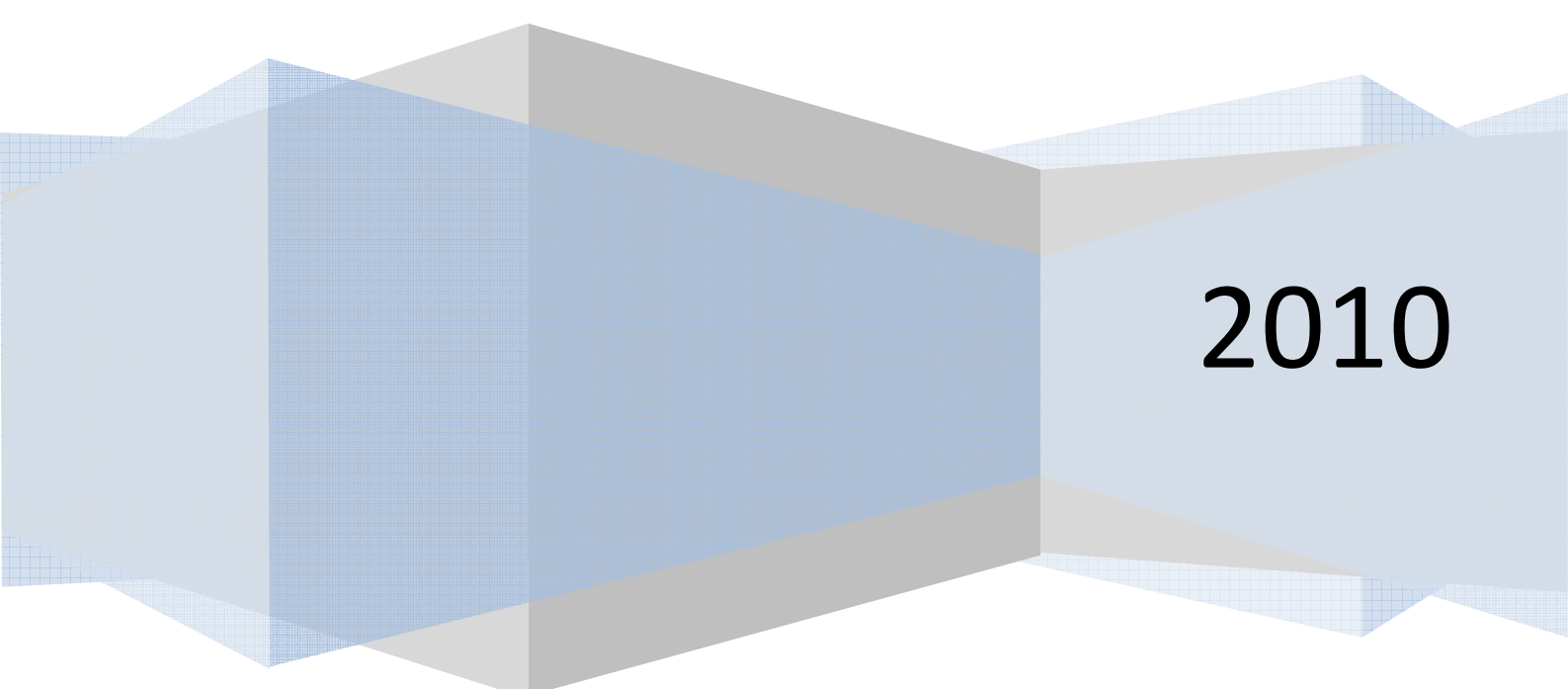


# GPU-Surf with OGRE3D

Open-source implementation of Surf on GPU

Henri ASTRE



2010

# Table of contents:

---

- Introduction
  - Definition of SURF from Wikipedia
  - Existing implementations
  - Papers
- Feature detector
  - Presentation
  - RGB → Gray + down-sampling
  - Gaussian Filtering (2-pass)
  - Determinant Hessian
  - Non-Maximum-Suppression
  - Feature extraction (Cuda)
  - Feature interpolation
- Feature descriptor
- Feature matching
- Conclusion

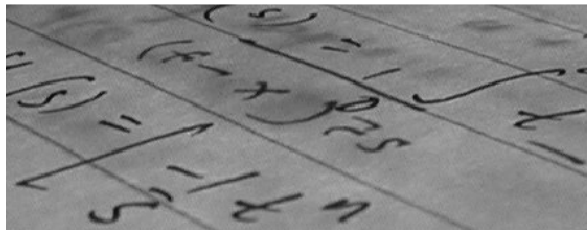
# Introduction

---

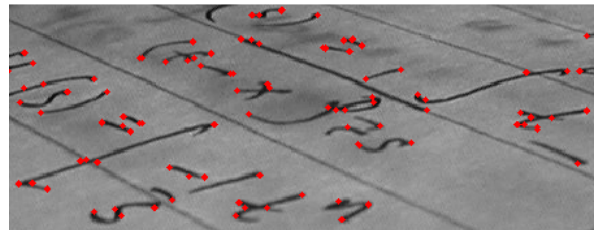
This implementation of GPU-Surf is **open-source** (license **MIT**) and should be cross-platform (currently **tested on Windows only**). It is based on Ogre3D to get DirectX/OpenGL abstraction. This implementation is using pixel shaders and Cuda for GPU computing. This is a partial implementation of Surf algorithm (feature detector implemented but feature descriptor missing).

## Definition of SURF from Wikipedia:

**SURF (Speeded Up Robust Features)** is a robust image detector & descriptor, first presented by Herbert Bay et al. in 2006, that can be used in computer vision tasks like object recognition or 3D reconstruction. It is partly inspired by the SIFT descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT. SURF is based on sums of approximated 2D Haar wavelet responses and makes an efficient use of integral images. As basic image features it uses a Haar wavelet approximation of the determinant of Hessian blob detector.



