

# Homework 1

Computer Vision 2017 Spring

2017.3.14

# OpenCV



- Highly optimized C++ implementation of many CV algorithms
- Crossplatform
- Many bindings (Python, Matlab, Java . . .)

<http://opencv.org>

# Install

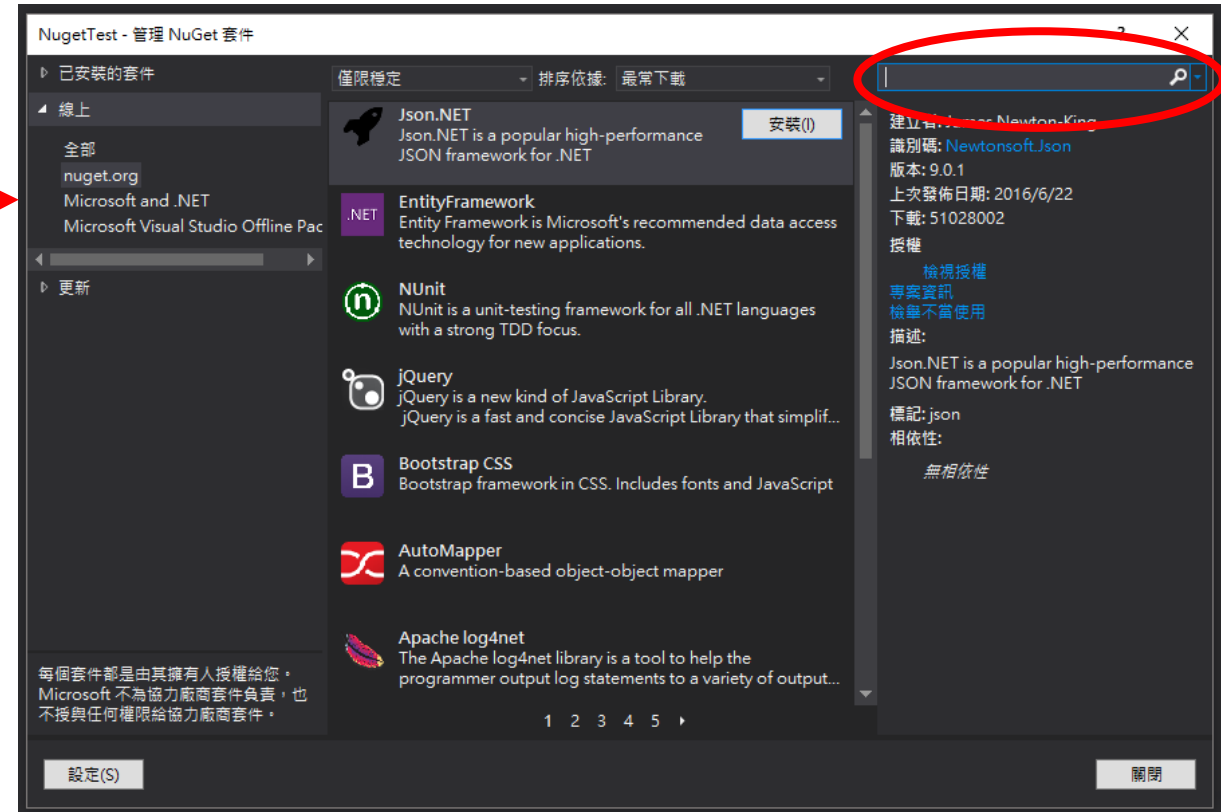
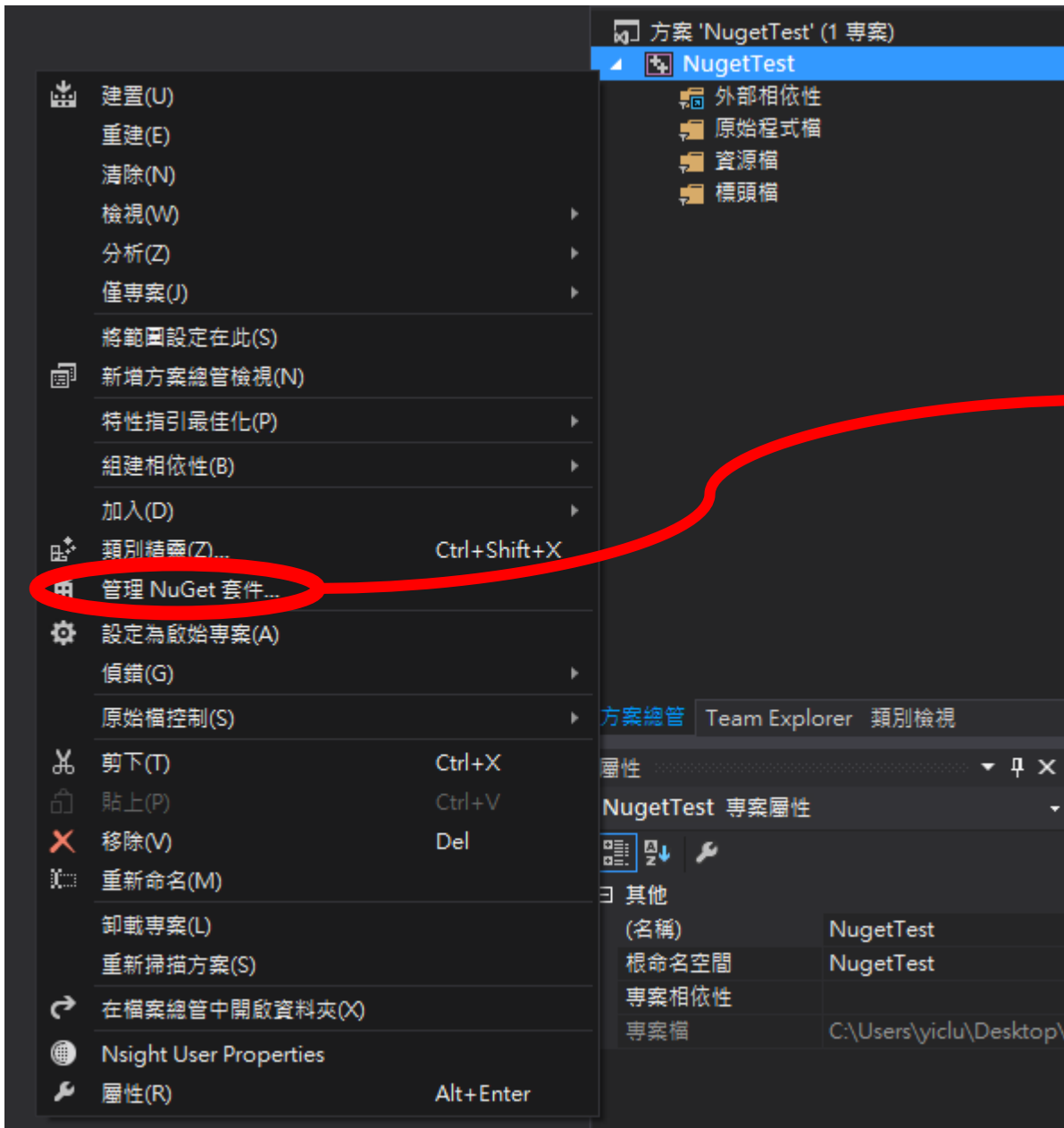
- VS 2013 、 VS 2015

