Diseño de un esquema de muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos

Sebastián Arboleda Duque

Universidad Nacional de Colombia Sede Manizales

Facultad de Ingeniería y Arquitectura, Departamento de Ingeniería Eléctrica y Electrónica

Manizales, Caldas, Colombia

2015

Diseño de un esquema de muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos

Sebastián Arboleda Duque

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:

Magister en Ingeniería – Automatización Industrial

Director:

Ph.D. Juan Bernardo Gómez Mendoza

Línea de Investigación:

Computación Gráfica (Ciencias de la Computación)

Grupo de Investigación:

Computación Aplicada Suave y Dura (SHAC)

Universidad Nacional de Colombia Sede Manizales

Facultad de Ingeniería y Arquitectura, Departamento de Ingeniería Eléctrica y Electrónica

Manizales, Caldas, Colombia

2015

Agradecimientos

Me gustaría expresar mi gratitud a mi tutor de tesis Ph.D. Juan Bernardo Gómez Mendoza por los valiosos comentarios, observaciones y la participación a través del proceso de aprendizaje de esta tesis de maestría, también agradecerle por haberme introducido en el tema, así como por el apoyo en el camino y por haber compartido voluntariamente su tiempo precioso durante este proceso. Además, me gustaría dar las gracias a mis compañeros de maestría que han compartido sus conocimientos conmigo durante este tiempo. Me gustaría dar las gracias a mi familia, que me ha apoyado a lo largo de todo el proceso, especialmente a mi madre que siempre ha sido la columna vertebral de mis estudios. Finalmente, agradecer a mi padre que siempre ha sido mi fuente de inspiración y modelo a seguir, que en paz descanse.

Resumen

En este documento se presenta una técnica para simplificar nubes de puntos, la cual se basa en la información espacial contenida en una nube de puntos no estructurada. Para lograr esto, se pretende reorganizar el conjunto de datos de manera tal que al inicio del archivo se ubiquen los puntos más significativos para el modelo 3D, esto permite que el conjunto de datos pueda ser cargado, visualizado o almacenado de forma progresiva. Para clasificar los puntos de acuerdo a su importancia dentro del conjunto de datos, el modelo 3D es dividido en segmentos relativamente planos usando un algoritmo de crecimiento de regiones, luego cada región es analizada para clasificar los datos dentro de ella como puntos pertenecientes a algún borde o puntos pertenecientes a zonas intermedias de la superficie. Finalmente se realiza un muestreo de los datos clasificados y se ordenan en el archivo. De esta forma el usuario puede tener el control de la cantidad de puntos que desea cargar del archivo y así se obtiene un modelo de diferentes resoluciones.

Palabras clave: *Simplificación de nubes de puntos, estructuras de datos espaciales, sistemas multirresolución, algoritmo de niveles de detalle, carga progresiva.*

Abstract

Design of a data sampling scheme for the progressive loading and storage of point clouds. In this document a technique to simplify point clouds is presented, which is based in the spatial information contained in a unstructured point cloud. To achieve this, the intention is to reorganize the data set in such a way that the most significant dots for the 3D model are located at the beginning of the file, which allows that the data set can be uploaded, viewed or stored in a progressive way. To classify the dots according to their importance within the data set, and in this way re arrange the file, the 3D model is split into segments relativity plane using a region growing algorithm, next each region is

analyzed to classify the data inside it as dots belonging to any border or dots belonging to intermediate zones of the surface. Finally, a sampling of the classified data is made and they are arranged in the file. In this way the user can control the amount of dots that he wants to upload from the file and so he gets a model of different resolutions.

Keywords: *Point cloud simplification, spatial data structures, multiresolution systems, level of detail algorithms, progressive loading.*

Contenido

Lista de figuras

Pág.

Lista de tablas

Pág.

1. Introducción

La visión artificial o visión por computadora es una área de gran interés debido a que los humanos percibimos nuestro entorno en términos de imágenes. Esta área de estudio está compuesta por una gran cantidad de procesos que sirven para mejorar la información gráfica y de este modo facilitar la interpretación por el ojo humano. Estos procesos también pueden ser usados para almacenar, transmitir y/o adecuar la información en sistemas de percepción autónomos, este campo de estudio se conoce con el nombre de procesamiento digital de imágenes. Una de las ramas de la visión por computadora es el área que se encarga del procesamiento de imágenes en tres dimensiones (3D), las cuales representan la entrada básica a un sistema de percepción en formato 3D [**1**]. Cada una de las muestras adquiridas, además de su correspondiente coordenada (x, y, z) en el espacio, pueden llevar asociada información de carácter físico, como color de la superficie, índice de reflectividad del material, temperatura, etc. Al conjunto combinado de todos estos datos se le denomina *Point Cloud* o *Nube de Puntos*.

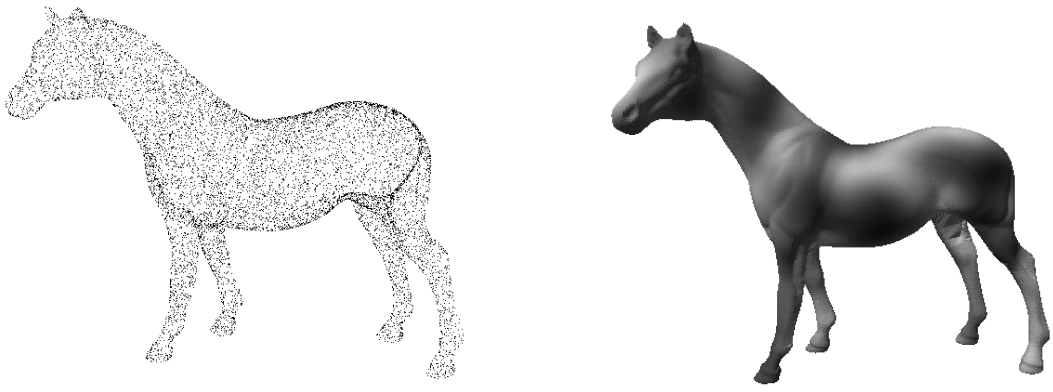


Figura 1.1. A la izquierda de la figura se observa una nube de puntos sin una malla definida, mientras que el modelo de la derecha si posee una malla definida.

Pero, ¿por qué utilizar nubes de puntos si existen imágenes en 2D las cuales son mucho más sencillas de procesar?, simple, porque el mundo real es en 3D no en 2D, las imágenes en dos dimensiones no siempre infieren suficiente información del mundo real, en cambio las nubes de puntos entregan información precisa, como por ejemplo las dimensiones reales del modelo. El propósito de estos conjuntos de datos habitualmente es ser objeto de algoritmos de carácter espacial que pueden sustraer información relevante para un usuario, como por ejemplo adquirir planos CAD (*Computer Aided Design*), reconstruir superficies y curvas de nivel o analizar estructuras civiles. Las nubes de puntos pueden representar explícitamente superficies, por ejemplo, en ciencias de la tierra [**2**], superficies volumétricas o iso-superficies de datos como en aplicaciones médicas [**3**] o en la visualización de campos vectoriales [**4**].

Además sus aplicaciones van desde el entretenimiento hasta la investigación científica, son usadas por compañías como PIXAR® para realizar películas animadas, pueden recrear un museo como el museo de ciencia de Londres, son usadas también para simular partes mecánicas o estructuras civiles, recrear mapas de ciudades en 3D o por ejemplo, utilizarlas para que un robot pueda conocer su entorno con precisión; sus aplicaciones son bastante amplias [**1**].

Los recientes avances en las técnicas de adquisición de nubes de puntos, permiten que las representaciones 3D del mundo real sean digitalizadas con precisión milimétrica. Existen, en la actualidad, digitalizadores 3D para la adquisición de superficies con la capacidad de producir nubes de puntos que contienen millones de muestras del modelo real [**5**] y dado que cada vez se utilizan para problemas más complejos que requieren mayor precisión, esto puede llegar a suponer varias decenas o centenas de Gigabytes de información lo cual es computacionalmente muy costoso sólo para almacenarla, incluso en los últimos niveles de la jerarquía de memoria (discos duros) [**6**], hasta la visualización 3D más simple se torna problemática debido a su gran tamaño.

1.1 Motivación

Como se mencionó anteriormente, una de las principales dificultades que ha obstaculizado el desempeño en el procesamiento de modelos 3D es la gran cantidad de datos. Por ejemplo, los investigadores del proyecto de digitalización del David de Miguel

Ángel [**5**], escanearon la estatua con el propósito de conservar un modelo digital para el estudio de su estructura, este trabajo necesitó cientos de millones de puntos. Por otra parte, los sistemas LIDIAR [**6**] (*Light Detection And Ranging*), recolectan puntos en el espacio a partir de un sensor ubicado en una aeronave, estos sistemas capturan billones de puntos de la superficie muestreada. Manejar esta cantidad de datos, se ha convertido en un desafío para poderlos procesar, visualizar o transmitir.

La salida más cruda de un digitalizador 3D consiste generalmente en una nube de puntos que tiene una redundancia considerable en sus datos. Parte del proceso típico para el procesamiento de una modelo 3D consiste en convertir una nube de puntos en una malla construida a partir de polígonos a través de algoritmos de reconstrucción de superficies, los cuales son computacionalmente costosos. Reducir la complejidad de estos conjuntos de datos es un punto clave en las técnicas de pre-procesamiento para poder visualizarlas o trabajar con ellas. Es una práctica común realizar procesamientos posteriores para modificar o simplificar el tamaño de estas superficies basadas en modelos poligonales, a menudo la simplificación de las mallas resulta tan costosa computacionalmente que se vuelve un proceso difícil de realizar [**7**] [**8**] [**9**].

Este es el caso del proyecto del David de Miguel Ángel, desde el año 1992 el profesor Marc Levoy y su grupo de estudiantes han investigado en métodos para la digitalización de objetos usando digitalizadores 3D basados en escáneres que proyectan una luz láser sobre el modelo real [**10**]. En 1997 comenzaron a trabajar en el proyecto del David de Miguel Ángel, para el año 1999 el modelo digital poseía 2 mil millones de polígonos y 7000 imágenes en color. Este volumen de información era algo que difícilmente se podía visualizar en un computador de aquella época, por tal motivo surgió la necesidad de desarrollar un software que tuviera la capacidad de procesar esta información de una manera eficiente. Por tal motivo durante el curso del proyecto del David, se diseño un programa para la visualización de modelos geométricos de gran volumen de información, a este proyecto se le conoce con el nombre de *QSplat*.

QSplat [**11**], permite el renderizado de modelos que poseen millones de datos de información espacial, ya que se basa en técnicas que cargan el modelo de forma progresiva. Este combina técnicas de jerarquías de estructuras de datos basado en

esferas envolventes y un sistema de renderización basado en puntos primitivos de la tarjeta de memoria gráfica. Este tipo de sistemas se basaban en modelo poligonales.

Sin embargo, el surgimiento de nuevas tecnologías ha conducido al desarrollo de sistemas de adquisición 3D más sofisticados, debido a esto los modelos de nubes de puntos poseen cada vez un mayor número de muestras de la superficie. Por tal motivo, los polígonos que se forman entre puntos son cada vez más pequeños, esto ha incrementado el interés en representaciones basadas en nubes de puntos sin una malla definida. Los polígonos en la superficie han sido reducidos en tamaño para obtener mejor precisión y fidelidad de estos modelos, consecuentemente durante la renderización la resolución de la pantalla se ve saturada y muchos de estos polígonos son innecesarios, por esta razón los modelos basados en nubes de puntos pueden llegar a ser más eficientes en este tipo de casos [**12**].

Este es el caso de Moenning y Dodgson [**13**], los cuales presentan un algoritmo para la simplificación de nubes de puntos en el cual se obtienen diferentes densidades del conjunto de datos, el algoritmo es computacionalmente eficiente y no requiere que la nube de puntos tenga una malla construida previamente, este algoritmo hace uso de una técnica desarrollada por los mismos autores conocida con el nombre de *Fast Marching farthest point sampling for implicit surfaces and point clouds* [**14**]. Esta técnica se basa en diagramas de Voronoi para muestrear la nube de puntos, los resultados de esta técnica son muy buenos, sin embargo los nuevos puntos no tienen las mismas coordenadas de los puntos originales ya que el conjunto de datos nuevo es una versión muestreada de la nube de puntos original. El la **¡Error! No se encuentra el origen de la referencia.** de [**13**], Moenning y Dodgson presentan se una nube de puntos en varias resoluciones muestreada con esta técnica.

Otra técnica es propuesta por Dey et al. [**15**], en la cual plantean un algoritmo para simplificar nubes de puntos el cual detecta puntos redundantes en el conjunto de datos de entrada basándose en diagramas de Voronoi y características locales de los puntos. El objetivo principal de este trabajo es eliminar puntos dentro del conjunto de datos y determinar un parámetro que sirve para determinar el nivel de diezmado del conjunto de datos. Sin embargo este método es muy sensible a los valores de la curvatura, además

necesita mantener durante toda su ejecución los diagramas 3D de Voronoi, lo cual tiende a ser computacionalmente costoso y necesita mucho espacio en memoria. En la Figura 2 de [**15**], Dey et al. presentan algunos resultados de esta técnica.

Por otra parte, Boissonnat and Cazals [**16**], presentan un algoritmo para simplificación de nubes de puntos, este algoritmo toma un conjunto de puntos aleatorio del conjunto de datos original y usa triangulación de Delaunay para definir una función de distancias alrededor del conjunto. Esta función implícita se utiliza para extender el conjunto inicial hasta que se encuentre una cantidad de puntos que estén dentro de una tolerancia de error aproximado definido por el usuario. Posteriormente, se construye una malla a partir del subconjunto extendido utilizando triangulación de Delaunay [**17**]. Si la superficie reconstruida no cumple con la condición de error preestablecida, se insertan puntos adicionales, organizados a partir de su distancia con respecto a la cara más cercana de la superficie. Este método para simplificar nubes de puntos es computacionalmente costoso, puesto que se debe mantener la triangulación de Delaunay durante todo el proceso. Por lo tanto no aprovecha un procesamiento basado únicamente en la nube de puntos, ya que este está restringido a la triangulación.

Mohan y Narayanan presentan un algoritmo de descomposición progresiva de nubes de puntos sin planos locales [**12**], el cual tiene como objetivo el reordenamiento de los datos para descomponer la nube de puntos en diferentes niveles de resolución basándose en la proximidad que existe entre los puntos. Este método reordena los puntos en conjuntos de aproximaciones de los datos y conjuntos de niveles de detalles, algo parecido a la descomposición a través de la transformada de wavelet.

Los puntos se reordenan de forma recursiva basándose en un emparejamiento óptimo usando la técnica propuesta en [**18**], viendo el conjunto de datos como si fuera un grafo. Luego realizan una cuantización balanceada en cada nivel de división, esto resulta en una mayor compresión de los datos. Después de cuantizar los datos, realizan una compresión sin pérdida usando un codificador aritmético [**19**].

Finalmente también plantean un esquema para la representación progresiva de los puntos usando la transformada discreta de wavelet para comprimir los vectores de detalles. Con este último proceso se pueden incluir selectivamente diferentes números de

coeficientes de la descomposición de wavelet en cada nivel de detalle y así se van añadiendo detalles selectivamente de forma que la nube de puntos se va refinando a medida que se le añaden puntos, hasta llegar de nuevo a la nube original.

Este esquema no requiere de los planos o reconstrucción de la superficie para codificar o decodificar el conjunto de datos, además la posición de los puntos se mantiene en sus coordenadas originales en todos los niveles de descomposición y la nube de puntos original puede ser reconstruida sin pérdida a partir de la descomposición.

Pauly et al. [**20**] adaptan varias técnicas comunes para la simplificación de mallas a la simplificación de nubes de puntos sin una malla definida. Uno de ellos es el método iterativo de simplificación basado en errores cuadráticos, el cual produce conjuntos de puntos con errores bajos de aproximación al modelo original, sin embargo este método es muy sensible en tiempo de ejecución al tamaño de la nube de puntos. Otro método propuesto es una técnica basada en la simulación de partículas para simplificar nubes de puntos, este método también presenta un error bajo al comparar la nube simplificada con el modelo original, no obstante es computacionalmente ineficiente. Pauly et al. además destacan dos métodos basados en la segmentación del conjunto de datos en regiones o *clusters*, la idea principal de este tipo de métodos es dividir el modelo en subconjuntos y reemplazar todos los puntos que pertenecen a un mismo subconjunto por un punto común. El primero es un método incremental uniforme el cual segmenta la nube de puntos usando un algoritmo de crecimiento de regiones, posteriormente se analiza cada segmento individualmente para simplificarlo. El segundo método es un algoritmo de agrupamiento jerárquico, el cual calcula los subconjuntos de la nube de puntos dividiéndola de forma recursiva usando una técnica conocida con el nombre de división binaria del espacio (BSP). Ambos métodos son computacionalmente eficientes, aunque el error con respecto al conjunto de datos original es un poco más alto que los otros métodos propuestos por ellos, además hay que tener en cuenta que estos métodos son sensibles a ciertas características asociadas a las nubes de puntos, conocidas con el nombre de *características principales*, las cuales serán abordadas más adelante.

Gobbetti y Marton, en su artículo *Layered Point Clouds* [**21**], exponen un sistema para la visualización y renderización de nubes de puntos. Dentro de su trabajo proponen una

técnica para ordenar la nube de punto en un árbol binario, de tal forma que cada nivel del árbol representa un nivel de detalle (*LOD*). En otras palabas, se construye un árbol jerárquico a partir de la nube de puntos, reordenando y agrupando el conjunto de puntos en subconjuntos, donde cada subconjunto es un nodo de un árbol binario. El modelo final tiene exactamente la misma cantidad de puntos y estos no son movidos de sus coordenadas originales, solo que este está dividido en segmentos y cada segmento corresponde a un nivel de detalle. Para construir el árbol, primero se escogen M cantidad de puntos uniformemente distribuidos sobre toda la nube de puntos y se agregan al nodo raíz del árbol, los puntos restantes pertenecerán a los nodos hijos de la raíz. Para definir los nodos hijos, se encierran la nube de puntos en una caja envolvente (*bounding box*) y se divide el conjunto de datos a lo largo del eje más largo, los puntos restantes a la izquierda de la división pertenecen al hijo de la izquierda y los de puntos restantes a la derecha de la división pertenecen consecuentemente al hijo de la derecha y se repite el mismo proceso con cada subconjunto, teniendo en cuenta que siempre se deben escoger M puntos distribuidos uniformemente sobre todo el subconjunto; por lo tanto los nodos hoja son aquellos que son indivisibles, ya que tienen una cantidad de puntos menor al límite preestablecido M. La raíz del árbol representa el modelo completo pero en baja resolución, esta estructura se puede comprender mejor en la Figura 1.2.

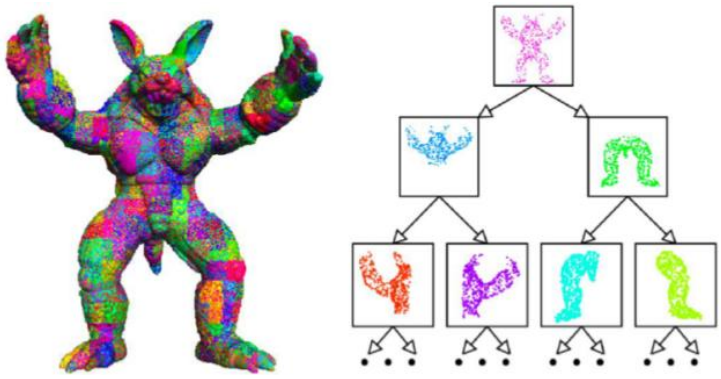


Figura 1.2. Esquema del árbol binario para generar niveles de detalles propuesto por Gobbetti y Marton. Esta figura fue tomada de [21].

Por lo tanto resulta interesante plantear una alternativa al proceso de simplificación de nubes de puntos antes que esta sea procesada para la reconstrucción de su superficie a través de polígonos. La simplificación de nubes de puntos es un proceso mucho más eficiente y tiende a ser computacionalmente menos demandante, además tiene un uso de memoria menor ya que no se tiene que mantener la estructura de los datos de la

malla. Una vez sea simplificada la nube de puntos, se puede proceder a realizar cualquier procesamiento sobre esta, es común que uno de los primeros pasos sea la reconstrucción de la superficie a partir de polígonos que conectan los puntos del conjunto de datos como el algoritmo de los cubos marchantes [**22**].

1.2 Estado del arte

Benhabiles et al. en su trabajo titulado *Fast simplification with sharp feature preserving for 3D point clouds* [**23**], proponen un algoritmo para simplificar nubes de puntos, este método está basado en una técnica hibrida que combina técnicas de segmentación y técnicas de *coarse-to-fine*, con el fin de identificar regiones que contengan bordes sobresalientes o agudos. La nube simplificada resulta en un conjunto de datos que posee baja densidad de puntos en las áreas planas y una mayor densidad de puntos en áreas de curvatura alta. Para evitar calcular la curvatura de todos los puntos, primero segmentan el modelo original en *voxels* de tamaño uniforme, posteriormente todos los puntos que están dentro de un mismo *voxel* son aproximados a su centroide y finalmente calculan la curvatura de los puntos de este nuevo conjunto de datos. Luego escogen los puntos de mayor curvatura en este nuevo conjunto de datos y seleccionan sus vecinos más cercanos de la nube original, los cuales son incluidos en el modelo nuevo. En la Figura 1 de [**23**] se ilustra este proceso. Sin embargo, este algoritmo no conserva la totalidad de los puntos del conjunto de datos original y no es posible revertir el proceso o recuperar el modelo original a partir del modelo resultante. Además, la técnica de simplificación usada es muy sensible al ruido.

Zhaoxia y Wenming en su documento *Kd-tree Based Nonuniform Simplification of 3D Point Cloud* [**24**], presentan una técnica para simplificar nubes de puntos de manera no uniforme, es decir que la densidad de puntos alrededor de toda la superficie no es la misma. Este método utiliza estructuras arboles k-dimensionales (*Kd-trees*) para calcular los vecinos más cercanos a cada punto y la normal de cada punto, luego se calcula la tasa de cambio del vector normal de cada punto con respecto a sus vecinos más cercanos y se eliminan los puntos que estén por debajo de un umbral σ definido por el usuario. Sin embargo esta es una simplificación con perdidas y no se puede controlar la

cantidad de puntos que se quieren eliminar. En la Figura 3 de [**24**], Zhaoxia y Wenming ilustran un modelo simplificado con esta técnica.

Li et al. presentan un algoritmo para simplificar nubes de puntos basado en la desviación estándar del vector normal en su artículo titulado *A Self-adaption Fast Point Cloud Simplification Algorithm Based on Normal Eigenvalues* [**25**]. Primero organizan los datos en un *Kd-tree* y encuentran los vecinos más cercanos a cada punto, en el articulo proponen encontrar 4 vecinos para cada punto y posteriormente calculan la distancia promedio entre cada punto y sus vecinos. Luego realizan un muestreo de puntos teniendo en cuenta la estructura del *Kd-tree*, las dimensiones de cada *voxel* y la distancia promedio entre puntos para simplificar el modelo tridimensional. Finalmente calculan los puntos de alta curvatura en la nube de puntos y combinan la nube de puntos simplificada con los puntos de alta curvatura. En la Figura 10 de [**25**], Li et al. presentan uno de los resultados de este método.

Xianglin y Mingyong en su trabajo *Point Sets Simplification Using Local Surface Analysis* [**26**], desarrollaron un algoritmo el cual preserva la información de cambios bruscos en la superficie, como por ejemplo las esquinas agudas del modelo original. El diezmado del conjunto de datos se realiza eliminando puntos de la nube original y no cargándolos en un nuevo modelo. El principio que utilizan para simplificar las nubes de puntos se basa en el análisis de la superficie subyacente a cada punto del conjunto de datos. Primero se calculan los k vecinos más cercanos a cada punto dentro del conjunto de datos y también calculan la densidad local de puntos ρ de cada vecindario. Luego organizan todas las muestras a partir de la densidad asociada a cada punto y comienza el proceso de simplificación a partir del punto p_i de menor densidad, con respecto a este último se elimina el vecino más cercano y se actualizan los datos de densidad y distancia del vecindario de p_i, los cuales son introducidos nuevamente a la lista de puntos organizada según la densidad local. Este proceso se repite hasta que se haya alcanzado el tamaño de reducción deseado del modelo. Una de las implementaciones de este algoritmo es ilustrada en la Figura 5 de [**26**].

XiWei et al. proponen un algoritmo que simplifica modelos tridimensionales a partir de las características principales de cada punto del conjunto de datos en su trabajo

Simplification of Scattered Point Cloud Based on Feature Extraction [**27**]. Primero dividen el conjunto de datos en cubos de tamaño uniforme con el fin de dividir el conjunto de datos en listas pequeñas y de este modo acceder más rápido a los datos espaciales, donde cada cubo es una lista. Posteriormente construyen esferas envolventes alrededor de cada punto del conjunto de datos, a partir de las cuales calculan los vecinos más cercanos. Para cada punto de la nube se calcula su curvatura y se etiquetan los puntos como puntos *característicos* y puntos *no característicos*; los puntos *característicos* los definen a partir del punto de mayor curvatura de cada esfera envolvente y el resto de puntos se etiquetan como *no característicos*, este proceso se realiza recursivamente hasta etiquetar todos los puntos.

El proceso de simplificación depende del radio de la esfera envolvente, si la distancia entre los puntos *no característicos* y los puntos *característicos* es igual o menor al radio de la esfera envolvente, esto quiere decir, según XiWei et al., que estos puntos *no característicos* son redundantes y se pueden eliminar.

Por otra parte Park and Lee [**28**], construyen un *octree* de profundidad uniforme, cada nodo almacena un plano el cual ha sido calculado a partir de una análisis de componentes principales a partir de los datos de cada nodo, luego codifican estos planos guardando la distancia a lo largo de la dirección normal de cada plano contenido en los nodos hijos con respecto al plano contenido en su nodo padre en toda la estructura *octree* usando el codificador propuesto por Shapiro en [**29**]. En la Figura 7 de [**28**], se ilustra la reconstrucción progresiva de un modelo 3D.

Un desarrollo similar es propuesto por Smith et al. [**30**], ellos presentan un algoritmo para compresión de superficies generadas a partir de nubes de puntos, en este desarrollo se construye un *octree* a partir de la nube de puntos, donde cada nodo hoja del árbol corresponde a un sub-conjunto de datos que puede ser aproximado a una superficie plana utilizando un ajuste de mínimos cuadrados para determinar la ecuación del plano, donde la profundidad del *octree* es especificada por el usuario. Luego, dado un *octree* que contiene en cada nodo un plano, reconstruyen la superficie usando un algoritmo de cubos marchantes [**22**]. A partir los polígonos resultantes se calculan los coeficientes de la transformada de wavelet de la función que encierra cada nodo usando el algoritmo de rasterización propuesto por Manson et al. [**31**].

Una vez la nube de puntos está organizada bajo esta estructura, proceden a reducir el tamaño del *octree* removiendo la información redundante en cada nodo. Finalmente Smith et al. codifican la estructura *octree* y las ecuaciones de los planos con un codificador aritmético [**32**]. En la Figura 13 de [**30**] se muestra un resultado de este algoritmo. Sin embargo en este desarrollo se modifican los valores de la nube de puntos original en casi todos sus procesos.

1.3 Contribución

Este trabajo presenta un algoritmo para simplificar las nubes de puntos basado en la reordenación de los datos para descomponer el modelo en niveles de detalles. Por lo general los datos en un archivo asociado a una nube de puntos no tienen un orden que dependa de la geometría del modelo o de la relevancia que tiene cada punto para formar la superficie. En la Figura 1.3, se han cargado progresivamente el 10%, 30%, 50%, 80% y el 100% de los puntos de un modelo 3D respetando el orden original de los datos, este modelo pertenece al repositorio de la Universidad de Stanford. Como se puede ver en la Figura 1.3, los puntos para este modelo están en el orden en el cual este fue digitalizado.