Original image



Corner detection

## Existing implementations:

Num	Name	Open-Source	GPU	Windows	Linux	Language
0	Surf	No	No	Yes	Yes	C++
1	OpenSurf	Yes (GPL v3)	Partially	Yes	Yes	C++
2	GPU-Surf	No	Yes	No	Yes	GLSL, Cuda
3	GPU-Surf	Yes (BSD)	Yes	Yes	Yes	C++, Cuda

- [0] : Surf : <http://www.vision.ee.ethz.ch/~surf>
- [1] : OpenSurf : <http://code.google.com/p/opensurf1/>
- [1] : OpenSurf Partial GPU : [http://cs264.org/projects/web/Cowgill\\_Michael/cowgill/Index.html](http://cs264.org/projects/web/Cowgill_Michael/cowgill/Index.html)
- [2] : GPU-Surf : <http://homes.esat.kuleuven.be/~ncorneli/gpusurf/>
- [3] : GPU-Surf : <http://asrl.utias.utoronto.ca/code/gpusurf/>

## Papers:

- a) H. Bay, A. Ess, T. Tuytelaars, L. Van Goo, *SURF : Speeded-Up Robust Features*
  - [ftp://ftp.vision.ee.ethz.ch/publications/articles/eth\\_biwi\\_00517.pdf](ftp://ftp.vision.ee.ethz.ch/publications/articles/eth_biwi_00517.pdf)
  - Correspond to Surf [0]
- b) N. Cornelis, L. Van Gool: *Fast Scale Invariant Feature Detection and Matching on Programmable Graphics Hardware*
  - [http://homes.esat.kuleuven.be/~ncorneli/gpusurf/ncorneli\\_cvpr2008.pdf](http://homes.esat.kuleuven.be/~ncorneli/gpusurf/ncorneli_cvpr2008.pdf)
  - Correspond to GPU-Surf [2]
- c) Paul Furgale, Chi Hay Tong and Gaetan Kenway, *Speeded-Up Speeded-Up Robust Features*
  - [http://asrl.utias.utoronto.ca/~ptf/docs/gpusurf\\_report09.pdf](http://asrl.utias.utoronto.ca/~ptf/docs/gpusurf_report09.pdf)
  - Correspond to GPU-Surf [3]

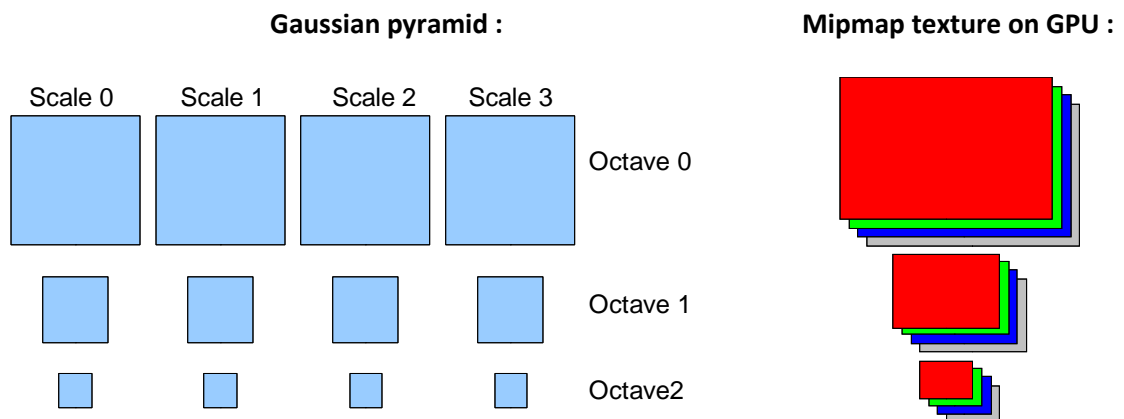
This implementation is based on the paper of Nico Cornelis [b] and the Non-Maximum-Suppression stage was done thanks to a very in deep analysis of his closed-source GPU implementation.

# Feature detector

## Presentation

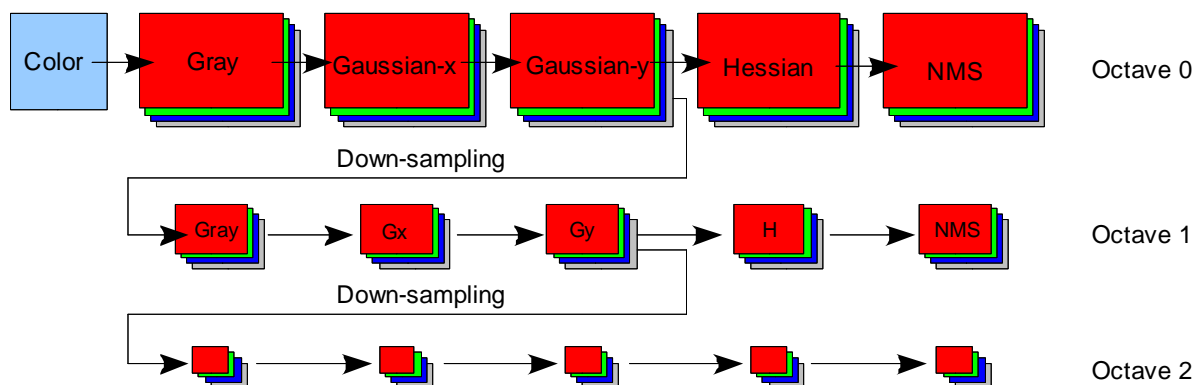
In order to detect features position you have to do some computation on multiple scales, this is usually done using a single image and applying 2D box filter of increasing size. To speed-up box filter normalization, an integral image is computed and used to compute sum of pixels of a specific area.

All CPU versions are based on this idea but this is not the case of the GPU version of Nico Cornelis. Indeed the main idea of this paper was to reduce image dimension and to apply a box filter of constant size. All down-sampled version of the original imager are stored in a gaussian pyramid. The brilliant idea of Nico Cornelis was to use texture mipmap to store this Gaussian pyramid.



As you can see a 2D texture with mipmap can be used to store a Gaussian pyramid with 4 scales (RGBA), each octave corresponding to a mipmap level.

