



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

Métodos Numéricos
2do Cuatrimestre - 2017

Integrante	LU	Correo electrónico
Jonathan Seijo	592/15	jon.seijo@gmail.com
Lucas De Bortoli	736/15	lu_cas_.97@hotmail.com.ar
COMPLETAR	COMPLETAR	COMPLETAR
COMPLETAR	COMPLETAR	COMPLETAR



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

Índice

1. Introducción	3
2. Calibración	4
3. Cálculo de normales	6
3.1. Eliminación Gaussiana	6
3.2. Factorización LU	7
4. Estimación de profundidades	8
4.1. Cholesky	10
5. Resultados	11
6. Experimentación	11
7. Discusión/Conclusiones	12

1. Introducción

Este trabajo consiste en la digitalización de objetos 3D basándose en imágenes producidas con cámaras tradicionales, utilizando la técnica de *fotometría estéreo*. Mostraremos que utilizando luces provenientes de diferentes ángulos, podemos aproximar las normales a la superficie y estimar las profundidades de cada punto.

Para esto debemos resolver varios sistemas de ecuaciones lineales, los cuáles resolveremos algorítmicamente de forma matricial. Usaremos en un primer caso el método clásico de Eliminación Gaussiana, y veremos como utilizando factorización LU podemos reducir los tiempos de cómputo. Luego, utilizaremos la factorización de Cholesky.

Los experimentos []

Completar breve pantallazo a experimentos

2. Calibración

Un paso anterior al cálculo de las profundidades, es el cálculo de los vectores normales a todo punto de la superficie. Para esto, se eligen tres luces diferentes (por ejemplo si elegimos 1, 2 y 3, nuestros vectores luz serán s^1 , s^2 y s^3 respectivamente) y utilizando las intensidades ya registradas (I_i) queremos encontrar los m que son solución. Cabe aclarar que el sistema se deberá resolver para cada pixel de la imagen, por lo que ya conocemos las intensidades del pixel dado para toda imagen. Nos encontramos entonces con el primer problema, porque los vectores luz no son un dato conocido.

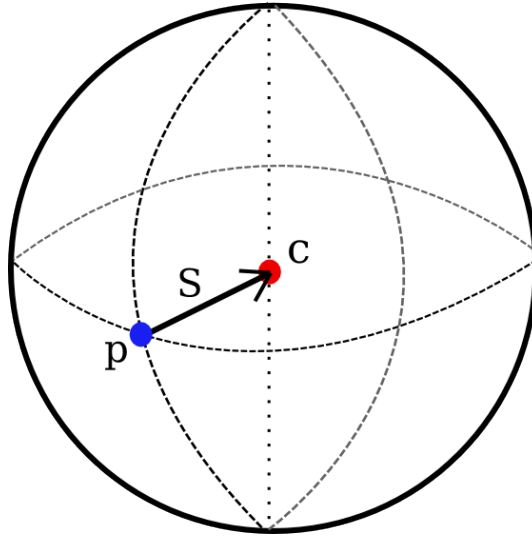
El sistema que queremos resolver es el siguiente:

$$\begin{pmatrix} s_x^1 & s_y^1 & s_z^1 \\ s_x^2 & s_y^2 & s_z^2 \\ s_x^3 & s_y^3 & s_z^3 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix}$$

Pero no conocemos los s_j^i .

Tenemos que $S = (s_x^i, s_y^i, s_z^i)$ es el vector luz en la imagen i . Dado que vamos a explicar el cálculo para una imagen cualquiera, omitiremos el supraíndice i para una notación mas relajada.

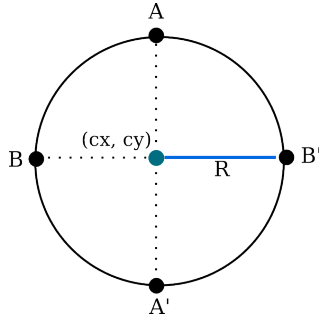
Llamemos $c = (c_x, c_y, c_z)$ al centro de la esfera. Pensemos la luz como un vector que apunta hacia el centro. El vector S toca la superficie en un cierto punto $p = (p_x, p_y, p_z)$, pero p no es un punto al azar, sino que es el punto más iluminado de la esfera. Por lo tanto, la dirección de luz que nos interesa es $S = c - p$.