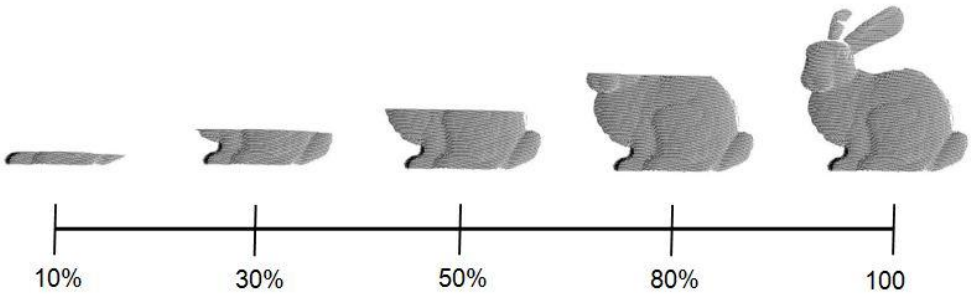


Figura 1.3. Carga progresiva de una nube de puntos sin estructura.

Un algoritmo de muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos, pretende reordenar el conjunto de datos teniendo en cuenta la relevancia de los puntos dentro de la geometría de la superficie basándose en modelos sin una malla definida, lo cual permite simplificar el conjunto de datos sin la preocupación de mantener la conectividad entre los puntos.

Esta técnica consiste en determinar qué información es la más relevante dentro del modelo 3D y de esta forma reordenar los puntos teniendo en cuenta la posición de cada uno de ellos dentro de la superficie. Así, se puede trabajar con la información más importante manteniendo la geometría de la nube original. El volumen de información puede ser reducido considerablemente, incluso en un 90%[1] del modelo original y no perder la esencia morfológica matemática de este.

La carga progresiva de puntos plantea un sistema que permita cargar el conjunto de datos en niveles de detalles definidos en porcentajes. Teniendo en cuenta que las primeras posiciones del archivo contienen la nube de puntos en baja resolución o en baja densidad de puntos, estos son cargados progresivamente y a medida que se cargan puntos se mejora la resolución de la nube hasta llegar al modelo original. Por lo tanto una técnica de carga progresiva de puntos permite que el usuario o un sistema, puedan escoger el nivel de detalle para procesar el conjunto de datos, ya sea para un algoritmo de investigación, para una renderización o animación, para un sistema de percepción 3D de un robot, etc.

Para lograr este objetivo, se deben clasificar los datos según la relevancia de cada uno de ellos dentro del conjunto de datos, la relevancia de cada punto está definida por parámetros asociados a la geometría subyacente alrededor de cada punto que pertenece al modelo, estos parámetros son conocidos como características principales y serán abordados en el capítulo segundo de esta tesis. Uno de estos parámetros se conoce con el nombre de *curvatura*, el cual define la tasa de cambio de los vectores normales a la superficie, por lo tanto las zonas de baja curvatura son regiones suaves, mientras que las zonas de alta curvaturas son regiones de cambios bruscos como los bordes. La idea principal del algoritmo es segmentar la nube de puntos en regiones de baja curvatura, lo que da resultado a regiones relativamente planas, este tema se tratará en el capítulo tercero.

Posteriormente se analiza el tamaño de cada segmento para determinar la importancia de cada uno de estos dentro de la geometría de la superficie y los puntos de cada región

[1] Este valor será justificado en el cuarto capítulo.

son clasificados a partir de este tamaño en conjuntos que discriminan la relevancia de los puntos. Una vez los datos son clasificados en zonas de relevancia, se realiza un muestreo uniforme de cada zona y se ordenan estratégicamente en un nuevo archivo, este análisis y muestreo de la información será abordado en el capítulo cuarto.

En el quinto capítulo se presentan los resultados del trabajo a partir de un sondeo de percepción que se realizó a través de encuestas, también se hace un análisis de los tiempos que toma el algoritmo en procesar diferentes tamaños de nubes de puntos. Finalmente en el sexto capítulo se presentan las conclusiones de la tesis.

2. Estimación de parámetros para determinar el muestreo en nubes de puntos

Un punto p_i, que pertenece a una nube de puntos $\mathcal{P}$ es representado únicamente por sus coordenadas con respecto al origen. Sin embargo, el procesamiento de una nube de puntos requiere algo más que la ubicación coordenada de cada dato. En muchas aplicaciones se necesita comparar cada punto de la nube con el resto de puntos o por lo menos con sus puntos adyacentes. Por lo tanto no se puede considerar a un punto como una entidad singular, por el contrario se representa por sus coordenadas locales acompañado de sus *características geométricas locales* también llamadas *características principales*. En el *Apéndice A* se explica el proceso de adquisición de nubes de puntos y se establecen conceptos matemáticos que serán usados durante este documento.

En esta sección se explicarán tres características principales en nubes de puntos: el concepto de puntos vecinos, estimación de normales a la superficie y la curvatura de cada punto. Estas tres características son de gran importancia dentro de este documento, ya que a partir de ellas, como se verá en el cuarto capítulo, se determina la relevancia que tiene cada punto para la geometría de la nube de puntos. Además, al final de este capítulo, se explica el filtro de puntos espurios, el cual hace parte del algoritmo de *"muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos".*

2.1 Puntos vecinos

Como su nombre lo indica, los puntos vecinos son aquellos puntos p_j^k más cercanos en términos de distancia a un punto específico p_i. En otras palabras, dado un punto que

pertenece a una nube de puntos, $p_i \in \mathcal{P}$, existe un conjunto de puntos $\mathcal{P}^k$ el cual contiene todos los k puntos p_j más cercanos a p_i. Se conoce a $\mathcal{P}^k = \{p_1^k, p_2^k, ..., p_j^k\}$ como la vecindad de un punto p_i, matemáticamente los vecinos p_j^k de un punto p_i se consideran vecinos sí y solo sí cumplen con la Ecuación 2.1, donde $d_{\max}$ especifica la distancia máxima permitida para que un punto p_j^k se considere vecino del punto p_i. También, como se puede ver en la notación de $\mathcal{P}^k$ y p_j^k, el valor de k indica el número de vecinos del punto p_i. Si la distancia $\|p_j^k - p_i\|$ es mayor a $d_{\max}$, entonces se dice que p_j^k y p_i hacen parte de vecindades diferentes [**33**].

$$\|p_j^k - p_i\| \leq d_{\max} \tag{2.1}$$

Sin embargo, este proceso contempla comparar la distancia entre cada uno de los puntos de la nube $\mathcal{P}$ y el punto de interés p_i, luego se ordenan todas las distancias y se escogen las k distancias más pequeñas que cumplan con la condición de la Ecuación 2.1. Esto se conoce como un proceso de fuerza bruta y computacionalmente es muy costoso, por lo tanto pierde sentido usar esta técnica en aplicaciones donde es necesario conocer frecuentemente los k vecinos más cercanos a un punto p_i [**33**].

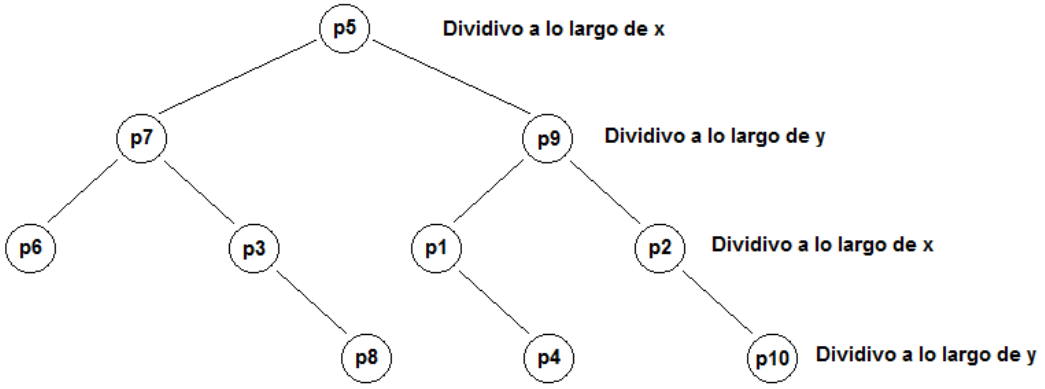


Figura 2.1. Árbol k-dimensional de dos dimensiones.

La técnica usada en esta tesis para encontrar los k vecinos más cercanos a un punto, se basa en arboles k-dimensionales, también conocidos como *k-d trees* [**34**] [**35**] [**36**], el cual es un método que divide el conjunto de datos de un espacio *k-dimensional* de manera

recursiva, esta división genera una representación del conjunto de datos por medio de una estructura en forma de árbol binario (ver Figura 2.1), la cual facilita la búsqueda de puntos dentro del conjunto de datos. Este tipo de técnicas son conocidas como partición binaria del espacio (*BSP*), los *k-d trees* son una rama de estas técnicas, en el *Apéndice B* se explica la construcción de un árbol k-dimensional.

Ahora, para encontrar los k vecinos más cercanos a un punto, primero se debe conocer cómo encontrar el vecino más cercano a un punto p_i. Suponga que se tiene la hipótesis que el vecino más cercano al punto p_4 es el punto p_1, como se muestra en la Figura 2.2.

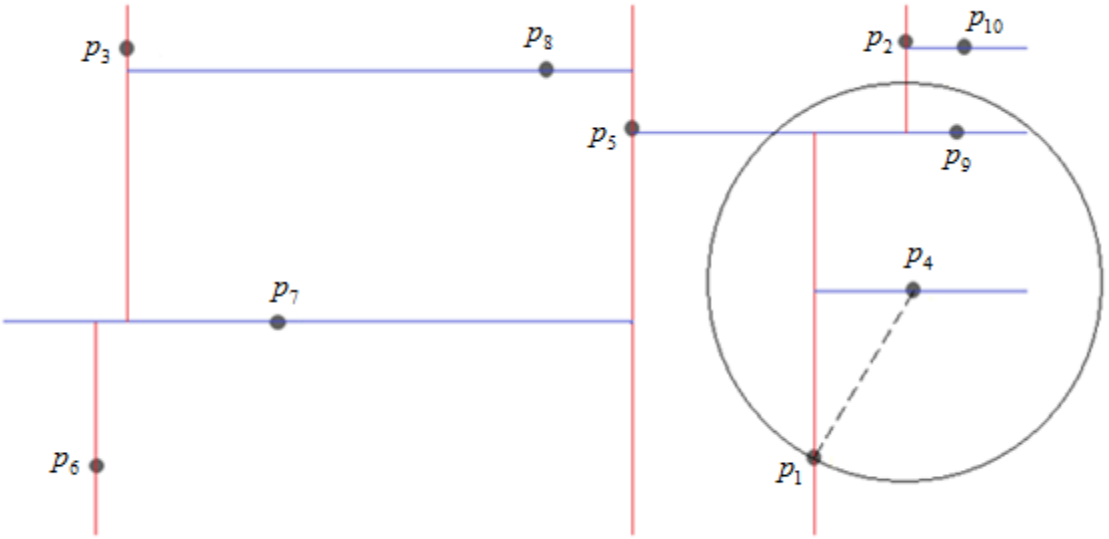


Figura 2.2. División del espacio 2D a partir de un árbol k-dimensional.

Ahora, esta hipótesis posiblemente este equivocada, pero permite hacer una importante observación, si existe algún punto en el conjunto de datos que esté más cerca al punto de interés p_4, este debe existir dentro de la circunferencia marcada por el radio entre el punto p_4 y p_1. Si se tratara de un espacio $\mathbb{R}^3$, la región sería encerrada con una esfera. Conocer el radio de esta circunferencia permite descartar las partes del árbol que no contienen el vecino más cercano. En particular, se puede observar en la Figura 2.2 que la circunferencia no toca la recta del nodo raíz que divide el espacio, consecuentemente se pueden descartar todo los nodos a la izquierda de la raíz.

A partir de la Figura 2.2 es fácil observar que la circunferencia no cruza la *híper-recta* del nodo raíz, sin embargo esto debe ser implementado matemáticamente. Para saber si un circulo y una *híper-recta* se traslapan (para el caso 3D sería una esfera y un *híper-plano*), se debe tener en cuenta que la *híper-recta* solo corta a lo largo de una dimensión (ver *Apéndice B*), por lo tanto sólo basta comparar el valor donde la *híper-recta* corta el espacio con la misma dimensión de la circunferencia, como se muestra en la Figura 2.3. Del mismo modo, solo se tienen en cuenta los nodos que pertenecen a las *híper-rectas* que cortan con la circunferencia de la hipótesis [**37**] [**38**] [**39**].

Como se puede observar en la Figura 2.3, la distancia entre la circunferencia de la derecha y la *híper-recta* $y = y_0$ es equivalente al valor absoluto de la diferencia entre y_0 y la dimensión de la circunferencia correspondiente a la *híper-recta*, en este caso es la dimensión y, note que $|y_1 - y_0| > r_1$, por lo tanto esta circunferencia no corta la *híper-recta*, mientas que la circunferencia de la derecha de la Figura 2.3 si corta la *híper-recta* debido a que $|y_2 - y_0| \geq r_2$.

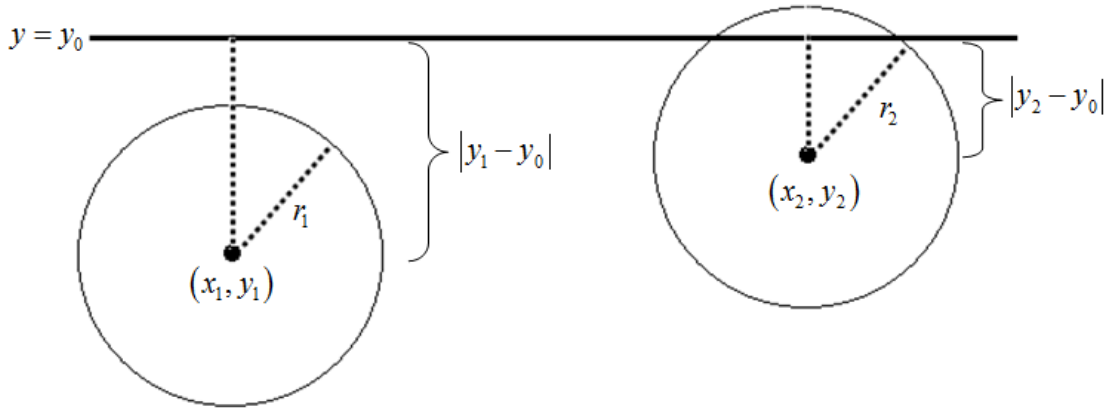


Figura 2.3. A la izquierda de la figura se observa una circunferencia que no es cortada por una *híper-recta*, a la derecha se observa una circunferencia que si es cortada por la *híper-recta*.

La técnica mencionada hasta el momento sirve para encontrar el vecino más cercano a un punto, para encontrar los k vecinos más cercanos a un punto p_i se debe implementar la técnica anterior recursivamente, teniendo en cuenta que siempre se debe usar el radio mayor de las circunferencias existentes entre cada vecino y el punto de interés.

Ahora, se podría decir que, desde un punto de vista práctico, existen dos formas de calcular los vecinos más cercanos a un punto a partir de un árbol k-dimensional. La primera sería haciendo una k-búsqueda (*k-search*) de los puntos más cercanos, la segunda opción es buscar todos los vecinos alrededor de un radio específico. En el desarrollo de esta tesis se utilizó la primera opción, la cual fue implementada usando una librería llamada FLANN (*Fast Library for Approximate Nearest Neighbors*) [**40**], esta contiene una colección de algoritmos escritos en C++ que permiten encontrar los vecinos más cercanos de un punto específico que pertenece a un conjunto de datos.

2.2 Normal y curvatura asociadas a un punto

Los vecinos más cercanos a un punto pueden ser usados para estimar características locales que capturan la geometría de la superficie subyacente alrededor de un punto p_i, una característica importante es la normal asociada a la superficie.

Las normales de una superficie son propiedades importantes que permiten conocer la orientación de esta en un sistema de coordenadas, en el área de visión artificial son ampliamente usadas para determinar la iluminación de la superficie y otros efectos visuales. En este documento son de gran importancia, ya que a partir de las normales de una superficie se pueden calcular las curvaturas asociadas a cada punto, las cuales son un dato clave para determinar la importancia de un punto dentro de la geometría de una nube.

Existen diferentes métodos para encontrar las normales asociadas a una nube de puntos [**41**] [**42**] [**43**], uno de los métodos más simples es el propuesto por Berkmann y Caelli en [**44**], el cual se basa en la estimación de un plano tangente a la superficie. Hay que tener en cuenta que un punto p_i geométricamente no tiene ningún vector normal asociado, los vectores normales están asociados a las superficies y estos son perpendiculares a la misma. Por lo tanto, el problema de determinar la normal a un punto, se aproxima al problema de estimar un plano tangente a la superficie que pase a lo largo de la vecindad de un punto p_i y calcular la normal a este plano.

El primer paso para poder calcular el vector normal asociado a un punto p_i, es entonces estimar la ecuación del plano tangente que pasa a lo largo de la vecindad de dicho punto. Como se menciona en el *Apéndice A sección A.2*, un plano que contiene un punto $q=(x_0,y_0,z_0)$ y tiene un vector normal $\vec{n}=(a,b,c)$, puede representarse de forma canónica como se muestra en la Ecuación 2.2, donde las variables (x,y,z) representan un punto p genérico, el cual pertenece al plano, por lo tanto el producto punto entre el vector $\overrightarrow{qp}=(x-x_0,y-y_0,z-z_0)$ y el vector normal $\vec{n}=(a,b,c)$ tiene que ser igual a cero ya que son perpendiculares entre sí, $\overrightarrow{qp}\cdot\vec{n}=0$.

$$a(x-x_0)+b(y-y_0)+c(z-z_0)=0 \tag{2.2}$$

Este plano se debe estimar a partir de los k vecinos a un punto p_i, lo que quiere decir que se estima el mejor plano utilizando el conjunto de datos de los vecinos $\mathcal{P}^k$. Para realizar esto se tiene en cuenta que la distancia desde cualquier punto p_j^k que pertenezca a $\mathcal{P}^k$ y el plano tangente, debe estar definida como la proyección del vector $\overrightarrow{vp}_j^k$ sobre el vector normal $\vec{n}$, así:

$$d_j=\left(p_j^k-v\right)\cdot\vec{n} \tag{2.3}$$

Donde v es un punto que pertenece al plano tangente y su valor está definido como el centro de masa del conjunto de vecino $\mathcal{P}^k$, se calcula como la media aritmética, así:

$$v=\frac{1}{k}\sum_{j=1}^{k}p_j \mid p_j\in\mathcal{P}^k \tag{2.4}$$

Por lo tanto, para estimar el plano tangente se debe utilizar un algoritmo de mínimos cuadrados de forma que:

$$d_j^2=\sum_{\langle k\rangle}\left[\vec{n}\cdot\left(p_j^k-v\right)\right]^2\rightarrow 0 \tag{2.5}$$

Donde el vector normal $\vec{n}$ es el parámetro que se quiere encontrar y éste tiene que ser diferente de cero, por lo tanto se debe minimizar la expresión $\left(p_j^k-v\right)^2$, de forma tal que

la Ecuación 2.5 tienda a cero. Para minimizar la expresión anterior el algoritmo realiza un análisis de componentes principales (PCA, ver *Apéndice C*), de la siguiente forma:

- Se calcula el centro de masa de todos los $p_j \in \mathcal{P}^k$, a partir de la Ecuación 2.4.
- Se calcula la matriz de covarianza $C \in \mathbb{R}^{3\times3}$ de los datos con respecto al centro de masa, así:

$$C = \frac{1}{k}\sum_{j=1}^{k}\left(p_j - v\right)\cdot\left(p_j - v\right)^T = \frac{1}{k}\sum_{j=1}^{k}C_j \tag{2.6}$$

- A partir de la matriz de covarianza se analizan los vectores propios y los valores propios de esta.
- De este análisis resultan 3 valores propios $\lambda_j \in \mathbb{R}$ y tres vectores propios $\vec{v}_j \in \mathbb{R}^3$, los vectores son ortogonales entre sí, lo que indica que dos vectores propios son paralelos al plano y uno de ellos es perpendicular a él, ver Figura 2.4. El vector propio asociado al valor propio más pequeño es la aproximación del vector normal $\vec{n}$.

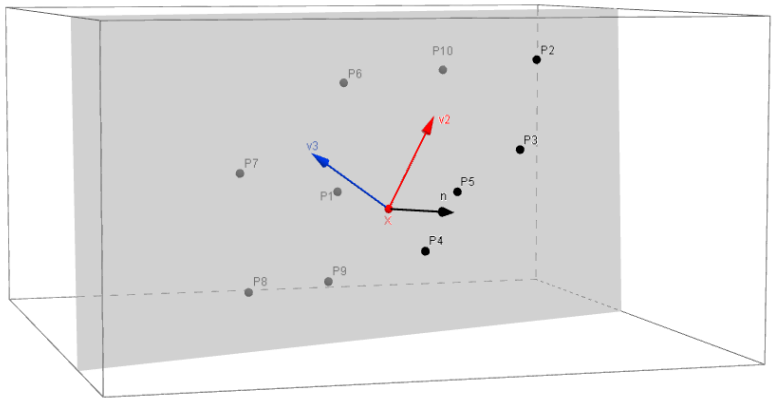


Figura 2.4. Vectores propios asociados a la matriz de covarianza.

La curvatura es una medida de cuán rápidamente se comba una superficie o una curva. Como se muestra en la Figura 2.5, una curva en $\mathbb{R}^2$ que tiene dos puntos p y q se comba más rápido en p que en q debido a que la magnitud de la pendiente de la recta tangente es mayor en p lo cual indica que su curvatura es mayor.

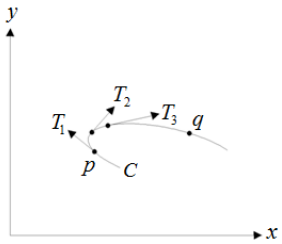


Figura 2.5. La longitud de la tasa de cambio de T con respecto a la longitud de arco es la curvatura.

Por lo tanto, la curvatura se define en $\mathbb{R}^2$, como la tasa de cambio del vector unitario tangente a la curva con respecto al tiempo. Para el caso $\mathbb{R}^3$, la definición es similar, la curvatura denota qué tan rápido cambia el vector normal de un plano tangente a la superficie con respecto al tiempo. Entonces en una nube de puntos, las áreas en las que la superficie es relativamente plana, tienen curvatura baja y las áreas donde la superficie que tienen cambios bruscos en su geometría, tienen alta curvatura. Hay muchas formas para definir la curvatura de una superficie en un determinado punto p_i, pero por lo general estas técnicas necesitan que la superficie tenga definida una malla que conecte los puntos. Sin embargo, en [**45**] y [**46**] se definen dos métodos para encontrar la curvatura de un punto que pertenece a una nube que no tiene una malla definida.

Una solución para encontrar la curvatura asociada a un punto p_i es la propuesta en [**47**], esta técnica hace uso de los valores propios calculados en el análisis de componentes principales anteriormente mencionado, con los cuales se encuentra el vector normal a la superficie.

Para encontrar la curvatura, se escoge el valor propio más pequeño asociado al vector normal $\vec{n}$, cuantitativamente el valor propio $\min\{\lambda_j\}$ describe la variación a lo largo del vector normal, es decir estima qué tanto los puntos vecinos se desvían del plano tangente.

$$\sigma_k(p) = \frac{\min\{\lambda_j\}}{\lambda_1 + \lambda_2 + \lambda_3} \tag{2.7}$$

La Ecuación 2.7 define la variación de una superficie alrededor de p_i en un vecindario $\mathcal{P}^k$, por lo tanto el resultado de esta operación lleva a la aproximación de la tasa de cambio de la superficie alrededor de p_i. Si $\sigma_k(p) = 0$, esto indica que todos los k

vecinos de p_i yacen sobre el plano tangente a la superficie. La máxima desviación es $\sigma_k(p) = 1/3$, esto indica que los puntos están distribuidos de manera isotrópica en el conjunto de vecinos.

A la izquierda de la Figura 2.6, se observa una superficie irregular, la cual tiene áreas relativamente planas y áreas donde el cambio en la superficie es notable; a la derecha de la misma figura se observa la misma nube de puntos, solo que esta vez sus curvaturas están calculadas y representadas con un color, donde el color que se aproxima más al espectro rojo significa alta curvatura, mientras que el color que se aproxima más al espectro azul quiere decir baja curvatura.

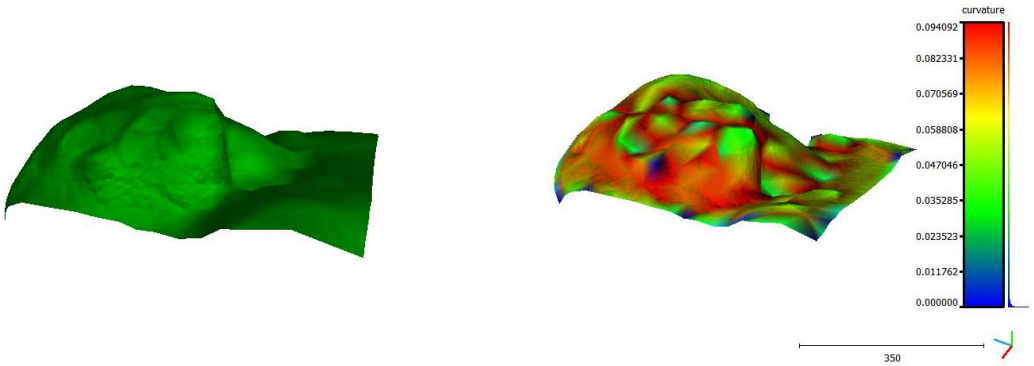


Figura 2.6. A la izquierda se observa una nube de puntos de una superficie irregular, a la derecha se observa la misma nube de puntos coloreada según su curvatura.

Hay que tener en cuenta que la curvatura y la normal a un punto en la superficie son parámetros que no depende exclusivamente del punto p_i, estos parámetros dependen del conjunto de sus k vecinos más cercanos, lo cual indica entonces que la cantidad de vecinos que se escoja influye en la estimación de estas características, por eso es importante que los vecinos se calculen teniendo en cuenta el valor de k y no el valor del radio de la *híper-esfera*, de este modo estas características se calculan de manera uniforme alrededor de toda la nube de puntos.

2.3 Filtro de puntos espurios

Un scanner de rango genera nubes de puntos a partir de la proyección de un láser sobre una superficie, estos pueden generar nubes de diferentes densidades. Sin embargo, el

láser no es del todo exacto y puede tener lecturas de ruido o medidas erróneas que generan puntos espurios en la nube de puntos (*outliers*). Estos puntos, por lo general, son puntos que están situados en coordenadas por fuera de las desviaciones estándar de la mayoría de los puntos, y por lo tanto afectan las medidas de características locales como la normal y la curvatura. Por lo tanto antes de estimar las características de un punto con respecto a sus vecinos, es importante analizar si estos vecinos son una buena representación de la superficie subyacente.

El ruido en las nubes de puntos puede aparecer debido a que el material de la superficie escaneada refleja el láser del scanner, también puede ocurrir cuando el láser se desplaza entre dos superficies debido a la oclusión, la cual también es conocida como salto en los bordes, discontinuidades en profundidad u oclusión en los bordes. En la Figura 2.7 se puede observar el ruido en una nube de puntos, el cual ha sido marcado con círculos.

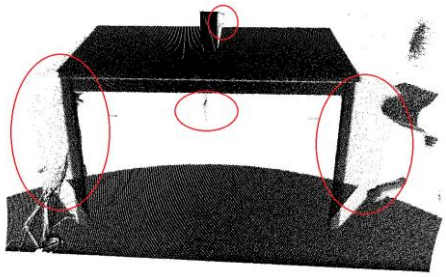


Figura 2.7. Nube de puntos adquirida con un digitalizador 3D, algunos puntos que se consideran ruido se encuentran encerrados en círculos rojos.

Algunas de estas irregularidades se pueden resolver con un análisis estadístico en los vecinos de cada punto y remover el ruido de acuerdo a ciertos criterios. El filtro utilizado en este desarrollo se basa en las distribuciones de distancia desde un punto p_i y sus k vecinos más cercanos. Para cada punto p_i se calcula la distancia media entre él y todos sus vecinos. Esto conlleva a una distribución gaussiana con una media y sus desviaciones estándar, todos los puntos en los cuales la distancia esté por fuera del intervalo de confianza definido por la desviación estándar pueden ser considerados puntos espurios y se eliminan del conjunto de datos [**48**] [**49**], todos los puntos dentro del intervalo de confianza son aquellos que no son considerados como ruido y se conocen con el nombre de *inliers*. En la Figura 2.8 se observan los puntos que han sido removidos, ya que se consideran ruido según el análisis estadístico y en la Figura 2.9 se observa la nube sin ruido (*inliers*).

