



DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

Métodos Númericos
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
Roberto Grings	345/08	robertoegrings@gmail.com
Agustín Penas	668/14	agustinpennas@gmail.com



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. Estimacion de profundidades	9
4.1. Cholesky	12
5. Resultados finales y experimentación	14
5.1. Resultados	14
5.2. Calibración de luces	19
5.3. Normales	20
5.4. Eliminacion gaussiana y Factorización LU	22
5.5. Cholesky	23
6. Conclusión	25

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.

En primer lugar resolveremos el problema de la calibración, es decir, encontraremos a partir de imágenes de una esfera cuales son los diferentes vectores de iluminación. Utilizaremos estos datos para calcular las normales a la superficie en cada punto. Para esto nos enfrentaremos con varios sistemas de ecuaciones lineales, los cuales resolveremos algorítmicamente de forma matricial. Usaremos en un primer caso el método clásico de Eliminación Gaussiana, y nos aprovecharemos de la estructura de nuestro problema para encontrar una factorización LU e intentar reducir los tiempos de cómputo.

En la última instancia, para el cálculo de profundidades, aprovecharemos la forma de la matriz final para crear una nueva estructura mas eficiente, donde aplicaremos el algoritmo de Cholesky para resolver el sistema.

2. Calibración

El 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, donde m es un vector que depende del vector normal y de una constante del objeto. 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 $S = c - p$.

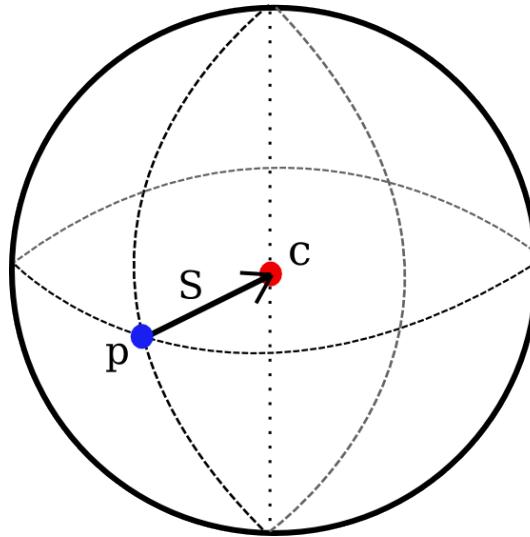
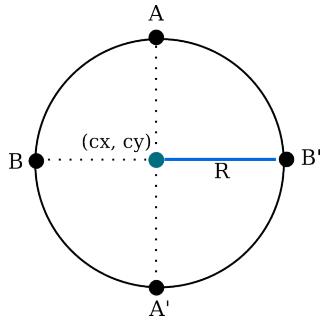


Figura 1: Vemos que el vector que nos interesa es $S = c - p$

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 más 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 más iluminada. Utilizando este método 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 tercera 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 imágenes. 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

En esta sección resolveremos el problema de calcular los vectores normales a la superficie, conociendo los vectores luz y los valores de la intensidad para cada píxel dado. Resolveremos utilizando el algoritmo de eliminación gaussiana, y veremos en la siguiente sección una forma de optimizar los cálculos aprovechándonos de las propiedades de nuestro problema.

3.1. Eliminación Gaussiana

Para cada píxel, tenemos un sistema listo para resolver:

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

Donde cada s_c^i es la coordenada c del vector de luz en la imagen i , los I_i las intensidades de luz (del píxel actual) en la imagen i , y los m_j nuestras incógnitas. El vector $m = (m_x, m_y, m_z)$ no es exactamente el valor de la normal n , sino que $m = I_0 \rho \cdot n$, con $I_0, \rho \in \mathbb{R}$ constantes desconocidas que dependen del objeto. Lo que nos interesa es encontrar el valor de n , pero para esto primero debemos hallar m .

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 ∈ [1..min(n,m)] do
        for i ∈ [k + 1..m] do
            if M[k][k] ≠ 0 then
                mult ← M[i][k] / M[k][k]
                for j ∈ [k + 1..n] do
                    M[i][j] ← 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, pero veremos 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 ningun 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:

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

```

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

```

Resolviendo el sistema con la forma expuesta, obtenemos el vector solución m . Pero m no es lo que buscábamos, sino que queremos obtener el valor de n . Recordemos:

$$m = I_0 \rho \cdot (n_x, n_y, n_z) = I_0 \rho \cdot n$$

Con $I_0 \rho \in \mathbb{R}$. Tomando norma:

$$\|m\| = |I_0 \rho| \|n\|$$

Pero $\|n\| = 1$, pues queremos el vector unitario. Entonces:

$$\|m\| = |I_0 \rho|$$

Sabiendo esto, podemos despejar y obtener el valor de n (con $m \neq 0$):

$$n = \frac{m}{\|m\|}$$

Obtuvimos así para cada píxel vector normal a la superficie.

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 traingulando una y otra vez una matriz donde 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

$$A = L \cdot U$$

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 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 descomposición 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 cual hablamos. Siguiendo con la técnica de fotometría estéreo, el siguiente paso a realizar es el cálculo de las profundidades utilizando estas normales. Para esto, consideraremos una aproximación al plano tangente de cada píxel. La 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:

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

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 está 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 ecuación 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 erronea. 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 $M^t M$

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

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.

Toda matriz multiplicada por su traspuesta es simétrica, por lo que nuestra A también es simétrica, como corroboramos en la cuenta. Nos gustaría ver que además es definida positiva, esto nos será de utilidad en la siguiente sección cuando querremos aplicar Cholesky.

Antes de meternos a hacer cuentas, analicemos un poco como son los valores de los elementos de nuestra matriz. Para todo píxel a, b , tenemos que $n_{a,b}^2 \geq 0$ por ser números reales elevados al cuadrado. Recodemos que los n que aparecen en la matriz A son en realidad la componente z del vector normal n .

Veamos que pasa en la ecuación original si para algún x, y tenemos que $n_z = 0$.

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

Pero eso no puede pasar en ningún punto de la imagen, porque n fue elegido para que $\|n\| = 1$.

Llevando la cuenta a nuestro problema, podrían aparecer $n_z = 0$ pero solo para aquellos píxeles que están fuera de la imagen, los píxeles sin profundidad. Esto podemos saberlo porque tenemos la máscara que nos permite conocer cuáles píxeles pertenecen a la imagen y cuáles no. Por lo tanto, podemos construir nuestra matriz A utilizando $n_{x,y} > 0$ para todo x, y perteneciente a la imagen.

