

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



## BÁO CÁO

### SURFACE RECONSTRUCTION FROM POINT CLOUD

**Môn : Đồ họa máy tính**

**Nhóm HTCv \_ K15**

**Thành Viên :**

Lâm Khả Hân – 1512151

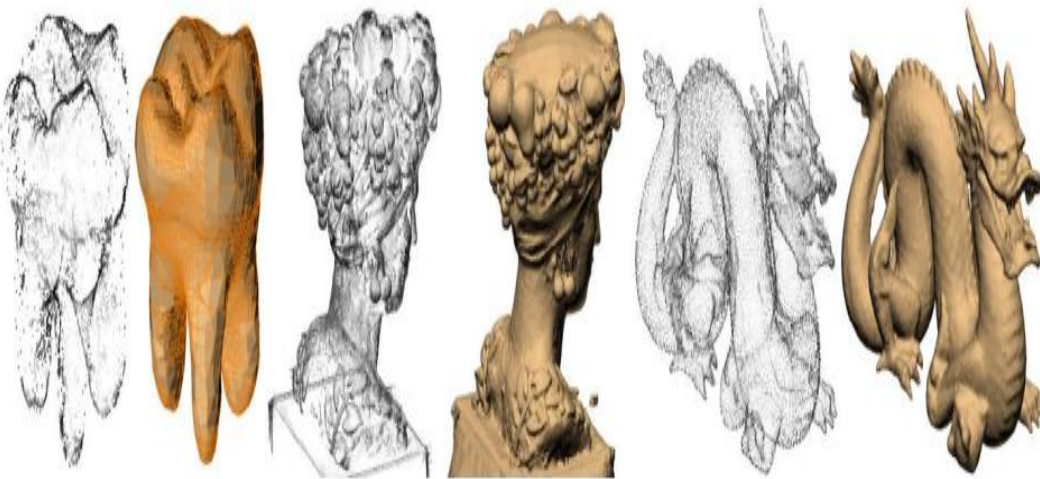
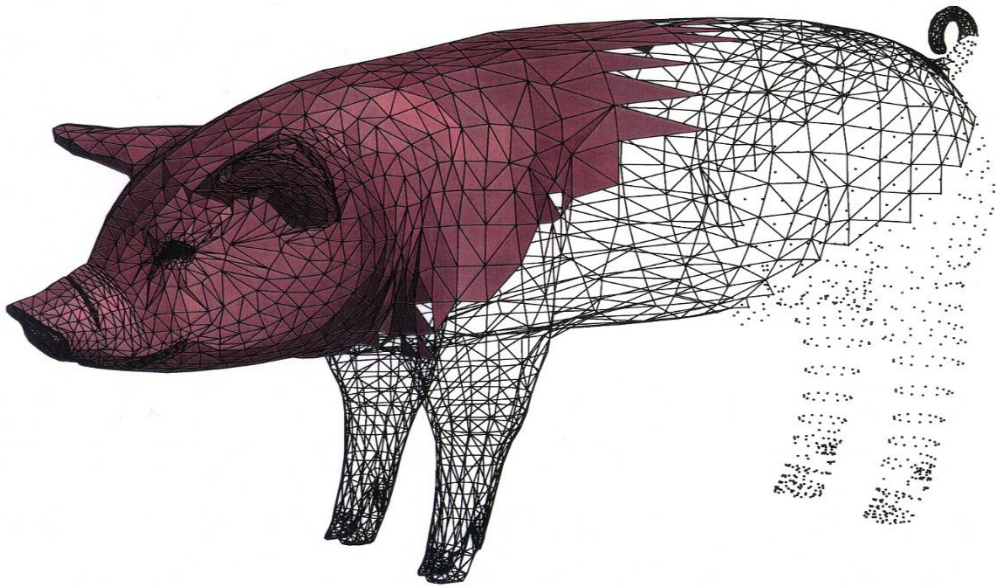
Phạm Thị Thanh Hoài – 1512172

Đào Minh Toàn - 1512581

Nguyễn Sinh Tú - 1512647

Nguyễn Việt Trung - 1212454

**Giáo viên hướng dẫn : Lý Quốc Ngọc**



## TABLE OF CONTENTS

I.	Mở đầu .....	4
II.	Giới thiệu .....	5
III.	Vấn đề .....	6
IV.	Những thuật ngữ quan trọng.....	6
V.	Công cụ và thư viện.....	7
1.	Giới thiệu các công cụ và thư viện sử dụng trong đồ án.....	7
2.	Cài đặt và thiết lập môi trường làm việc.....	8
VI.	Tổng quan các bước tái tạo bề mặt .....	10
VII.	GREEDY PROJECTION ALGORITHM .....	11
VIII.	NEAREST NEIGHBOR SEARCH.....	15
1.	KdTree.....	17
2.	Constructor .....	18
3.	Adding elements.....	19
4.	Removing elements.....	20
5.	Balancing.....	20
6.	Search .....	20
7.	Complexity.....	22
IX.	Normal estimation(ước lượng pháp tuyến) .....	22
X.	Các giai đoạn của đồ án.....	24
XI.	Poisson.....	28
1.	Giới thiệu .....	28
2.	Công việc liên quan .....	29
3.	Tiếp cận poisson reconstruction algorithm.....	29
4.	Thực hiện .....	30
4.1	. Vấn đề Discretization.....	30
4.2	. Định nghĩa trường vector .....	31
4.3	Giải pháp poisson.....	31
4.4	Chiết xuất Isosurface .....	32
4.5	Các mẫu không đồng nhất.....	33
5.	Kết Quả .....	34
5.1	Giải quyết.....	34
5.2	So sánh với công việc trước đây.....	35

5.3 Hiệu quả và khả năng mở rộng.....	37
6. Kết Luận .....	39
XIV. Tài liệu tham khảo.....	40

## I. Mở đầu

Những tiến bộ gần đây trong kỹ thuật thu thập ba chiều cung cấp dữ liệu đầu vào rất lớn cho bộ dữ liệu điểm ba chiều chưa được tổ chức. Với sự tăng lên của các ứng dụng sử dụng các đám mây điểm, nhu cầu ngày càng tăng để tái tạo lại một biểu diễn bề mặt liên tục cung cấp một sự biểu diễn đích thực của các tập điểm không tổ chức và làm cho bề mặt và render bề mặt(surface) cho visualization.

Mục tiêu chính của đề án này là nghiên cứu các thuật toán tái tạo bề mặt và tạo ra mô hình 3D của đối tượng từ đám mây điểm. Các đám mây điểm được sử dụng trong Đề án là những point cloud đã được thu thập trước đó. Trong đề án này chúng tôi sử dụng thư

viện PCL(point cloud library), PCL cung cấp nhiều thuật toán cho việc tái tạo như fast greedy triangle projection, Marching cubes, Grid Projection, poisson. Trong đồ án này chúng tôi sẽ tập trung vào thuật toán Greedy triangle projection, một thuật toán có thời gian chạy khá nhanh. Tuy output có thể không tốt như những thuật khác nhưng rất thích hợp khi làm việc với dữ liệu đám mây điểm lớn. bên cạnh đó có cài đặt thêm các thuật toán khác(không đi vào chi tiết) để so sánh độ hiệu quả của thuật toán.

## II. Giới thiệu

Tái tạo bề mặt là một vấn đề đầy thách thức trong lĩnh vực đồ họa máy tính với nhiều lĩnh vực ứng dụng như hình ảnh y học, thực tế ảo, trò chơi điện tử, phim ảnh, thương mại điện tử và các ứng dụng đồ họa khác. Các đám mây điểm không tổ chức bắt nguồn từ các đầu vào khác nhau như dữ liệu máy quét laze, các phép đo hình ảnh bằng hình ảnh, cảm biến chuyển động ... tạo ra một vấn đề khó khăn về tái thiết, không hoàn toàn được giải quyết và đầy thử thách trong trường hợp dữ liệu không đầy đủ, lộn xộn và thưa thớt.

Việc xây dựng lại các mô hình có thể đáp ứng nhu cầu mô hình hoá và có tình hình dung cao của các ứng dụng như vậy là một vấn đề không thể thiếu mà chưa được giải quyết hoàn toàn bởi vì dữ liệu đầu vào chưa đầy đủ, lộn xộn và thưa thớt. Các điểm đầu vào thô thường không được tổ chức, thiếu cấu trúc cố hữu hoặc thông tin định hướng.

Việc tái thiết bề mặt từ dữ liệu hình học thô đã nhận được sự chú ý ngày càng tăng nhờ vào các bộ cảm biến hình học mở rộng (ví dụ như Microsoft Kinect), máy quét dữ liệu laser và các thuật toán computer vision cung cấp ít hoặc không có các thuộc tính đáng tin cậy. Sự hiện diện của tiếng lộn xộn và dữ liệu bị nhiễu là không thể tránh khỏi làm cho thách thức thậm chí còn lớn hơn và bất kỳ sự tiến bộ nào theo hướng này cũng có thể đem lại lợi ích trực tiếp cho tái thiết từ dữ liệu đầu vào điểm hoặc các đám mây điểm có các thuộc tính.

Mục đích là để tạo ra mô hình của một đối tượng phù hợp nhất với thực tế. Lưới đại giác là sự biểu hiện đồ họa được chấp nhận rộng rãi, với sự hỗ trợ rộng rãi từ phần mềm và phần cứng hiện tại.

### III. Vấn đề

Mục tiêu của việc tái tạo bề mặt là xác định một bề mặt  $S$  từ một tập hợp các điểm  $P$ , lấy mẫu từ một bề mặt trong  $R^3$  (không gian 3 chiều) sao cho các điểm của tập  $P$  nằm trên  $S$ . Mặt  $S$  xấp xỉ với tập các điểm  $P$

Theo quan điểm toán học, một bề mặt trong không gian ba chiều Euclide  $R^3$  được định nghĩa như là một đa dạng các đối tượng hai chiều nhỏ gọn, kết nối và chứa thông tin về định hướng mặt. Nói cách khác, chúng ta có thể nói rằng một bề mặt là một "liên tục" tập hợp các điểm trong  $R^3$  là hai chiều cục bộ

Một bề mặt có thể có một đường biên, đường biên này có thể trống, hoặc nó có thể bị đóng, khi ranh giới trống. Vấn đề xây dựng lại bề mặt có thể được công thức hoá như sau:

Cho một tập hợp điểm  $S = \{P_i = (X_i, Y_i, Z_i) / (X_i, Y_i, Z_i) \in M \subset R^3, i = 1..k, M \text{ surface in } R^3\}$

Tìm một bề mặt  $M'$  mà nội suy hoặc xấp xỉ  $M$ , sử dụng tập dữ liệu  $S$ .