Vemos que el vector que nos interesa es $S = c - p$

Mas precisamente, $S = (c_x - p_x, c_y - p_y, c_z - p_z)$. En principio no conocemos ninguno de estos valores, pero concentrémonos en calcular sobre el eje x e y . De la imagen de la esfera (2D) podemos conocer algunos datos. En la implementación utilizamos la máscara provista por la cátedra para simplificar los cálculos.

Como los píxeles en la máscara son blancos o negros, es muy sencillo identificar los puntos que pertenecen a la esfera con sólo ver su color. Recorriendo todos los puntos y tomando máximos y mínimos obtenemos 4 puntos clave: El punto del círculo que está mas a arriba A , el más abajo A' , el de más a la izquierda B y el de más a la derecha B' .



Puede verse fácilmente que el radio R del círculo es la mitad de la distancia entre B y B'

$$R = \frac{|B'_x - B_x|}{2}.$$

Además, sabiendo que c es el centro del círculo,

$$\begin{aligned} c_x &= B_x + R \\ c_y &= A_y + R \end{aligned}$$

Necesitamos encontrar también quién es p . Una primera idea fue recorrer todos los píxeles y quedarnos con el de mayor intensidad. Esto nos trajo problemas pues en una imagen no hay un único píxel mas blanco que el resto, sino que existe un pequeño sector que se encuentra más iluminado. Nos encontramos con imágenes diferentes (pero con similares intensidades de luz) para los cuales se calculaba el mismo punto p .

Para resolver esto consideramos para cada píxel una pequeña vecindad (en principio de 5×5 , pero se fueron probando otras) y nos quedamos con el píxel que tuviese la vecindad mas iluminada. Utilizando este metodo podemos hallar el $p = (p_x, p_y)$ que queríamos.

Tenemos entonces los datos de c_x , c_y , p_x , p_y y R . Veamos cómo podemos despejar lo que nos falta. Recordemos lo que queríamos calcular:

$$S = (c_x - p_x, c_y - p_y, c_z - p_z).$$

Sólo la tercer componente de S es una incógnita. Sabemos que el radio del círculo es igual al radio de la esfera y el radio de la esfera es igual a la distancia euclídea entre el centro y un punto en la superficie. En particular, el radio es igual a la distancia entre p y c . Es decir:

$$\|c - p\| = R$$

Despejando..

$$\sqrt{(c_x - p_x)^2 + (c_y - p_y)^2 + (c_z - p_z)^2} = R$$

$$(c_x - p_x)^2 + (c_y - p_y)^2 + (c_z - p_z)^2 = R^2$$

$$(c_z - p_z)^2 = R^2 - (c_x - p_x)^2 - (c_y - p_y)^2$$

$$(c_z - p_z) = \sqrt{R^2 - (c_x - p_x)^2 - (c_y - p_y)^2}$$

Finalmente conseguimos lo que buscábamos. Repitiendo este procedimiento, podemos obtener todas las componentes del vector de luz para todas las imagenes. Podemos escribir a cualquier vector de luz utilizando datos conocidos:

$$\begin{aligned} s_x &= c_x - p_x \\ s_y &= c_y - p_y \\ s_z &= \sqrt{R^2 - (c_x - p_x)^2 - (c_y - p_y)^2} \end{aligned}$$

Ya estamos en condiciones de resolver el sistema original pues conocemos todos sus coeficientes. Analizaremos en las siguientes secciones diferentes formas para resolverlo.

3. Cálculo de normales

Explicar un poco mas que representa el sistema y como resolviendolo logramos lo de las normales.
Ver que quizá esto convenga ponerlo luego de describir los metodos

3.1. Eliminación Gaussiana

Tenemos un sistema listo para resolver, dónde sólo los m_i son incógnitas:

$$\begin{pmatrix} s_x^1 & s_y^1 & s_z^1 \\ s_x^2 & s_y^2 & s_z^2 \\ s_x^3 & s_y^3 & s_z^3 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix}$$

La pregunta ahora es ¿cómo lo resolvemos?. Dado que en principio no sabemos cómo, nos gustaría llevarlo a una forma equivalente que sea mas fácil de resolver. Podemos hacer esta conversión a un sistema equivalente usando el algoritmo de eliminación de Gauss. Lo que hace este algoritmo es llevar una matriz a su forma triangular superior, de dónde luego es muy sencillo hacer los despejes finales.