Calculemos determinantes :)

$$\det A_{11} = 2n_{1,1}^2 > 0$$

$$\begin{aligned} \det A_{22} &= 2n_{1,1}^2(n_{1,1}^2 + 2n_{1,2}^2) - ((-n_{1,1}^2)).(-n_{1,1}^2)) \\ &= 2n_{1,1}^4 + 4n_{1,1}^2n_{1,2}^2 - n_{1,1}^2 \\ &= n_{1,1}^4 + 4n_{1,1}^2n_{1,2}^2 > 0 \end{aligned}$$

$$\begin{aligned} \det A_{33} &= -(-n_{1,2}^2).(2n_{1,1}^2.(-n_{1,2}^2)) + (n_{1,2}^2 + 2n_{1,3}^2). \det A_{22} \\ &= -2n_{1,1}^4n_{1,2}^2 + (n_{1,2}^2 + 2n_{1,3}^2).(n_{1,1}^4 + 4n_{1,1}^2n_{1,2}^2) \\ &= -2n_{1,1}^4n_{1,2}^2 + n_{1,1}^4n_{1,2}^2 + 4n_{1,2}^4n_{1,2}^2 + 2n_{1,3}^2n_{1,1}^2 + 8n_{1,1}^2n_{1,2}^2n_{1,3}^2 \\ &= n_{1,1}^4n_{1,2}^2 + 2n_{1,1}^4n_{1,2}^2 + 2n_{1,3}^2n_{1,1}^2 + 8n_{1,1}^2n_{1,2}^2n_{1,3}^2 > 0 \end{aligned}$$

$$\begin{aligned} \det A_{44} &= n_{1,1}^2.(-n_{1,1}^2.((n_{1,1}^2 + 2n_{1,2}^2).(n_{1,2}^2 + 2n_{1,3}^2) - (n_{1,2}^2.n_{1,2}^2))) + (n_{1,1}^2 + 2n_{2,1}^2). \det A_{33} \\ &= .. \\ &= n_{1,1}^4n_{1,2}^4 + 4n_{1,1}^4n_{1,2}^2n_{1,3}^2 + 2n_{2,1}^2n_{1,1}^2n_{1,1}^4 + 4n_{2,1}^2n_{1,2}^4n_{1,1}^2 + 4n_{2,1}^2n_{1,3}^2n_{1,1}^4 + 16n_{1,1}^2n_{1,2}^2n_{1,3}^2n_{2,1}^2 \\ &> 0 \end{aligned}$$

...

Si bien sabemos que lo expuesto arriba no es una demostración formal, podemos asegurar que los determinantes de las submatrices principales son siempre > 0 , ya que la matriz respeta siempre el mismo patrón. Por lo visto en clase, sabemos que si los menores principales de una matriz son positivos la matriz es definida positiva. Por lo tanto, *A es simétrica definida positiva*.

Aunque lo que mostramos 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

$$4556250000 * 8\text{bytes} = 36,45\text{gigabytes}$$

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 es 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 mas que dispuestos a pagar este precio. Además, cuando accedemos a un elemento en general lo hacemos porque estamos recorriendo la matriz, por lo que para recorrer una fila (anteriormente enorme) ahora 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 eliminacion 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.

Como nuestra matriz A es simétrica y definida positiva (*por lo analizado en la sección anterior*) 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.

Es importante aclarar que si bien no lo resolvimos con Gauss, la resolución en teoría es perfectamente posible. Como esta matriz es definida positiva, tiene determinante mayor a cero, por lo que es inversible. Entonces podríamos aplicar Gauss sin dejar ceros en la diagonal.

5. Resultados finales y experimentación

5.1. Resultados

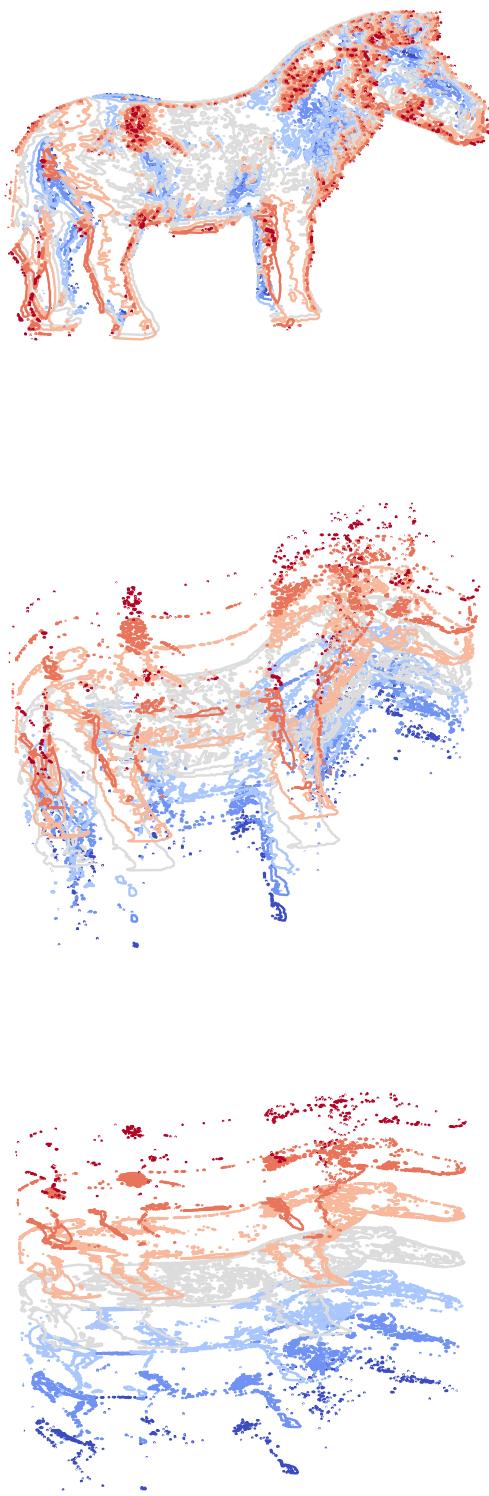


Figura 2: Curvas de nivel de la función de profundidad (para cada píxel, su profundidad estimada) Los valores fueron calculados utilizando las luces que obtuvimos en nuestra calibración.