Surf feature detector can be decomposed in multiple stages:



1. RGB→Gray + down-sampling
2. Gaussian Filtering (2-pass)
3. Determinant Hessian
4. Non-Maximum-Suppression
5. Feature extraction (Cuda)
6. Feature interpolation

## RGB → Gray + down-sampling

RGB → Gray:

```
void GPGPU_rgb2gray_fp(
    float4 uv          : TEXCOORD0,
    uniform sampler2D tex0 : register(s0),
    out float4 result   : COLOR
)
{
    float luminance = dot(tex2D(tex0, uv), float4(0.299, 0.587, 0.114, 0));
    result = float4(luminance, luminance, luminance, 1.0);
}
```

Downsampling :

```
void GPGPU_downsampling_fp(
    float4 uv          : TEXCOORD0,
    uniform sampler2D tex0 : register(s0),
    uniform float octave,
    out float4 result   : COLOR
)
{
    float4 coord = float4(uv.x, uv.y, 0, octave-1.0);
    float luminance = tex2Dlod(tex0, coord).a;
    result = float4(luminance, luminance, luminance, 1.0);
}
```

The down-sampling of an octave  $O$  is computed by reading values from the octave  $O-1$ . As explained in the paper the values are read from the alpha channel of the previous octave to reduce aliasing.

## Gaussian Filtering (2-pass)

The separable 19x19 Gaussian filters are implemented as a two-pass operation (Gaussian-x and Gaussian-y). Usually, scales of an octave are computed from each other (serial computation  $n = f(n-1)$ ). But we can compute the four scales in one step by using a Gaussian kernel with increasing radius.

```
K = 2^(1/nbScale) //nbScale = 4 (RGBA)

Original image: level = sigma
-----
Octave 0, scale 0: level = (K)*sigma
Octave 0, scale 1: level = (K^2)*sigma
Octave 0, scale 2: level = (K^3)*sigma
Octave 0, scale 3: level = (K^4)*sigma = 2.0*sigma
-----
Octave 1, niveau 0 : échelle = 2.0*(K)*sigma
```

```

Octave 1, niveau 1 : échelle = 2.0*(K^2)*sigma
Octave 1, niveau 2 : échelle = 2.0*(K^3)*sigma
Octave 1, niveau 3 : échelle = 2.0*(K^4)*sigma = 4.0*sigma
-----
...

```

Gaussian kernel used:

Scale 0	Scale 1	Scale 2	Scale 3
0	0	0	3.1577E-07
0	0	0	5.3688E-06
0	0	4.4724E-07	6.5405E-05
0	0	1.5648E-05	0.00057093
0	1.4867E-06	0.00031683	0.00357099
0	0.00013383	0.00371264	0.01600408
1.1855E-05	0.00443185	0.02517766	0.05139345
0.00495558	0.05399097	0.09881531	0.11825507
0.18527372	0.24197073	0.22444478	0.19496965
0.61951762	0.39894229	0.29503345	0.23032942
0.18527372	0.24197073	0.22444478	0.19496965
0.00495558	0.05399097	0.09881531	0.11825507
1.1855E-05	0.00443185	0.02517766	0.05139345
0	0.00013383	0.00371264	0.01600408
0	1.4867E-06	0.00031683	0.00357099
0	0	1.5648E-05	0.00057093
0	0	4.4724E-07	6.5405E-05
0	0	0	5.3688E-06
0	0	0	3.1577E-07

## Determinant Hessian

The determinant of Hessian is computed using:

$$detH(x, y) = \begin{vmatrix} \frac{\partial^2 G(x, y)}{\partial x^2} & \frac{\partial^2 G(x, y)}{\partial x \partial y} \\ \frac{\partial^2 G(x, y)}{\partial x \partial y} & \frac{\partial^2 G(x, y)}{\partial y^2} \end{vmatrix}$$

Second-order derivative:

$$L_{xx} = \frac{1}{4} [L(x+2, y) - 2L(x, y) + L(x-2, y)]$$

$$L_{yy} = \frac{1}{4} [L(x, y+2) - 2L(x, y) + L(x, y-2)]$$

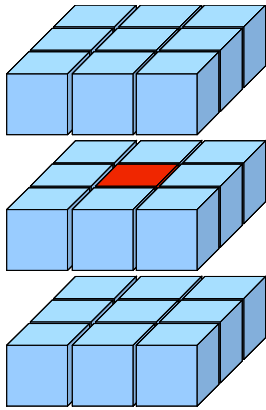
$$L_{xy} = \frac{1}{4} [L(x-1, y-1) - L(x-1, y+1) - L(x+1, y-1) + L(x+1, y+1)]$$

$$detH(x, y) = L_{xx} \cdot L_{yy} - L_{xy} \cdot L_{xy}$$

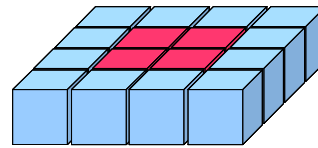
## Non-Maximum-Suppression

In this stage we've got Hessian values for each pixel in each octaves and scales. But only the maximum values are relevant, so we need to compute NMS (*Non-Maximum-Suppression*) to remove all small values: features are located in the maximum of Hessian values.

We need to perform a 3x3x3 NMS filtering in order to isolate the feature. It consists in keeping a value if it's the maximum value in a 26 neighboring.

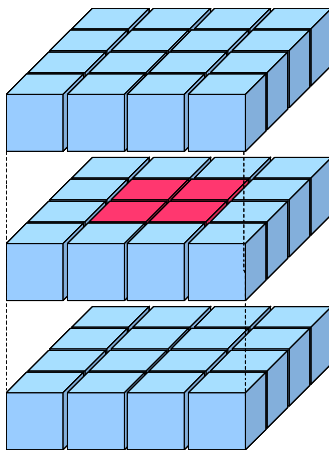


3x3x3 Non Maximum Suppression  
(26 neighboring)



There is only one possible maximum in 2x2  
image-space (red zone).

To reduce memory occupancy, you can store the result of the non maximum suppression in a texture mipmap of half dimensions in x and y. Because it's not possible to have two maximum in 2x2 image space.



