



PROYECTO FINAL DE CARRERA
INGENIERÍA EN INFORMÁTICA
UNIVERSIDAD NACIONAL DEL LITORAL

Informe final

Reconstrucción de volúmenes para impresión 3D mediante fusión y rellenado de mallas de superficies parciales

Walter Bedrij

Director: Francisco Mainero
Codirector: Pablo Novara

31 de agosto de 2020

Resumen

Gracias a los avances en las técnicas para determinar la profundidad utilizando sistemas de visión estereoscópica o sistemas de luz estructurada, es posible obtener con gran precisión una nube de puntos que represente la forma, posición y dimensiones de un objeto cualquiera. Sin embargo, esta nube de puntos es parcial, ya que solamente refleja la porción observable desde el punto de captura. Por esta razón, en este proyecto se propone el desarrollo de una biblioteca de software que implemente técnicas para alinear y combinar estas vistas parciales y así obtener un modelo tridimensional de todo el objeto, posibilitando su posterior construcción mediante técnicas de impresión 3D.

Palabras clave: fusión de mallas, imágenes de profundidad, impresión 3D, reconstrucción 3D.

Índice general

Resumen	i
1 Introducción	1
1 Justificación	1
2 Objetivos	4
2.1 General	4
2.2 Específicos	4
3 Alcance	4
2 Reconstrucción tridimensional	5
1 Adquisición	5
1.1 Ruido de adquisición	7
2 Registración	7
2.1 Perturbaciones pequeñas: algoritmo iterativo del punto más cercano (ICP)	8
2.2 Distancias mayores: búsqueda de correspondencias mediante vectores de características	9
2.3 Registración de múltiples capturas	14
3 Determinación de la superficie	14
3.1 <i>Mesh zippering</i>	15
3.2 Fusión volumétrica (<i>volumetric merge</i>)	15
3.3 Representación de surfel	15
4 Rellenado de huecos	16
4.1 Identificación de los huecos	16
4.2 Advancing front	16
4.3 Reconstrucción de Poisson	17
3 Herramientas	19
1 Base de datos	19
2 Tecnologías	20
2.1 KinectFusion	20
2.2 Open Source Computer Vision Library (OpenCV)	20
2.3 <i>The Point Cloud Library</i> (PCL)	21
2.4 CloudCompare, Meshlab	21
2.5 Otras herramientas de software	21

4 Diseño	22
1 Casos de uso	22
2 Especificación de requerimientos	25
2.1 Descripción	25
2.2 Requerimientos funcionales	25
2.3 Requerimientos no funcionales	26
3 Diagrama de clases	26
5 Desarrollo	28
1 Módulo de preprocesso	28
2 Módulo de registración	30
2.1 Alineación mediante <i>sample consensus</i>	30
2.2 Alineación mediante búsqueda de clúster	33
2.3 Refinamiento	38
3 Módulo de fusión	40
3.1 Método	40
4 Módulo de relleno de huecos	41
4.1 Advancing front	42
4.2 Reconstrucción de Poisson	42
6 Pruebas y resultados	45
1 Módulo de registración	45
2 Módulo de fusión	47
3 Módulo de relleno de huecos	49
3.1 Método de advancing front	49
3.2 Reconstrucción de Poisson	49
4 Tiempo de ejecución	50
7 Conclusiones y trabajos futuros	55
1 Conclusiones del producto	55
2 Conclusiones del proceso	55
3 Trabajos futuros	56
A Uso de la biblioteca	60
1 Instalación y compilación	60
2 Representación de las capturas	60
3 Módulo de preprocesso	61
4 Módulo de registración	61
5 Módulo de fusión	62
6 Rellenado de huecos	62
7 Reconstrucción de Poisson	62

Índice de figuras

1.1	Reconstrucción tridimensional. A partir de un objeto (1.1a) se obtiene su modelo geométrico (1.1b) para lograr su posterior impresión 3D (1.1c).	1
1.2	Medición de las características geométricas de los objetos mediante múltiples capturas utilizando cámaras de profundidad.	3
2.1	Proceso de reconstrucción. De izquierda a derecha y de arriba a abajo se tiene: objeto a reconstruir, adquisición de capturas parciales, registración de cada captura a un marco de referencia global, determinación de la superficie, rellenado de huecos.	6
2.2	Error debido a un reflejo parcial	7
2.3	Medidas de error punto-a-plano entre dos superficies	9
2.4	Matriz de confusión del descriptor FPFH	10
2.5	Representación gráfica del marco de Darboux	12
2.6	Representación gráfica de los ángulos ϕ y θ calculados por el descriptor PFH	12
2.7	Visualización del error de bucle	14
2.8	Ilustración del método de reconstrucción de Poisson	17
2.9	Reconstrucción del modelo Ángel usando condiciones de borde Dirichlet y Neumann	18
3.1	Modelos de la base de datos Stanford	20
4.1	Diagrama de casos de uso	22
4.2	Diagrama de clases del sistema	27
5.1	Proceso de reconstrucción tridimensional	28
5.2	Visualización de la curvatura	29
5.3	Diagrama de flujo de la registración	31
5.4	Visualización de los keypoints	32
5.5	Diferencias en la rotación estimada para el objeto <i>happy stand</i> (SAC-IA)	34
5.6	Fallo en el algoritmo de <i>sample consensus</i>	34
5.7	Marcos de referencia	35
5.8	Correspondencias supervivientes luego de realizar el descarte según la transformación estimada por cada marco de referencia.	36
5.9	Estimación de la transformación de alineación mediante los marcos de referencia	37
5.10	Diferencias en la rotación estimada para el objeto <i>happy</i>	38

ÍNDICE DE FIGURAS

v

5.11	Diferencias en la rotación estimada para el objeto <code>bunny</code>	39
5.12	Visualización de los valores de confianza de los súrfeles	40
5.13	Superficie reconstruida	42
5.14	Creación de los nuevos triángulos mediante un algoritmo de advancing front	43
5.15	División del frente en dos	43
5.16	Reconstrucción de la superficie mediante el método de <i>Poisson</i> . .	44
6.1	Comparación entre las transformaciones de alineación. Se observa el efecto producido en un punto orientado C que simula la posición de la cámara.	46
6.2	Métrica de alineación para el modelo <code>dragon stand</code>	47
6.3	Medida de error en la fusión	48
6.4	Relleno de un hueco pequeño mediante <i>advancing front</i>	49
6.5	Fallo en el algoritmo de <i>advancing front</i>	49
6.6	Resultado de las reconstrucciones	50
6.7	Acercamiento a la oreja derecha de <code>bunny</code>	51
6.8	Acercamiento a la mecha de <code>drill</code>	51
6.9	Acercamiento al vientre de <code>dragon</code>	51
6.10	Acercamiento a la base de apoyo de <code>armadillo</code>	52
6.11	Tiempo de ejecución de la registración, ajustado a una función de orden $\mathcal{O}(n^2)$	54

Introducción

1 Justificación

La impresión 3D es un método de fabricación en el cual se deposita el material (como por ejemplo, plástico o metal) en capas para producir un objeto tridimensional. Avances en los materiales utilizados han permitido la creación de objetos de calidad comparable a la de métodos tradicionales, demostrando su viabilidad para muchas aplicaciones médicas, incluyendo la creación de prótesis e implantes dentales[SvLD14]. Los modelos geométricos que desean imprimirse pueden obtenerse aplicando procesos de reconstrucción tridimensional (figura 1.1) a objetos ya existentes.

Entre las aplicaciones de la reconstrucción tridimensional podemos nombrar:

- Reconstrucción de órganos para la planificación de cirugías y la realización de prácticas por parte de estudiantes de ciencias médicas y veterinarias. En particular, la empresa argentina *mirai*¹ realiza estos biomodelos basándose en imágenes de resonancia magnética y tomografía computarizada.

¹<https://www.modelosmedicos.com/>



(a) Objeto



(b) Modelo geométrico



(c) Impresión 3D

Figura 1.1: Reconstrucción tridimensional. A partir de un objeto (1.1a) se obtiene su modelo geométrico (1.1b) para lograr su posterior impresión 3D (1.1c).



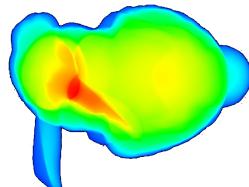
- Digitalización de obras de arte para salvaguardar el patrimonio cultural, ayudar en su restauración e incrementar la accesibilidad a las mismas. A este respecto, puede mencionarse el *Digital Michelangelo Project* de la Universidad de Stanford, que tiene como objetivo crear un repositorio de modelos 3D de alta calidad de las esculturas y la arquitectura de Miguel Ángel.



- Realización de prototipos y modificaciones a productos.



- Simulación de propiedades físicas sobre los objetos.



- Digitalización de ambientes y objetos para utilizarlos en aplicaciones de realidad virtual o de diseño.



Uno de los métodos para la obtención de los modelos geométricos se basa en la utilización de dispositivos de luz estructurada, como ser el dispositivo Kinect[MG], para realizar mediciones sobre el objeto. Debido a que los puntos capturados sólo reflejan la porción de la superficie del objeto visible desde

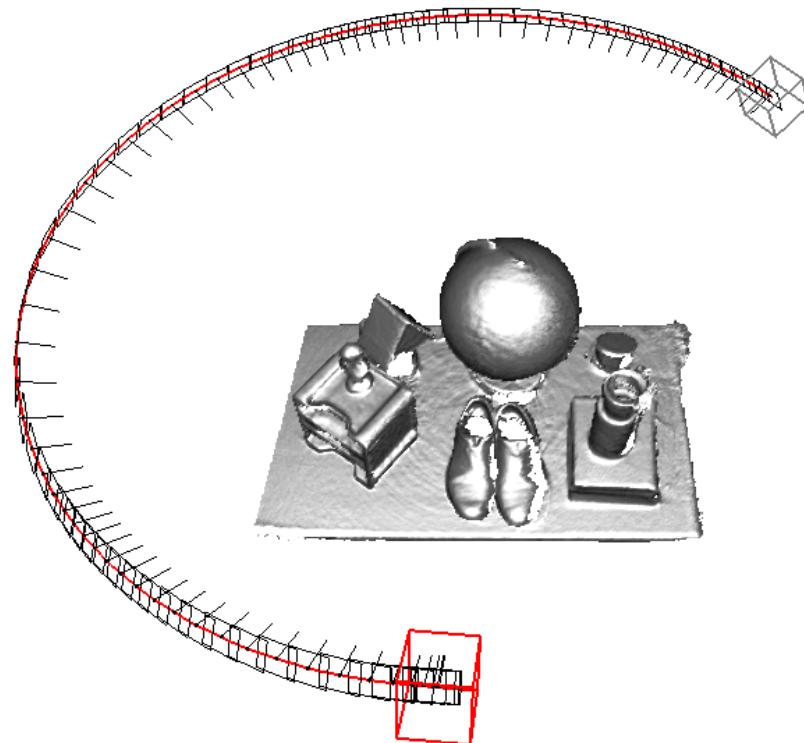


Figura 1.2: Medición de las características geométricas de los objetos mediante múltiples capturas utilizando cámaras de profundidad.

la cámara y a que la propia geometría del objeto podría generar obstrucciones, reflejos o sombras que imposibilitan la adquisición de datos en esas zonas («huecos»), es necesario combinar múltiples capturas del objeto en diferentes orientaciones respecto a la cámara, de modo que cada parte de la superficie del objeto esté representada en al menos una captura (figura 1.2).

El principal problema para combinar las múltiples vistas es encontrar las transformaciones de rotación y translación que permitan relacionar cada vista parcial, para luego fusionarlas y así reconstruir el objeto.

Para esto existen métodos automáticos que requieren que la posición del dispositivo de captura entre las distintas capturas no haya variado de forma significativa[BM92], lo cual genera que el proceso de adquisición de datos sea engorroso, requiriendo de una supervisión constante para asegurar que el movimiento del dispositivo de captura no sea excesivo y donde un fallo puede exigir el reinicio del proceso. Este es el caso de *KinectFusion*, un algoritmo desarrollado por Microsoft que utiliza el sensor Kinect para reconstruir escenas tridimensionales en tiempo real, pero que presenta las siguientes limitaciones:

- utiliza un alto costo computacional, requiriendo de una poderosa GPU;
- los movimientos de cámara deben ser lentos y pequeños;
- está limitado al dispositivo Kinect;
- algunos objetos podrían no aparecer en el mapa de profundidad debido a que absorben o reflejan demasiada luz infrarroja;

- no utiliza información de color.[PL14]

Existen, además, métodos manuales o semiautomáticos donde se permiten distancias mayores entre las posiciones del dispositivo de captura. Por ejemplo, el usuario posiciona de forma interactiva las capturas[TL94] o selecciona iterativamente puntos coincidentes entre ellas[CCC⁺08].

Con el fin de evitar o limitar el uso de los recursos humanos, en este proyecto se analizarán técnicas de fusión de mallas, obtenidas permitiendo mayor distancia entre capturas, para lograr un modelo tridimensional de un objeto que permita su posterior replicación mediante una impresora 3D.

2 Objetivos

2.1 General

Diseñar y desarrollar una biblioteca de funciones que permita la reconstrucción de una malla 3D cerrada a partir de mallas de superficies parciales.

2.2 Específicos

- Desarrollar algoritmos que determinen las posiciones relativas de cada malla parcial.
- Desarrollar algoritmos de fusión de las mallas.
- Desarrollar algoritmos para relleno de huecos producto de obstrucciones.
- Evaluar el desempeño de cada bloque desarrollado y realizar ajustes.

3 Alcance

- Para posibilitar la posterior impresión 3D, la malla resultante deberá ser una malla cerrada (*watertight*).
- No se impondrán topologías específicas a los objetos a reconstruir.
- Los algoritmos desarrollados deberán ser robustos frente al ruido producido por los sensores.
- No se pretenderá el funcionamiento de los algoritmos en tiempo real.
- La biblioteca será de código libre y multiplataforma.

Reconstrucción tridimensional

La reconstrucción tridimensional es un proceso por el cual se combinan diversas mediciones de un objeto para obtener un modelo tridimensional que lo reproduzca fielmente. Este proceso puede dividirse en las siguientes etapas (figura 2.1):

1. Adquisición: en esta etapa se realizarán las mediciones del objeto, obteniéndose las posiciones $\{x, y, z\}$ de puntos muestrados sobre su superficie.
2. Registración: en caso de que el método de adquisición no capture la totalidad del objeto, esta etapa establecerá las relaciones entre las medidas parciales para ajustarlas a un mismo marco de referencia.
3. Determinación de la superficie: en esta etapa se estimará la superficie del objeto a partir de los puntos muestrados.
4. Rellenado de huecos: es posible que existan zonas donde no se hayan obtenido muestras y, por lo tanto, la superficie presente huecos. Dependiendo de la aplicación, puede ser necesario llenar estos huecos estimando nuevos puntos a partir de los lindantes.

A continuación se describen con más detalle estas etapas y los métodos utilizados en las mismas.

1 Adquisición

Los distintos métodos de adquisición se diferencian de acuerdo al fenómeno físico de interacción con la superficie del objeto de interés. De esta manera, se pueden clasificar como:

- Métodos táctiles o de contacto, donde sensores en las articulaciones de un brazo robótico determinan las coordenadas relativas de la superficie. Estos son de los más robustos, introduciendo poco ruido, pero también de los más lentos y suelen tener problemas con superficies cóncavas.
- Métodos de no-contacto, donde se utiliza luz (métodos ópticos), sonido u ondas electromagnéticas.

En particular, los métodos ópticos son los más populares con una rápida velocidad de adquisición[VMC97]. Dentro de los métodos ópticos podemos distinguir:

- Métodos pasivos o de visión estereoscópica, que utilizan dos o más cámaras y buscan correspondencias entre los puntos de la escena capturados en cada fotografía.
- Métodos activos o de luz estructurada, que proyectan patrones de luces conocidos sobre la escena de modo de analizar sus deformaciones.

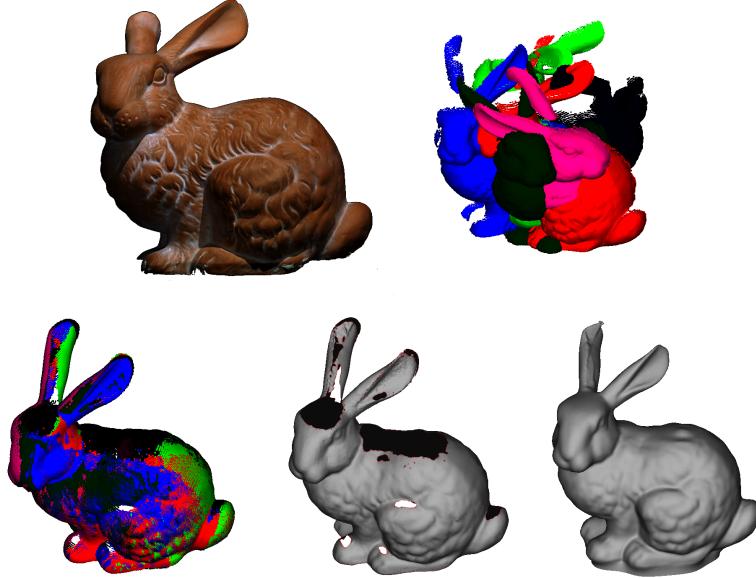


Figura 2.1: Proceso de reconstrucción. De izquierda a derecha y de arriba a abajo se tiene: objeto a reconstruir, adquisición de capturas parciales, registración de cada captura a un marco de referencia global, determinación de la superficie, relleno de huecos.

Identificar los patrones lumínicos es un problema más simple que el de encontrar las correspondencias entre imágenes, por lo que los métodos de luz estructuradas resultan más rápidos y robustos[Mai19].

A partir de los métodos ópticos se obtendrá como resultado una imagen de profundidad, donde el valor de cada píxel representa la distancia entre el punto observado y la cámara, lo cual se conoce como imagen 2.5D. Conociendo los parámetros intrínsecos K de la cámara, puede obtenerse para cada píxel $\{X, Y, d\}$ sus coordenadas $\{x, y, z\}$ en el espacio:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = K^{-1} \begin{pmatrix} X \\ Y \\ d \end{pmatrix}$$

Estas son coordenadas locales, definidas en un sistema de referencia centrado en la cámara.