### IV. Những thuật ngữ quan trọng

*Modeling (toán học)* : Khởi tạo và thực hiện trên máy tính của một đối tượng, bằng cách xác định các điểm trong một mảng 3 chiều. Mảng này dựa trên các trục  $X$ ,  $Y$  và  $Z$  của không gian hình học. Sau đó, các bộ khác nhau của các điểm tạo thành các đường, các

đường để tạo đa giác và các đa giác tham gia để tạo ra đối tượng. Kết quả đơn giản nhất thường được hiển thị dưới dạng mô hình wireframe.

*Rendering*: được gọi là bản vẽ thực sự của các đối tượng 3D sử dụng công nghệ máy tính. Để tạo ra một đối tượng, các thuộc tính nhất định như mờ(transparency),(colour)màu sắc, khuếch tán hoặc phản xạ và khúc xạ phản chiếu phải được gán cho nó và môi trường xung quanh.

*Mesh*: đó là một tập hợp các mặt tiếp giáp, không chồng chéo (hoặc tứ giác) nối tiếp nhau dọc theo các cạnh của chúng. Một lưới chứa các đỉnh, cạnh và mặt và cách biểu diễn đơn giản nhất của nó là một mặt duy nhất. Đôi khi nó còn được gọi là TIN, Triangulated Irregular Network

Bề mặt: một khối 2D hoặc 3D nhỏ gọn kết nối, định hướng, có thể có ranh giới(boundary). Một bề mặt không có ranh giới là một bề mặt khép kín. Một bề mặt có thể được biểu diễn theo hình học dưới không tường minh ( $F(x, y, z) = 0$ ) hoặc dạng tham số.

## V. Công cụ và thư viện

### 1. Giới thiệu các công cụ và thư viện sử dụng trong đồ án

Open GL: Thư viện OpenGraphics là ngôn ngữ chéo (cross-language), giao diện lập trình ứng dụng đa nền tảng (API) cho đồ họa máy tính 2D và 3D. Giao diện lập trình đồ họa 3D ban đầu được thiết kế bởi Silicon Graphics Inc.

Eigen: Eigen là thư viện C++ cấp cao thực hiện các tính toán liên quan đến đại số tuyến tính và các phép toán ma trận liên quan.

Boost : là tập hợp các thư viện ngôn ngữ lập trình C++ hỗ trợ các tác vụ và cấu trúc như đại số tuyến tính, tạo số giả ngẫu nhiên, đa luồng, xử lý hình ảnh, biểu thức thông thường và unit testing

FLANN : là một thư viện để thực hiện tìm kiếm xấp xỉ gần nhất gần trong không gian nhiều chiều. Nó chứa một tập hợp các thuật toán chúng tôi tìm thấy để làm việc tốt nhất cho tìm kiếm láng giềng gần nhất(nearest neighbor search) và một hệ thống tự động lựa chọn thuật toán tốt nhất và các tham số tối ưu tùy thuộc vào tập dữ liệu

VTK : Hỗ trợ thu thập dữ liệu, hiển thị, ước lượng thể tích vật thể.

PCL: Thư viện Point Cloud (PCL) là một dự án mở, độc lập, quy mô lớn dành cho xử lý hình ảnh 2D / 3D và điểm mây. Nó thực hiện một tập hợp các thuật toán được thiết kế để giúp làm việc với dữ liệu 3 chiều, trong những đám mây điểm đặc biệt.

CMake : là nền tảng miễn phí, mã nguồn mở giúp xây dựng project cho cho các môi trường làm việc khác nhau, ở đây chúng tôi sử dụng CMake để tạo project làm việc trên visual studio 2015

Qt : môi trường hỗ trợ lập trình giao diện

## 2. Cài đặt và thiết lập môi trường làm việc

Tải và cài đặt PCL ALL in One (PCL + Boost + Qhull + FLANN + OpenNI + VTK + Eigen) , Qt tại đây : <http://unanancyowen.com/en/pcl18/>, lưu ý là chọn đúng phiên bản phù hợp với phiên bản visual studio đang sử dụng



#### META

Log in

Entries [RSS](#)

Comments [RSS](#)

WordPress.org

now.

#### Visual Studio 2015

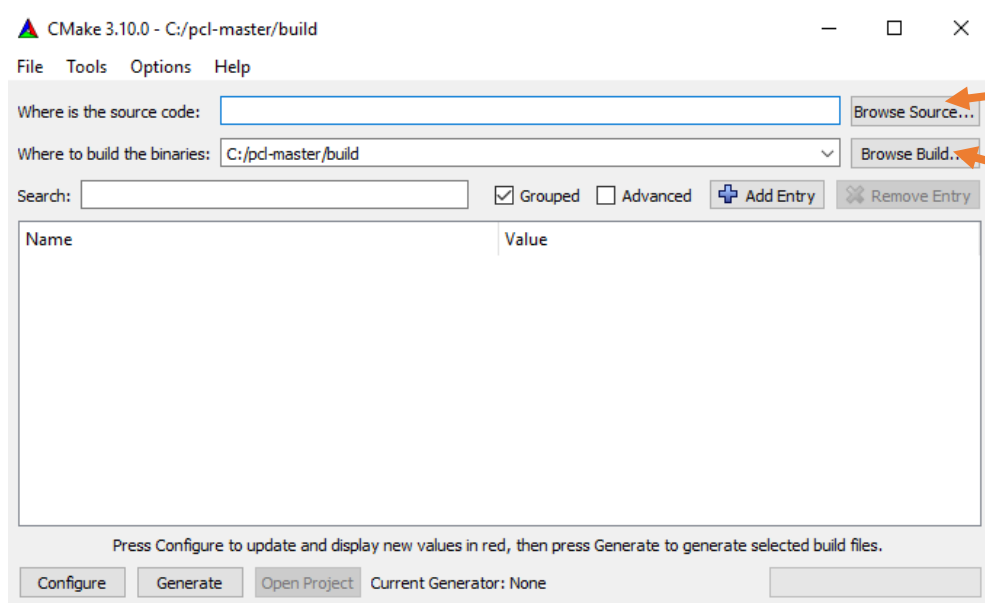
- **PCL 1.8.0 All-in-one Installer MSVC2015 Win32**
- PCL 1.8.0 PDB MSVC2015 Win32 (symbol files)
- **PCL 1.8.0 All-in-one Installer MSVC2015 x64**
- PCL 1.8.0 PDB MSVC2015 x64 (symbol files)

#### Visual Studio 2013

- **PCL 1.8.0 All-in-one Installer MSVC2013 Win32**
- PCL 1.8.0 PDB MSVC2013 Win32 (symbol files)
- **PCL 1.8.0 All-in-one Installer MSVC2013 x64**
- PCL 1.8.0 PDB MSVC2013 x64 (symbol files)

Tải Cmake : <https://cmake.org/download/>

Sử dụng CMake để tạo project



Đường dẫn chứa  
file \*.cpp và  
CMakeList.txt

Đường dẫn nơi  
project sẽ được  
tạo ra

Sau khi cấu hình xong đường dẫn ta ấn Configure để CMake tìm các thư viện liên quan

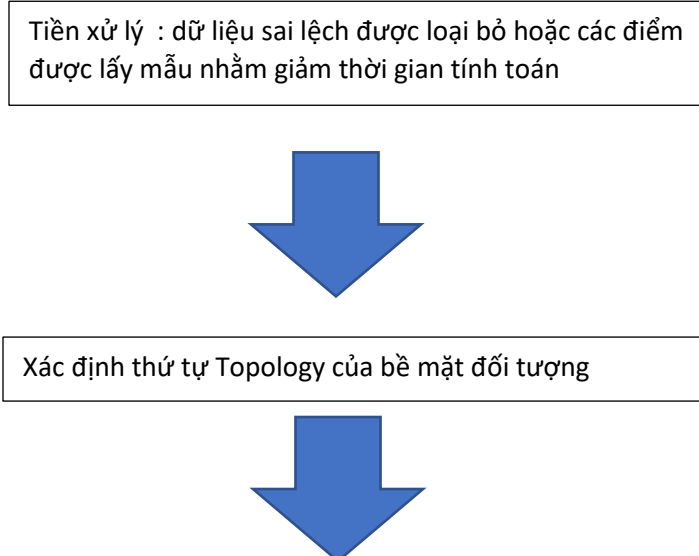
Cấu trúc một CMakeList như sau :

Sau khi tạo được project ta mở visual studio và tìm file \*.sln mở lên và thực hiện build chương trình

## VI. Tổng quan các bước tái tạo bề mặt

Các tài liệu nghiên cứu về tái thiết bề mặt là rất lớn và có nhiều thuật toán và cách tiếp cận khác nhau cho vấn đề.

Các bước chính liên quan đến bất kỳ thuật toán nào đều tuân theo mô hình dưới đây:



Tạo ra các bề mặt đa giác : lưới Tam giác(hoặc Tứ diện) được tạo ra đáp ứng những yêu cầu nhất định vd : giới hạn về kích thước mặt lưới, không có giao điểm của đường nét vv.v



Hậu xử lý : khi mô hình được tạo ra, các hoạt động chỉnh sửa là thường được sử dụng để tinh chỉnh và hoàn thiện bề mặt đa giác

## VII. GREEDY PROJECTION ALGORITHM

Thuật toán hoạt động bằng cách duy trì 1 list các điểm mà các điểm đó được kết nối bằng lưới các đường viền, sau đó mở rộng dần cho đến khi tất cả các điểm có thể kết nối với nhau. Việc kết nối tất cả các điểm đó dựa vào thuật tìm kiếm tham lam tất cả các điểm lân cận. Thuật toán có thể xử lý đối với các điểm không cần sắp xếp từ 1 hoặc nhiều lần quét, hoặc với các phần kết nối với nhau. Nó sẽ hoạt động tốt nhất nếu bề mặt phẳng và sự chuyển tiếp giữa các phần khác nhau có mật độ các điểm cao.