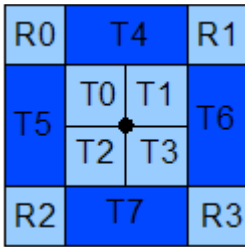
As a result, a 4x4x3 (width, height, space) image zone can give 1 possible value. But you need to add extra information to localize the maximum in the original red zone (boolean x and y offset).



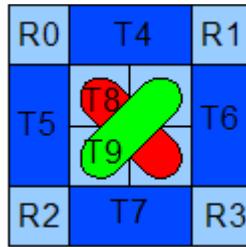
Easy case: first octave only

Maximum extraction in image-space

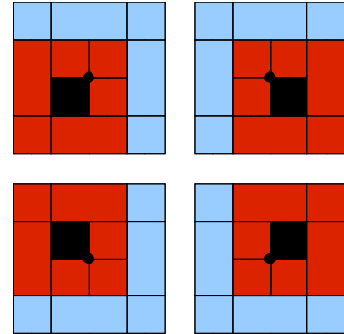
There is only one possible maximum possible in 2x2 image space. But each pixel in this 2x2 image-space zone need extra border pixel to compute 3x3 comparisons. So you need to use 4x4 image-space zones to compute a 3x3 comparison for pixels in centered 2x2 zones.



4x4 image-space with 2x2 centered zone (T0,T1,T2,T3)



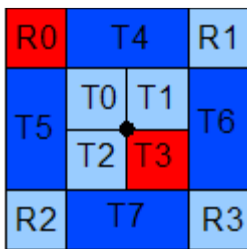
Extra maximum comparison T8 and T9



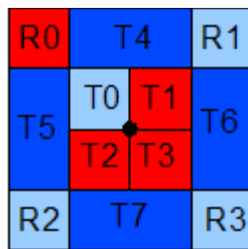
3x3 comparison for each pixel in 2x2 image-space

Example:

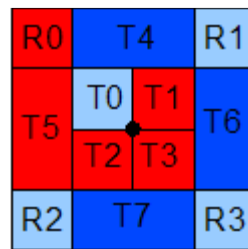
How to extract maximum for T0 ?



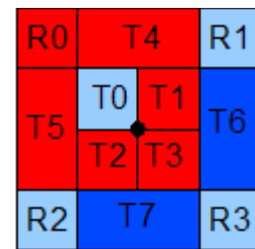
$$R0 = \max(R0, T3)$$



$$R0 = \max(R0, T9)$$

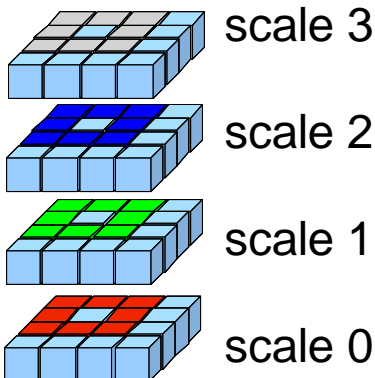


$$R0 = \max(R0, T5)$$



$$R0 = \max(R0, T4)$$

The maximum comparison are performs on vectors rather than scalars, which leads to significant performance enhancement. So we get maximum values in image-space for each scale simultaneously.



R0 is now a vector4(x,y,z,w) or (r,g,b,a)  
 $R0.x = \max(\text{red zone})$   
 $R0.y = \max(\text{green zone})$   
 $R0.z = \max(\text{blue zone})$   
 $R0.w = \max(\text{grey zone})$

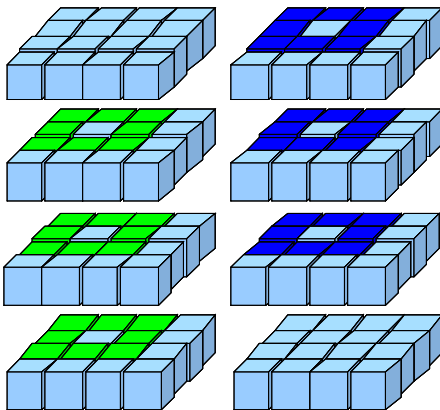
Maximum extraction in scale-space

R0 is the vector4 (x,y,z,w) you get from previous stage.

- a)  $R0.xyz = \max(R0.xyz, R0.yzw)$
- b)  $R0.yzw = \max(R0.xyz, R0.yzw)$

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \xrightarrow{a} \begin{pmatrix} \max(x, y) \\ \max(y, z) \\ \max(z, w) \\ w \end{pmatrix} \xrightarrow{b} \begin{pmatrix} \max(x, y) \\ \max(\max(x, y), \max(y, z)) \\ \max(\max(y, z), \max(z, w)) \\ \max(\max(z, w), w) \end{pmatrix} \rightarrow \begin{pmatrix} ? \\ \max(x, y, z) \\ \max(y, z, w) \\ ? \end{pmatrix}$$

(? means irrelevant value)

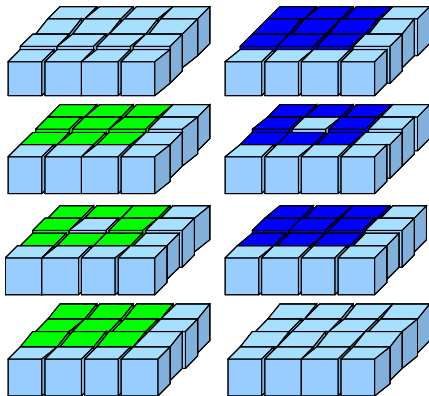


R0.y = max(blue zone)  
R0.z = max(green zone)

R0 is a vector4 (x,y,z,w) and T0 is a vector4 (i,j,k,l)

- a)  $R0.xyz = \max(R0.xyz, T0.yzw)$
- b)  $R0.yzw = \max(T0.xyz, R0.yzw)$

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \xrightarrow{a} \begin{pmatrix} \max(x, j) \\ \max(y, k) \\ \max(z, l) \\ w \end{pmatrix} \xrightarrow{b} \begin{pmatrix} \max(x, j) \\ \max(i, \max(y, k)) \\ \max(j, \max(z, l)) \\ \max(k, w) \end{pmatrix} \rightarrow \begin{pmatrix} ? \\ \max(y, i, k) \\ \max(z, j, l) \\ ? \end{pmatrix}$$



R0.y = max(blue zone)  
R0.z = max(green zone)

*Maximum thresholding*

You need to threshold maximum value in order to keep only relevant ones.