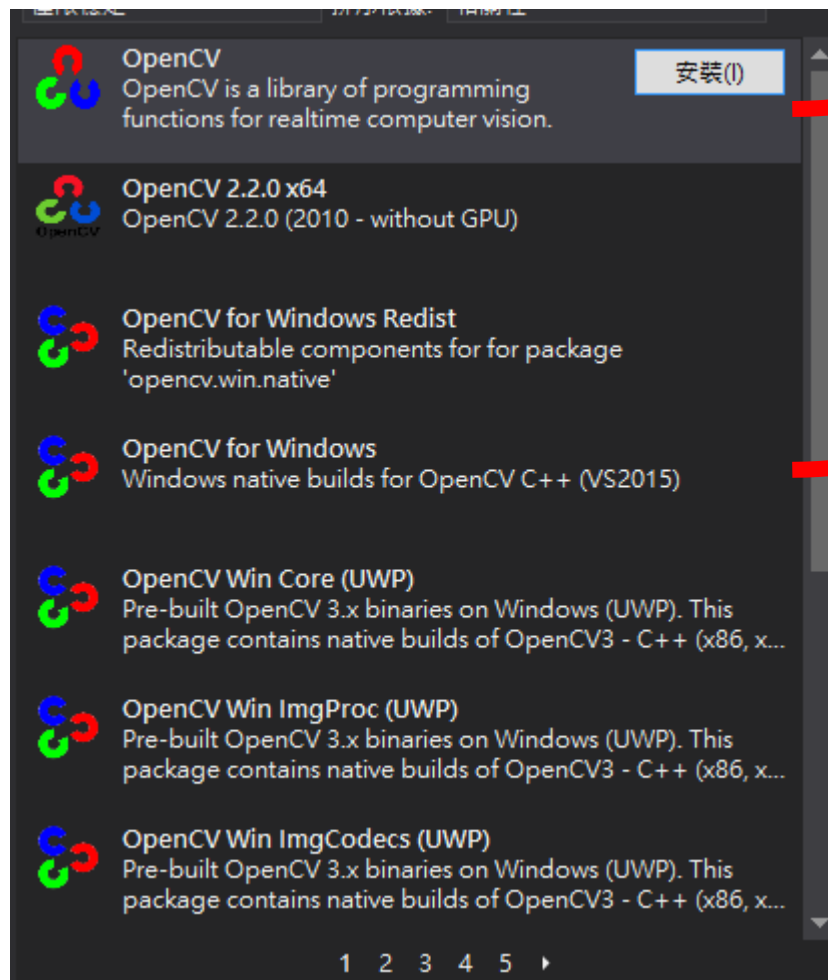
1 , Open your visual Studio

2 , New a Empty Project

3 , Install OpenCV

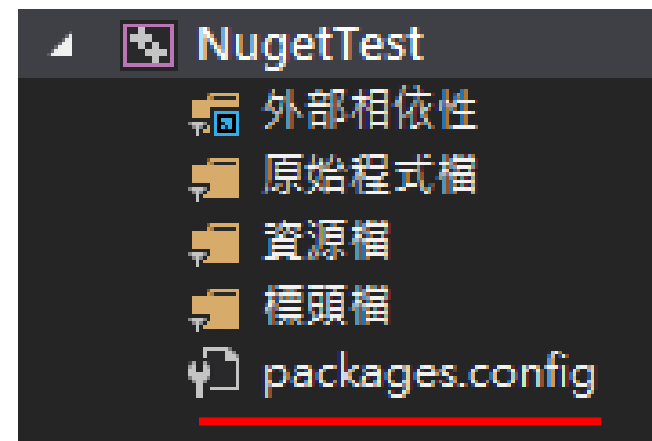
