



## 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

<b>1. Introducción</b>	<b>3</b>
<b>2. Calibración</b>	<b>4</b>
<b>3. Cálculo de normales</b>	<b>6</b>
3.1. Eliminación Gaussiana . . . . .	6
3.2. Factorización LU . . . . .	7
<b>4. Estimación de profundidades</b>	<b>9</b>
4.1. Cholesky . . . . .	12
<b>5. Resultados finales y experimentación</b>	<b>13</b>
5.1. Calibración de luces . . . . .	13
5.2. Normales . . . . .	15
5.3. Profundidades . . . . .	18
5.4. Eliminacion gaussiana y Factorización LU . . . . .	24
5.5. Cholesky . . . . .	25
<b>6. Conclusión</b>	<b>27</b>

## 1. Introducción

Este trabajo consiste en la reconstrucción 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 cuáles 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 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 segunda instancia, para el cálculo de profundidades aprovecharemos la forma de la matriz final para crear una nueva estructura más eficiente, donde aplicaremos el algoritmo de Cholesky para resolver el sistema.

En último lugar, realizaremos experimentos que intentarán mostrar cómo se comporta nuestro sistema tanto en la calibración, como en la obtención de normales y los cálculos de profundidad. Veremos cuáles son los resultados finales y cómo las diferentes elecciones de luces repercuten en el modelo generado.

## 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 de 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 más 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  $S = c - p$ .

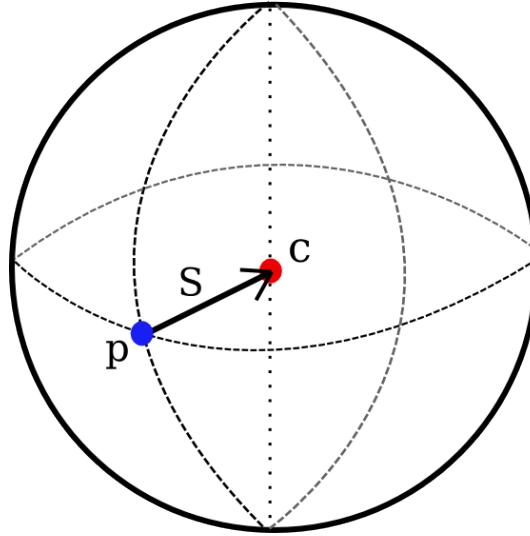
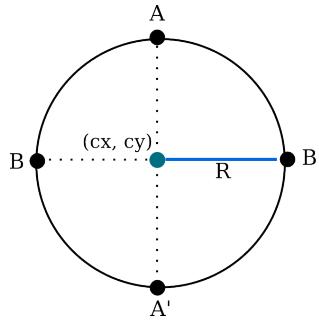


Figura 1: Imagen del vector que nos interesa,  $S = c - p$

Más 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 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 brillante 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  $5 \times 5$ , 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 sólo **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 sólo 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 nuestra matriz extendida 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 el 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 veamos de qué 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 más eficientemente.

## 4. Estimación 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. 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 cómo sería el sistema matricial para una imagen de 2x3 pixeles, y veremos cómo 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 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 tiene cualquier forma, sino que tiene una forma particular.

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  $M^t M$

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

Es fácil darse cuenta del patrón. Vamos tomando  $p$  el píxel 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  $A$  tambien 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 querramos aplicar Cholesky.

En nuestra matriz podrían aparecer  $n_z = 0$  pero sólo 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, por lo que nuestra matriz no tendrá ningún  $n_z = 0$

La matriz  $M$ , por lo dicho anteriormente, no tiene columnas nulas.

Además, si tomamos una fila cualquiera veremos que solo hay dos columnas que tienen algún elemento distinto de cero en esa fila, y los demás elementos de las columnas estarán en posiciones diferentes. Es decir, por la forma en la que construimos la matriz y la cantidad de ceros que existen, no hay ninguna columna que sea combinació lineal de otra. Por lo tanto, las columnas de  $M$  son linealmente independientes.

Entonces, el rango columna de  $M$  es completo, por lo que su núcleo será solo el vector nulo. Es fácil ver esto, pues como las columnas de  $M$  son LI la única combinación lineal que forma al cero es aquella con coordenadas únicamente cero. Utilizando esto, podemos observar que:

$$\forall x \neq 0, Mx \neq 0$$

Queremos probar que  $A$  es definida positiva. Recordando que  $A = M^t M$  tenemos que:

$$x^t Ax = x^t M^t Mx = (Mx)^t Mx = \|Mx\|_2^2 \geq 0 \quad \forall x \iff \|Mx\|_2 \geq 0 \quad \forall x$$

Pero por lo desarrollado anteriormente,  $Mx \neq 0$  pues  $x \neq 0$ , y una norma es 0 sí y solo sí se le es aplicada al 0, entonces:

$$\|Mx\|_2 > 0 \quad \forall x \neq 0$$

De esta manera:

$$x^t Ax > 0 \quad \forall x \neq 0$$

Por lo tanto,  $A$  es definida positva. Sumado a que es simétrica, demostramos que  $A$  es simétrica definida positiva, de lo cual concluimos que tiene factorización de Cholesky.

Aunque lo que mostramos antes de la demostración 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 más 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 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 más 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 tiene 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 matriz  $L$  basándose 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 más bajas 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 aunque 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. Calibración de luces

Lo primero que analizaremos será la calidad de nuestra calibración. Nos parece importante conseguir un buen resultado de luces ya que serán utilizadas en todos los pasos siguientes. Muy amablemente la cátedra nos cedió los datos de las luces reales con las cuales podremos comparar.

Dado que nuestro método de calibración se basa en utilizar imágenes de esferas, utilizaremos las imágenes *mate* y la *cromada* para ver con cuál conseguimos mejores resultados. Como ambas son esferas, esperamos obtener los mismos resultados con ambas.

En primer lugar veremos la esfera *mate*. Para cada imagen estimamos el vector de luz correspondiente y graficamos los tres ejes. No es demasiado importante el valor que toma cada eje, sino que lo que nos interesa es ver la diferencia con los valores de la cátedra.

Graficamos del mismo color cada eje. La línea llena se corresponde a los valores de la cátedra mientras que la punteada son los valores obtenidos por nosotros.