R0 is the vector4 (x,y,z,w) you get from previous stage.

- a)  $R0 = \max(R0, \text{threshold})$
- b)  $T4 = T0 \geq R0$

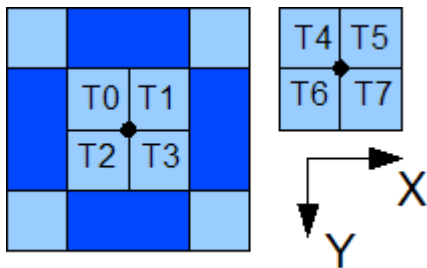
Explanation a):

- $R0.x = \text{irrelevant value}$
- $R0.y = \max(\max(\text{green zone}), \text{threshold})$
- $R0.z = \max(\max(\text{blue zone}), \text{threshold})$
- $R0.w = \text{irrelevant value}$

Explanation b):

- $T4.x = \text{irrelevant value}$
- $T4.y = \text{is } T0.y \text{ a } 3 \times 3 \times 3 \text{ maximum } > \text{threshold} ?$
- $T4.z = \text{is } T0.z \text{ a } 3 \times 3 \times 3 \text{ maximum } > \text{threshold} ?$
- $T4.w = \text{irrelevant value}$

*Maximum encoding*



- T4 = is T0 maximum ?
- T5 = is T1 maximum ?
- T6 = is T2 maximum ?
- T7 = is T3 maximum ?

	x-offset	y-offset	extremum				
<table border="1"><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	1	0	0	0	0	0	1
1	0						
0	0						
<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td></tr></table>	0	1	0	0	1	0	1
0	1						
0	0						
<table border="1"><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	0	0	0	1	1	1	1
0	0						
0	1						
<table border="1"><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td></tr></table>	0	0	1	0	0	1	1
0	0						
1	0						
<table border="1"><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0
0	0						
0	0						

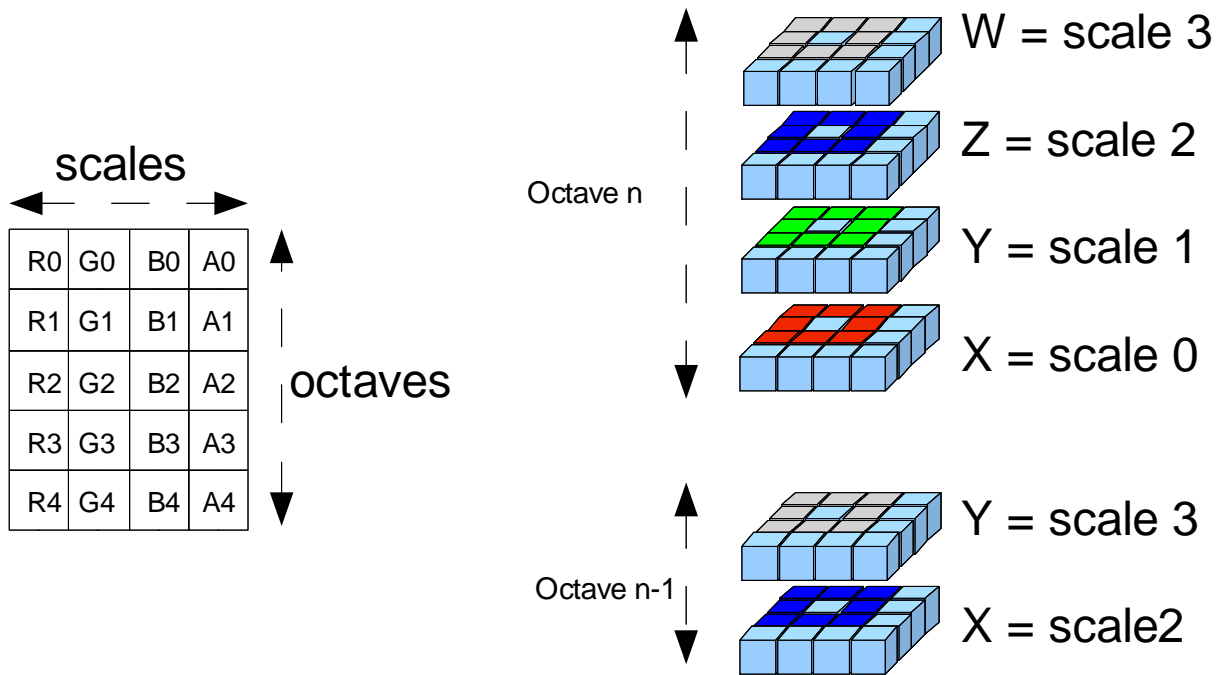
- x-offset = T5+T7
- y-offset = T6+T7
- extremum = T4+T5+T6+T7

Hard-case: remaining octaves:

How to get 4 detectable scales?