El pseudocódigo del algoritmo de Gauss es el siguiente:

```

function ELIMINACIONGAUSSIANA(Matriz M[n][m])
  for  $k \in [1..min(n, m)]$  do
    for  $i \in [k + 1..m]$  do
      if  $M[k][k] \neq 0$  then
         $mult \leftarrow M[i][k] / M[k][k]$ 
        for  $j \in [k + 1..n]$  do
           $M[i][j] \leftarrow M[i][j] - mult * M[k][j]$ 
      else
        Hay un cero en la diagonal!

```

Como puede verse, funciona correctamente solo **suponiendo que no hay ceros en la diagonal**. Es claro que puede modificarse para que realice intercambios de filas y no tenga el problema del cero, veamos que para nuestro problema no es importante. En nuestra implementación aplicaremos el algoritmo de Gauss en la siguiente matriz ampliada:

$$\left(\begin{array}{ccc|c} s_x^1 & s_y^1 & s_z^1 & I_1 \\ s_x^2 & s_y^2 & s_z^2 & I_2 \\ s_x^3 & s_y^3 & s_z^3 & I_3 \end{array} \right)$$

Para empezar, nuestros s_j^i inicialmente son todos distintos de cero, así que nunca habrá un cero en la primer fila. Dado que son solo 12 vectores de luces, tomamos todas las posibles combinaciones de tres luces y corrimos el algoritmo de gauss sin pivoteos. En ningún caso se realizó división por cero ni tampoco apareció ningún cero en la diagonal. (El código de lo realizado puede encontrarse en *TestTieneLU.cpp*). Esto cobrará importancia cuando querramos encontrar factorización LU.

Dado que pudimos triangular correctamente la matriz ampliada, entonces ya estamos en condiciones de despejar de nuestra matriz de 3 x 4:

$$\left(\begin{array}{ccc|c} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ 0 & m_{2,2} & m_{2,3} & m_{2,4} \\ 0 & 0 & m_{3,3} & m_{3,4} \end{array} \right)$$

```
function DESPEJAR(Matriz M[n][m])
    // En X se guardan los m-1 coeficientes solución (Recordemos que M es ampliada)
    X[m-1] ← {}

    for j ∈ [1..m-1] do (j es índice de columna)
        if M[j][j] ≠ 0 then
            X[j] ← M[j][m] / M[j][j]
            for i ∈ [j-1,0] do (i es índice de fila)
                M[i][m] ← M[i][m] - ( M[i][j] * X[j] )
        else
            Hay un cero en la diagonal!
    Retornar X
```

3.2. Factorización LU

Recordemos nuestro sistema para hallar las normales:

$$\begin{pmatrix} s_x^1 & s_y^1 & s_z^1 \\ s_x^2 & s_y^2 & s_z^2 \\ s_x^3 & s_y^3 & s_z^3 \end{pmatrix} \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix}$$

Una vez fijas las luces a utilizar (en este caso 1,2 y 3) para despejar la normal en cada píxel, debemos resolver el sistema en cada píxel. Es decir, estaremos triangulando una y otra vez una matriz dónde lo único que cambia es el término a la derecha de la igualdad. Por lo tanto, es interesante plantearse si existe una forma de evitar aplicar Gauss en cada punto.

La factorización LU podría no existir, pero veámos de que se trata. Dada una matriz A , la factorización LU consiste en encontrar dos matrices: una matriz L triangular inferior con unos en la diagonal y una matriz U triangular superior, de forma que se cumpla

$$A = L.U$$

Por lo visto en clase, puede demostrarse que la L tiene en la diagonal unos, ceros por arriba, y por debajo los multiplicadores que se utilizaron en la eliminación Gaussiana para colocar un cero en la triangulación. En la U se colocan ceros debajo de la diagonal y en el resto los coeficientes que quedaron en la matriz ya triangulada.

