# VFX 共同筆記

## 共同筆記範圍：

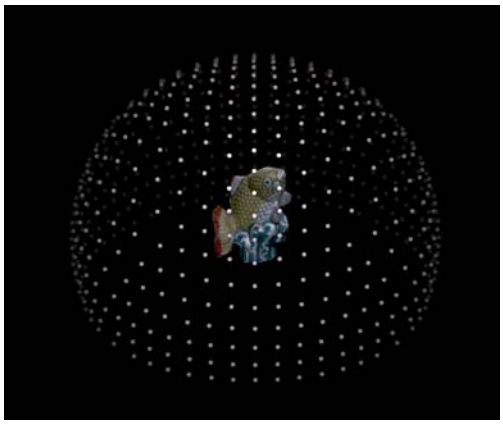
lec12_3dphoto.ppt

3D photography

## 共同筆記同學：

資工碩一 王順英
資管碩一 王亮凱
資管碩一 黃鈞澤

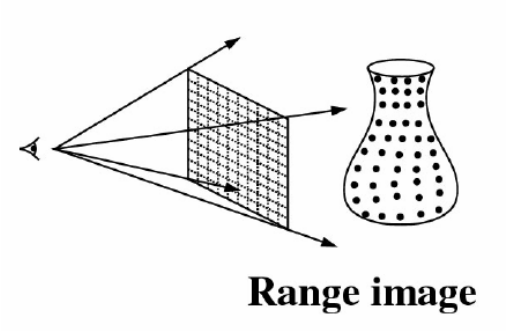
## 3D photography



用攝影的方式把物體的 3D 資訊擷取出來，例如光線、geometry、material。他與 2D photography 最大的不同在於他有 3D information 與物體的 material，所以使用者可以自由變換視角。上圖中（左圖），每一個點代表一個 camera 的 position。

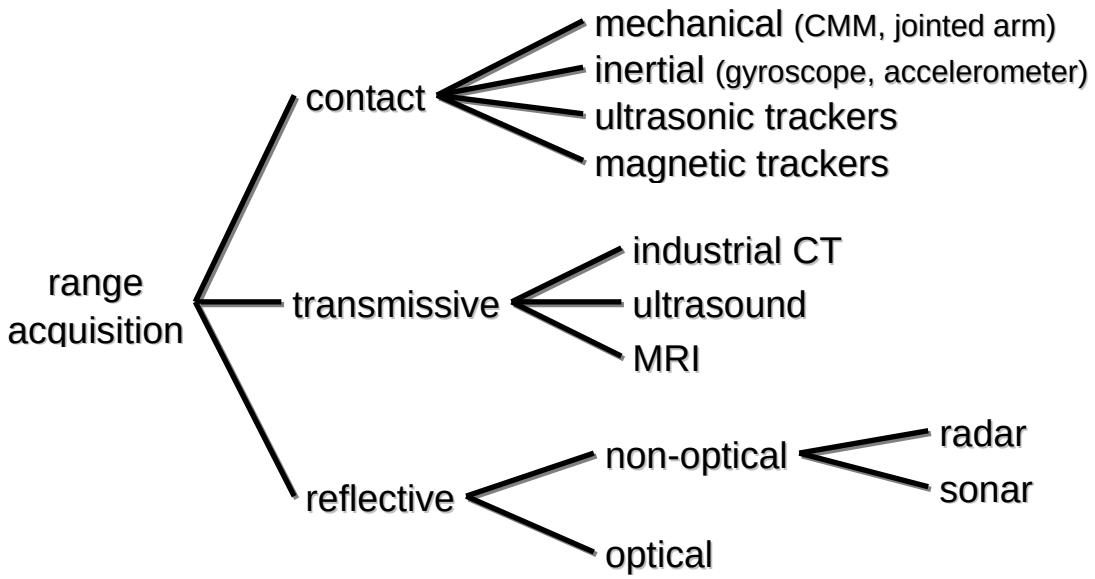
Input：一堆不同視角的 camera images。Output（右圖）：可以從各個不同的角度自由的轉動這個魚去看他

## Range image




從某一個角度看花瓶，並將這個視角的花瓶影像儲存起來，但是儲存的影像並非花頻的色彩而是「深度值」。利用多張 Image Plane 的深度資訊就可以重建這個花瓶的 3D 模型。

## 如何取得 **Range Image**




Range acquisition 主要有三個方法，

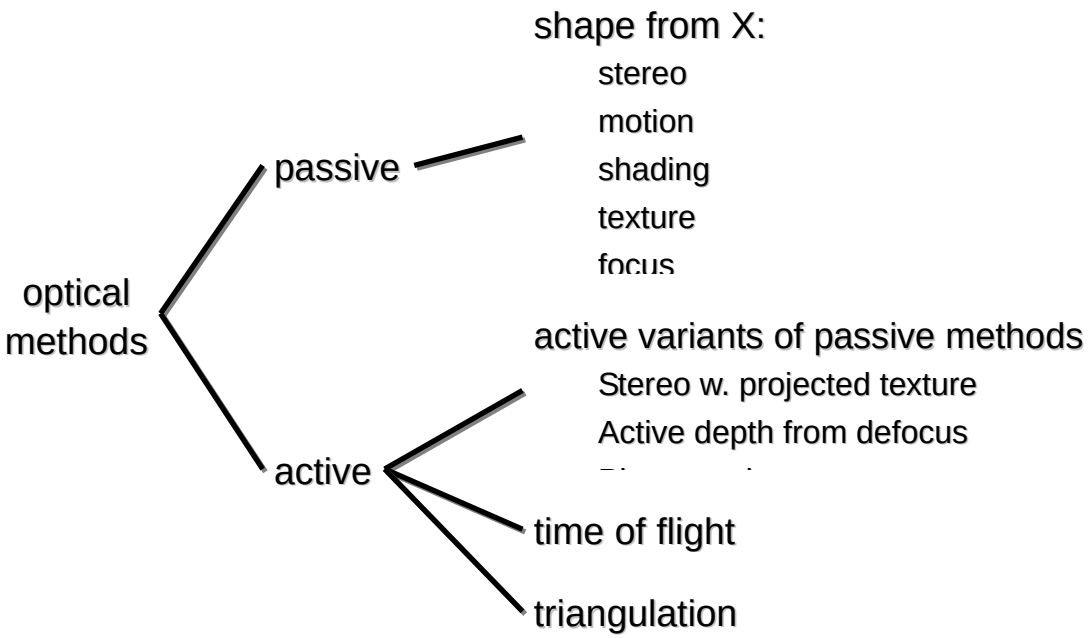
>接觸，是利用機械手臂去點物體，知道手臂的角度與長度就可以推敲接觸到的點的 3D 位置。或者利用電磁波，藉由 transmitter 和 receiver 去計算說兩著之間的距離就可以算出 3D 的座標。

>切下去，例如斷層掃瞄，一層一層切下去，利用穿透性不同得到不同層物體的 2D image 的樣子並把他組合起來就可得到 3D 模型。

>反射，一種是利用 optical 另外一種不是。原理是用到反射，EX 光雷達電磁波 SONAR 等等。

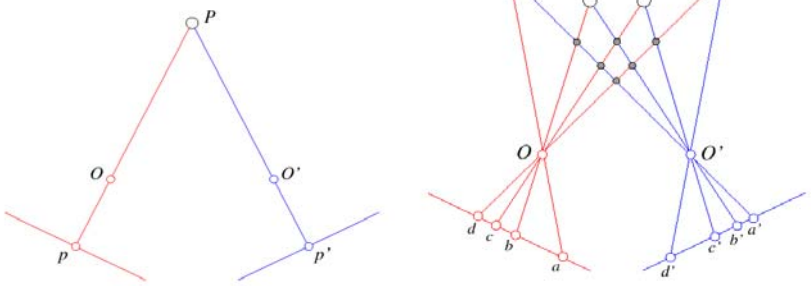
## **Reflective** 中的 **Optical** 的方法

射出一個光波，計算射出去再回來的時間就可以算出兩點的距離，利用這些資訊來重建物體的 geometry。主要分爲 passive 和 non passive。Passive 方式是利用一台或多台 device 所得到的 image 去分析估計物體的 geometry，而 active 的方式則是由 device 涉入一些資訊，並且觀察這些資訊的變化來估計物體的 geometry。