Si bien esperábamos obtener una superficie mucho mas *suave*, creemos que los resultados fueron satisfactorios. Puede observarse perfectamente en los gráficos de curvas de nivel de arriba que en cada píxel la profundidad estimada es adecuada y se corresponde con la iluminación de las fotos originales. Veamos una representación de la función de profundidad con los mismos datos en un modelado 3D:

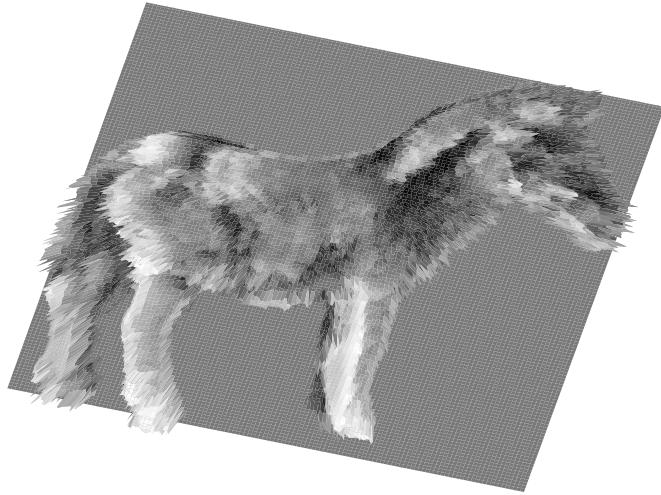
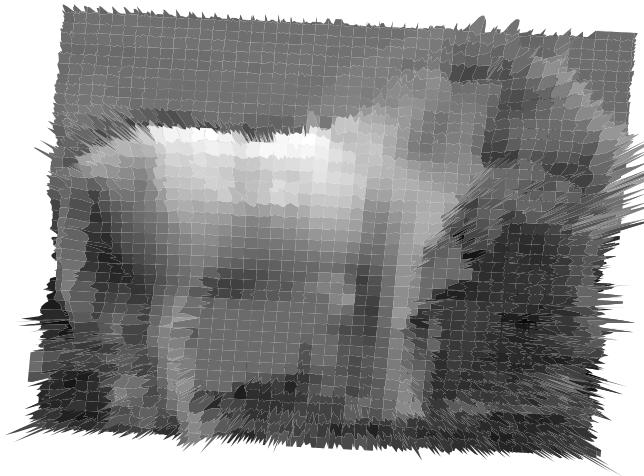


Figura 3: Profundidades calculadas con luces propias, 5, 7, 8

En la imagen de arriba podemos ver lo que mencionamos sobre la *suavidad* de la superficie. Es claro que es diferente a lo visto en la imagen de ejemplo del enunciado, pero incluso así son distinguibles las alturas de los diferentes píxeles. El *fondo* del modelo es un plano perfecto por el hecho de haber utilizado la máscara para no realizar cálculos innecesarios. Sospechamos que los picos tiene que ver con la forma en que aproximamos los planos tangentes.

Veamos que obtenemos si partimos directamente de las **normales** provistas por la cátedra:



Si bien las luces con las que fueron obtenidas las normales de la cátedra son desconocidas para nosotros, podemos hacer algunas comparaciones. Aunque esté llena de *picos*, sobretodo en los bordes, parece ser una superficie un poco mas suave que la obtenida por nosotros. Dado que no sabemos cuales fueron las luces utilizadas en el cálculo, no podemos descartar que sea un tema de elección de luces. Sin embargo, incluso aunque fuesen las mismas, veremos mas adelante que lo calibrado por nosotros no se corresponde en un 100 % por lo que es esperable que se observen diferencias.

Otra cosa a notar, es que en ambas los detalles del rostro del caballo se pierden. Si bien era esperable ya que son áreas delicadas, nos dá la pauta de que nuestras aproximaciones pueden no ser del todo correctas. De todos modos, no podemos saber si en una estimación 'bien hecha' los detalles se mantienen o no.

Nos gustaría ver qué sucede cuando utilizamos otras luces sobre la misma figura y así observar, por ejemplo, si obtenemos la misma cantidad de picos. Además, es importante notar que para obtener las intensidades de iluminosidad estábamos utilizando el promedio de los colores, así que aprovecharemos la siguiente prueba para ver qué diferencias podemos encontrar si consideramos alguna componente en particular. No esperamos obtener diferencias significativas entre tomar el promedio o una sola componente del color.

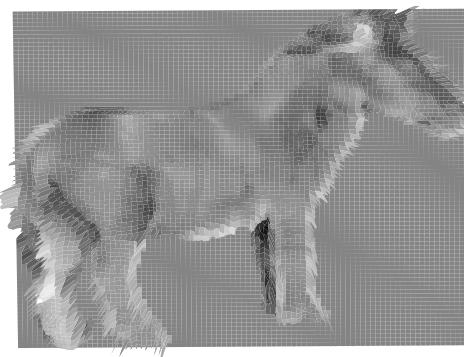
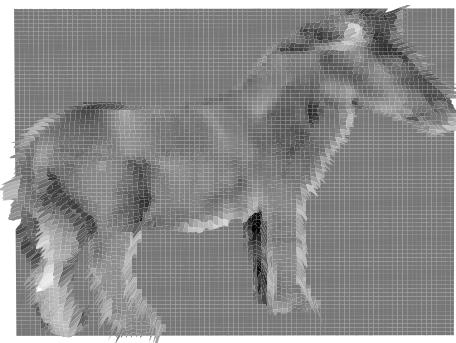
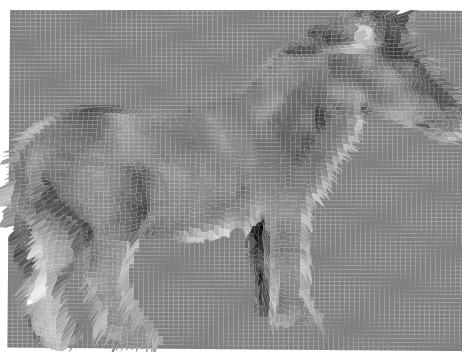


Figura 4: ↑ Luces 0,1,2, promedio de colores

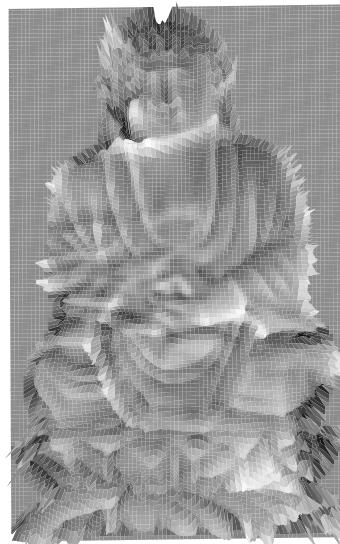


Luces 0,1,2, componente azul (izq), componente roja (der)

Cómo sospechábamos, no hay ninguna diferencia apreciable en tomar las imágenes con diferentes componentes de color. Pensamos que una posibilidad es que la nula diferencia se deba al color del objeto original.