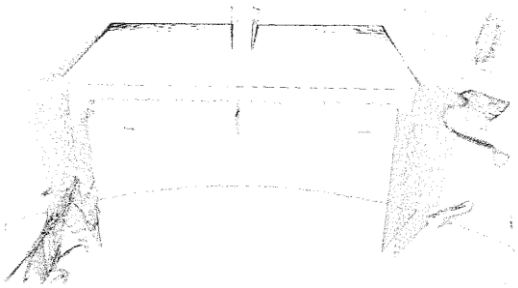


Figura 2.8. Puntos espurios de la nube de puntos de la Figura 2.7.

Este filtrado sólo se debe realizar cuando la nube de puntos no es sintética, ya que este elimina puntos espurios producto de la adquisición de la nube de puntos, si se aplica a una nube sintética se pueden perder datos que no son ruido.

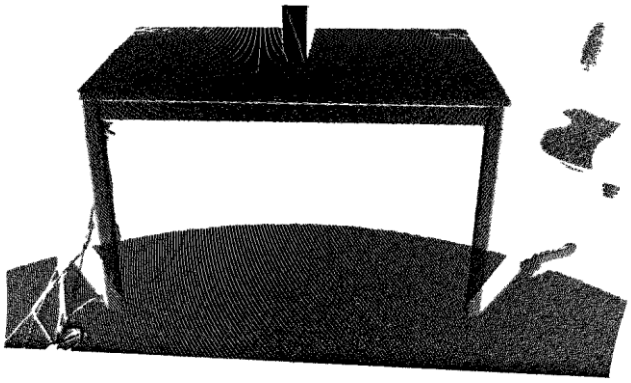


Figura 2.9. Inliers de la nube de puntos de la Figura 2.7.

3. Segmentación del conjunto de datos en regiones de baja curvatura

La segmentación tiene como objetivo principal separar grandes cantidades de información en trozos más pequeños, con el fin de procesar individualmente cada uno de estos. Los subconjuntos resultantes tienen la ventaja de estar compuestos por puntos agrupados que poseen propiedades y características similares, como podría ser la curvatura o el color.

Un algoritmo de segmentación tiene entonces como propósito principal agrupar estructuras similares con el fin de obtener conjuntos más simples de analizar. La segmentación se puede usar para localizar primitivas geométricas (planos, cilindros, esferas, conos, etc.) que se ajusten en secciones del conjunto de datos, o también pueden ser usada para encontrar limites en una nube de puntos [**50**].

Un algoritmo de segmentación que busca estructuras o patrones en una nube de puntos depende básicamente de la complejidad de la estructura a buscar y el nivel de ruido presente en el conjunto de datos. RANSAC (RANdom Sampling and Consensus) es un método iterativo propuesto por Fischer and Bolles [**51**], esta técnica busca ajustar regiones en un conjunto de datos a un modelo geométrico, basándose en generadores de hipótesis heurísticos, el cual realiza un número de iteraciones predefinido para alcanzar un nivel de probabilidad satisfactorio.

La segmentación, como se mencionó anteriormente, también puede buscar límites dentro del conjunto de datos, asignando una etiqueta a cada punto p_i de acuerdo a ciertas características. El algoritmo de segmentación usado en este trabajo se conoce con el nombre de segmentación por crecimiento de regiones [**52**], *region growing segmentation*

algorithm. Este algoritmo asocia, en un conjunto o clúster, puntos que están cerca en términos de suavidad de la tasa cambio del vector normal alrededor de una superficie.

Para diseñar un esquema de muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos, el cual es el objetivo de este trabajo, es necesario encontrar en primera instancia regiones en la nube de puntos que sean lo más planas posibles, lo cual será discutido a fondo a lo largo de este capítulo.

Una vez el conjunto de datos es segmentado en grupos que poseen características similares, se realiza un proceso de análisis estadístico de cada una de las regiones, con el fin de etiquetar cada segmento según sus características geométricas dentro del conjunto de datos para darles un grado de importancia dentro de la superficie y posteriormente reordenar los puntos teniendo en cuenta estas etiquetas, como se verá a lo largo del capítulo cuarto.

3.1 Segmentación por crecimiento de regiones

El algoritmo de crecimiento de regiones es apropiado usarlo en este contexto, ya que permite encontrar límites en el conjunto de datos a partir de la tasa de cambio del ángulo θ que existe entre el vector normal asociado a un punto p_i y el vector normal asociado a cada uno de sus vecinos [**53**] (ver Figura 3.1), por lo tanto si se agrupan los puntos que tengan ángulos relativamente bajos entre ellos, cada región se podría etiquetar como relativamente plana [**54**].

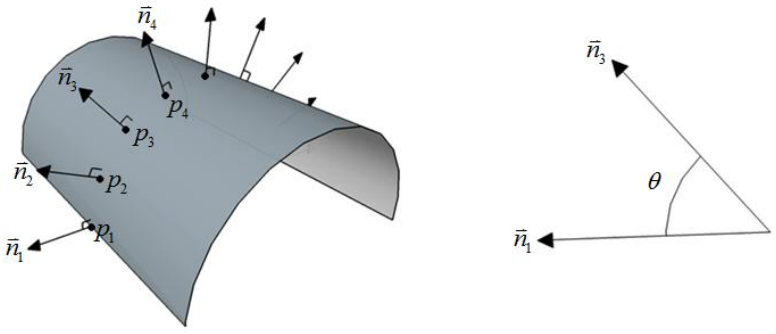


Figura 3.1. Ángulo θ que existe entre las normales de dos puntos.

El algoritmo de crecimiento de regiones trabaja con base en dos características principales, las curvaturas de los puntos y el ángulo entre las normales de un punto p_i y sus vecinos p_j^k [**52**]. Por tal motivo primero se deben encontrar las normales y las curvaturas asociadas a cada uno de los puntos del conjunto de datos.

Una vez determinadas las normales y las curvaturas, el algoritmo comienza a crecer una región a partir de un punto llamado *seed* (semilla) y detiene su crecimiento cuando la región llega a los limites definidos por los parámetros del algoritmo; luego el algoritmo cambia de semilla y crea una nueva región, esto se repite hasta que la totalidad de la nube de puntos ha sido evaluada.

En el algoritmo de crecimiento de regiones, se organizan los puntos en una lista según su curvatura, de menor a mayor. El punto con menor curvatura se le conoce como la primera semilla y este es el punto a partir del cual comienza el crecimiento de la primera región (la curvatura mínima representa el área más plana en la superficie). Se repite el siguiente proceso hasta agotar los puntos existentes en la lista de de curvaturas:

i. El primer punto de la lista de curvaturas es añadido a un conjunto de semillas llamado *seeds*. Para cada semilla se calculan sus k vecinos más cercanos y se procede como sigue:
   a. Se calcula el ángulo entre la normal de la semilla y cada vecino. Si el ángulo es más pequeño que un umbral θ_{th} previamente determinado, entonces el vecino p_j es añadido a la región actual.
   b. Para cada vecino p_j que es añadido a la región, se debe comparar su curvatura con cierto umbral de c_{th} definido previamente, si la curvatura es menor a dicho umbral, entonces este punto es agregado al conjunto de semillas.
   c. La semilla actual es retirada del conjunto *seeds*.

ii. Si el conjunto de semillas queda vacío, significa que el algoritmo ha hecho crecer une región y todos los puntos evaluados son retirados de la lista de curvaturas. El

proceso se repite desde el paso número i con el siguiente punto de la lista de curvaturas.

Según Ushakov [**55**], el pseudo código del algoritmo de crecimiento de regiones es:

Inputs:

Point cloud = $\{\mathcal{P}\}$ — *Neighbour finding function* = $\Omega(.)$

Point normals = $\{N\}$ — *Curvature threshold* = c_{th}

Points curvatures = $\{C\}$ — *Angle threshold* = θ_{th}

Initialize:

Region list: $R \leftarrow \varnothing$ — *Available points list:* $\{A\} \leftarrow \{1,...,|\mathcal{P}|\}$

Algorithm:

While $\{A\}$ *is not empty* ***do:***

Current region: $\{R_c\} \leftarrow \varnothing$

Current seeds: $\{S_c\} \leftarrow \varnothing$

Point with minimum curvature in $\{A\} \rightarrow P_{\min}$

$\{S_c\} \leftarrow \{S_c\} \cup P_{\min}$

$\{R_c\} \leftarrow \{R_c\} \cup P_{\min}$

$\{A\} \leftarrow \{A\} \setminus P_{\min}$

For i=0 to size $(\{S_c\})$ *do:*

Find nearest neighbours of current seed point $\{B_c\} \leftarrow \Omega(S_c\{i\})$

For j=0 to size $(\{B_c\})$ *do:*

Current neighbour point $P_j \leftarrow B_c\{j\}$

If $\{A\}$ *contains* P_j *and* $\cos^{-1}\left(\left|N\{S_c\{i\}\}, N\{S_c\{j\}\}\right|\right) < \theta_{th}$ *then*

$\{R_c\} \leftarrow \{R_c\} \cup P_j$

$\{A\} \leftarrow \{A\} \setminus P_j$

If $C\{P_j\} < c_{th}$ *then*

$\{S_c\} \leftarrow \{S_c\} \cup P_j$

End if

End if

End for

End for

Add current region to global segment list $\{R\} \leftarrow \{R\} \cup \{R_c\}$

End while

Return $\{R\}$

Como se puede observar en el pseudo código, el algoritmo requiere tres parámetros para poder segmentar la nube de puntos en regiones. El primer parámetro es la cantidad k de vecinos, el segundo parámetro es el umbral de ángulo entre las normales θ_{th}, el cual define que tan plana es cada región y por último el umbral de la curvatura c_{th}, éste parámetro define qué tanto crece cada región ya que a partir de éste las semillas de cada una se pueden seguir expandiendo.

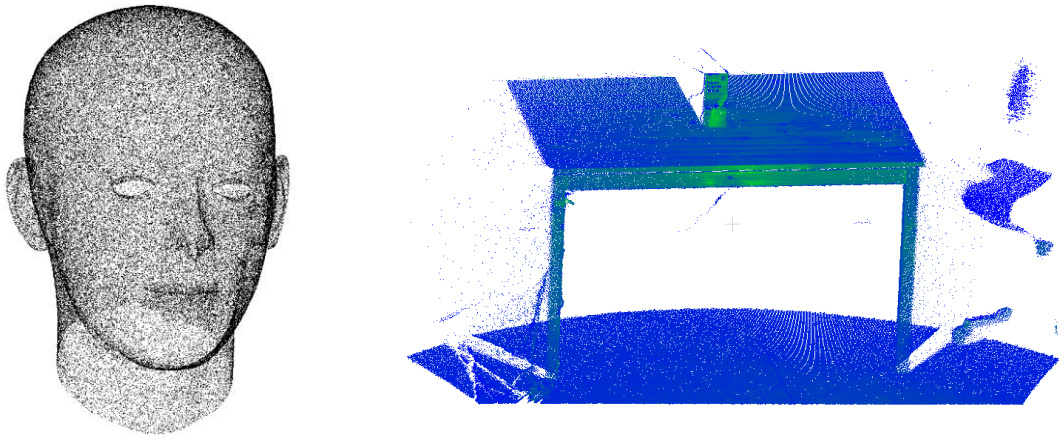
3.2 Análisis para segmentar el conjunto de datos en regiones de baja curvatura

Es importante conocer el motivo por el cual se habla de segmentar la superficie en regiones relativamente planas. Una superficie plana se puede representar con menos puntos que una superficie curva, ya que un plano tiene una tasa de cambio baja en su curvatura, entonces este puede ser representado a partir de sus esquinas, mientras que una superficie que tenga una tasa de cambio alta en su curvatura, necesitará puntos intermedios que permitan su correcta representación. Por lo tanto, una región que sea plana permite omitir puntos, a excepción de sus bordes.

En la tabla 3.1, se presentan varios resultados del algoritmo bajo diferentes parámetros, teniendo en cuenta que el umbral de curvatura c_{th} se tomó en todos los casos como la curvatura media de la nube de puntos.

$$\overline{c} = \frac{1}{N}\sum_{i=1}^{N} c_i \tag{3.1}$$

En el análisis comparativo de la tabla 3.1, se usaron dos nubes de puntos, ver Figura 3.2. Una de estas nubes de puntos es sintética, es decir creada a partir de un software (izquierda de la Figura 3.2) y la otra nube de puntos fue tomada con un digitalizador 3D (derecha de la Figura 3.2).



a) b)

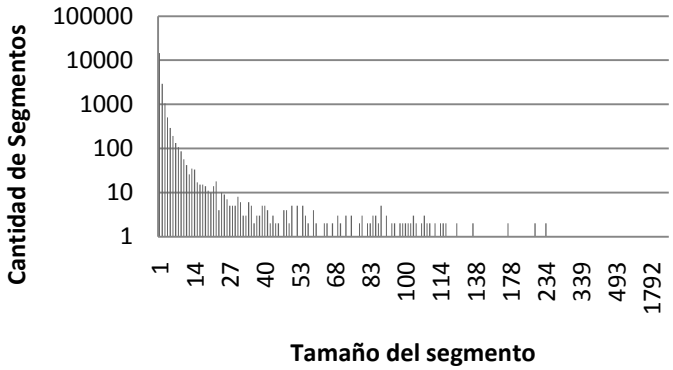
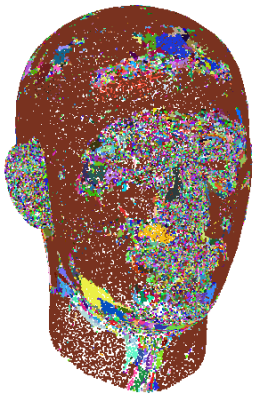
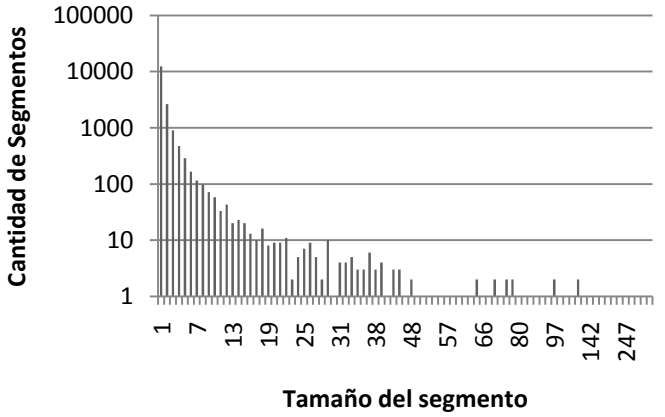
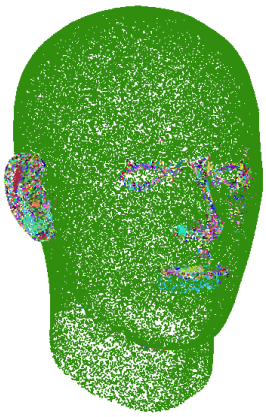
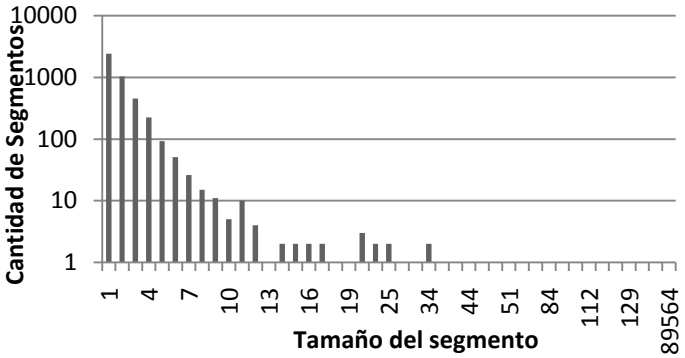
Figura 3.2. a) Nube de puntos sintética de una cabeza humana, contiene 100000 puntos. b) Nube de puntos de una mesa, la cual tiene un envase rectangular sobre ella y una porción de una silla a la derecha, esta nube fue adquirida con un digitalizador 3D, contiene 460400 puntos.

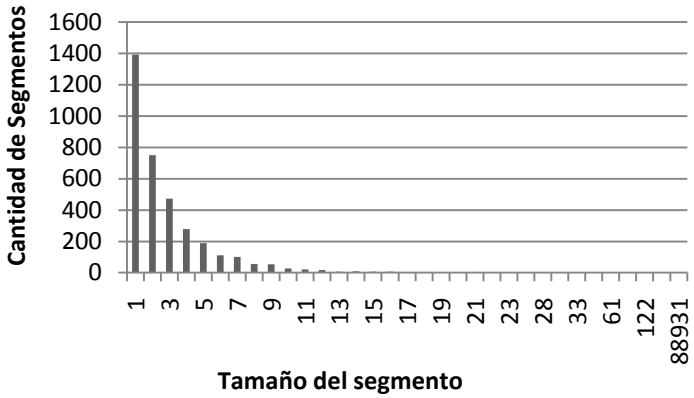
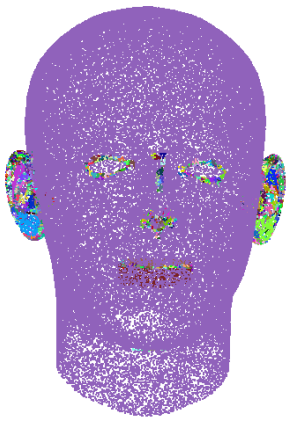
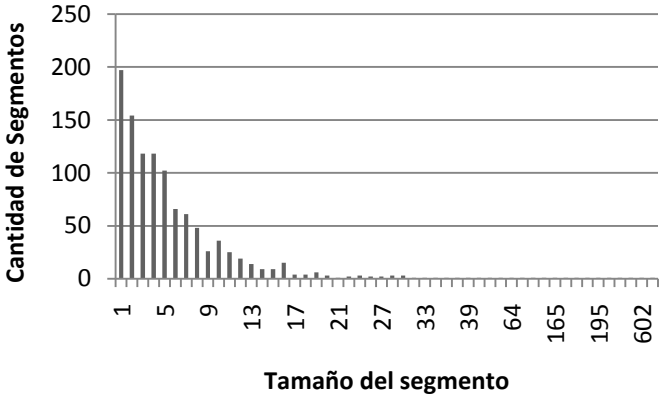
La tabla 3.1 está dividida en tres secciones por cada resultado, la primera sección es la fila superior, en esta fila se encuentra de izquierda a derecha:

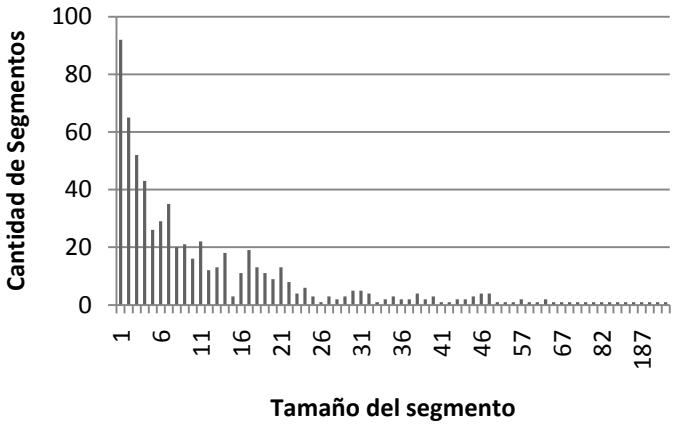
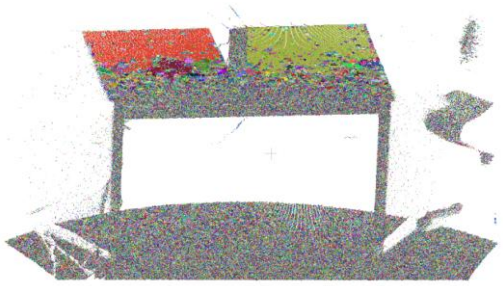
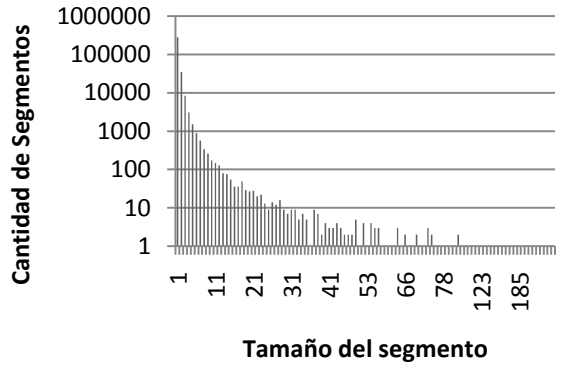
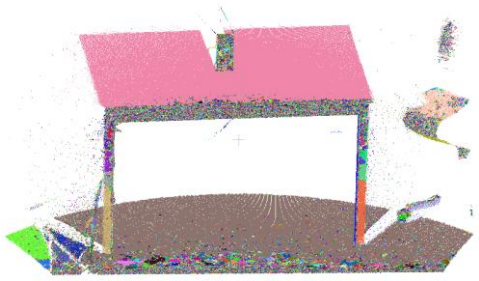
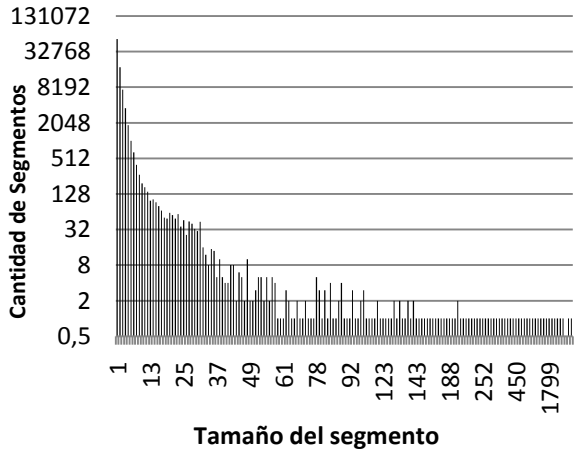
- La cantidad de segmentos en los cuales la nube de puntos ha sido dividida.
- La cantidad de puntos que contiene el segmento más pequeño.
- La cantidad de puntos que contiene el segmento más grande.
- El número k de vecinos, el cual se utiliza para estimar las normales, las curvaturas y para definir la cantidad de vecinos de un punto p_i con los cuales se van a comparar sus normales,
- En las dos últimas columnas de la primera fila se definen el umbral θ_{th} entre los vectores normales y el umbral de curvatura c_{th} el cual se tomó como la media aritmética de todas las curvaturas.

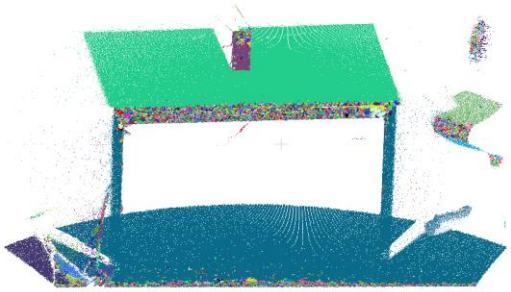
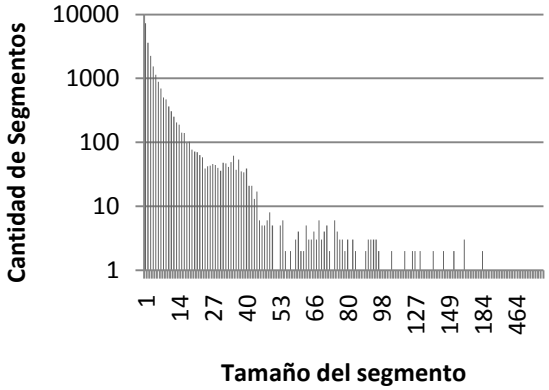
En la segunda fila de la tabla se observa: en la primera columna el resultado gráfico de la segmentación, en el cual se etiqueta cada segmento con un color y en la columna siguiente hay un histograma que indica los tamaños de segmentos vs. La cantidad de segmentos que hay por cada tamaño.

Tabla 3.1. Resultados de la segmentación de las nubes de puntos de la Figura 24.

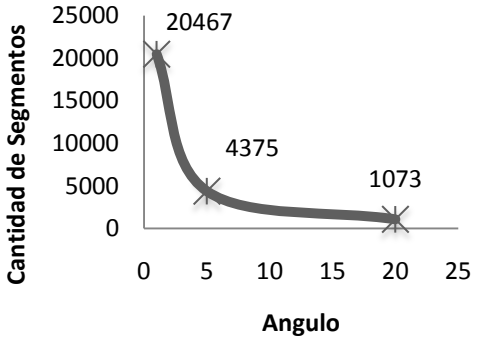
1	*Segmentos: 20467.*	*Segmento más pequeño: 1 punto.*	*Segmento más grande: 8171 puntos.*	*vecinos: 30.*	$\theta_{th} = 1°$	$\bar{c} = 0.0061308$
						
2	*Segmentos: 17471.*	*Segmento más pequeño: 1 punto.*	*Segmento más grande: 60937 puntos.*	*vecinos: 100.*	$\theta_{th} = 1°$	$\bar{c} = 0.0128322$
						
3	*Segmentos: 4375.*	*Segmento más pequeño: 1 punto.*	*Segmento más grande: 89564 puntos.*	*vecinos: 30.*	$\theta_{th} = 5°$	$\bar{c} = 0.0061308$
						

4	*Segmentos: 3531.*	*Segmento más pequeño: 1 punto.*	*Segmento más grande: 88931 puntos.*	*vecinos: 100.*	$\theta_{th} = 5°$	$\bar{c} = 0.0128322$
						
5	*Segmentos: 1073.*	*Segmento más pequeño: 1 punto.*	*Segmento más grande: 91443 puntos.*	*vecinos: 30.*	$\theta_{th} = 20°$	$\bar{c} = 0.0061308$
						
6	*Segmentos: 647.*	*Segmento más pequeño: 1 punto.*	*Segmento más grande: 91066 puntos.*	*vecinos: 100.*	$\theta_{th} = 20°$	$\bar{c} = 0.0128322$

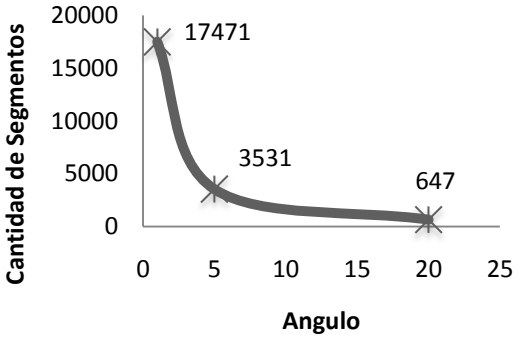
<table>
<tr><td></td><td>  </td><td colspan="4"> 
</td></tr>
<tr><td rowspan="2">7</td><td>Segmentos: 330332.</td><td>Segmento más pequeño: 1 punto.</td><td>Segmento más grande: 14396 puntos.</td><td>vecinos: 30.</td><td>$\theta_{th} = 1°$</td><td>$\bar{c} = 0.0544728$</td></tr>
<tr><td colspan="3">  </td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">8</td><td>Segmentos: 88545.</td><td>Segmento más pequeño: 1 punto.</td><td>Segmento más grande: 135115 puntos.</td><td>vecinos: 30.</td><td>$\theta_{th} = 5°$</td><td>$\bar{c} = 0.0544728$</td></tr>
<tr><td colspan="3">  </td><td colspan="3"> 
</td></tr>
<tr><td>9</td><td>Segmentos: 21522.</td><td>Segmento más pequeño:</td><td>Segmento más grande:</td><td>vecinos: 30.</td><td>$\theta_{th} = 20°$</td><td>$\bar{c} = 0.0544728$</td></tr>
</table>

	1 punto.	*209105 puntos.*			
					

A partir de los resultados de la tabla 3.1, se puede observar que el parámetro que define el ángulo máximo θ_{th} que puede haber entre los vectores normales de dos puntos, tiene un papel importante en la cantidad de regiones resultantes. A la izquierda de la Figura 3.3 se grafica la cantidad de segmentos vs. el umbral θ_{th} para las nubes de puntos sintéticas que se segmentaron con una vecindad de $k=30$ vecinos y a la derecha de la misma figura para $k=100$.



a)



b)

Figura 3.3. a) Gráfico de la cantidad de segmentos vs. el umbral θ_{th} de la nube sintética de la Figura 3.2 con k=30. b) Gráfico de la cantidad de segmentos vs. el umbral θ_{th} de la nube sintética de la Figura 3.2 con k=100.