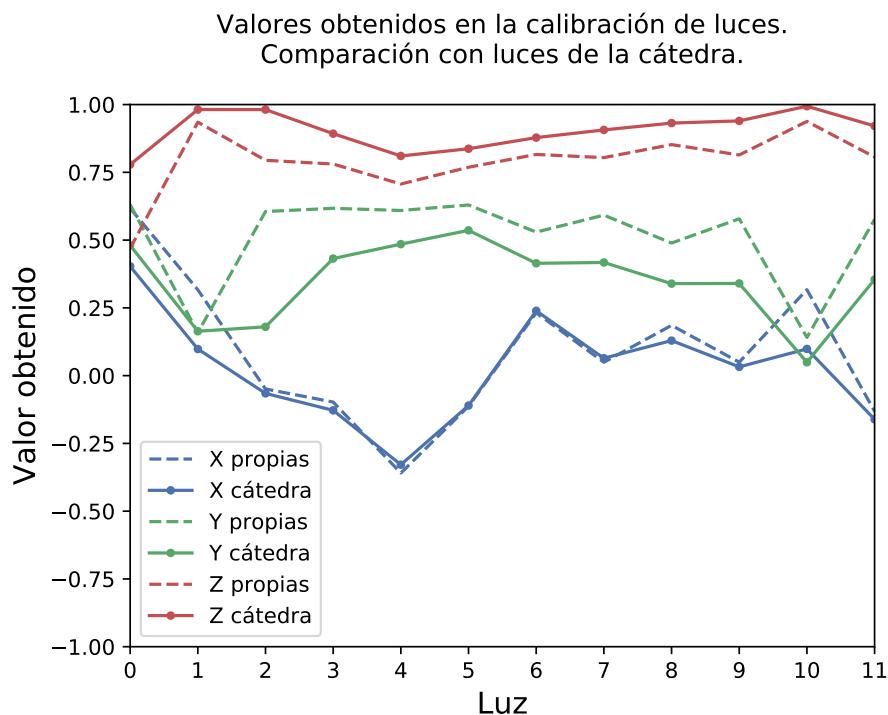


Figura 2: Calibración de luces utilizando la esfera mate.

Lo que observamos es que en líneas generales los valores nuestros y los de la cátedra dan parecidos, las curvas son similares. El eje X es el que más se aproxima al valor real, pero no sabemos por qué sucede esto sólo con este eje. Las diferencias que existen son de hasta 0.30 puntos (sobre 1.00), que si bien no parece demasiado, creemos que esto puede afectar el resultado final.

Veamos qué sucede si consideramos la esfera *cromada*:

Valores obtenidos en la calibración de luces.  
Comparación con luces de la cátedra.

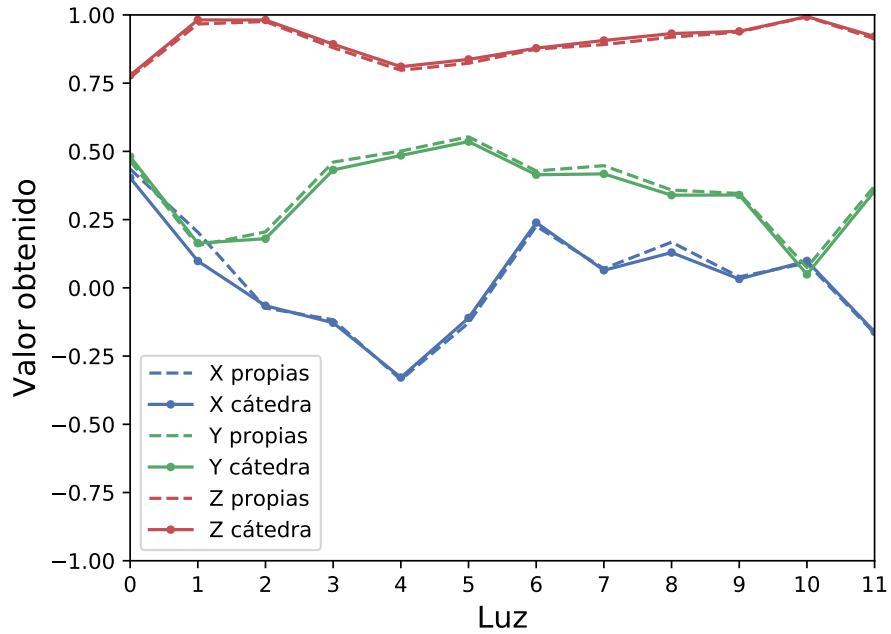


Figura 3: Calibración de luces utilizando la esfera cromada.

Para nuestra sorpresa, es claro que obtuvimos mejores resultados con la esfera cromada. Esperábamos obtener valores similares en ambas. Creemos que la razón por la que esto sucede, es porque en la esfera cromada es extremadamente notorio cuál es el punto más brillante, por lo que es más exacta la identificación. En la esfera mate, en cambio, se tienen blancos muy similares en toda la imagen.

## 5.2. Normales

En esta sección veremos cómo la calibración y elección de las luces afecta el resultado de las normales. En primer lugar, aprovecharemos que encontramos diferencias entre las luces obtenidas por la esfera mate y la esfera cromada para ver los errores que producen un set de luces 'desviado'. Por simplicidad, llamaremos a las luces de las tres categorías simplemente como: luces mate, cromadas y cátedra.

Sería muy razonable encontrarnos con algo de diferencia entre los sets de mate y cromada. Esperamos tambien que no haya diferencias entre cromada y cátedra, ya que los valores de luces eran prácticamente iguales.

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'. Hay cientos de posibles combinaciones de luces a elegir, asi que sólo mostramos algunas.