# VS 2013

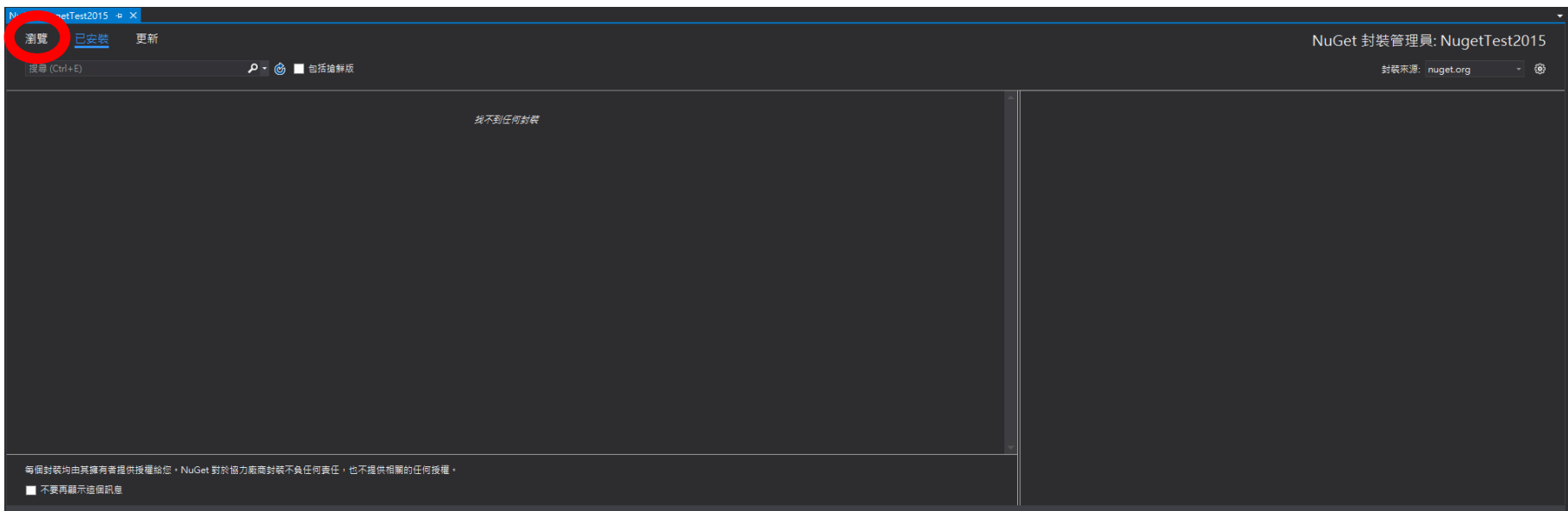




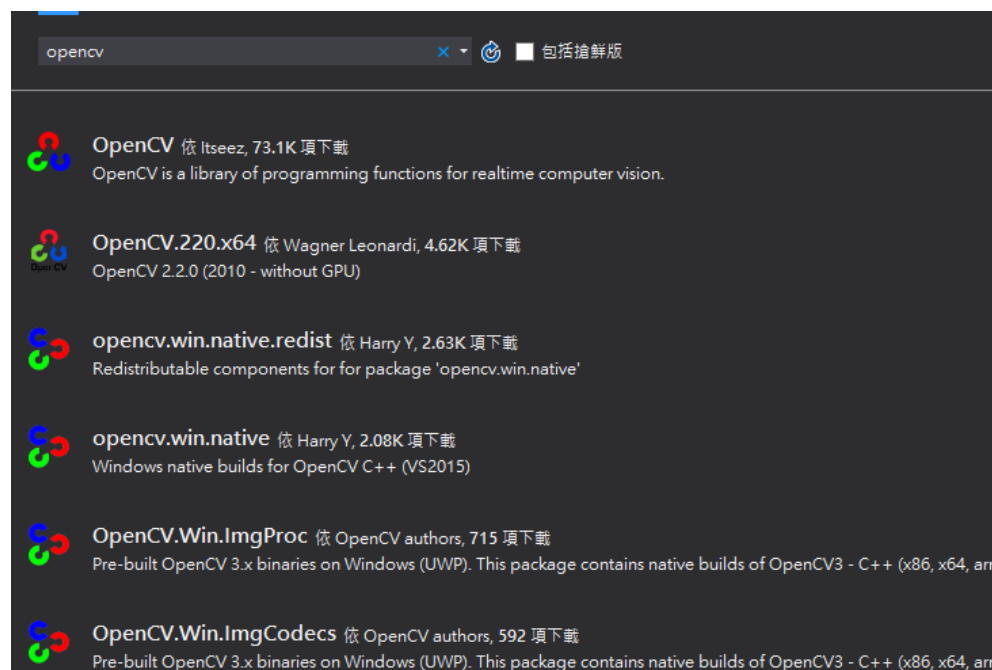
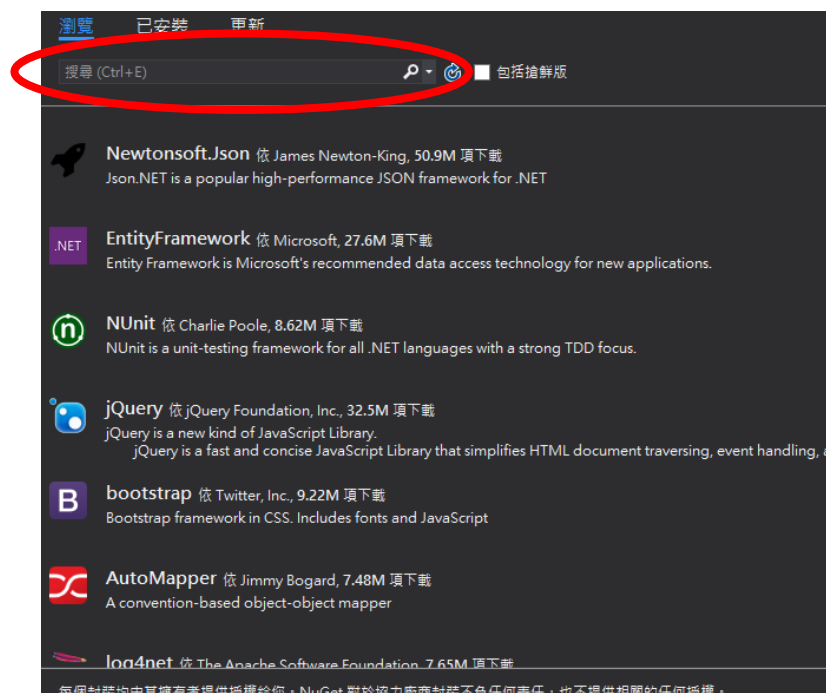
VS 2013

VS 2015