Se tendrá entonces una *nube de puntos*, es decir, una colección de puntos sin información de conectividad entre ellos. Esta nube de puntos será una representación parcial de la superficie del objeto, ya que se corresponderá únicamente con la porción observable desde la cámara.

Si bien este proyecto no abarca la etapa de adquisición, no es posible ignorar el método de adquisición, ya que este influye en las características de la nube de puntos resultante. Por esta razón, se supondrá que las capturas provienen de métodos ópticos de luz estructurada.

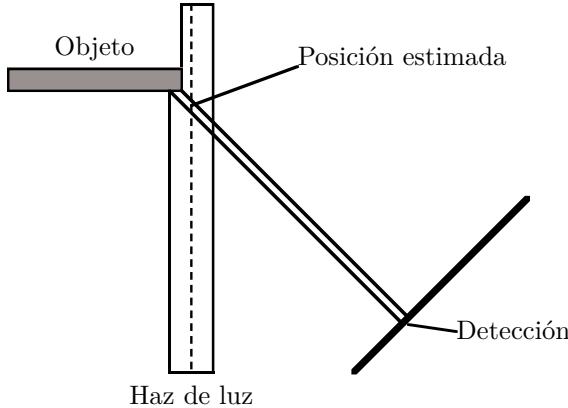


Figura 2.2: Error en la posición estimada debido a un reflejo parcial del patrón de luz.

1.1 Ruido de adquisición

Es imposible obtener la posición exacta de los puntos muestreados sobre la superficie del objeto. Se tendrán distorsiones debidas a particularidades de la escena, del objeto, de los dispositivos utilizados y del método empleado. En los escáneres de luz estructurada dos fuentes de error son particularmente relevantes:

- *Ángulo de incidencia excesivo:* El rayo de luz proyectado impacta en una porción de la superficie del objeto que es casi paralela a su dirección. Entonces, el sensor capta una versión estirada y con menor intensidad del patrón, lo que agrega incertidumbre en la posición de los puntos.
- *Reflejo parcial:* Se produce cuando una porción del patrón no incide completamente en el objeto (figura 2.2). Debido a que el método de triangulación supone que todo el ancho de la línea impactó en el objeto, se estima una posición incorrecta del punto muestreado. Esto resulta en bordes distorsionados y en posiciones más alejadas que las reales.[TL94]

2 Registración

Una vez finalizada la etapa de adquisición, se dispone de una nube de puntos que representa la porción observada del objeto. Para lograr una reconstrucción total es necesario combinar múltiples capturas variando la posición relativa cámara-objeto. Contar con un dispositivo que nos permita definir con precisión tanto la posición como orientación de la cámara es altamente costoso, por esta razón, en la etapa de registración se deben estimar las transformaciones que ubiquen cada captura en un sistema de referencia global, de forma que las zonas comunes encajen perfectamente.

Esta operación se realiza usualmente entre dos capturas, buscando las transformaciones de translación y rotación que ubiquen la captura de partida en el marco de referencia de la captura objetivo. De esta manera, el problema general cuenta con seis grados de libertad. A continuación se presenta el caso en que la transformación de alineación es pequeña, más adelante se discute el ca-

so de incrementos mayores, y finalmente se extiende el proceso para abarcar n capturas.

2.1 Perturbaciones pequeñas: algoritmo iterativo del punto más cercano (ICP)

Antes de describir este algoritmo, se planteará una versión simplificada del problema que permitirá establecer ciertas definiciones.

Se supone que se dispone de una nube A a la cual se le aplica una transformación de translación y rotación arbitraria, y luego se perturba levemente las posiciones de los puntos transformados, obteniéndose la nube B . Se observa que ambas nubes poseen la misma cardinalidad y se pueden establecer las relaciones de origen a destino $a_j \rightarrow b_j$ para cada punto $a_j \in A$. Entonces, se puede definir el error de alineación como:

$$\text{Error} = \frac{1}{N} \sum_{j=1}^n \|b_j - T(a_j)\|$$

y buscar la transformación T que minimice este error.

Existe una solución cerrada para este problema, obteniéndose la rotación al calcular los eigenvectores de una matriz simétrica de 4×4 , cuyos detalles pueden consultarse en [Hor87].

Sin embargo, estas suposiciones son demasiado fuertes. Al trabajar con dos capturas realizadas desde distintas vistas, el solapamiento no será total, ya que habrá puntos que serán observables en tan sólo uno de las vistas. Por lo tanto, primero es necesario determinar cuáles son los puntos comunes a ambas capturas, y, además, definir las relaciones de origen-destino entre esos puntos comunes. El algoritmo iterativo del punto más cercano (ICP) supone que las nubes se encuentran lo suficientemente cerca como para establecer estas correspondencias mediante las coordenadas de los puntos. El algoritmo se describe de la siguiente manera:

1. Obtener las correspondencias: Para cada punto $a_j \in A$ buscar el punto más cercano en la nube B . Si la distancia entre esos puntos supera un umbral d_{\max} , se considera que a_j no pertenece a la zona común y no es considerado en esta iteración.
2. Calcular la transformación que alinee los pares de puntos de la zona común.
3. Aplicar la transformación.
4. Repetir el proceso hasta que el error de alineación esté por debajo de un umbral τ .

Si bien el algoritmo converge a un mínimo local, puede que este no sea el mínimo global buscado. Debe cumplirse la suposición de cercanía para obtener una correcta registración.[BM92]

En [CM92] se presenta una variante del algoritmo de ICP, llamada ICP punto-a-plano, que resulta más robusta y precisa al trabajar con nubes de puntos en 2.5D. Una vez establecida una correspondencia $a_j \rightarrow b_j$, si bien no se trata del mismo punto, se realiza la suposición de que pertenecen al mismo plano. Por esta razón, podemos decir que conocemos con mucha confianza la posición del punto a lo largo de su normal, pero no su ubicación en el plano. Para reflejar esto, se cambia la métrica del error, proyectando el punto transformado sobre

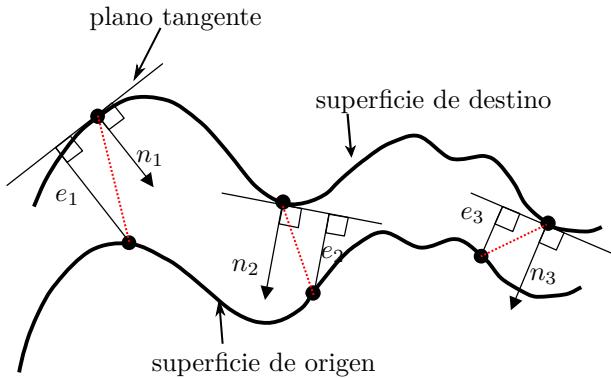


Figura 2.3: Medidas de error punto-a-plano entre dos superficies (Imagen cortesía de [Low04]).

la normal del punto destino (figura 2.3).

$$\text{Error} = \frac{1}{N} \sum_{j=1}^n \|n_j \cdot (b_j - T(a_j))\|$$

2.2 Distancias mayores: búsqueda de correspondencias mediante vectores de características

Cuando la transformación de alineación requerida para alinear las dos nubes de puntos no es lo suficientemente pequeña, no se obtiene una buena aproximación al establecer las correspondencias utilizando únicamente las posiciones de los puntos. Una posibilidad es emplear métodos semiautomáticos, donde el usuario provee de una alineación inicial de las nubes o determina las correspondencias seleccionando puntos. Sin embargo, este es un método lento y engorroso, por lo que surge la necesidad de plantear un algoritmo completamente automático.

Para hallar las correspondencias de manera automática, se calcula un vector de características o *descriptor* sobre una vecindad de cada punto. Al comparar los descriptores con alguna medida de distancia, si la distancia es pequeña (cercana a 0), entonces las vecindades son similares y podría plantearse una correspondencia entre los puntos. Un buen descriptor deberá poseer las siguientes características:

- Ser discriminante, es decir, la distancia entre los descriptores debe reflejar la distancia entre las superficies de las vecindades. Esto implica que superficies similares localmente producen descriptores parecidos y en superficies diferentes la distancia entre descriptores será grande (figura 2.4).
- Ser invariante respecto a translaciones y rotaciones.
- Ser robusto respecto al ruido de muestreo.
- Ser robusto respecto a la densidad de muestreo.
- Ser robusto respecto a la ausencia de muestras debido a occlusiones.

Además de la elección del descriptor, un parámetro importante a determinar es el tamaño de la vecindad que éste representará. En general, la vecindad de

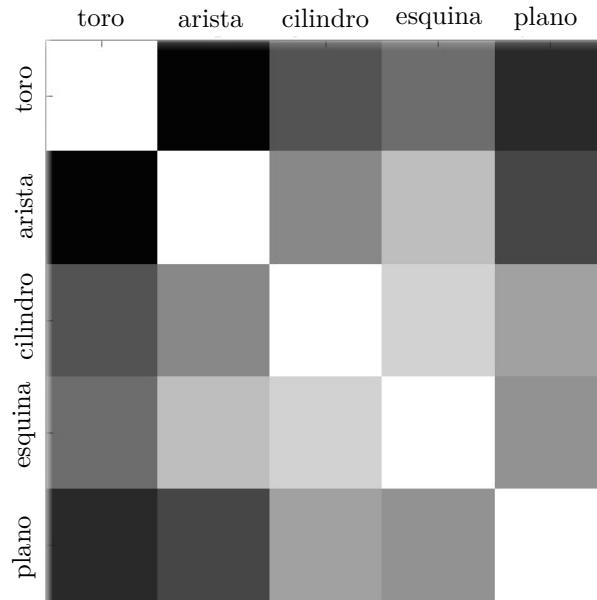


Figura 2.4: Matriz de confusión del descriptor FPFH para distintas clases de superficies utilizando la distancia χ^2 (Imagen cortesía de [Rus09]).

un punto p incluye a todos aquellos puntos q que se encuentran dentro de una esfera de radio r centrada en p :

$$|p - q| \leq r$$

Si el valor de r es demasiado pequeño, el descriptor perderá su poder discriminante o incluso no podrá calcularse al no contar con la cantidad de vecinos mínimos necesarios. En cambio, si el valor de r es demasiado grande, se corre el riesgo de considerar dentro de la vecindad puntos pertenecientes a otra superficie, distorsionando el valor del descriptor. La determinación de un valor adecuado de r limita la posibilidad de obtener un método completamente automático para el cálculo de los descriptores[Rus09].

A continuación se describe el proceso de construcción de dos descriptores que han mostrado buenos resultados en problemas de registración[RBB09].

Estimación de normales

Para poder describir la geometría de una superficie, primero debe estimarse su orientación en el sistema de coordenadas, es decir, estimar su normal. Una opción es aproximar la normal mediante la estimación de un plano tangente a la superficie, lo que transforma el problema a ajustar un plano a la vecindad de p mediante el método de mínimos cuadrados. Representando el plano mediante un punto x_j y un vector normal n_j , la distancia de un punto q de la vecindad a este plano queda definida como $d = (q_j - x_j)n_j$. Para minimizar la sumatoria

del cuadrado de todas las distancias, se tiene que:

$$x_j = \frac{1}{N} \sum_{k=1}^N q_j^{(k)}$$

para los N puntos de la vecindad de p . Y para obtener n_j se analizan los eigenvalores y eigenvectores de la matriz de covarianza de la vecindad:

$$C_{jk} = \sum_{r=1}^N N(q_j^{(r)} - x_j)(q_k^{(r)} - x_k)$$

Se observa que C_{jk} es una matriz simétrica y por lo tanto todos sus eigenvalores son reales $\lambda_j \in \mathbb{R}$. El eigenvector que corresponde al eigenvalor de menor magnitud corresponde a la normal del plano tangente, es decir a $+n_j$ o a $-n_j$ [BC94]. Si bien no puede determinarse matemáticamente el signo de n_j , puede resolverse la ambigüedad al considerar que la nube de puntos que se obtiene del proceso de adquisición es 2.5D y se conoce la ubicación de la cámara v_j en un sistema de referencia local. Por lo tanto, debe cumplirse que:

$$n_j v_j > 1$$

invirtiendo el signo de n_j en caso de ser necesario.

Histograma de características del punto (PFH)

Dado que las normales no son invariantes a rotaciones y no capturan mucho detalle de la superficie, no presentan un gran poder discriminante. Por esta razón, no resulta adecuado utilizarlas como único descriptor. Sin embargo, al representar las relaciones entre todos los puntos de la vecindad y sus normales, pueden capturarse con más detalle las variaciones de la superficie, aumentando la representatividad del descriptor. Esta es la idea principal del histograma de características del punto (PFH) presentado en [Rus09].

El vector de características de PFH es un histograma de los valores de tres ángulos y una distancia calculados para cada par de puntos y sus normales en la vecindad de p . Para asegurar la replicabilidad del descriptor, se define un marco de referencia de Darboux determinado únicamente de la siguiente manera:

- Dado el par de puntos p_j y p_k , se determina el origen p_s del marco de referencia como:

$$\vec{n}_j \cdot (p_k - p_j) > \vec{n}_k \cdot (p_j - p_k) \begin{cases} \text{Verdadero:} & p_s = p_j \quad p_t = p_k \\ \text{Falso:} & p_s = p_k \quad p_t = p_j \end{cases}$$

- A partir de este origen, se determinan los tres vectores que definen el marco de referencia:

$$\begin{cases} u = n_s \\ v = u \times \frac{p_t - p_s}{|p_t - p_s|} \\ w = u \times v \end{cases}$$

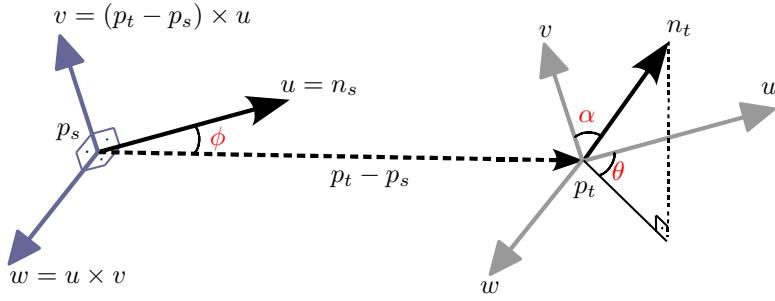


Figura 2.5: Representación gráfica del marco de Darboux y las características calculadas por el descriptor PFH (Imagen cortesía de [Rus09]).

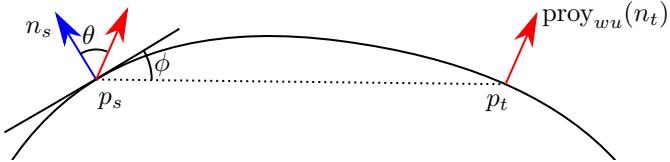


Figura 2.6: Representación gráfica de los ángulos ϕ y θ calculados por el descriptor PFH.

Y entonces se procede a calcular las siguientes relaciones para estos puntos:

$$\begin{aligned} d &= |p_t - p_s| \\ \alpha &= v \cdot n_t \\ \phi &= u \cdot \frac{p_t - p_s}{d} \\ \theta &= \text{atan}(w \cdot n_t, u \cdot n_t) \end{aligned}$$

Sin embargo, la distancia entre puntos d no es de importancia para escaneos 2.5D, ya que la distancia entre vecinos se incrementa con la distancia a la cámara, y suele ser beneficioso omitirla en estos casos [Rus09].

Para entender el poder discriminante del descriptor PFH, es preciso comprender el significado de los ángulos calculados entre los pares de puntos (ver figura 2.6). Considerando el plano tangente a p_s , el ángulo ϕ mide la separación de la posición de p_t respecto a este plano. Luego, considerando el plano wu , donde se encuentran p_s , p_t y n_s , y cuya normal es v :

- el ángulo α mide el desvío de la normal n_t respecto a este plano,
- el ángulo θ mide la distancia entre las proyecciones de las normales sobre este plano.

De esta manera, se logra describir con buen detalle las características de la superficie en la vecindad de cada punto.

PFH rápido (FPFH)

Debido a que para obtener el descriptor PFH de un punto es necesario calcular las relaciones entre todos los pares de puntos de su vecindad, el orden de ejecución para una nube con n puntos es $\mathcal{O}(nk^2)$, donde k es la cantidad

de puntos en la vecindad, y por esta razón el cálculo de este descriptor puede convertirse en uno de los cuellos de botella del proceso.

Para solventar este problema, en [RBB09] se propone una simplificación del descriptor PFH, llamada PFH rápido o simplemente FPFH, que nos permite reducir el tiempo de ejecución a $\mathcal{O}(nk)$ manteniendo el poder descriptivo de PFH.

El cálculo de este nuevo descriptor se realiza de la siguiente manera:

1. Calcular por cada punto p y sus vecinos las tuplas $\langle\alpha, \phi, \theta\rangle$. Llamamos a estas características PFH simplificado (SPFH).
2. Realizar un promedio ponderado de los valores del SPFH de p y sus vecinos:

$$FPFH(p) = SPFH(p) + \frac{1}{N} \sum_{j=1}^N \frac{1}{w_j} SPFH(p_j)$$

donde w_j es una medida de distancia entre p y su vecino.

Las principales diferencias respecto a PFH son:

- FPFH no interconecta todos los vecinos de p .
- FPFH puede incluir puntos fuera del radio r de vecindad, pero a no más de $2r$.

Puntos salientes (*keypoints*)

Al describir las características de un buen descriptor, mencionamos la capacidad discriminante del mismo. Se presenta una gran diferencia entre descriptores de superficies diferentes, pero la distancia es cercana a 0 en superficies parecidas, y de esta forma se pueden establecer las correspondencias entre los puntos de dos nubes al comparar sus descriptores.

Al analizar una zona homogénea en una nube, como, por ejemplo, un plano, todos los puntos presentarán descriptores parecidos. Surgen problemas al intentar establecer una correspondencia entre estos puntos y los presentes en la otra nube, ya que todos son candidatos igualmente válidos.

