImage ∗img_2, VglImage ∗dst, float disparity, float max_lod)
- void **vglFindDisparity** (VglImage ∗img_dif, VglImage ∗img_disp, float disparity)
- void **vglFindDisparityDiff** (VglImage ∗img_sum, VglImage ∗img_disp, VglImage ∗img_best, float disparity)
- void **vglGreenDiffDisparity** (VglImage ∗img_ref, VglImage ∗img_2, VglImage ∗dst, float disparity)
- void **vglHomography** (VglImage ∗img_src, VglImage ∗img_dst, float ∗f_homo)
- void **vglMapTo3D** (VglImage ∗img_map, VglImage ∗img_3d, float f, float b, float D, float disp_k=0.0, float h=10.0)
- void **vglMeanMipmap** (VglImage ∗img_dif, VglImage ∗img_out, float max_lod)
- void **vglMeanSq3** (VglImage ∗img_dif, VglImage ∗img_out)
- void **vglRectify** (VglImage ∗img_src, VglImage ∗img_dst, float ∗f_dist, float ∗f_proj, float ∗f_homo)
- void **vglSumDiff** (VglImage ∗img_dif, VglImage ∗img_out)
- void **vglSumDiffMipmap** (VglImage ∗img_dif, VglImage ∗img_out, float max_lod)
- void **vglUndistort** (VglImage ∗img_src, VglImage ∗img_dst, float ∗f_dist, float ∗f_proj)

### 4.3.1 Function Documentation

#### 4.3.1.1 void vglAbsDiffDisparity ( VglImage ∗ *img_ref,* VglImage ∗ *img_2,* VglImage ∗ *dst,* float *disparity* )

Calculate absolute difference between img_ref and img_2. Disparities considered are in the closed interval [4∗disparity, 4∗disparity+3].

The four differences are stored in the RGBA image dst.

#### 4.3.1.2 void vglAbsDiffDisparityMipmap ( VglImage ∗ *img_ref,* VglImage ∗ *img_2,* VglImage ∗ *dst,* float *disparity,* float *max_lod* )

Calculates average absolute difference between img_ref and img_2 at levels of detail in [0, max_lod]. Disparities considered are in the closed interval [4∗disparity, 4∗disparity+3].

The four differences are stored in the RGBA image dst.

#### 4.3.1.3 void vglFindDisparity ( VglImage * *img_dif,* VglImage * *img_disp,* float *disparity* )

Find best disparity. The first input image, img_dif, contains absolute differences between a pair of images at disparities [4∗disparity, 4∗disparity+3].

The second input image contains the smallest differences found in channel R, and corresponding disparity value in channel A, Is also an output image, and is updated whenever a smaller difference is found.

#### 4.3.1.4 void vglFindDisparityDiff ( VglImage * *img_sum,* VglImage * *img_disp,* VglImage * *img_best,* float *disparity* )

Do the same as vglFindDisparity, but the smallest difference is stored in img_best, and corresponding disparity in img_disp. Both are input and output images.

#### 4.3.1.5 void vglGreenDiffDisparity ( VglImage * *img_ref,* VglImage * *img_2,* VglImage * *dst,* float *disparity* )

Calculate absolute difference between green channel of img_ref and img_2. Disparities considered are in the closed interval [4∗disparity, 4∗disparity+3].

The four differences are stored in the RGBA image dst.

#### 4.3.1.6 void vglHomography ( VglImage * *img_src,* VglImage * *img_dst,* float * *f_homo* )

Apply homography in img_src and stores result in img_dst.

Important: for matrices the cmponents are written in column major order:

$\texttt{mat2 m} = \texttt{mat2}(1, 2, 3, 4) \Leftrightarrow \texttt{m} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$

In C we build the matrix in line major order, then we must transpose tbe matrix before using it in OpenGL context.