La tendencia de los datos de la Figura 3.3 supone que la cantidad de segmentos tiene un decaimiento exponencial con respecto al ángulo θ_{th}, en otras palabras, a medida que se

aumenta el umbral del ángulo permitido entre los vectores normales que existe entre un punto p_i y sus vecinos p_j^k, el número de regiones en la segmentación será menor.

Esto tiene sentido ya que si el ángulo es pequeño, la cantidad de puntos etiquetados como vecinos a un punto p_i será menor y por lo tanto la cantidad de segmentos total en la nube de puntos será grande, es decir que esta será segmentada en una gran cantidad de regiones de pocos puntos, como se puede observar en el resultado número uno de la tabla 3.1, donde se observa que la nube de puntos tiene una gran cantidad de regiones debido a que $\theta_{th} = 1°$, lo cual considera que el ángulo entre los vectores normales es tan pequeño que ambos vectores son *"casi paralelos entre sí"* por decirlo de alguna forma y esto indica que las regiones definidas son planas con respecto a sus vecinos.

Por el contrario, si se toma un umbral grande, entonces la nube será segmentada en pocas regiones de muchos puntos, ya que a partir de cada semilla, la región R_i asociada a ella puede crecer con más libertad debido a que el ángulo entre los vectores normales no es tan sesgado. Por ejemplo, la segmentación número 5 de la tabla 3.1, muestra que la nube de puntos tiene una región muy grande alrededor de la forma ovalada de la cabeza y las zonas que tienen cambios bruscos en torno al ángulo entre las normales (e.g. la zona de transición entre la cabeza y una oreja) se etiquetan como regiones diferentes.

Esta segmentación no indica que las regiones sean planas con respecto a sus vecinos, por el contrario, si el ángulo θ_{th} tiende a $360°$, el resultado de la segmentación sería una sola región que contiene a todos los puntos de la nube. Por lo tanto, cuando el ángulo es muy grande, no es una buena opción para el propósito de este trabajo.

En resumen, un ángulo θ_{th} muy pequeño es una buena elección para buscar regiones planas, sin embargo produce muchas regiones en la segmentación si este tiende a valores muy pequeños, lo cual es contraproducente ya que se necesitaría analizar muchas regiones que posiblemente podrían ser agrupadas entre ellas. Por otro lado, un ángulo muy grande segmenta el conjunto de datos en pocas regiones, sin embargo no se puede garantizar que estas sean regiones relativamente planas. Por lo tanto se debe

escoger un ángulo θ_{th} bajo y que segmente el conjunto de datos en la menor cantidad posible de regiones. Una buena elección para el ángulo θ_{th} está entre $4° \leq \theta_{th} \leq 15°$, según las pruebas realizadas con diferentes ángulos y diferentes nubes de puntos.

Un caso satisfactorio de esta última conclusión se evidencia en la segmentación número 8 de la tabla 3.1, se observa que las regiones que son planas han sido agrupadas correctamente, mientras que la segmentación número 7 dividió el segmento plano de la mesa en varias regiones y en la segmentación número 9 el piso de la escena y las patas de la mesa fueron etiquetadas como una misma región.

Debido a que la cantidad de vecinos influye en la estimación de la normal y la curvatura, entonces también afecta directamente la segmentación, escoger este parámetro es una labor que sobrepasa los alcances de esta tesis, la mayoría de los resultados se obtuvieron usando entre 30 y 50 vecinos, ya que entre este rango los resultados fueron más satisfactorios. Este tema en particular ha sido investigado en trabajos como *Scale Selection for Classification of Point-sampled 3-D Surfaces* [**56**] y *Estimating surface normals in noisy point cloud data* [**57**], en los cuales tratan de definir un número k de vecinos de manera automática para estimar los vectores normales de una nube de puntos. Sin embargo estos métodos son lentos y dependen de suposiciones con respecto a una constante relacionada a la densidad del conjunto de datos. Una discusión acerca de esto puede encontrar en la sección 4.6 de [**1**].

El algoritmo de crecimiento de regiones usado en esta tesis es el propuesto por Ushakov en [**55**], el cual entrega en el resultado de la segmentación en un vector de la librería *standard* de C++, el tipo de dato de cada posición del vector es de tipo *pcl::PointIndices,* este tipo de dato está definido en la librería de PCL [**58**]. *pcl::PointIndices* es un vector de datos enteros, donde cada valor entero hace referencia a un punto p_i en la nube de puntos $\mathcal{P}$ ver Figura 3.4.

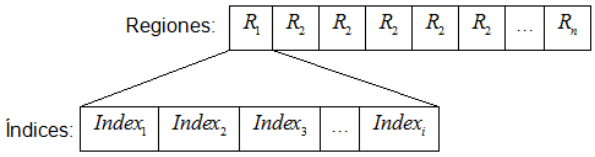


Figura 3.4. Formato del resultado de la segmentación

4. Clasificación, muestro y reordenación del conjunto de datos

El muestreo de puntos consiste en asignar a cada punto del conjunto de datos una etiqueta que permita determinar el peso que tiene un punto p_i dentro de la nube de puntos. En otras palabras, éste tiene como objetivo principal escoger cuáles puntos son más relevantes para la geometría de la superficie y de este modo organizar los puntos del conjunto de datos según su importancia, con el fin de poderlos cargar y almacenar de forma progresiva y tener diferentes porcentajes de resolución. Esto permite que la cantidad de puntos de una nube sea reducida sin comprometer la geometría de la misma.

El primer paso que se debe dar para lograr este objetivo es escoger las categorías en las cuales se clasificarán los puntos para determinar su relevancia dentro del conjunto de datos. Esto se puede realizar a partir de los histogramas de segmentación, los cuales se mencionaron en la sección 3.2, teniendo en cuenta que todas las regiones de la segmentación, independientemente de la cantidad de puntos que cada región tenga, se asumen como regiones planas.

4.1 Clasificación de puntos a partir del análisis de la segmentación del modelo

Suponga que las regiones de la segmentación se clasificarán como regiones grandes o pequeñas. Las regiones grandes en la segmentación representan trozos de la superficie que no tienen cambios bruscos en los ángulos que existen entre las normales de los puntos de la región, además la tasa de cambio de las curvaturas asociadas a cada punto es baja, por tal motivo estas regiones grandes representan regiones suaves, las cuales aportan puntos intermedios a la superficie.

Las regiones que son pequeñas, son regiones que no se expandieron lo suficiente debido a que los ángulos entre sus vecinos superan el umbral θ_{th}, además estos puntos poseen curvaturas muy altas con respecto a la curvatura media c_{th}, lo cual indica que el cambio en la geometría entre las regiones pequeñas y las regiones alrededor de ellas es un cambio significativo con respecto a los parámetros de la segmentación, por lo tanto estas regiones contienen puntos que representan detalles pequeños y bordes en la superficie.

Sin embargo, resulta limitado clasificar las regiones de la segmentación únicamente como regiones pequeñas y grandes. Por tal motivo, en este trabajo se consideran cuatro clasificaciones en las cuales se agruparan las regiones resultantes de la segmentación.

- ***Zona I:*** puntos que pertenecen a regiones con muy pocos elementos, se interpretan como los bordes de la nube, estos elementos poseen alta curvatura.
- ***Zona II:*** puntos que pertenecen a regiones de tamaño medio, estos se interpretan como detalles intermedios en la nube de puntos.
- ***Zona III:*** acá se sitúan los puntos que pertenecen a regiones relativamente planas, es decir, regiones donde la curvatura es baja, estos pertenecen a regiones grandes con respecto a su número de elementos.
- ***Zona IV:*** por último, se ubican los puntos que se filtraron con el filtro de puntos espurios, estos puntos no son relevantes en la geometría de la nube.

Tres de estas clasificaciones resultan al analizar los histogramas de segmentación, en los cuales se puede observar que estos siguen una distribución normal. En la Figura 4.1, se observa un histograma dividido en tres zonas, que denotan las tres primeras clasificaciones de los puntos en el conjunto de datos.

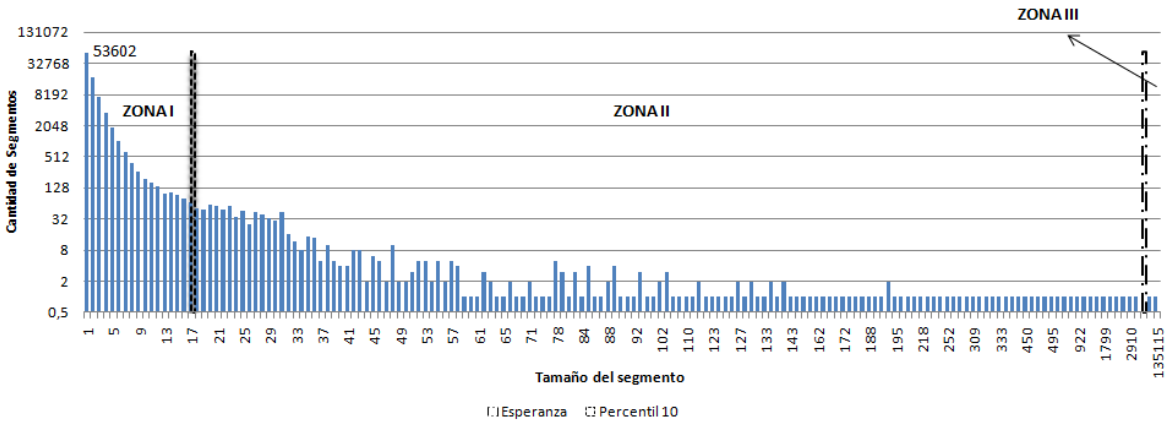


Figura 4.1. Histograma del resultado número ocho de la Tabla 1.

Para determinar estas clasificaciones se realizó un estudio de probabilidad sobre los datos del histograma. Al realizar una segmentación, su resultado arroja una cantidad de regiones y cada región contiene un conjunto de puntos. Por ejemplo, el histograma de la Figura 4.1, el cual corresponde al resultado número 8 de la Tabla 3.1, se observa en este que sus valores extremos indican que existen 53602 regiones que contiene solamente un punto y existe una región que contiene 135115 puntos. Por lo tanto el eje horizontal denota las regiones existentes, mientras que el eje vertical representa su frecuencia de ocurrencia en la segmentación. Dicho esto, se puede plantear la variable aleatoria X como la probabilidad que un punto $p_i \in \mathcal{P}$ esté ubicado en una región R_i de la segmentación.

Ahora, a partir de la variable aleatoria X , se calcula un estadístico de tendencia central conocido con el nombre de valor medio esperado o esperanza matemática, el cual representa la media aritmética de la distribución de los valores de la segmentación. Todas las regiones que estén por encima del valor esperado, se consideran regiones de la *zona III*.

$$E\left[X\right] = \sum_{i=1}^{N} x_i P\left(X = x_i\right) \tag{4.1}$$

La esperanza matemática se calcula a partir de la Ecuación 4.1, donde x_i representa el tamaño de cada región y $P\left(X = x_i\right)$ la probabilidad que un punto p_i pertenezca a una región de tamaño x_i , la cual se calcula a partir de la Ecuación 4.2, donde f_i representa la frecuencia de aparición y N la cantidad de tamaños de regiones.

$$P\left(X = x_i\right) = x_i \frac{f_i}{N} \tag{4.2}$$

Por otra parte, si se conoce la esperanza matemática de los datos, es posible calcular la desviación estándar, la cual es una medida de dispersión que permitiría conocer cuáles regiones están muy alejadas de la media y así poderlas clasificar como grandes o pequeñas con respecto a esta, sin embargo, debido a la naturaleza de los datos, se puede observar que la distribución tiene valores en sus colas que son extremos entre sí, cuando se tiene una distribución de datos con colas tan pesadas como las de la Figura 4.1, se recomienda usar otras medidas de dispersión más robustas ya que la desviación estándar es muy sensible a estas colas.

El estimador de dispersión utilizado en este trabajo se basa en *percentiles*, este método consiste en ordenar los datos de la variable aleatoria X de menor a mayor y se dividen en una escala equidistante de 0 a 100%, por ejemplo, el percentil 50 indica la mediana de los datos y el percentil 30 indica todos los datos que están por debajo del 30% de los tamaños de las regiones. La *zona I* son todas las regiones que están por debajo del décimo percentil, es decir, todas las regiones por debajo del 10% de los tamaños de las regiones, este valor se determinó a partir del análisis de los histogramas de segmentación, donde se encontró que en promedio la mayoría de las curvaturas altas se encuentran por debajo de este valor.

Finalmente, la *zona II* son todas las regiones que se encuentran entre el décimo percentil y la esperanza matemática. La *zona IV*, como se mencionó anteriormente, son todos los puntos que pertenecen al conjunto de *puntos espurios*, los cuales son filtrados al comienzo del algoritmo, esto ocurre solamente si la nube de puntos es adquirida con un digitalizador 3D, en nubes de puntos sintéticas la *zona IV* es un conjunto vacío.

En la Figura 4.2, se observa la nube de puntos utilizada en el resultado número 8 de la Tabla 3.1, los puntos de la nube han sido coloreados según su clasificación. La zona I, la cual busca los puntos de mayor curvatura, corresponde a todos los puntos en color rojo, se observa que todos estos datos hacen parte de zonas de la superficie que deben ser representadas con muchos puntos para no perder la geometría de la misma, por ejemplo en la Figura 4.3.a se observa que la superficie tiene un cambio aproximadamente de 90°

en el borde de la mesa y en la Figura 4.3.b la superficie constituye el asiento de una silla el cual tiene una forma curva, en ambos casos el número de puntos para representar este tipo de cambios en la superficie debe ser alto para que no se pierda resolución en los bordes y quede incompleta la superficie.

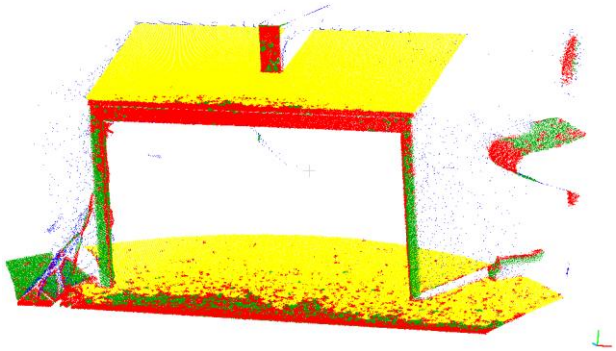


Figura 4.2. Resultado número 8 de la Tabla 1, los puntos de la nube han sido coloreados según su clasificación, zona I en rojo, zona II en verde, zona III en amarillo y zona IV en azul.

Ahora, los puntos coloreados en verde han sido clasificados como puntos de la zona II, estos puntos son relevantes en la geometría de la nube de puntos, pero no se necesita un gran número de estos para representar la superficie subyacente a ellos, por lo tanto su importancia dentro del conjunto de datos es menor a los etiquetados en la zona I.

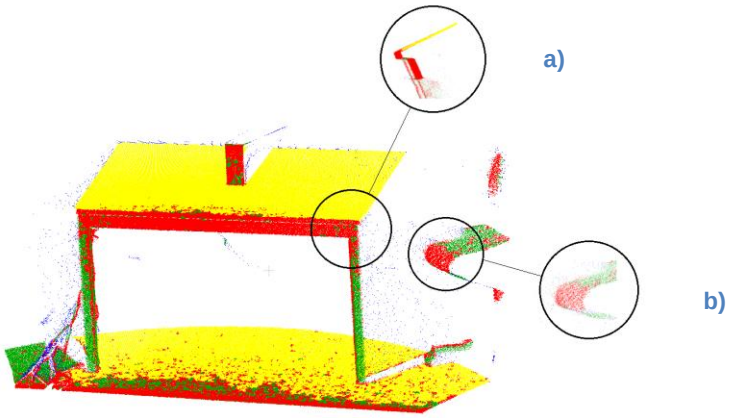


Figura 4.3. a) Borde de la mesa. b) Borde de la silla.

La zona III ha sido coloreada en amarillo, ésta representa las grandes áreas de la superficie que son uniformes con respecto a sus vecinos, por lo tanto la cantidad de puntos necesarios para representar la geometría subyacente son pocos debido a que no son necesarios muchos puntos intermedios ya que la tasa de cambio de la curvatura en estas regiones es baja con respecto a las otras zonas. Por último, en color azul se observan los puntos espurios, lo cuales se consideran como ruido al momento de la

digitalización 3D, estos hacen parte de la zona IV, son los puntos menos relevantes en la clasificación.

4.2 Muestreo y reordenación de nubes puntos

Una vez clasificados los puntos del conjunto de datos, se procede a re-organizar la nube de puntos de forma tal que al cargarla se pueda escoger un porcentaje de detalle. Para lograr esto se deben mezclar las zonas en una sola lista de puntos, teniendo en cuenta que los puntos más relevantes estarán al comienzo de la lista y los menos relevantes al final de la misma, esta lista representa una nueva nube de puntos, solo que esta vez sus puntos están organizados teniendo en cuenta la relevancia de cada uno de ellos y de esta forma al cargar un determinado porcentaje de datos de la lista se obtienen diferentes resoluciones.

Como se acabó de mencionar, es necesario agrupar todas las zonas en una sola lista, la pregunta es, ¿Cómo agrupar estas zonas de manera adecuada? Este proceso no es simplemente agrupar las zonas en un sólo vector, en este proceso se debe tener en cuenta que el objetivo de este trabajo es realizar una carga progresiva de nubes de puntos, por lo tanto para un porcentaje de detalle dado, se debe garantizar una carga uniforme de toda la superficie, en otras palabras, suponga el caso en el cual todos los puntos de la zona I queden al comienzo de la lista y se decide cargar la nube de puntos en baja resolución, podría ocurrir entonces que solamente sean cargados los puntos que corresponden a los bordes y esto no representa la nube de puntos en su totalidad debido a que los detalles intermedios que proporcionan las otras zonas también son importantes para representar la nube de puntos de manera uniforme.

Para crear la lista se toma un punto de cada zona de manera aleatoria, de forma tal que el primer punto aleatorio se toma de la zona I, el segundo de la zona II, el tercero es nuevamente un punto de la zona I y el cuarto punto se toma aleatoriamente de la zona III. Cuando se agotan los puntos de la zona I, se procede a terminar de llenar la lista usando un punto aleatorio de la zona II y un punto aleatorio de la zona III hasta agotar los puntos de la zona II, una vez esto ocurre se termina de completar la lista con los puntos restantes de la zona III. Finalmente, en la cola de la lista se ponen los puntos espurios.

Sin embargo, existe un problema en este planteamiento, éste sólo puede ser correcto si la zona I tiene menos puntos que la zona II, consecuentemente la zona II debe tener menos puntos que la zona III. Suponga que la zona II tiene un número mayor de puntos que la zona I, cuando se agoten los puntos de la zona I, ya se habrán agotado mucho antes los de la zona II, esto generaría un error en el algoritmo. Con los puntos correspondientes a la zona IV no existe este problema ya que estos en cualquier caso sólo se cargan hasta el final de la lista.

Para resolver este inconveniente se debe evaluar el tamaño de cada zona, de forma tal que los puntos aleatorios que se escogen de cada una de ellas, se tomen teniendo en cuenta un orden según el tamaño de la zona y de esta forma agotar cada una de manera uniforme. El proceso para crear la lista se resume así:

- La zona I se etiqueta con el nombre de *edgesZone.*
- Se busca cuál es la zona más grande entre la zona II y la zona III. La más grande entre ambas se etiqueta con el nombre *bigZone* y la más pequeña entre las dos se etiqueta con el nombre *smallZone*. Se repite el siguiente proceso hasta que se acaben los puntos de la zona etiquetada como *edgesZone:*

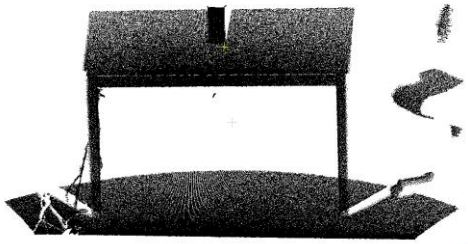
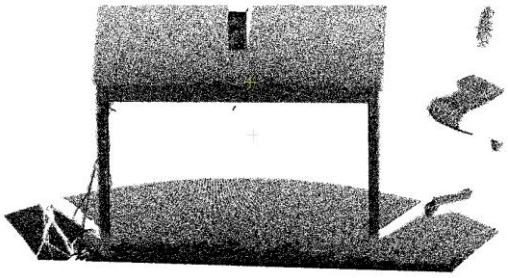
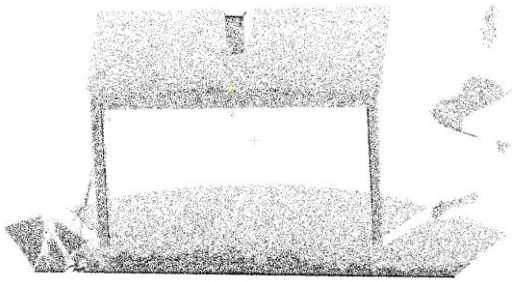
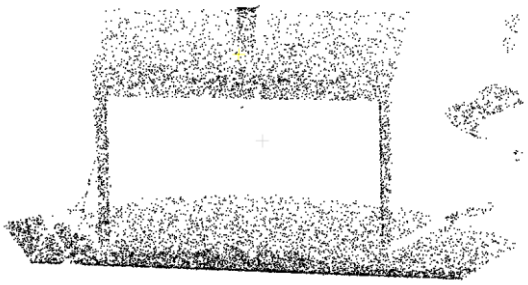
    i. Se toma un punto aleatorio de *edgesZone* y es agregado a la cola del archivo.
    ii. Si la iteración es par y aún existen puntos en *smallZone,* se toma un punto de este conjunto y se agrega a la cola del archivo.
    iii. Si la iteración es impar y aún existen puntos en *bigZone,* se toma un punto de este conjunto y se agrega a la cola del archivo.

- Una vez se acaben los puntos de *edgesZone,* se repite el siguiente proceso si aún existen puntos en *smallZone* hasta que se agoten los mismos:

    i. Se toma un punto aleatorio de *smallZone* y es agregado a la cola del archivo.
    ii. Si aún existen puntos en *bigZone,* se toma un punto de este conjunto y se agrega a la cola del archivo.

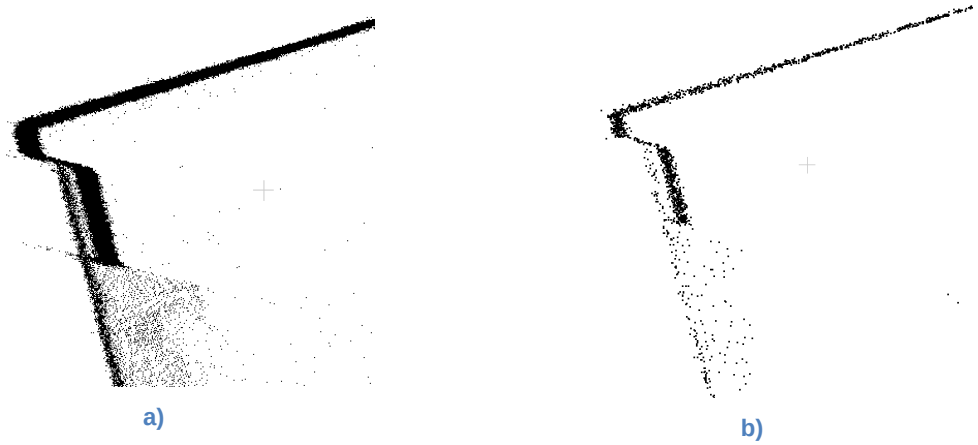
- Una vez se acaben los puntos de *smallZone,* se toman los puntos restantes de *bigZone* hasta que se agoten los mismos.
- Finalmente se agregan todos los puntos espurios al final del archivo.

Tabla 4.1. Carga progresiva de nubes de puntos para 80%, 50%, 20%, 10% y 2% de los datos.

80%		
50%		
10%		
2%		

Al desarrollar un esquema de muestreo de datos que permita la carga y almacenamiento progresivo de nubes de puntos, se logra una lista que tiene los mismos puntos que la

nube original, pero sus datos están organizados de manera que se puedan cargar progresivamente. En la tabla 4.1 se presentan algunos resultados del algoritmo de muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos, las nubes usadas para estos resultados son las de la Figura 3.2 y los parámetros usados para segmentar las nubes fueron $\theta_{th} = 5°$ y el número de vecinos es $k = 30$, estos son las únicas entradas del algoritmo a parte de la nube de puntos. En la tabla 4.1 se observan 4 resoluciones diferentes para cada nube (80%, 50%, 20%, 10% y 2%).



a) b)

Figura 4.4. a) Borde de la mesa de la nube de puntos original. b) Borde de la mesa con el 2% de los puntos cargados del modelo original después de haber aplicado el algoritmo de carga y almacenamiento progresivo de nubes de puntos.

Se evidencia que a medida que la resolución va disminuyendo la cantidad de puntos es menor, sin embargo la superficie se mantiene uniforme. El cambio en la nube de puntos que solo contiene el 50% de los datos iniciales sigue manteniendo la misma forma que la nube original. Ahora, las nubes de puntos que tienen una resolución de 2% siguen manteniendo la geometría de la nube inicial, y lo que es más importante aún, lo bordes que marcan cambios en la geometría de la superficie se mantienen correctamente, en la Figura 4.4 se ve el borde de la mesa de perfil, al lado izquierdo el de la nube original y el al lado derecho es el borde de la nube con una resolución del 2%.

5. Resultados.

Para determinar la valoración perceptual del esquema del muestreo de datos para la carga y almacenamiento progresivo de nubes de puntos, se desarrolló una encuesta que busca comparar la percepción visual del esquema de muestreo propuesto en este documento con otros dos algoritmos, el primero es el propuesto por Gobbetti y Marton, en su artículo *Layered Point Clouds* [**21**] y el segundo es una técnica de simplificación clásica la cual consiste en realizar un *muestreo aleatorio uniformemente distribuido* de datos sobre toda la superficie. Este estudio pretendía determinar si el algoritmo mantiene los detalles y los bordes de las superficies teniendo en cuenta la geometría del modelo.

De aquí en adelante se denominará a la técnica de muestreo aleatorio uniformemente distribuido como *técnica I*, al algoritmo propuesto por Gobbetti y Marton en [**21**] como *técnica II* y a la técnica propuesta en este documento como *técnica III*.

La percepción se considera como la capacidad de obtener información por parte de un sujeto sobre su ambiente, a partir de los estímulos que llegan a sus sentidos sensoriales. En el caso de la percepción visual se refiere a la capacidad de obtener información de una imagen de su entorno físico, a partir de la luz emitida por cada uno de los objetos [**59**].

En el proceso de valoración perceptual del algoritmo propuesto en este documento y en el desarrollo de la encuesta, se tuvieron en cuenta las tres fases de la percepción, las cuales son la visión temprana, la organización perceptiva y el reconocimiento.

En la primera fase, que corresponde a la visión temprana, el sistema visual crea una representación básica de los elementos que componen la imagen tales como el movimiento, la posición y la disposición espacial de los objetos, entre otras. En la

segunda fase, que es la organización perceptiva, el sistema visual confirma la información obtenida en la primera fase, además de determinar el modo como se organizan los elementos de la imagen como una totalidad para poder relacionar los distintos objetos entre sí y las superficies que conforman la misma [**60**] [**61**] [**62**].

La última fase permite obtener información de la imagen sobre aspectos como la identidad, el significado y la función de los elementos de la imagen. En general, se considera que el reconocimiento perceptivo está basado en establecer algún tipo de correspondencia entre la información visual obtenida en cada momento con conocimiento almacenado a largo plazo sobre el aspecto visual de las cosas. Normalmente, el resultado final de todo este conjunto de procesos es la percatación consciente de las distintas características y aspectos de los diversos objetos y entidades que nos rodean [**59**].

La encuesta se diseñó teniendo en cuenta estos tres aspectos fundamentales de la percepción visual, un ejemplo de una de las encuesta se muestra en el *Apéndice D*, se usaron diferentes modelos para no sesgar el estudio a un único modelo. Se plantearon tres preguntas las cuales tenían la siguiente dinámica: En la primera pregunta al encuestado se le presentaron tres nubes de puntos del mismo modelo, las cuales se mostraron usando el visualizador de PCL (*pcl_pcdViewer*) [**58**], dos de estos modelos fueron simplificados utilizando los algoritmos de la técnica I y la técnica II y el tercer modelo fue simplificado con la técnica III, teniendo en cuenta que las tres nubes de puntos tenían la misma cantidad de datos.