Para solucionar este problema, es necesario identificar puntos salientes (*keypoints*) en la nube, para entonces establecer las correspondencias entre los descriptores de estos keypoints. Si bien un punto cualquiera será considerado como saliente o no dependiendo del detector utilizado, un buen detector presentará las siguientes características:

- Dispersión: un subconjunto pequeño de los puntos de la nube serán considerados como keypoints.
- Repetibilidad: un keypoint deberá ser detectado en la misma ubicación sobre el objeto sin importar la posición de la cámara al realizar la captura.
- Distinguibilidad: la superficie en la vecindad del keypoint presentará características únicas que serán capturadas por un descriptor.

Como ejemplos de detectores se puede mencionar el detector de esquinas Harris, utilizado usualmente en imágenes 2D, adaptado al espacio tridimensional traduciendo cambios en la intensidad de píxeles por ángulos entre las normales de puntos vecinos; y el detector ISS, que analiza los eigenvalores de la matriz de covarianza en la vecindad de un punto.

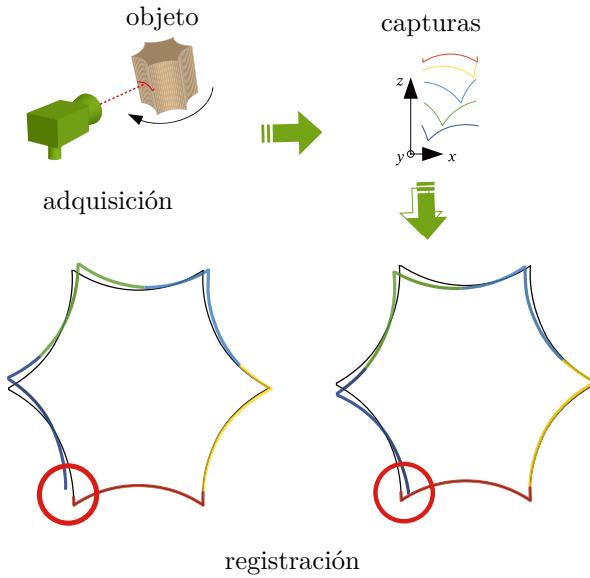


Figura 2.7: Visualización del error de bucle. Errores de tan sólo 1° en cada registración producen una discrepancia considerable donde el modelo debería cerrarse (círculo rojo) (Imagen cortesía de [WWLV09]).

2.3 Registración de múltiples capturas

Previamente se describieron métodos de registración entre dos capturas, sin embargo, un objeto difícilmente pueda ser observable completamente desde tan sólo dos posiciones. Por lo tanto, para lograr la reconstrucción tridimensional se tendrá una lista de capturas A_1, A_2, \dots, A_n correspondientes a distintas posiciones relativas cámara-objeto. Surge entonces el problema de cómo obtener la lista de transformaciones T_1, T_2, \dots, T_n que las alineen a todas correctamente.

Una opción es utilizar una captura B que presente zonas comunes con todas las capturas A_j y entonces calcular la registración $A_j \rightarrow B$. Esta captura B puede obtenerse mediante un escaneo cilíndrico: el objeto se ubica sobre una superficie que gira lentamente a una velocidad controlada mientras que se proyecta una línea vertical sobre el objeto y se miden las distancias de los puntos que caen sobre esta línea.

Una alternativa es realizar la registración entre pares sucesivos $A_2 \rightarrow A_1, A_3 \rightarrow A_2, \dots, A_n \rightarrow A_{n-1}$. Sin embargo, los errores producidos se propagarán con cada captura incorporada, siendo especialmente apreciables al completar una vuelta alrededor del objeto (figura 2.7). Para corregir este error, se ajustará la alineación de la captura que cierre el bucle, propagando luego esta corrección a las demás.

3 Determinación de la superficie

Una vez alineadas todas las capturas en un mismo sistema de referencia, debemos combinar estas nubes de puntos para obtener un único modelo que describa la geometría del objeto escaneado. Cada captura presentará nueva in-

formación en las zonas que no pudieron ser observadas desde otras posiciones, y confirmará o refutará la información ya presente en las zonas comunes a otras capturas.

A continuación se presentarán métodos para lograr la integración de las distintas capturas.

3.1 *Mesh zippering*

Este método combina dos mallas poligonales en una única superficie, obteniéndose el objeto completo al agregar las capturas una a una a la superficie final. El proceso consta de los siguientes pasos:

1. Triangulación de la captura. Debido a que este método trabaja con mallas, cada nube de puntos debe ser primero triangulada.
2. Eliminación de superficies redundantes. Se eliminan los triángulos de ambas mallas que se encuentren completamente dentro del área solapada. De esta manera, los únicos triángulos que presentarán un solapamiento serán aquellos pertenecientes a los bordes de las mallas.
3. *Clipping*. Los triángulos de la malla A se recortan por el borde de la malla B. Es decir, se eliminan las porciones de los triángulos que caen dentro de la otra malla.

De esta forma se obtiene una malla triangular que representa burdamente la topología del objeto. Para ajustar los detalles de la superficie, se realiza un proceso de refinamiento, donde cada vértice de la malla final es perturbado por un promedio ponderado de los vértices de los triángulos que caen cerca de él en las capturas originales[TL94].

3.2 Fusión volumétrica (*volumetric merge*)

En este método, cada captura se representa de forma volumétrica, definiendo un volumen 3D que mapea las coordenadas $\{x, y, z\}$ de los puntos a posiciones $\{w, h, d\}$ de una grilla de vóxeles. En cada uno de estos vóxeles se codifica una función de distancia a la superficie, siendo positiva si el punto se encuentra adelante de la superficie, y negativa si se encuentra después. De esta manera, la superficie queda definida de forma implícita en el cruce por cero, donde los valores cambian de signo, y debe ser extraída mediante un algoritmo de *marching cubes* o similar.

En cada nueva registración se actualizan las distancias de los vóxeles mediante raycasting, desde la posición estimada de la cámara hacia la superficie, realizando un promedio ponderado con los valores anteriormente calculados.[CL96] Debido a que sólo nos interesan las zonas donde la distancia es cercana a 0 es posible utilizar una representación rala para los vóxeles.[SKC13]

3.3 Representación de surfel

En este método, la superficie se representa como un conjunto de discos orientados o elementos de superficie (*surfels*). Cada uno de estos *surfels* posee una posición p_j , un vector normal n_j , un radio r y una medida de confianza de visibilidad v , que nos indica cuántas capturas confirmaron las características del *surfel*. Esta representación facilita la actualización, agregado o eliminación de vértices y además nos permite detectar ciertos puntos atípicos[WWLV09].

El modelo se actualiza con cada nueva captura, comparándose la nueva información con la ya existente en el modelo y realizando las operaciones correspondientes:

- Se crean nuevos *surfels* para las porciones que el modelo actual no puede explicar.
- Se ajustan las posiciones y confianza de aquellos que están en concordan-
cia.
- Se eliminan los *surfels* con baja confianza y que presentan conflictos con
la nueva información.

Al finalizar el proceso, se utilizan las posiciones y normales de los *surfels* para realizar una triangulación y así obtener una malla que represente al objeto.

4 Rellenado de huecos

Los huecos son regiones que ninguna de las vistas logró capturar, por lo que se carece de información de la superficie en esas zonas. Si una malla presenta huecos, entonces no es cerrada, no puede definirse un interior y un exterior y no es posible su posterior impresión 3D. Por esta razón, estos huecos deben ser llenados, estimando los valores de la superficie en los mismos.

Las características de los huecos dependerán no sólo de la geometría del objeto y las posiciones desde donde se realizaron las capturas, sino también del método usado para integrar las capturas.

A continuación se presentan métodos para el llenado los huecos de forma de conseguir una malla cerrada.

4.1 Identificación de los huecos

Antes de poder llenar cada hueco es necesario identificar los vértices y aristas que los delimitan. Este proceso se facilita al utilizar una representación de lista de medias aristas (DCEL) para describir la malla.

En esta representación, se almacena cada cara, arista y vértice de la malla. Cada arista se divide en dos medias aristas gemelas, que comparten los mismos vértices pero poseen orientaciones opuestas. Cada media arista posee punteros *siguiente* y *anterior*, formando la triada que delimita una cara (aquella que queda a la izquierda al recorrerla).

Si una media arista no delimita una cara, entonces forma parte del borde de la malla. Como la malla que reconstruye el objeto debe ser cerrada, no puede presentar aristas de borde, por lo cual estas aristas nos identifican los huecos, y siguiendo sus punteros es posible obtener el contorno de cada hueco.

Debe hacerse la salvedad de que lo anterior es cierto únicamente cuando la malla conste de una sola componente conectada. En caso de que la malla conste de varias porciones desconectadas (islas), se dificulta la identificación del borde del hueco al estar formado por el contorno de diversas mallas.

4.2 Advancing front

Los algoritmos de advancing front construyen una malla del dominio a partir de su contorno. El proceso puede ser resumido de la siguiente manera:

1. Inicialización del frente.

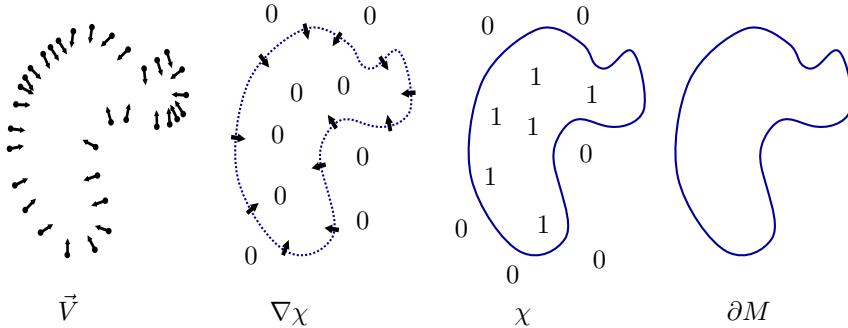


Figura 2.8: Ilustración del método de reconstrucción de Poisson. Tomando como entrada una nube de puntos con sus normales \vec{V} se define el gradiente de la función indicadora $\nabla\chi$. Mediante la ecuación de Poisson se calculará el valor de χ , y al extraer una isosuperficie se obtendrá la superficie del objeto ∂M . (Imagen cortesía de [KBH06]).

2. Análisis del frente para definir los puntos a partir de los cuales se crearán los nuevos elementos (candidatos).
3. Creación de nuevos elementos a partir de los candidatos.
4. Actualización del frente.
5. Repetición de los pasos 2–4 hasta vaciar el frente.

Se puede utilizar estas técnicas para el relleno de huecos inicializando el frente a partir del contorno de cada hueco y rellenando hacia su interior. Los diversos métodos se diferenciarán según la elección de los próximos candidatos y la determinación de las posiciones de los nuevos puntos, los cuales deben caer dentro del dominio.

Los detalles de implementación se discutirán en la sección 4.1.

4.3 Reconstrucción de Poisson

En este método el relleno de huecos se aborda como un problema de reconstrucción de la superficie a partir de puntos orientados. Por esta razón, se consideran todos los puntos a la vez sin consideraciones especiales de pertenencia o no a los contornos de los huecos. Además, se convierte el problema de la reconstrucción a la resolución de una ecuación de Poisson.

En esta ecuación de Poisson, se calcula una función indicadora χ que toma el valor 1 en los puntos que se encuentran dentro del modelo y 0 en los puntos que se encuentran fuera. De esta manera, extrayendo una isosuperficie apropiada se obtiene la frontera del modelo.

Para poder plantear la ecuación, debe notarse que el gradiente de la función indicadora $\nabla\chi$ es 0 en casi todos los puntos, excepto en aquellos cerca de la superficie, donde es equivalente a las normales de los puntos muestreados (\vec{V}), pero en sentido contrario (figura 2.8). Entonces, se busca calcular la función χ cuyo gradiente aproxime de la mejor forma posible el campo vectorial \vec{V} . Aplicando el operador de divergencia se obtiene la ecuación de Poisson

$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}$$

Entre las ventajas de esta formulación podemos nombrar:

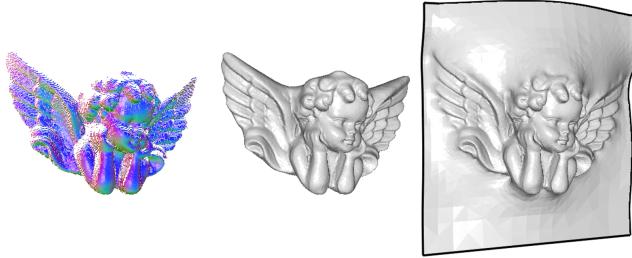


Figura 2.9: Reconstrucción del modelo Ángel (izquierda) usando condiciones de borde Dirichlet (centro) y Neumann (derecha).

- La superficie reconstruida es una superficie suave y cerrada.
- Los sistemas de Poisson son resistentes al ruido de entrada.
- El uso de funciones de soporte local para aproximar χ , produce un sistema lineal bien condicionado y ralo.[KBH06]

Condiciones de borde

Para definir el sistema lineal, es necesario especificar el comportamiento de la función a lo largo del contorno del dominio de integración. Tomando en consideración la definición de la función indicadora χ , se pueden establecer fácilmente condiciones de borde Dirichlet ($\chi = 0$) o también condiciones de borde Neumann ($\nabla\chi = 0$).

Si se utilizan condiciones Dirichlet, se asegurará que la superficie resultante sea cerrada. En cambio, en caso de que se tengan regiones sin información (huecos) de tamaño considerable, usar condiciones Neumann permitirá que la superficie se extienda fuera del dominio de integración, cruzando el borde de forma ortogonal (figura 2.9)[KH].

Extracción de la isosuperficie

Para poder obtener la superficie del modelo ∂M , es necesario definir un isovalor de χ para así obtener la isosuperficie correspondiente. Para aproximar las posiciones de los puntos de entrada, se utiliza como isovalor el promedio de los valores de χ en esos puntos:

$$\partial M \equiv q \in \mathbb{R}^3 | \chi(q) = \gamma \quad \text{con} \quad \gamma = \frac{1}{N} \sum \chi(p)$$

Esta elección de isovalor presenta la propiedad de que un escalado de χ no cambia la isosuperficie.

Herramientas

A continuación se describen las principales herramientas de software utilizadas en el desarrollo y evaluación de la biblioteca de reconstrucción.

1 Base de datos

Uno de los supuestos de este proyecto era contar con un repositorio propio de mallas tridimensionales. Para la creación de este repositorio, se utilizarían los algoritmos de reconstrucción desarrollados en [Mai19], ubicando al objeto de interés en una base giratoria y realizando capturas en ángulos espaciados hasta completar una vuelta. De esta forma, las posiciones de las vistas describirían un círculo centrado en el objeto y cada captura contendría información de posición (*xyz*) y de textura (*rgb*). Debido a los tiempos requeridos para calibrar el dispositivo de captura, este repositorio nunca se materializó, por lo que fue necesario buscar otro con características similares.

Se decidió utilizar *The Stanford 3D Scanning Repository*[Sta] que brinda acceso a escaneos tridimensionales y reconstrucciones detalladas para ser usados en investigación.

Las capturas fueron obtenidas mediante un escáner láser de barrido Cyberware 3030 MS. Se realizaron escaneos del objeto en diversas posiciones sobre una base giratoria y luego estas capturas fueron combinadas para producir una única malla triangular mediante el método de *zipping*[TL94] o el de *volumetric merging*[CL96], ambos desarrollados en Stanford.

La base de datos provee un archivo de configuración con las transformaciones de alineación requeridas por cada captura. Estas transformaciones fueron obtenidas realizando la registración de cada captura contra un escaneo cilíndrico del objeto mediante un método semiautomático, donde el usuario establece una alineación inicial que luego es ajustada mediante un algoritmo basado en ICP[TL94].

De esta base de datos se escogieron los modelos **armadillo**, **bunny**, **dragon**, **drill** y **happy**, los cuales presentan distintos niveles de detalles, cantidad de escaneos y niveles de ruido.

Desgraciadamente, no se cuenta con información de color en estos escaneos. Se decidió adaptar los algoritmos a esta situación, en lugar de agregar artificialmente valores de color para los puntos.

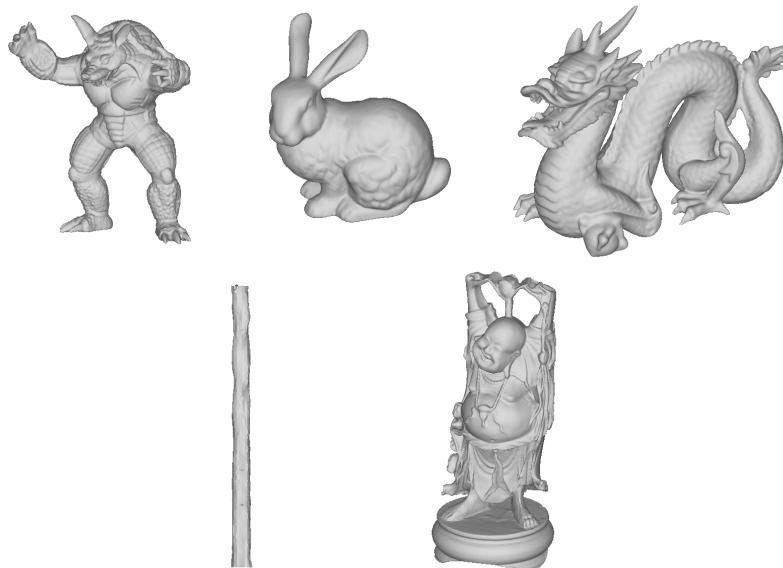


Figura 3.1: Modelos de la base de datos Stanford. De izquierda a derecha y de arriba a abajo, los modelos son: `armadillo`, `bunny`, `dragon`, `drill` y `happy`. En el caso del modelo `drill`, sólo se presenta la reconstrucción de la mecha, descartando la estructura de apoyo.

2 Tecnologías

Las siguientes tecnologías fueron consideradas y evaluadas para el diseño y desarrollo de las funcionalidades de la biblioteca, si bien algunas debieron ser descartadas por no ajustarse a las restricciones y supuestos establecidos para el producto.

2.1 KinectFusion

El algoritmo de KinectFusion fue desarrollado por Microsoft para lograr reconstrucciones tridimensionales utilizando el dispositivo Kinect.

Debido a que uno de sus objetivos era lograr una implementación en tiempo real, el algoritmo de registración requiere de poca variación de la posición relativa cámara-objeto entre capturas, por lo que fue descartado.