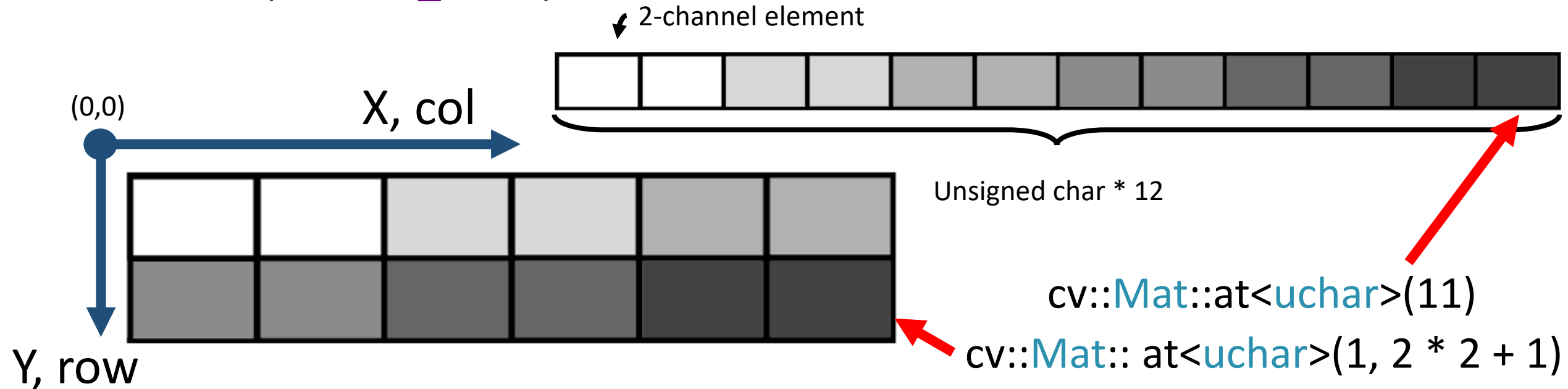
# VS 2015



# Mat

- Multi-dimensional array
- Handle the memory automatically

```
cv::Mat(2, 3, CV_8UC2);
```