Thuật toán dựa trên nguyên lý tăng trưởng bề mặt gia tăng (Mercl, 1998), theo cách tiếp cận loại tham lam. Thuật toán bắt đầu bằng cách tạo ra một tam giác và tiếp tục thêm các hình tam giác mới cho đến khi tất cả các điểm trong đám mây dùng để tạo thành tam giác hoặc không có hình tam giác hợp lệ nào có thể được kết nối với lưới kết quả.

1. Tìm kiếm hàng xóm gần nhất(Nearest neighbor search) : Với mỗi điểm 'p' trong đám mây điểm, một vùng k- neighborhood được chọn. Vùng này được tạo ra bằng cách tìm kiếm các điểm lân cận của điểm chuẩn gần nhất trong phạm vi bán kính r. Bán kính được định nghĩa là  $\mu \cdot d$ , trong đó d là khoảng cách của điểm p từ hàng xóm gần nhất của nó và  $\mu$  là hằng số do người dùng xác định để tính đến mật độ đám mây điểm. (sẽ đề cập sau) Để tìm những người hàng xóm gần nhất cho điểm nhất định trong đám mây điểm, Cấu trúc Kdtree nearest neighbor search được sử dụng.

Các điểm trong đám mây được gán các trạng thái khác nhau tùy thuộc vào sự tương tác của chúng với thuật toán: *free*, *fringe*, *boundary*, and *completed*.

- a. Ban đầu, tất cả các điểm trong đám mây đều ở trạng thái tự do và các điểm tự do được định nghĩa là những điểm không có tam giác tạo thành.
  - b. Khi tất cả các tam giác tới của một điểm đã được xác định, điểm được gọi là trạng thái hoàn thành.
  - c. Khi một điểm đã được chọn làm điểm tham chiếu nhưng có một số tam giác bị thiếu do tham số góc tối đa cho phép, nó được gọi là một điểm ranh giới.
  - d. Điểm Fringe là những điểm chưa được chọn làm điểm tham chiếu
2. Phép chiếu vùng lân cận(Neighborhood projection ) sử dụng các mặt phẳng tiếp tuyến: vùng lân cận được chiếu lên trên một mặt phẳng xấp xỉ tiếp tuyến bề mặt được hình thành bởi vùng lân cận và được đặt xung quanh điểm p.
  3. Tỉa hình : Các điểm được cắt tỉa theo tiêu chí về tầm nhìn và khoảng cách tiêu chuẩn, và kết nối với p và các điểm liên tiếp bằng các cạnh, tạo hình tam giác có tiêu chuẩn góc tối đa(maximum angle criterion – người lập trình tự đặt những thông số này) và tiêu chuẩn góc tối thiểu tùy chọn. Các điểm trong đám mây điểm được tỉa theo nhiều tiêu chí.

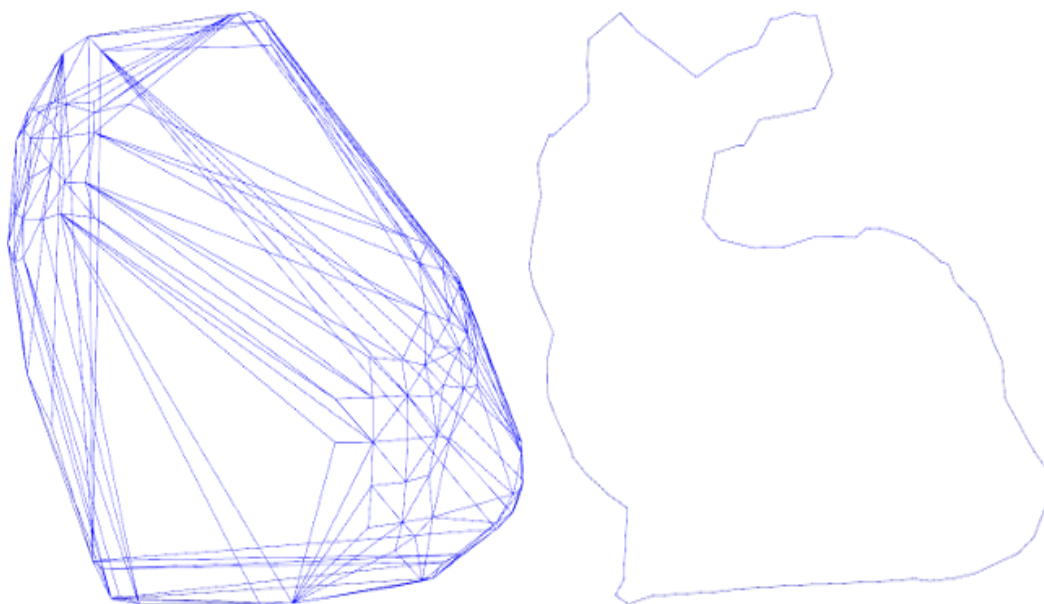
- a. Việc cắt tỉa theo tiêu chí khoảng cách: một tiêu chuẩn khoảng cách được áp dụng để tỉa xuống tìm các điểm lân cận trong khoảng cách không gian của điểm tham chiếu hiện tại sử dụng kd-tree
- b. Các điểm xa hơn nằm ngoài phạm vi ảnh hưởng trung tâm tại điểm tham chiếu bị loại. Các điểm được chọn được gọi là điểm ứng cử viên.
- c. Lựa chọn mặt phẳng chiếu: tập hợp các điểm thu được sau khi áp dụng các tiêu chuẩn khoảng cách được chiếu lên trên mặt phẳng tiếp tuyến xấp xỉ.
- d. Angle ordering : một hệ tọa độ cục bộ mới được định nghĩa với điểm tham chiếu là gốc và mặt phẳng chiếu của bước trước phục vụ như là mặt phẳng xy. Tất cả các điểm trong tập điểm ứng viên được chiếu trên mặt phẳng này. Thứ tự xung quanh điểm tham chiếu dựa trên góc ( $\Theta$ ) giữa trục x của hệ tọa độ cục bộ và vector từ gốc đến điểm ứng viên dự kiến
- e. Visibility : các điểm có khả năng tạo thành một mắt lưới tự cắt nhau sẽ bị loại bỏ. Thuật toán xác định hai loại cạnh để kiểm tra điều kiện này:
  - Boundary Edge: một cạnh với một hình tam giác chỉ xảy ra trên đó. Các cạnh kết nối với cạnh rìa và / hoặc các điểm ranh giới
  - Internal Edge: kết nối các điểm đã hoàn thành (các điểm đã tạo thành lưới) với bất kỳ điểm nào khác
- f. Các hàm set giá trị trong PCL

*setMu(double)* : cung cấp kích thước đối với 1 vùng các điểm lân cận. Từ đó xác định có bao nhiêu điểm lân cận được tìm thấy.

- *setMaximumNearestNeighbors(unsigned)* xác định số lượng tối đa các điểm lân cận nhằm điều chỉnh mật độ.
- *setSearchRadius(double)*: Dùng để xác định các điểm lân cận theo 1 tham số Radius, từ tham số Radius đó, t dùng 1 hình cầu để xác định xem đó có phải là điểm lân cận không. Tham số này người dùng có thể đặt ra.
- *setMinimumAngle(double)* and *setMaximumAngle(double)*: là góc nhỏ nhất và lớn nhất trong mỗi tam giác. Nếu yếu tố thứ nhất không được

*bảo đảm, yếu tố thứ hai cũng vậy. Thông thường các giá trị nằm trong khoảng 10 đến 120 độ*

- *setMaximumSurfaceAngle(double) and setNormalConsistency(bool): trong trường hợp hình dạng của các tam giác hoặc các góc nơi mà 2 bên bề mặt rất gần sát nhau. Trong trường hợp này, các điểm sẽ không kết nối với các điểm nằm ngoài góc quy định. Các góc này được tính bằng cách ước lượng nhằm đảm bảo tính nhất quán.*



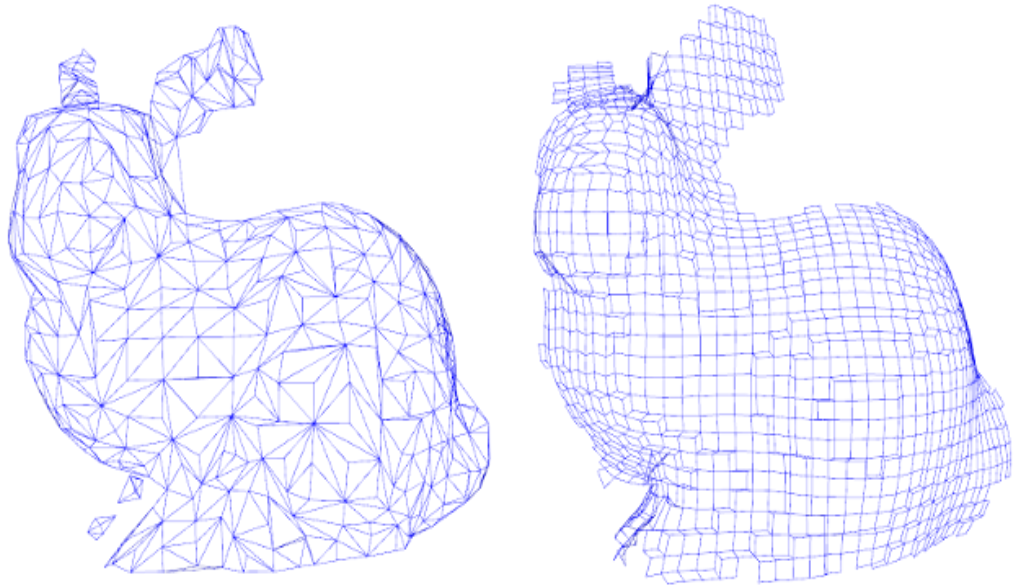
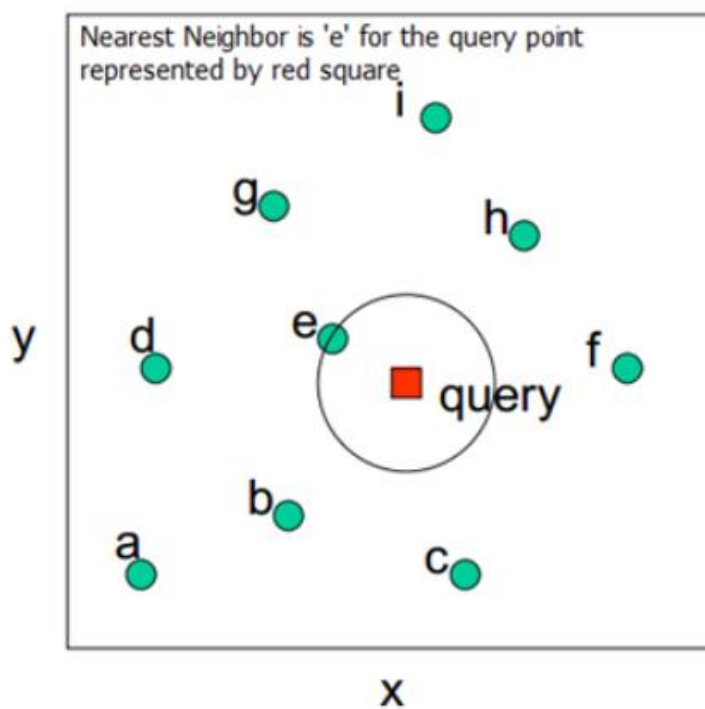
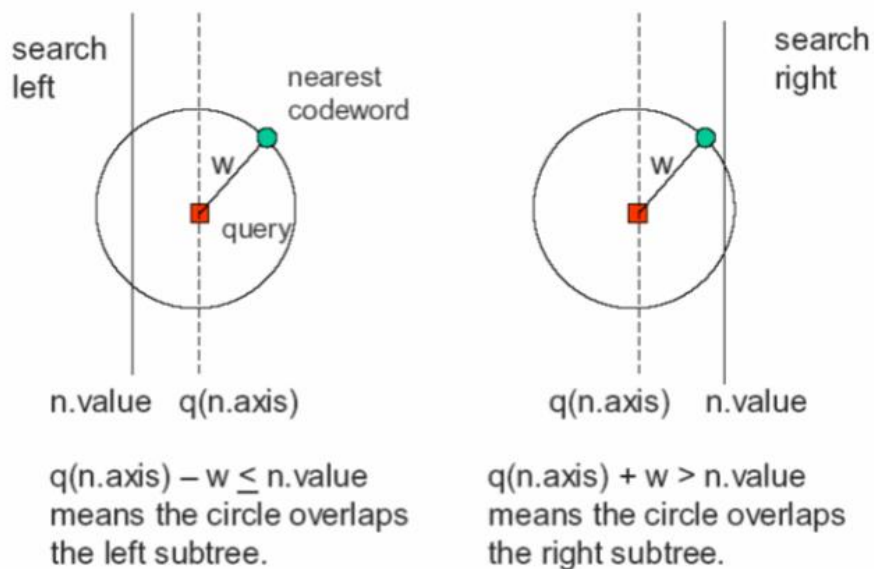