La fusión se realiza mediante una variación del método de *volumetric merging* sobre GPU[IKH⁺11].

2.2 Open Source Computer Vision Library (OpenCV)

Es una biblioteca de código abierto de visión computacional y aprendizaje maquinal. Cuenta con módulos de procesamiento de imágenes de profundidad y registración.

En un principio se consideró utilizar la información de color de las capturas para poder lograr la registración, pero debido a que la base de datos finalmente

elegida contiene únicamente información geométrica no se podrán aprovechar las funcionalidades de esta biblioteca.

2.3 *The Point Cloud Library (PCL)*

Es un framework de código abierto multiplataforma para el procesado de imágenes 2D/3D y nubes de puntos. Provee numerosos algoritmos modernos para reducción de ruido, extracción de puntos salientes, cálculo de descriptores, registración, reconstrucción de superficies, entre otros[RC11]. Además, contiene el módulo KinFu, el cual es una implementación libre del algoritmo KinectFusion donde se elimina la dependencia con el dispositivo Kinect, se presentan algoritmos para trabajar a una mayor escala y se permite la integración de la información de textura a la superficie final reconstruida[PL14].

La documentación incluye tutoriales para cada módulo de la biblioteca y además se cuenta con listas de correos y canales de IRC para brindar soporte.

PCL se encuentra disponible para ser usada en C++. Debido al uso intensivo de código templatizado, la compilación del código cliente suele requerir de un tiempo considerable. Existen proyectos para portarla a Python y Java, pero no se encuentran suficientemente avanzados.

2.4 *CloudCompare, Meshlab*

Son programas de procesamiento y edición de mallas de puntos 3D. Presentan herramientas de registración semiautomática (a partir de puntos seleccionados por el usuario), y cuentan con una implementación del algoritmo *Poisson Surface Reconstruction* para reconstrucción de superficies[CCC⁺08].

Se utilizaron especialmente para visualización y comparación de resultados.

2.5 *Otras herramientas de software*

Para aquellas funcionalidades no provistas por la biblioteca PCL se buscaron herramientas de software libre multiplataforma. En particular, se escogió *delaunator-cpp*¹ para realizar triangulaciones Delaunay 2D, y *DKM*² para la búsqueda de clústeres mediante el algoritmo de k-means.

¹<https://github.com/delfrrr/delaunator-cpp>

²<https://github.com/genbattle/dkm>

Diseño

En este capítulo se presentan los requerimientos identificados para el sistema y los diagramas de clases definidos para su implementación.

1 Casos de uso

Para la identificación de requerimientos se utilizó la técnica de casos de usos, cuyos resultados se reproducen a continuación.

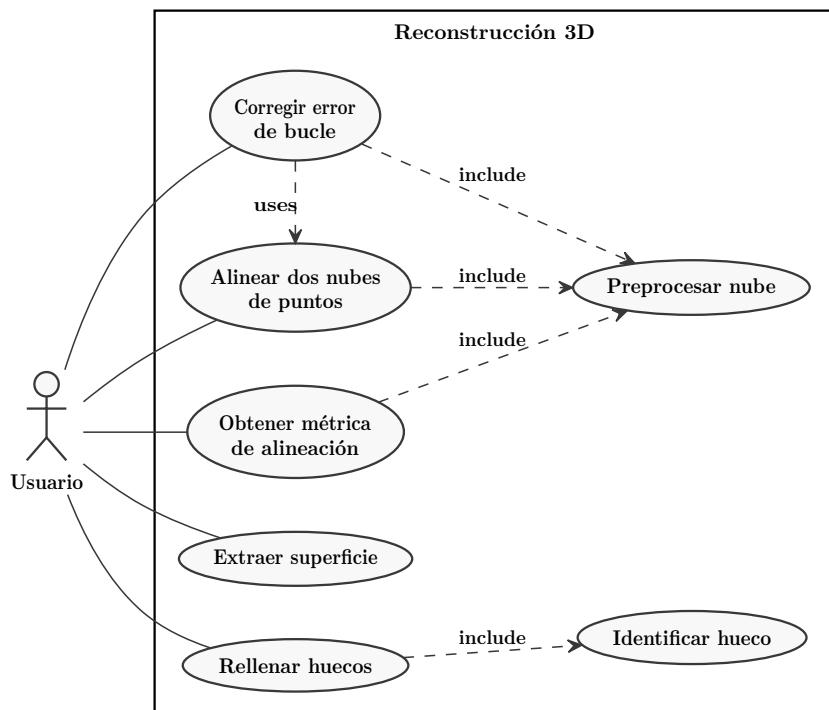


Figura 4.1: Diagrama de casos de uso.

Caso de uso: Preprocesar nube**Actor:** Usuario**Curso Normal:**

1. El usuario introduce el nombre de archivo de la nube de puntos.
2. El sistema lee la malla del disco.
3. El sistema realiza una reducción de ruidos y eliminación de outliers.
4. El sistema calcula normales y vecindades de los puntos de la nube.
5. El sistema devuelve una nube de puntos con sus normales y vecindades.

Caso de uso: Alinear dos nubes de puntos**Actor:** Usuario**Descripción:** Se calcula la transformación que posiciona la nube *fuente* en el sistema de referencia de la nube *objetivo*.**Curso Normal:**

1. El usuario introduce las nubes preprocesadas *fuente* y la nube *objetivo*. **Include (Preprocesar nube)**
2. El usuario selecciona los parámetros del algoritmo.
3. El sistema selecciona puntos salientes.
4. El sistema calcula descriptores y sus correspondencias.
5. El sistema elimina correspondencias espurias.
6. El sistema estima una transformación a partir de las correspondencias.
7. El sistema devuelve la transformación de alineación.

Caso de uso: Obtener métrica de alineación**Actor:** Usuario**Descripción:** Se calculan medidas sobre las nubes alineadas para reflejar la calidad de la alineación.**Curso Normal:**

1. El usuario introduce las nubes preprocesadas *fuente* y la nube *objetivo*. **Include (Preprocesar nube)**
2. El usuario introduce una *transformación*.
3. El sistema aplica la *transformación* a la nube *fuente*.
4. El sistema determina los puntos que corresponden al área común entre las nubes (área solapada).
5. El sistema devuelve medidas sobre el área solapada: cantidad de puntos en relación a las nubes originales, distancia punto a punto entre las nubes.

Caso de uso: Corregir error de bucle**Actor:** Usuario**Descripción:** Se ajustan las transformaciones de las nubes para lograr que el bucle cierre correctamente.**Requisitos:**

- Las nubes describen una vuelta completa.
- Las nubes se encuentran alineadas.

Curso Normal:

1. El usuario introduce una lista de nubes preprocesadas. **Include (Preprocesar nube)**
2. El sistema calcula la transformación de la última nube para lograr que la superficie cierre. **Use (Alinear dos nubes de puntos)**
3. El sistema propaga el ajuste hacia las otras nubes.
4. El sistema devuelve una lista de las transformaciones.

Caso de uso: Extraer superficie**Actor:** Usuario**Descripción:** Se obtendrá una malla poligonal que represente una colección de nubes de puntos.**Requisito:** Las nubes se encuentran alineadas.**Curso Normal:**

1. El usuario introduce una lista de nubes.
2. El sistema une todas las nubes en una sola.
3. El sistema filtra el ruido y elimina los outliers de la nube.
4. El sistema calcula las vecindades de los puntos y realiza una triangulación.
5. El sistema calcula las normales para cada triángulo.
6. El sistema devuelve la superficie representativa.

Caso de uso: Identificar huecos**Actor:** Usuario**Descripción:** Se determinarán las zonas donde existen huecos.**Curso Normal:**

1. El usuario introduce una malla poligonal.
2. El sistema identifica los vértices que forman parte de un hueco.
3. El sistema calcula los contornos de los huecos.
4. El sistema devuelve una lista con cada hueco y los vértices de su contorno.

Caso de uso: Rellenar huecos**Actor:** Usuario**Descripción:** Se estimarán las posiciones de puntos dentro de las zonas donde existen huecos.**Curso Normal:**

1. El usuario introduce una malla poligonal.
2. El usuario elige un hueco a llenar. **Include (Identificar huecos)**
3. El sistema agrega puntos en el interior del hueco.
4. El sistema integra los nuevos puntos a la malla.
5. El sistema devuelve la malla resultante.

2 Especificación de requerimientos

A partir del análisis de las herramientas de software existentes, el estudio de la bibliografía relevante y el diagrama de los casos de uso se definieron los requerimientos detallados a continuación.

2.1 Descripción

Se dispone de un sistema cámara-superficie giratoria, cuyas posiciones se encuentran fijas en el espacio y el eje de giro de la superficie se encuentra alineado con el eje vertical del dispositivo de captura. El objeto de interés se ubica sobre la superficie giratoria, y se realizan capturas a diversos intervalos de giro hasta totalizar una vuelta completa (360°).

Los algoritmos desarrollados para la registración de las capturas parciales, integración de las mallas resultantes y relleno de huecos tendrán como resultado final una superficie cerrada y triangulada que represente al objeto.

Se tendrá como entrada una nube de puntos con valores de posición $\{x, y, z\}$. No se dispondrá de información de textura, normales o conectividades.

Suposiciones

El ángulo máximo entre dos mallas no podrá exceder los 60° .

La cámara no se encontrará demasiado elevada respecto a la superficie giratoria. En ningún caso deberá superar el punto más alto del objeto.

2.2 Requerimientos funcionales

Se identificaron las siguientes funcionalidades para el sistema:

Eliminación de puntos atípicos

El sistema debe detectar y eliminar puntos considerados atípicos.

Alineación Inicial

El sistema debe poder calcular una transformación de alineación para dos mallas que las acerque lo suficiente como para poder utilizar luego ICP.

Área solapada

El sistema debe poder establecer los puntos en común (o una buena aproximación) entre dos mallas ya alineadas burdamente.

Métricas

El sistema debe poder evaluar la calidad de una registración.

Corrección de bucle

El sistema debe corregir el error propagado durante la registración una vez que se haya realizado una vuelta con las capturas.

Combinación de nubes

El sistema debe generar una malla de consenso, ajustando los puntos y sus normales según la información provista por cada malla de entrada.

Triangulación

El sistema debe poder triangular una nube de puntos tridimensional.

Relleno

El sistema debe disponer de funciones para lograr que una malla sea cerrada. Se estimará una superficie en las zonas donde se carezca de información.

2.3 Requerimientos no funcionales

Se identificaron los siguientes requerimientos no funcionales:

Tiempo de ejecución

No se espera una ejecución a tiempo real de los algoritmos implementados.

Interfaces con software

Las operaciones sobre las mallas y nubes de puntos se realizará mediante la *Point Cloud Library* (PCL). Debido a esto, se desarrollará en el lenguaje de programación C++.

Sistemas operativos

El producto desarrollado estará destinado a utilizarse en los sistemas operativos Windows y Linux.

3 Diagrama de clases

El sistema se dividió en cuatro módulos principales:

1. Preproceso: donde se realiza una reducción de ruido a las nubes de entrada antes de ser procesadas por las siguientes etapas.
2. Registración: donde se obtienen las transformaciones de rotación y translación que lleven a cada captura a un sistema global de forma que las zonas comunes encajen perfectamente.
3. Fusión: donde se combina la información presente en cada captura para obtener finalmente una malla que represente la totalidad del objeto.
4. Rellenado de huecos: donde se estima la superficie del objeto en las zonas sin información.

A continuación se presenta el diagrama de clases del sistema y una breve descripción de las clases más importantes.

- **Nube:** Representa una captura del objeto desde una posición de la cámara en particular. Es una colección de *Puntos* sin organización. Clase provista por PCL.
- **Punto:** contiene las coordenadas $\{x, y, z\}$ obtenidas por el dispositivo de captura. Sus normales se estimarán durante el preproceso.
- **Registración:** se encarga de obtener la *Transformación* que permita alinear dos *Nubes* entre sí. Para esto establece correspondencias entre los *Anclajes*.
- **Anclaje:** a partir de puntos salientes de *Nube* calcula *descriptores* que permitan asociarlos y un *marco de referencia* para obtener una estimación de la *Transformación*.
- **Transformación:** representa una transformación de rotación y translación que será aplicada a una *Nube* para alinearla.

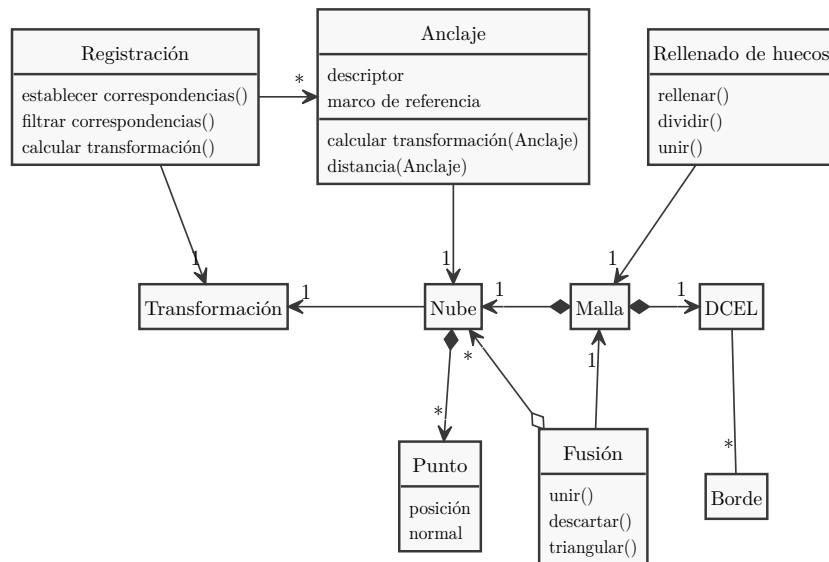


Figura 4.2: Diagrama de clases del sistema.

- **Fusión:** une las *Nubes* ya alineadas para obtener una superficie global que represente al objeto. Esto supone corregir la posición de los puntos, descartar aquellos considerados como ruido y triangular la superficie.
- **Malla:** triangulación de la *Nube* representada por un grafo de conectividades (*DCEL*).
- **DCEL:** relaciona las caras, aristas y vértices de una superficie mediante una estructura de medias aristas.
- **Rellenado de huecos:** La clase se encarga de estimar nuevos puntos en zonas donde se carece de información (huecos) y triangularlos para que la *Malla* sea cerrada.
- **Borde:** Es una colección de puntos ordenados que representa un borde de un hueco en la *Malla*.

Desarrollo

El proceso de reconstrucción es lineal: se elimina el ruido de las capturas parciales del objeto, se las alinea en un marco de referencia global, se las combina para obtener una sola malla total y se rellenan los huecos de esta malla para obtener una superficie cerrada que represente al objeto.

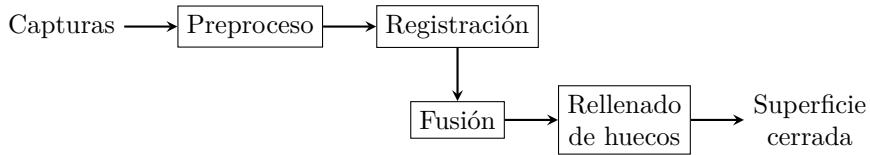


Figura 5.1: Proceso de reconstrucción tridimensional.

A continuación se desarrollan los detalles de implementación de cada uno de estos procesos.

1 Módulo de preprocess

El módulo de preprocess se encarga de realizar una reducción de ruido a las nubes de entrada antes de ser procesadas por las siguientes etapas. Para esto, es necesario considerar qué características posee el ruido y cómo se produce, y por esta razón no podemos independizarnos completamente del dispositivo de captura.

Considerando las fuentes de error más importantes de los escáneres de luz estructurada, se desarrolló el siguiente algoritmo para el preprocess de cada nube de entrada:

- Para independizarse de la escala de las capturas, todos los parámetros que impliquen una distancia o vecindad se establecen en relación a una medida de resolución de las nubes, definida como el promedio de las distancias entre cada punto contra su par más cercano (algoritmo 1). Si bien pueden presentarse zonas de mayor densidad de puntos que otras, surgen problemas para definir la localidad de estos agrupamientos, por lo que se optó por una medida global.
- Para reducir las pequeñas perturbaciones en las posiciones de los puntos, se los proyecta a una superficie estimada mediante el método de mínimos cuadrados móviles. Para construir esta superficie se utiliza la clase `pcl::MovingLeastSquares`, definiendo una vecindad de seis veces la resolución de la nube con el fin de obtener aproximadamente 100 puntos para realizar la estimación. Los resultados de este proceso se evaluaron

Algoritmo 1 Medida de resolución de la nube como promedio entre las distancias de los pares más cercanos.

```

1: RESOLUCIÓN(nube)
2:   distancia ← 0
3:   for all  $p \in \text{nube.puntos}$  do
4:     v ← cercano( $p, \text{nube.puntos}$ )[1]
5:     distancia ← distancia +  $d(p, v)$ 
6:   return distancia / nube.tamaño

```

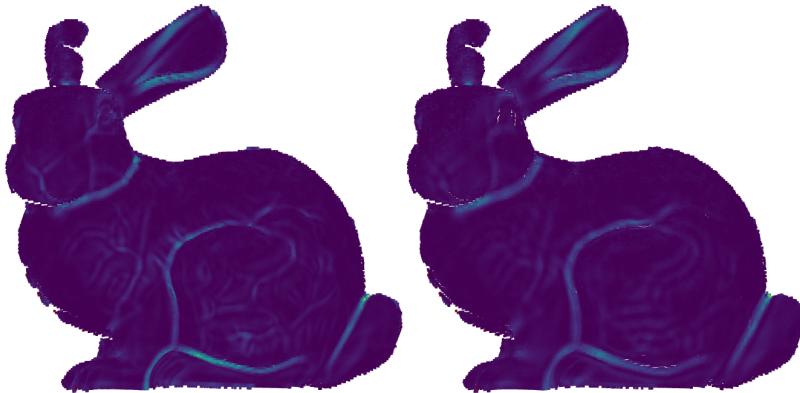


Figura 5.2: Visualización de la curvatura después de aplicar mínimos cuadrados móviles en una vecindad de seis veces la resolución (izquierda) y diez veces la resolución (derecha). Los detalles del cuerpo y rostro son percibidos con menor intensidad al aumentar el tamaño de la vecindad.

visualmente, pudiendo observarse la pérdida de detalles al aumentar en exceso la vecindad (figura 5.2).