Digamos entonces que ya conocemos la factorización LU para una matriz dada, ¿Cómo la utilizamos para resolver nuestro sistema?

$$Ax = b \iff LUx = b$$

Si consideramos $Ux = y$, nos queda para resolver:

$$Ly = b$$

Donde L es triangular inferior. Por lo tanto podemos despejar y obtener y sin necesidad de aplicar eliminación Gaussiana. Una vez que conocemos y , como U también esta triangulada despejamos en:

$$Ux = y$$

Obteniendo así el x que queríamos encontrar inicialmente.

Por lo expuesto en la sección anterior, experimentalmente comprobamos que en nuestra matriz de luces podemos aplicar Gauss normalmente sin encontrarnos con ceros en la diagonal y sin tener que hacer

ninguna permutación de filas.

Por lo visto en la clase teórica, si podemos triangular una matriz usando Gauss sin tener que permutar filas, es suficiente para afirmar que la factorización LU existe, entonces con el procedimiento explicado podemos hallar la descomposicion de nuestra matriz de luces y resolver el sistema mas eficientemente.

4. Estimacion de profundidades

Utilizando los métodos anteriores pudimos resolver el sistema que incluye las luces y calcular las normales para todo píxel de la imagen. Recordemos que para un cierto píxel (a, b) la normal en ese punto es de la forma:

$$n^{(a,b)} = (n_x^{a,b}, n_y^{a,b}, n_z^{a,b})$$

A partir de aquí en ocasiones omitiremos el supraíndice (a, b) para relajar la notación cuando es claro cuál es el píxel del que hablamos. Siguiendo con la técnica de fotometría estéreo, el siguiente paso a realizar es el cálculo de las profundidades utilizando estas normales. Para esto, consideramos una aproximación al plano tangente de cada píxel. Las ecuaciones que tenemos que resolver son las siguientes, para cada píxel (x, y) .

$$\begin{cases} n_y + n_z * (z_{x,y+1} - z_{x,y}) = 0 \\ n_x + n_z * (z_{x+1,y} - z_{x,y}) = 0 \end{cases}$$

O equivalentemente

$$\begin{cases} n_z * z_{x,y+1} - n_z * z_{x,y} = n_y \\ n_z * z_{x+1,y} - n_z * z_{x,y} = n_x \end{cases}$$

Consideremos como sería el sistema matricial para una imagen de 2x3 pixeles, y veremos como puede generalizarse:

$$\underbrace{\begin{pmatrix} -n_z^{1,1} & 0 & 0 & n_z^{1,1} & 0 & 0 \\ -n_z^{1,1} & n_z^{1,1} & 0 & 0 & 0 & 0 \\ 0 & -n_z^{1,2} & 0 & 0 & n_z^{1,2} & 0 \\ 0 & -n_z^{1,2} & n_z^{1,2} & 0 & 0 & 0 \\ 0 & 0 & -n_z^{1,3} & 0 & 0 & n_z^{1,3} \\ 0 & 0 & -n_z^{1,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & -n_z^{2,1} & 0 & 0 \\ 0 & 0 & 0 & -n_z^{2,1} & n_z^{2,1} & 0 \\ 0 & 0 & 0 & 0 & -n_z^{2,2} & 0 \\ 0 & 0 & 0 & 0 & -n_z^{2,2} & n_z^{2,2} \\ 0 & 0 & 0 & 0 & 0 & -n_z^{2,3} \\ 0 & 0 & 0 & 0 & 0 & -n_z^{2,3} \end{pmatrix}}_{\mathbf{M}} \underbrace{\begin{pmatrix} z_{1,1} \\ z_{1,2} \\ z_{1,3} \\ z_{2,1} \\ z_{2,2} \\ z_{2,3} \end{pmatrix}}_{\mathbf{Z}} = \underbrace{\begin{pmatrix} n_y^{1,1} \\ n_x^{1,1} \\ n_y^{1,2} \\ n_x^{1,2} \\ n_y^{1,3} \\ n_x^{1,3} \\ n_x^{2,1} \\ n_y^{2,1} \\ n_x^{2,2} \\ n_y^{2,2} \\ n_x^{2,3} \\ n_y^{2,3} \end{pmatrix}}_{\mathbf{V}}$$