# Sample

- See OpenCV documentation and samples for more information

```
#include <opencv/highgui.h>
```

Include this in VS 2013

```
#include <opencv2/highgui/highgui.hpp>
```

Include this in VS 2015

```
using namespace cv;

int main() {
    Mat Image = imread("test/bunny/pic1.bmp", IMREAD_GRAYSCALE);

    Mat tempImage = Image.clone();
    Mat smallImage(Image.rows / 2, Image.cols / 2, CV_8U);

    for (int rowIndex = 0; rowIndex < smallImage.rows; rowIndex++) {
        for (int colIndex = 0; colIndex < smallImage.cols; colIndex++) {
            smallImage.at<uchar>(rowIndex, colIndex) = tempImage.at<uchar>(rowIndex * 2, colIndex * 2);
        }
    }

    Mat result(tempImage.rows + smallImage.rows, tempImage.cols, CV_8U, Scalar(0));
    tempImage.copyTo(result(Rect(0, 0, tempImage.cols, tempImage.rows)));
    smallImage.copyTo(result(Rect(0, tempImage.rows, smallImage.cols, smallImage.rows)));

    imshow("CV", result);
    waitKey();

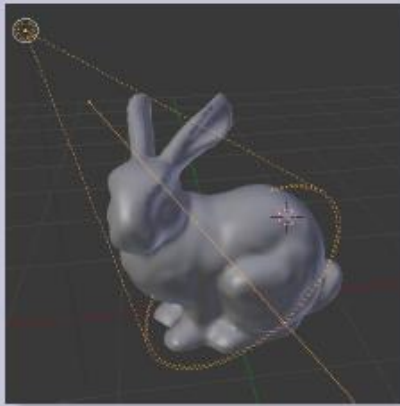
    return 0;
}
```





# Photometric Stereo

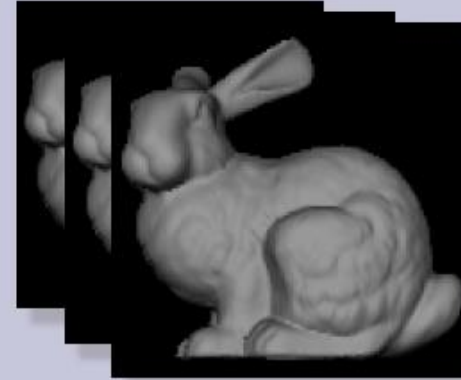
grayscale



*Lambertian Reflection*

$$i = k_d l (s^T n)$$

$$i = k_d l (s^T \mathbf{n})$$



$i_{x,y}^{(m)}$  the intensity of the  $m$ th image at pixel  $(x, y)$

$k_d$  the color of the surface

$l_m$  the intensity of the  $m$ th incoming light

$s_m$  the **unit vector** pointing from the surface to the  $m$  incoming light

$n_{x,y}$  the surface's normal vector (**unit vector**) at pixel  $(x, y)$

For brevity, we omit the dependence of  $x, y$  and  $m$ .

