Informe proyecto final.

Daniel Méndez *Miembro IEEE,* Camila Sanhueza *Miembro IEEE*

***Resumen*—Basados en el paper de Hake [1], en este informe mostramos los resultados que obtuvimos al intentar hacer la reconstrucción 3D de un objeto, siguiendo lo expuesto por Hake.**

Para comenzar, introducimos explicando las dificultades que se nos presentaron durante el proceso. También definimos las diferencias del resultado final con lo que propusimos al comenzar el desarrollo del proyecto. Hemos de explicar también, el diseño y la implementación del código generado y utilizado.

Se muestran también en detalle los resultados obtenidos y un análisis de éstos.

I. Introducción

La reconstrucción en 3D de un objeto, es un área que tiene muchos, y muy variados, acercamientos. Desde 1980 con la introducción de métodos fotométricos, o a través de la técnica llamada *shape-from-shading*, se ha intentado construir en realidad aumentada los objetos en el computador.

Con la puesta en el mercado de la Kinect, se presentó una nueva forma de lograr el tan anhelado objetivo. Es así como nuevas mentes encontraron formas distintas de hacer una construcción en tres dimensiones.

Para éste proyecto, partimos basándonos en el paper de Hake [1], el cual plantea el uso del receptor infrarrojo para captar las diferentes imágenes del objeto. Además sugiere usar una fuente externa de luz infrarroja, para obtener mejores resultados de resolución a la hora de reconstruir. Los problemas al querer seguir este método, se presentaron a la hora de obtener las normales para hacer el mapa de profundidad, dado que la documentación de Hake es escasa al ser un método propuesto por él.

Para resolver este problema, combinamos el acercamiento de Hake con el de Schindler [2]. Usamos fotometría no calibrada para hacer el calculo de las normales a partir de 4 fotos tomadas con la Kinect y con una fuente externa de luz infrarroja. Luego, para el armar el mapa de profundidad, utilizamos el método de Gauss-Seidel con relajación.

Finalmente, las pruebas son generadas haciendo una modificación del código de Kai Wolf [3], para adaptarlo a nuestras condiciones de trabajo, es decir, utilizar la Kinect como cámara y la fuente externa de luz infrarroja.

II. Diseño e implementación

Para lograr el objetivo de realizar reconstrucciones $3D$, utilizamos el código de Kai Wolf, que nos brindaba la fotometría no calibrada [3]. A éste trabajo le realizamos una serie de modificaciones y agregamos el código que nos permite utilizar la Kinect como cámara capturadora. Para poder realizar las reconstrucciones en $3D$ utilizamos una luz infrarroja, que no entregó resultados favorables. Cambiamos entonces a una halógena con filtro dicroico, filtro que aporta un componente infrarrojo suficiente para captarlo conla Kinect, pero no tan fuerte como la luz infrarroja.

mail: daniel.mendezf@mail.udp.cl
mail: camila.sanhueza@mail.udp.cl

Código 1. Prototipo función `exportMesh`.

```
1 void exportMesh(cv::Mat Depth, cv::Mat Normals, cv::Mat
      texture);
```

A. Código que permite la utilización de la Kinect

- Diseño: Al momento de analizar el uso de la Kinect, se nos presentaron diferentes librerías (`OpenNi, Freenect, etc`) que nos brindaban herramientas para el desarrollo. Al momento de realizar pruebas con las librerías mencionadas, ninguna resultó de la forma deseada. Lo que nos obligó a recurrir a otros medios, llegando finalmente a utilizar `Openframenworks`.
- Implementación: para poder utilizar la Kinect como sensor de profundidad y de intensidades. Usamos las herramientas de `Openframenworks`, específicamente `ofxKinect`. Este código nos permite obtener y almacenar las imágenes de intensidad y mapas de profundidad de las pruebas. Esto se realiza al presionar la tecla "S", con la que podemos almacenar las intensidades del frame que en ese instante esta siendo visto por la Kinect. Al presionar "D", almacenamos los mapas de profundidad. El funcionamiento de esta libreria consiste en que es necesario determinar las variables esenciales a utilizar en el archivo `ofApp.h`, el cual crea una clase. En el archivo `ofApp.cpp` se encuentra las siguientes funciones: setup, update y draw, que siempre se ejecutan el orden mencionado, luego se continua realizando ciclos entre update y draw. La función `setup` configura las opciones necesarias para iniciar la Kinect, `update` como su nombre lo indica, renueva la información entregada por la Kinect y finalmente `draw`, dibuja la información necesaria en la pantalla. Además incluye la función `pressKey`, que permite realizar acciones al presionar diferentes teclas.