Es interesante notar que utilizando estas combinaciones de luces obtenemos diferentes resultados que la usando la primer combinación. En la primera había muchos picos y la diferencia entre blancos y negros era muy pronunciada. Con esta nueva combinación, los colores visibles son mucho más uniformes y la cantidad de picos disminuyó, aunque siguen manteniéndose los picos en los bordes. Siguen sin verse correctamente los detalles en el rostro del caballo.

Realizamos los cálculos de profundidad con una imagen que tiene más detalles, para ver si sucede lo mismo que con el rostro del caballo. La siguiente es un gráfico con las profundidades de Buda, con las luces 0, 1 y 2 calibradas por nosotros.



Los detalles del cuerpo se mantuvieron bastante bien, y el área central e inferior tiene más suavidad que la que habíamos logrado con la anterior imagen. En los bordes nuevamente tenemos picos, se deben a la diferencia brusca que hay con el plano. Es interesante notar que se produjo una deformación en el

sector superior, creemos que tiene que ver con la gran cantidad de cambios de luz en un área tan pequeña.

5.2. Calibración de luces

Mencionamos anteriormente que se encontraron diferencias entre las luces calibradas por nosotros y las luces de la cátedra. Veamos los resultados que obtuvimos. En general, ninguna luz quedó *exactamente* igual, pero sí quedaron similares.

Dado que es complicado visualizar diferencias entre vectores tridimensionales, analizaremos por un lado el eje z y por el otro los ejes x e y . En el siguiente gráfico tomamos para cada luz, los ejes z y calculamos su diferencia.

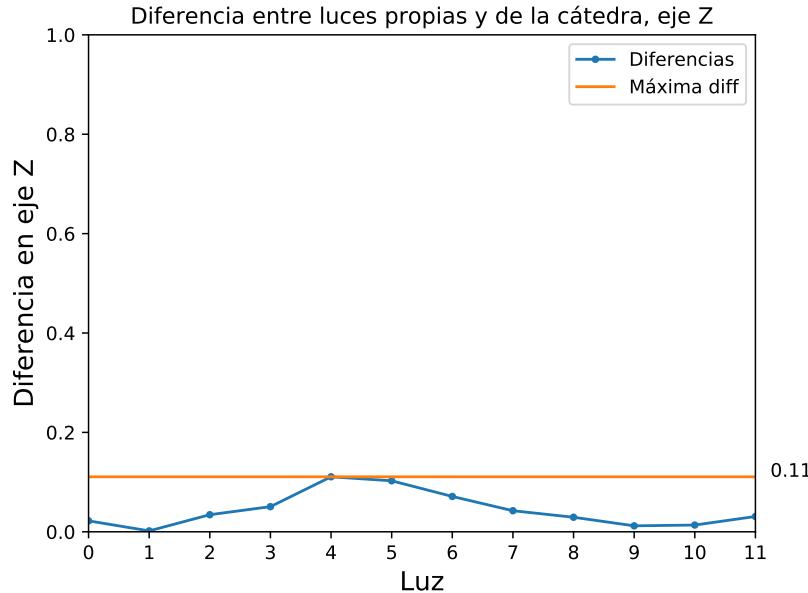
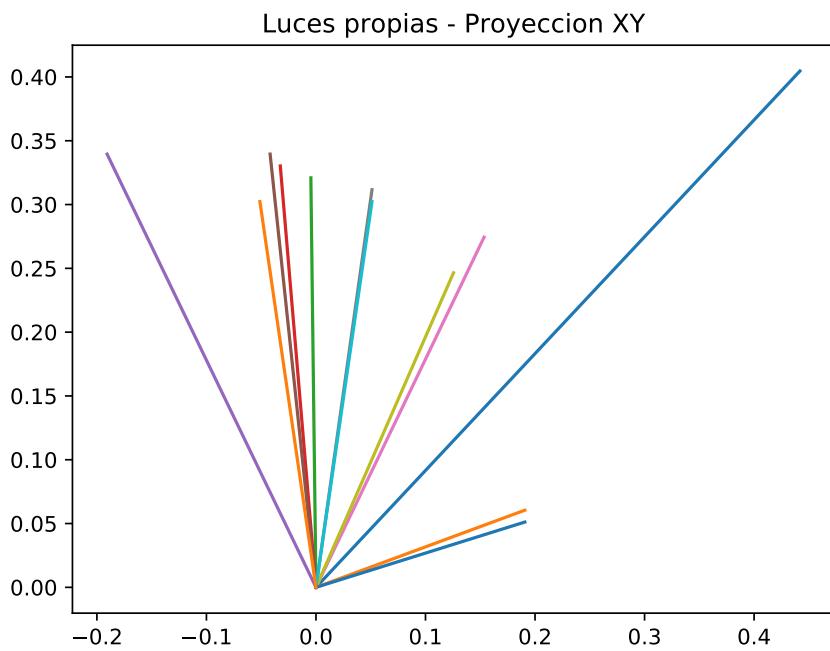
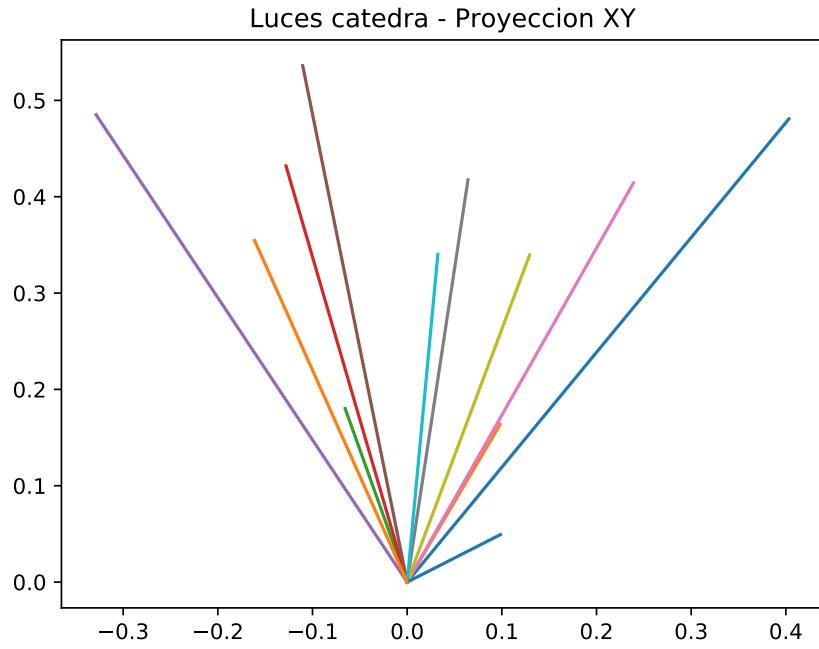