# Normal Estimation

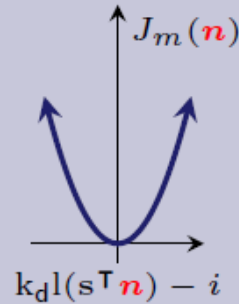
According to the reflection model, we suppose that the unknown normal vector and the intensity in the  $m$ th image at pixel  $(x,y)$  will satisfy

$$i_{x,y}^{(m)} \stackrel{?}{=} k_d l_m(\mathbf{s}_m^\top \mathbf{n})$$

To estimate how **bad** a specific **n** is, we define the least squares loss function

$$J(\mathbf{n}) = \sum_m J_m(\mathbf{n}) = \sum_m \|k_d l_m(\mathbf{s}_m^\top \mathbf{n}) - i^{(m)}\|^2$$

# Normal Estimation



- ▶ The more loss  $J(\mathbf{n})$   $\mathbf{n}$  has, the less chance  $\mathbf{n}$  is the correct normal vector at pixel  $(x, y)$ .
- ▶ Solve `(cv::Mat::inv, cv::Mat::t)` the following linear system to get the  $\mathbf{n}$  with minimum loss:

$$\mathbf{S}^T \mathbf{S} \mathbf{b} = \mathbf{S}^T \mathbf{i}, \quad \text{where } \mathbf{S} = \begin{bmatrix} l_1 \mathbf{s}_1^T \\ l_2 \mathbf{s}_2^T \\ \vdots \\ l_m \mathbf{s}_m^T \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} i^{(1)} \\ i^{(2)} \\ \vdots \\ i^{(m)} \end{bmatrix} \quad \text{and } \mathbf{b} = k_d \mathbf{n}$$

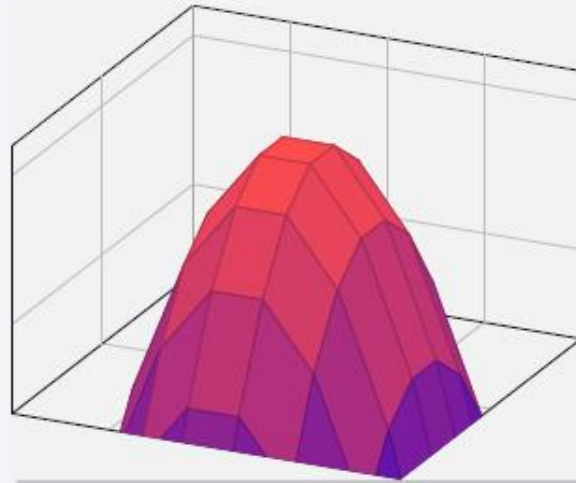
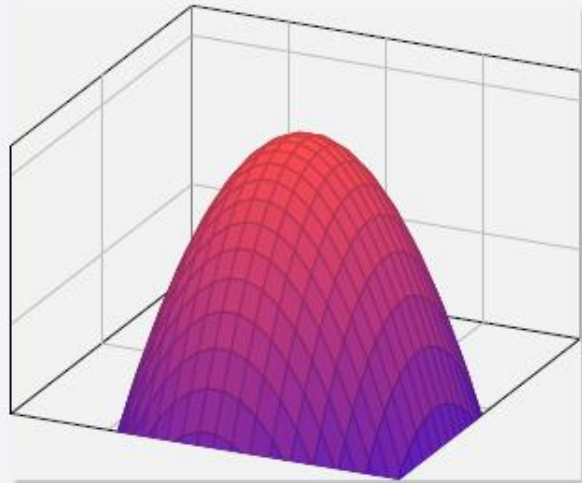
In practice we solve the problem with *QR* or *SVD*.

# Surface Reconstruction

The surface  $z(x, y)$  near pixel  $(x^*, y^*)$  can be approximated by the tangent plane:

$$n_1(x - x^*) + n_2(y - y^*) + n_3(z - z(x^*, y^*)) = 0 \quad (1)$$

where  $(n_1, n_2, n_3)^\top$  is the normal vector at  $(x^*, y^*)$ .