A partir de estas tres nubes de puntos, el encuestado debía escoger la técnica con la cual le parecía que el modelo se veía mejor. Con esto se pretendía determinar, a través de aspectos como la visión temprana y la organización perceptiva, la técnica que le permitiera reconocer de forma más clara y detallada el modelo presentado, en otras palabras, lo que se quiere medir es la capacidad que tiene el algoritmo propuesto en este documento, frente a los otros dos algoritmos evaluados, de mantener una buena resolución en los bordes y detalles complejos de la geometría del modelo.

En la segunda pregunta el encuestado debía dar una lista de aspectos y características que lo llevaron a escoger una de las tres opciones de la primera pregunta, esto pretende evaluar cuáles aspectos se resaltan más del modelo seleccionado en la pregunta anterior, identificando así los detalles de la escena y de este modo determinar el reconocimiento de la misma por parte del encuestado, lo cual corresponde a la tercera fase de la percepción visual. De esta forma se busca comparar de manera perceptual la capacidad que el algoritmo tiene para mantener los detalles con respecto a los otros dos algoritmos.

En la tercera pregunta de la encuesta se presentaron tres nubes de puntos de diferentes modelos, cada una se presentó con resoluciones de 0.5%, 1%, 2%, 10% y 20%. Al encuestado se le presentó cada nube de puntos progresivamente desde la resolución más baja hasta la resolución en la cual él identificó completamente el modelo, esta resolución fue anotada como el resultado de esta pregunta. Se buscó nuevamente a partir de aspectos como la visión temprana y la organización perceptiva, determinar la calidad del algoritmo en resoluciones bajas con respecto a la resolución usada comparándolo con los algoritmos de la técnica I y II.

Se encuestaron 100 personas de la Universidad Católica de Manizales y de la Universidad de Caldas, todos los encuestados fueron estudiantes y profesores de carreras afines a la ingeniería, arquitectura, diseño visual y publicidad. A la primera pregunta, en la cual los encuestados debían decidir con cuál de las tres técnicas le parecía que el modelo se veía mejor, los encuestados respondieron como se muestra en la Tabla 5.1, se puede observar que la moda corresponde a la técnica III, el 9% de los encuestados prefirieron la técnica I con respecto a las otras dos, mientras que el 12% de los encuestados prefirieron la técnica II y el 79% de los encuestados coincidieron con la técnica III, la cual corresponde al algoritmo propuesto en este documento.

Tabla 5.1. Resultados primera pregunta de la encuesta de percepción.

Técnica	Encuestados
Técnica I	9
Técnica II	12
Técnica III	79

Ahora, para la segunda pregunta la cual busca establecer las características que llevaron a los encuestados a escoger una técnica sobre las otras, se pudo tabular usando las características mencionadas por cada encuestado. El grupo de personas que escogieron la técnica I y la técnica II piensan que estas técnicas son mejores ya que el modelo se ve más poblado de puntos alrededor de toda la superficie. Por otra parte, para los que escogieron la técnica III fueron más específicos en sus respuestas. De los 79 de 100 encuestados que escogieron esta técnica, el 81% (64 encuestados) mencionaron que los bordes de esta técnica se ven mejor que en las otras dos, el 72% (57 encuestados) mencionaron que los detalles como la cabeza y extremidades de los modelos se veían mejor definidas y a los que se les mostraron nubes de puntos de escenas de exteriores o interiores, mencionaron que seleccionaron esta técnica debido a que los objetos se veían mejor gracias a la definición que tenían en los contornos.

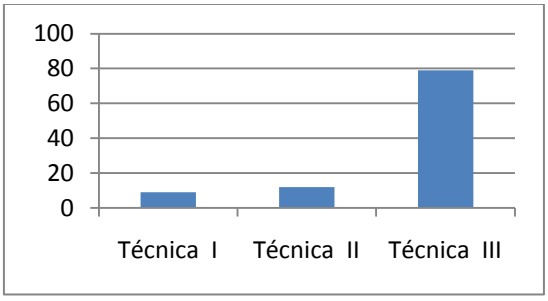


Figura 5.1. Grafico de resultado primera pregunta de la encuesta de percepción.

Para la tercera pregunta, en la cual los encuestados debían seleccionar el nivel de resolución en el cual reconocían claramente el modelo que representaba la nube de puntos, se obtuvieron los resultados de la Tabla 5.2. para cada técnica, el encuestado observaba una nube de puntos y se iba incrementando la resolución de la misma hasta que este reconociera el modelo.

Tabla 5.2. Resultados tercera pregunta de la encuesta de percepción

Resolución	Técnica I	Técnica II	Técnica III
0,5%	9	11	48
1%	28	37	41
2%	52	43	9
10%	11	9	2
20%	0	0	0
TOTAL	100	100	100

Para la técnica I, el 52% de los encuestados coinciden en que reconocieron el modelo con el 2% de los puntos de la totalidad del modelo y el 28% dice que con una resolución de 1% de los puntos es capaz de reconocer el modelo. Para la técnica II, se puede observar que el 43% de los encuestados dicen que es necesaria una resolución del 2% para identificar el modelo mientras que el 37% solo necesito el 1% de los datos. Ahora para la técnica III propuesta en este documento, hay que resaltar, que el 48% de los encuestados fueron capaces de reconocer el modelo con tan solo el 0.5% de los datos y el 41% reconocieron el modelo con el 1%, lo cual indica que esta técnica mantiene la forma de la nube de puntos más clara que las otras dos técnicas.

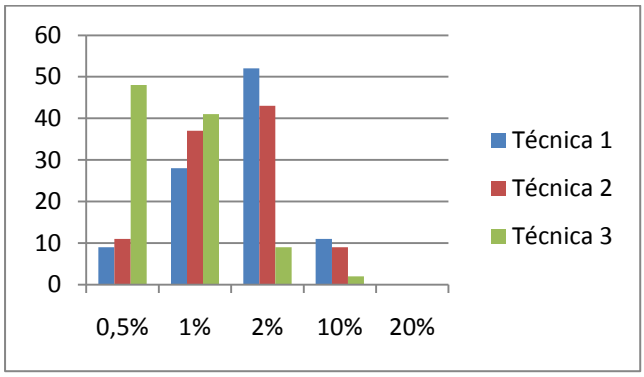


Figura 5.2. Gráfico de resultado tercera pregunta de la encuesta de percepción.

Por otra parte, se tomaron varias nubes de puntos de diferentes tamaños con el fin de medir el tiempo que tardó el algoritmo en procesar cada conjunto de datos para reordenar los puntos teniendo en cuenta las condiciones ya mencionadas. Las nubes de puntos usadas, el tamaño de cada una y el tiempo de ejecución del algoritmo para procesarlas se observar en la Tabla 5.3. Las características del computador que se utilizó para ejecutar el algoritmo son mencionadas en la Tabla 5.4.

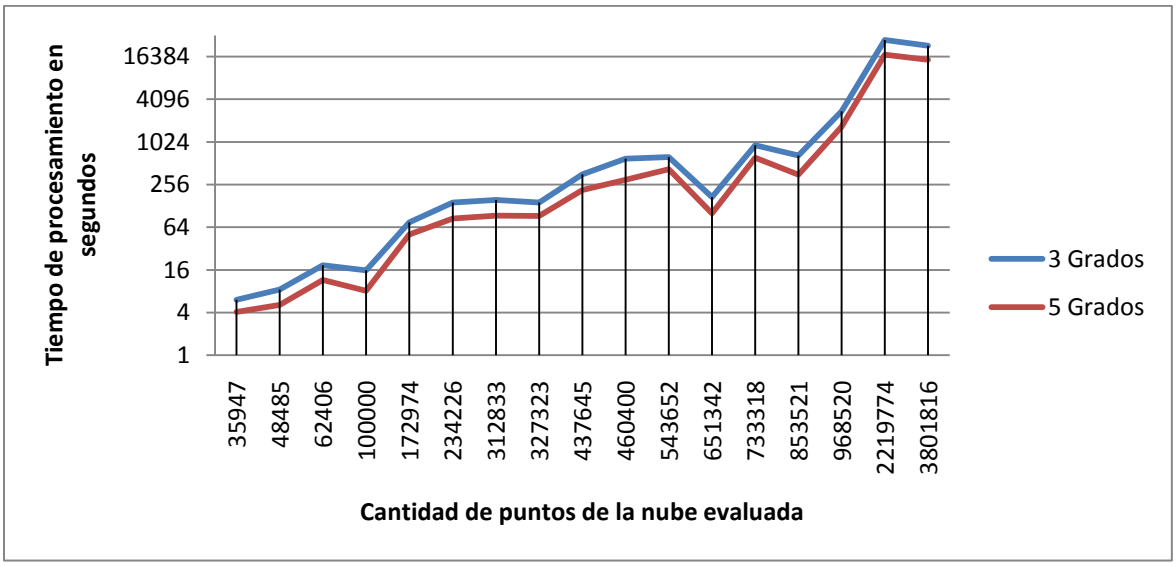


Figura 5.3. Gráfico de la cantidad de puntos de diferentes nubes vs. el tiempo que le tomó al algoritmo procesar cada una con diferentes ángulos θ_{th}.

Como era de esperarse, a medida que el tamaño del conjunto de datos aumenta, también lo hace el tiempo que requiere el algoritmo para procesar la nube, es decir que existe una relación directamente proporcional entre el tamaño de la nube de puntos y el tiempo de procesamiento. Sin embargo se observaron algunos valores en los que el conjunto de datos aumentó y el tiempo de procesamiento del algoritmo disminuyó.

Debido a esto se decidió hacer una segundo prueba, esta vez con dos ángulos θ_{th} diferentes, el primer ángulo fue de $3°$ y el segundo de $5°$, de aquí se dedujo que a medida que el umbral θ_{th} aumenta, el tiempo de procesamiento disminuye (ver Figura 5.3), lo cual indica que el algoritmo de crecimiento de regiones influye en el tiempo.

Además, también se pudo observar que la complejidad en la geometría del modelo influye en el tiempo que tarda el algoritmo en procesar cada nube. Como se puede observar en la Tabla 5.3, la nube *Happy Budda* tiene 543.652 puntos mientras que la nube *Duck* tiene 651.342 puntos, sin embargo el algoritmo tardo un tiempo menor en este último. Incluso, el algoritmo tardo un tiempo relativamente similar en las nubes *Duck* y *Hand*, la cual posee aproximadamente la mitad de los datos que *Duck*. Este último es un modelo con una geometría simple, por lo tanto, el algoritmo lo procesa más rápido en comparación a los otros.

Tabla 5.3. Tiempo de ejecución del algoritmo para procesar diferentes tamaños de nubes de puntos.

Nube de puntos	Tamaño en puntos	Tiempo en segundos $\theta_{th} = 3°$	Tiempo en segundos $\theta_{th} = 5°$
Bunny	35.947	6,04	4,06
Horse	48.485	8,43	5,10
Armchair	62.406	18,56	11,57
Model	100.000	15,81	8,14
Armadillo	172.974	75,22	50,72
Pez	234.226	143,71	84,99
Stairs	312.833	155,26	93,52
Hand	327.323	143,40	92,66
Dragon	437.645	358,49	215,32
Table	460.400	595,67	298,22
Happy Budda	543.652	628,07	422,18
Duck	651.342	169,75	101,38
Laura	733.318	917,26	610,74
Shell	853.521	657,91	354,87
Room	968.520	2.772,28	1.674,86
Cell Tower	2.219.774	28.196,38	17.429,02
Denver Pipes Portion	3.801.816	23.372,25	14.824,91

Tabla 5.4. Características del equipo donde se ejecutó el algoritmo.

Marca	TOSHIBA
Modelo	PORTEGE R835-SP3134L
Procesador	Intel® inside CORE™ i5 – 2410M CPU @ 2.30GHz
Memoria RAM	4 GB
Disco de almacenamiento	SATA 50 GB
Sistema operativo	Ubuntu 14.04 de 64 bits

Finalmente, se evaluó la capacidad que tiene el algoritmo para mantener una densidad de puntos mayor en las regiones de alta curvatura que en las de baja curvatura. Para realizar esto se simplificó una nube de puntos con dos técnicas y se generó una malla sobre cada modelo con el fin de comparar estos resultados, la malla de ambas nubes de puntos simplificadas se reconstruyó usando el algoritmo propuesto por Bernardini et al. en su artículo *The Ball-Pivoting Algorithm for Surface Reconstruction* [**63**], ver Figura 5.4.

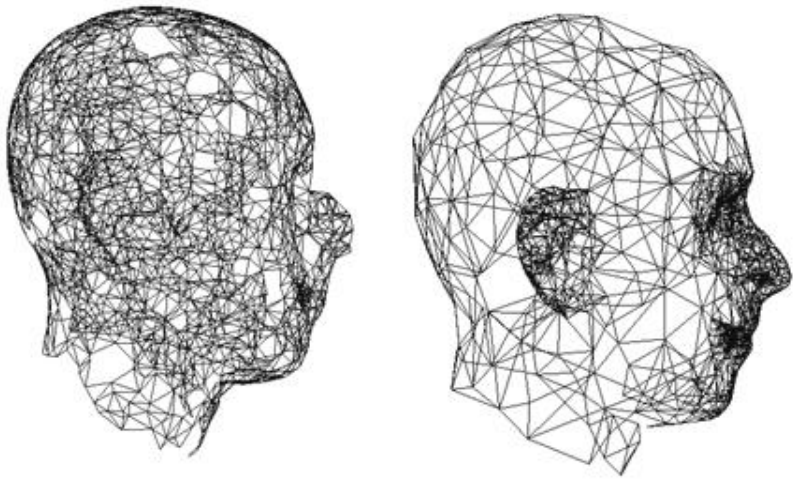


Figura 5.4. Ambos modelos tienen el 2% de sus puntos originales, el modelo de la izquierda fue simplificado con la técnica de muestreo aleatorio uniformemente distribuido y el modelo de la derecha fue simplificado con la técnica propuesta en este documento. La reconstrucción de la malla se realizó con el algoritmo propuesto en [63].

Los modelos presentados en la Figura 5.4 solo poseen los primeros puntos que corresponden al 2% de la totalidad de los datos, el modelo de la izquierda de la Figura 5.4 fue simplificado usando el algoritmo muestreo aleatorio uniformemente distribuido y el modelo de la derecha fue simplificado con la técnica propuesta en este documento. Se puede observar en la Figura 5.4 que este último mantiene mayor densidad de puntos en las zonas de curvatura alta y baja densidad de puntos en zonas donde la superficie es suave.

Es claro que la forma del rostro y las orejas se ven mejor definidos en el modelo de la derecha y que en las zonas donde la superficie es suave los polígonos son más grandes, mientras que en el modelo de la izquierda los polígonos son aproximadamente del mismo tamaño en toda la superficie, lo cual indica que el rostro y las orejas, los cuales requieren más detalles que el resto de la superficie, tienen la misma densidad de puntos que el resto del modelo.

6. Conclusiones y trabajo futuro.

En este documento se presentó un algoritmo de simplificación para nubes de puntos, el cual se basa en el reordenamiento de los datos, los cuales se almacenan en un archivo de formato estándar para modelos 3D y así cargar este archivo reorganizado de forma progresiva. El algoritmo es fácil de implementar y no requiere la existencia de una malla definida que conecte los puntos, es decir, el algoritmo se basa únicamente en la información de la ubicación espacial de cada punto.

Este reorganiza los puntos teniendo en cuenta las características locales que capturan la geometría de la superficie subyacente alrededor de un punto, a partir de las cuales, se calcula el cambio en la curvatura con respecto a los vecinos de cada punto y de esta forma identificar las zonas de la superficie en las cuales es necesario mantener una mayor densidad de puntos con respecto a zonas donde la curvatura tiene cambios suaves, las cuales se pueden interpretar como zonas relativamente planas y que no necesitan una gran densidad de puntos para su representación.

Es así como el algoritmo etiqueta cada punto a partir de la información de la curvatura de la superficie, donde da paso a que los puntos en el conjunto de datos puedan reorganizarse a partir de las etiquetas o pesos que han sido asignados teniendo en cuentas las características ya mencionadas.

Para reordenar el conjunto de datos, el algoritmo le da prioridad a los puntos de zonas de alta curvatura, sin embargo estos son mezclados con puntos que pertenecen a zonas de baja curvatura, con el fin de preservar no solamente los bordes de la nube de puntos, si no también datos de regiones suaves. De este modo se garantiza que los cambios bruscos en la superficie se preserven con una alta densidad de datos y que sean acompañados con puntos que permitan representar las zonas intermedias entre bordes.

Además, el algoritmo entrega una estructura simple de cargar, ya que los datos están organizados estratégicamente de forma que se pueda representar el modelo con la cantidad de puntos que el usuario desee.

Hay que tener en cuenta que el algoritmo es sensible a la estimación de las características principales como la normal y la curvatura asociada a cada punto, además que estas estimaciones son lentas computacionalmente hablando. Por otra parte el algoritmo de crecimiento de regiones también es computacionalmente costoso, por tal motivo el trabajo futuro para esta tesis es mejorar el desempeño del algoritmo optimizando estos procesos o buscar soluciones alternativas a estos.

A. Apéndice A: Nubes de puntos y adquisición

Una nube de puntos $\mathcal{P}$ es una estructura digital en tres dimensiones del mundo real, tal como un objeto, una persona, un animal o escenas de exteriores o interiores. Como su nombre lo indica, esta es un conjunto de puntos coordenados en un espacio tridimensional que forman una superficie. Usualmente cada punto p_i es definido por coordenadas (x, y, z), donde $p_i \in \mathcal{P}$, aunque cada punto podría también tener más información tal que $p_i = (x, y, z, f_1, f_2, f_3, f_n)$, donde f_1, f_2, f_3, f_n son características adicionales de cada punto como el color en RGB, la información de reflectancia o absorbancia de la luz en ese punto, una propiedad física, la distancia desde el sensor que capturo la nube hasta el punto o simplemente una etiqueta con información necesaria según la aplicación para la cual será utilizada la nube de puntos.

En este apéndice, se presentan las bases que describen el proceso de formación de la imagen, lo que lleva al desarrollo de ecuaciones que permiten la recuperación de un modelo tridimensional a través de un proceso de triangulación geométrica. Además se introducen conceptos básicos de geometría analítica (e.g. puntos, vectores, líneas, rayos y planos), los cuales son útiles para comprender el modelo geométrico.

A.1 Adquisición de datos

Las coordenadas de un punto $p_i \in \mathcal{P}$ ubicadas en un plano tridimensional, por lo general, tienen como origen coordenado el dispositivo de sensado utilizado para adquirir la nube. Esto indica que cada punto p_i representa la distancia que hay entre el punto de vista en que se capturo la nube de puntos y la superficie original.

Existen muchas formas de muestrear una superficie y convertirla en una nube de puntos, aquí mencionaremos dos de las técnicas sin contacto con la superficie, más comunes para realizar este proceso [**1**]. En la Sección A.2 se explicará el principio matemático que rige a la técnica utilizada para capturar nubes de puntos en esta tesis.

La primer categoría son los sistema que proyectan ondas de luz (e.g. laser) o sonido (e.g. sonar) sobre la superficie, las cuales será reflejadas de nuevo hacia el origen del sensor. Por ejemplo podemos encontrar los sistemas de tiempo de vuelo o TOF (por sus siglas en ingles *Time Of Flight*) [**64**], los cuales miden el retardo que hay desde que se emite una señal que golpea la superficie a muestrear hasta que esta regresa al transmisor, de esta forma se calcula la distancia desde el sensor hasta la superficie, también existen otros dispositivos como los sistemas laser de medida (LMS) o LIDIAR, radares y sonares. Para trabajar con estos dispositivos es necesario conocer la velocidad de propagación de la onda y se mide el tiempo que tarda la onda en ir hasta la superficie y retornar hasta el sensor, con lo cual se calcula la distancia a partir de:

$$d = \frac{ct}{2} \tag{A.1}$$

Donde c representa la velocidad de propagación de la onda y t representa el tiempo que tardo la señal en ir y volver, como la onda hace el mismo recorrido dos veces, entonces se debe dividir entre dos [**1**].

La segunda categoría son los sistemas que utilizan técnicas de triangulación a partir de dos sensores, este tipo de técnicas estiman la distancia a la superficie por medio de las correspondencias de dos puntos de vista tomados con ambos sensores al mismo tiempo [**65**]. Para calcular las distancias a la superficie, ambos sensores deben ser calibrados uno con respecto al otro, además sus propiedades intrínsecas y extrínsecas deben ser conocidas. Usualmente las distancias se estiman así:

$$d = \frac{fT}{\left\| x_1 - x_2 \right\|} \tag{A.2}$$

Donde f representa la distancia focal de ambos sensores, T es la distancia entre los sensores, x_1 y x_2 son los puntos correspondientes en cada sensor. Un ejemplo de esta técnica es la *visión estéreo* [**1**].

Las nubes de puntos que se utilizaron en este documento fueron tomadas usando técnicas de triangulación [**66**]. El equipo para obtener las nubes hace uso de un tipo de sensor que proyecta luz estructurada sobre el objeto para obtener modelos 3D de manera precisa. Este tipo de sensores obtienen las nubes de puntos de una superficie a partir de la combinación de una cámara que captura imágenes y un espejo rotativo que refleja luz laser con un patrón determinado [**67**]. Específicamente se utilizó un digitalizador 3D Konica Minolta VIVID 9i [**68**], el cual pertenece a la Universidad Nacional de Colombia Sede Manizales, el patrón de la luz laser de este digitalizador 3D es un plano. La ventaja de los equipos de adquisición de nubes de puntos basados en triangulación que hacen uso de luz estructurada sobre los sistemas de visión estéreo que usan dos cámaras, es que se evita el problema de correspondencia de puntos entre los planos de la imagen de ambas cámaras [**1**].

Las imágenes capturadas con el digitalizador 3D Konica Minolta VIVID 9i solo cubren la superficie de la escena o del objeto de forma parcial, es decir, desde un punto de vista especifico. Por lo tanto, se tienen que tomar varias capturas desde diferentes ángulos con respecto a la superficie para poder cubrirla en su totalidad sin tener oclusiones. Como se verá en la Sección A.3, las coordenadas de cada punto toman la posición del sensor como origen del sistema coordenado. Por tal motivo, cuando se realizan capturas desde diferentes puntos de vista (ver Figura A.1), estas deben ser alineadas en un mismo espacio coordenado. Al proceso de obtener capturas desde diferentes ángulos y alinear cada una de estas se le conoce con el nombre de *registro* [**69**].

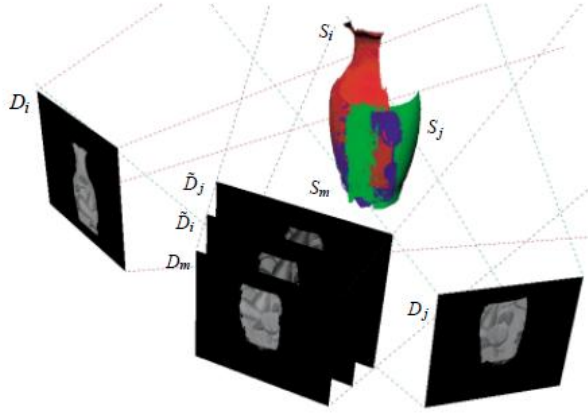


Figura A.1. Registro para la digitalización completa de un modelo, figura tomada de [69].

Una vez se realice el proceso de registro, comienza un proceso llamado *integración*, el cual consiste en generar una sola nube de puntos utilizando la información de las capturas parciales tomadas durante el registro, por lo tanto se deben encontrar correspondencias entre cada captura para poder integrarlas de manera precisa [**70**]. Hay que tener en cuenta que la ubicación cartesiana de cada punto p_i no es suficiente para encontrar estas correspondencias, se necesita información adicional de cada punto, esta información adicional se le conoce con el nombre de *características principales* o *características geométricas*. El proceso de integración también busca eliminar información redundante en los traslapes que se presentan al encontrar los puntos de correspondencia entre capturas y rellenar las oclusiones presentes, con el fin de llegar a un único modelo [**71**].

A.2 Principio matemático de la triangulación

Para estimar las coordenadas de cada punto $p_i \in \mathcal{P}$ de un objeto o superficie es necesario proyectar sobre este una luz con un patrón conocido y captar estos patrones con una cámara. El patrón de luz proyectado sobre la superficie consiste en un laser que es reflejado en una superficie cilíndrica, la cual convierte el rayo de luz en un plano, y este a su vez es reflejado sobre un espejo que puede cambiar de ángulo para cubrir el área de la superficie a escanear. En esta sección se estudiaran los principios básicos para calcular las coordenadas de un punto en la superficie.

A.2.1 Representaciones geométricas

Ya que la luz se propaga en línea recta, en medios homogéneos como el aire, entonces se deducen las ecuaciones de reconstrucciones 3D a partir de construcciones geométricas que implican la intersección de líneas y planos. Estas ecuaciones se basan en algebra y geometría analítica. Las letras en minúscula denotan un punto en el espacio y las letras en minúscula, como $\vec{v}$, denotan un vector en el espacio, todos los vectores serán tratados como vectores columna con coordenadas en valores reales $(x, y, z) \in \mathbb{R}^3$, los cuales también se pueden considerar matrices tres filas y una sola columna $\vec{v} \in \mathbb{R}^{3x1}$.

El tamaño de un vector $\vec{v}$ es un escalar tal que $\|\vec{v}\| = \sqrt{v_x{}^2 + v_y{}^2 + v_z{}^2} \in \mathbb{R}$. Se utilizarán multiplicaciones de matrices para el *producto interno* o *producto punto* de dos vectores: $\vec{v}_1^T \cdot \vec{v}_2 \in \mathbb{R}$ el cual es un escalar; aquí $\vec{v}_1^T$ es un vector fila $\in \mathbb{R}$, el cual es el resultado de calcular la matriz transpuesta de $\vec{v}_1$. El valor del *producto interno* de dos vectores $\vec{v}_1$ y $\vec{v}_2$ es igual a $\|\vec{v}_1\|\|\vec{v}_2\|\cos(\alpha)$, donde α es el ángulo formado por los dos vectores. El *producto vectorial* o *producto cruz* de dos vectores $\vec{v}_1 \times \vec{v}_2 \in \mathbb{R}^3$ es un vector perpendicular a ambos vectores $\vec{v}_1$ y $\vec{v}_2$, el tamaño de este vector esta dado por $\|\vec{v}_1\|\|\vec{v}_2\| sen(\alpha)$, y la dirección está determinada por la ley de la mano derecha; en particular dos vectores $\vec{v}_1$ y $\vec{v}_2$ son linealmente dependientes si y solo si el *producto vectorial* es igual a cero [**72**] [**73**].

A.2.2 Cámara esteneopeica

Una cámara proyecta los rayos de luz del mundo real 3D (espacio del objeto) sobre un plano 2D conocido como el plano de la imagen. Una cámara esteneopeica (ver Figura A.2), también conocida como *pin hole camera* por su nombre en inglés, es una cámara en la cual se considera como centro de proyección o foco, el origen del sistema coordenado Euclidiano y además el plano $z = f$ como *el plano de la imagen* o *plano focal*.

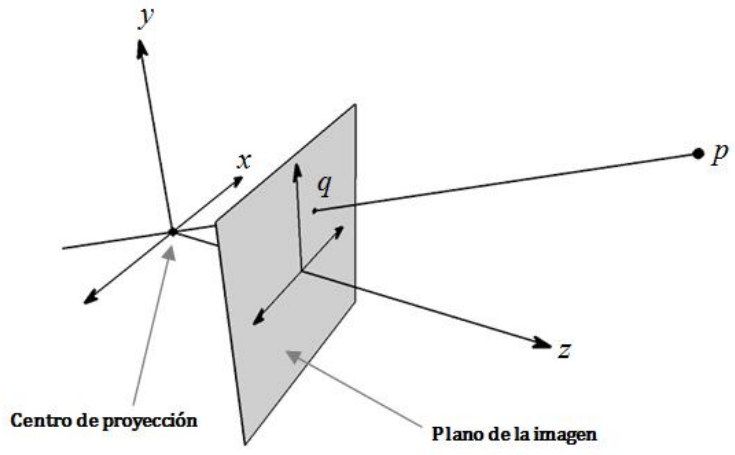


Figura A.2. Proyección de perspectiva bajo el modelo de la cámara esteneopeica.