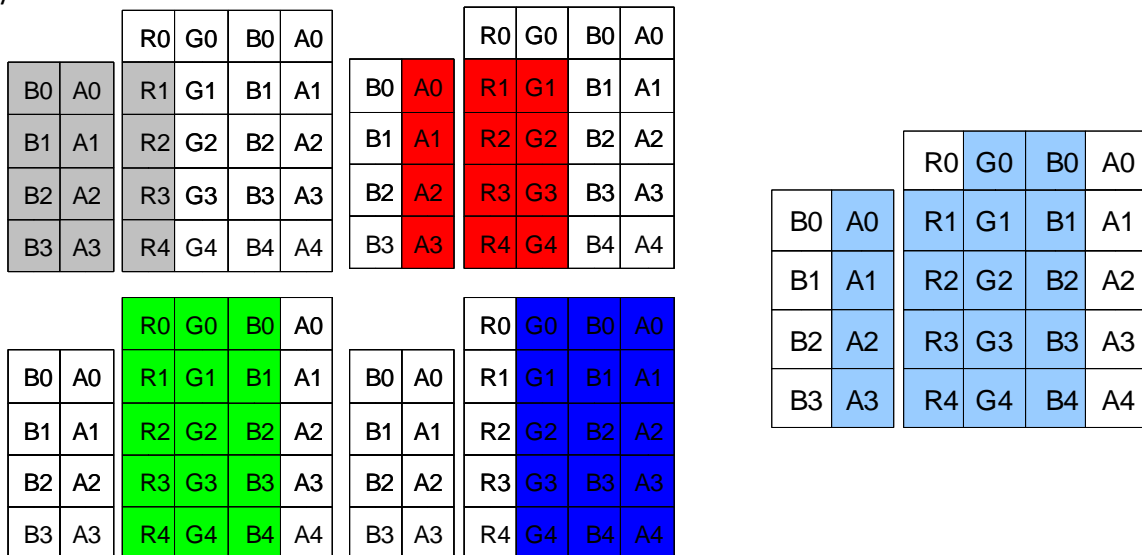
Those examples were taken for the trivial case (green and blue channel ↔ scale 1 & 2). But for the other scales it's a bit harder because there are lacking one border scale for performing maximum extraction in scale-space.

You can sub-sample last two scales of previous octave to get again 4 detectable scales.



Schematic representation of Gaussian pyramid with 5 octaves and 4 scales

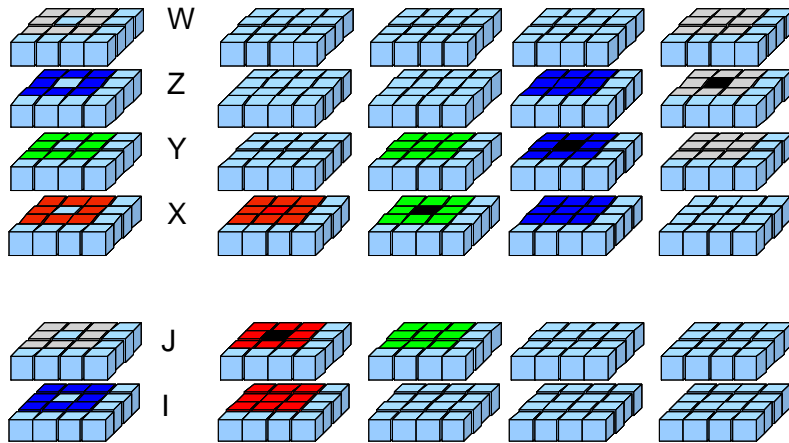
Octave N + 2 scales sub-sampled from Octave N-1



Scale 3 = alpha channel needs 2 sub-sampled channels  
Scale 0 = red channel needs 1 sub-sampled channel

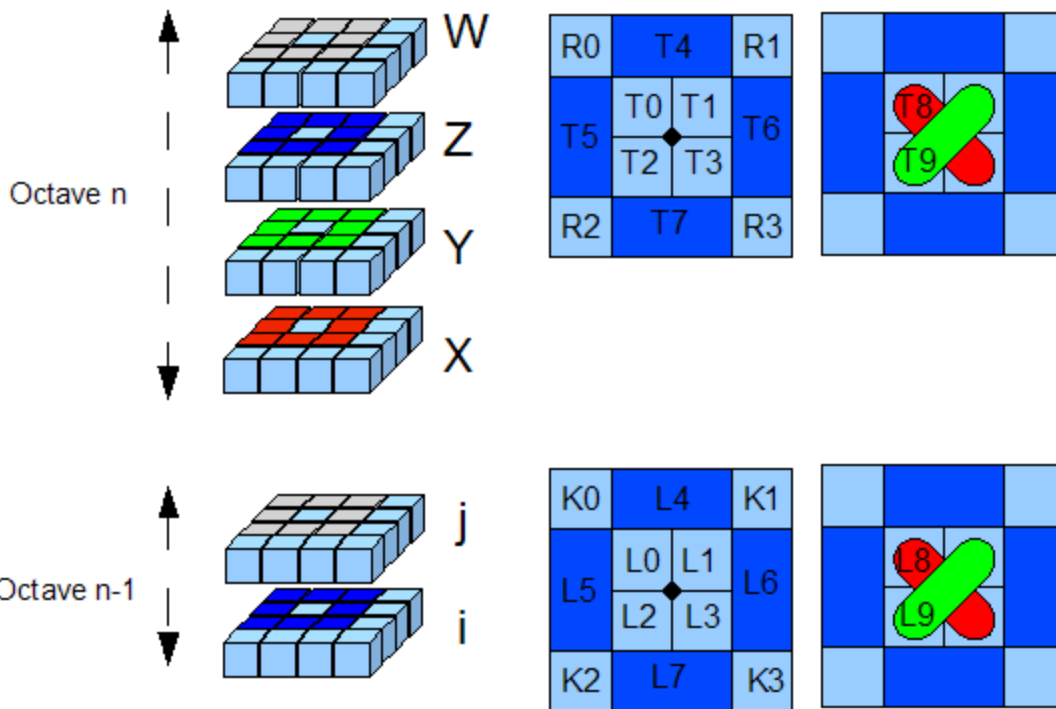
You get 4 detectable scales per octave (except for the first octave)

Maximum extraction in scale-space for the remaining octaves



As you can see to be able to perform 3x3x3 non maximum suppression on 4 scales you need 6 scales.

This is done by sub-sampling the 2 last scales of the previous octaves.



Maximum extraction in image-space

The spatial maximum extraction is not very different from the one explained for the first octave. You just have to do the comparisons on both octaves.