- Para evitar las distorsiones debidas al ángulo de incidencia excesivo, se estiman las normales en cada punto mediante la técnica de covarianza propuesta en [BC94]. Luego se eliminan los puntos cuyas normales se encontraban a más de 80° del eje z .
- Se eliminan aquellos puntos que pueden ser producto de distorsiones debidas al reflejo parcial. Estos puntos se identifican de la siguiente manera:
 - Considerando que la captura se encuentra en 2.5D, se proyecta la nube al plano $z = 0$ para realizar una triangulación Delaunay y así trasladar estas conexiones a los puntos en el espacio.
 - Se eliminan aquellas aristas cuya longitud supera un umbral. Estas aristas corresponden a saltos excesivos de profundidad o a regiones en las que no se registraron puntos.
 - Los puntos buscados son aquellos que delimitan bordes en la malla o se encuentran aislados.

Si bien al finalizar esta etapa se obtiene una triangulación de la nube de entrada, los algoritmos implementados en PCL no están diseñados para considerar las conectividades y trabajan únicamente con las vecindades.

2 Módulo de registración

El módulo de registración se encarga de obtener las transformaciones de rotación y translación que lleven cada vista a un sistema global de forma que las zonas comunes encajen perfectamente.

Debido a que «el algoritmo de *Iterative Closest Point* (ICP) se ha convertido en el método dominante para realizar la registración de modelos tridimensionales utilizando únicamente la información de geometría de los mismos[RHHL02]» y a que a pesar de que existe garantía de convergencia del método hacia un mínimo local este puede no ser el mínimo global buscado, es necesario proveer una alineación inicial lo suficientemente cercana para obtener una registración correcta[BM92].

Por esto, y considerando que la distancia entre capturas sucesivas es considerable, se desarrollaron algoritmos para obtener una alineación inicial con cada par de capturas para luego refinar la registración mediante ICP. Estos algoritmos constan de los siguientes pasos:

1. Selección de puntos de la entrada.
2. Cálculo de descriptores y determinación de correspondencias.
3. Filtrado de correspondencias.
4. Estimación de la transformación de alineación.

Variaciones en la ejecución de estos pasos nos permiten implementar diversos métodos.

Utilizando la primera captura para definir el sistema de coordenadas global, cada nueva vista se alinea con la anterior hasta completar una vuelta sobre el objeto, propagando el error de registración. Por esta razón, se incorporó un algoritmo de corrección de bucle al proceso de registración (figura 5.3).

A continuación se describen dos métodos para calcular la alineación inicial entre un par de capturas.

2.1 Alineación mediante *sample consensus*

Selección de keypoints

En base a los resultados obtenidos por [Zho09], se consideró utilizar el algoritmo de detección de keypoints basado en *Intrinsic Shape Signatures* (algoritmo 2), el cual se halla implementado en PCL en la clase `ISSKeypoint3D`, permitiendo definir el radio de la vecindad y el nivel de disimilitud de los eigenvalores.

Algoritmo 2 Determinación de los keypoints mediante ISS

```

1: ISS KEYPOINTS(nube,  $r_1$ , umbral1, umbral2,  $r_2$ )
2:   keypoints  $\leftarrow \emptyset$ 
3:   for all  $p \in$  nube.puntos do
4:     vecinos  $\leftarrow$  obtener puntos cercanos(nube, p,  $r_1$ )
5:     m  $\leftarrow$  matriz de covarianza(vecinos)
6:      $\lambda \leftarrow$  eigenvalores(m)
7:     if  $\lambda_1/\lambda_2 >$  umbral1 and  $\lambda_2/\lambda_3 >$  umbral2 then
8:       keypoints.insert(p)
9:   return Non-Max Suppression(keypoints,  $r_2$ )

```

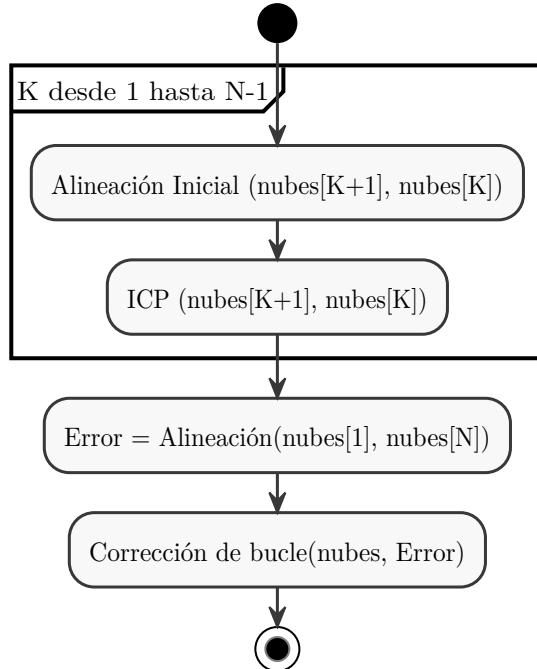


Figura 5.3: Diagrama de flujo de la registración.

Sin embargo, no pudieron encontrarse los parámetros adecuados. Al observar las zonas comunes en los casos de estudio utilizados como referencia durante el desarrollo, eran pocos los keypoints que realmente se encontraban lo suficientemente cerca como para generar una correspondencia válida (figura 5.4a).

Por esta razón, se cambió el método de selección de keypoints, eligiendo un análisis de persistencia multiescala:

1. Por cada punto de la nube se calcula su descriptor para distintos tamaños de vecindad (escala).
2. A partir de todos los descriptores en todas las escalas se estima una distribución gaussiana que los aproxime.
3. Los keypoints quedan definidos como aquellos puntos cuyos descriptores se encuentran alejados de la media[RBB09].

El algoritmo se encuentra implementado en PCL en la clase `MultiscaleFeaturePersistence` permitiendo ajustar las escalas a utilizar, el umbral para ser considerado saliente y el descriptor a utilizar. Debido a que es necesario calcular un descriptor para cada punto, y además en diferentes escalas, se eligió utilizar el método *Fast Point Feature Histograms* (FPFH) para su construcción, el cual es lineal en la cantidad de puntos de la vecindad.

Con este algoritmo, los keypoints se agrupan formando líneas en zonas de cambio brusco de curvatura (figura 5.4b).

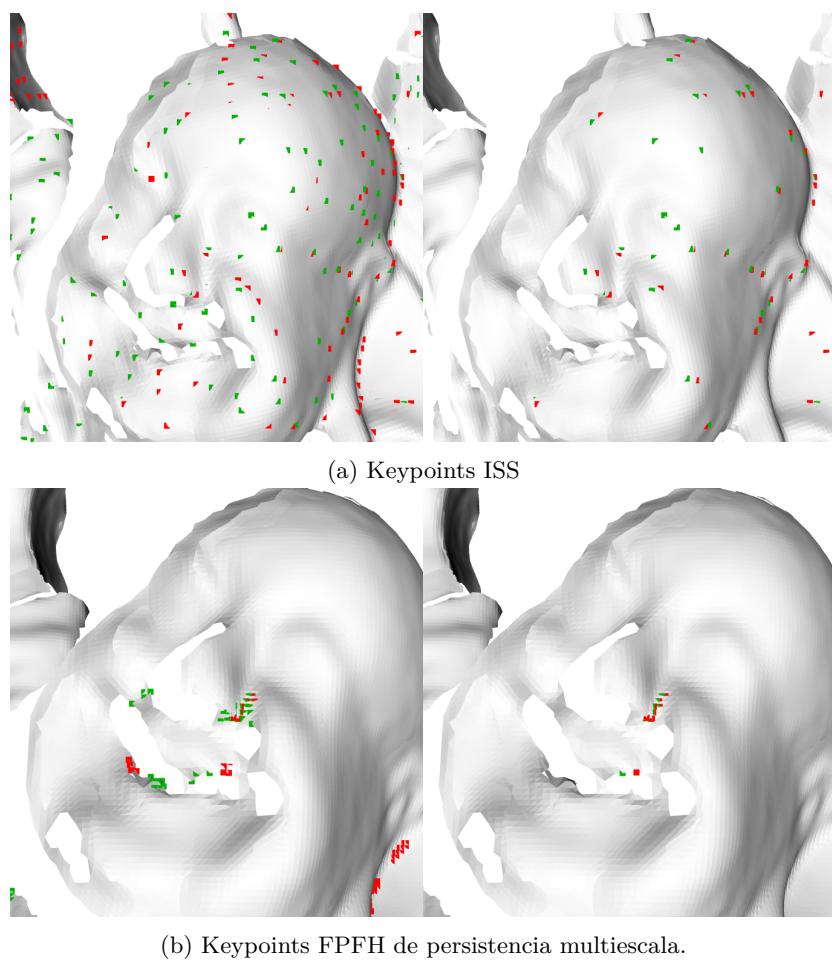


Figura 5.4: Visualización de los keypoints calculados en las vistas `happy_0` (verde) y `happy_24` (rojo). A la derecha se seleccionaron aquellos cuyo par más cercano se encontraban a menos de 4 veces la resolución de la nube.

Estimación de la transformación

Se utilizó el algoritmo de *sample consensus initial alignment (SAC-IA)* para obtener una estimación inicial de la transformación de alineación. Este algoritmo consiste en:

1. Seleccionar al azar m puntos de la nube A.
2. Por cada punto, buscar aquellos con descriptores similares en B y seleccionar uno al azar.
3. Calcular la transformación definida por estos puntos y sus correspondencias. Es decir, calcular la transformación que minimice

$$\sum |b_j - T(a_j)|$$

para los puntos muestreados.

4. Calcular una medida del error de transformación. Es decir, calcular el error

$$\sum |b_j - T(a_j)|$$

para todos los puntos de la nube.

5. Repetir varias veces y devolver aquella transformación que produzca el menor error.[RBB09]

Búsqueda de parámetros

Para ajustar los valores de los parámetros del algoritmo se eligieron dos modelos de la base de datos Stanford[Sta] Seleccionando pares de capturas de cada modelo, se estableció un conjunto de parámetros que alinee apropiadamente a todos los pares, buscando, además, independencia respecto a las características geométricas del modelo.

Los modelos escogidos fueron `happy stand` y `bunny` debido a las diferencias geométricas que presentan:

- Las capturas de `happy stand` se realizaron a intervalos de ángulos equiespaciados (cercaos a 24°) y presentan zonas suaves en la base, vientre y espalda.
- Los ángulos entre las capturas de `bunny` son más variados y espaciados, yendo desde 35° hasta 90° , y se observa una superficie más irregular por todo el cuerpo (figura 3.1).

Primero se trabajó sobre el modelo `happy stand`. Se observó que una selección de keypoints demasiado agresiva no permitía la convergencia del algoritmo de SAC-IA, mientras que una selección demasiado permisiva elevaba el costo computacional de forma excesiva. Finalmente, utilizando escalas diádicas y estableciendo un umbral de un desvío y medio se obtuvieron buenos resultados en la mayoría de las capturas (figura 5.5).

Sin embargo, fue imposible hallar los parámetros adecuados para el modelo `bunny`. Exceptuando las alineaciones entre las capturas $\{315, 000\}$ y $\{000, 045\}$, el algoritmo de registración establecía correspondencias completamente erróneas (figura 5.6). Por esta razón, no se continuó con el desarrollo de este algoritmo.

2.2 Alineación mediante búsqueda de clúster

En este método alternativo de alineación no se seleccionan keypoints, sino que se realiza un submuestreo uniforme de los puntos de la nube para reducir

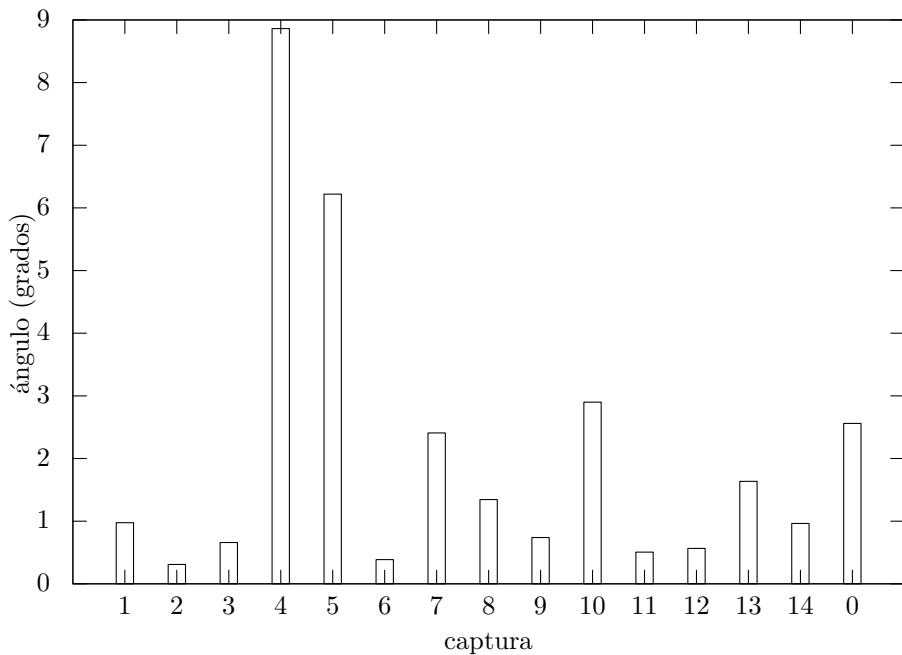


Figura 5.5: Diferencias absolutas entre la rotación estimada y el *ground truth* para el objeto *happy stand* utilizando el algoritmo de sample consensus.



Figura 5.6: Ejemplo de fallo del algoritmo de *sample consensus* producto de malas correspondencias (par de capturas {270, 315} del modelo *bunny*)

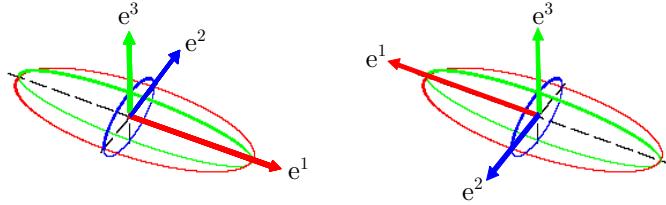


Figura 5.7: Marcos de referencia. Se observa una ambigüedad equivalente a un giro de 180° sobre el eje e^3 (Imagen cortesía de [Zho09]).

el costo computacional. Además del descriptor, en cada punto se estableció un marco de referencia que permite estimar una transformación de alineación considerando solamente los dos puntos que conforman una correspondencia[Zho09]. Entonces, se procede a un proceso de votación para determinar la alineación final.

Para determinar el marco de referencia se utiliza la matriz de covarianza de la vecindad del punto:

- Se computan los eigenvalores $\lambda_1, \lambda_2, \lambda_3$ en orden decreciente y sus eigenvectores correspondientes e^1, e^2, e^3 .
- Debido a que e^3 representa la normal del punto, se ajusta su sentido para que coincida con el del eje z .
- Los otros ejes se definen mediante e^1 y $e^1 \times e^3$.

Se presenta una ambigüedad en el marco de referencia según el sentido que se le asigne a e^1 (figura 5.7), la cual se resuelve definiendo un eje de giro para la alineación.

Para establecer las correspondencias se eligió nuevamente el descriptor FPFH, comparando los histogramas entre cada par de puntos mediante la distancia χ^2 :

$$\chi^2 = \sum_{j=1}^N \frac{(a_j - b_j)^2}{a_j + b_j}$$

Para identificar y eliminar las correspondencias erróneas se utilizan los marcos de referencia y las suposiciones de ubicación de la cámara en la obtención de las capturas. Es decir, se descartan aquellas correspondencias que requieren un movimiento en y excesivo o una rotación sobre un eje no vertical (figura 5.8).

Cada correspondencia define un ángulo de giro θ sobre el eje y y una translación en el plano xz , observándose una agrupación de los parámetros de estas transformaciones (figura 5.9). Mediante el algoritmo de k-means, se busca el centroide del clúster más grande para estimar la transformación final.

Búsqueda de parámetros

Para establecer los valores de los parámetros se procedió de la misma manera que en el caso del algoritmo de SAC-IA. Se definieron umbrales permisivos para el desplazamiento y el ángulo de giro (8 veces la resolución de la nube y 10° respecto a la vertical, respectivamente) y se realizaron incrementos discretos del tamaño de la vecindad para el cálculo de los marcos de referencia y los

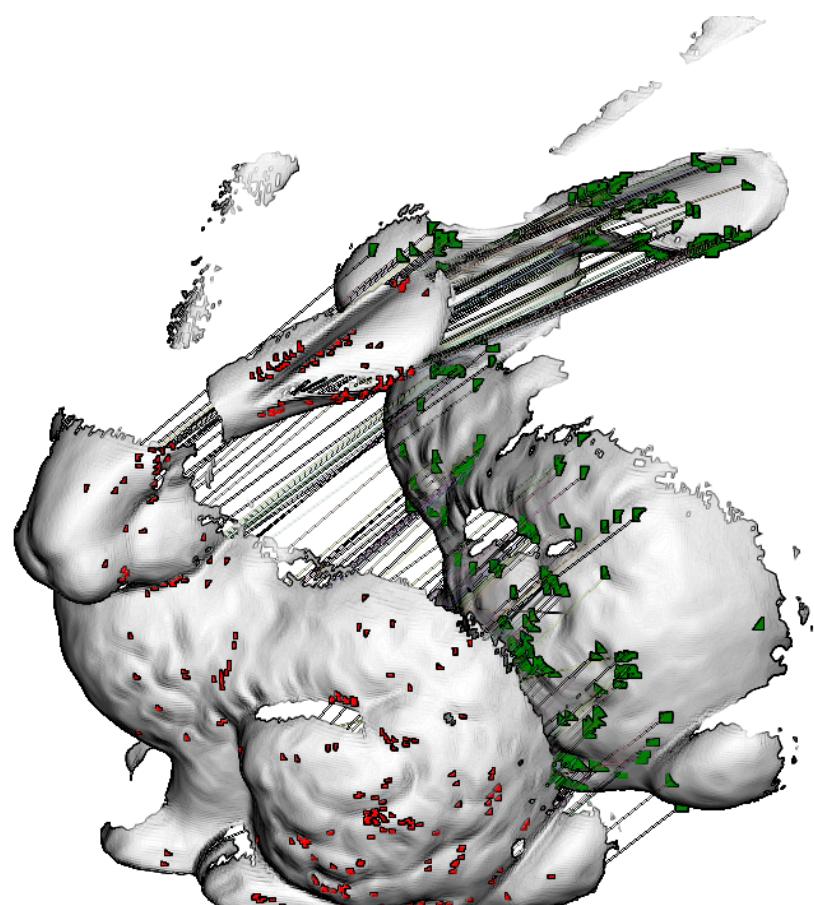


Figura 5.8: Correspondencias supervivientes luego de realizar el descarte según la transformación estimada por cada marco de referencia.