Bajo este esquema, un punto p en el espacio con coordenadas $(x, y, z)^T$, es proyectado en un punto sobre el plano de la imagen, donde una línea une el punto p con el centro de proyección y el punto q (2D). La proyección del punto 3D p es donde la línea intersecta el plano de la imagen, es decir que cualquier punto 3D (diferente al centro de proyección) determina una línea única que pasa a través del centro de proyección o foco, si esta línea no es paralela al plano de la imagen entonces esta debe intersecar a este en algún punto q (*image point*). En matemática, este mapeo de puntos 2D a partir de puntos 3D se conoce como *proyección de perspectiva*. Por similaridad de triángulos, podríamos decir que el punto $p = (x, y, z)^T$ es proyectado sobre el plano de la imagen como el punto $q = \left(fx/z, fy/z, f \right)^T$ [**72**].

A.2.3 Puntos y vectores

Ya que los vectores forma un espacio vectorial, estos pueden ser multiplicados por escalares y pueden ser sumados unos con otros. Los puntos, por otro lado, no forman espacio vectoriales, sin embargo los puntos y los vectores están relacionados así: un punto más un vector ($p + \vec{v}$) es otro punto y la diferencia entre dos puntos ($p - q$) es un vector. Si p es un punto, λ es un escalar y $\vec{v}$ es un vector, entonces $q = p + \lambda\vec{v}$ es otro punto, en esta expresión $\lambda\vec{v}$ es un vector de tamaño $|\lambda|\|\vec{v}\|$. Multiplicar un punto por un escalar λp, no está definido.

A.2.4 Representación paramétrica de líneas y rayos

Una línea L puede ser descrita especificando uno de sus puntos q y un vector director $\vec{v}$. Cualquier otro punto p sobre la línea L puede ser descrito como el resultado de añadir un múltiplo escalar $\lambda\vec{v}$ del vector director $\vec{v}$ al punto q (λ puede ser negativo, positivo o cero).

$$L = \{ p = q + \lambda\vec{v} : \lambda \in \mathbb{R} \} \tag{A.3}$$

Esta es la representación paramétrica de una línea (ver Figura A.3), donde el escalar λ es el parámetro, ver Ejemplo A.1 y Ejemplo A.2.

Esta representación no es única, ya que q puede ser representado por cualquier otro punto sobre la línea L y $\vec{v}$ puede ser reemplazado por cualquier otro vector con la misma dirección diferente de cero. Sin embargo, para cada q y $\vec{v}$ que se escojan, la correspondencia entre p y λ es única [**72**].

Un rayo es un segmento de una línea, mientras que en una línea el parámetro λ puede tomar cualquier valor, en un rayo λ solo puede tomar valores positivos.

$$R = \{ p = q + \lambda \vec{v} : \lambda \geq 0 \} \tag{A.4}$$

En este caso, si el punto q cambia, resulta en un rayo diferente, ya que el punto q se conoce como el origen del rayo. El vector director $\vec{v}$ puede ser reemplazado por cualquier múltiplo escalar positivo de este. Si se reemplaza el vector director $\vec{v}$ por un múltiplo escalar negativo de este, resultará en un rayo con dirección opuesta. Por convención, en los proyectores los rayos de luz atraviesan a lo largo de la dirección determinada por el vector director, mientras que en las cámaras, la luz atraviesa en la dirección opuesta al vector director, es decir, en la dirección en la que disminuye λ [**72**].

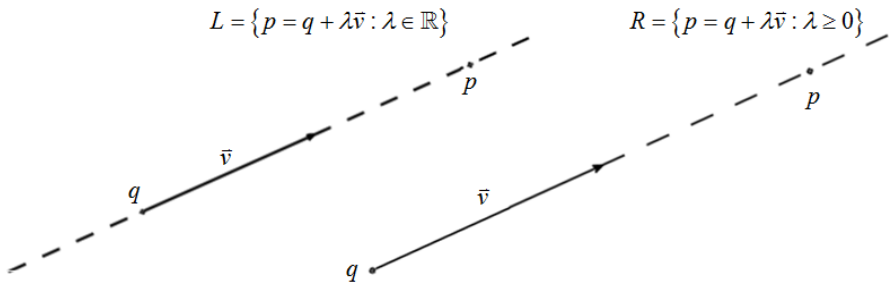


Figura A.3. Representación paramétrica de líneas (figura de la izquierda) y rayos (figura de la derecha).

Ejemplo A.1: Hallar un conjunto de ecuaciones paramétricas de la recta que pasa por los puntos $(-2,1,0)$ y $(1,3,5)$. A partir de $p = (-2,1,0)$ y $q = (1,3,5)$, construimos un vector director $\overrightarrow{pq}$ así:

$$\vec{v} = \overrightarrow{pq} = q - p = (1-(-2), 3-1, 5-0)$$

$$\vec{v} = (a, b, c) = (3, 2, 5)$$

Usando los números de dirección $a = 3$, $b = 2$, $c = 5$ y el punto $p = (-2, 1, 0)$, obtenemos:

$$L = \{m : p + \lambda\vec{v} \Rightarrow x = 3\lambda - 2, y = 2\lambda + 1, z = 5\lambda\}$$

Donde m tiene coordenadas (x, y, z), m es cualquier punto sobre la recta.

Ejemplo A.2: Encontrar la ecuación de la recta en el espacio ($\mathbb{R}^3$).

$$\text{Si } q \in L \Rightarrow \overrightarrow{oq} = \overrightarrow{op} + \lambda\vec{d}$$

$$(x, y, z) = (x_0, y_0, z_0) + \lambda(d_x, d_y, d_z)$$

$$\text{Si } q \in L \Leftrightarrow \exists\lambda \in \mathbb{R} \mid x = x_0 + \lambda d_x; y = y_0 + \lambda d_y; z = z_0 + \lambda d_z$$

Hallar la ecuación de la recta L que pasa por $p = (-3, 2, 1)$ y $q = (5, 1, 4)$. Además demostrar si los puntos $w = (21, -1, 10) \in L$ y $m = (20, 10, -2) \in L$.

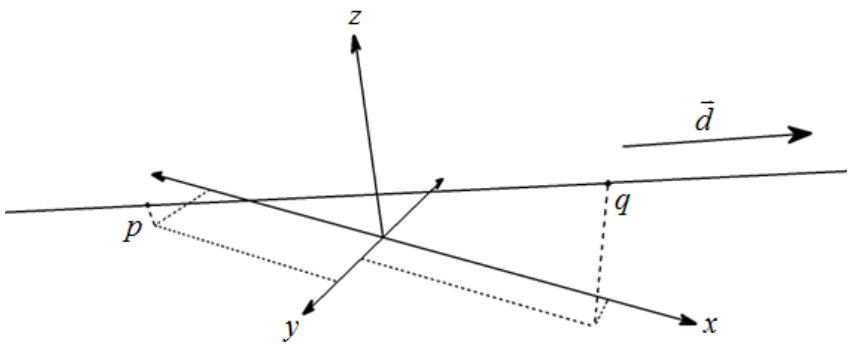


Figura A.4. Ecuación de la recta en el espacio.

Como la recta pasa por los puntos q y p, entonces el vector director $\vec{d}$ lo encontramos a partir de de $\vec{d} = \overrightarrow{qp} = p - q = (-3, 2, 1) - (5, 1, 4) = (-8, 1, -3)$. Una vez definido $\vec{d}$, se puede definir L como:

$$L = \{u \in L = q + \lambda\vec{d}\}$$

Donde u es un punto cualquiera que pertenece a la recta L.

$$(x, y, z) = (5, 1, 4) + \lambda(-8, 1, -3)$$

$$\begin{cases} x = 5 - 8\lambda \\ y = 1 + \lambda \\ z = 4 - 3\lambda \end{cases}$$

Ahora, si los puntos w y m pertenecen al plano debe existir un mismo $\lambda \in \mathbb{R}$ que satisfaga la expresión anterior. Para $w = (21, -1, 10)$ tenemos que:

$$\begin{cases} x = 5 - 8\lambda \Rightarrow 21 = 5 - 8\lambda \Rightarrow \lambda = -2 \\ y = 1 + \lambda \Rightarrow -1 = 1 + \lambda \Rightarrow \lambda = -2 \\ z = 4 - 3\lambda \Rightarrow 10 = 4 - 3\lambda \Rightarrow \lambda = -2 \end{cases}$$

Como si existe un λ que satisfaga el sistema de ecuaciones, entonces $w = (21, -1, 10) \in L$. Ahora para $m = (20, 10, -2)$ tenemos que:

$$\begin{cases} x = 5 - 8\lambda \Rightarrow 20 = 5 - 8\lambda \Rightarrow \lambda = -15/8 \\ y = 1 + \lambda \Rightarrow 10 = 1 + \lambda \Rightarrow \lambda = 9 \\ z = 4 - 3\lambda \Rightarrow -2 = 4 - 3\lambda \Rightarrow \lambda = 2 \end{cases}$$

Como no existe un $\lambda \in \mathbb{R}$ que satisfaga las tres ecuaciones, entonces el punto $m = (20, 10, -2)$ no pertenece a la línea L.

A.2.5 Representación paramétrica de planos

Similar a como las líneas son representadas de forma paramétrica, un plano π puede ser descrito especificando uno de sus puntos q y dos vectores directores linealmente independientes $\vec{v}_1$ y $\vec{v}_2$, ver Figura A.5. Cualquier otro punto p sobre el plano π puede ser descrito como el resultado de añadir múltiplos escalares $\lambda_1\vec{v}_1$ y $\lambda_2\vec{v}_2$ de los dos vectores al punto q [**72**], así:

$$\pi = \{ p = q + \lambda_1\vec{v}_1 + \lambda_2\vec{v}_2 : \lambda_1, \lambda_2 \in \mathbb{R} \} \tag{A.5}$$

Donde $\lambda_1, \lambda_2 \in \mathbb{R}$, si $p \equiv (x, y, z) \in \pi \Rightarrow (x, y, z) = (x_0, y_0, z_0) + \lambda_1\vec{v}_1 + \lambda_2\vec{v}_2$ para cada punto p del plano existe un único λ_1 y λ_2. Además, las ecuaciones paramétricas del plano serían:

$$(x, y, z) = (x_0, y_0, z_0) + \lambda_1(v_{1x}, v_{1y}, v_{1z}) + \lambda_2(v_{2x}, v_{2y}, v_{2z}) \tag{A.6}$$

$$\begin{cases} x = x_0 + \lambda_1 v_{1x} + \lambda_2 v_{2x} \\ y = y_0 + \lambda_1 v_{1y} + \lambda_2 v_{2y} \\ z = z_0 + \lambda_1 v_{1z} + \lambda_2 v_{2z} \end{cases} \quad \text{(A.7)}$$

Si dado un (x, y, z), se encuentra un λ_1 y un λ_2 que cumpla con las tres ecuaciones, entonces el punto pertenece al plano π. Si no existe un λ_1 y un λ_2 tal que se cumplan las tres ecuaciones, entonces el punto no pertenece al plano, ver *Ejemplo A.3.*

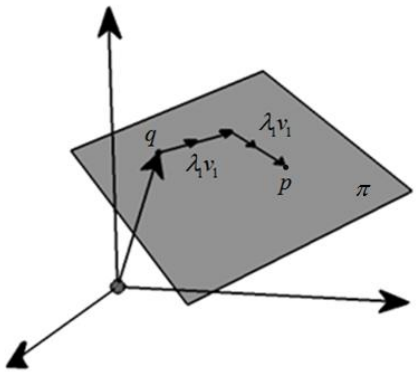


Figura A.5. Representación paramétrica de planos.

Como en el caso de las líneas, esta representación no es única, el punto q puede ser reemplazado por cualquier otro punto en el plano y los vectores $\vec{v}_1$ y $\vec{v}_2$ pueden ser reemplazados por cualquier otro par de vectores linealmente independientes, $\vec{v}_1$ y $\vec{v}_2$ que sean paralelos al plano.

A.2.6 Representación implícita de planos

Un plano π también se puede representar de *forma implícita* como un conjunto de ceros de una ecuación lineal de tres variables. Geométricamente, el plano puede ser descrito por uno de sus puntos q y un vector normal $\vec{n}$ al plano (ver Figura A.6). Un punto $p \in \pi$ si y solo si los vectores $\overrightarrow{qp} = (p - q)$ y $\vec{n}$ son ortogonales [**72**], de forma que:

$$\pi = \left\{ p : \vec{n}^T \cdot (p - q) = 0 \right\} \quad \text{(A.8)}$$

De nuevo, esta representación no es única, el punto q puede ser reemplazado por cualquier punto que pertenezca al plano π y el vector normal puede ser reemplazado por

cualquier vector $\vec{n}$, normal al plano, multiplicado por un escalar λ, tal que λ sea diferente de cero ($\lambda\vec{n}$), ver Ejemplo A.3.

Para convertir de la forma paramétrica a la forma implícita, podemos tomar el vector normal como el producto cruz de los vectores base $\vec{v}_1$ y $\vec{v}_2$, $\vec{n}=\vec{v}_1\times\vec{v}_2$. Ahora, para convertir de la forma implícita a la forma paramétrica, debemos encontrar dos vectores $\vec{v}_1$ y $\vec{v}_2$ linealmente independientes y que sean ortogonales al vector $\vec{n}$ o paralelos al plano π. De hecho, es suficiente con encontrar un solo vector $\vec{v}$, el otro se puede encontrar a partir de $\vec{v}_2=\vec{n}\times\vec{v}_1$. En ambos casos, q puede ser el mismo.

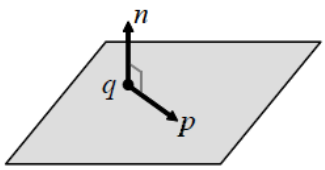


Figura A.6. Representación implícita de planos.

Ejemplo A.3: Hallar la ecuación paramétrica y la forma implícita del plano π a partir de la forma ecuación general:

$$\pi=\left\{(x,y,z)\in\mathbb{R}^3 : 2x+3y-z+8=0\right\}$$

Para encontrar la forma implícita del plano $\pi=\left\{p:\vec{n}^T\cdot(p-q)=0\right\}$ (Ecuación A.8) se necesita un punto q y un vector normal al plano $\vec{n}$. A partir de la forma general del plano $a(x-x_0)+b(y-y_0)+c(z-z_0)=0$, se sabe que los valores de (a,b,c) son las coordenadas del vector normal $\vec{n}$ al plano y (x_0,y_0,z_0) son las coordenadas de un punto que pertenece al plano [**74**]. Entonces se puede deducir que el vector normal al plano π es $\vec{n}=(2,3,-1)$. Ahora, para obtener el punto q que pertenece al plano se pueden, por ejemplo, dar valores arbitrarios a x y y, así:

$$2x+3y-z+8=0\Rightarrow z=2x+3y+8$$

$$x=3;\ y=-5 \text{ (Valores arbitrarios)}$$

$$z=2(3)+3(-5)+8$$

$$z=-1$$

Por lo tanto, el punto $q \in \pi$ es $q = (3, -5, -1)$. Ahora con q y $\vec{n}$ definidos, se puede escribir la ecuación implícita del plano así:

$$\pi = \left\{ p : \vec{\mathrm{n}}^T \cdot \overrightarrow{pq} = 0 \right\}$$

$$\pi = \left\{ p : (2,3,-1)^T \cdot (x-3, y+5, z+1) = 0 \right\}$$

Para encontrar la forma paramétrica del plano $\pi = \{ p = q + \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2 : \lambda_1, \lambda_2 \in \mathbb{R} \}$, se debe realizar una descomposición del punto genérico p a partir de $2x + 3y - z + 8 = 0$, así:

$$\pi = \{ p = q + \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2 : \lambda_1, \lambda_2 \in \mathbb{R} \}$$

$$p = q + \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2$$

$$(x, y, z) = (x_0, \mathrm{y}_0, \mathrm{z}_0) + \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2$$

Despejando z de la ecuación general del plano, el punto genérico $p = (x, y, z)$ queda:

$$(x, y, z) = (x, y, 2x + 3y + 8)$$

Ahora, se debe descomponer p en tres componentes principales así:

$$(x, y, z) = (x, 0, 2x) + (0, \mathrm{y}, 3y) + (0, 0, 8)$$

$$(x, y, z) = x(1, 0, 2) + y(0, 1, 3) + (0, 0, 8)$$

Por último se reemplaza $x = \vec{v}_1$ y $y = \vec{v}_2$, de tal modo que la forma paramétrica del plano quede así:

$$(x, y, z) = \underbrace{\lambda_1 (1, 0, 2)}_{\lambda_1 \vec{v}_1} + \underbrace{\lambda_2 (0, 1, 3)}_{\lambda_2 \vec{v}_2} + \underbrace{(0, 0, 8)}_{p}$$

Se debe verificar que $\vec{v}_1$ y $\vec{v}_2$ sean perpendiculares al vector normal $\vec{n}$, de esta forma se asegura que $\vec{v}_1$ y $\vec{v}_2$ sean paralelos al plano.

- $\vec{v}_1 \cdot \vec{n} = (1, 0, 2) \cdot (2, 3, -, 1) = 0$
- $\vec{v}_2 \cdot \vec{n} = (0, 1, 3) \cdot (2, 3, -, 1) = 0$

Si son perpendiculares ya que el producto punto entre el vector normal $\vec{n}$ y cada vector directo $\vec{v}_i$ es igual a cero. Además se debe verificar que el punto $p = (0, 0, 8)$ si pertenezca al plano π :

$$2x+3y-z+8=0 \Rightarrow 2(0)+3(0)-8+8=0 \therefore p \in \pi$$

Las ecuaciones paramétricas a partir de $(x,y,z)=\lambda_1(1,0,2)+\lambda_2(0,1,3)+(0,0,8)$, son:

$$\begin{cases} x=\lambda_1+0+0 \\ y=0+\lambda_2+0 \\ z=2\lambda_1+3\lambda_2+8 \end{cases}$$

A.2.7 Representación implícita de líneas

Una línea L puede ser descrita de forma implícita como la intersección de dos planos, ambos representados de forma implícita [**72**], de manera que:

$$L=\left\{p:\vec{n}_1^T\cdot\overrightarrow{pq}=\vec{n}_2^T\cdot\overrightarrow{pq}=0\right\} \quad \text{(A.9)}$$

Donde los dos vectores normales $\vec{n}_1$ y $\vec{n}_2$ son linealmente independientes (si $\vec{n}_1$ y $\vec{n}_2$ son linealmente dependientes, las dos ecuaciones describirían el mismo plano). Cuando $\vec{n}_1$ y $\vec{n}_2$ son linealmente independientes, las representaciones implícitas de los dos planos pueden ser definidas con respecto a un punto en común que pertenece a ambos planos (ver Figura A.7).

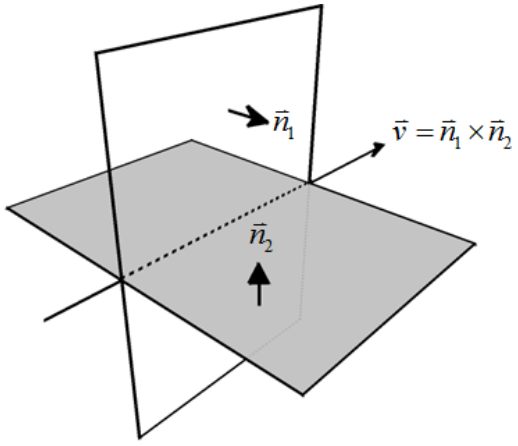


Figura A.7. Representación implícita de líneas.

Ya que una línea puede ser descrita como la intersección de muchos pares de planos, la representación no es única. El punto q puede ser reemplazado por cualquier otro punto que pertenezca a la intersección de ambos planos. Para convertir la forma paramétrica de una línea (Ecuación A.3) a la forma implícita (Ecuación A.9), se necesita encontrar

dos vectores linealmente independientes $\vec{n}_1$ y $\vec{n}_2$ (Ecuación A.9) ortogonales al vector director $\vec{v}$ (Ecuación A.3). Una forma de hacerlo es encontrando un vector no nulo $\vec{n}_1$ que sea ortogonal a el vector director $\vec{v}$ y luego tomar $\vec{n}_2$ como el producto cruz, $\vec{n}_2 = \vec{v} \times \vec{n}_1$. Para convertir la forma implícita a la forma paramétrica, se necesita encontrar un vector director no nulo $\vec{v}$ que sea ortogonal a $\vec{n}_1$ y $\vec{n}_2$, $\vec{v} = \vec{n}_1 \times \vec{n}_2$ y también cualquier múltiplo escalar de $\vec{v}$, ver Ejemplo A.4.

Ejemplo A.4: Una de las formas de la ecuación de la recta es plantearla como la intersección de dos planos.

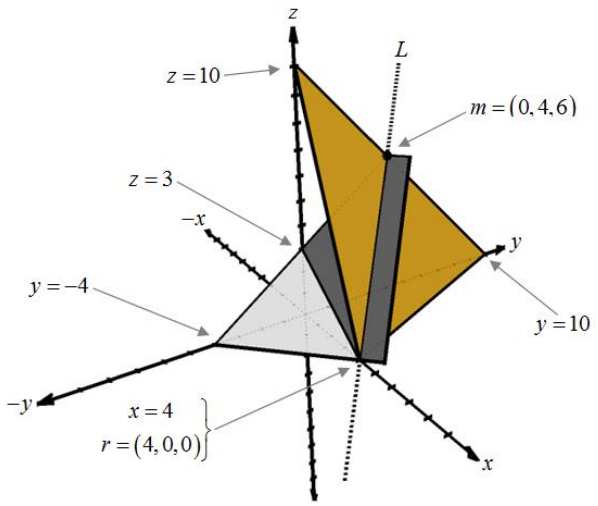


Figura A.8. Línea como la intersección de dos planos.

A partir de la Figura A.8, se encuentran las ecuaciones de los planos π_1 (blanco) y π_2 (amarillo), teniendo por donde cada plano corta los ejes del espacio coordenado se pueden escribir las ecuaciones segmentadas de los planos, así:

$$\pi_1 = \frac{x}{4} + \frac{y}{-4} + \frac{z}{3} = 1$$

$$\pi_2 = \frac{x}{4} + \frac{y}{10} + \frac{z}{10} = 1$$

El área coloreada en gris es la porción del plano π_1 que intersecta el plano π_2, simplificando las ecuaciones la línea L queda definida como:

$$L=\begin{cases}\pi_1=3x-3y+4z=12\\ \pi_2=5x+2y+2z=20\end{cases}$$

Para llevar esta expresión a la ecuación paramétrica del la línea en el espacio, donde $L=\{p=q+\lambda\vec{v}\}$, se deben descomponer las dos ecuaciones en una sola variable, así:

$$\left.\begin{aligned}\pi_1: z&=\frac{12-3x+3y}{4}\\ \pi_2: z&=\frac{20-5x-2y}{2}\end{aligned}\right\}\Rightarrow \frac{12-3x+3y}{4}=\frac{20-5x-2y}{2}$$

$$\boxed{y=4-x}$$

$$z=\frac{20-5x-2(4-x)}{2}\Rightarrow\boxed{z=6-\frac{3}{2}x}$$

Entonces, la descomposición, en este caso se está descomponiendo en términos de x, queda de la siguiente forma:

$$p=(x,y,z)=\left(x,4-x,6-\frac{3}{2}x\right)$$

$$\underbrace{(x,y,z)}_{p}=\underbrace{(0,4,6)}_{q}+\underbrace{x\left(1,-1,-\frac{3}{2}\right)}_{\lambda\vec{v}}$$

Para comprobar si es la ecuación de la línea L en el espacio, la cual corta los plano π_1 y π_2, se debe verificar que el vector director $\vec{v}=\left(1,-1,-3/2\right)$ sea paralelo al vector $\overrightarrow{rm}$ (ver Figura A.8), es decir que $\overrightarrow{rm}$ es una versión escalada de $\vec{v}$.

$$\overrightarrow{rm}=(0,4,6)-(4,0,0)=(-4,4,6)$$

$$4\cdot\vec{v}=4\cdot\left(1,-1,-\frac{3}{2}\right)=(-4,4,6)$$

Por lo tanto son paralelos, ya que el vector $\overrightarrow{rm}$ es un múltiplo del vector $\vec{v}$. También se puede encontrar la ecuación de la línea L buscando un punto que pertenezca a la intersección entre π_1 y π_2, por ejemplo, en la Figura A.8 se puede observar que el punto $r=(4,0,0)\in\pi_1\cap\pi_2$. Debido a que el punto r pertenece a la intersección de los planos, también pertenece la línea L, por lo tanto solo resta encontrar el vector director $\vec{v}$ el cual tiene que ser perpendicular a los vectores normales de los planos, $\vec{v}=\vec{n}_1\times\vec{n}_2$.

Ejemplo A.5: Calcular la intersección entre las rectas L_1 y L_2, además encontrar el ángulo entre ellas.

$$L_1 = \{(x, y, z) = (2, 0, -2) + \lambda(1, 4, 2) : \lambda \in \mathbb{R}\}$$

$$L_2 = \{(x, y, z) = (-3, 10, -2) + \alpha(4, 1, 3) : \alpha \in \mathbb{R}\}$$

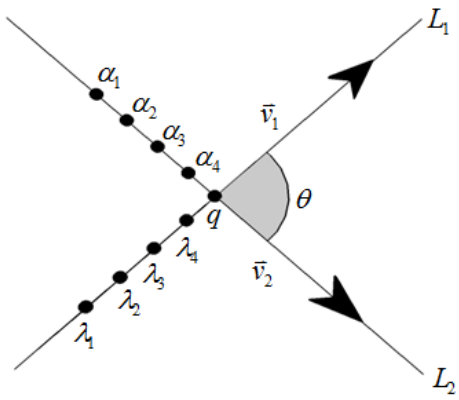


Figura A.9. Intersección entre dos rectas.

El punto q donde ambas rectas intersectan (ver Figura A.9) debe tener las mismas coordenadas en ambas ecuaciones de L_1 y L_2:

$$L_1 = \begin{cases} x = 2 + \lambda \\ y = 4\lambda \\ z = -2 + 2\lambda \end{cases} ; \; L_2 = \begin{cases} x = -3 + 4\alpha \\ y = 10 + \alpha \\ z = -2 + 3\alpha \end{cases}$$

Se iguala cada componente y se resuelve el sistema de ecuaciones:

$$2 + \lambda = -3 + 4\alpha$$
$$4\lambda = 10 + \alpha$$
$$-2 + 2\lambda = -2 + 3\alpha$$

Donde se deduce que $\lambda = 3$ y $\alpha = 2$. Para encontrar el punto de intersección se reemplaza el valor de λ y α en su correspondiente ecuación:

$$L_1 = \begin{cases} x = 2 + (3) = 5 \\ y = 4(3) = 12 \\ z = -2 + 2(3) = 4 \end{cases} \quad L_2 = \begin{cases} x = -3 + 4(2) = 5 \\ y = 10 + (2) = 12 \\ z = -2 + 3(2) = 4 \end{cases}$$

Por lo tanto, el punto de intersección $q=(5,12,4)$. Finalmente para calcular el ángulo θ, se usan los vectores directores $\vec{v}_1$ y $\vec{v}_2$ y se calcula el ángulo θ a partir de la definición del producto punto.

$$\vec{v}_1 \cdot \vec{v}_2 = \|\vec{v}_1\|\|\vec{v}_2\| Cos(\theta)$$

$$\theta = Cos^{-1}\left(\frac{14}{\sqrt{21}\cdot\sqrt{26}}\right)$$

Ejemplo A.6: Calcular la distancia de un punto a una recta en el espacio, se tiene una línea L en $\mathbb{R}^3$ y un punto q que no pertenece a L, así:

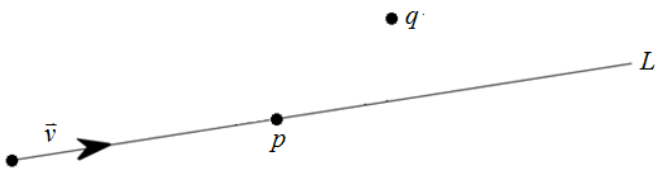


Figura A.10. Distancia entre una recta y un punto.

Se necesita encontrar algo que permita calcular la distancia más corta entre ambos, para esto se hace uso del teorema de Pitágoras y la proyección de un vector sobre otro vector (ver Figura A.11), teniendo en cuenta que:

- El cateto opuesto es equivalente a la proyección del vector $\overrightarrow{pq}$ sobre $\vec{v}$.
- La distancia más corta a la recta es el modulo del cateto opuesto, $dist = \|\overrightarrow{c.o}\|$.
- Se puede plantear al vector $\overrightarrow{pq}$ como las suma del vector $\overrightarrow{c.a}$ y el vector $\overrightarrow{c.o}$.