Figure 1: a: triangle mesh, b : cube mesh

## VIII. NEAREST NEIGHBOR SEARCH

*Vấn đề Tìm kiếm Láng giềng Gần nhất : Cho một tập  $S$  của các điểm trong không gian  $n$ -chiều, xây dựng một cấu trúc dữ liệu đưa ra bất kỳ điểm truy vấn  $Q$ , tìm điểm trong tập  $S$  với khoảng cách nhỏ nhất đến  $Q$ .*



Cách tiếp cận đơn giản nhất để tìm hàng xóm gần nhất đến điểm  $Q$  là tính khoảng cách từ  $Q$  tới mỗi điểm trong  $P$ . Cách tiếp cận là quét tất cả các điểm xung quanh. Cách này

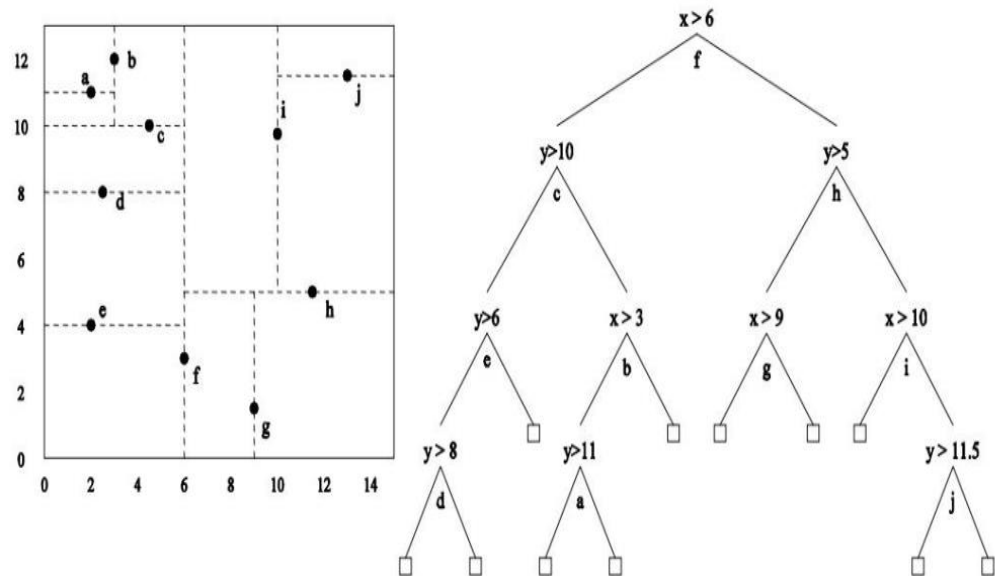
*có thể chấp nhận được đối với các bộ dữ liệu nhỏ nhưng đối với các tập hợp dữ liệu lớn thì rất tốn kém. Vì vậy, chúng tôi sử dụng Kd-Trees để giải quyết vấn đề này.*

*Điểm mấu chốt của cấu trúc này là tập dữ liệu  $S$  được coi là cố định. Điểm truy vấn(query point) có thể thay đổi theo yêu cầu, nhưng  $S$  vẫn không thay đổi.*

## **1. KdTree**

k-d tree là một binary tree trong đó mỗi node là một điểm k chiều.

Mỗi nút không phải là lá có thể nghĩ như một cách ngầm tạo ra một siêu phẳng phân chia không gian thành hai phần, được hiểu như nửa không gian. Các điểm phía bên trái của siêu mặt phẳng này được đại diện bởi cây con trái của node và điểm bên phải của siêu phẳng được đại diện cây con bên phải. Khuynh hướng siêu phẳng được chọn theo cách sau: một node của cây liên kết với k chiều, với siêu phẳng vuông góc trục. Vì vậy, cho ví dụ, Nếu cho phân vùng phân chia theo trục "x" được chọn, tất cả các điểm trong cây con nhỏ hơn giá trị "x" thì node sẽ xuất hiện trong cây con trái và tất cả các điểm lớn hơn giá trị x sẽ nằm trong cây con phải. Trong trường hợp như vậy, siêu phẳng sẽ set bằng giá trị x của điểm và thường nó sẽ là trục x.



Một cây kd-tree tương tự như cây quyết định, ngoại trừ việc chúng ta chia bằng cách sử dụng giá trị trung vị theo chiều có độ biến thiên cao nhất. Mỗi nút nội bộ lưu trữ một điểm dữ liệu, và các nút lá rỗng.

kd-tree cho phép thực hiện tìm kiếm hiệu quả với các câu phát biểu vấn đề như "tìm k-hàng xóm gần nhất của Q" hoặc "tìm tất cả các điểm ở khoảng cách thấp hơn R từ Q"

## 2. Constructor

Các method kinh điển để xây dựng cây k chiều có những ràng buộc sau:

- Khi di chuyển xuống cây, một chu kỳ qua các trục sử dụng để chọn phân tách không gian.
- Điểm được thêm vào bằng cách chọn median của điểm đặt vào cây con, đối với các tọa độ của chúng trong trục được sử dụng để tạo phân tách không gian.

Cho một danh sách  $n$  điểm, theo dõi thuật toán sử dụng median-finding sort để xây dựng cây  $k$  chiều cân bằng chứa  $n$  điểm đó.

```
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtree
    node.location := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node;
}
```

### 3. Adding elements

Thêm điểm mới vào  $k$ -d tree theo cách tương tự như cách thêm một phần tử vào cây tìm kiếm khác. Thứ nhất, đi qua cây, bắt đầu từ root và di chuyển xuống cây trái hoặc phải tùy thuộc vào việc có nên chèn điểm vào bên trái hay bên phải của phân tách không gian. Một khi lấy một nút con dưới cần được đặt, thêm điểm mới như con trái hoặc phải của node lá, một lần nữa tùy thuộc vào phía bên nào của phân tách không gian chứa node mới.

Thêm điểm theo cách này có thể làm cho cây mất cân bằng, dẫn đến hiệu suất giảm. Tốc độ giảm hiệu suất tùy thuộc vào sự phân bố không gian của các điểm đang được thêm vào và số điểm được thêm vào liên quan đến kích thước cây.

Nếu cây trở nên mất cân bằng, cần phải cân bằng lại để khôi phục lại các truy vấn dựa vào việc cân bằng cây, chẳng hạn như tìm kiếm hàng xóm gần nhất.

#### 4. Removing elements

Remove một điểm từ cây k chiều đã tồn tại, mà không vi phạm bất biến, cách đơn giản nhất là tạo thành tập hợp tất cả các nút và bỏ từ con của node đích, và tạo lại phần đó của cây.

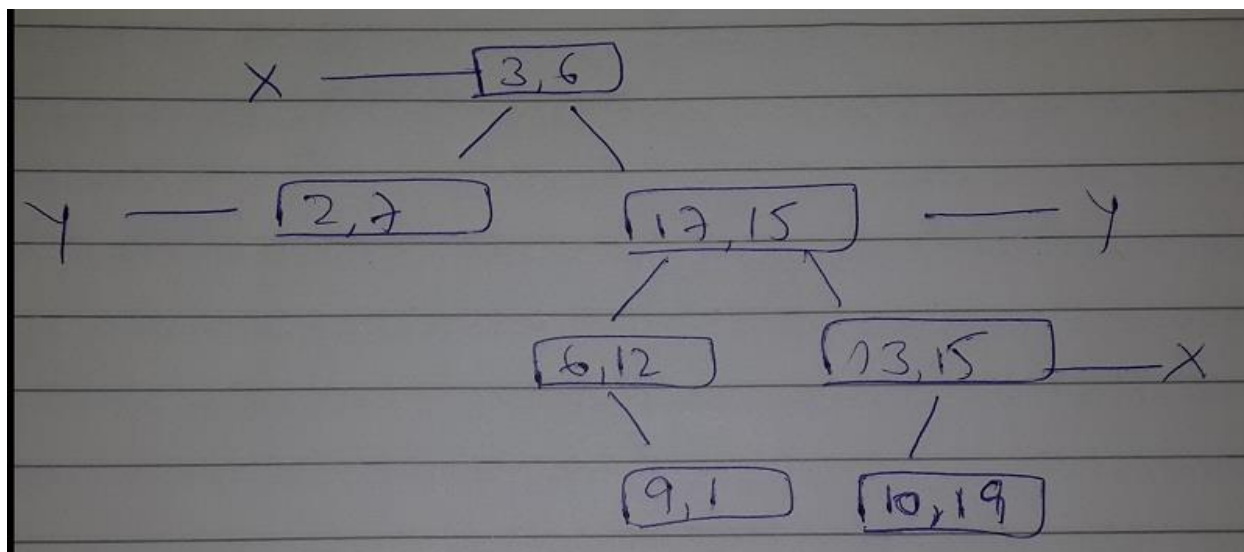
#### 5. Balancing

Cân bằng cây k chiều được quan tâm bởi vì k-d tree là sắp xếp nhiều chiều, do đó kỹ thuật quay vòng cây không thể được sử dụng để cân bằng chúng vì điều này có thể phá vỡ sự bất biến.