B. Código de la fotometría no calibrada

1) Función `exportMesh`:

- Diseño: esta función nos permite transformar la información de las matrices de profundidad y de normales, en una representación $3d$. Para poder visualizar las reconstrucciones realizadas utilizamos la herramienta `meshlab`.
- Implementación: es función las variables `ofstream`, en donde esta crea un archivo con extensión `.obj` en donde los datos son ingresados escribiendo en el mismo archivo los valores necesarios de estas matrices.

Código 2. Prototipo función `imageMask`.

```
1 Mat imageMask(vector<Mat> camImages, int numPics, Mat
      ambient);
```

Código 3. Prototipo función `computeNormals`.

```
1 Mat computeNormals(vector<Mat> camImages,Mat Mask =Mat());
```

2) Función `imageMask`:

- Diseño: esta función consiste en eliminar todo el fondo de la imagen, transformándolo a color negro. Donde se encuentra el objeto de prueba, se deja la silueta del objeto, rellenándolo con color blanco.
- Implementación: en el código 2 se puede observar el prototipo de esta función. El parámetro `camUmagenes`, es un vector que posee las imágenes de las distintas iluminaciones, `numPics` son las cantidad de imágenes que posee este arreglo y finalmente el `ambiente` es una imagen más oscura para obtener el fondo. Para obtener la mascara se sumaron todas las imágenes y se les restó el ambiente. Para no obtener valores fuera del rango de escala de grises se realiza un threshold.

3) Función `computeNormals`:

- Diseño: esta función consiste en calcular la normal para cada pixel de las imágenes, y asignarle a la matriz resultante un valor que es representado con diferentes colores. Como no se podía controlar o calibrar las fuentes de iluminación, fue necesario utilizar un método de fotometría estero no calibrada.
- Implementación: al igual que en la función anterior se le entrega el arreglo de imágenes, más la mascara calculada anteriormente. Inicialmente esta función crea una matriz que almacene la información de intensidad de las fotos y en las diferentes columnas de esta se representan imágenes con diferentes iluminaciones. Posterior a crear esta matriz se le realiza una descomposición de valores singulares para obtener los valores propios y así poder obtener la información en una matriz de tres canales (RGB).

4) Función `cvtFloatToGrayscale`:

- Diseño: esta función, tal como su nombre lo dice, transforma una matriz de valores flotantes a un matriz en escala de grises.
- Implementación: en el código 4 se ingresa una matriz para que sea transformada a una escala de grises, principalmente a valores que no sobrepasen el 255.

5) Función `localHeightfield`:

- Diseño: esta función nos permite, a través del calculo de de normales, realizar una estimación de las profundidades de la imagen. Se utiliza el método de Gauss-Seidel con relajación, de esta manera se calcula la profundidad mediante un método iterativo, actualizando el valor de cada pixel en relación a su vecindario. De esta forma se utilizan los valores de las normales y el mapa de profundidad, que comienza siendo plano, va aumentando la profundidad a medida que avanzan las iteraciones.
- Implementación: para estimar las profundidades se realizaron iteraciones sobre una serie de pirámides de la matriz de las normales. En éstas iteraciones se calcula la profundidad de un píxel mediante el análisis de su vecindario. Después de la estimación de esta matriz de profundidad, buscamos los valores máximos y mínimos para que nuestra profundidad no supere estos valores.

Código 4. Prototipo función `cvtFloatToGrayscale`.

```
1 Mat cvtFloatToGrayscale(cv::Mat F, int limit = 255);
```

Código 5. Prototipo función `localHeigthfield`.

```
1 Mat localHeightfield(cv::Mat Normals);
```

6) Función `E_d`:

- Diseño: como menciona Hake [1] en su paper, es necesario realizar diferentes funciones de costos, una de estas es calcular el costo para las normales. Este se rige por

$$E_d(\hat{Z}) = \sum_p w_p ||u_p||^2 (Z_p - \hat{Z}_p)^2 \qquad (1)$$

donde $u_p = [\frac{-x}{f} \frac{-y}{f} 1]^T$, w_p representa los valores propios de $\sum_q N_q N_q^T$ y donde Z representa las profundidades para los distintos valores de los pixeles.

- Implementación: para este se implementó la ecuación 1, para cada valor de las distintas profundidades, ya que se ingresan las profundidades estimadas y las profundidades calculadas con la Kinect.