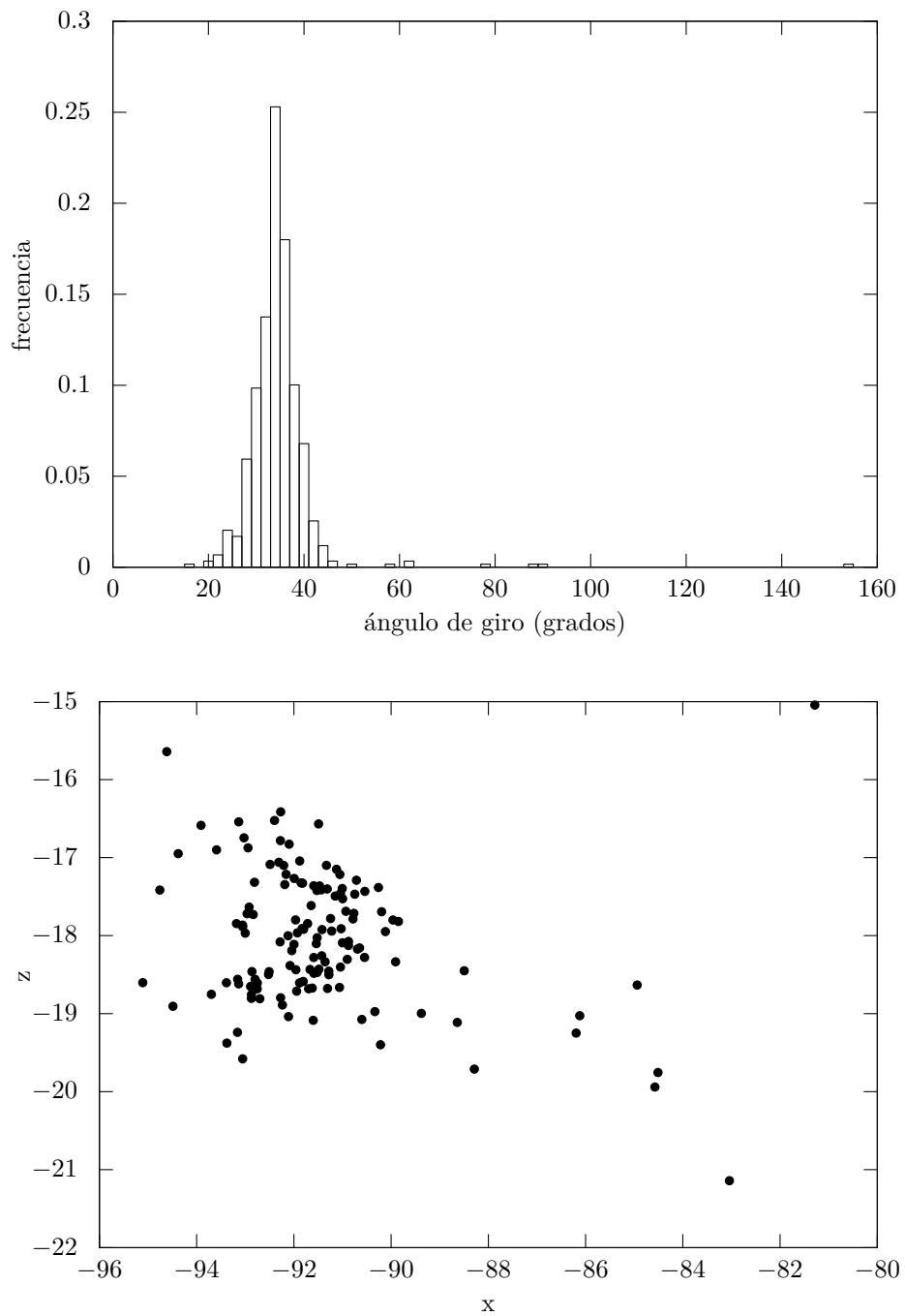


Figura 5.9: Estimación de la transformación de alineación mediante los marcos de referencia de los puntos de cada correspondencia entre las capturas `bun000` y `bun045`. Los desplazamientos se encuentran medidos en unidades de resolución.

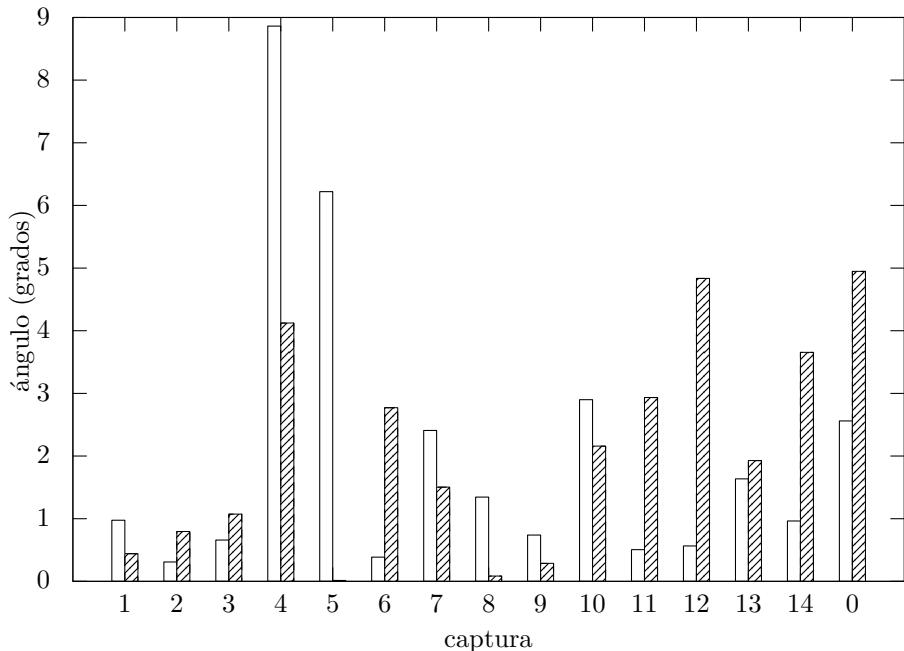


Figura 5.10: Diferencias absolutas entre la rotación estimada y el *ground truth* para el objeto **happy**, para los algoritmos de sample consensus (claro) y búsqueda de clúster (oscuro).

descriptores, obteniéndose los mejores resultados para una vecindad de 8 veces la resolución de la nube.

Para el modelo **happy**, las diferencias respecto al *ground truth* no superaron los 5° , teniendo un error promedio de 2° (figura 5.10). Tampoco se tuvieron problemas con el modelo **bunny**, a pesar de presentar saltos cercanos a 90° en las alineaciones contra la captura **bun180**. Si bien este salto supera las restricciones impuestas, en ningún caso se produjo una diferencia mayor a 5° respecto al *ground truth* (figura 5.11).

De esta manera, se logró acercar las capturas lo suficiente como para intentar alinearlas por ICP.

2.3 Refinamiento

Para ajustar las alineaciones iniciales se realiza una segunda alineación utilizando el algoritmo de ICP provisto por la biblioteca PCL. Se consideran únicamente las áreas solapadas, restringiendo el espacio de búsqueda de las correspondencias, y se minimiza la distancia entre los puntos de la nube a transformar hacia los planos definidos por las normales en la nube objetivo.

Luego, para reducir el error propagado por cada alineación, se realiza una corrección de bucle ajustando la última captura para que correspondiese con la primera, y agregando esta transformación a las otras alineaciones ponderándola de forma proporcional a su posición en el bucle (algoritmo 3).

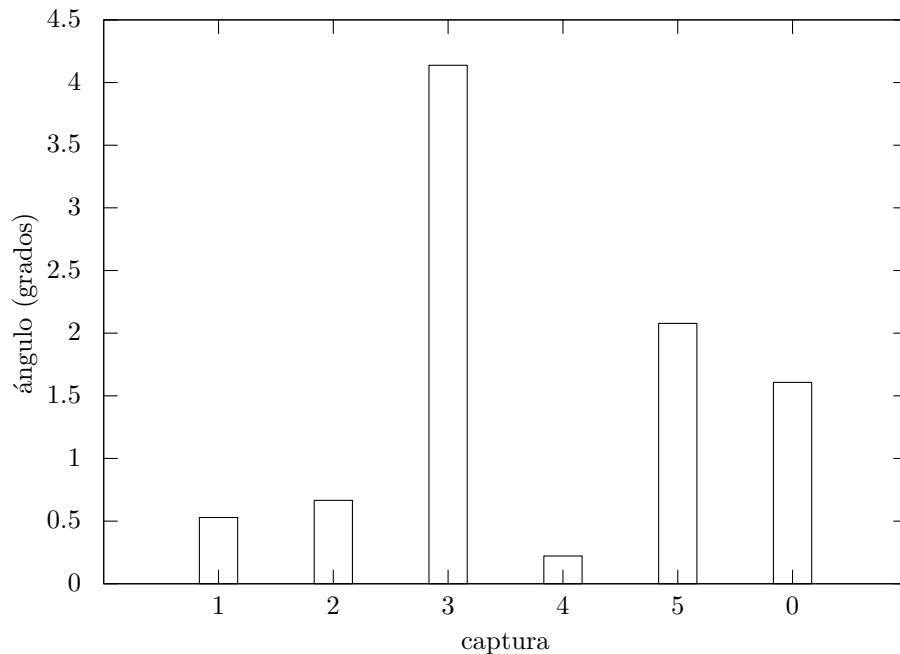


Figura 5.11: Diferencias absolutas entre la rotación estimada y el *ground truth* para el objeto *bunny* utilizando el algoritmo de búsqueda de clúster.

Algoritmo 3 Corrección de la propagación del error de alineación.

```

1: CORRECCIÓN DE BUCLE(nubes, N)
2:   peso  $\leftarrow \frac{1}{N-1}$ 
3:   error  $\leftarrow$  Alineación(desde=nubes[1], hacia=nubes[N])
4:    $[q|t] \leftarrow$  inversa(error)
5:   for all  $K \in 1 : N$  do
6:     rotación  $\leftarrow$  slerp( $K * \text{peso}$ ,  $q$ , Identidad)
7:     translación  $\leftarrow K * \text{peso} * t$ 
8:     nubes[K]  $\leftarrow$  transformar([rotación|translación], nubes[K])

```

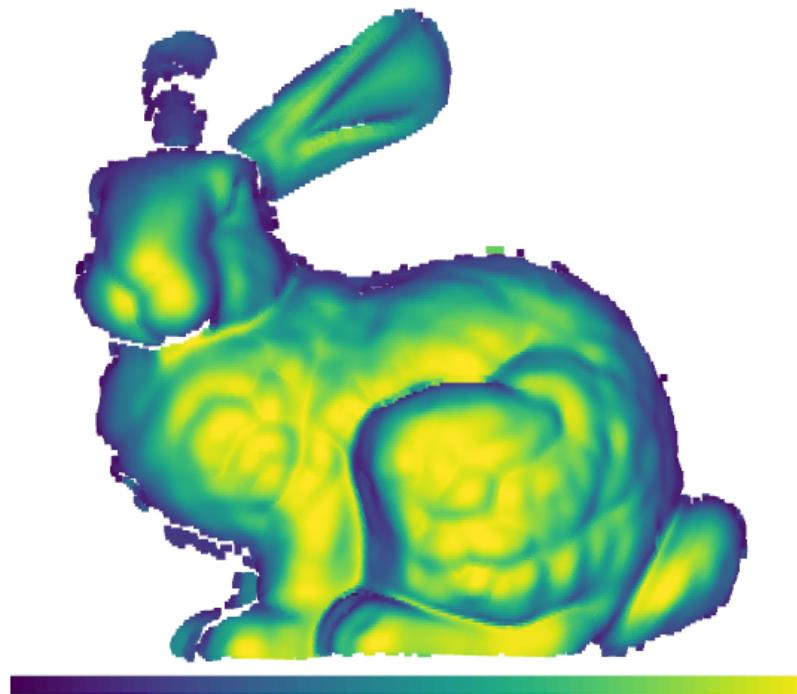


Figura 5.12: Visualización de los valores de confianza de los súrfeles para la captura bun000.

3 Módulo de fusión

El módulo de fusión tiene como objetivo obtener un modelo que describa la geometría del objeto escaneado. Para esto, el modelo se inicializa a partir de una captura cualquiera y se procesan una a una las capturas correspondientes a las otras vistas. En cada nueva vista se agrega información del objeto en zonas que no eran antes visibles y además se confirma o refuta la información ya presente en el modelo. Al combinar esta información, el módulo de fusión obtiene finalmente una malla que represente a la totalidad del objeto.

3.1 Método

Se utiliza una representación de *surfel* para cada punto similar a la propuesta en [WWLV09] debido a la facilidad de implementación de las funciones de actualización de los puntos. Como es necesario integrar la información de posición y normal de cada punto, se idearon funciones de conversión del formato de nube usado en el módulo de registración.

Cada surfel tiene asociado un valor de confianza y las vistas en las que fue observado. El valor de confianza nos indica la probabilidad de que sus valores de posición y normal no sean producto de un error de muestreo. Este valor se inicializa según el ángulo de su normal respecto a la línea de la cámara (figura 5.12).

El algoritmo 4 describe el agregado de una nueva vista. Por cada punto de la vista se busca qué surfel lo contiene y se actualiza su posición y normal ponderando según el nivel de confianza.

$$\hat{P}_k = \frac{\sum_j \alpha_j P_k^{(j)}}{\sum_j \alpha_j}$$

$$\hat{n}_k = \frac{\sum_j \alpha_j n_k^{(j)}}{\sum_j \alpha_j}$$

En caso de que haya caído fuera del dominio de la reconstrucción actual, se considera que es un nuevo punto y se lo agrega. Si la distancia de proximidad elegida es demasiado pequeña, nunca se realizará la actualización ya que todos los puntos serán considerados como nuevos surfels. En cambio, si la distancia es muy grande, se considerarán puntos que deberían haber sido descartados como atípicos.

Algoritmo 4 Actualización de la reconstrucción al agregar una nueva vista.

```

1: AGREGAR VISTA(vista, reconstrucción)
2:   for all  $p \in$  vista.puntos do
3:     surfel  $\leftarrow$  reconstrucción.buscar( $p$ )
4:     if surfel =  $\emptyset$  then
5:       reconstrucción.agregar( $p$ )
6:     else
7:       surfel.actualizar( $p$ )

```

Al terminar de procesar todas las vistas, habrá surfels que hayan sido observados desde sólo una posición y tengan un nivel de confianza bajo. Estos son considerados como ruido y eliminados de la reconstrucción.

Finalmente, se procede a la obtención de una triangulación de los surfels. Para esto se empleó el algoritmo de *Greedy Projection Triangulation* que provee PCL, el cual realiza una proyección de la vecindad de un punto en el plano definido por su normal y procede a conectar los puntos de modo que los triángulos resultantes respeten umbrales de longitud y ángulos definidos por el usuario (figura 5.13).

4 Módulo de relleno de huecos

Al finalizar el algoritmo de fusión se obtiene una malla triangular a partir de la información proveniente de cada vista. Sin embargo, esta malla no es cerrada ya que existen zonas que ninguna vista pudo capturar y por lo tanto carecen de puntos, produciendo huecos en la misma.

El módulo de relleno de huecos se encarga de estimar, de forma automática, la superficie del objeto en estas zonas para así obtener finalmente una malla cerrada.

Se plantearon dos métodos:

- Un algoritmo de *advancing front*, que trabaja localmente con los puntos que forman el contorno de cada hueco.
- Reconstrucción de Poisson, que trabaja con todos los puntos de la nube a la vez.

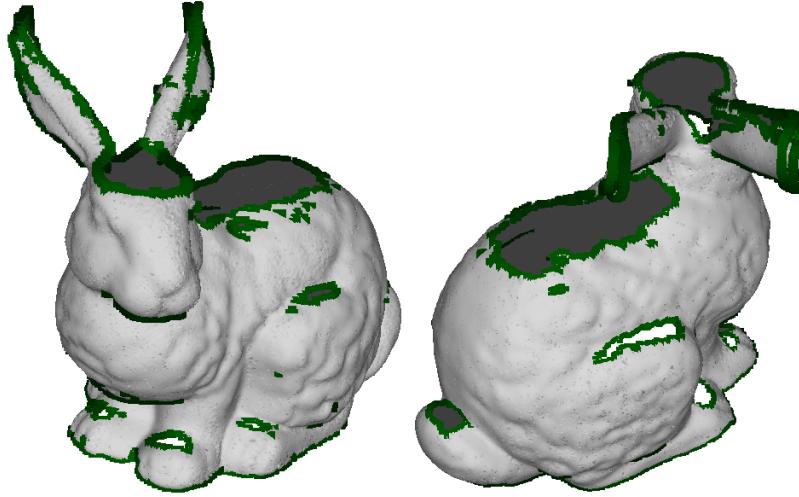


Figura 5.13: Superficie reconstruida. En verde se destacan los bordes de los huecos.

4.1 Advancing front

Se implementó una variante del método propuesto por [WCLL11], la cual se halla descripta en el algoritmo 5. Cada hueco se rellena de forma independiente a los otros, utilizando únicamente los puntos que conforman cada contorno para estimar las posiciones de los nuevos puntos.

Es posible que dentro de un hueco se observen islas, es decir, elementos que no lograron conectarse al resto de la malla. Si bien estas islas nos brindan información de la superficie, su presencia dificulta la identificación del contorno de cada hueco. Por esta razón, se eliminan todas las islas, quedándose únicamente con la componente conectada que contiene la mayor cantidad de puntos. De esta forma, una arista que defina sólo un triángulo forma parte de un hueco, y puede obtenerse el contorno del mismo recorriendo el grafo de conectividades.