Figura 5

Podemos observar que la diferencia máxima es aproximadamente 0,1 mientras que la diferencia mínima está cercana a cero (es exactamente 0,001821 en la luz 1). Creemos que si bien no es perfecto, es una diferencia aceptable. Dado que todos los vectores son unitarios (pues fueron normalizados) es lógico esperar diferencias entre los ejes x e y . A continuación puede verse el gráfico resultante. Las luces que se corresponden con las propias y las de la cátedra están señaladas con el mismo color.

Podemos observar que si bien las diferencias son notorias, todas se encuentran en el mismo rango en inclinación. No hay ninguna que haga cosas extrañas, como por ejemplo apuntar en sentido inverso. Creemos entonces que las diferencias no afectaron demasiado el resultado final.



5.3. Normales

Dado que las luces son utilizadas para el cálculo de las normales, queremos ver cómo nos afecta la calibración en esta etapa. Para eso, resolveremos el sistema de ecuaciones y guardaremos los vectores normales obtenidos. Como las mayores diferencias entre luces podían verse en los ejes x, y estos ejes serán los que usaremos en la comparación.

En los gráficos que se encuentran a continuación, para cada píxel se grafica un vector que corresponde a la normal en ese punto. Como son *miles* de normales, puede dar la impresión de que se grafican puntos pero no es así: son los vectores en forma de 'flechas'.

En todos los gráficos se utiliza exactamente la misma área de la imagen para que sean comparables. Para la resolución del sistema que calcula las normales se ultiliza eliminación gaussiana con pivoteo parcial para tratar de minimizar el error numérico. Para el primer experimento tomaremos nuestra 'mejor luz', la 1 junto con las luces 4 y 5 que parecen ser las 'peores' (Consideramos como mejor a las luces que más se acercan a las provistas por la catedra, osea las que tienen menor diferencia en el grafico de la Figura 5). Esperábamos obtener malos resultados ya que dos luces no eran buenas, sin embargo los resultados fueron bastante buenos.

Las diferencias mas notorias se encuentran en el ojo y en la oreja del caballo. Creemos que tiene que ver porque son las áreas con más detalle pero menor iluminación. Sin embargo, no creemos que sea una diferencia demasiado significativa.

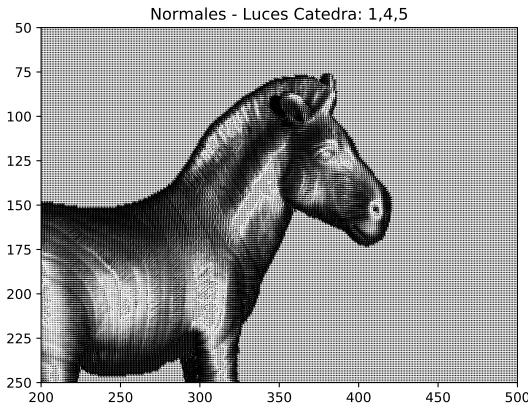


Figura 6: Normales con luces catedra: 1, 4, 5

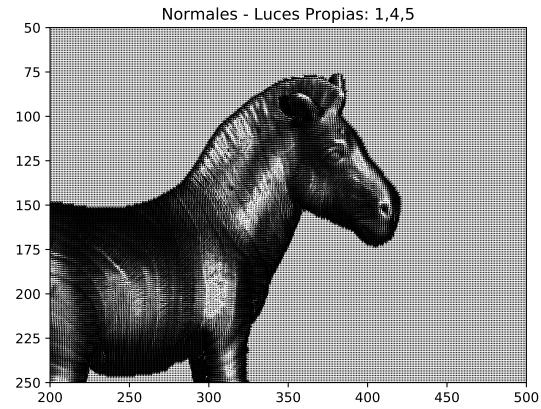


Figura 7: Normales con luces propias: 1, 4, 5

Aprovecharemos las siguientes imágenes podremos observar dos cosas. La primera es ver un poco más sobre las diferencias entre las luces de la cátedra y las calculadas por nosotros, y lo segundo es como cambian las normales cuando se toma un set de luces diferentes.

En el siguiente caso no tomamos la luz número 1, que era la mejor que teníamos y en su lugar tomamos otra de las 'malas'. En este caso, consideraremos las luces 4,5 y 6. Podemos observar que tanto las de la cátedra como las propias son demasiado oscuras, sin embargo la nuestra lo es mucho más. Los rasgos faciales dejaron de apreciarse.