7) Función `E_n`:

- Diseño: para el costo de las normales, se utilizo la siguiente formula

$$E_n(\hat{Z}) = \sum_p (N_p T_x)^2 + (N_p T_y)^2, \qquad (2)$$

donde T_x y T_y, representan las derivadas parciales a sus respectivos ejes en Z y donde N_p es el valor de la normal para el píxel p.

- Implementación: para implementar esta función se utilizo la ecuación 2, realizando la sumatoria de todos los pixeles con los valores antes mencionados.

8) Función `E_s`:

- Diseño: del mismo modo que las funciones anteriores este se rige por $\bigtriangledown^2(\hat{Z})$, en cual es el operador Laplaciano para coordenadas cartesianas.
- Implementación:para realizar esto es necesario realizar las deribadas de Z con respecto a los ejes x e y. Para esto se utilizo la función de `sobel` implementada por `openCV`, se realizo dos veces cada derivada a los diferentes ejes y luego estas se sumaron.

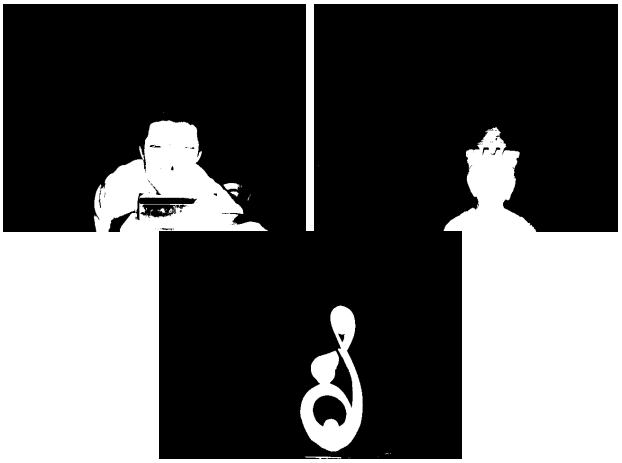

Figura 1. Mascaras auto-generadas para eliminar el fondo.

9) Función `main:` en esta función se utilizan todas las funciones mencionadas anteriormente, el orden de esto generar las mascaras de las imágenes, se elimina el fondo de las imágenes con la mascara ,se calculan las normales, se calculan las profundidades y finalmente se realiza el mesh.

III. Experimentos

Como mencionamos anteriormente, nos basamos en el código de Kai Wolf [3] para obtener las normales y el mapa de profundidad, pero le aplicamos algunos cambios que se reflejaron en reconstrucciones de mejor calidad.

Siguiendo la idea de Hake, decidimos aplicarle un filtro Gaussiano a las imágenes obtenidas con la Kinect, para eliminar ruido y lograr mayor suavidad en las reconstrucciones. Una vez obtenidas las fotos, hicimos el procesamiento de la máscara (Ver figura 1), y le aplicamos un `bitwise_and` a las imágenes con ésta. Éste proceso Wolf no lo hacía, solo aplicaba la máscara a las normales. Hacer este cambio nos permitió eliminar mejor el fondo y trabajar solo con los datos que nos interesaban. En la figura 2, se puede ver una comparación de ambos métodos. Usando las imágenes de ejemplo proporcionadas por Kai Wolf, hicimos la reconstrucción usando ambos algoritmos.

Luego de obtener las imágenes y pre procesarlas, encontramos las normales y el mapa de profundidad. Para calcular las normales, se necesita la suma de las 4 fotos, menos la mascara. A partir de esta imagen se construyen las normales. El mapa de profundidad, se obtiene con la información de la máscara y de la textura de la imagen, para la cual se usa una de las 4 previamente capturadas. La figura 3 y la figura 4, muestran las normales y el mapa de profundidad respectivamente.

Durante el desarrollo del proyecto, tuvimos varios problemas para armar el set-up de Kai Wolf. Esto debido a la dificultad para controlar el ambiente. Uno de los problemas que tuvimos fue la luz infrarroja. La primera que teníamos era muy grande y de luminosidad muy fuerte. Esto producía que el objeto se viera en extremo iluminado y que por ende su mapa de profundidad no sea adecuado. La figura 5, muestra un resultado fallido.

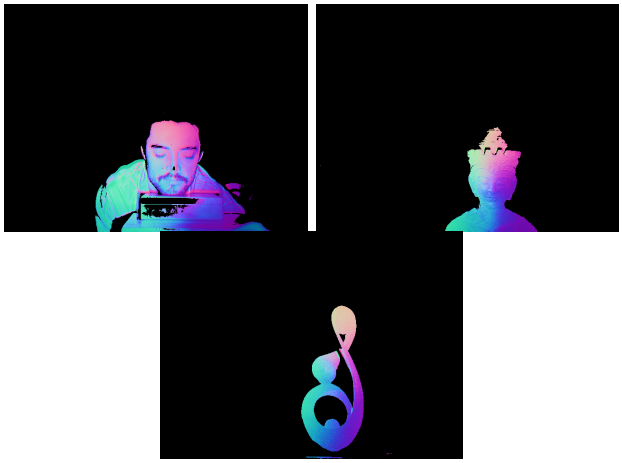

Figura 3. Imágenes que representan las normales.