## Passive approaches - Steoreo system

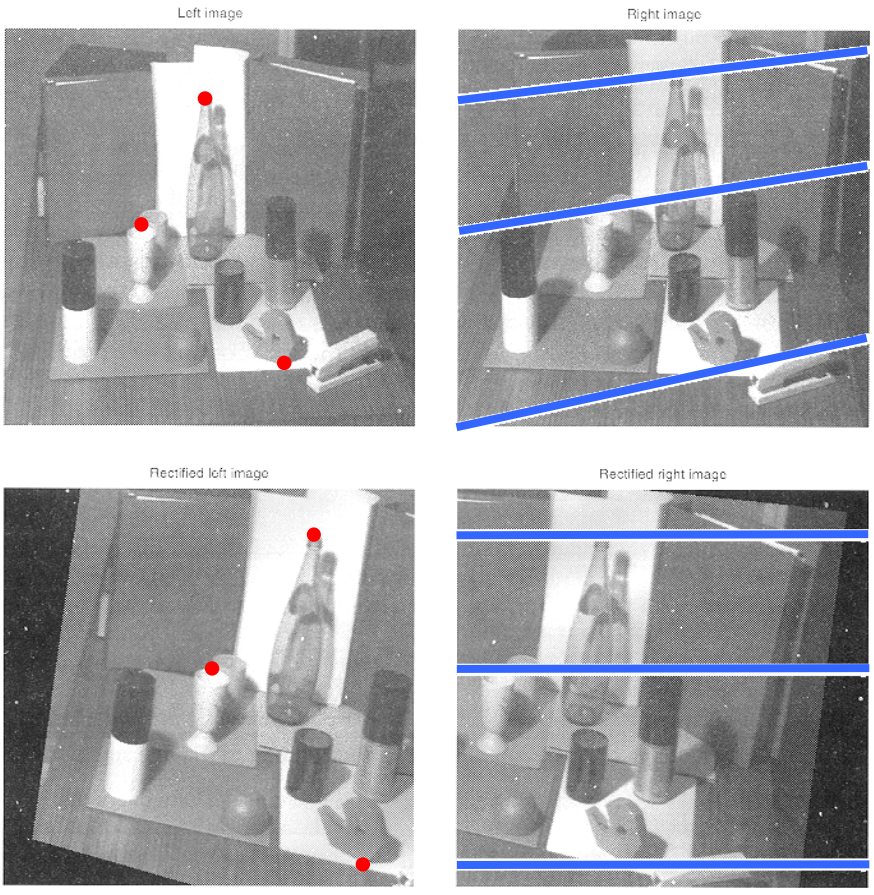
利用兩張估計人左眼和右眼看到的東西是不一樣的，利用 blue channel 和 red channel 射出兩張影像，這兩章影像的距離不同來讓人感覺物體便立體了




取得兩張 image plane 和已經調較好的 camera optical center，我們可以分別從兩個 image plane 中的兩點利用 triangulation 求得 P 的真正位置。但是要怎麼知道 image plane 中那些點是對應的是我們必須先克服的。

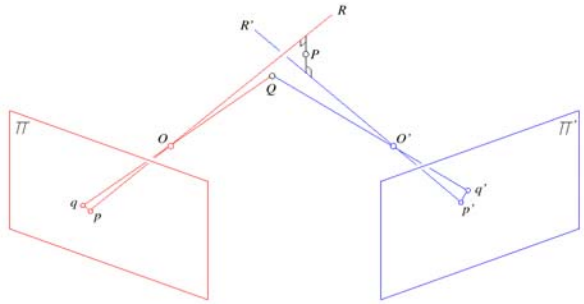
步驟

1. Camera calibration
2. Image rectification: simplifies the search for correspondences
   利用先前介紹過的 Epipolar line 去加快對應點的收尋。



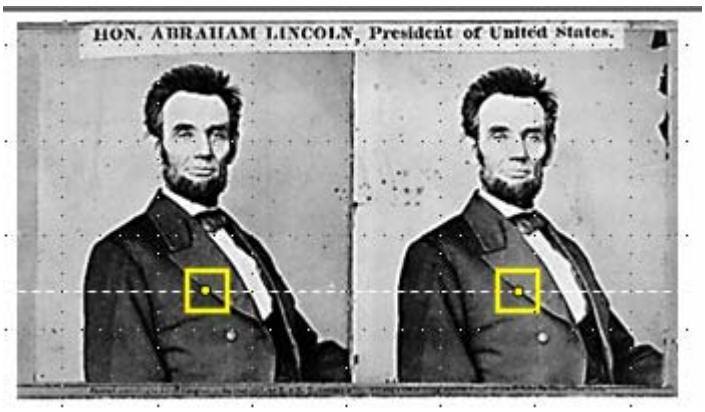
3. Correspondence: which item in the left image corresponds to which item in the right image
4. Reconstruction: recovers 3-D information from the 2-D correspondences

假設我們已經調整好影像，取得兩張影像對應的 epipolar line，我們要如何去求得 correspondence？ 利用 Disparity，定義爲左邊影像中某個 pixel 在右邊影像的 epipolar line 上的 horizontal displacement 。基本上 disparity 與物體的距離成反比，如同日常生活中的體驗，離我們月接近的物體移動一點點眼睛中的影像位移就會很大。




求得兩張影像中相對應點的 disparity 之後利用 p,q 兩點與 camera optical center 所形成的兩條射線的交點去求出點實際 3D 的座標，如果這兩條射線並沒有相交，則去找空間中一點最接近這兩條射線當作他們的交點。

## 如何找出對應點的演算法 Basic stereo algorithm




由於我們已經知道左邊影像中 epipolar line 上的每一點在右邊影像中的對應點必定落在右邊影像的 epipolar line 上面，所以演算法如下

For each epipolar line in the left image

For each pixel

Compare every pixel on the same epipolar line in the right image with the minimum cost (顏色差異)

問題是會 noisy，改進的方法是利用 windows，在 epipolar line 上面不利用點來比較，而是開一個 window 來比較 window 中每個點的差異。並且加入一個 disparity 的參數讓這個 window 可以作上下左右等等的 shift。這個 disparity 代表的是在兩張 image 中各開一個 window，他們的 window center 座標的差值。

改進後的演算法如下：

For each pixel

For each disparity

For each pixel in the window

Computer the difference

Fine the disparity with the min SSD

改變一下 LOOP 的順序加快速度：

For each disparity

For each pixel

For each pixel in the window

Computer the difference

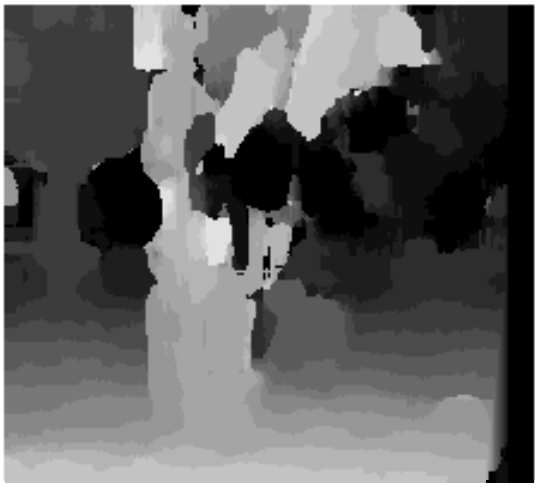
Fine the disparity with the min SSD at each pixel