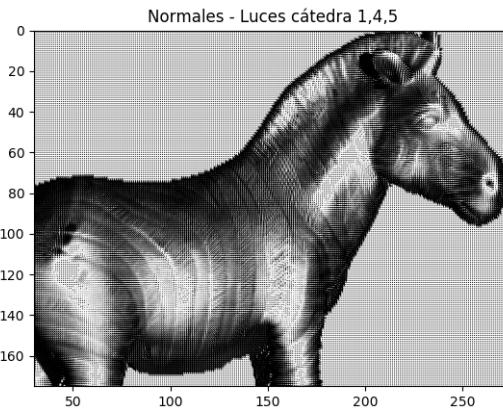


Figura 4: Luces cátedra 1,4,5

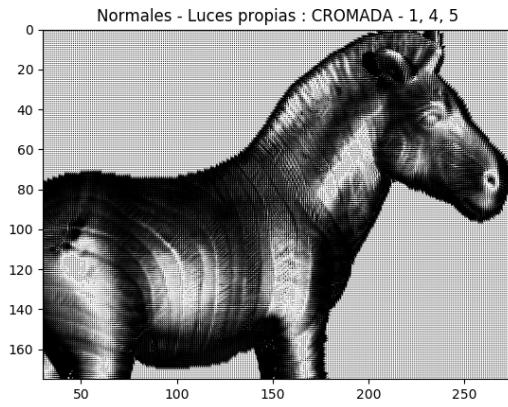


Figura 5: Luces cromadas 1,4,5

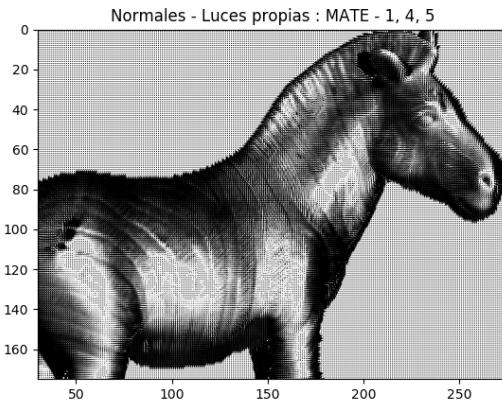


Figura 6: Luces mate 1,4,5

Con este set de luces podemos ver que no hay ninguna diferencia notable entre las normales producidas por el set de luces de la cátedra y el set de luces cromada. La mate es muy similar tambien, pero tiene zonas que son un poco mas claras, como por ejemplo en el rostro del caballo. De todos modos, las tres son similares entre sí.

En la mayoría de los sets que probamos obtuvimos resultados similares, con las luces cromada y cátedra obtenemos resultados casi idénticos y un poco de diferencia con el set mate. A continuación uno de

los ejemplos un poco mas extremos que encontramos:

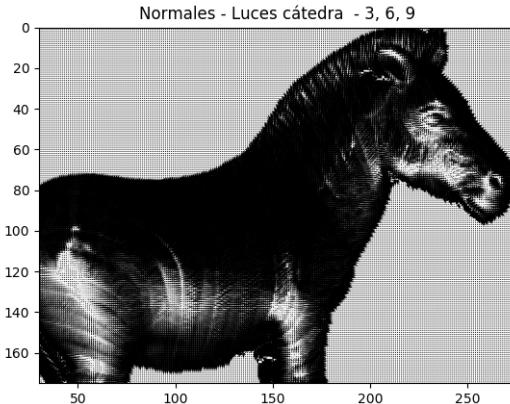


Figura 7: Luces cátedra 3,6,9

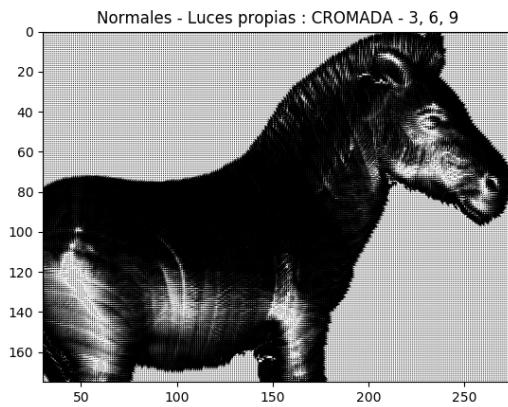


Figura 8: Luces cromadas 3,6,9

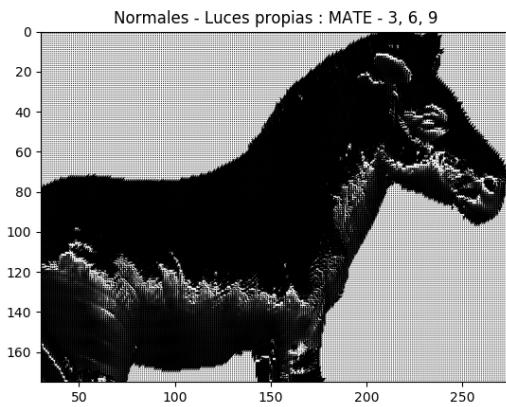


Figura 9: Luces mate 3,6,9

En este caso es muy notoria la diferencia con el set mate. Esto muestra que no usar un set de luces correcto puede traer consecuencias notorias para el cálculo de las normales. Una calibración errónea acarea errores notorios para el resto de las etapas. La similaridad entre los sets cromadas y cátedra se sigue mateniendo siempre, por lo que en los siguientes experimentos sólo consideraremos nuestro set de luces cromadas.

Lo siguiente que nos interesa ver es cómo afecta la elección de las tres luces en el cálculo de las normales. Para esto dejaremos fijas dos luces y moveremos una tercera. Elegimos para dejar fijas las luces 1 y 4 pues en las imágenes parecieran que apuntan en sentido inverso.

A continuación mostramos algunos de los resultados obtenidos, la mayoría se ven similares.