#### 4.3.1.7 void vglMapTo3D ( VglImage * *img_map,* VglImage * *img_3d,* float *f,* float *b,* float *D,* float *disp_k* = `0.0`, float *h* = `10.0` )

Convert depth map to affine reconstruction

This algorithm ignores the infinite homography.

img_map: input depth map

img_3d: output reconstruction

f: focal length in pixels

b: baseline in cm

D: fixation point or maximum depth

is_float: if true, output image will store z in cm. If false output image will store z as 255 * (depth / D). if depth == D then z = 0.

disp_k: If set, single disparity will be used.

h: height of camera in cm

#### 4.3.1.8 void vglMeanMipmap ( VglImage * *img_dif,* VglImage * *img_out,* float *max_lod* )

Mean of pixel values of levels of detail in [0, max_lod]. Result is stored in img_out.

#### 4.3.1.9 void vglMeanSq3 ( VglImage * *img_dif,* VglImage * *img_out* )

Mean filter with a 3x3 square mask.

#### 4.3.1.10 void vglRectify ( VglImage * *img_src,* VglImage * *img_dst,* float * *f_dist,* float * *f_proj,* float * *f_homo* )

Undistort, correct projection and rectify img_src and stores result in img_dst, for use with stereo algorithm

The input float array f_dist contains the coefficient of radial distortion, and f_proj contains the intrinsinc parameters of the camera: center of projection (x and y); focal length in pixels (x and y). The focal lengths are the same when the pixels are square.

The input float array f_homo contains the homography that rectifies the image.

Important: for matrices the cmponents are written in column major order:

$\texttt{mat2 m} = \texttt{mat2}\,(1, 2, 3, 4) \Leftrightarrow \texttt{m} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$

In C we build the matrix in line major order, then we must transpose tbe matrix before using it in OpenGL context.

#### 4.3.1.11 void vglSumDiff ( VglImage * *img_dif,* VglImage * *img_out* )

VglSumDiff

Sum of differences

#### 4.3.1.12 void vglSumDiffMipmap ( VglImage * *img_dif,* VglImage * *img_out,* float *max_lod* )

VglSumDiffMipmap

Sum of differences

#### 4.3.1.13 void vglUndistort ( VglImage * *img_src,* VglImage * *img_dst,* float * *f_dist,* float * *f_proj* )

Correct camera lens distortion of img_src and stores the result in img_dst.

The input float array f_dist contains the coefficient of radial distortion, and f_proj contains the intrinsinc parameters of the camera: center of projection (x and y); focal length in pixels (x and y). The focal lengths are the same when the pixels are square.

Reference:

`http://www.cognotics.com/opencv/docs/1.0/ref/opencvref_cv.htm#cv_3d`

## 4.4 src/vglImage.cpp File Reference

`#include <iostream> #include <GL/freeglut_std.h> #include <GL/freeglut_ext.h> #include <opencv2/highgui/highgui.hpp> #include "vglContext.h" #include "vglImage.h" #include "vglLoadShader.h" #include "glsl2cpp_shaders.h"`

### Functions

- int **vglInit** ()
- int **vglInit** (int w, int h)
- void **vglUpload** (VglImage *image, int swapRGB)
- VglImage * **vglCopyCreateImage** (VglImage *img_in)
- VglImage * **vglCopyCreateImage** (IplImage *img_in, int dim3, int has_mipmap)
- VglImage * **vglCreateImage** (VglImage *img_in)
- VglImage * **vglCreateImage** (IplImage *img_in, int dim3, int has_mipmap)
- VglImage * **vglCreateImage** (CvSize size, int depth, int nChannels, int dim3, int has_mipmap)
- void **vglReleaseImage** (VglImage **p_image)
- void **vglReplaceIpl** (VglImage *image, IplImage *new_ipl)
- void **vglDownloadFaster** (VglImage *image)
- void **vglDownload** (VglImage *image)
- void **vglDownloadFBO** (VglImage *image)
- void **vglDownloadPPM** (VglImage *image)
- void **vglDownloadPGM** (VglImage *image)
- VglImage * **vglLoadImage** (char *filename, int iscolor, int has_mipmap)
- void **iplPrintImageInfo** (IplImage *ipl)
- void **vglPrintImageInfo** (VglImage *image)
- void **vglCopyImageTex** (VglImage *src, VglImage *dst)
- void **vglCopyImageTexFS** (VglImage *src, VglImage *dst)
- void **vglCopyImageTexVFS** (VglImage *src, VglImage *dst)
- void **vglVerticalFlip2** (VglImage *src, VglImage *dst)
- void **vglHorizontalFlip2** (VglImage *src, VglImage *dst)