會加快速度的原因是，這樣可以利用到之前已經計算過的 pixel 差值，不必重複計算（window shift 的時候會覆蓋到重複的 pixel）

## Window size 的影響

在這樣的演算法下，比較小的 window 給你比較 depth 上的 detail 比較多，但是比較 noisy。用 window size 大的話，會 over smooth。所以要選擇一個適當的大小。



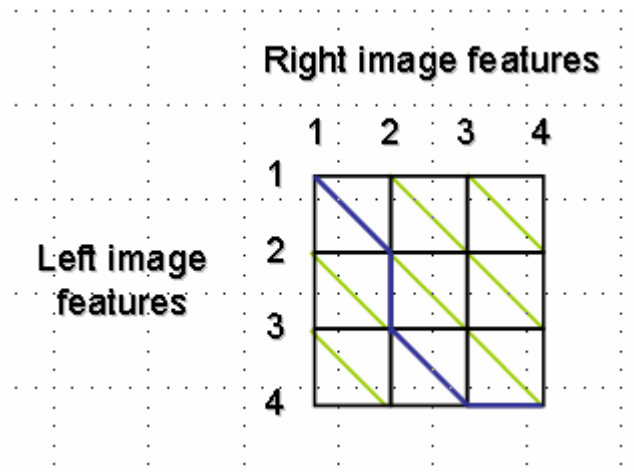
## Non-square windows

另外一種演算法的變化，就是改變 window 的形式，例如使用其他形狀的 window，或者是給與不同權重的 window，或者是用一半的 window。例如在 occlusion 的情況下對應點的 pixel 被遮蔽住了使得 disparity 會變的 noisy 這時候可以藉由一半的 window 去避開 occlusion 所造成的影響。

另外一種改進的方式是利用 Global disparity 來找出最好的 correspondence。由於之前得演算法都是利用單一個 pixel 算完以後的結果當作結果，並沒有作整體的考量，所以結果會很 noisy。如果加上了杭潢慬的資訊便可以減少這種 noisy 的現象，這類型的方法有三種：

## 1 Dynamic programming

首先假設我們的影像是符合 ordering constrain，也就是說，在沒有 occlusion 的情況下，我們在 epipolar line 上面找到的對應點一定是依序出現的，不會出現忽左忽右震盪的情形（只會單純往右或者往左依序遞增）。一旦符合這樣的條件，我們可以把計算 disparity 的問題 map 到 dynamic programming：




將兩張影像上 epipolar line 上面的 pixel 分別對應到 row,column，從這個圖形的左上角開始找一條最短的路徑到圖形的右下角，所得的結果將會使得這條 epipolar line 的 disparity 是 global 最小。

## 2 Energy minimization

在這裡我們首先假設兩張 image 之間每個 pixel 的 disparity 是很 smooth 的。則我們將試著滿足以下式子：

Minimize energy function:

$$E_{data}+\lambda E_{smoothness}$$

$E_{data}$: how well does disparity match data

$E_{smoothness}$: how well does disparity match that of neighbors – regularization

其中 Edata 的意思代表說存在某一個 function D(x)，而左邊 image 中任何一個 pixel x 的顏色值會跟右邊 image 位置爲 ( x + D(x) ) 的 pixel 相近，換句話說 D(X)就代表一個適當的對應關係，他描

述了左邊的 image 是如何與右邊的 image 互相對應的。

而 Esmooth 代表說每一個 pixel 與其對應 pixel 之間的 disparity 必須是沒有震盪的，例如在(2,2)到(20,2)這些點的 disparity 都是 2，但是其中(5,20)的 disparity 卻是 35，這樣就是不 smooth。也就是說(5,20)這個點所猜測的對應點，只是顏色相似，但是他的 spatial 位置可能跑太遠了因此有可能是不好的 correspondence。

我們將以上的演算法概念寫成數學式如下：

- Matching Cost Formulated as Energy
  - "data" term penalizing bad matches

$$D(x,y,d) = \left|\mathbf{I}(x,y) - \mathbf{J}(x+d,y)\right|$$

  - "neighborhood term" encouraging spatial smoothness

$$V(d_1,d_2) = \text{cost of adjacent pixels with labels d1 and d2}$$

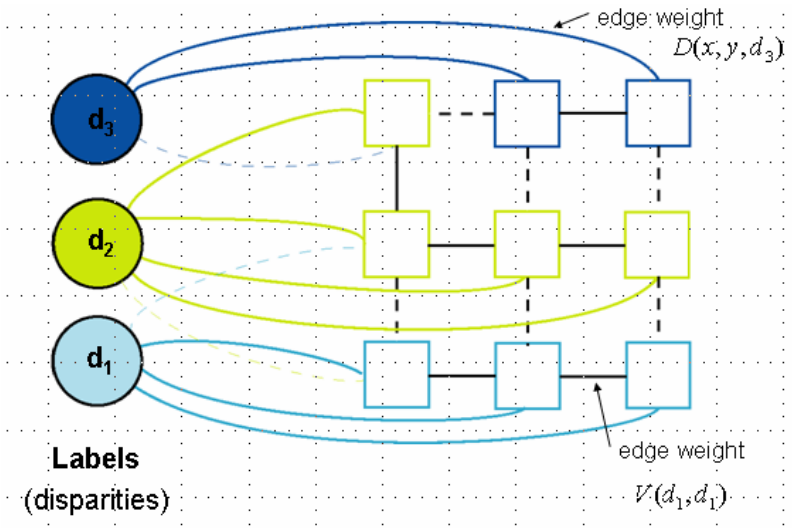
$$= \left|d_1 - d_2\right| \quad \text{(or something similar)}$$

$$E = \sum_{(x,y)} D(x,y,d_{x,y}) + \sum_{neighbors\,(x1,y1),(x2,y2)} V(d_{x1,y1}, d_{x2,y2})$$

其中 neighborhood term 的部分可以依造需求的不同作調整，如果已經知道 image 本身是一個比較 smooth 的 disparity，則我們可以簡單的用 gradient descent 來描述他的 smooth。相反的，在 smooth 情況較差下我們要設計一個比較複雜的式子來提高 robust。

## 3 Energy minimization via graph cuts

首先我們定義一個 graph cuts 的圖形：圖形中一開始會有 N + L 個 node，代表了 n 個 pixel 與 L 個 不同的 disparity。每個 pixel node 會連出一條 edge 到某一個 disparity node 上，而且 pixel node 之間相鄰的會存在一條 edge。很明顯的，pixel node 所連到的 disparity node 代表了該 pixel 的 disparity 數值。另外，相鄰兩個 pixel 之間會有一個 edge cost (兩點間 disparity 的差值)。




目的是爲了將圖形中屬於相同 disparity 的 pixel node 當作一個 component，並且只留下同一個 component 之間的 pixel node 相鄰的 edge。換句話說，執行到最後不同 component 之間不會有 edge 相連、同一個 component 才會有 edge，而且刪除的 edge 的 cost 必須是最大。這樣一來所得的各 component 之間的 cost 會是最小也就是達到 energy 最小的目的了。

爲什麼我們要把這個問題轉換成 graph cuts 的題目呢？原因是已經存在一演算法來解 graph cuts 的 minimum cost 的問題。所以可以讓我們簡單的求出 energy function 的最佳化。