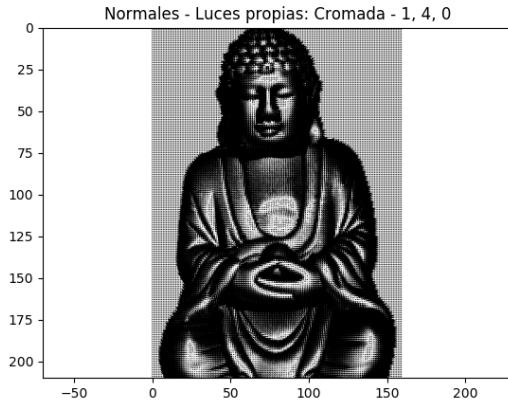


Figura 10: ↑ Luces cromadas 1,4,0

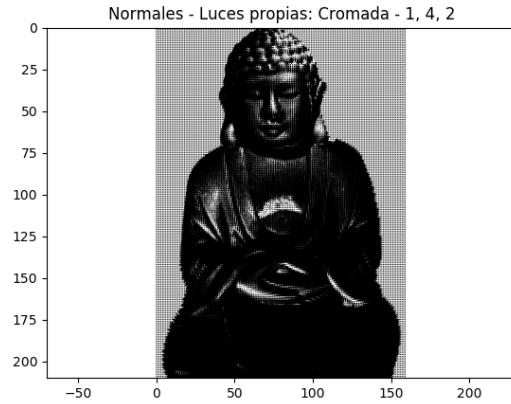


Figura 11: ↑ Luces cromadas 1,4,2

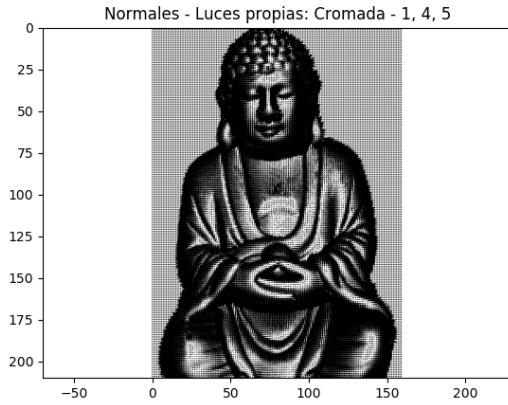


Figura 12: ↑ Luces cromadas 1,4,5

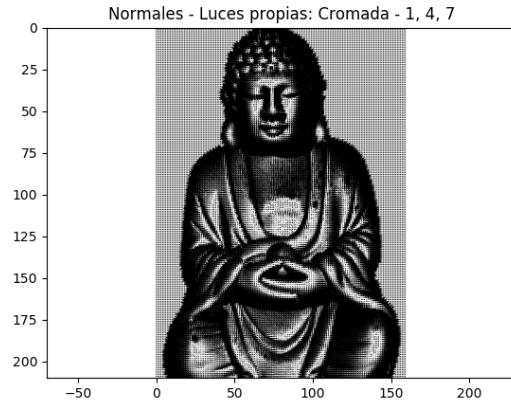


Figura 13: ↑ Luces cromadas 1,4,7

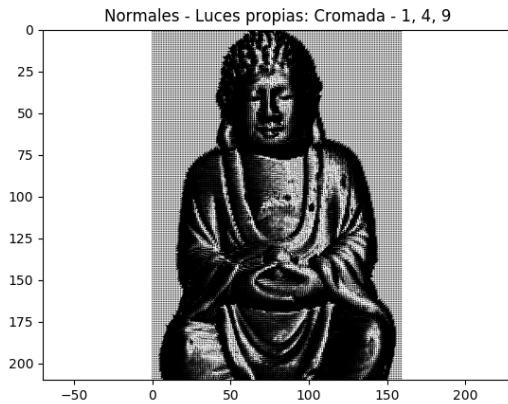


Figura 14: ↑ Luces cromadas 1,4,9

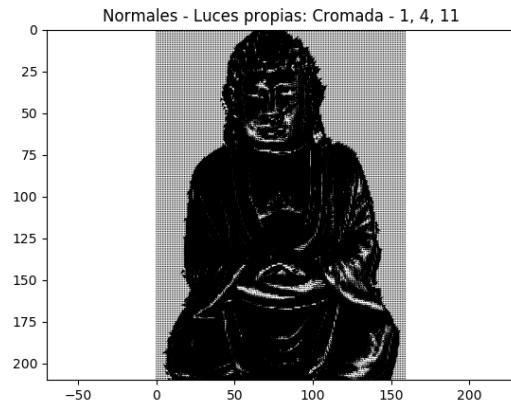


Figura 15: ↑ Luces cromadas 1,4,11

En la mayoría de los casos se obtuvieron resultados similares. Las excepciones se ven en la *Figura 11* y la *Figura 15* dónde la tercera luz es la 2 y la 11 respectivamente. Observando las imágenes reales, la imagen con luz 11 es muy similar a la que tiene luz 4, mientras que la imagen con luz 2 es similar a la de luz 1. Para el resto de las imágenes hay diferencias más notorias entre los ángulos de luz. Esto último nos da la pauta de que se obtienen mejores resultados si los ángulos de luces son mas variados.

### 5.3. Profundidades

En esta sección analizaremos los resultados finales de la aplicación del método. Más precisamente, analizaremos las profundidades que obtuvimos siguiendo el proceso descripto en la sección desarrollo, utilizando nuestras luces de calibración (que resultaron casi idénticas a las de la cátedra).

En primer lugar queremos darnos una idea general de qué tan correctos son nuestros valores. Para esto tomaremos las profundidades obtenidas y graficaremos las curvas de nivel, marcando con diferentes colores cada capa de profundidades.

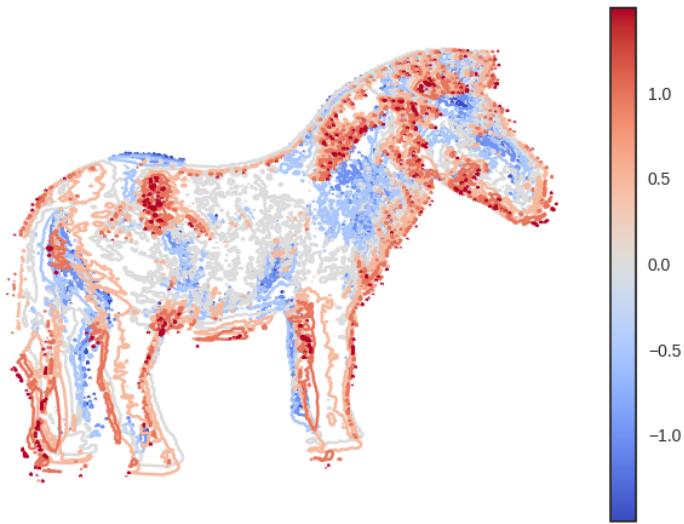


Figura 16: Superficies de nivel, vista superior

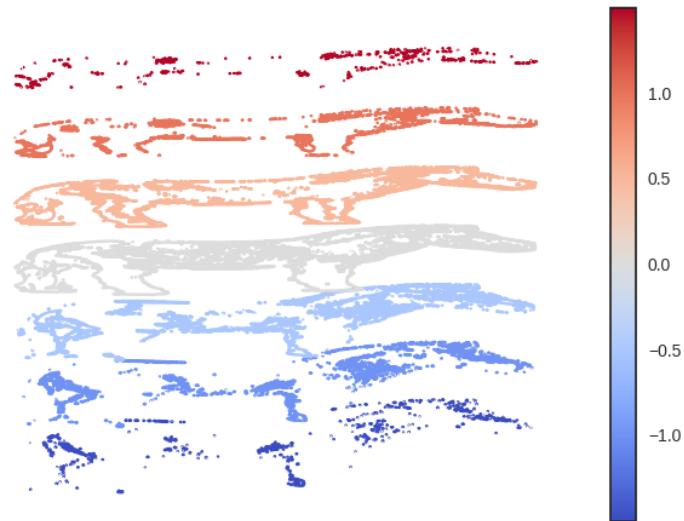


Figura 17: Superficies de nivel, vista lateral

Con esto podemos ver que las profundidades son adecuadas y la figura original del caballo es distinguible. Notamos que en los bordes de la imagen hay gran cantidad de rojos: creemos tiene que ver con que al

haber usado la máscara, la diferencia de alturas es muy brusca entre la figura y el plano del fondo.

Si consideramos la superficie completa de profundidad en un modelo 3D, los resultados no fueron perfectos tampoco. Los sets de luces utilizados fueron elegidos arbitrariamente.