- void **vglClear** (VglImage ∗image, float r, float g, float b, float a)
- void **vglOpenSq3** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, int times)
- void **vglCloseSq3** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, int times)
- void **vglErodeSq3Sep** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, int times)
- void **vglErodeSq5Sep** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, int times)
- void **vglCErodeCross3** (VglImage ∗src, VglImage ∗mask, VglImage ∗dst, VglImage ∗buf, int times)
- int **SavePPM** (char ∗filename, int w, int h, void ∗savebuf)
- int **vglSavePPM** (VglImage ∗img, char ∗filename)
- int **SavePGM** (char ∗filename, int w, int h, void ∗savebuf)
- int **vglSavePGM** (VglImage ∗img, char ∗filename)
- IplImage ∗ **LoadPGM** (char ∗filename)
- VglImage ∗ **vglLoadPGM** (char ∗filename)
- int **SaveYUV411** (char ∗filename, int w, int h, void ∗savebuf)
- void **vglDistTransformCross3** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, VglImage ∗buf2, int times)
- void **vglDistTransformSq3** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, VglImage ∗buf2, int times)
- void **vglDistTransform5** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, VglImage ∗buf2, int times)
- void **vglGetLevelDistTransform5** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, VglImage ∗buf2, int times)
- void **vglThinBernard** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, int times)
- void **vglThinChin** (VglImage ∗src, VglImage ∗dst, VglImage ∗buf, int times)
- void **vglBaricenterVga** (VglImage ∗src, double ∗x_avg, double ∗y_avg, double ∗pix_count)
- void **vglMultiOutput_model** (VglImage ∗src, VglImage ∗dst, VglImage ∗dst1)
- void **vglInOut_model** (VglImage ∗dst, VglImage ∗dst1)
- void **vglMultiInput_model** (VglImage ∗src0, VglImage ∗src1, VglImage ∗dst)

### 4.4.1 Function Documentation

#### 4.4.1.1 void iplPrintImageInfo ( IplImage ∗ *ipl* )

Print information about image.

Print width, height, depth and number of channels

#### 4.4.1.2 IplImage∗ LoadPGM ( char ∗ *filename* )

Load image data from PGM file, 1 channel, unsigned byte

Referenced by vglLoadPGM().

#### 4.4.1.3 int SavePGM ( char ∗ *filename,* int *w,* int *h,* void ∗ *savebuf* )

Save image data to PGM file, 1 channel, unsigned byte

Referenced by vglSavePGM().

#### 4.4.1.4 int SavePPM ( char * *filename,* int *w,* int *h,* void * *savebuf* )

Save image data to PPM file, 3 channels, unsigned byte

Time to save a VGA image = 3.5ms

Referenced by vglSavePPM().

#### 4.4.1.5 int SaveYUV411 ( char * *filename,* int *w,* int *h,* void * *savebuf* )

Save compressed YUV411 image data to file.

Requires one half of the disk space required to save an uncompressed PPM.

#### 4.4.1.6 void vglBaricenterVga ( VglImage * *src,* double * *x_avg,* double * *y_avg,* double * *pix_count* )

Calculates baricenter of vga image.