Graph cuts 也是一種 labeling 的問題，當圖形中只有兩種 label 數值時很好解出答案，他可以在 polynomial time 完成。如果 labeling 大於 2，就變成 NP hard。但是，已經有人提出一個 Efficient approximation algorithms 來解答，他保證算出來的 local minimum 會小於 global minimum 的兩倍。

這個改進的演算法如下：利用兩個 binary evaluation（expansion, swap）去將圖形的 energy 下降，這個演算法保證每一步一定會下降 energy， 直到無法再下降 energy（收斂了）就結束。

SWAP

1. Start with an arbitrary labeling
2. Cycle through every label pair *(A,B)* in some order

2.1 Find the lowest *E* labeling within a single *AB*-swap

2.2 Go there if it's lower *E* than the current labeling

3. If *E* did not decrease in the cycle, we're done

Otherwise, go to step 2

EXPANSION

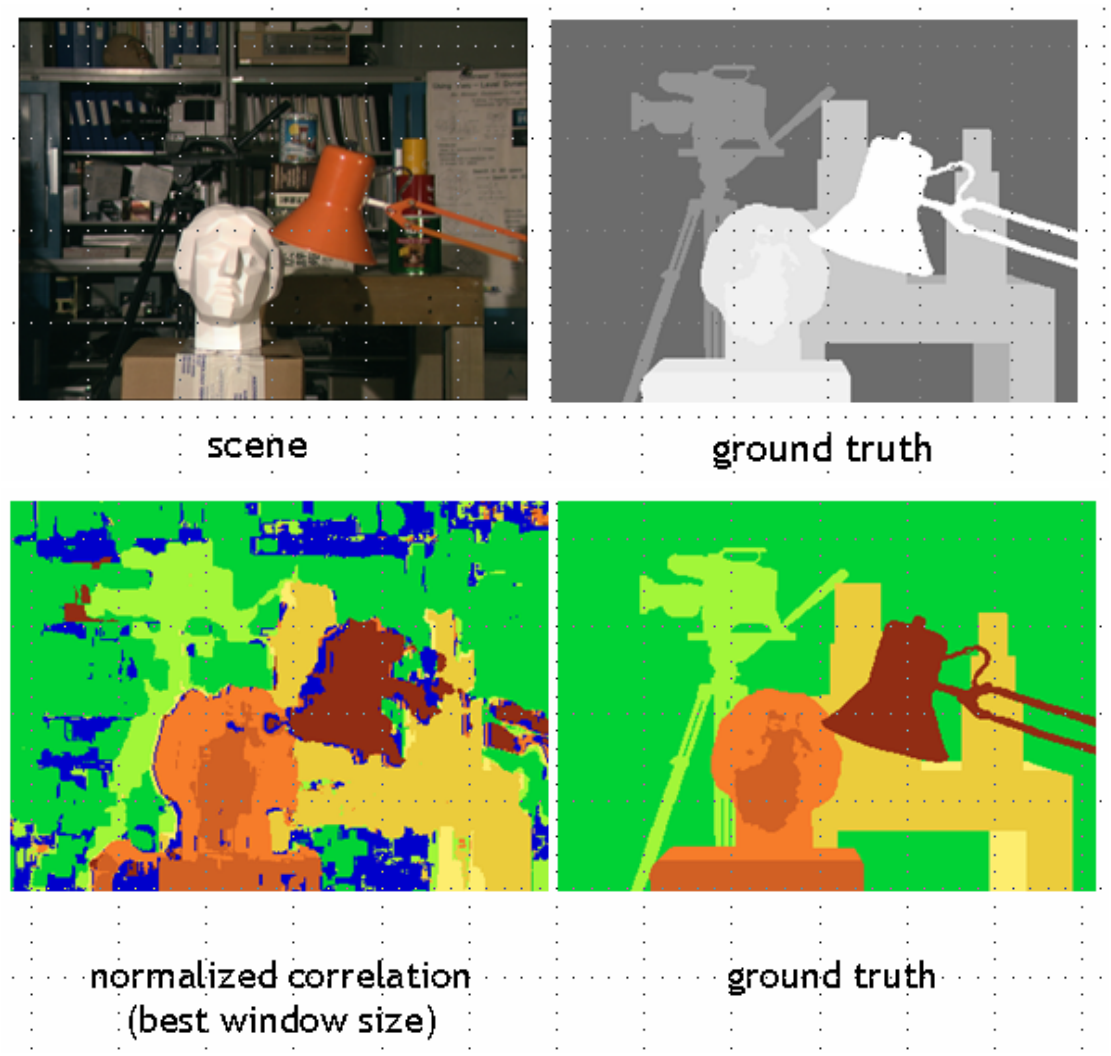
1. Start with an arbitrary labeling
2. Cycle through every label *A* in some order

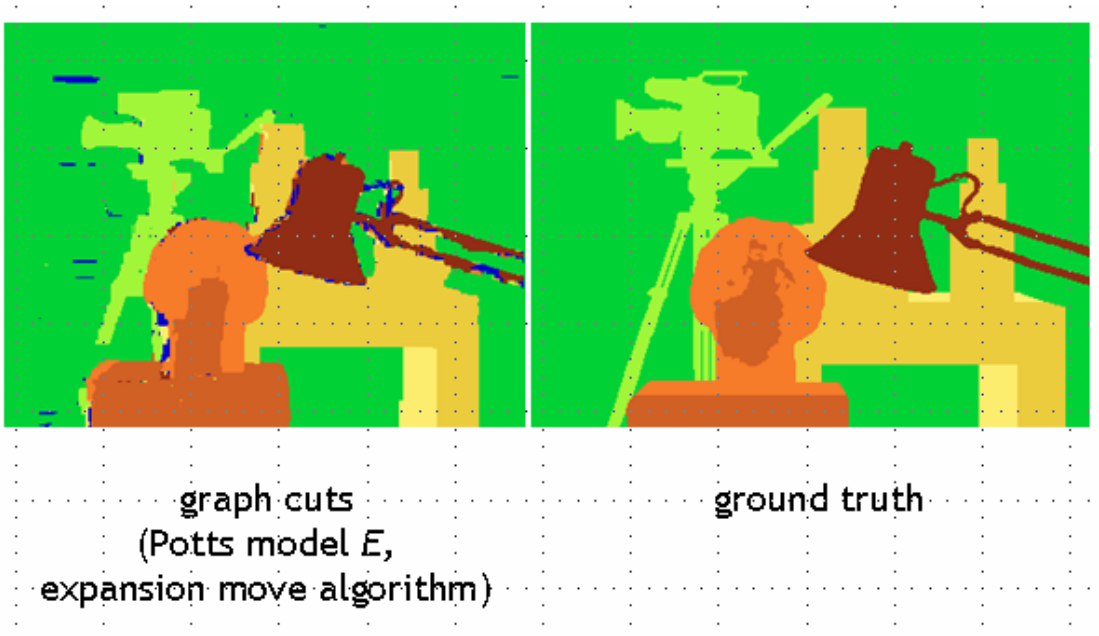
2.1 Find the lowest *E* labeling within a single *A*-expansion

2.2 Go there if it's lower *E* than the current labeling

3. If *E* did not decrease in the cycle, we're done  Otherwise, go to step 2

比較一下 window 與 graph cuts 兩種 labeling 的方法：明顯的知道誰比較好.... Graph cuts.