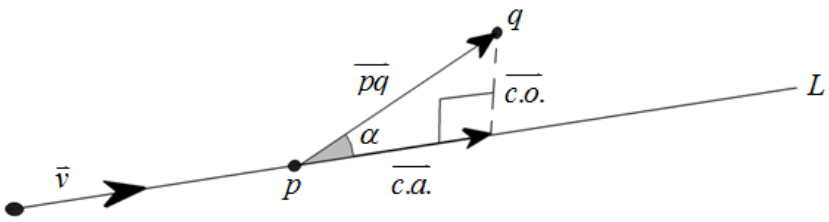


Figura A.11: Distancia entre una recta y un punto.

Por lo tanto, para calcular la distancia mínima se puede proceder de la siguiente forma:

$$\overrightarrow{pq} = proy_{\vec{v}}\,\overrightarrow{pq} + \|\overrightarrow{c.o}\|$$

$$\left\|\overrightarrow{c.o}\right\| = \overrightarrow{pq} - \frac{\overrightarrow{pq}\cdot\vec{v}}{\left\|\vec{v}\right\|^2}\cdot\vec{v}$$

Ejemplo A.7: Calcular la distancia entre dos rectas paralelas en el espacio.

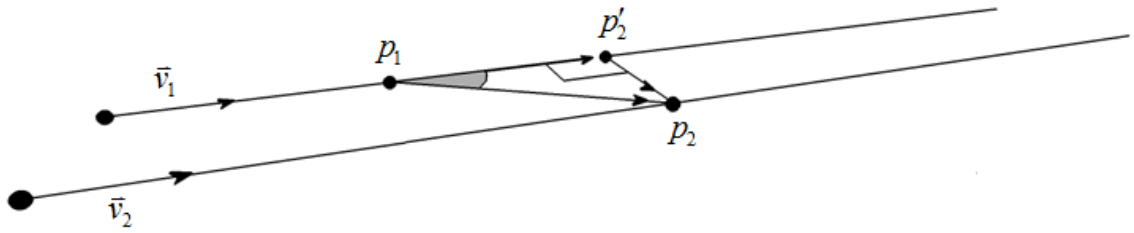


Figura A.12. Distancia entre dos líneas paralelas.

El vector $\overrightarrow{p_1p_2'}$ es la proyección de $\overrightarrow{p_1p_2}$ sobre $\vec{v}_1$, además, teniendo en cuenta el ejemplo anterior, se puede deducir que $\overrightarrow{p_1p_2} = \overrightarrow{p_1p_2'} + \vec{x}$, por lo tanto:

$$\vec{x} = \overrightarrow{p_1p_2} - \underset{proy_{\vec{v}_1}\overrightarrow{p_1p_2}}{\overrightarrow{p_1p_2'}}$$

$$\vec{x} = \overrightarrow{p_1p_2} - \frac{\overrightarrow{p_1p_2}\cdot\vec{v}_1}{\left\|\vec{v}_1\right\|^2}\cdot\vec{v}_1$$

Donde la distancia mínima entre ambas rectas es igual al modulo de $\vec{x}$, $\left\|\vec{x}\right\|$.

Ejemplo A.8: Calcular la distancia entre líneas alabeadas en el espacio. En el espacio dos líneas pueden ser oblicuas entre sí (no paralelas) y no cortarse en ningún punto, ver Figura A.13. Esto quiere decir que son líneas que no perteneces al mismo plano. Sin embargo el cálculo del ángulo que forman entre ellas es el mismo que el del Ejemplo A.5.

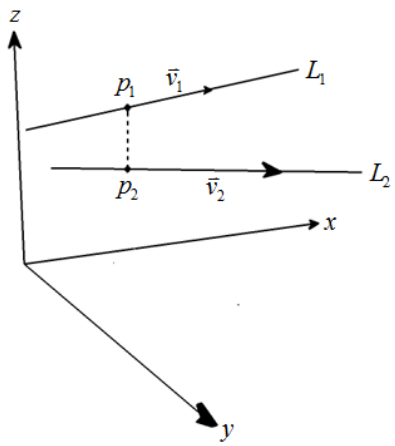


Figura A.13. Distancia entre líneas alabeadas.

Si se tienen dos planos paralelos, la distancia entro esos dos planos se debe medir sobre la perpendicular a ambos. Como son paralelos debe existir una línea perpendicular a ambos planos y cualquier vector definido por un punto del plano y un punto del otro plano proyectado sobre la dirección de la normal, será entonces la distancia mínima.

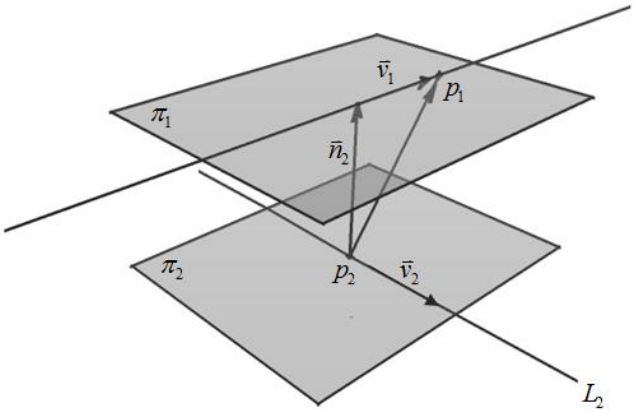


Figura A.14. Distancia entre líneas alabeadas.

No importa qué puntos se tomen de los planos, si un punto pertenece al plano π_1 y el otro pertenece al plano π_2, la distancia siempre es la misma porque los planos son paralelos.

Se debe encontrar un vector normal al plano π_1 y al plano π_2, el cual tiene que ser perpendicular a L_1 y L_2. Entonces, el vector normal a los planos $\vec{n}$ puede ser calculado

a partir del producto vectorial de los vectores directores de cada línea, $\vec{n} = \vec{v}_1 \times \vec{v}_2$. Por lo tanto la distancia mínima entre los dos vectores sería el modulo del vector $\vec{x} = proy_{\vec{n}} \overrightarrow{p_1 p_2}$, (la distancia mínima es $\|\vec{x}\|$).

A.3 Reconstrucción a través de triangulación

Es una práctica común que los patrones de iluminación contengan líneas o punto identificables. Bajo el esquema de un proyector, una línea proyectada sobre la superficie crea un plano de luz [**75**]. Ahora, la intersección de un rayo de luz de la cámara con el objeto escaneado, se considera como un punto iluminado, mientras que la intersección de un plano de luz con el objeto generalmente contiene varios segmentos curvos iluminados [**73**].

Dichos segmentos curvos está compuesto de muchos puntos iluminados, se asumirá que la posición y la orientación del proyector y la cámara [**72**] con respecto al sistema coordenado son conocidas. Bajo este supuesto, la ecuación del plano proyectado así como el del rayo de la cámara, correspondientes a los puntos iluminados, son definidos por parámetros medibles. A partir de estas medidas, la ubicación de los puntos iluminados puede ser recuperada a partir de la intersección del plano de luz proyectado con los rayos correspondientes de la cámara a los puntos iluminados. Mediante estos procedimientos se puede corregir la ambigüedad al calcular la profundidad con una cámara esteneopeica [**76**] [**77**] [**78**].

A.2.8 Intersección entre líneas y planos

Calcular la intersección entre una línea y un plano se realiza de manera directa si la línea es representada de forma paramétrica (Ecuación A.3).

$$L = \{ p = q_L + \lambda \vec{v} : \lambda \in \mathbb{R} \}$$

Y si el plano está definido de forma implícita (Ecuación A.8).

$$\pi = \{ p : \vec{n}^T \cdot (p - q_\pi) = 0 \}$$

La línea y el plano puede que no se intersecten, en cuyo caso se dice que la línea y el plano son paralelos, este es el caso en el que el vector director $\vec{v}$ de la línea L y el vector normal $\vec{n}$ del plano π, son ortogonales. También podrían no intersectarse si la línea L pertenece al plano π, en ambos casos se tiene que $\vec{n}^T \cdot \vec{v} = 0$. Si los vectores $\vec{n}$ y $\vec{v}$ no son ortogonales, entonces la línea y el plano se intersectan en exactamente un punto p. Ya que el punto $p \in L$, puede ser escrito como $p = q_L + \lambda\vec{v}$, para un valor de λ que debe ser calculado teniendo en cuenta que el punto p también pertenece al plano, $p \in \pi$. El valor de λ debe satisfacer la siguiente ecuación lineal:

$$\vec{n}^T \cdot (p - q_\pi) = \vec{n}^T \cdot (\lambda\vec{v} + q_L - q_\pi) = 0 \tag{A.10}$$

Despejando λ de la ecuación queda:

$$\lambda = \frac{\vec{n}^T \cdot (q_\pi - q_L)}{\vec{n}^T \cdot \vec{v}} \tag{A.11}$$

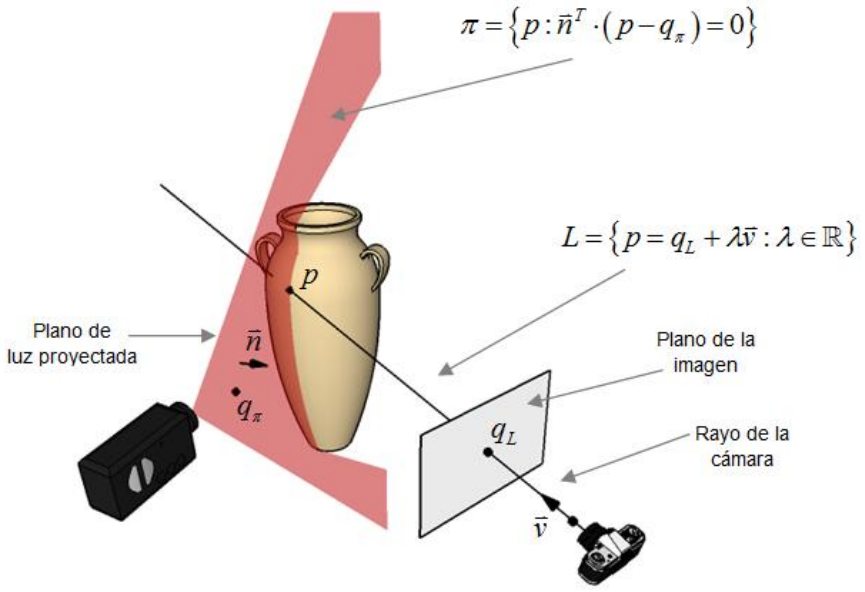


Figura A.15. Triangulación a partir de la intersección entre un plano y una línea.

Ya que se asume que la línea L y el plano π no son paralelos ($\vec{n}^T \cdot \vec{v} \neq 0$) esta expresión es válida [**72**]. Geométricamente podemos interpretar la intersección entre una línea y un plano como se muestra en la Figura A.15, teniendo en cuenta que la línea se

interpreta como el rayo de luz en dirección hacia la cámara y el plano se interpreta como el plano del laser que es proyectado sobre el objeto [**76**].

B. Apéndice B: KD-tree

Cada nivel de un *kd tree* divide a todos los nodos hijos a lo largo de una dimensión específica (ya sea x, y o z), usando un *híper-plano* el cual es perpendicular a un determinado eje, en *BSP* (*Binary Space Partitioning*) es común hablar de *híper-planos*, estos son planos que pasan a través de un punto especifico que parte el espacio en dos.

Cada nodo en un árbol puede tener dos hijos, hijo izquierdo y derecho, estos serán seleccionados basándose en la dimensión por la cual fue dividido el espacio, teniendo en cuenta que los hijos del lado izquierdo serán los puntos menores a la mediana de los valores en la dimensión en la cual se dividió el espacio y los hijos del lado derecho serán los mayores a la mediana en dicha dimensión. Si un nodo no tiene hijos, este nodo se conoce con el nombre de *nodo* hoja.

La raíz del árbol es el punto que más se acerca a la mediana de los datos en una dimensión específica. Este proceso se repite hasta que se analicen todos los puntos, teniendo en cuenta que en cada iteración se usa la dimensión siguiente. En la primera iteración del algoritmo, se puede escoger la dimensión por la cual se dividirá el espacio teniendo en cuenta en cuál de los ejes los datos estén más dispersos.

Ejemplo B.1: Encontrar el k-d tree del conjunto de datos $\mathcal{U}$ de dos dimensiones.

$$\mathcal{U} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}$$

$$p_1 = (0.8147, 0.1576) \quad p_6 = (0.0975, 0.1419)$$
$$p_2 = (0.9058, 0.9706) \quad p_7 = (0.2785, 0.4218)$$
$$p_3 = (0.1270, 0.9572) \quad p_8 = (0.5469, 0.9157)$$
$$p_4 = (0.9134, 0.4854) \quad p_9 = (0.9575, 0.7922)$$
$$p_5 = (0.6324, 0.8003) \quad p_{10} = (0.9649, 0.9595)$$

Lo primero que se debe hacer es partir el espacio con una *híper-recta*, ya que el espacio es de dos dimensiones, la cual divide en dos el conjunto de datos, a lo largo de una dimensión. La pregunta es, ¿a lo largo de que dimensión se debe partir el espacio?, pues hay varias técnicas para hacerlo, acá se encontrará la dispersión de los datos en x y en y, el espacio será dividido a lo largo del eje de mayor dispersión.

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(p_{x_i} - \bar{p}_x\right)^2} = 0.3459 \quad \sigma_y = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(p_{y_i} - \bar{p}_y\right)^2} = 0.3308$$

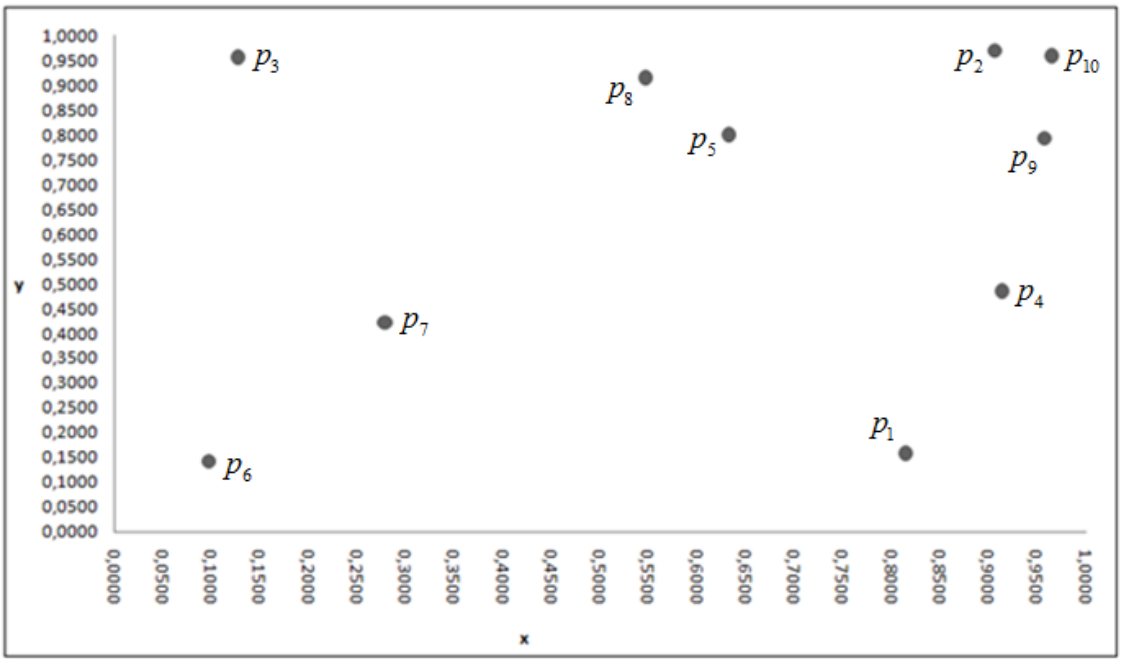


Figura B.1. Conjunto de datos ubicado en el espacio coordenado.

Como la mayor dispersión en los datos se encuentra en el eje x, entonces la *híper-recta* corta a lo largo de este eje. Ahora se debe encontrar la mediana de los datos en la dimensión x ($mediana = 0.7235$). A partir del valor de la mediana encontramos la raíz del árbol, la cual será el valor que más se acerque a ella por debajo en la dimensión x, este valor corresponde al punto p_6, a lo largo de este se traza una línea en $x = 0.6324$ que divida el espacio coordenado en dos. La raíz no se considera en los subconjuntos de la izquierda y derecha de la híper-recta. El conjunto $\mathcal{U}$ queda dividido así:

$$raiz = p_5 = (0.6324, 0.8003)$$

Lado izquierdo de la híper-recta

$$p_3 = (0.1270, 0.9572)$$
$$p_6 = (0.0975, 0.1419)$$
$$p_7 = (0.2785, 0.4218)$$
$$p_8 = (0.5469, 0.9157)$$

Lado derecho de la híper-recta

$$p_1 = (0.8147, 0.1576)$$
$$p_2 = (0.9058, 0.9706)$$
$$p_4 = (0.9134, 0.4854)$$
$$p_9 = (0.9575, 0.7922)$$
$$p_{10} = (0.9649, 0.9595)$$

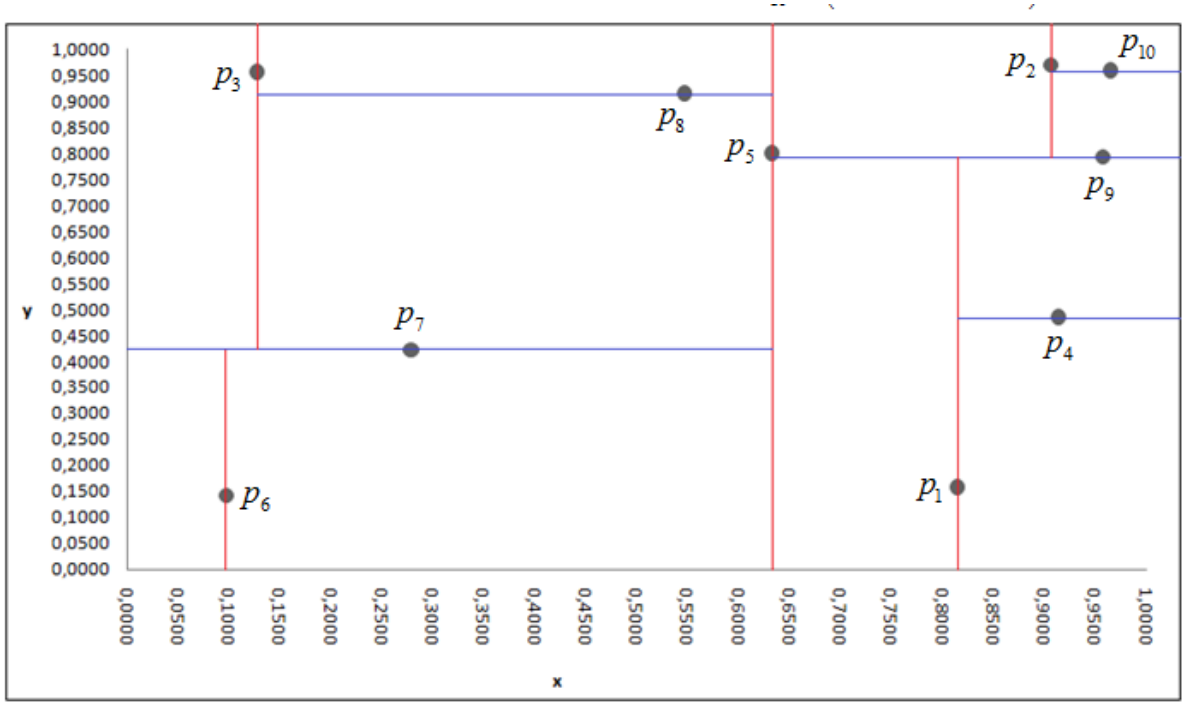


Figura B.2. Partición binaria del espacio usando un árbol k-dimensional.

Ahora se repite el mismo proceso para los hijos de la izquierda y para los de la derecha, solo que esta vez se divide el espacio a lo largo del eje y, los puntos que se obtengan como raíces de estas particiones, serán el hijo izquierdo y el hijo derecho de la raíz principal, ver Figura B.2. Este proceso se repite hasta que se procesen todos los puntos, teniendo en cuenta que a medida que se baja de nivel en el árbol, la dimensión por la cual se divide el espacio se va intercalando. Finalmente el árbol k-dimensional se puede ver en la Figura B.3.

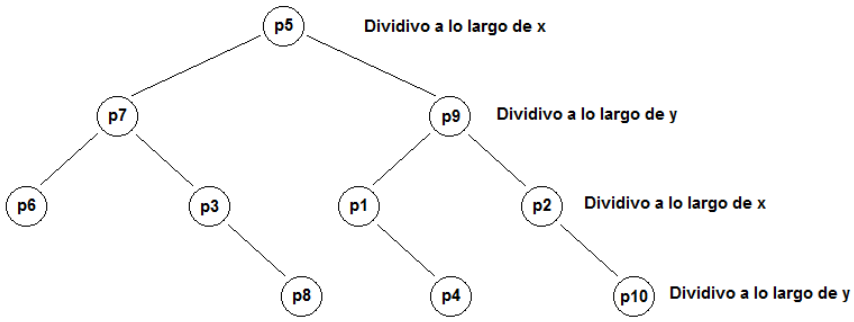


Figura B.3. Árbol k-dimensional.

Si se quisiera encontrar el punto p_2 en el árbol, por ejemplo, se comienza la búsqueda siempre a partir del nodo raíz, como este nodo parte el espacio a lo largo del eje x, entonces se compara la coordenada x del punto p_2 con la del nodo raíz, ya que $p_{2x} = 0.9649 > p_{5x} = 0.6324$, la búsqueda del punto se realiza hacia el lado derecho del árbol, lo cual indica que todos los nodos a la izquierda de la raíz son descartados. Ahora se continúa la búsqueda a partir del hijo derecho de la raíz, el cual parte el espacio a lo largo del eje y, por lo tanto comparamos la coordenada y del punto p_2 con la del punto p_9, como $p_{2y} = 0.9706 > p_{9y} = 0.7922$, entonces la búsqueda continúa hacia la derecha del árbol. Es aquí donde se encuentra el punto, al notar que las coordenadas son iguales.

C. Apéndice C: Calculo de la normal asociada a un punto

A lo largo de este apéndice se pretende calcular el vector normal de un punto $p_i \in \mathcal{P}$, donde $\mathcal{P}$ está definido como una nube de puntos que representan la superficie de un camaleón, ver Figura C.1.

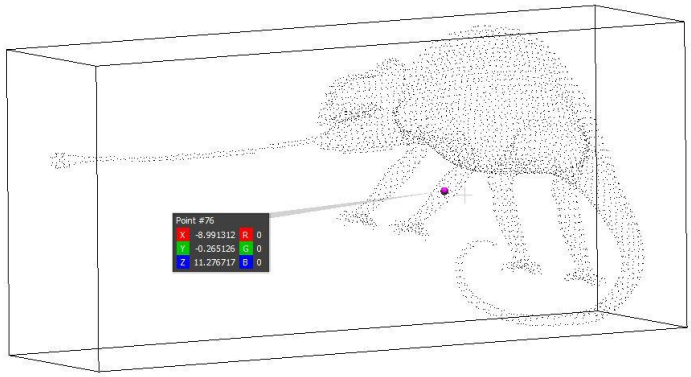


Figura C.1. Nube de puntos de un camaleón, en el recuadro se muestra la información de los puntos de interés.

El punto señalado en la Figura C.1, es el punto de interes p_{76} a partir del cual se calculará el vector normal $\vec{n}$ a la superficie. Teniendo en cuenta el analisis de componente principales mencionado anteriormente en esta seccion, el primer paso para poder estimar la normal es encontrar los k vecinos mas cercanos, por motivos practicos en este ejemplo solo se calculan 10 vecinos. Estos se denotan como los $\mathcal{P}^{10}$ de p_{76}, entonces el conjunto de los 10 vecinos mas cercanos a p_{76} es: $\mathcal{P}^{10} = \left\{ p_1^{10}, p_2^{10}, p_3^{10}, ... p_9^{10}, p_{10}^{10} \right\}$ donde los valores de los $p_j^{10} \in \mathcal{P}^{10}$ son:

$$p_1^{10} = (-8.9,-0.26,11.3) \quad p_6^{10} = (-8.9,-0.2,12.3)$$
$$p_2^{10} = (-8.7,1.6,12.5) \quad p_7^{10} = (-8.8,-1.2,11.5)$$
$$p_3^{10} = (-8.3,1.3,11.7) \quad p_8^{10} = (-8.9,-1.1,10.4)$$
$$p_4^{10} = (-8.3,0.4,10.8) \quad p_9^{10} = (-8.7,-0.4,10.5)$$
$$p_5^{10} = (-8.6,0.8,11.3) \quad p_{10}^{10} = (-9.05,0.8,12.4)$$

Ahora, dada una vecindad $\mathcal{P}^k$ de un punto p_i, el siguiente paso es encontrar el centro de masa del conjunto de puntos, para lo cual se utiliza la Ecuación 2.4, el centro de masa es:

$$v = \frac{1}{10}\sum_{j=1}^{10} p_j = (-8.71,0.174,11.47)$$

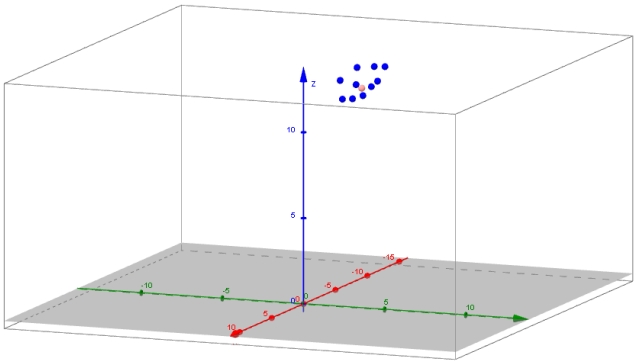


Figura C.2. Puntos vecinos del punto de interés.

A partir del centro de masa se debe minimizar $\left(p_j^k - v\right)^2$, de forma tal que la Ecuación 2.5 tienda a cero. Para realizar esto se calcula la matriz de covarianza así:

$$C = \frac{1}{k}\sum_{j=1}^{k}\left(p_j - v\right)\cdot\left(p_j - v\right)^T$$

$$C = \frac{1}{k}\sum_{j=1}^{k} C_j$$

Donde, cada C_j está definida de la siguiente manera:

$$C_j = \left(p_j - v\right)\cdot\left(p_j - v\right)^T = \begin{bmatrix} p_{jx} - v_x \\ p_{jy} - v_y \\ p_{jz} - v_z \end{bmatrix}\cdot\begin{bmatrix} p_{jx} - v_x & p_{jy} - v_y & p_{jz} - v_z \end{bmatrix}$$

$$C_j = \begin{bmatrix} (p_{jx}-v_x)^2 & (p_{jx}-v_x)(p_{jy}-v_y) & (p_{jx}-v_x)(p_{jz}-v_z) \\ (p_{jx}-v_x)(p_{jy}-v_y) & (p_{jy}-v_y)^2 & (p_{jy}-v_y)(p_{jz}-v_z) \\ (p_{jx}-v_x)(p_{jz}-v_z) & (p_{jy}-v_y)(p_{jz}-v_z) & (p_{jz}-v_z)^2 \end{bmatrix}$$

La sumatoria de todos los C_j, dividido el número de vecinos, da como resultado la matriz de covarianza del conjunto de datos.

$$C = \begin{bmatrix} 0.0580 & 0.0938 & -0.0440 \\ 0.0938 & 0.8305 & 0.3694 \\ -0.0440 & 0.3694 & 0.5261 \end{bmatrix}$$

La solución para el vector normal $\vec{n}$ es dada a partir del análisis de los vectores y valores propios de C.

$$\begin{matrix} & \textit{eigenvalues} & & & \textit{eigenvectors} & \\ \begin{bmatrix} \lambda_1=0.0244 & 0 & 0 \\ 0 & \lambda_2=0.3096 & 0 \\ 0 & 0 & \lambda_3=1.0807 \end{bmatrix} & & & \begin{matrix} \vec{v}_1 \rightarrow \\ \vec{v}_2 \rightarrow \\ \vec{v}_3 \rightarrow \end{matrix} & \begin{bmatrix} 0.9431 & 0.3282 & 0.0527 \\ -0.2228 & 0.5064 & 0.8330 \\ 0.2467 & -0.7974 & 0.5507 \end{bmatrix} \end{matrix}$$

Donde, el vector propio asociado al valor propio mas pequeño es la estimación de la normal en el punto de interés, como el valor propio mas pequeño equivale a $\lambda_1=0.0244$, entonces el vector normal $\vec{n}$ equivale a $\vec{v}_1=(0.9431,0.3282,0.0527)$, ver Figura C.3. El plano tangente a la superficie, en este ejemplo, se calcula a partir de los vectores $\vec{v}_2$, $\vec{v}_3$ y el vector normal $\vec{n}=\vec{v}_1$.