Octave N (R,T) ↔ Octave N-1 (K,L)

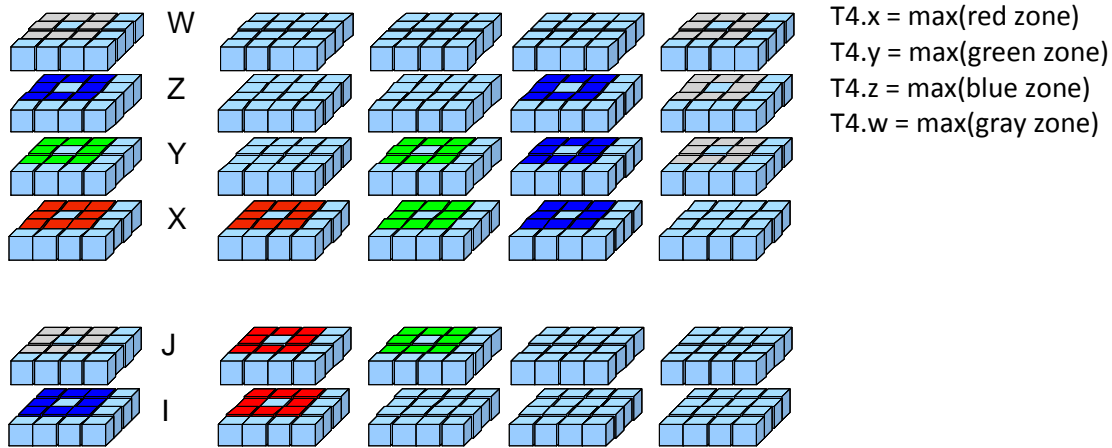
So you also have L8 and L9 equivalent of T8 and T9.

Maximum extraction in scale-space

K0 is a vector2 (i,j) and R0 is a vector4(x,y,z,w)

- a)  $T4 = \max(\text{float4}(K0.y, R0.xyz), R0)$
- b)  $T4 = \max(\text{float4}(K0.xy, R0.xy), T4)$

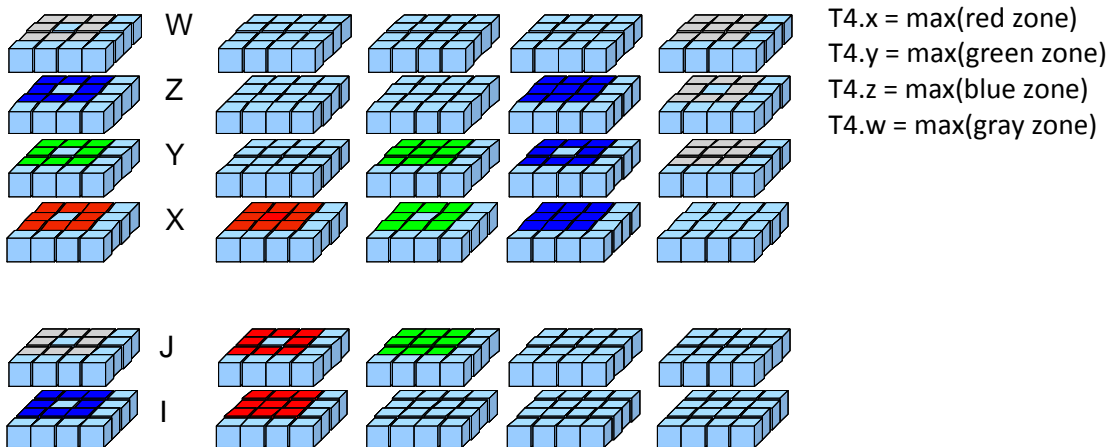
$$\begin{matrix} \begin{matrix} i \\ j \end{matrix} \\ \begin{matrix} x \\ y \\ z \\ w \end{matrix} \end{matrix} \xrightarrow{a} \begin{matrix} \max(j, x) \\ \max(x, y) \\ \max(y, z) \\ \max(z, w) \end{matrix} \xrightarrow{b} \begin{matrix} \max(i, \max(j, x)) \\ \max(j, \max(x, y)) \\ \max(x, \max(y, z)) \\ \max(y, \max(z, w)) \end{matrix} \rightarrow \begin{matrix} \max(i, j, x) \\ \max(j, x, y) \\ \max(x, y, z) \\ \max(y, z, w) \end{matrix}$$



L0 is a vector2 (i,j), T0 is a vector4(x,y,z,w) and T4 is a vector4(l,m,o,p)

- a)  $T4 = \max(T4, T0)$
- b)  $T4 = \max(T4, \text{float4}(L0.xy, T0.xy))$

$$\begin{matrix} \begin{matrix} l \\ m \\ o \\ p \end{matrix} \\ \begin{matrix} x \\ y \\ z \\ w \end{matrix} \end{matrix} \xrightarrow{a} \begin{matrix} \max(l, x) \\ \max(m, y) \\ \max(o, z) \\ \max(p, w) \end{matrix} \xrightarrow{b} \begin{matrix} \max(i, \max(l, x)) \\ \max(j, \max(m, y)) \\ \max(x, \max(o, z)) \\ \max(y, \max(p, w)) \end{matrix} \rightarrow \begin{matrix} \max(i, l, x) \\ \max(j, m, y) \\ \max(x, o, z) \\ \max(y, p, w) \end{matrix}$$



### *Maximum Thresholding*

Thresholding for the remaining octaves....

T4 is the vector4 you get from the previous stage

- a)  $T4 = \max(T4, \text{threshold})$
- b)  $T4 = \text{float4}(L0.y, T0.xyz) \geq T4$

Explanation a):

- $T4.x = \max(\text{red zone}, \text{threshold})$
- $T4.y = \max(\text{green zone}, \text{threshold})$
- $T4.z = \max(\text{blue zone}, \text{threshold})$
- $T4.w = \max(\text{gray zone}, \text{threshold})$

Explanation b):

- $T4.x = \text{is } L0.y \text{ a } 3 \times 3 \times 3 \text{ maximum } > \text{threshold} ?$
- $T4.y = \text{is } T0.x \text{ a } 3 \times 3 \times 3 \text{ maximum } > \text{threshold} ?$
- $T4.z = \text{is } T0.y \text{ a } 3 \times 3 \times 3 \text{ maximum } > \text{threshold} ?$
- $T4.w = \text{is } T0.z \text{ a } 3 \times 3 \times 3 \text{ maximum } > \text{threshold} ?$

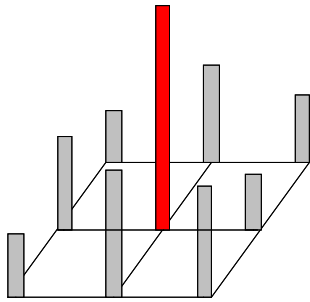
### *Maximum encoding*

No difference with the version for the first octave.