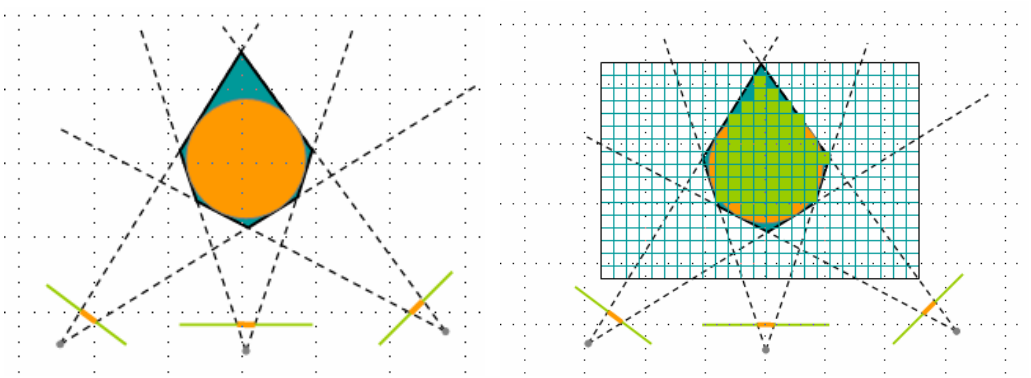





而 labeling 這個問題可以應用在 denoising 上面，想辦法去 smooth 你的影像讓你的 noise 消失，但是卻不會讓你原來的影像走樣。有很多 computer vision 的問題可以用 labeling 來解。

## 4 Volumetric multiview approaches

一開始我們先假設物體爲一個 3D 的正立方體，接著我們利用每一張 image 根據 camera 的位置去雕刻這個正立方體，讓這個正立方體在這個角度的 convex hull 會跟 image 上面看到的很像。當然，這樣會有誤差，例如花瓶是中空的，但是用這個方法估計出來的 model 卻不是中空的。



其中一個方法叫做 Silhouette carving，方法就是利用影像中的物體當作 visual hull，並從這個 visual hull 的邊緣延伸出射線並且交於我們一開始假設的 3D 正立方體。落在 visual hull 裡面的部分我們就留著，否則就捨棄，所以他是一個 binary 的結果。

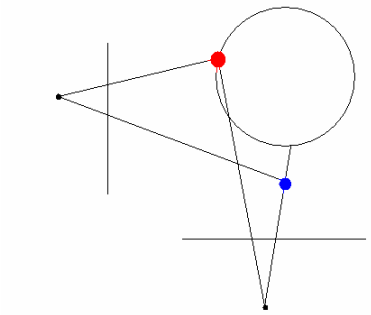
但是這個方法很容易出現問題，結果不太好。當使用者給的 input images 太少的時候結果總是很粗糙。

改進的另一個演算法叫做 Voxel coloring 。他的目的是根據每一個 image 上觀察到的 pixel 是否落在那個 3d 正立方體中之外還去檢查他的顏色數值，根據每個 view image 的顏色數值來估計這個正立方體在這個 pixel 的位置到底是不是真的存在而且顏色是什麼。

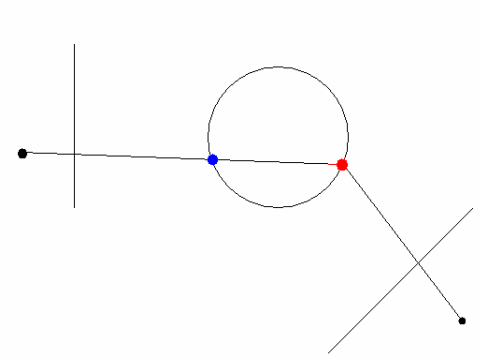
## Voxel Coloring

Basic idea: 把 voxel 投影到 n 張 image 上，每一個 pixel 的顏色都很接近則表示，則表示這個 voxel 是物體表面上的一個 voxel，所以顏色才會 consistency。把這個 voxel 的顏色當成 n 個顏色的平均。

如果這個 n 個 pixel 的顏色不夠接近，則表示大家看到的東西不一樣。實際上這個 voxel 不存在。如下圖的藍點:

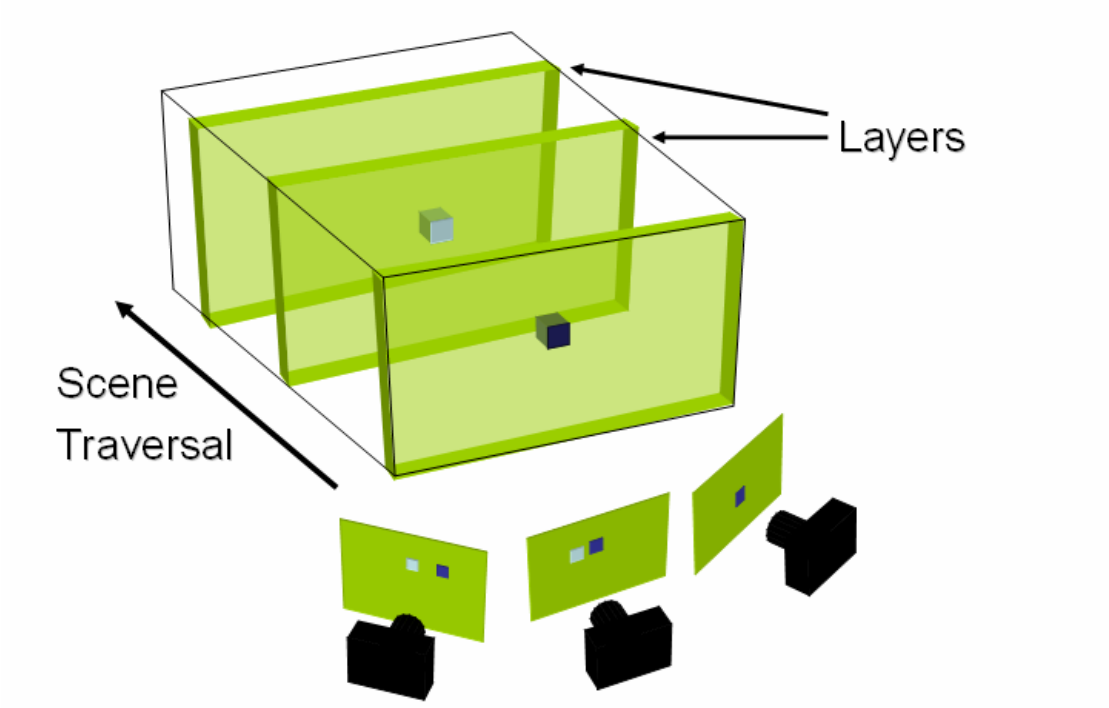


但會有 occlusion 的情況要怎麼辦，如下圖:



紅色的 voxel 雖然存在，可是投影到兩張 image 上卻是不同的顏色。所以我們要可以處理 occlusion，不然一開始的 basic idea 就不對了。

我們先來看以下的圖，有兩個 voxel，一個是深藍色，一個是淺藍色的，深藍色的 voxel 沒有問題在三張 image 上的顏色是一樣的，但是淺藍色的，在最右邊的 image 卻是深藍色的。

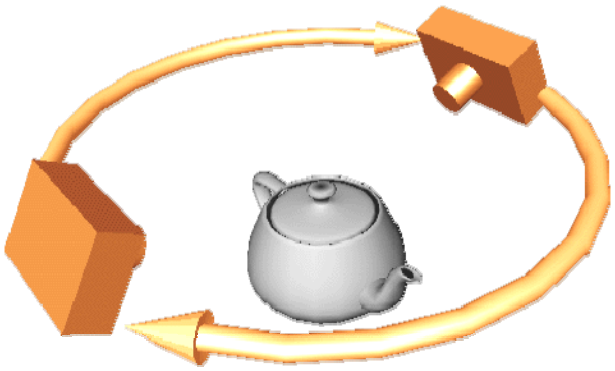



但如果我們知道淺藍色是被深藍色擋住的話，就可以解決這個問題，但我們要如何知道?基本上把所有會擋到別人的 voxel 你都先做，所以處理淺藍色的點時，你已經知道深藍色這個點是存在的，他的投影其實是被深藍色所拿去的，而不是顏色不一致，而判別爲不存在。