The resuld is stored in image RGB with one pixel. R = M(0, 0) = sum( f(x, y) ) G = M(1, 0) = sum( x * f(x, y) ) B = M(0, 1) = sum( y * f(x, y) )

Reference:

William K. Pratt, Digital Image Processing, Second Edition

References vglBaricenterInit(), vglCreateImage(), vglDownload(), vglSelfSum22(), vglSelfSum3v(), vglSelfSum4h(), vglSelfSum5h(), and vglSelfSum5v().

#### 4.4.1.7 void vglCErodeCross3 ( VglImage * *src,* VglImage * *mask,* VglImage * *dst,* VglImage * *buf,* int *times* )

Morpological conditional erosion by cross structuring element 3x3. A buffer is required. Source and destination may be the same.

The structuring element is a 3x3 cross. The parameter "times" indicates how many times the erosion will be applied.

References vglErodeCross3(), and vglOr().

#### 4.4.1.8 void vglClear ( VglImage * *image,* float *r,* float *g,* float *b,* float *a* )

Referenced by vglDistTransform5(), vglDistTransformCross3(), and vglDistTransformSq3().

#### 4.4.1.9 void vglCloseSq3 ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* int *times* )

Morphological closing by square structuring element. A buffer is required. Source and destination may be the same.

The structuring element is a 3x3 square. The parameter "times" indicates how many times the closing will be applied.

References vglDilateSq3(), and vglErodeSq3().

#### 4.4.1.10 VglImage* vglCopyCreateImage ( VglImage * *img_in* )

Create image with same format and data as img_in.

References vglCopy(), and vglCreateImage().

#### 4.4.1.11 VglImage* vglCopyCreateImage ( IplImage * *img_in,* int *dim3,* int *has_mipmap* )

Create image with same format and data as img_in

References vglCreateImage(), and vglUpload().

#### 4.4.1.12 void vglCopyImageTex ( VglImage * *src,* VglImage * *dst* )

Copy data from src texture to dst texture

#### 4.4.1.13 void vglCopyImageTexFS ( VglImage * *src,* VglImage * *dst* )

Copy data from src texture to dst texture using a fragment shader

#### 4.4.1.14 void vglCopyImageTexVFS ( VglImage * *src,* VglImage * *dst* )

Copy data from src texture to dst texture using a fragment shader and a vertex shader

#### 4.4.1.15 VglImage* vglCreateImage ( VglImage * *img_in* )

Create image with same format as img_in

Referenced by vglBaricenterVga(), vglCopyCreateImage(), vglCreateImage(), and vglLoadPGM().

#### 4.4.1.16 VglImage* vglCreateImage ( IplImage * *img_in,* int *dim3,* int *has_mipmap* )

Create image with same format as img_in

References vglCreateImage().

#### 4.4.1.17 VglImage* vglCreateImage ( CvSize *size,* int *depth,* int *nChannels,* int *dim3,* int *has_mipmap* )

Create image as described by the parameters

References vglUpload().

#### 4.4.1.18 void vglDistTransform5 ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* VglImage * *buf2,* int *times* )

Distance transform given by alternating an elementary cross and a square 3x3.

Perform successive erorions on input image thresholded to 1/256. The sum of the erosions results is returned as the distance transform result.

References vglClear(), vglCopy(), vglErodeCross3(), vglErodeSq3(), vglSum(), and vglThresh().

#### 4.4.1.19 void vglDistTransformCross3 ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* VglImage * *buf2,* int *times* )

Distance transform given by elementary cross.

Perform successive erorions on input image thresholded to 1/256. The sum of the erosions results is returned as the distance transform result.

References vglClear(), vglErodeCross3(), vglSum(), and vglThresh().

#### 4.4.1.20 void vglDistTransformSq3 ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* VglImage * *buf2,* int *times* )

Distance transform given by square 3x3.

Perform successive erorions on input image thresholded to 1/256. The sum of the erosions results is returned as the distance transform result.