Profundidades - Luces: 3,6,9

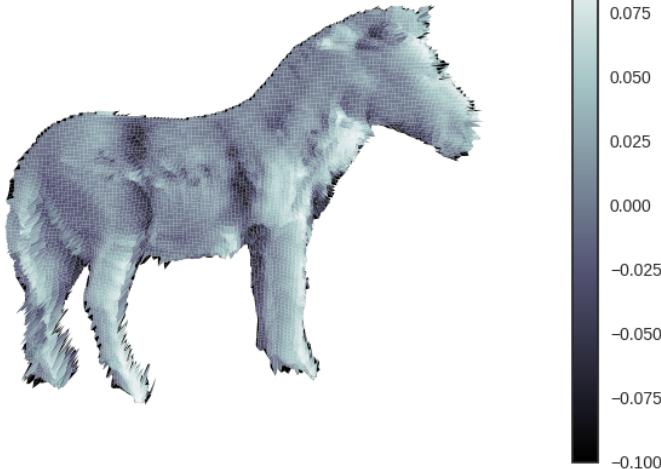


Figura 18: Profundidades utilizando luces 3,6,9

Aquí podemos ver como la forma general se distingue bastante bien, y que las zonas mas iluminadas del modelo se corresponden con las zonas iluminadas de las imágenes originales. Sin embargo, hay muy pocos detalles presentes, sobre todo en el rostro del caballo. Es posible que el problema sea la elección de las luces, pero no es el caso, pues se obtienen modelos similares también con otras elecciones:

Profundidades - Luces: 5,7,8

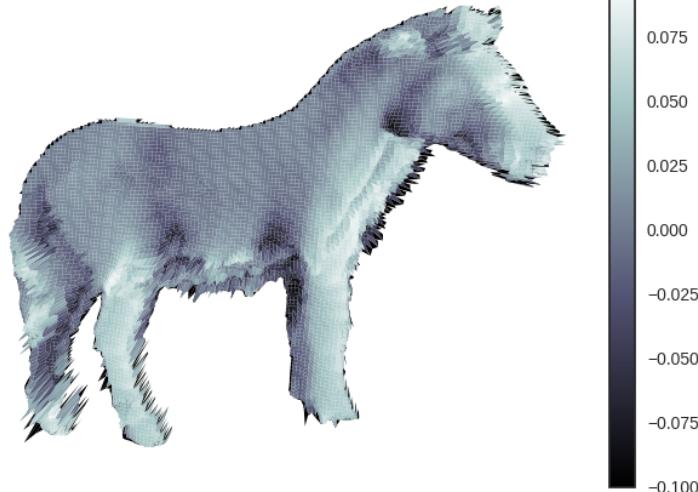


Figura 19: Profundidades utilizando luces 5,7,8

Probar algunas combinaciones mas pues probablemente haya alguna mejor

Aunque las profundidades son distinguibles en los casos que mostramos, los resultados no son lo que esperábamos. Los modelos obtenidos no se corresponden con el modelo propuesto en el enunciado. Podemos ver que nuestro modelo está lleno de picos, sobre todo en el borde de la figura. Creemos que los resultados tienen que ver con una combinación de errores numéricos y los cálculos de los clanos tangentes, no pudimos detectar una falla específica y obtener resultados mejores. Aún así, consideramos que lo que obtuvimos son resultados razonables.

Otra cosa clara es que los detalles (por ejemplo el rostro) se pierden. Si bien era esperable ya que son áreas delicadas, nos da la pauta de que nuestras aproximaciones parecen no ser del todo correctas. De todos modos, no podemos saber si en una estimación 'bien hecha' los detalles se mantienen o no.

En los resultados que presentamos utilizamos el promedio de los colores para obtener los datos de la imagen original. Veremos qué sucede si en vez del promedio consideramos las diferentes componentes de color por separado. No esperamos obtener diferencias significativas entre tomar el promedio o una sola componente del color.

#### Profundidades de caballo mismo set de luces, diferentes tomadas de colores

Cómo sospechabamos, 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. Tomemos entonces una imagen con colores, como es el caso del gato. Creemos que seguirá sin haber diferencias apreciables, pues el promedio de los colores de cierta forma engloba a todos los datos.

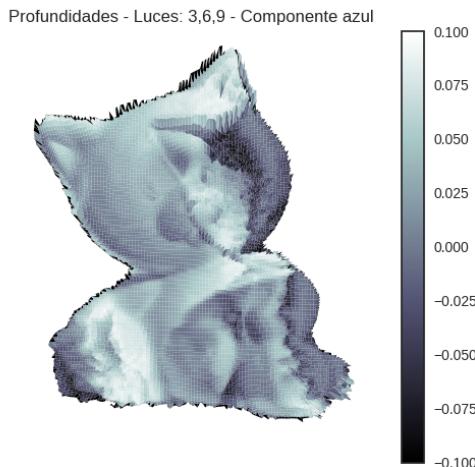


Figura 20: Luces 3,6,9 - Componente Azul

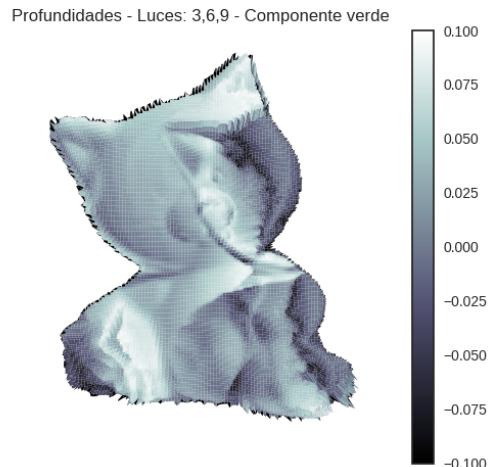


Figura 21: Luces 3,6,9 - Componente Verde

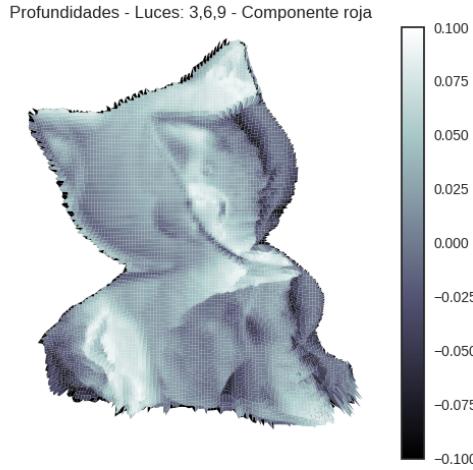


Figura 22: Luces 3,6,9 - Componente Roja

Las tres componentes se ven bastante similares, aunque hay algunas diferencias. La componente azul es la que parece tener los detalles más marcados, ver por ejemplo en los ojos del gato. La componente roja tiene menos detalles y en todo el modelo se vé de un color más claro que los demás, es decir la profundidad obtenida es un poco diferente. Entonces en una imagen en color, contrario a lo que sucede con una imagen blanco y negro, las diferentes componentes de color tienen influencia en el resultado final, aunque no demasiada.