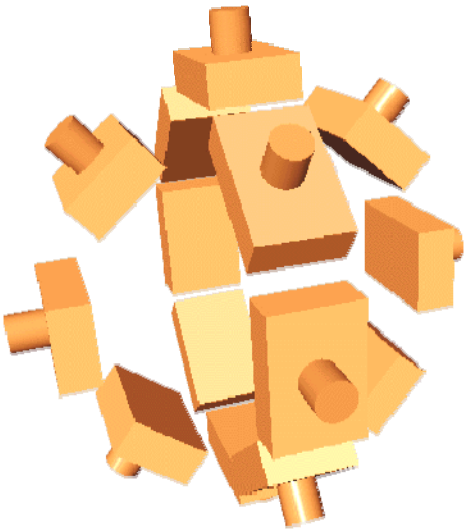
所以爲了解決 occlusion 的問題，你的 solution 就是會擋到別人的都先做。你必需

要規定你的 camera 有一定的 order，如上圖 camera 都在同一面，一層一層往後面做就可以避免 occlusion。

除了這樣的 camera 的 configuration 滿足所有的 occluder 都會先做之外，還有其它一些可能的情況。下圖所有的 camera 都在茶壺的上方，由上面往下看它，



或是所有的 camera 都往外看，都可以用剛才的演算法來做，但其它的情況不見得會 work。它的方式是找到一個 camera configuration 的方式，確保你的 camera 都會在 sweep plane 的同一邊。當 plane 由一邊 sweep 到另一邊時，所有會擋到別人的都會被先被處理過。



下圖的例子就是 camera 由上往下看，把要 scan 的物體放在桌子上，如花或是恐龍，原則上是 camera rotate  但效果同等於 table rotate，總共拍了 21 張 image，

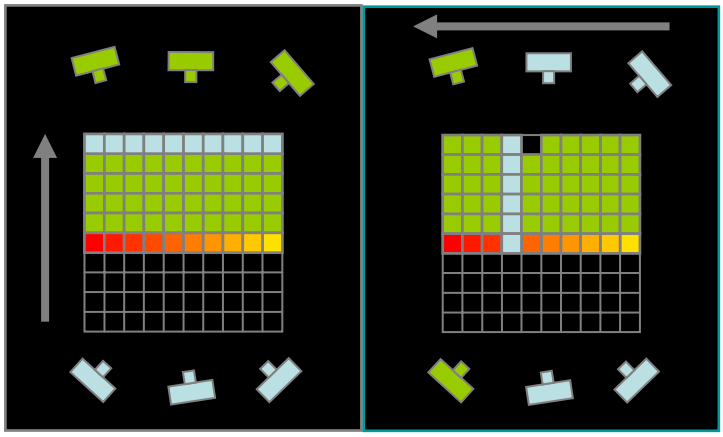
結果的 image :

它除了把 volume 找出來外，每個 voxel 也找出它的顏色，它也把顏色貼上去。邊線地方會比較 noisy 因爲會有 color blending，所有會有誤判的情況。

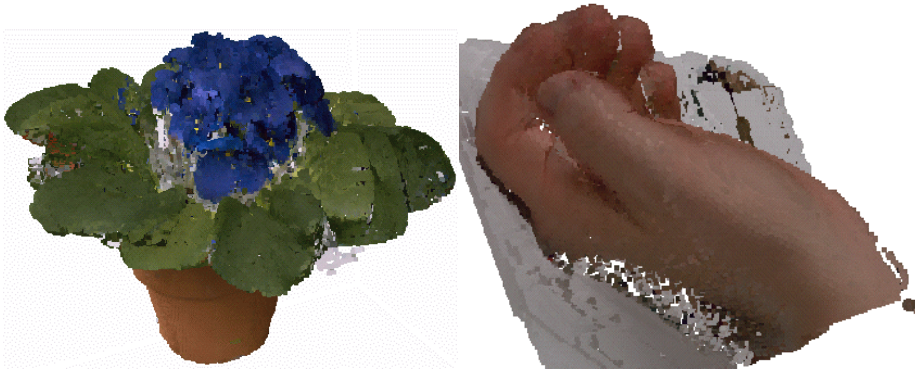
Voxel coloring 會有什麼問題呢 ？ 第一個是你的 camera configuration 必須要確保所有的 camera 是在物體的同一邊，有時不能達到這樣的要求，我們需要一個更 general 的 algorithm 來解決這個問題，Unconstrained camera positions，Unconstrained scene geometry。

## Space carving

基本上想法和之前差不多，先 initialize 一個 volume ，這個 volume 要比真正的 object 要大，一開始是一個 cube。在這個 volume 表面上選一個 voxel，投影到每一張 image 上，如果投影點的顏色是不一致的話，就把這個 voxel 挖掉，一個一個 repeat 直到收斂爲止。你可以在這個 volume 上任意的選點，做到不動爲止。但是這樣會比較慢，最好是從 volume 的六個面做 sweeping，每次 sweep 時只考慮在這個 sweep plane 同一邊的 camera，在另一邊的 camera 就不考慮。
如下圖 ：



結果：



## Active approaches

之前所講的是 passive approach ，只從 image 本身去分析 geometry。而 active approach 會投射一些 information 到 scene 裡面，再去分析那些 information，來求得 geometry。

## Time of flight

射出去某種聲波或是光波，計算它彈回來的時間，除以 2 就可以得到距離。可是這種方法一次只能看一個點。掃了很多個點之後，把這些點集合起來就可以得到 3D model。好處是可以掃比較大的東西，但是精確度會比較低。

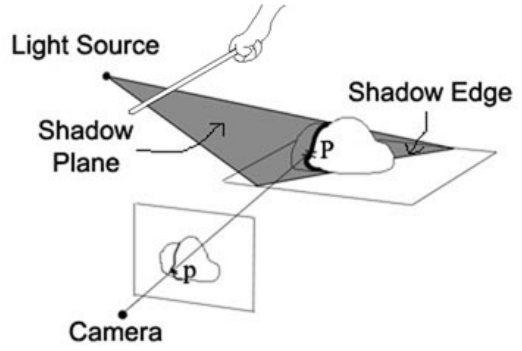
## Laser scanning (triangulation)

是現在最常用的儀器，好處是很精確，但是能掃的範圍比較小一點。它用的方式也是 triangulation。原理是利用已知 laser 所投影的平面，加上 pixel 和 camera center 的射線(因爲 camera calibrate 過的 )，利用平面和射線的交點求出 geometry。只是這樣的 device 很貴。