References vglClear(), vglErodeSq3(), vglSum(), and vglThresh().

#### 4.4.1.21 void vglDownload ( VglImage * *image* )

Force transfer of image from GPU to RAM. Add RAM as valid context.

Transfer done by glGetTexImage. Color order is compatible with iplImage, that is, BGR.

Time to transfer a VGA image = 2.5ms

References vglPrintImageInfo().

Referenced by vglBaricenterVga().

#### 4.4.1.22 void vglDownloadFaster ( VglImage * *image* )

Force transfer of image from GPU to RAM. Add RAM as valid context.

Transfer done by glReadPixels. Color order is compatible with iplImage, that is, BGR.

Time to transfer a VGA image = 1.0 to 1.5ms

References vglPrintImageInfo().

#### 4.4.1.23 void vglDownloadFBO ( VglImage * *image* )

Force transfer of image from FBO to RAM. Add RAM as valid context.

Transfer done by glReadPixels. Color order is compatible with iplImage, that is, BGR.

References vglPrintImageInfo().

#### 4.4.1.24 void vglDownloadPGM ( VglImage * *image* )

Transfer image from GPU to RAM in format suitable for saving as PGM.

Use it imediately before vglSavePGM. It is different from vglDownload in two points. The unpack alignment is 1 and color is grayscale

Time to transfer a VGA image = 10ms

Referenced by vglSavePGM().

#### 4.4.1.25 void vglDownloadPPM ( VglImage * *image* )

Transfer image from GPU to RAM in format suitable for saving as PPM.

Use it imediately before vglSavePPM. It is different from vglDownload in two points. The unpack alignment is 1 and color order is RGB

Time to transfer a VGA image = 3ms

Referenced by vglSavePPM().

#### 4.4.1.26 void vglErodeSq3Sep ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* int *times* )

Morpological erosion by square structuring element 3x3. A buffer is required. Source and destination may be the same.

The structuring element is a 3x3 square. The parameter "times" indicates how many times the erosion will be applied.

References vglErodeHL3(), and vglErodeVL3().

#### 4.4.1.27 void vglErodeSq5Sep ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* int *times* )

Morpological erosion by square structuring element 5x5. A buffer is required. Source and destination may be the same.

The structuring element is a 5x5 square. The parameter "times" indicates how many times the erosion will be applied.

References vglErodeHL5(), and vglErodeVL5().

#### 4.4.1.28 void vglGetLevelDistTransform5 ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* VglImage * *buf2,* int *times* )

Get level curve of distance transform5

Perform successive erorions on input image thresholded to 1/256. The returned image is the difference between the results obtained in the iterations "times" and "times" - 1.

References vglAbsDiff(), vglCopy(), vglErodeCross3(), and vglErodeSq3().

#### 4.4.1.29 void vglHorizontalFlip2 ( VglImage * *src,* VglImage * *dst* )

Flip image horizontally, that is, left becomes right.

Image flip done by texture mapping, that is, by the fixed pipeline.

#### 4.4.1.30 int vglInit ( )

Initialize GLUT and create output window with default size (1280, 960).

Referenced by vglUpload().

#### 4.4.1.31 int vglInit ( int *w,* int *h* )

Initialize GLUT and create output window with size (w, h).

#### 4.4.1.32 void vglInOut_model ( VglImage * *dst,* VglImage * *dst1* )

Test and model for IN_OUT semantics.

First parameter is input and output. Second parameter is output.

#### 4.4.1.33 VglImage* vglLoadImage ( char * *filename,* int *iscolor,* int *has_mipmap* )

Load image from file to new VglImage.

This function uses cvLoadImage to read image file.

References vglUpload().

#### 4.4.1.34 VglImage* vglLoadPGM ( char * *filename* )

Load image from PGM file, 1 channel, unsigned byte

References LoadPGM(), and vglCreateImage().