Một số biến thể của k-d tree. Chúng bao gồm *k-d tree*, *pseudo k-d tree*, *k-d B-tree*, *hB-tree* and *Bkd-tree*. Nhiều của các biến thể này là adaptive k-d trees.

#### 6. Search

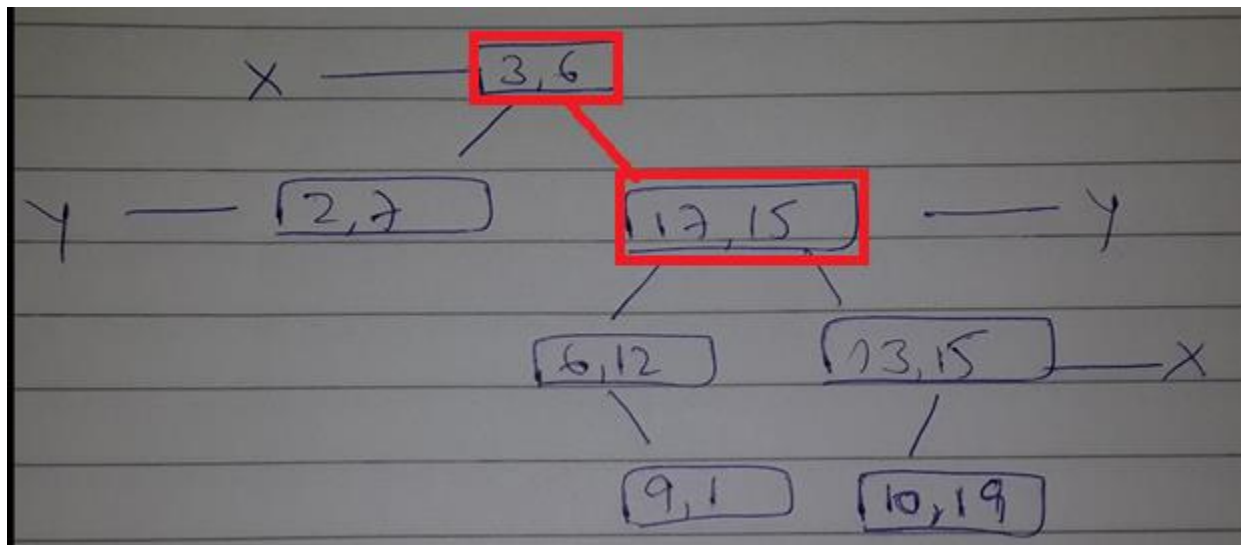
Ví dụ ta có kdTree như hình:



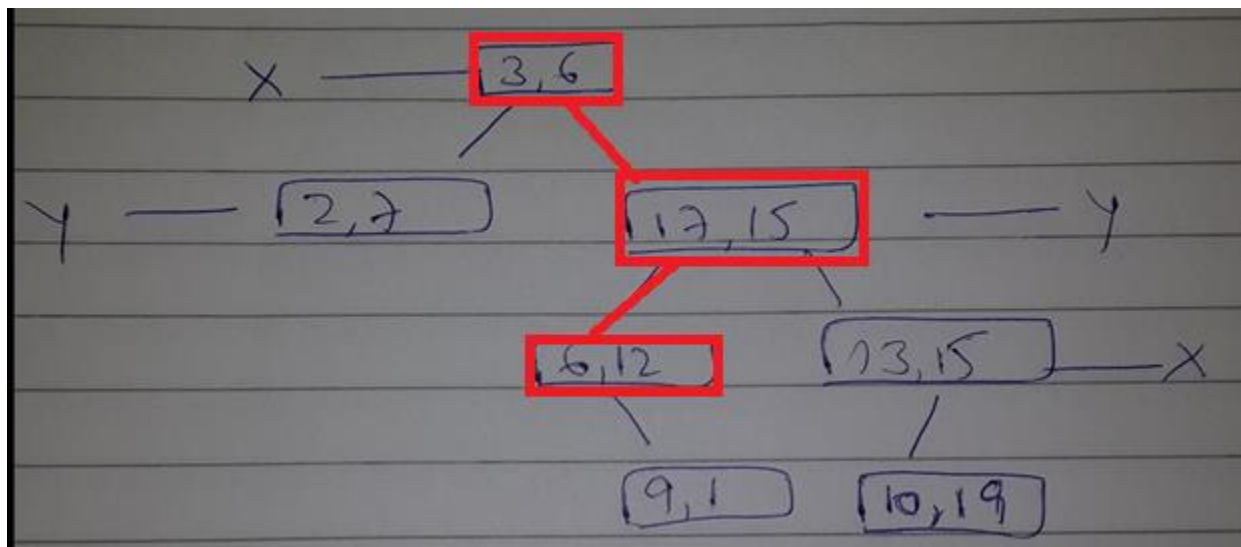
Như trên ta thấy root của tree là chia theo giá trị của x rồi y rồi x....

Giả sử ta có điểm  $q(10, 13)$ , ta cần tìm điểm hàng xóm gần nhất của  $q$ , ta chạy thuật toán như sau:

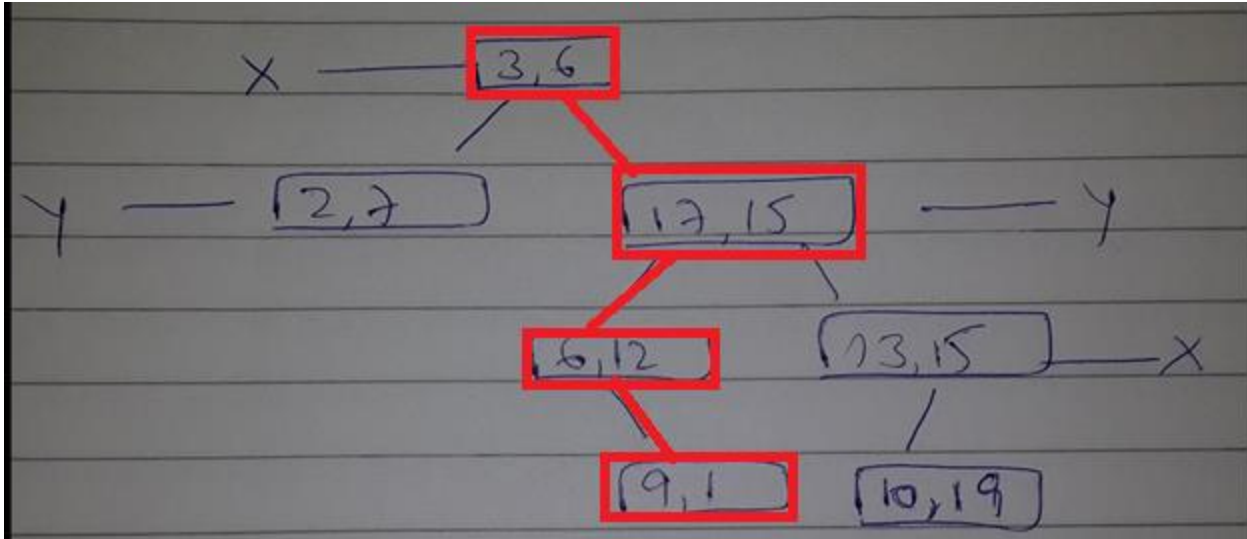
đầu tiên ta so sánh  $X_q = 10 > 3$  ta đi về phía bên phải của cây là  $(17, 15)$



Tiếp theo node này là ở Y nên ta so sánh giá trị  $Y_q = 13 < 15$ , ta đi về phía trái của cây



+ Tiếp tục tại node này ta so sánh  $X_q = 10 > 6$ , ta đi về phía phải của cây



Tại đây đã đến node cuối của cây nên ta có điểm hàng xóm gần nhất của điểm  $q(10, 13)$  là điểm  $(9, 1)$

## 7. Complexity

- Xây dựng cây k-d tree tính từ  $n$  điểm có thể dẫn đến độ phức tạp xấu nhất sau:
  - $O(n \log^2 n)$  nếu  $O(n \log n)$  sắp xếp với Heapsort hoặc Mergesort được sử dụng để tìm kiếm median tại mỗi mức của cây.
  - $O(n \log n)$  nếu  $O(n)$  thuật toán median of medians được sử dụng để chọn median tại mỗi mức của cây.
  - $O(kn \log n)$  nếu  $n$  điểm được sắp xếp theo mỗi chiều sử dụng  $O(n \log n)$  sắp xếp với HeapSort hoặc MergeSort trước khi xây dựng cây.
- Insert:  $O(\log n)$
- Removing:  $O(\log n)$

## IX. Normal estimation(ước lượng pháp tuyến)

Trong đồ họa máy tính, các mặt phẳng pháp tuyến được sử dụng để đưa vào các nguồn sáng để tạo ra bóng của vật thể. Trong quá trình tái tạo bề mặt, các normals xác định hướng của mặt đa giác. Ước tính normal của bề mặt hình học thường là một nhiệm vụ tầm thường, nhưng các điểm trong đám mây điểm không có bất kỳ bề mặt, chúng chỉ là một điểm. Vì lý do này, normal estimation là cần thiết

Có hai cách tiếp cận khác nhau đối với ước tính normal.