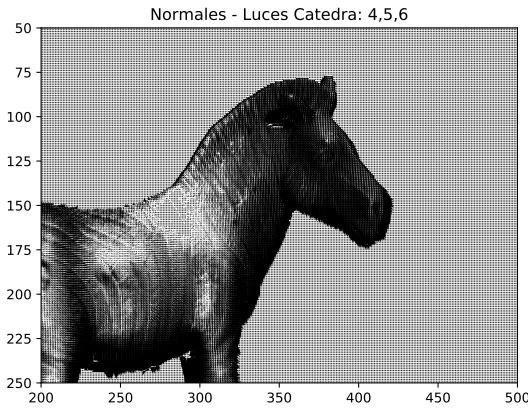


Figura 8: Normales con luces catedra: 4, 5, 6

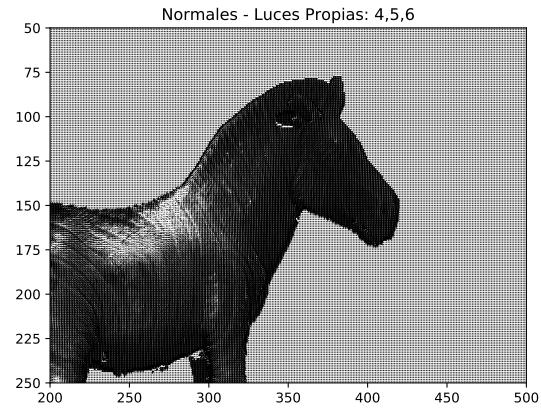


Figura 9: Normales con luces propias: 4, 5, 6

Al haber tomado otro set de luces, la forma general de la imagen no cambia demasiado, pero si hay

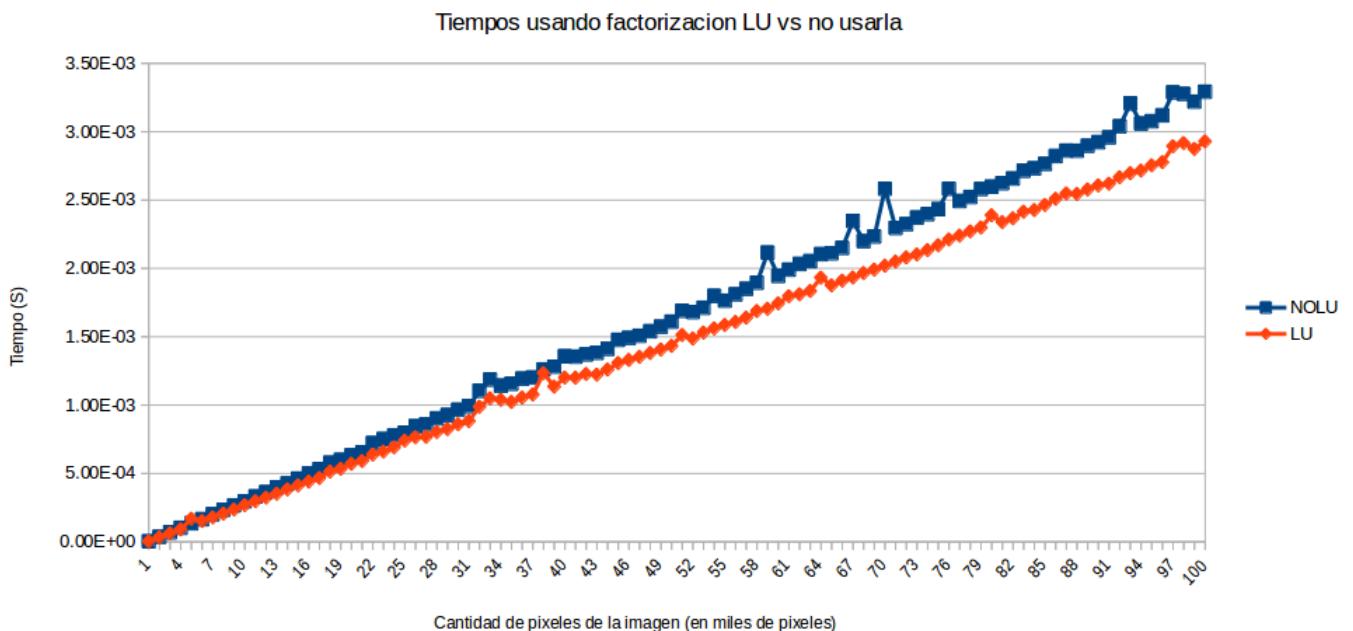
bastante diferencia entre los rangos mas finos.

5.4. Eliminacion gaussiana y Factorización LU

Lo que queremos lograr con esta experimentación es comparar el resultado del cálculo sobre todos los pixeles usando el método de eliminación gaussiana vs factorizar la matriz y resolver los dos sistemas triangulados. Para esto tomamos una matriz cualquiera de las generadas por 3 luces y resolvimos el sistema para N términos independientes distintos generados al azar. Esto simula el correr el algoritmo sobre una imagen de N pixeles.

Medimos únicamente el tiempo que tarda en resolver el sistema para cada píxel, dejando de lado todo cálculo extra por afuera de la medición (como por ejemplo la generación aleatoria de los términos independientes). Luego sumamos cada uno de estos tiempos como la medición del experimento de tamaño N. Hicimos 100 mediciones para cada cantidad de pixeles y luego dividimos por esa cantidad para sacar el promedio, de esta manera reducimos el ruido que pudo haberse visto durante la medición.

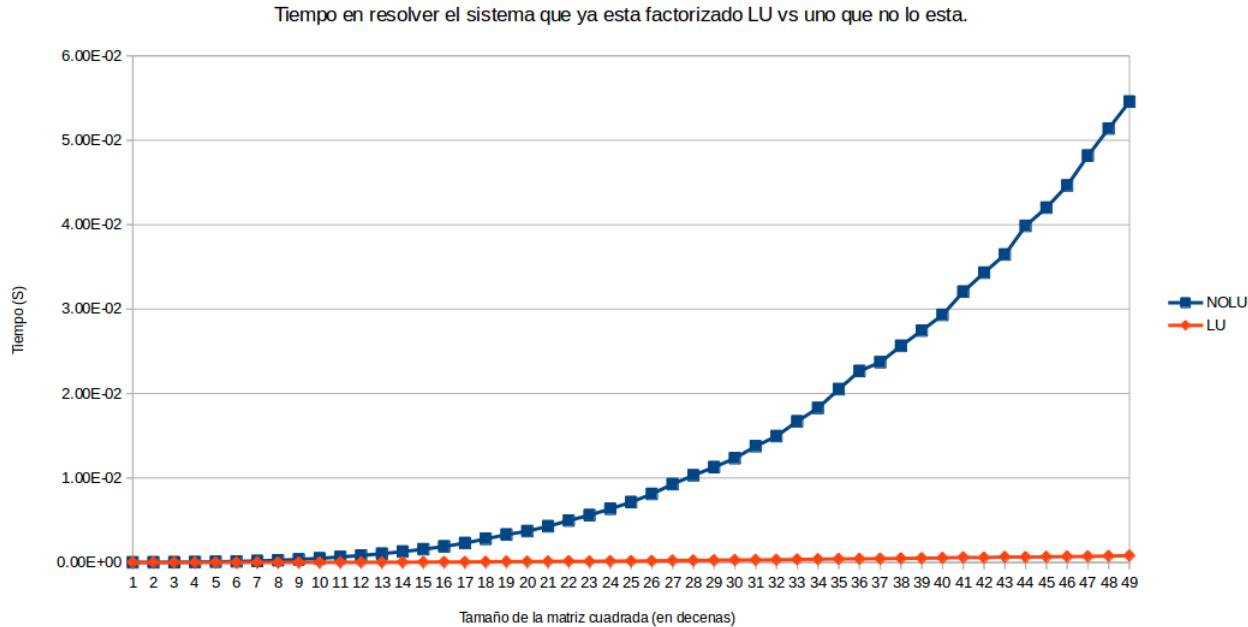
Hicimos varias mediciones variando la cantidad de pixeles y estos fueron los resultados:



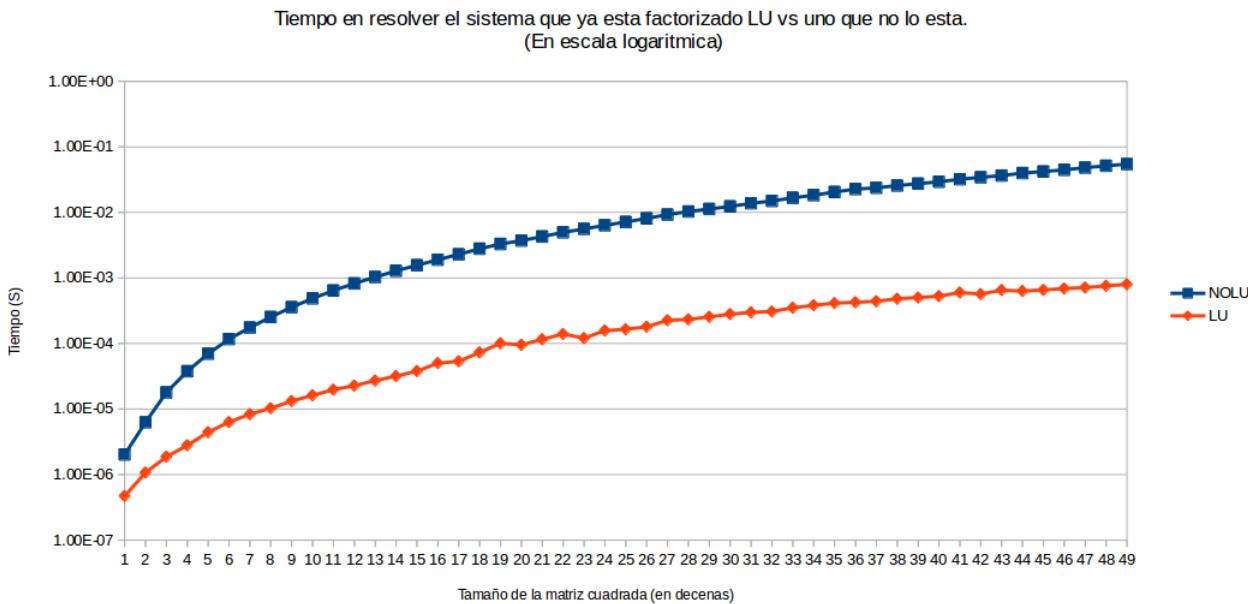
Como vemos ambos casos están en función lineal con la cantidad de pixeles, esto es gracias a que, aunque resolver un sistema sea cuadrático y el otro cubico, ambos están resolviendo sistemas de 3×3 N veces, por lo tanto el tiempo que tarda en resolver un sistema u otro se vuelve una constante que multiplica a N y por lo tanto tenemos ordenes lineales para los dos casos.

Es por esto que quisimos hacer otro experimento más para comparar la diferencia entre el tiempo cuadrático de resolver el sistema ya factorizado y de resolver el sistema utilizando eliminación gaussiana. Para esto creamos instancias de matrices cada vez más grandes y las duplicamos, a una le aplicamos eliminación gaussiana y a la otra la factorizamos y luego resolvemos los dos sistemas (L y U). Es importante destacar que no tomamos el tiempo que se toma la factorización ya que este es un tiempo que se amortiza en la cantidad de operaciones y no era lo que queríamos comprobar.

El resultado fue el siguiente:



Dado que los tiempos para la medición de LU son muy pequeños, decidimos hacer otro gráfico con escala logarítmica para apreciar mejor la diferencia.



Como podemos ver, efectivamente hay un crecimiento polinomial de ambos y la factorización LU crece en menor magnitud con el tamaño del sistema a resolver.