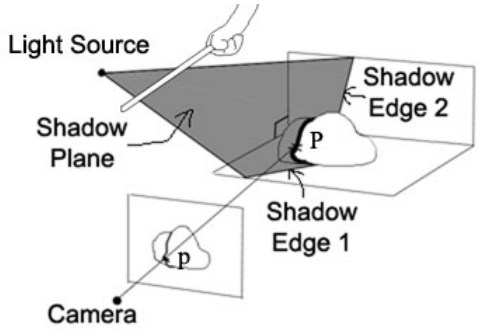
## Shadow scanning

所以有人提出比較便宜的方案，利用你的燈加上一根棍子來取代 laser，加上一個 camera，來 scan 物品的 geometry。

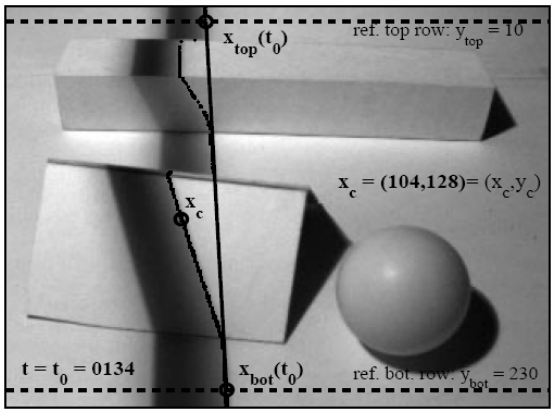
基本的原理是假設我們知道 light source 在那裡，找出 shadow edge，light source 和 shadow edge 可以求得 shadow plane，加上有 calibrate 後的 camera 知道那物體上一點 p 到 camera center 形成的射線。利用 Shadow plane 和射線的交點的 3D 位置，和 triangulation，就可得物體的 geometry。




這個方法的壞處是你必須知道你的 light source 才能得到 shadow plane，如果不想去算 light source 的 3D 位置時，你就用兩個 plane，由兩個 plane 上的 shadow edge 就可以求出 shadow plane。




再來的問題是要如何去求得 shadow line?




利用在平面上的影子的兩端，可以求得原始沒有物體時的 shadow line，而要求出影子在某一時間的位置，就去看某一個位置的水平 scan line 的 intensity histogram( 假設平面都是白的 )，找一個 threshold 和 intensity function 第一個交點的位置所對應的 x，即爲影子在這條水平線上的位置，把在平面上的水平線的所

有影子的位置都算出，再做一個 fitting，就可以求出每個時間點的 shadow line 的位置了。

在算平面上的影子時，我們是用 spatial analogy，而要算物體上的影子，因爲物體並不一定是白色，所以要用 temporal analogy。對一個物體來講，去看時間軸上，它的 intensity 是如何變化的，什麼時間點它第一次被 shadow cover 到。

這個方法的 error 約只有 0.1%，還有之前雖然有說到關於物體的 texture 雖然沒有限制，但是還是會影響結果。而當物體的某一部份一直都在影子裡時，就沒辦法知道影子何時 cover 到它，或是位在視角沒有被看到的地方，都沒有辦法求出那個地方的 geometry。所以看是要變換光源，視角或是旋轉 model 來解決。

另外也可以使用太陽來當 light source，就可以掃比較大的東西，不一定是要於在桌上。

## Active variants of passive approaches

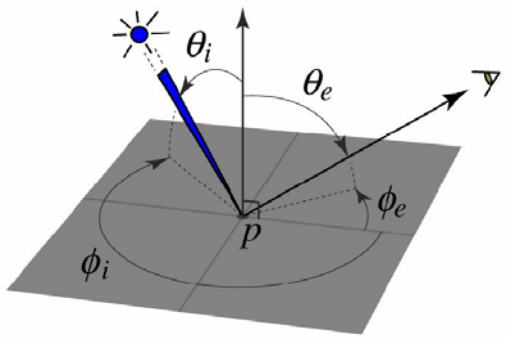
### BRDF

是描述材質反射特性的一種函數，給定某一個視角，某一個光入射的方向，那個光有百分之多少會進入你的眼睛，你看到的亮度是多少(ratio)。

BRDF　　光的強度和 cos 成正比

$$I = \rho(l,v) \times (l \cdot n)\gamma$$

看到的亮度　　光的強度




### Diffuse reflection ( Lambertian )

如果物體是 diffuse 的話，那不管光在什麼方向，視角在那裡，光的反射率都是一樣的，所以它的 BRDF 是 constant function( albedo )，像是石膏像，牆壁等等的。

$$\rho(l,v) = k_d$$

而且假設光的強度是 1，則

$$I = k_d \mathbf{N} \cdot \mathbf{L}$$

## Photometric stereo

就是想辦法估計某一點的 depth，把物體固定不動，照相機也不動，移動光估計那一點的 depth。那我們先去估計那一點的 normal，

$$I_1 = k_d \mathbf{N} \cdot \mathbf{L_1}$$

$$I_2 = k_d \mathbf{N} \cdot \mathbf{L_2}$$

$$I_3 = k_d \mathbf{N} \cdot \mathbf{L_3}$$

先分別打三盞燈，得到 I1，I2，I3。假設已知光的方向，那我們現在要求的是這一個點的 albedo 和 normal。Kd 是一個未知數，而 normal 是 3 個未知數，但把 normal normalize 之後只需要兩個未知數。所以我們需要三個等式才能解。

$$\underbrace{\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}}_{\mathbf{I}_{3\times 1}} = \underbrace{\begin{bmatrix} \mathbf{L_1}^T \\ \mathbf{L_2}^T \\ \mathbf{L_3}^T \end{bmatrix}}_{\mathbf{L}_{3\times 3}} \underbrace{k_d \mathbf{N}}_{\mathbf{G}_{3\times 1}}$$

$$\mathbf{G} = \mathbf{L}^{-1}\mathbf{I}$$

$$k_d = \|\mathbf{G}\|$$

$$\mathbf{N} = \frac{1}{k_d}\mathbf{G}$$

求的過程如上，最後求出的 vector G，它的長度就是 albedo，而 normalize 後的 vector 方向就是 normal。

如果有多盞燈時，那就更 robust 一點。當燈有 n 盞時，就是爲 n 個 equation 求三個未知數。就用 SVD 來解這個 equation。目前所假設的都是沒有 reflection，沒有 shadow 的情況之下。

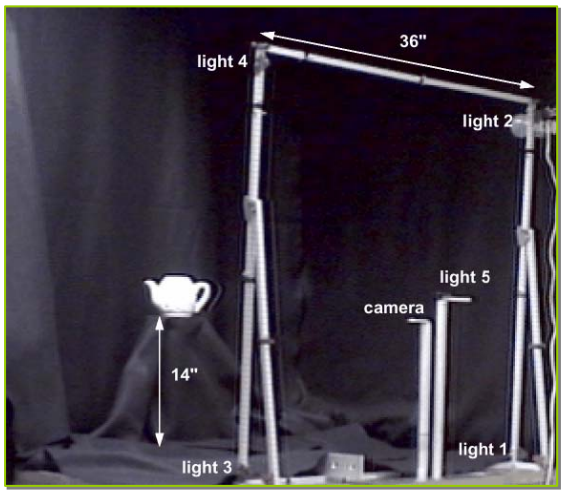
## Trick for handling shadows

當有更多盞燈時，就可以解決 shadow 的問題，把每一個 equation 乘上一個 weight

值，這個 weighting 和這個 pixel 的 intensity 有關。因爲 shadow 比較暗，所以當一個 pixel 的 intensity 比較暗時，就比較不重要。或是去猜測那些點可能在 shadow 裡面，用不在 shadow 裡的那些點去求 N 和 albedo 就好。也可以用 RANSAC，任意選 3 個點去求 N 和 albedo，再去其它點是否會滿足這個 N 和 albedo。