1. Tái tạo bề mặt mà các điểm đại diện, và tính toán các normal từ đó
2. Xấp xỉ trực tiếp dữ liệu normal trực tiếp từ đám mây điểm

Đám mây điểm đầu vào trong đồ án chỉ lấy giá trị vị trí của đám mây điểm, và chúng tôi đang sử dụng phương pháp thứ hai nêu trên để tính toán surface normal từ đám mây điểm.

Với mỗi điểm  $p$  trong cloud  $P$ :

1. Tìm các hàng xóm gần nhất của điểm  $P$  bằng cách sử dụng cây kd-tree
  - Tính điểm điểm trung bình cho vùng lân cận và trừ đi giá trị trung bình của tất cả các điểm
  - Sử dụng giá trị đỉnh trung bình của khu vực lân cận, tính giá trị điểm trọng tâm cho các khu vực lân cận
  - Giá trị trọng tâm được sử dụng để tính toán ma trận hiệp phương sai
  - Tính toán các Eigen Vector(vector riêng) và Eigen value(giá trị riêng) của ma trận hiệp phương sai
2. Một hệ tọa độ địa phương bằng cách tính toán giá trị tính vô hướng của vector pháp tuyến riêng(eigen vector normal) và giá trị vector đơn vị của nó(unit orthogonal)
 

```
LocalV = eigenNormal.unitOrthogonal();
LocalU = eigenNormal.cross(localV);
```
3. Kiểm tra xem  $n$  có liên tục định hướng theo viewpoint

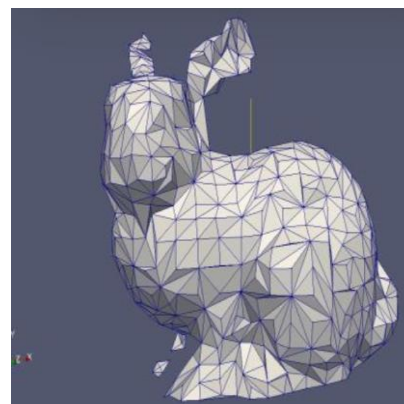
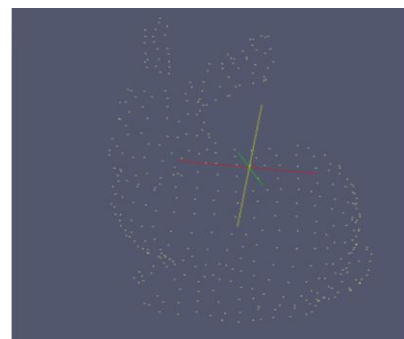
## X. Các giai đoạn của đề án

1. Giai đoạn một vạch ra hướng đi và tìm kiếm tài liệu

Kết quả thu được của đề án :

Surface reconstruction from point cloud(point cloud to mesh)

- Sử dụng PCL(point cloud library)
- Sử dụng một mẫu test để test thuật toán Greedy Projection
- Kết quả được lưu dưới dạng là một VTK file
- Sử dụng những phần mềm có thể mở được những file này để xem kết quả



- Đồ án tập trung vào việc tạo ra một lưới tam giác từ một đám mây điểm ban đầu, và sau khi sử dụng thuật toán Greedy Projection trong thư viện PCL nhóm đã thu được kết quả khá khả quan. Và những ghi chú tại trên website là tài liệu khá hữu ích

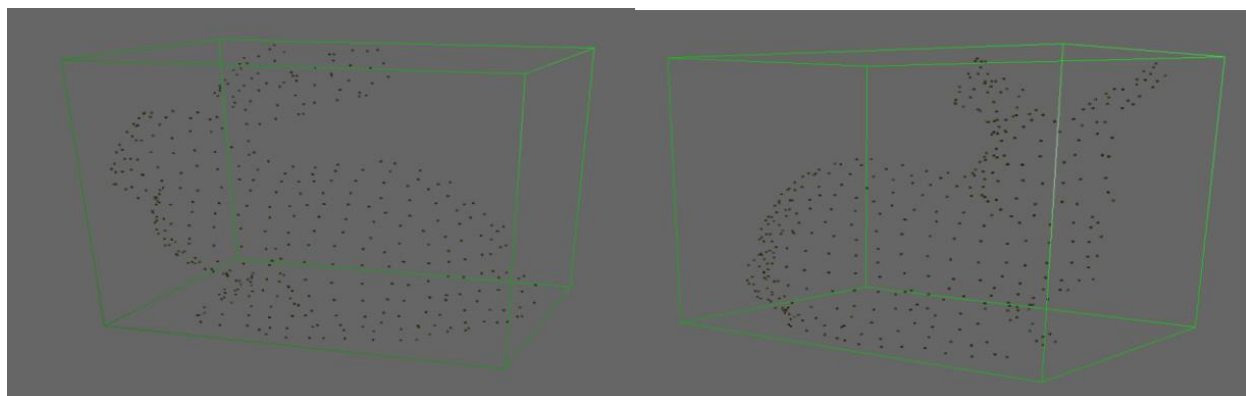
## 2. Giai đoạn 2 : Hiểu thuật toán Greedy projection algorithm(cách thực hiện và cài đặt)

- Thuật toán dựa trên tài liệu kỹ thuật của Rusu R.B. (2009) đã được thực hiện trong PCL. Nó là bao gồm hơn 2000 + dòng code
- Nhóm chúng tôi đã hoàn thành các bước ban đầu như đơn gian hóa đám mây điểm, tính toán các phép chiếu cho mặt phẳng pháp tuyến để ước lượng bề mặt, sử dụng phép chiếu của hệ tọa độ cục bộ, góc giữa các điểm lân cận so với điểm tham chiếu được tính toán. Hình tam giác đầu tiên cho tất cả k-điểm lân cận được tạo sử dụng dữ liệu chiếu này

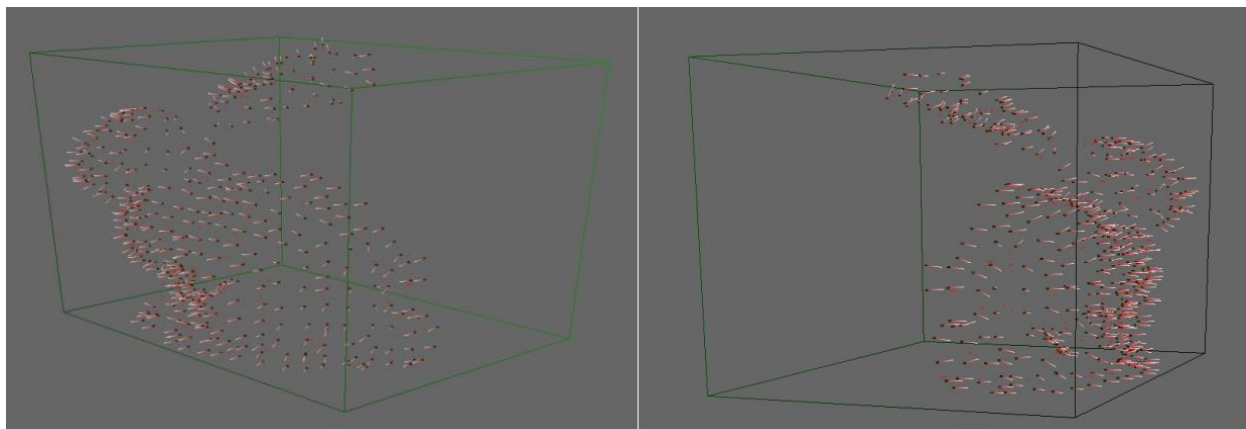
## 3. Giai đoạn 3 : kết quả cuối cùng

Các bước xử lý đám mây điểm

- Xử lý dữ liệu đám mây điểm đầu vào chỉ chứa tọa độ vị trí  $(x, y, z)$  trong không gian 3D. Đầu vào là một tệp văn bản chứa các giá trị vị trí  $(x, y, z)$ .



b. Ước lượng pháp tuyến cho bề mặt



c. Xuất dữ liệu ra file kết quả

4. So sánh kết quả của các thuật toán

		Greedy triangle projection	Poisson
Phát sinh bề mặt		Có thể chỉ tạo ra tam giác đầu tiên cho mỗi các hàng xóm(điểm lân cận)	Việc tái thiết bề mặt từ tập điểm được xử lý một quá trình tuần tự với các ước sau:

			<p>1. Quét tập hợp các điểm.</p> <p>2. Loại bỏ outlier</p> <p>3. Đơn giản hóa để giảm số lượng các điểm đầu vào.</p> <p>4. Làm mượt để giảm dữ liệu đầu vào</p> <p>5. Ước tính và định hướng khi các chuẩn mực chưa được cung cấp.</p> <p>6. Khôi phục lại bề mặt</p>
Tốc độ		nhANH	Chậm hơn greedy triangle projection
Độ phức tạp thuật toán		$O(n)\log_2(n)$	$O(n)\log_2(n)$

Xử lý dữ liệu		Các bước để hợp nhất và đơn giản hóa đám mây điểm đã được thực hiện đã giúp giảm được lưới điểm 30k xuống 4k điểm.	Việc ước lượng và loại bỏ outlier thực hiện để đơn giản hóa dữ liệu đầu vào.

## XI. Poisson

### 1. Giới thiệu

Tái tạo bề mặt 3D từ các mẫu điểm là một vấn đề được nghiên cứu kỹ lưỡng trong đồ họa máy tính. Nó cho phép lấp các dữ liệu được quét, làm đầy các lỗ bề mặt, và remeshing của các mô hình hiện có. Chúng tôi cung cấp một cách tiếp cận mới lạ biểu hiện tái thiết bề mặt như là giải pháp cho phương trình Poisson.

Cái nhìn sâu sắc của chúng ta là có một mối quan hệ không thể tách rời giữa các điểm định hướng lấy mẫu từ bề mặt của mô hình và chức năng chỉ thị của mô hình. Cụ thể, độ dốc của chức năng chỉ thị là trường vector không có ở hầu hết mọi nơi (vì chức năng chỉ thị không thay đổi ở hầu hết mọi nơi), ngoại trừ các điểm gần bề mặt, nơi nó bằng bề mặt bên trong bình thường. Do đó, các mẫu điểm định hướng có thể được xem như các mẫu gradient cho chức năng chỉ thị của mô hình (Hình 1).