Más allá de las diferencias entre componentes, es claro que en la figura del gato hay una sombra muy extraña en la mitad derecha del rostro, que parece ser muy similar al contorno de una oreja izquierda. Para descartar que sea problemas con el set de luces elegido, repetimos el mismo experimento con un set de luces diferentes. Aprovechamos además para ver si los colores influyen de diferente manera con otro set.

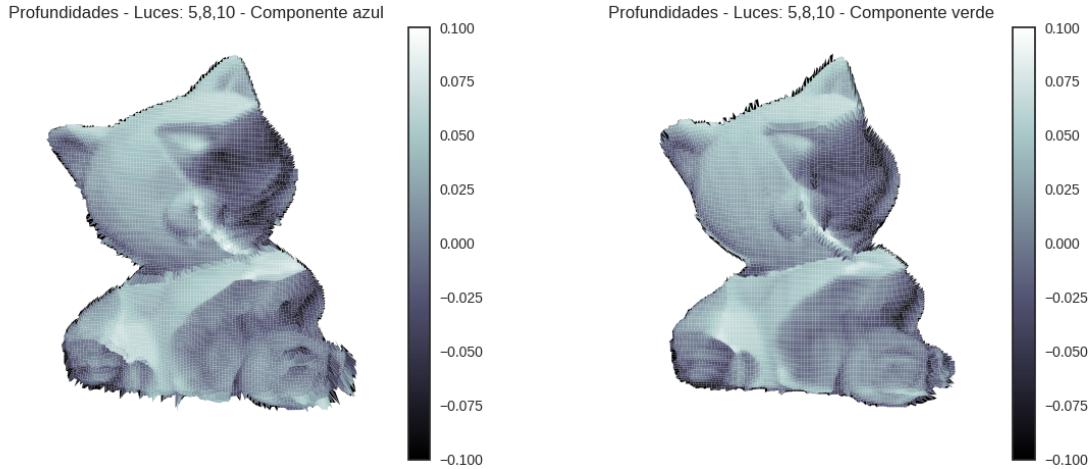


Figura 23: Luces 5,8,10 - Componente Azul

Figura 24: Luces 5,8,10 - Componente Verde

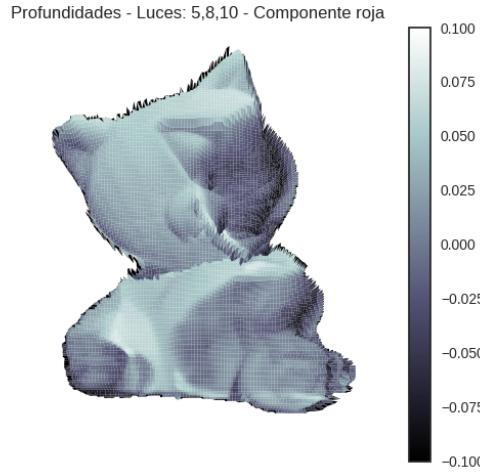


Figura 25: Luces 5,8,10 - Componente Roja

En las últimas tres figuras utilizamos un set diferente pero tenemos un comportamiento similar al de antes: la componente azul maneja una cantidad de detalles un poco mayor que las demás, notorios en la parte inferior de la figura. Sin embargo con este set de luces la diferencias entre las componentes son mucho menores. Además, seguimos observando la sombra extraña en el lado derecho del rostro, que ahora estamos bastante seguros que es algún problema con nuestro sistema. Lamentablemente no pudimos hallar su causa para solucionarlo y obtener mejores resultados.

Es interesante notar que en los ejemplos anteriores utilizamos diferentes combinaciones de luces y obtuvimos profundidades diferentes (pero similares) entre los sets. En el ejemplo del gato, con el primer set de luces logramos conseguir más detalles que con el segundo set, pero pensábamos que las diferencias iban a ser más contundentes. En el siguiente experimento intentaremos ver cómo la elección de luces repercute en las profundidades finales. Para ello, realizamos los cálculos de profundidad con una imagen que tiene más detalles.

Los siguientes son gráficos con las profundidades de Buda, fijando las dos primeras luces y haciendo variar una tercera, al igual que hicimos con el experimento de las normas en la sección anterior.