## Photometric Stereo Setup

有五盞燈和一個 camera，燈會分別閃，可得到五張不同 intensity 的 image



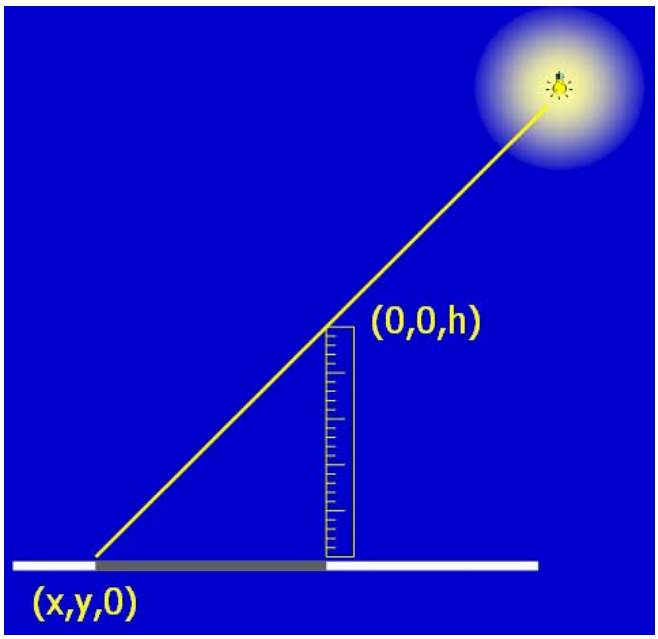

## Procedure

第一步是先 calibrate camera，但如果只是要求點的 normal，則不需要此步。
第二步是 Calibrate light directions/intensities( Li )
第三步是 Photographing objects (HDR recommended)，前面是 physical 的 case 是 linear 的，但是 camera 本身不是 linear 的，所以最好用 HDR 可以高正確性且 linear 的 response，比較合於原來的 model。
第四步就是利用之前所提的線性方式來 Estimate normal。
第五步就是把 normal 變成 depth。

## Estimating light directions

Camera calibrate 之前已經講過了，所以先來討論怎麼估計 light direction。一個方式是先放一個金屬球在你的場景之中，由金屬球的 specular 可得光的方向在那，而球上每一點的 normal 方向都可以求出來。

另一個方法是放一個尺，已知尺的高度 h，尺在(0,0,0)的位置(因爲 camera calibrate 過，可知地板的位置)，由燈的影子可以求得燈的方向。




再來就是去拍物體，是使用 HDR，那每一盞燈都需要 normalize intensity，因爲沒有辦法確定每一盞的強度都是一樣的，且每一盞到物體的距離都不一樣，所以要把這些都去 normalize，讓光的強度都是一。

再來是如何把 normal 轉成 depth，depth 是在 z 軸上的方向，而 x 方向 normal 實際上就是 z 方向對 x 做偏微分，y 方向 normal 就是 z 對 y 做偏微分。把 normal 的 z 值 scale 成-1，x，y 變成 p 和 q，p 和 q 就是我們的 input。意思是說我們求出來的 z plane，對 x 微分會長的像 p，對 y 微分會長的像 q。是我們求出的 normal 給 depth 的限制。稱之爲 Edata。

我們要估計 depth map，給 depth map 評方，分數最高的就是我們想要的 depth map。評分的方式是首先它要滿足 Edata，猜出來的好的 depth map，它每一個點的微分(z 對 x 和 y 微分)要和 normal 一樣像。第二是 depth 是 smooth 的，z 的 function 要是平滑的，它的二階微分要越小越好。第三個是 Econs，就是說如果我知道某一些點的 depth 是多少，那就可把這些點的 depth 當成 constrain。把這個加起來就是我們的 energy function，只要 minimize 這個 function 就好。

而我們要求的 zij 是一個 quadratic function，要求一個 quadratic function 的最小值，最後一定會轉成一個線性系統，只就解這個線性系統就可以了。

$$(n_x, n_y, n_z) = (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, -1) = (p, q, -1)$$

$$E = E_{data} + E_{smooth} + E_{cons}$$

$$= \sum_{i,j} w_{data} * \left[ \left( \frac{\partial z(i,j)}{\partial x} - p_{ij} \right)^2 + \left( \frac{\partial z(i,j)}{\partial y} - q_{ij} \right)^2 \right]$$

$$+ \sum_{i,j} w_{smooth} * \left[ \left( \frac{\partial^2 z(i,j)}{\partial x^2} \right)^2 + 2 \left( \frac{\partial^2 z(i,j)}{\partial x \partial y} \right)^2 + \left( \frac{\partial^2 z(i,j)}{\partial y^2} \right)^2 \right]$$

$$+ \sum_{(i,j) \in Cons} w_{cons} * \left( z(i,j) - c_{ij} \right)^2$$

$$E = \frac{1}{2} z^T A z - b^T z + c \quad \equiv \quad Az = b$$

## Limitations

1.只能處理 diffuse 的 object(laser scanning 也是如此)

2.不能處理 shadow

3.沒有 inter-reflection

4.camera 需要 calibrate

如果要處理 shining object 只能把物體塗上白粉，再去 scan 它的 geometry。

## Example-based photometric stereo

爲了處理不是diffuse的object，基本的原理是在場景中放一個和測量的物體是同一材質做的球。因爲球每一點的normal已知，不管物體BRDF是什麼，假設光在足夠遠的地方，那麼對球和物體的入射光的方向是一樣的，照相機足夠遠，viewing direction也是constant。

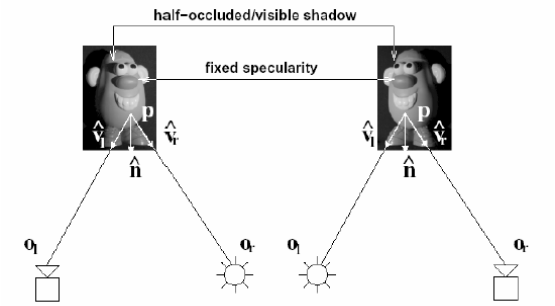
所以對每一點L和V是一樣的，且BRDF對球和物體是一樣的。所以在物體上某一點的normal和球上一點是一樣的話，那他們的顏色應該也是一樣的，對不同的光之下，看到的顏色也是一樣的。

把物體上的每一個點，對不同的光產生的intensity形成一個vector，和球上的點形成的vector，比較那一個點的vector和物體的vector是最像的(表示在不同的光下形成的intensity都一樣)，他們越有可能有同樣的normal。

這個限制稱爲orientation constrain。即一個物有同樣的normal，在distant light和distant camera的假設之下，對不同的光產生的intensity變化應該要一模一樣。求得normal後用之前的方法就可求得物體的depth了。

## Helmholtz Stereo

基本原理是對於一個BRDF，把入射角L和視覺V交換，所得值和原本是一樣的，且和物體的材質無關，任何的BRDF都會成立。對一個物體交換L和V，拍兩張照片，來估計normal。有趣的地方是在這兩張照片中，occlusion會變成shadow，shadow會換成occlusion。




看到的亮度( I )和 BRDF，光的角度，光的強度有關。首先假設光的強度是 1，又利用上述的原理，所以我們可得

$I_L * V_R = I_R * V_L$ ($V_R$ , $V_L$是光照方向和normal的夾角)

$I_L$ ， $I_R$是已知的，normal是未知的。

所以我們要去猜深度，P 是一個深度的 function，所以下面整個 matrix 也是 p 的 function。想辦法猜不同的深度，那個 matrix 的 rank 越接近 2 時，就是越好的深度。而 matrix 的 null space 就是我們要找的 normal。

$$\left( i_l \frac{\hat{\mathbf{v}}_l}{|\mathbf{o}_l - \mathbf{p}|^2} - i_r \frac{\hat{\mathbf{v}}_r}{|\mathbf{o}_r - \mathbf{p}|^2} \right) \cdot \hat{\mathbf{n}} = \mathbf{w}(d) \cdot \hat{\mathbf{n}} = 0 \quad \mathbf{W}(d^\star) \text{ will be rank 2}$$