Vấn đề tính toán hàm chỉ thị làm giảm để đảo ngược toán tử gradient, tức là tìm hàm vô hướng  $\chi$  có gradient tốt nhất xấp xỉ một trường vector  $v$  được xác định bởi các mẫu, tức là  $\min_{\chi} k \nabla \chi - v$ . Nếu ta áp dụng toán tử divergence, vấn đề variational biến đổi thành một vấn đề Poisson tiêu chuẩn: tính toán vô hướng chức năng  $\chi$  có Laplacian (phân kỳ của gradient) tương đương với sự phân kỳ của trường vector  $\sim V$ ,

$$\Delta \chi \equiv \nabla \cdot \nabla \chi = \nabla \cdot V.$$

Hơn nữa, trong nhiều chương trình kết hợp ngầm, giá trị của hàm ẩn được hạn chế chỉ gần các điểm lấy mẫu, và do đó việc tái thiết có thể chứa các tấm bề mặt giả mạo cách xa các mẫu này. Thông thường vấn đề này được làm giảm đi bằng cách giới thiệu các điểm "ngoài bề mặt" phụ trợ (ví dụ [CBC \* 01, OBA \* 03]). Với tái tạo bề mặt Poisson, các tấm bề mặt như vậy hiếm khi nảy sinh vì gradient của hàm ẩn được hạn chế ở tất cả các điểm không gian. Đặc biệt nó được hạn chế để không xa các mẫu.

Đã có nhiều nghiên cứu liên ngành về giải quyết:

Các vấn đề Poisson và nhiều phương pháp hiệu quả và mạnh mẽ đã được phát triển. Một khía cạnh cụ thể của vấn đề.

Ví dụ là một giải pháp chính xác cho phương trình Poisson chỉ là cần thiết gần bề mặt được tái tạo. Điều này cho phép chúng tôi để thúc đẩy thích ứng Poisson solvers để phát triển tái thiết thuật toán phức tạp không gian và thời gian tỷ lệ thuận với kích thước của bề mặt đã được tái tạo

## 2. Công việc liên quan

Surface reconstruction Việc tái thiết bề mặt từ các điểm định hướng có một số khó khăn trong thực tế.

Việc lấy mẫu điểm thường không đồng nhất. Các vị trí và normals nói chung là ồn do lấy mẫu không chính xác và quét sai đăng ký. Và, những hạn chế khả năng tiếp cận trong suốt quét có thể để lại một số vùng bề mặt không có dữ liệu. Với những thách thức này, các phương pháp tái thiết cố gắng suy luận về topo của bề mặt không rõ, vừa chính xác vừa (nhưng không trang phục) các dữ liệu ồn ào, và điền vào lỗ hổng lý.

Các vấn đề Poisson Các phương trình Poisson phát sinh trong nhiều lĩnh vực ứng dụng. Ví dụ, trong đồ họa máy tính, nó được sử dụng để lập bản đồ giai điệu các hình ảnh dải động cao [FLW02], chỉnh sửa liên mạch vùng hình ảnh [PGB03], cơ học chất lỏng [LGF04], và chỉnh sửa lưới [YZX \* 04]. Các giải pháp Multigrid Poisson thậm chí đã được điều chỉnh để tính toán GPU hiệu quả [BFGS03, GWL \* 03].

## 3. Tiếp cận poisson reconstruction algorithm

Defining the gradient field Bởi vì chức năng chỉ thị là một hàm cố định piecewise, rõ ràng tính toán của nó gradient trường sẽ dẫn đến một trường vector với không bị chặn giá trị tại ranh giới bề mặt. Để tránh điều này, chúng tôi hoán đổi chức năng chỉ thị với một bộ lọc làm mịn và xem xét trường gradient của hàm mịn. Sau đây lemma chính thức hóa mối quan hệ giữa gradient của chức năng chỉ thị mịn màng và trường bình thường bề mặt.

Lemma: Cho một  $M$  thể rắn với ranh giới  $\partial M$ , cho phép  $\chi_M$  biểu thị hàm chỉ thị của  $M$ ,  $\sim N \partial M(p)$  là hàm số bên trong bề mặt bình thường tại  $p \in \partial M$ ,  $F \sim (q)$  là một bộ lọc làm mịn, và  $F_p(q) = F(q-p)$  bản dịch của nó đến điểm  $p$ . Độ dốc của chức năng chỉ thị mịn bằng với trường vector thu được bằng cách làm phẳng bề mặt bình thường lĩnh vực:

Proof: Để chứng minh điều này, chúng tôi hiển thị sự bình đẳng cho mỗi thành phần của trường vector. Tính toán một phần phái sinh của các chức năng hiển thị mịn với  $x$ , chúng tôi nhận được:

Approximating the gradient field Of Tất nhiên, chúng ta không thể đánh giá sự tích phân bề mặt vì chúng ta chưa biết bề mặt hình học. Tuy nhiên, bộ đầu vào của các điểm định hướng cung cấp chính xác đủ thông tin để gần đúng tách rời với một tổng kết rời rạc. Cụ thể, sử dụng điểm đặt  $S$  để phân vùng  $\partial M$  vào các mảng vá khác biệt  $P_s \subset \partial M$ , chúng ta có thể ước lượng các tích phân trên một vá  $P_s$  bởi giá trị tại điểm mẫu  $s.p$ , được thu nhỏ bởi diện tích của miếng vá: Solving the Poisson problem Đã hình thành một trường vector  $V$ , chúng ta muốn giải quyết cho hàm  $\chi$  sao cho  $\nabla \chi = V$ .

Tuy nhiên,  $\sim V$  nói chung không thể tích (tức là nó không phải là curlfree), do đó, một giải pháp chính xác thường không tồn tại. Để tìm giải pháp xấp xỉ bình phương tốt nhất, chúng tôi áp dụng.

## 4. Thực hiện

### 4.1 . Vấn đề Discretization

Defining the function space Cho một tập các mẫu điểm  $S$  và chiều sâu cây  $D$  tối đa, chúng ta xác định octree  $O$  là octree tối thiểu với thuộc tính mà mỗi mẫu điểm rơi vào một nút lá ở độ sâu  $D$ .

Tiếp theo, chúng ta xác định một không gian của các hàm thu được như một khoảng dịch và quy mô của một hàm cố định, tách rời đơn vị, cơ sở  $F: R^3 \rightarrow R$ . Đối với mỗi nút  $o \in O$ , chúng ta đặt  $F_o$  là nút "nút tích hợp" đơn vị trung tâm về nút  $o$  và kéo dài bởi kích thước  $o$ :

$$F_o(q) = F\left(\frac{q-o.c}{o.w}\right) \cdot \frac{1}{o.w}$$

Nơi  $o.c$  và  $h.o.w$  là trung tâm và chiều rộng của nút  $o$ .

## 4.2 . Định nghĩa trường vector

Để cho phép độ chính xác của nút phụ, chúng ta tránh clamping vị trí của một mẫu đến trung tâm của nút lá có chứa và thay vào đó sử dụng phép nội suy tam giác để phân phối mẫu qua tám nút gần nhất. Do đó, chúng ta định nghĩa xấp xỉ của chúng ta với trường gradient của hàm chỉ thị như sau:

$$\vec{V}(q) = \sum_{s \in S} \sum_{o \in Ngbr(x)} a_{os} F_o(q) \cdot \vec{N}$$

Trong đó  $NgbrD$  là tám nút sâu-D gần nhất với  $s.p$  và  $\{a_{os}, s\}$  là trọng số nội suy tam giác. (Nếu hàng xóm không có trong cây, chúng tôi tinh chỉnh nó để bao gồm chúng.)

## 4.3 Giải pháp poisson

Sau khi định nghĩa trường vector  $V$ , ta muốn giải các hàm  $\chi \in FO, F$  sao cho gradient của  $\chi$  gần nhất với  $V$ , tức là một giải pháp cho phương trình Poisson  $\Delta\chi = \nabla \cdot V$ .

Một thách thức của việc giải quyết cho  $\chi$  là mặc dù  $\chi$  và các hàm phối hợp của  $V$  nằm trong khoảng trống  $FO, F$  không nhất thiết là trường hợp các hàm  $\Delta\chi$  và  $\nabla \cdot V$ .

Để giải quyết vấn đề này, chúng ta cần phải giải bài toán cho hàm  $\chi$  such sao cho phép chiếu của  $\Delta\chi$  lên không gian  $FO, F$  gần nhất với phép chiếu của  $\nabla \cdot V$ . Vì, nói chung, các chức năng  $F_o$  không tạo thành một cơ sở trực giao, giải quyết vấn đề này trực tiếp là tốn kém. Tuy nhiên, chúng ta có thể đơn giản hóa vấn đề bằng cách giải quyết cho hàm  $\chi$  giảm thiểu:

$$\sum_{o \in \mathcal{O}} \left\| \langle \Delta\tilde{\chi} - \nabla \cdot \vec{V}, F_o \rangle \right\|^2 = \sum_{o \in \mathcal{O}} \left\| \langle \Delta\tilde{\chi}, F_o \rangle - \langle \nabla \cdot \vec{V}, F_o \rangle \right\|^2.$$

Do đó, với vector  $v$  có vector  $v$  có tọa độ thứ  $i$  là  $v = h \cdot v$ , với, mục tiêu là giải quyết cho hàm  $\chi$  sao cho vector thu được bằng cách chiếu Laplacian của  $\chi$  vào mỗi của  $F_o$  là càng gần  $v$  càng tốt.

Để diễn đạt điều này trong dạng ma trận, hãy để  $\chi\chi = \sum_o X_o F_o$ , để chúng ta giải được vector  $x \in \mathbb{R}^{|O|}$ . Sau đó, hãy xác định  $|O| \times |O|$  ma trận  $L$  sao cho  $Lx$  trả về sản phẩm dấu chấm của Laplacian với mỗi  $o$ . Cụ thể, với tất cả  $o, o' \in O$ , phần  $(o, o')$  lần nhập thứ  $L$  được đặt thành:

$$L_{o,o'} \equiv \left\langle \frac{\partial^2 F_o}{\partial x^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial y^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial z^2}, F_{o'} \right\rangle.$$

Vì vậy, giải quyết cho  $\chi$  số tiền để tìm kiếm

$$\min_{x \in \mathbb{R}^{|O|}} \|Lx - v\|^2.$$

Lưu ý rằng ma trận  $L$  là thưa thớt và đối xứng. (Thưa thớt vì  $F_o$  được hỗ trợ một cách chặt chẽ và đối xứng bởi vì  $R f'' g = -R f' g'$ ). Ngoài ra, có một cấu trúc đa kiến trúc cố hữu trên  $FO, F$ , vì vậy chúng ta sử dụng một phương pháp tương tự như phương pháp multigrid trong [GKS02], giải quyết hạn chế  $L_d$  của  $L$  tới khoảng không gian được kéo dài bởi các hàm  $d$  chiều sâu (sử dụng một giải phóng gradient liên hợp) và đưa dung dịch cố định trở lại  $FO, F$  để cập nhật phần dư.