A modo de ejemplo, tomemos la tercer fila y hagamos el producto:

$$-n_z^{1,2} * z_{1,2} + n_z^{1,2} * z_{1,3} \iff n_z^{1,2} * z_{1,3} - n_z^{1,2} * z_{1,2}$$

y vemos que se corresponde con nuestro sistema original. Puede verse además que las dimensiones para realizar el producto cuadran perfectamente. Si n' y m' eran el alto y ancho de la imagen original, la nueva matriz tiene $2 * n' * m'$ filas y $n' * m'$ columnas.

Si bien la matriz esta planteada con unas dimensiones en particular, por su forma es sencillo de generalizar. Dado un cierto píxel (x, y) , las dos ecuaciones correspondientes son:

$$\begin{pmatrix} 0 & \dots & 0 & n_z & 0 & \dots & n_z & \dots \\ 0 & \dots & 0 & n_z & n_z & 0 & \dots & \dots \end{pmatrix}$$

Para cada píxel tenemos la primer ecuacion correspondiente a n_y , donde los dos n_z están separados a m' de distancia (*ancho de la imagen original*), y en la segunda ecuación se encuentra la correspondiente a n_x donde los dos n_z se encuentran juntos. También puede verse que cada vez que pasamos al siguiente píxel bajando de fila se produce un corrimiento en una columna hacia la derecha. Hay que tener especial cuidado con los bordes de la imagen, porque el producto dará como resultado una ecuación errónea. Para que esto no sea un problema, colocamos ceros en los lugares problemáticos.

Queremos entonces resolver el sistema:

$$Mz = v$$

Multiplicando por M^t a ambos lados se obtiene:

$$\underbrace{M^t M}_A z = \underbrace{M^t v}_b$$

Para así llegar al sistema final que nos interesa resolver

$$Az = b$$

La matriz A no es cualquier cosa, sino que tiene una forma particular. En primer lugar, es simétrica porque es el resultado de haber multiplicado una matriz con su traspuesta.

En este caso si (a, b) es nuestro píxel escribiremos $n_{a,b}$ en vez de $n^{a,b}$ para no confundir con el exponente que está potenciando al elemento. Veamos como es la forma de A , obtenida simplemente haciendo la cuenta $M^t M$

$$\begin{pmatrix} \underline{2n_{1,1}^2} & \underline{-n_{1,1}^2} & 0 & \underline{-n_{1,1}^2} & 0 & 0 \\ \underline{-n_{1,1}^2} & \underline{n_{1,1}^2 + 2n_{1,2}^2} & \underline{-n_{1,2}^2} & 0 & \underline{-n_{1,2}^2} & 0 \\ 0 & \underline{-n_{1,2}^2} & \underline{n_{1,2}^2 + 2n_{1,3}^2} & 0 & 0 & -n_{1,3}^2 \\ \underline{-n_{1,1}^2} & 0 & 0 & \underline{n_{1,1}^2 + 2n_{2,1}^2} & -n_{2,1}^2 & 0 \\ 0 & \underline{-n_{1,2}^2} & 0 & -n_{2,1}^2 & \underline{n_{1,2}^2 + n_{2,1}^2 + 2n_{2,2}^2} & -n_{2,2}^2 \\ 0 & 0 & -n_{1,3}^2 & 0 & -n_{2,2}^2 & \underline{n_{1,3}^2 + n_{2,2}^2 + 2n_{2,3}^2} \end{pmatrix}$$

Es fácil darse cuenta del patrón. Vamos tomando p el pixel actual, pensando en los píxeles ordenados $((1, 1), (1, 2), (1, 3), \dots)$. Colocamos $2p^2$ en la diagonal. Colocamos $-p^2$ una celda a la derecha, m' celdas a la derecha, una celda abajo, m' celdas abajo. Sumamos p^2 una celda siguiente sobre la diagonal y en la celda m' siguiente sobre la diagonal. El único cuidado es cuando llegamos al borde de la imagen, de no colocar ese píxel en el borde inmediato.