El frente se inicializa a partir del contorno del hueco, y se recorre el borde calculando los ángulos entre las aristas a fin de encontrar los tres puntos que definen el menor ángulo α . Según el valor de α , se determina si insertar o no un nuevo punto (figura 5.14). Los nuevos puntos insertados son elegidos de forma que los triángulos resultantes sean aproximadamente equiláteros, como se detalla en el algoritmo 6. Estos puntos son luego proyectados en un plano de soporte definido mediante las normales de los puntos del ángulo candidato.

En caso de que el nuevo punto cayese cerca de otro ya existente, se utiliza este último. Esto implica dividir en dos el hueco, y llenar cada nueva porción de forma independiente (figura 5.15).

4.2 Reconstrucción de Poisson

Implementación

La clase **Poisson** de la biblioteca *PCL* implementa este método de reconstrucción imponiendo condiciones de borde Neumann.

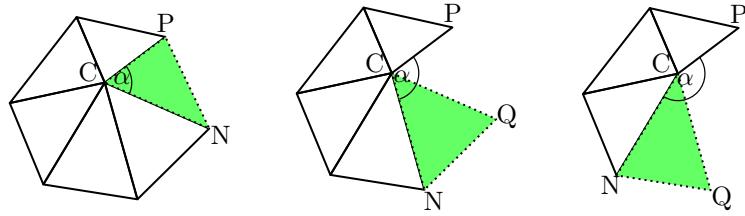


Figura 5.14: Creación de los nuevos triángulos mediante un algoritmo de advancing front. Dependiendo del valor del ángulo candidato α se utilizan los puntos existentes (izquierda), o se crea un nuevo punto en $\alpha/2$ (centro) o $\alpha/3$ (derecha).

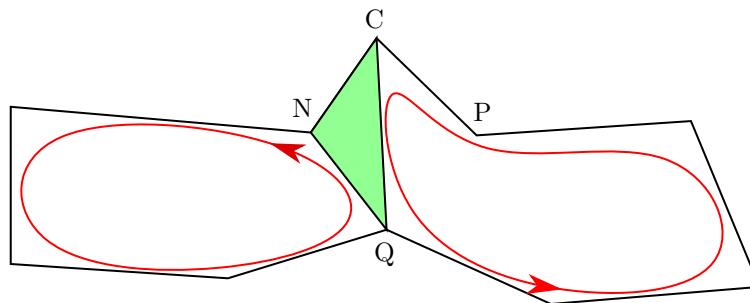


Figura 5.15: División del frente en dos al utilizar un punto de la malla para crear el nuevo triángulo.

Algoritmo 5 Relleno de huecos mediante el método de *advancing front*. Los umbrales fueron elegidos de forma de obtener triángulos con ángulos cercanos a 60° .

```

1: ADVANCING FRONT(Malla, Contorno)
2: AF ← Contorno
3: repeat
4:    $\alpha = \angle PCN = \text{ángulo mínimo(Contorno)}$ 
5:   if  $\alpha < 75^\circ$  then
6:     Malla.agregar triángulo(P, C, N)
7:     AF.eliminar punto(C)
8:   else if  $\alpha < 135^\circ$  then
9:     Q ← crear punto(P, C, N,  $\alpha/2$ )
10:    Malla.agregar punto(Q)
11:    Malla.agregar triángulo(Q, C, N)
12:    AF.insertar punto(Q)
13:   else if  $\alpha < 180^\circ$  then
14:     Q ← crear punto(P, C, N,  $\alpha/3$ )
15:     Malla.agregar punto(Q)
16:     Malla.agregar triángulo(Q, C, N)
17:     AF.insertar punto(Q)
18:   until  $AF \neq \emptyset$ 
```

Algoritmo 6 Creación del nuevo punto

```

1: CREAR PUNTO(P, C, N,  $\theta$ )
2:   planoA  $\leftarrow$  plano(P, C, N)
3:   planoB  $\leftarrow$  {
   .punto  $\leftarrow$  promedio(P, C, N)
   .normal  $\leftarrow$  promedio(P.normal, C.normal, N.normal)
4:   Q  $\leftarrow$  rotar( punto = N, origen = C,
   normal = planoA.normal,  $\theta$  )
5:   return proyección(Q, planoB)

```

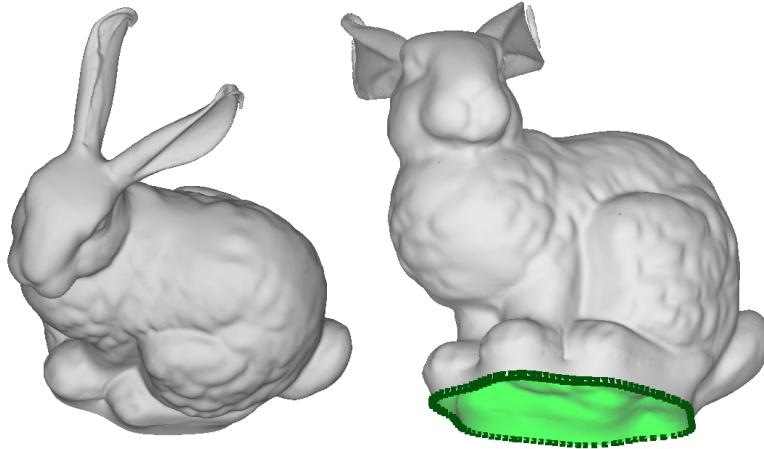


Figura 5.16: Reconstrucción de la superficie mediante el método de *Poisson*. Todos los huecos fueron rellenos a excepción del presente en la base de apoyo (destacado en verde), donde se observa, además, un estiramiento debido a las condiciones de borde Neumann.

Debido a que sólo es de interés el valor de χ en los puntos cercanos a la superficie, se utiliza un octree para representar esta función. Se provee de parámetros para establecer la profundidad del octree, controlando de esta manera la resolución de la superficie reconstruida. Sin embargo, debe considerarse que el consumo de memoria y el tiempo se incrementan de forma cuadrática con la resolución, observándose un incremento en un factor de 4 por cada aumento de la profundidad del octree[KBH06].

Debido a que no se cuentan con puntos en la base de apoyo de los objetos, el uso de condiciones de borde Neumann produce un estiramiento hacia abajo de la superficie resultante y la presencia de un hueco plano en la base (figura 5.16).

Pruebas y resultados

A continuación se detallan las pruebas realizadas y los resultados obtenidos por cada módulo desarrollado utilizando como base los modelos `armadillo`, `bunny`, `dragon`, `drill` y `happy` del repositorio de Stanford[Sta].

De cada modelo se seleccionaron aquellas capturas que fueron realizadas completando una vuelta alrededor del objeto. Se observa que la distancia entre capturas presenta un mínimo de 24°

1 Módulo de registracióñ

Para la registracióñ se utilizó el método basado en la búsqueda de clúster, seguido de un refinamiento mediante ICP y una corrección de bucle. Se plantearon dos métodos para evaluar la calidad de cada registracióñ:

- Mediante la comparación entre la transformación calculada y aquella provista por la base de datos (*ground truth*).
- Mediante una métrica de *fitness* que se obtiene a partir de la nube transformada y las nubes de entrada.

Para comparar las alineaciones contra el *ground truth*, se observa el efecto de las mismas sobre un punto orientado simulando la cámara (figura 6.1). El punto *eye* (C) se ubica inicialmente en las coordenadas $\{0, -0.1, 0.7\}$ (valores obtenidos de la base de datos), y se orienta el vector *target* hacia $-z$ y el *up* hacia y . El error de posicionamiento es la razón entre la distancia al punto de inicio y la distancia al punto obtenido por el *ground truth*.

$$\text{Error} = \frac{|C' - C_{gt}|}{|C_{gt} - C|}$$

Los errores de *target* y *up* se corresponden al ángulo formado contra los vectores respectivos obtenidos por el *ground truth*.

En el cuadro 6.1 se presentan los errores de registracióñ promedio para cada orientación de los modelos. En la mayoría de los casos, los errores no superan 1° en orientación ni 1% en posicionamiento, observándose dos excepciones: `bunny` y `dragon stand`. El aumento en el error del modelo `bunny` se debe a que la captura `bun180` presenta una distancia cercana a 90°, superando las restricciones impuestas en este trabajo. Sin embargo, en el caso de `dragon stand` el error refleja una mala alineación en la captura 12, debida a una mala selección de los parámetros. Mediante un posterior ajuste de los parámetros, en particular, del tamaño de la vecindad para el cálculo de los descriptores, se obtuvo una alineación correcta.

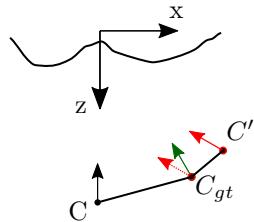


Figura 6.1: Comparación entre las transformaciones de alineación. Se observa el efecto producido en un punto orientado C que simula la posición de la cámara.

Modelo	Eye	Target (grados)	Up (grados)
armadillo			
back	0.0062159	0.221725	0.15211
head	0.0036356	0.102321	0.211231
head offset	0.0029806	0.086309	0.229574
stand	0.0022145	0.049612	0.105862
stand flip	0.0045019	0.125330	0.146033
bunny			
	0.0104809	0.598354	0.817185
dragon			
side	0.0070872	0.178650	0.212932
stand	0.0536199	1.379760	0.207754
up	0.0058265	0.139297	0.0675651
drill			
	0.0082317	0.238639	0.100126
happy			
back	0.0088540	0.189885	0.207247
side	0.0072675	0.175860	0.17525
stand	0.0050124	0.101383	0.097800

Cuadro 6.1: Errores de registración.

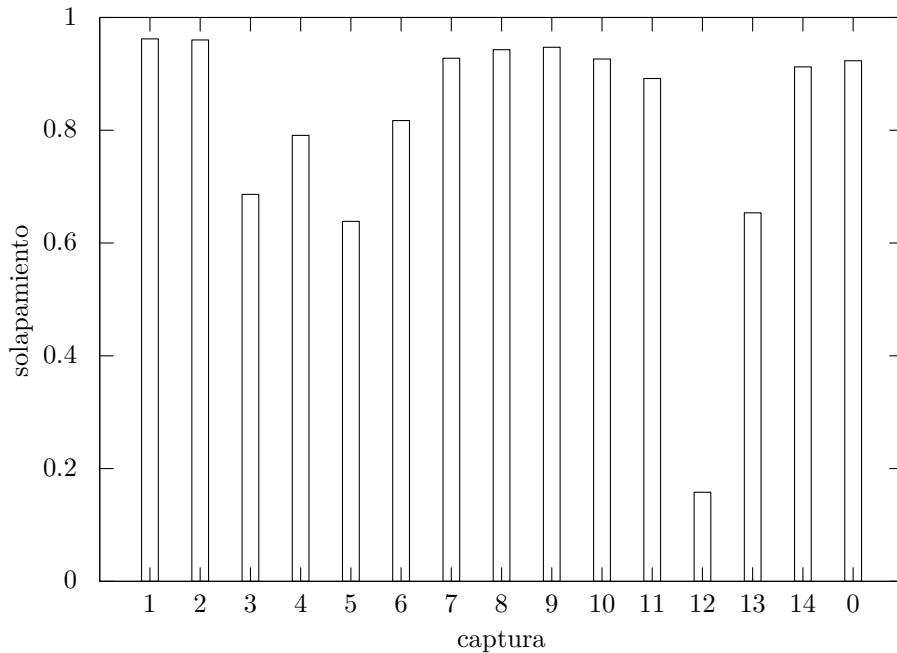


Figura 6.2: Métrica de alineación para el modelo `dragon stand`. El bajo porcentaje de solapamiento en la captura 12 se corresponde con un error de registración.

Para evaluar la alineación entre un par de capturas prescindiendo del *ground truth*, y situándonos en un escenario más realista, se diseñó una medida de *fitness*. Esta medida se define como el porcentaje del área solapada entre las nubes una vez alineadas, donde un bajo solapamiento nos indicaría un posible error de alineación, como se observa en la figura 6.2.

2 Módulo de fusión

Como medida de error de la fusión se utilizó la distancia entre los puntos de la nube reconstruida respecto al punto más cercano en el *ground truth* (cuadro 6.2). Esta medición no se realizó para el modelo `armadillo` debido a que su reconstrucción se encontraba a una escala distinta a la de las capturas. Nuevamente se destaca el error de `dragon stand` producto de una mala alineación.

En todos los modelos, se observa, además, una inflación/deflación de los objetos reconstruidos debida a la propagación del error de alineación. Así, la primera captura coincide casi exactamente, pero el error se incrementa a medida que nos alejamos de ella (figura 6.3).

Modelo	Error promedio	Desvío
bunny	1.28464	0.74131
dragon side	1.19651	0.69846
dragon stand	2.83930	2.41398
dragon up	1.14363	0.88966
drill (contra vrip)	1.48515	0.96336
drill (contra zip)	1.74326	1.39883
happy back	1.65632	1.27056
happy side	1.35371	1.01163
happy stand	1.79513	1.25758

Cuadro 6.2: Errores en la fusión, normalizados respecto a la resolución de las capturas.

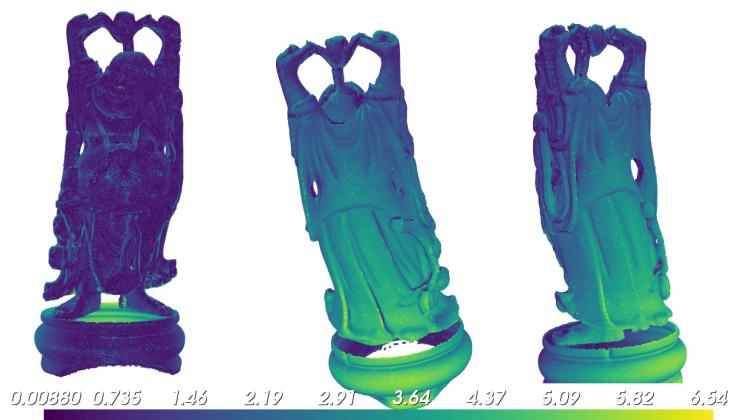


Figura 6.3: Diferencia contra el *ground truth* del modelo happy.

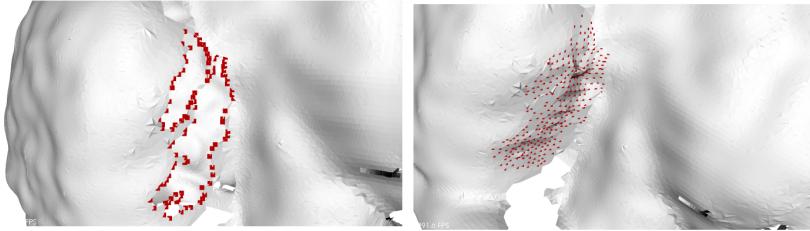


Figura 6.4: Relleno de un hueco pequeño mediante *advancing front*.

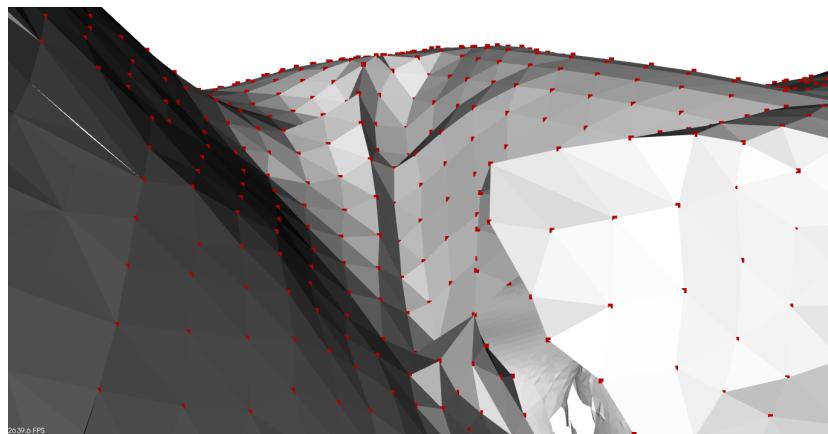


Figura 6.5: Fallo en el algoritmo de *advancing front*. Se intentó completar un triángulo con un punto que no pertenecía al borde.

3 Módulo de relleno de huecos

3.1 Método de advancing front

Al utilizar el método de advancing front sobre la superficie reconstruida de `bunny`, se logró el relleno de agujeros pequeños, obteniéndose una malla regular (figura 6.4). Sin embargo, debido a la localidad con la que se generan los nuevos puntos, el frente puede diverger o pretender unirse a puntos que no forman parte del contorno del hueco, resultando una malla mal formada, con aristas que corresponden a más de dos caras (figura 6.5). Por estas razones, el método no resulta adecuado para el relleno automático.

3.2 Reconstrucción de Poisson

Como se mencionó anteriormente, la reconstrucción de Poisson nos garantiza el relleno de todos los huecos (a excepción de la base) mediante una superficie suave. Se procedió, entonces, a una valoración visual de los objetos reconstruidos (figura 6.6):

- En `bunny` (figura 6.7) se observan desperfectos debidos a una mala registración de la captura `bun180`, que se encontraba aproximadamente a 90° respecto a sus vecinos.



Figura 6.6: Resultado de las reconstrucciones luego del relleno de huecos mediante el método de Poisson. De izquierda a derecha y de arriba a abajo, los modelos son: **armadillo**, **bunny**, **dragon**, **drill** y **happy**.

- En **drill** (figura 6.8) se tienen componentes inconexas debido a una mala fusión en una zona de alta curvatura.
- En **dragon**(figura 6.9) se observa la creación de un puente entre dos regiones. Esta es una de las limitaciones conocidas del método, al no poder incorporar la información de línea de vista de las capturas[KBH06].
- En todos los casos, la base de apoyo del objeto presenta una deformación hacia abajo (figura 6.10) con un hueco al final, producto de utilizar condiciones de borde Neumann. Debido a que la deformación ocurre en la base de apoyo, ésta puede corregirse al ignorar los puntos que presenten una coordenada y menor a la mínima de la nube de entrada.