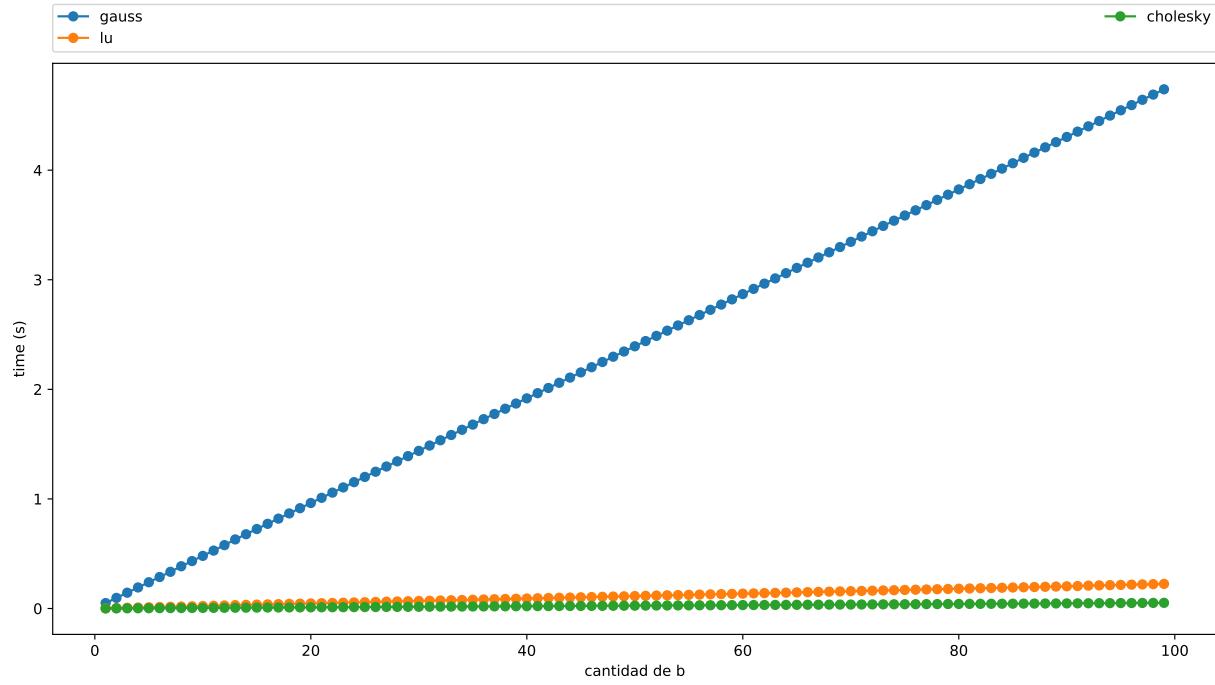
5.5. Cholesky

A continuación presentamos un análisis temporal de los métodos utilizados comparados con Cholesky. Para realizar estos análisis y poder hacer comparaciones con la eliminación gaussiana y LU, generamos matrices que todos los métodos pueden resolver. Creamos de forma automatizada matrices simétricas definidas positivas, generando matrices A con coeficientes aleatorios de 0 a 1 de tamaño n y las multiplicamos por su traspuesta A^t . Luego les sumamos el valor n en la diagonal garantizando que sean diagonal dominante y se las multiplica por un escalar para evitar valores muy cercanos al 0 que podrían ser inconvenientes para los métodos menos estables.

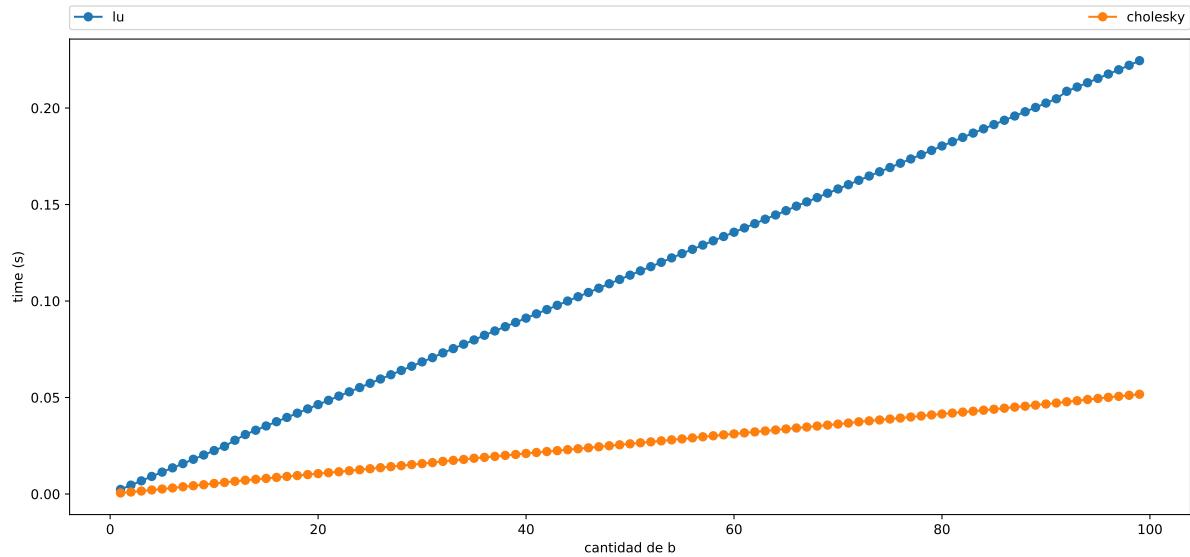
Una de las ventajas más importantes de las descomposiciones utilizadas es la de no tener que recalcular la matriz a resolver en cada iteración realizada si solo se cambia el término independiente. Para

observar este comportamiento utilizamos matrices de dimensión $500 * 500$ y resolvimos el sistema $Ax = b$ varias veces midiendo el tiempo que se tomaba en resolver el mismo.

Lo esperado es que Gauss sea el método más lento en este tipo de pruebas, ya que debe re-triangular la matriz cada vez que quiere resolver para un nuevo b . En cambio, Cholesky y LU una vez que su descomposición es encontrada, solo deben realizarse despejes. Sobre el eje x puede verse la *cantidad acumulada* de términos independientes, y en el eje y el tiempo en segundos.



Podemos observar en el gráfico que se cumple lo esperado. Dado que la diferencia entre LU y Cholesky no es apreciable, veamos los mismos datos utilizados en el gráfico anterior pero esta vez sin incluir a Gauss.



6. Conclusión

En primer lugar vimos cómo calibrar el sistema para adaptarlo a cualquier ángulo de luz, obteniendo resultados similares a los provistos por la cátedra.

Resolvimos sistemas lineales para encontrar los vectores normales a la superficie. Lo hicimos utilizando el algoritmo de eliminación gaussiana. Luego nos aprovechamos de la estructura de nuestras ecuaciones para mostrar que existe la factorización LU y la utilizamos en nuestro problema. Comprobamos en la experimentación que LU es bastante mas eficiente que Gauss, sin embargo, el tiempo que se tarda en el cálculo de normales se encuentra en el orden de los milisegundos para ambos métodos, mientras que el tiempo total se encuentra en el orden de los minutos, por lo que para nuestro problema es indistinto cuál usar.

Planteamos cómo sería un posible sistema matricial para el cálculo de profundidades a partir de las ecuaciones de los planos tangentes y vimos que cumplía ciertas propiedades. Para resolver el nuevo sistema, propusimos una nueva estructura de datos y resolvimos el sistema utilizando el algoritmo de Cholesky.

Finalmente, mostramos los resultados finales obtenidos. Consideramos las diferencias que encontramos y concluimos que aplicamos de manera satisfactoria la técnica de fotometría estéreo.