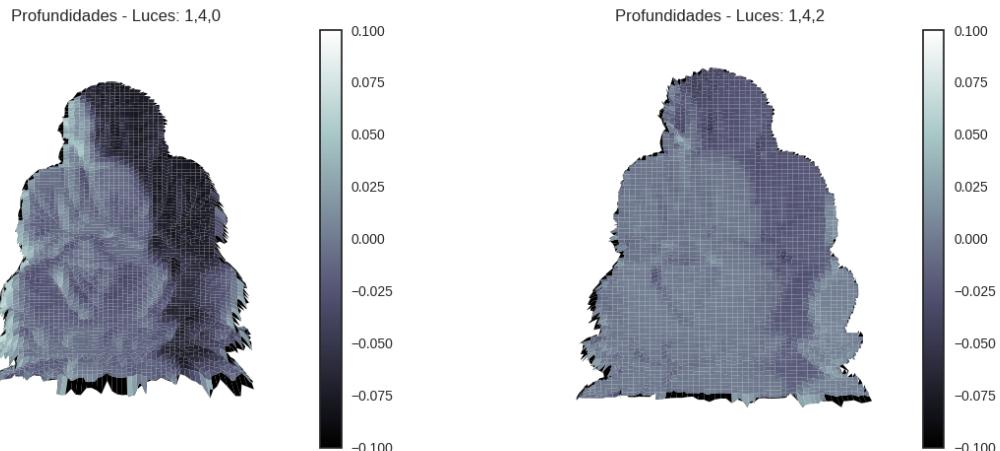


Figura 26: Profundidades - Luces 1,4,0

Figura 27: Profundidades - Luces 1,4,2

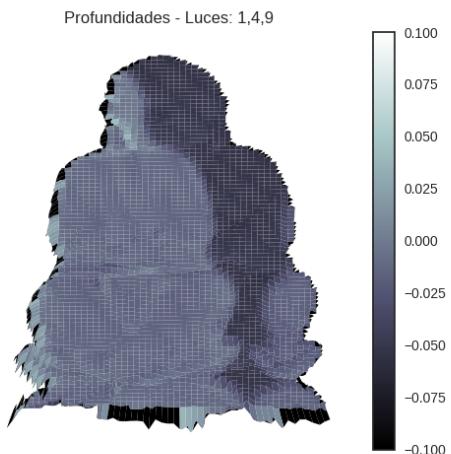


Figura 28: Profundidades - Luces 1,4,9

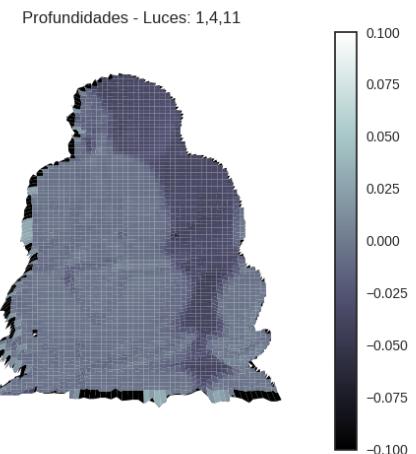


Figura 29: Profundidades - Luces 1,4,11

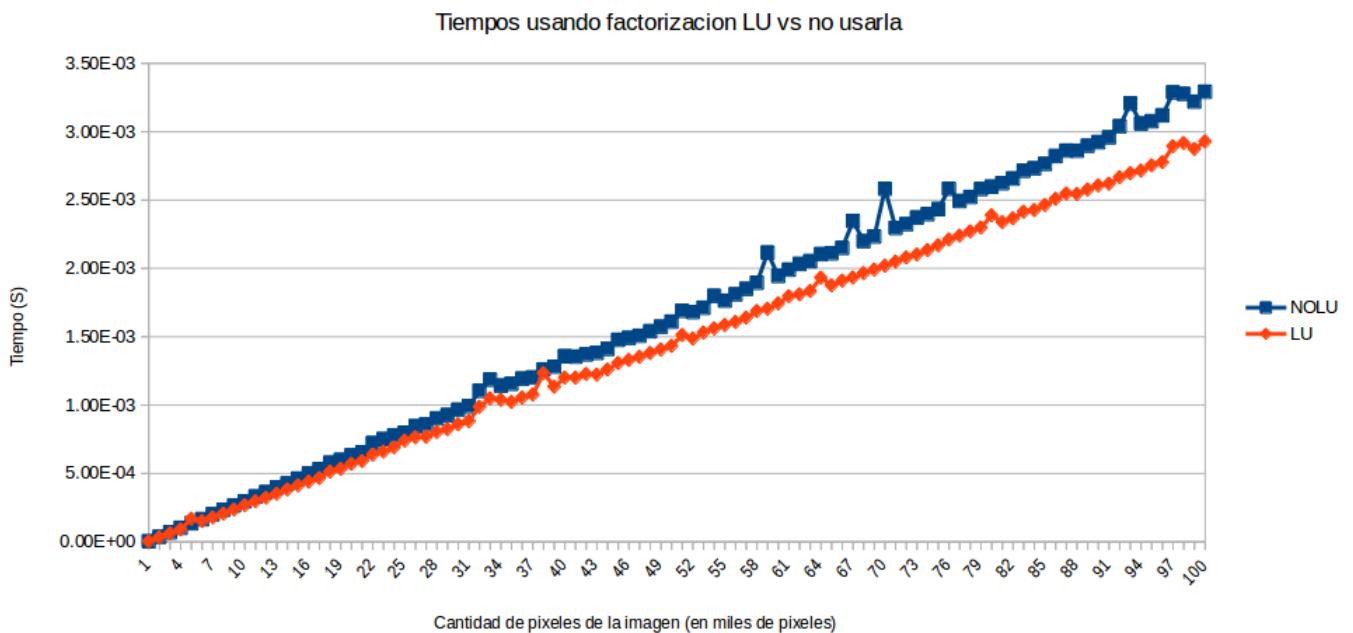
Conclusiones sobre los detalles y las cosas de buda

## 5.4. Eliminacion gaussiana y Factorización LU

Lo que queremos lograr con esta experimentación es comparar el resultado del cálculo sobre todos los píxeles 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 píxeles.

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 píxeles y luego dividimos por esa cantidad para sacar el promedio, de esta manera reducimos el ruido que pudo haberse visto durante la medición.

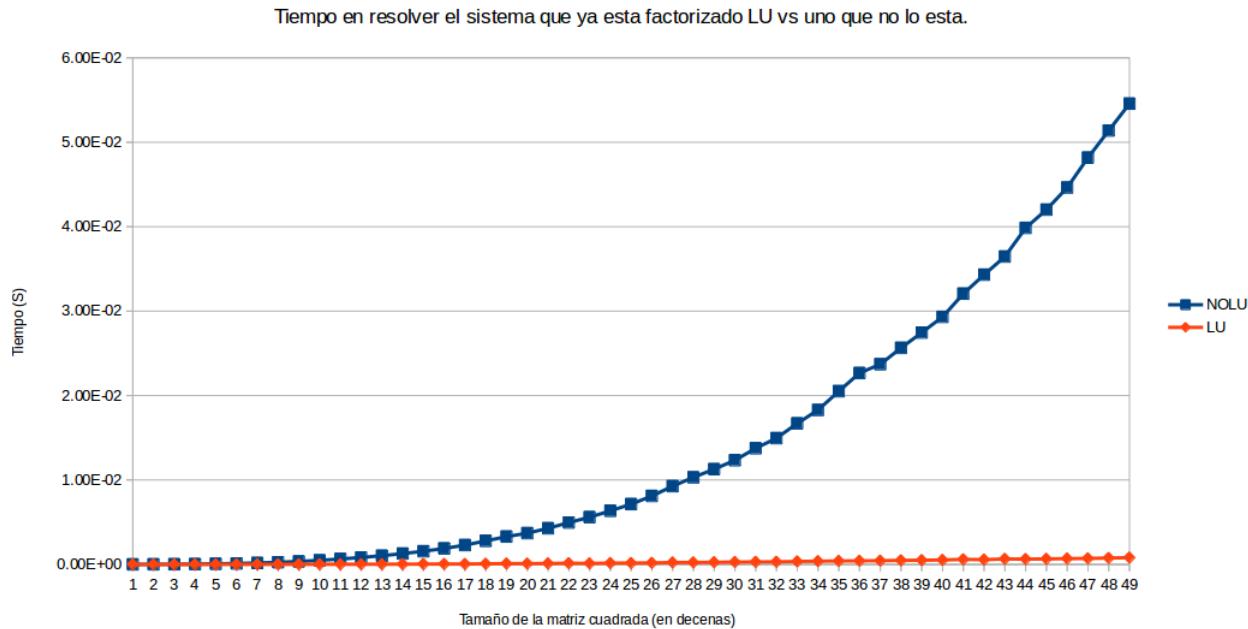
Hicimos varias mediciones variando la cantidad de píxeles y estos fueron los resultados:



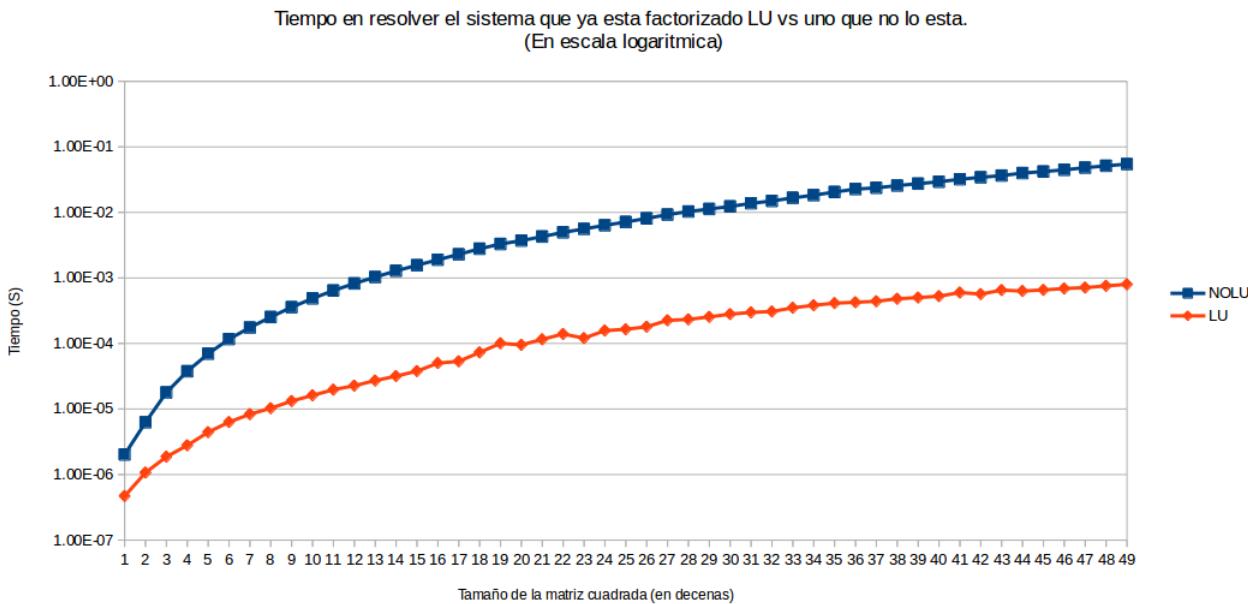
Como vemos ambos casos están en función lineal con la cantidad de píxeles, esto es gracias a que, aunque resolver un sistema sea cuadrático y el otro cúbico, ambos están resolviendo sistemas de  $3 \times 3$  N veces, por lo tanto el tiempo que tarda en resolver un sistema u otro se vuelve una constante que multiplica a N y por lo tanto tenemos órdenes 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 resolvimos los dos sistemas (L y U). Es importante destacar que no tomamos el tiempo que 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.

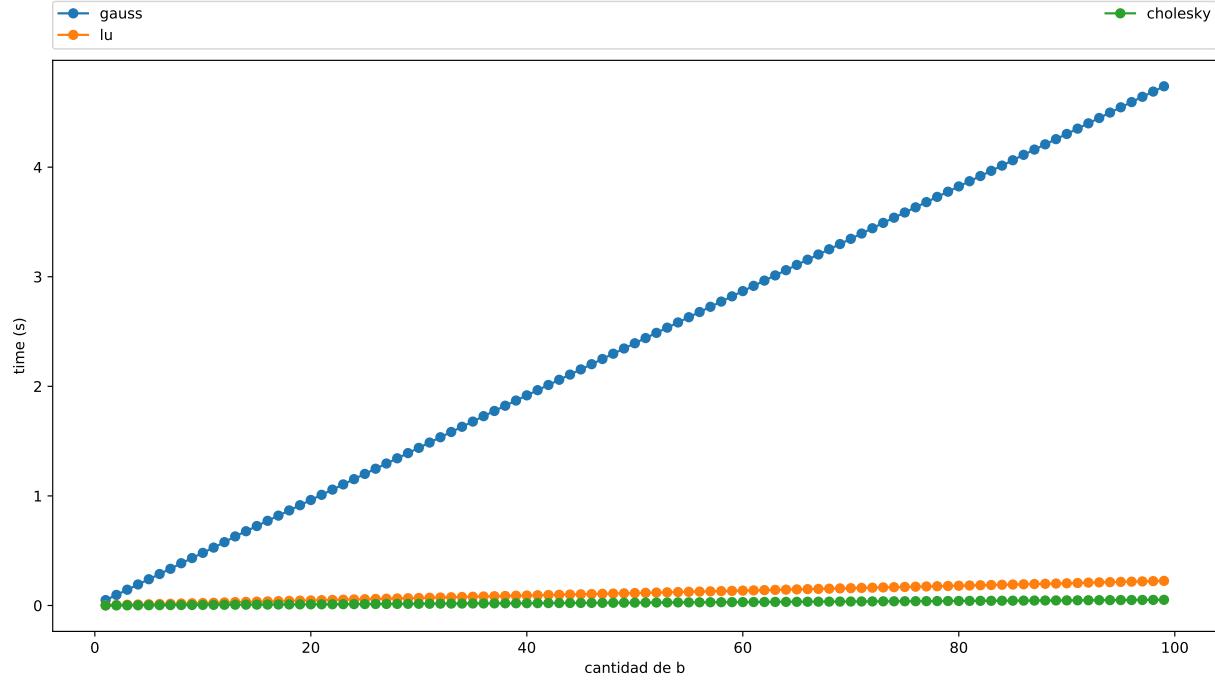
## 5.5. Cholesky

A continuación presentamos un análisis temporal de los métodos utilizados comparados con Cholesky. Para realizar estos análisis y poder hacer comparaciones con la eliminació 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 sólo 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, sólo 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. Algo quizás no contemplado es la pequeña diferencia de tiempos entre LU y Cholesky, que sospechamos fuertemente que está relacionada con la forma en que los implementamos.

## 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 más 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.