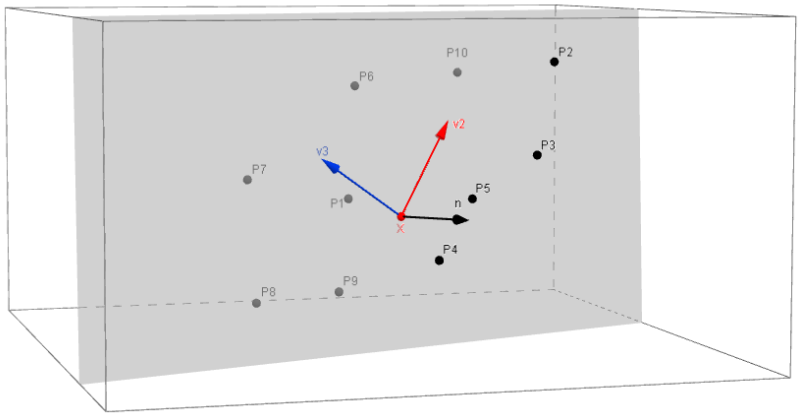


Figura C.3. Vectores propios asociados a la matriz de covariancia.

D. Apéndice D: Encuesta de percepción

1) ¿En cuál de las tres nubes de puntos le parece que es más clara para reconocer el modelo?

Técnica I ☐ Técnica II ☐ Técnica III ☐

Figura D.1. Modelo de dragón de la Universidad de Stanford. El modelo fue muestreado con la Técnica I, II y III respectivamente, las tres nubes de puntos tienen 8753 puntos, lo cual corresponde al 2% del total de puntos.

3) Mencione los aspectos y características que lo llevaron a escoger el modelo de la primera pregunta:

_____

_____

_____

2) ¿Con cuál resolución identifica el modelo para cada una de las tres nubes de puntos presentadas?

TÉCNICA I	**TÉCNICA II**	**TÉCNICA III**
○ 0.5	○ 0.5	○ 0.5
○ 1	○ 1	○ 1
○ 2	○ 2	○ 2
○ 10	○ 10	○ 10
○ 20	○ 20	○ 20

Figura D.2. El primer modelo de la figura es un caballo que pertenece al repositorio de la Universidad de Stanford, este fue muestrea con la técnica I. El modelo del centro es una cabeza, este pertenece al repositorio de PCL, este fue muestreado con la técnica II. Finalmente el modelo de la derecha es un conejo que pertenece al repositorio de la Universidad de Stanford, este fue muestreado con la técnica III. Cada modelo está representado por el 0.5% de la totalidad de sus datos.

En todas las preguntas de las encuestas se utilizaron nubes de puntos diferentes. Para la tercera pregunta, se presentaron tres nubes de puntos diferentes y se iba aumentando la resolución de cada uno según lo pidiera el encuestado. Las nubes de puntos que fueron mostradas en la encuesta se pueden ver en el Apéndice E.

E. Apéndice E: Modelos 3D

En este apéndice se muestran algunas imágenes de los modelos usados a lo largo del documento, sin embargo muchos de estos modelos no se pueden apreciar correctamente debido a que estos modelos son tridimensionales y no poseen una malla definida ni color. Por lo tanto, adjunto a esta tesis se encuentran los archivos de los modelos usados, estos han sido convertidos en formato *pcd* (*Point Cloud Data*), el cual es el formato estándar de la librería PCL [**58**].

Tabla E.1. Lista de nubes de puntos usadas a lo largo de este trabajo.

Nube de puntos	Repositorio	Cantidad de puntos
Armadillo.ply	The Stanford 3D Scanning Repository	172.974
Bunny.ply	The Stanford 3D Scanning Repository	35.947
Dragon.ply	The Stanford 3D Scanning Repository	437.645
Skeleton Hand.ply	Large Geometric Models Archive at Georgia Institute of Technology	327.323
Happy Buddha.ply	The Stanford 3D Scanning Repository	543.652
Horse.ply	Large Geometric Models Archive at Georgia Institute of Technology	48.485
Model.pcd	PLC Repository	48.485
Table_scene_lms400.pcd.	PLC Repository	460.400
Room.pcd	PLC Repository	968.520
Armchair.pcd	PLC Repository	62.406
duck_triangulate.ply	Meshlab samples	651.342
Pez.ply	Universidad Nacional sede Manizales	234.226
Seashell.ply	Meshlab samples	853.521
CellTower.pcd	PLC Repository (Trimble)	2’219.774
Laura.ply	Meshlab samples	733.318
Denver_Pipes_Portion.pcd	PLC Repository (Trimble)	3.801.816
Stairs.pcd	PLC Repository	312.833

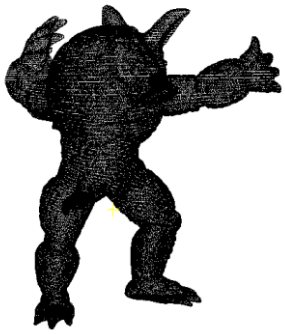


Figura E.1. Armdillo.ply

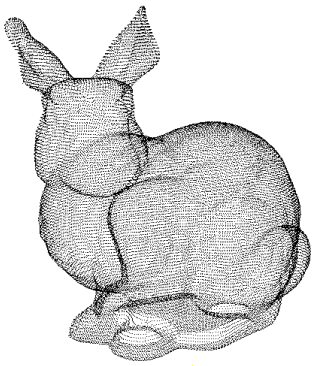


Figura E.2. Bunny.ply

Figura E.3. Dragon.ply

Figura E.4. Skeleton Hand.ply

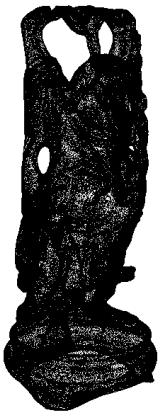


Figura E.5. Happy Buddha.ply

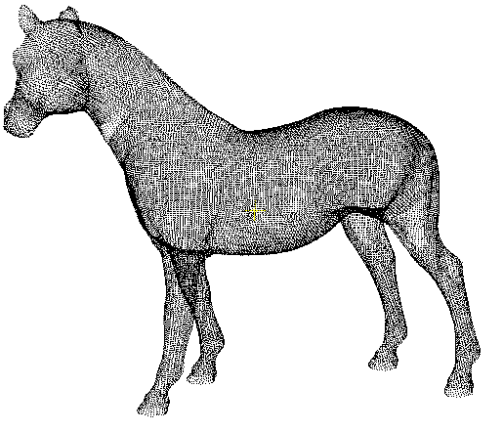


Figura E.6. Horse.ply

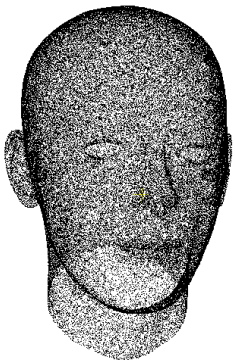


Figura E.7. model.pcd

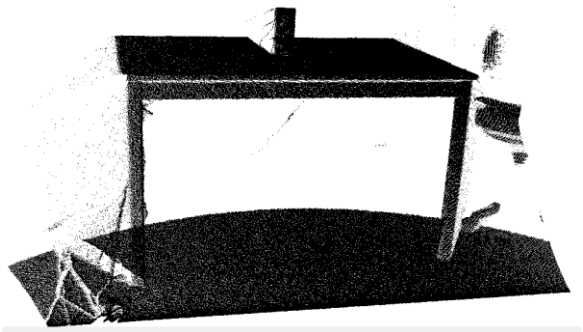


Figura E.8. table_scene_lms400.pcd

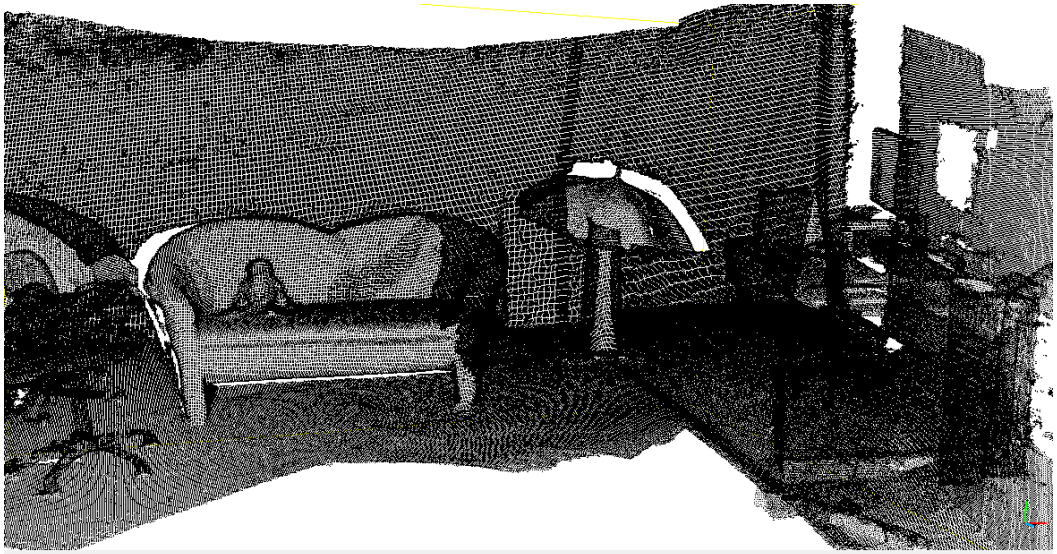


Figura E.9. room.pcd

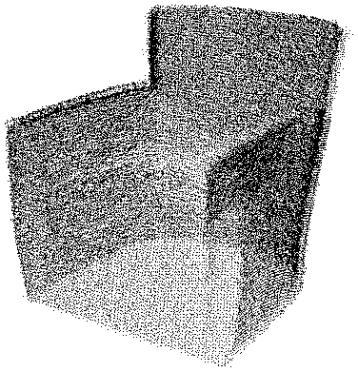


Figura E.10. Armchair.pcd

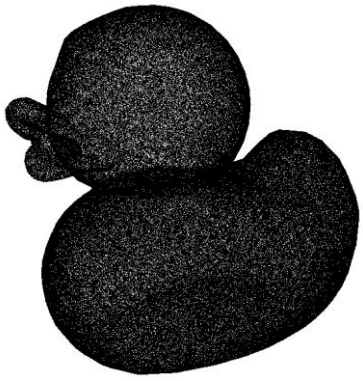


Figura E.11. duck_triangulate.ply

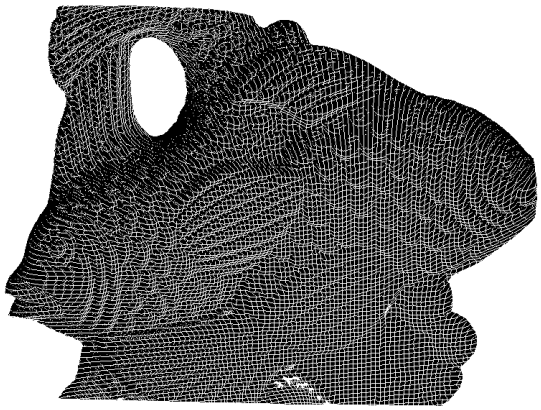


Figura E.12. Pez.ply

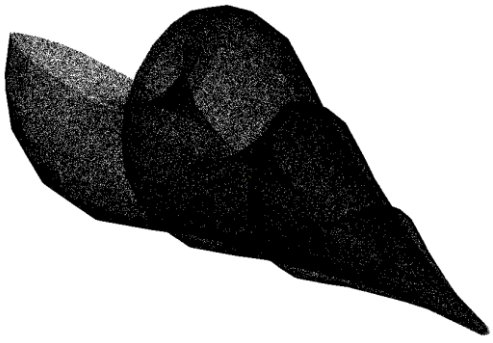


Figura E.13. Seashell.ply

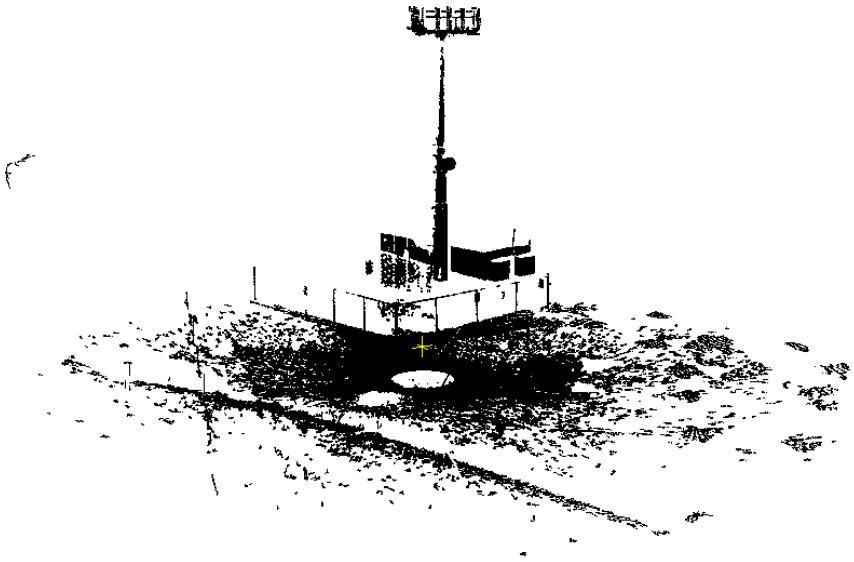


Figura E.14. CellTower.pcd

7. Bibliografía

[1] Radu Bogdan Rusu, *Semantic 3D Object Maps for Everyday Manipulation in Human Living Enviroments, Dissertation*. München, Germany: Institut für Informatik der Technischen Universität München, 2010.

[2] A., Gross, M. Hubeli, "Fairing of Non-Manifolds for Visualization. IEEE Visualization 00, 2000.".

[3] M. Gross, "Graphics in Medicine: From Visualization to Surgery. ACM Computer Graphics, Vol. 32, 1998".

[4] R., Roth, M. Peikert, "The “Parallel Vectors” Operator - A Vector Field Visualization Primitive. IEEE Visualization 99, 1999".

[5] M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., Fulk, D. Levoy, "The Digital Michelangelo Project: 3D Scanning of Large Statues. SIGGRAPH 00, 2000".

[6] Alberto Jaspe Villanueva, "Point Cloud Manager Sistema multirresolución para el tratamiento de grandes datasets de nubes de puntos 3D, A Coruña," 3 de Septiembre 2012.

[7] S., Strasser, W. Gumhold, "Real time compression of triangle mesh connectivity," *SIGGRAPH ’98*, pp. 133–140, 1998.

[8] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE Transactions on Visualization and Computer Graphics. Volume 5*, pp. 133–140, 1999.

[9] C., Gotsman, C. Touma, "Triangle mesh compression," *Graphics Interface*, pp. 26–34, 1998.

[10] Stanford University. (2015, March) The Digital Michelangelo Project. [Online]. http://graphics.stanford.edu/projects/mich/

[11] Szymon Rusinkiewicz and Marc Levoy, "QSplat: a multiresolution point rendering system for large meshes," *Proceedings of the 27th annual conference on Computer*

graphics and interactive techniques SIGGRAPH '00, pp. 343-352, 2000.

[12] Jag Mohan Singh and P. J. Narayanan, "Progressive Decomposition of Point Clouds Without Local Planes," *Computer Vision, Graphics and Image Processing, ISBN 978-3-540-68301-8*, pp. 364-375, 2006.

[13] C. Moenning and N. A. Dodgson., "A new point cloud simplification algorithm," *The 3rd IASTED International Conference on Visualization, Imaging and Image Processing, Belmadena, Spain, to appear*, 2003.

[14] C. Moenning and N. A. Dodgson., "Fast Marching farthest point sampling for implicit surfaces and point clouds," *Computer Laboratory Technical Report No. 565, University of Cambridge, UK*, 2003.

[15] J. Giesen and J. Hudson T. K. Dey, "Decimating Samples for Mesh Simplification," *Proc. 13th Canadian Conference on Computational Geometry, Waterloo, Canada*, pp. 85–88, 2001.

[16] J.-D. Boissonnat and F. Cazals, "Coarse-to-fine surface simplification with geometric guarantees," *EUROGRAPHICS ’01, Conf. Proc., Manchester, UK*, pp. 490–499, 2001.

[17] A. Okabe, B. Boots, and K. Sugihara, *Spatial Tessellations - Concepts and Applications of Voronoi Diagrams.*, 2nd ed. Chicester: John Wiley & Sons, 2000.

[18] W. Cook and A. Rohe, "Computing minimum-weight perfect matchings," *INFORMS Journal on Computing 11*, pp. 138–148, 1999.

[19] A. Moffat, R.M. Neal, and I.H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems (TOIS)*, vol. 16, pp. 256-294 , 1998.

[20] Mark Pauly, Markus Gross, and Leif P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Proceedings of the conference on Visualization '02 ISBN 0-7803-7498-3*. Boston, Massachusetts: IEEE Computer Society, 2002, pp. 163-170.

[21] Enrico Gobbetti and Fabio Marton, "Layered point clouds," *Computers & Graphics, Volume 28, ISSN 0097-8493*, pp. 815-826, 2004.

[22] H. Cline W. Lorensen, "Marching cubes: A high resolution 3D," *Proceedings of SIGGRAPH*, pp. 163–169, 1987.

[23] H. Benhabiles, O. Aubreton, H. Barki, and H. Tabia, "Fast simplification with sharp feature preserving for 3D point clouds," *International Symposium on Programming*

and Systems (ISPS), vol. 11, pp. 47-52, April 2013.

[24] Xiao Zhaoxia and Huang Wenming, "Kd-tree Based Nonuniform Simplification of 3D Point Cloud," *International Conference on Genetic and Evolutionary Computing*, vol. 3, pp. 339-342, Oct 2009.

[25] Haoyong Li, Pin Xu, and Yinghua Shen, "A self-adaption fast point cloud simplification algorithm based on normal eigenvalues," *International Congress on Image and Signal Processing (CISP)*, vol. 7, pp. 852-856, Oct 2014.

[26] Guo Xianglin and Pang Mingyong, "Point sets simplification using local surface analysis," *IEEE International Conference on Broadband Network & Multimedia Technology IC-BNMT '09.*, vol. 2, pp. 575-579, Oct 2009.

[27] Peng XiWei, Huang Wenming, Wen Peizhi, and Wu Xiaojun, "Simplification of scattered point cloud based on feature extraction," *International Conference on Genetic and Evolutionary Computing, WGEC '09.*, vol. 3, pp. 335-338, Oct 2009.

[28] S.-B. Park and S.-U. Lee, "Multiscale representation and compression of 3-d point data," *Trans. Multi. 11 (1)*, pp. 177–182, 2009.

[29] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing 41*, pp. 3445–3462, 1993.

[30] J. Smith, G. Petrova, and S. Schaefer, "Progressive encoding and compression of surfaces generated from point cloud data," *Computers & Graphics ISSN 0097-8493*, vol. 36, pp. 341-348, August 2012.

[31] J. Manson and S. Schaefer, "Wavelet rasterization," *Computer Graphics Forum 30*, pp. 395–404, 2011.

[32] F. Wheeler. (1996) Adaptive arithmetic coding source code. [Online]. http://www.cipr.rpi.edu/~wheeler/ac

[33] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891-923, Nov 1998.

[34] Sunil Arya and David M. Mount, "Algorithms for Fast Vector Quantization," *Data Compression Conference (ISBN 0-8186-3392-1)*, pp. 381 - 390, 1993.

[35] K. Fukunaga and Patrenahalli M. Narendra, "A Branch and Bound Algorithm for Computing k-Nearest Neighbors," *IEEE Transactions on Computers*, vol. C-24, no. 7,

pp. 750 - 753, 1975.

[36] A. Nuchter, K. Lingemann, and J. Hertzberg, "Cached k-d tree search for ICP algorithms," *Sixth International Conference on 3-D Digital Imaging and Modeling*, pp. 419 - 426, 2007.

[37] David G. Lowe and Marius Muja, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," *International Conference on Computer Vision Theory and Application VISSAPP'09*, pp. 331-340, 2009.

[38] David G. Lowe and Marius Muja, "Fast Matching of Binary Features," *Conference on Computer and Robot Vision (CRV)*, vol. 9, pp. 404 - 410, May 2012.

[39] David G. Lowe and Marius Muja, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227 - 2240, May 2014.

[40] David G. Lowe and Marius Muja. Fast Library for Approximate Nearest Neighbors. [Online]. http://www.cs.ubc.ca/research/flann/

[41] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss, "Comparison of surface normal estimation methods for range sensing applications," *IEEE International Conference on Robotics and Automation*, pp. 3206 - 3211, May 2009.

[42] Ming Liu, F. Pomerleau, Francis Colas, and Roland Siegwart, "Normal estimation for pointcloud using GPU based sparse tensor voting," *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 91 - 96, Dec 2012.

[43] H. Badino, D. Huber, Y. Park, and T. Kanade, "Fast and accurate computation of surface normals from range images," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3084 - 3091, May 2011.

[44] Jens Berkmann and Terry Caelli, "Computation of surface geometry and segmentation using covariance techniques," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 11, pp. 1114 - 1116, Nov 1994.

[45] Kai Hormann, Sun-Jeong Kim, and David Levin Nira Dyn, "Optimizing 3D triangulations using discrete curvature analysis," *Mathematical Methods for Curves and Surfaces (Oslo 2000)*, pp. 135–146, 2001.

[46] Jan J. Koenderink and Andrea J. van Doorn, "Surface shape and curvature scales. Image and Vision Computing, 10(8):557–565, 1992".

[47] Mark Pauly, Richard Keiser, and Markus Gross, "Multi-scale Feature Extraction on Point-Sampled Surfaces," *Computer Graphics Forum*, vol. 22, no. 3, pp. 281–289, Sep 2003.

[48] Radu Bogdan Rusu, Nico Blodow, Zoltan Marton, Alina Soos, and Michael Beetz, "Towards 3D Object Maps for Autonomous Household Robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927-941, Nov 2008.

[49] Zhengyou Zhang, "Iterative Point Matching for Registration of Free-Form Curves and Surfaces," *International Journal of Computer Vision*, vol. 13, no. 2, pp. 119-152, Oct 1994.

[50] Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*, 2nd ed.: Wiley-Interscience, 2000.

[51] A. Martin Fischler and C. Robert Bolles, "cartography, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated," *Communications of the ACM, 24(6):381–395, June 1981.*

[52] T. Rabbani, F. A. Van Den Heuvel, and G. Vosselman, "Segmentation Of Point Clouds Using Smoothness Constraint," *ISPRS Commission V Symposium 'Image Engineering and Vision Metrology'*, 2006.

[53] A. Hoover et al., "An experimental comparison of range image segmentation algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 673-689, Jul 1996.

[54] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Emanuel Dolha, and Michael Beetz, "Functional Object Mapping of Kitchen Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3525-3532, Sept 2008.

[55] Sergey Ushakov. (2013, Jul) Point cloud Library (PCL), documentacion, tutorial in Region Growing Segmentation. [Online]. http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php

[56] Ranjith Unnikrishnan, Nicolas Vandapel, and Martial Hebert. Jean-Francois Lalonde, "Scale Selection for Classification of Point-sampled 3-D Surfaces.," *In Proceedings of the 5th International Conference on 3-D Digital Imaging and Modeling (3DIM), Ottawa, Ontario, Canada, June 13-16 2005.*, pp. 285–292.

[57] Niloy J. Mitra and An Nguyen, "Estimating surface normals in noisy point cloud data.," *In Proceedings of The 19th ACM Symposium on Computational Geometry (SCG),*

San Diego, California, USA, June 8-10 2003., pp. 322–328.

[58] PCL. Point Cloud Library API Documentation. [Online]. http://pointclouds.org/documentation/

[59] Rosario García Viedma, *Percepción, Atención y Memoria*. España: Universidad de Jaén.

[60] Enric Munar, Jaume Rosselló, and Antonio Sánchez Cabaco, *Atención y Percepción*. Madrid, España: Alianza Editorial, 1999.

[61] E. Bruce Goldstein, *Sensación y percepción 6ª Edición*.: Paraninfo, 2006.

[62] Juan Mayor Sánchez, José Luis Pinillos, Soledad Ballesteros Jiménez, José Luis Fernández Trespalacios, and Pío Tudela Garmendia, *Tratado de Psicología General. Vol. 3. Atención y Percepción*. Madrid, España: Alhambra, 1992.

[63] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin, "The Ball-Pivoting Algorithm for Surface Reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349-359, Oct 1999.

[64] Suleman Surti, Joel S. Karp, Lucretiu M. Popescu, Margaret E. Daube-Witherspoon, and Matthew Werner, "Investigation of Time-of-Flight Benefit for Fully 3-D PET," *IEEE TRANSACTIONS ON MEDICAL IMAGING*, vol. 25, pp. 529-538, May 2006.

[65] Josep Forest Collado, *Tesi Doctoral: New methods for triangulation-based shape acquisition using laser scanners*. Girona, España: Universitat de Girona. Departament d'Electrònica, Informàtica i Automàtica, 2004.

[66] Akash Malhotra, Kunal Gupta, and Kamal Kant, "Laser Triangulation for 3D Profiling of Target," *International Journal of Computer Applications*, vol. 35, no. 8, pp. 47-50, December 2011.

[67] Lixia Rong, Weihai Chen, Shouqian Yu, and Weiyang Song, "Mathematical model of coordinate transformations for 3D Depth-of-field collection system," *IEEE International Conference on Industrial Informatics*, vol. 6, pp. 80-85, July 2008.

[68] KONICA MINOLTA, NON-CONTACT 3D DIGITIZER VIVID 9i/VI-9i INSTRUCTION MANUAL.

[69] Fausto Bernardini and Holly Rushmeier, "The 3D Model Acquisition Pipeline," *Computer Graphics Forum*, vol. 21, pp. 149–172, 2002.

[70] R.M. Bolle and B.C. Vemuri, "On Three-Dimensional Surface Reconstruction Methods," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 1, pp. 1-13, Jan 1991.

[71] R. Mencl and H. Müller, "Interpolation and Approximation of Surfaces from Three-Dimensional Scattered Data Points," *Scientific Visualization Conference*, Jun 1997.

[72] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*.: Cambridge University Press, 2004.

[73] Fausto Bernardini and Holly Rushmeier, "The 3D Model Acquisition Pipeline," *Computer Graphics Forum*, vol. 21, pp. 149–172, Jun 2002.

[74] Roland E. Larson, Robert P. Hostetler, and Bruce H. Edwards, *Cálculo y Geometría Analitica*, Sexta ed.: Mc Graw Hill, 1999, vol. 2.

[75] Stjepan Jecić and Nenad Drvar, "The assessment of structured light and laser scanning methods in 3D shape measurements," *Proceedings of the 4 th International Congress of Croatian Society of Mechanics*, pp. 237--244, 2003.

[76] Christian Teutsch, *Model-based Analysis and Evaluation of Point Sets from Optical 3D Laser Scanners*. Aachen, Germany: Shaker Verlag GmbH, 2007.

[77] Nie Jianhui, Hu Ying, Ma Zi, and Li Qingmeng, "Online Colored Point Cloud Acquisition by Reprojection," *International Conference on Intelligent System Design and Engineering Application*, vol. 2, pp. 648 - 652, Jan 2012.

[78] Jing Wang, Juan Zhang, and Qingtong Xu, "Research on 3D laser scanning technology based on point cloud data acquisition," *International Conference on Audio, Language and Image Processing*, pp. 631 - 634, 2014.

[79] Markus Gross, and Leif Kobbelt Mark Pauly, "Efficient simplification of point-sampled surfaces," *In Proceedings of IEEE Visualization, Boston, MA, USA*, October 27 - November 01 2002.