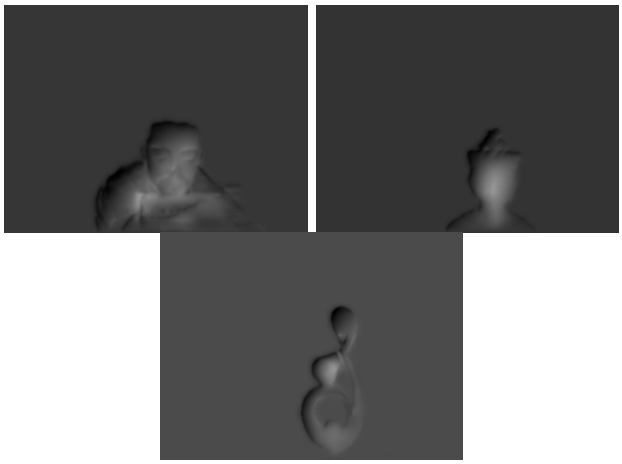

Figura 4. Ejemplos de mapas de profundidad.

A pesar de todo esto, logramos obtener reconstrucciones que definimos como exitosas. Pudimos conseguir una ampolleta más pequeña con componente infrarrojo suficiente para poder captarla con la Kinect. Además, encontramos objetos de prueba que cumplían con las características requeridas por la fotomometría estereo, es decir, eran lambertianos. La figura 6 y 7, muestran estos resultados.

IV. Conclusión

La reconstrucción de objetos en 3D, se hace todo en base a las probabilidades. Hemos notado que la gran mayoría de los métodos utilizados, lo que hacen es optimizar funciones que aproximan la representación del objeto en base a sus imágenes. Identificamos también que el procesamiento de previo que se le aplica a la imagen es de suma importancia, dado que nos permitirá obtener luego una mejor representación.

Luego de realizar una exhaustiva búsqueda de información y analizar los resultados de las reconstrucciones, hemos descubierto que es más complejo poder identificar los detalle de un objeto. También hemos podido observar que este campo ha ido en aumento ya sea por reconstrucciones en objetos en movimiento, de personas, de objetos con superficies no Lambertianas, mejoras de obtención de detalles, entre otros.

(a) Reconstrucción de Wolf. **(b)** Reconstrucción nuestra.

Figura 2. Comparación del algoritmo de Wolf y el nuestro para la reconstrucción.

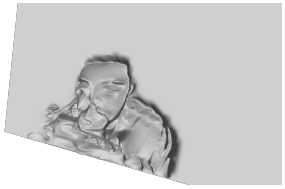

Figura 5. Reconstrucción 3D fallida.

Figura 6. Reconstrucción 3D de una figura de buda.

Es necesario comprender cuales son las variables claves para la obtención de una reconstrucción de alta calidad, ya que la mayor complejidad se encuentra en calcular la estimación de la superficie o profundidad de un objeto.

Se puede apreciar el enfoque de Haque [1], es mejorar la estimación de las profundidades utilizando un mapa de estas entregado por la Kinect y la ecuación de costos para calcularlo. Durante la investigación descubrimos que hay métodos mas robustos que la utilización de la relajación de Gauss-Seidel para las profundidad, algunos de estos son la utilización de Poisson, Multiple Process Unit (MPU), radial basis function (RBF), entre otros. Sin considerar que depende del enfoque que se le desee dar a la reconstrucción, es el método que se seleccionará para obtener la representación $3D$.

Figura 7. Reconstrucción 3D de una figura abstracta ecuatoriana.

REFERENCIAS

[1] S. M. Haque, A. Chatterjee, and V. M. Govindu, "High quality photometric reconstruction from a depth camera," 2014. [Online]. Available: http://www.ee.iisc.ernet.in/labs/cvl/papers/photometric_CVPR2014_paper.pdf

[2] G. Schindler, "Photometric stereo via computer screen lighting for real-time surface reconstruction."

[3] K. Wolf, "Uncalibrated-photometric-stereo." [Online]. Available: https://github.com/NewProggie/Uncalibrated-Photometric-Stereo