Si bien este fue un pequeño ejemplo, la matriz en caso general contiene **muchos** ceros. Originalmente nuestras matrices estaban hechas utilizando simples vectores. Esto es un problema **enorme** para la implementación, porque si tomamos una imagen mas grande, por ejemplo de $250 * 270$ px (como el tamaño de las normales provistas por la cátedra) la cantidad de elementos en la matriz será de

$$(250 * 270)^2 = 4556250000$$

Y dado que un *double* ocupa 8 bytes, el tamaño total de la matriz sería

$$4556250000 * 8bytes = 36,45gigabytes$$

Como no pudimos conseguir esa cantidad de memoria ni juntando a todos los miembros del equipo, decidimos hacer algo mejor.

Aprovechándonos de la gran cantidad de ceros en la matriz, implementamos cada fila de la matriz con la estructura *map*. Cada fila de la matriz era del tipo *map<int, double>* donde el *int* es utilizado para representar el número de columna del elemento, y el *double* es el elemento en cuestión. Los elementos que no se encuentran en el *map* los asumimos como que valen 0.

Utilizando esta idea redujimos ampliamente el tamaño que nos ocupa la matriz en memoria y solucionamos nuestro problema original, **pero no es gratis**: el acceso a un elemento de la matriz ya no será en tiempo constante, sino logarítmico en la cantidad de elementos de la fila. Como tenemos *muy pocos* elementos distintos de cero en cada fila, estamos dispuestos a pagar este precio. Sin embargo, cuando accedemos a un elemento en general lo hacemos porque estamos recorriendo la matriz, por lo que para recorrer una fila (anteriormente enorme) accedemos a poquitos elementos, compensando ampliamente.

4.1. Cholesky

Dada una matriz M cualquiera, la factorización de Cholesky consiste en encontrar una matriz triangular inferior L (no necesariamente todos unos en la diagonal) de forma tal que:

$$M = LL^t$$

Es importante aclarar que **no toda matriz tiene descomposición de Cholesky**. La ventaja que esto es que si tenemos que resolver un sistema podemos aprovechar que conocemos una descomposición para resolver el sistema sin necesidad de triangular, aprovechando que L es triangular inferior y L^t es triangular superior.

$$Mx = b \iff L \underbrace{L^t x}_y = b \iff \begin{cases} Ly = b \\ L^t x = y \end{cases}$$

A diferencia de la descomposición LU, no es necesario que realicemos el método de eliminación gaussiano para conseguir nuestra L , sino que existe un algoritmo que calcula la L basandose en realizar operaciones sobre los elementos de la matriz original.

Si hay tiempo incluir un pseudo simple de como funciona cholesky

Idea para la demo, no hablar de simetrica positiva sino de que plantear que la cholesky existe ..

MINGA ES TODO FRUTA NO DEMOSTRE UN CARAJO

Como nuestra matriz A es simétrica y definida positiva puede verse que utilizar la factorización de Cholesky es una buena idea, ya que tiene constantes mas baja que la eliminación gaussiana por lo que estaríamos reduciendo cálculos. Además, como nuestras matrices ya no son *vectores de vectores*, sino *maps*, no es trivial la implementación de la eliminación gaussiana para nuestro nuevo tipo. Sin embargo, sumar y multiplicar elementos es algo que sabemos hacer muy bien.

Continuar

5. Resultados

Dar imagenes minimo del resultado final

6. Experimentación

OBLIGATORIO ES LO SIGUIENTE, POR ENUNCIADO

2. Como afecta la calibracion del sistema en el resto de las etapas
3. ¿Como impacta la eleccion de las 3 direcciones de iluminacion para el calculo de las normales?

Experimentacion general pensar mas:

Diferencias de tiempos EG vs LU, con si y mascara mascara

Mostrar como con diferentes luces obtenemos diferentes normales

Mostrar resultados finales con mismas luces pero propias vs catedra

1. Comparar las direcciones de iluminacion obtenidas por el metodo de calibracion con las provistas por la catedra

7. Discusión/Conclusiones