4 Tiempo de ejecución

Para cada modelo se midieron los tiempos de ejecución requeridos por cada módulo de la reconstrucción sobre una PC Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz con 8GB de RAM corriendo en un solo hilo.

Se observa que la etapa de registración es responsable de la mayor parte del costo computacional, sobre todo al aumentar la cantidad de puntos en las capturas (cuadro 6.3). En el cuadro 6.4 se muestran los tiempos de ejecución promedio, discriminados en la alineación inicial y el refinamiento posterior. El

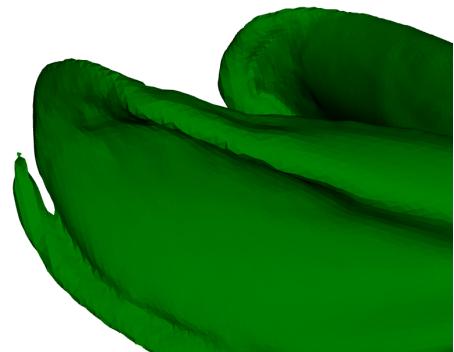


Figura 6.7: Acercamiento a la oreja derecha de **bunny**.



Figura 6.8: Acercamiento a la mecha de **drill**.



Figura 6.9: Acercamiento al vientre de **dragon**.

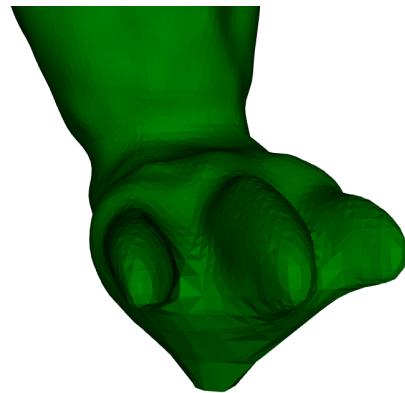


Figura 6.10: Acercamiento a la base de apoyo de `armadillo`. Se observa un estiramiento hacia abajo debido al uso de condiciones de borde Neumann.

orden $\mathcal{O}(n^2)$ de la alineación inicial se debe a la búsqueda entre todos los pares de puntos para establecer las correspondencias (figura 6.11).

Modelo	N	Puntos	Registración	Fusión	Rellenado	Total
armadillo						
back	11	25e3	84.4308	10.759	8.562	103.752
head	12	25e3	114.4926	12.501	9.892	136.886
head offset	11	25e3	100.7846	11.987	9.718	122.490
stand	12	25e3	102.3553	12.145	9.498	123.998
stand flip	11	25e3	101.7061	12.958	9.280	123.944
bunny	6	35e3	92.2872	11.246	11.862	115.395
dragon						
side	15	20e3	90.5427	14.021	8.086	112.650
stand	15	30e3	180.8010	23.485	12.680	216.966
up	15	30e3	164.1075	18.469	9.669	192.245
drill	12	4e3	4.3172	1.988	4.176	10.481
happy						
back	15	45e3	343.2705	29.618	7.528	380.417
side	15	45e3	452.5830	32.588	8.361	493.532
stand	15	75e3	907.2930	44.570	10.929	962.792

Cuadro 6.3: Tiempo de reconstrucción por cada modelo (en segundos).

Modelo	Puntos	Inicial (s)	ICP (s)	Total (s)
armadillo				
back	25e3	7.05694	0.618592	7.67553
head	25e3	8.94947	0.591576	9.54105
head offset	25e3	8.62241	0.539835	9.16224
stand	25e3	7.98686	0.542746	8.52961
stand flip	25e3	8.69949	0.546518	9.24601
bunny	35e3	14.0323	1.348890	15.3812
dragon				
side	20e3	5.52965	0.506529	6.03618
stand	30e3	11.3732	0.680195	12.0534
up	30e3	10.3322	0.608282	10.9405
drill	4e3	0.26898	0.090796	0.35977
happy				
back	45e3	21.2702	1.614520	22.8847
side	45e3	29.1103	1.061880	30.1722
stand	75e3	59.3281	1.158030	60.4862

Cuadro 6.4: Tiempos de ejecución promedio para la registración de a pares en los distintos modelos.

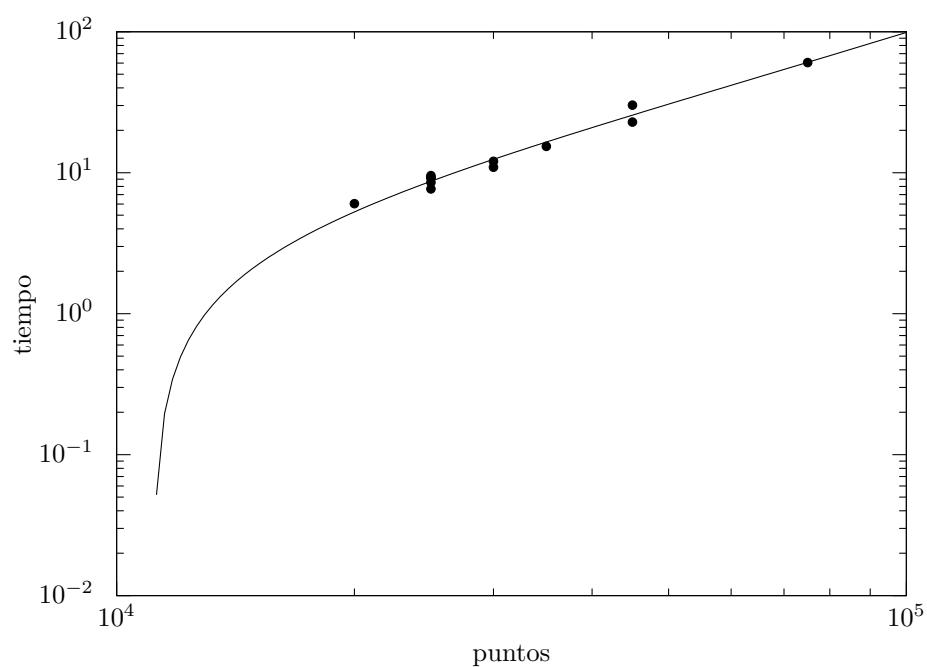


Figura 6.11: Tiempo de ejecución de la registración, ajustado a una función de orden $\mathcal{O}(n^2)$.

Conclusiones y trabajos futuros

1 Conclusiones del producto

En este proyecto se realizó el desarrollo de una biblioteca de software para lograr la reconstrucción tridimensional de un objeto a partir de capturas de vistas parciales. Para esto, se dividió el problema en tres módulos: registración, fusión y relleno de huecos, y se implementaron diversos algoritmos.

A pesar de que las capturas no contenían información de textura, el algoritmo de registración fue exitoso en casi todos los casos sin requerir ajustes a su conjunto de parámetros. Además, se cuenta con medidas de la calidad de la alineación, que permiten detectar fallas durante esta etapa sin requerir de una inspección visual.

Las reconstrucciones fueron obtenidas en tiempos razonables y sin requerir hardware especial. Si bien se observa un efecto de «inflación/deflación» debido a la propagación de los errores de registración, este se encuentra suficientemente acotado respecto al tamaño del objeto.

Debido a que se consideró solamente una posición del objeto sobre la base giratoria, se presenta una gran cantidad de occlusiones, lo que genera la aparición de huecos de tamaño considerable en la superficie reconstruida luego de la fusión. Aún así, el resultado final es una malla cerrada (a excepción de la base), y la superficie estimada en las zonas sin información se une suavemente al resto.

2 Conclusiones del proceso

Se tuvieron problemas al implementar la metodología seleccionada. El tiempo invertido en la etapa de investigación bibliográfica fue demasiado extenso y se desperdiciaron recursos al abordar el tratamiento de la información de textura, que finalmente debió ser descartada al no contar con un repositorio propio. Además, se produjo un desfasaje temporal entre la adquisición de los conocimientos y la implementación de los mismos, requiriendo un nuevo análisis. Estos inconvenientes se hubieran resuelto al utilizar directamente una metodología incremental en todo el proceso, con más incrementos de menor tamaño, como ser agregar un primer módulo de preprocessamiento que contenga la reducción de ruido y la operatoria básica con las nubes de puntos.

En cuanto al desarrollo, uno de los principales problemas fue la definición de métricas para evaluar los algoritmos y establecer los niveles de error aceptables en cada etapa. Muchas evaluaciones fueron primeramente visuales, resultando en un proceso lento que en ocasiones fallaba en detectar errores considerables.

Durante el desarrollo se efectivizó otro de los riesgos identificados para el proyecto: la falla en los equipos de trabajo requiriendo su reemplazo. Gracias a las copias de respaldo periódicas, fue posible recuperar fácilmente el trabajo realizado hasta ese momento. Sin embargo, debido a que el nuevo equipo de trabajo contaba con otro sistema operativo (Clear Linux), se requirió de un largo proceso de configuración para instalar la biblioteca PCL a partir de sus archivos fuentes.

3 Trabajos futuros

En esta sección se describen actividades que excedieron el alcance de este proyecto y podrían ser abordadas en una etapa posterior.

- Ajustar los métodos de registración para combinar escaneos del objeto en varias posiciones sobre la base giratoria, buscando de esta forma eliminar huecos y reducir la propagación del error de alineación. Esto requerirá eliminar la restricción del eje de giro en la registración y ajustar el algoritmo de corrección de bucle.
- Ajustar los métodos para trabajar con el volumen de puntos generados por [Mai19]. Es necesario analizar la robustez del algoritmo al submuestreo de la entrada, realizar una selección de keypoints de las nubes de entrada y utilizar un método más eficiente para la búsqueda de correspondencias.
- Paralelizar los algoritmos utilizados en el módulo de registración, en particular la obtención de descriptores y la determinación de correspondencias.
- Utilizar la métrica de fitness para el ajuste automático de los parámetros utilizados en la registración.
- Implementar métricas de calidad del mallado que consideren las características de la impresión 3D. En este proyecto solamente se consideraron las condiciones de que la malla resultase cerrada y no presente intersecciones consigo misma.
- Ajustar los métodos de relleno de huecos para evitar la creación de «puentes».
- Modificar el método de *advancing front* para que utilice una superficie de soporte para establecer la posición de los nuevos puntos, asegurando de esta forma la convergencia del método y la suavidad del parche generado.
- Modificar el método de *advancing front* para que considere las islas. Esto requerirá detectar dentro de qué hueco se encuentra cada isla.

Bibliografía

- [BC94] J. Berkmann and T. Caelli. Computation of Surface Geometry and Segmentation Using Covariance Techniques. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(11):1114–1116, November 1994.
- [BM92] Paul Besl and H.D. McKay. A method for registration of 3-D shapes. *IEEE Trans Pattern Anal Mach Intell. Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14:239–256, 03 1992.
- [CCC⁺08] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.
- [CL96] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.
- [CM92] Yang Chen and Gérard Medioni. Object Modeling by Registration of Multiple Range Images. *Image Vision Comput.*, 10:145–155, 01 1992.
- [Hor87] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.
- [IKH⁺11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.
- [KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.
- [KH] Michael Kazhdan and Hugues Hoppe. Screened Poisson Surface Reconstruction.

- [Low04] Kok-Lim Low. Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration. 01 2004.
- [Mai19] Francisco Mainero. Desarrollo de una biblioteca de Software para Reconstrucción y Rectificación de Superficies, 2019.
- [MG] S. Ladislao Mathe and G. Gomez. Estudio del funcionamiento del sensor kinect y aplicaciones para bioingeniería.
- [PL14] Claudia Raluca Popescu and Adrian Lungu. Real-Time 3D Reconstruction Using a Kinect Sensor. *Computer Science and Information Technology 2.2*, pages 95–99, 2014.
- [RBB09] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation, ICRA'09*, pages 1848–1853, Piscataway, NJ, USA, 2009. IEEE Press.
- [RC11] R.B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4, May 2011.
- [RHHL02] Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. Real-Time 3D Model Acquisition, 2002.
- [Rus09] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Technische Universität München, 2009.
- [SKC13] Frank Steinbrücker, Christian Kerl, and Daniel Cremers. Large-Scale Multi-resolution Surface Reconstruction from RGB-D Sequences. In *Proceedings of the 2013 IEEE International Conference on Computer Vision, ICCV '13*, pages 3264–3271, Washington, DC, USA, 2013. IEEE Computer Society.
- [Sta] The Stanford 3D Scanning Repository. <http://graphics.stanford.edu/data/3Dscanrep/>.
- [SvLD14] Carl Schubert, Mark C van Langeveld, and Larry A Donoso. Innovations in 3D printing: a 3D overview from optics to organs. *British Journal of Ophthalmology*, 98(2):159–161, 2014.
- [TL94] Greg Turk and Marc Levoy. Zippered Polygon Meshes from Range Images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94*, pages 311–318, New York, NY, USA, 1994. ACM.
- [VMC97] Tamás Várady, Ralph R. Martin, and Jordan Cox. Reverse Engineering of Geometric Models. *An Introduction, ComputerAided Design*, 29:255–268, 1997.

- [WCLL11] Xiaochao Wang, Junjie Cao, Xiuping Liu, and Baojun Li. Advancing front method in triangular meshes hole-filling application. *Jisuanji Fuzhu Sheji Yu Tuxingxue Xuebao/Journal of Computer-Aided Design and Computer Graphics*, 23:1048–1054, 06 2011.
- [WWLV09] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1630–1637, Sep. 2009.
- [Zho09] Yu Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. pages 689–696, 11 2009.

Uso de la biblioteca

En este apéndice se presenta una reseña de la biblioteca desarrollada. Se explica brevemente la instalación, compilación y el uso de las funciones fundamentales que provee cada módulo de la misma.

1 Instalación y compilación

La biblioteca se encuentra contenida en archivos de cabecera, los cuales serán incluidos por los fuentes del proyecto. Se requiere la instalación de las siguientes dependencias:

- PCL versión 1.9 (<https://pointclouds.org/downloads/>).
- delaunator-cpp versión 0.4 (<https://github.com/delfrrr/delaunator-cpp>).
- DKM (<https://github.com/genbattle/dkm>).

Para la compilación se recomienda el uso de la herramienta `cmake` con el siguiente esqueleto para su archivo de configuración `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
project(MI_PROYECTO)
find_package(PCL 1.9 REQUIRED)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(programa.bin main.cpp)
target_link_libraries(programa.bin ${PCL_LIBRARIES})
```

2 Representación de las capturas

La clase `cloud_with_transformation` agrupa la información de posición y normales de cada punto, y la transformación de alineación. Durante la registración, las posiciones y normales se mantienen en dos nubes separadas, sin embargo, los módulos de fusión y relleno de huecos requieren esta información integrada en una sola nube. Para convertir a este segundo formato se provee la función `join_cloud_and_normal(nube)`.

3 Módulo de preprocesso

- Carga la nube desde un archivo de disco y la prepara para ser usada por el módulo de registración

```
auto nube = load_cloud_ply("bun000.ply");
// suavizado del ruido
double resolucion = cloud_resolution(nube);
auto nube_sin_ruido = moving_least_squares(nube,
                                             4 * resolution);
// eliminación de puntos aislados, puntos de borde
// y puntos con normales alejadas a la cámara
auto nube_limpia = preprocess(nube_sin_ruido);
```

4 Módulo de registración

- Alineación inicial de dos nubes

```
alignment alineacion_inicial;
alineacion_inicial.set_sample_ratio(0.25);
alineacion_inicial.set_feature_radius(6);
alineacion_inicial.set_y_threshold(4);
alineacion_inicial.set_axis_threshold(10 * M_PI / 180);

// source y target son dos nubes
// a las que se les realizó el preprocess
auto transformacion = alineacion_inicial.align(source, target);
source.set_transformation(transformacion);
```

- Ajuste mediante el algoritmo de ICP

```
//modifica source
icp_correction(source, target, resolucion);
```

- Corrección de bucle. Se realiza sobre un vector que contiene todas las capturas realizadas alrededor del objeto.

```
std::vector<cloud_with_transformation> registered;
// ...
transformations_relative_to_absolute(registered);
loop_correction(registered);
```

- Medición de la calidad de la registración

```
for (size_t K = 1; K < registered.size(); ++K) {
    auto &prev = registered[K - 1];
    auto &current = registered[K];
    current.apply_transformation();
    auto [error_medio, desvio, solapamiento] =
        fitness(current.get_cloud(),
                 prev.get_cloud(),
                 4 * resolucion);
    //...
}
```

5 Módulo de fusión

- Fusión de las capturas alineadas y triangulación de la nube resultante

```
double umbral = 1.5 * resolucion;
auto result = fusionar(registered, umbral);

double longitud_maxima_de_arista = 5 * resolucion;
auto tmesh = triangulate_3d(result.get_cloud(),
                             5 * resolution);

write_cloud_ply(*result.get_cloud(), "bunny_fusion.ply");
write_polygons(*result.get_cloud(),
               tmesh,
               "bunny_fusion.polygon");
// formato de PCL, pierde información de normales
auto polygon_mesh = tmesh_to_polygon(*result.get_cloud(),
                                      tmesh);
pcl::io::savePolygonFilePLY("bunny_fusion_pmesh.ply",
                            polygon_mesh,
                            false);
```

6 Rellenado de huecos

- Detección de huecos

```
auto [cloud, mesh] = load_triangle_mesh("bun_fusion.ply",
                                         "bun_fusion.polygon");
extract_biggest_connected_component(cloud, mesh);
auto boundary_points = boundary_points(mesh);

• Rellenado mediante el método de advancing front. ¡Advertencia! El algoritmo falla excepto en huecos pequeños o planos. No se recomienda su uso
advancing_front hole_filling(cloud, mesh, boundary_points);
auto patch = hole_filling.fill(index);
```

7 Reconstrucción de Poisson

- Utilización del algoritmo de reconstrucción de Poisson implementado en la biblioteca PCL

```
auto nube = create<cloudnormal>();
pcl::io::loadPLYFile<pointnormal>("bunny_fusion.ply", *nube);

pcl::Poisson<pointnormal> poisson;
poisson.setInputCloud(nube);
pcl::PolygonMesh mesh;
poisson.setDepth(8);
poisson.reconstruct(mesh);
```

```
// al reconstruir se pierde la información de normales
pcl::io::savePolygonFilePLY("bunny_reconstructed.ply",
                             mesh,
                             false);
```