#### 4.4.1.35 void vglMultiInput_model ( VglImage * *src0,* VglImage * *src1,* VglImage * *dst* )

Test and model for functions with multiple input images.

First and second parameters are input. Third parameter is output.

#### 4.4.1.36 void vglMultiOutput_model ( VglImage * *src,* VglImage * *dst,* VglImage * *dst1* )

Test and model for functions with multiple output images.

First parameter is input. Second and third parameters are output.

#### 4.4.1.37 void vglOpenSq3 ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* int *times* )

Morpological opening by square structuring element. Opening is an erosion followed by a dilation. A buffer is required. Source and destination may be the same.

The structuring element is a 3x3 square. The parameter "times" indicates how many times the erosion will be applied.

References vglDilateSq3(), and vglErodeSq3().

#### 4.4.1.38 void vglPrintImageInfo ( VglImage * *image* )

Print information about image.

Print width, height, depth, number of channels, OpenGL texture handler, OpenGL FBO handler, and current valid context (RAM, GPU or FBO).

Referenced by vglDownload(), vglDownloadFaster(), vglDownloadFBO(), and vglUpload().

#### 4.4.1.39 void vglReleaseImage ( VglImage ** *p_image* )

Release memory occupied by image in RAM and GPU

#### 4.4.1.40 void vglReplaceIpl ( VglImage * *image,* IplImage * *new_ipl* )

Replace IplImage, stored inside a VglImage, with new IplImage.

Both new and old images must have exactly the same properties, dimensions, depth, type etc. Is useful when grabbing frames from a camera.

References vglUpload().

#### 4.4.1.41 int vglSavePGM ( VglImage * *img,* char * *filename* )

Save image to PGM file, 1 channel, unsigned byte

References SavePGM(), and vglDownloadPGM().

#### 4.4.1.42 int vglSavePPM ( VglImage * *img,* char * *filename* )

Save image to PPM file, 3 channels, unsigned byte

References SavePPM(), and vglDownloadPPM().

#### 4.4.1.43 void vglThinBernard ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* int *times* )

Structuring element thinning. Algorithm by Bernard and Manzanera 1999.

Receive as input the image to be thinned. The second image is an auxiliary image. The third image stores the result. Both the second and third images must have the same size and type as the first input image.

The fourth parameter is the number of iterations.

Reference:

M. Couprie, Note on fifteen 2D parallel thinning algorithms, 2006

T. M. Bernard and A. Manzanera, Improved low complexity fully parallel thinning algorithms, 1999

References vglCopy(), vglErodeCross3(), and vglThinBernardAux().

#### 4.4.1.44 void vglThinChin ( VglImage * *src,* VglImage * *dst,* VglImage * *buf,* int *times* )

Structuring element thinning. Algorithm by Chin, Wan Stover and Iverson, 1987.

Receive as input the image to be thinned, buffer image and number of times to iterate.

Neighborhood pixels are indexed as follows:

$$\begin{matrix} P3 & P2 & P1 \\ P4 & P8 & P0 \\ P5 & P6 & P7 \end{matrix}$$

Reference:

M. Couprie, Note on fifteen 2D parallel thinning algorithms, 2006

R. T. Chin et al., A one-pass thinning algorithm and its parallel implementation, 1987

References vglCopy(), and vglThinChinAux().

#### 4.4.1.45 void vglUpload ( VglImage * *image,* int *swapRGB* )

Send image data from RAM to GPU. Add GPU as valid context.

If swapRGB is true, channels R and B are swapped.

References vglInit(), and vglPrintImageInfo().

Referenced by vglCopyCreateImage(), vglCreateImage(), vglLoadImage(), and vglReplaceIpl().

#### 4.4.1.46 void vglVerticalFlip2 ( VglImage * *src,* VglImage * *dst* )

Flip image vertically, that is, top becomes bottom.

Image flip done by texture mapping, that is, by the fixed pipeline.

# Index