# Surface Reconstruction

The equation 1 can be rewritten as

$$z_{\text{approx}}(x, y) = \left(-\frac{n_1}{n_3}\right)x + \left(-\frac{n_2}{n_3}\right)y + \text{constant}$$

We can reconstruct the surface  $\tilde{z}(x, y)$  as we know the gradient of  $z_{\text{approx}}$  at each pixel, for example, by

$$\tilde{z}(x, y) = \sum_{i=0}^{x-1} \left. \frac{\partial z_{\text{approx}}}{\partial x} \right|_{(i,0)} + \sum_{j=0}^{y-1} \left. \frac{\partial z_{\text{approx}}}{\partial y} \right|_{(x,j)}$$

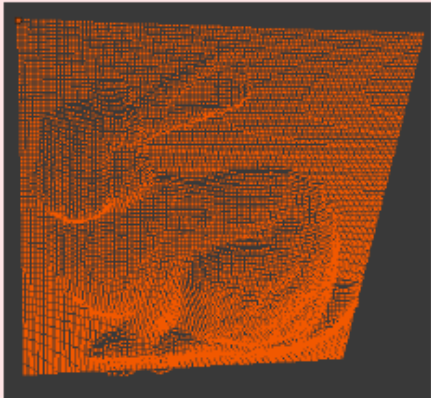
# Surface Reconstruction

## Note

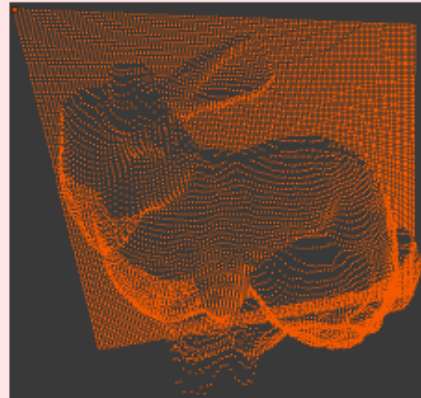
You will probably have to scale  $\tilde{z}$  for visualizing the surface:

$$\tilde{z}_{\text{vis}}(x, y) = \alpha \tilde{z}(x, y)$$

$\alpha = 1$



$\alpha = 4$



# Surface Reconstruction

- Other Tips

- **Weighted Least Squares** measure loss terms with different weights

$$J_{\mathbf{W}}(n) = \sum_m W_m J_m(n) = \sum_m W_m \|\text{kd}l_m(\mathbf{s}_m^T n) - i^{(m)}\|^2$$

$$= \|\mathbf{W} \mathbf{S} b - \mathbf{W} \mathbf{l}\|^2, \quad \text{where } \mathbf{W} = \begin{bmatrix} w_1 & & \\ & w_2 & \\ & & \ddots \end{bmatrix}, w_m = \sqrt{W_m}$$

- **Sanity Check**

$$\frac{\partial^2 z}{\partial x \partial y} = \frac{\partial^2 z}{\partial y \partial x}$$

# Input & Output

- Input:
  - 6 .bmp image
  - LightSource.txt
- Output:
  - .ply file
    - ply~end\_header照打
    - comment alpha有改的話再改數字
    - end\_header以下的點對應上面的x y z 顏色
  - <http://potree.org/demo/plyViewer/plyViewer.html>



# Grading

- 70% Follow the instructions in page 9-14 To reconstruct surfaces
- 10% Experiment with tips mentioned in class or in page 15
- 10% reconstruct surfaces for some special input data
- 15% Take pictures of real objects as input data
- 15% Reconstruct surfaces by solving optimization problems  
(see the Appendix in the course material)

# Deadline

- 期限: 2017/03/30 (四) 11:59:59 pm
- 請將作業壓縮並以學號命名: ex 0987654-hw1.zip
  - 資料夾內包含:
    - 1. 學號-hw1-report.pdf
    - 2. bunny-surface.ply, venus-surface.ply...
    - 3. code 或 完整專案
- 上傳至e3
- Code需要加上對應流程註解
- Report包含程式流程說明 & 執行方式 & 自己多做了哪些功能&其他你想寫的...