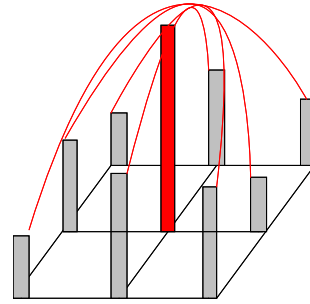
## **Feature extraction (Cuda)**

This is a pretty straight-forward implementation from the information given in the paper of Nico Cornelis.

## Feature interpolation



3x3 Hessian values (maximum center value)



Parabolic fitting maximum value

Fitting of parabol

We are looking for an X vector as  $H \cdot X = -G$

G = first-order derivative

H = second order derivative

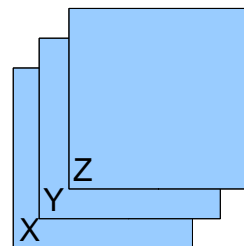
$$\begin{bmatrix} \frac{\partial^2 D(x, y, s)}{\partial x^2} & \frac{\partial^2 D(x, y, s)}{\partial x \partial y} & \frac{\partial^2 D(x, y, s)}{\partial x \partial s} \\ \frac{\partial^2 D(x, y, s)}{\partial x \partial y} & \frac{\partial^2 D(x, y, s)}{\partial y^2} & \frac{\partial^2 D(x, y, s)}{\partial y \partial s} \\ \frac{\partial^2 D(x, y, s)}{\partial x \partial s} & \frac{\partial^2 D(x, y, s)}{\partial y \partial s} & \frac{\partial^2 D(x, y, s)}{\partial s^2} \end{bmatrix} \begin{bmatrix} x \\ y \\ s \end{bmatrix} = - \begin{bmatrix} \frac{\partial D(x, y, s)}{\partial x} \\ \frac{\partial D(x, y, s)}{\partial y} \\ \frac{\partial D(x, y, s)}{\partial s} \end{bmatrix}$$

$$H \cdot X = -G \leftrightarrow \begin{bmatrix} dXX & dXY & dXS \\ dXY & dYY & dYS \\ dXS & dYS & dSS \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = - \begin{bmatrix} dX \\ dY \\ dS \end{bmatrix}$$

$$X = -H^{-1} \cdot G \leftrightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} = - \begin{bmatrix} dXX & dXY & dXS \\ dXY & dYY & dYS \\ dXS & dYS & dSS \end{bmatrix}^{-1} \cdot \begin{bmatrix} dX \\ dY \\ dS \end{bmatrix}$$

Maximum is situated in S4.y

S0	S1	S2
S3	S4	S5
S6	S7	S8





*First-order derivative:*

$$dX = \frac{1}{2} \cdot L(x+1, y, s) - \frac{1}{2} \cdot L(x-1, y, s) \leftrightarrow dX = 0.5 * S5.y - 0.5 * S3.y$$

$$dY = \frac{1}{2} \cdot L(x, y+1, s) - \frac{1}{2} \cdot L(x, y-1, s) \leftrightarrow dY = 0.5 * S7.y - 0.5 * S1.y$$

$$dS = \frac{1}{2} \cdot L(x, y, s+1) - \frac{1}{2} \cdot L(x, y, s-1) \leftrightarrow dS = 0.5 * S4.z - 0.5 * S4.x$$

*Second-order derivative:*

$$dXX = L(x+1, y, s) + L(x-1, y, s) - 2 \cdot L(x, y, s) \leftrightarrow dXX = S5.y + S3.y - 2 \cdot S4.y$$

$$dYY = L(x, y+1, s) + L(x, y-1, s) - 2 \cdot L(x, y, s) \leftrightarrow dYY = S7.y + S1.y - 2 \cdot S4.y$$

$$dSS = L(x, y, s+1) + L(x, y, s-1) - 2 \cdot L(x, y, s) \leftrightarrow dSS = S4.z + S4.x - 2 \cdot S4.y$$

$$dXY = \frac{1}{4} \cdot [L(x+1, y+1, s) - L(x-1, y+1, s) - L(x+1, y-1, s) + L(x-1, y-1, s)]$$

$$\leftrightarrow dXY = \frac{1}{4} \cdot [S8.y - S6.y - S2.y + S0.y]$$

$$dXS = \frac{1}{4} \cdot [L(x+1, y, s+1) - L(x-1, y, s+1) - L(x+1, y, s-1) + L(x-1, y, s-1)]$$

$$\leftrightarrow dXS = \frac{1}{4} \cdot [S5.z - S3.z - S5.x + S3.x]$$

$$dYS = \frac{1}{4} \cdot [L(x, y+1, s+1) - L(x, y-1, s+1) - L(x, y+1, s-1) + L(x, y-1, s-1)]$$

$$\leftrightarrow dYS = \frac{1}{4} \cdot [S7.z - S1.z - S7.x + S1.x]$$

# Feature descriptor

---

*This is not implemented in the current version.*

# Feature matching

---

*This is not implemented in the current version.*

This could be done using Cublas from Nvidia.

# Conclusion

---

The author of this implementation has started this project to learn GPGPU computing. He has created some libraries to ease GPGPU development with Ogre3D:

- **Ogre::Cuda:** Cuda 3.0 integration with Ogre3D: *allow to register/map DirectX or OpenGL texture with Cuda graphic interop.*
- **Ogre::OpenCL:** OpenCL integration with Ogre3D: *designed with the same goal as Ogre::Cuda but for OpenCL (in the future this version of GPUSurf could be using OpenCL instead of Cuda).*
- **Ogre::GPGPU:** GPGPU helper for Ogre3D: *add an abstraction over texture/pixel shader rename in result/operation.*
- **Ogre::Canvas:** 2D API for Ogre3D using Skia: *this is useful for visual debugging and is implemented using the same API as Html5 Canvas tag and also have the same Javascript bindings.*