#### 4.4 Chiết xuất Isosurface

Để có bề mặt được xây dựng lại  $\partial M$ , trước hết phải chọn một đẳng độ và sau đó chiết xuất isosurface tương ứng từ chức năng chỉ thị tính toán. Chúng tôi chọn chế độ ăn mòn để bề mặt chiết xuất gần xấp xỉ vị trí của các mẫu đầu vào. Chúng tôi làm điều này bằng cách đánh giá  $\chi$  ở vị trí mẫu và sử dụng trung bình các giá trị cho việc tách chiết isosurface:

$$\partial \tilde{M} \equiv \{q \in \mathbb{R}^3 \mid \tilde{\chi}(q) = \gamma\} \quad \text{with} \quad \gamma = \frac{1}{|S|} \sum_{s \in S} \tilde{\chi}(s.p).$$

Sự lựa chọn isovalue này có thuộc tính mà tỉ lệ  $\chi$  không thay đổi isosurface. Vì vậy, biết lĩnh vực vector  $V$  đến một hằng số nhân cung cấp đầy đủ thông tin để tái tạo lại bề mặt.

#### 4.5 Các mẫu không đồng nhất

Bây giờ chúng tôi mở rộng phương pháp của chúng tôi đến trường hợp các mẫu điểm phân phối không đồng đều. Như trong [Kaz05], phương pháp tiếp cận của chúng tôi là ước tính mật độ lấy mẫu tại địa phương và quy mô đóng góp của mỗi điểm cho phù hợp. Tuy nhiên, thay vì chỉ đơn giản là nhân rộng độ lớn của một hạt nhân có độ rộng cố định liên kết với mỗi điểm, chúng ta bổ sung thêm cho chiều rộng hạt nhân. Điều này dẫn đến tái thiết duy trì các tính năng sắc nét trong các khu vực lấy mẫu dày đặc và cung cấp một sự phù hợp trơn tru ở những khu vực thưa thớt.

Chúng ta thực hiện convolution theo cách tương tự như phương trình 3. Cho độ sâu  $D \leq D$  chúng ta thiết lập ước lượng mật độ là tổng các nút chức năng ở độ sâu  $D$ :

$$W_D(q) \equiv \sum_{s \in S} \sum_{o \in \text{Ngbr}_D(s)} \alpha_{o,s} F_o(q).$$

Tính toán trường vector Sử dụng ước lượng mật độ, chúng ta sửa đổi tổng kết trong Phương trình 3 sao cho sự đóng góp của mỗi mẫu tỷ lệ với diện tích liên kết của nó trên bề mặt. Cụ thể, sử dụng thực tế là khu vực này tỷ lệ nghịch với mật độ lấy mẫu, chúng tôi thiết lập:

$$\vec{V}(q) \equiv \sum_{s \in S} \frac{1}{W_D(s.p)} \sum_{o \in \text{Ngbr}_D(s)} \alpha_{o,s} F_o(q).$$

Sử dụng thực tế là nút chức năng ở độ sâu nhỏ hơn tương ứng với các bộ lọc làm phẳng rộng hơn, chúng ta xác định

$$\vec{V}(q) \equiv \sum_{s \in S} \frac{1}{W_{\hat{D}}(s.p)} \sum_{o \in \text{Ngbr}_{\text{Depth}(s.p)}(s)} \alpha_{o,s} F_o(q).$$

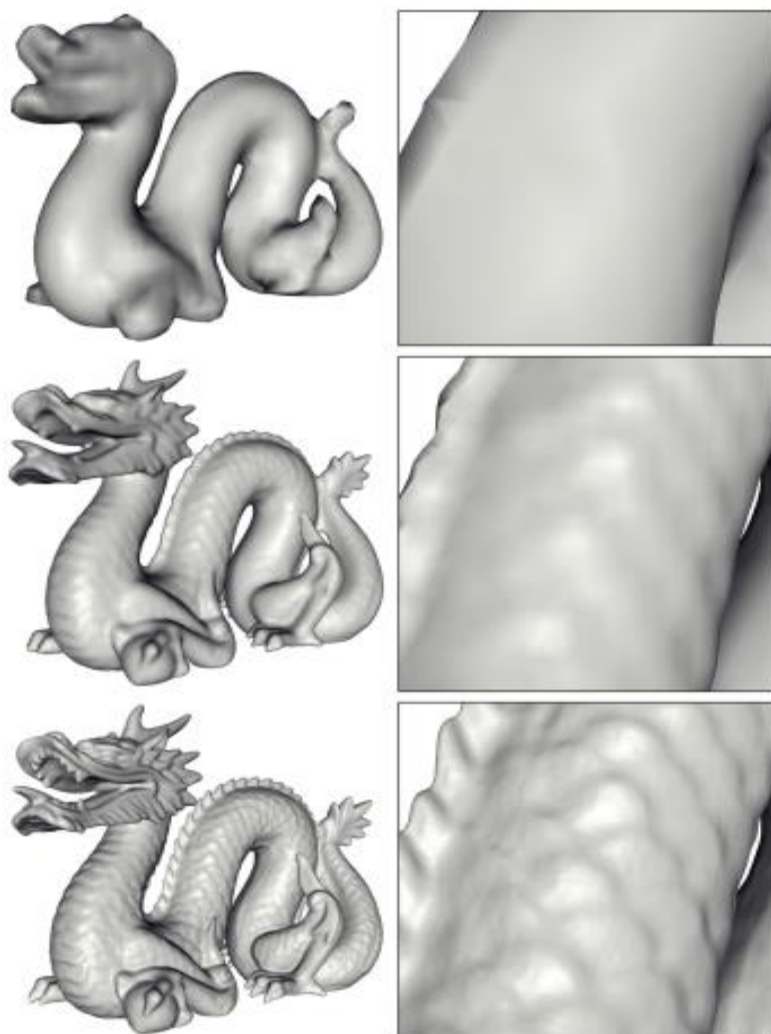
Chọn một đẳng thức. Cuối cùng, chúng tôi sửa đổi bước khai thác bề mặt bằng cách chọn một giá trị isovalue là trung bình trọng số của các giá trị  $\chi \sim$  tại các vị trí mẫu:

$$\partial \tilde{M} \equiv \{q \in \mathbb{R}^3 \mid \tilde{\chi}(q) = \gamma\} \quad \text{with} \quad \gamma = \frac{\sum \frac{1}{W_{\hat{D}}(s.p)} \tilde{\chi}(s.p)}{\sum \frac{1}{W_{\hat{D}}(s.p)}}.$$

## 5. Kết Quả

### 5.1 Giải quyết

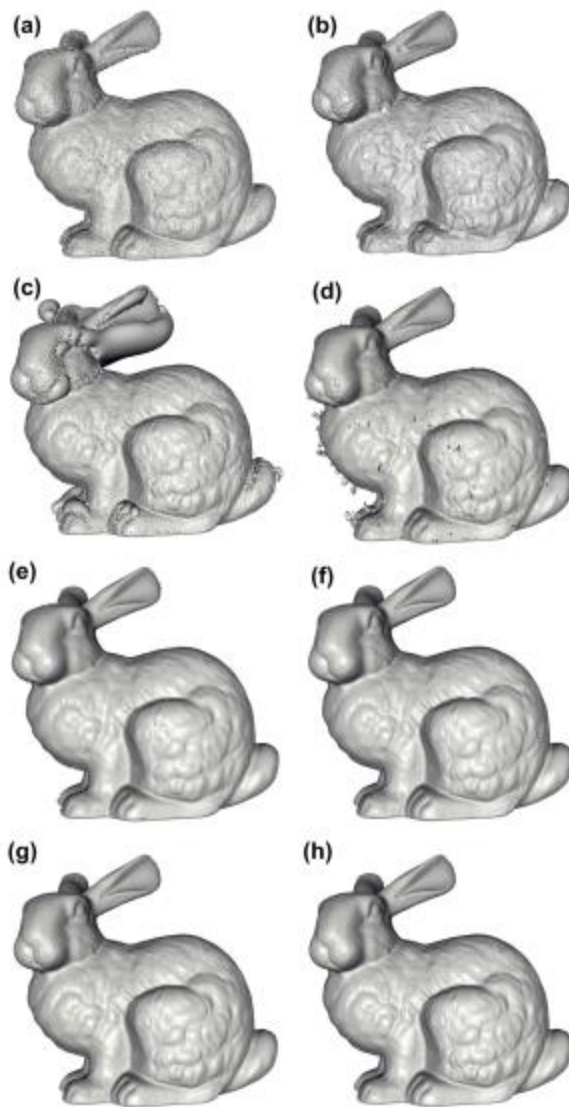
Hình 1 cho thấy các kết quả tái thiết của chúng ta cho mô hình "rỗng" ở các độ sâu 6, 8, và 10 octree (Trong bối cảnh tái thiết trên lưới bình thường, điều này tương ứng với độ phân giải lần lượt là 643, 2563 và 10243) độ sâu của cây được tăng lên, các chức năng có độ phân giải cao hơn được sử dụng để phù hợp với chức năng chỉ thị, và do đó các công trình tái hiện lại chi tiết hơn. Ví dụ, quy mô của con rồng, quá tinh vi khi bắt được ở độ phân giải thô nhất, bắt đầu xuất hiện và trở nên rõ nét hơn khi độ sâu octree tăng lên



*Figure 1 Các công trình tái tạo của mô hình rồng ở độ sâu octree 6 (trên cùng), 8 (giữa), và 10 (đáy)*

## 5.2 So sánh với công việc trước đây

Chúng tôi so sánh các kết quả của thuật toán tái thiết của chúng tôi với các kết quả thu được bằng cách sử dụng Power Crust [ACK01], Robust Cocone [DG04], Chức năng Fast Radial Basis (FastRBF) [CBC \* 01], Sự phân chia mức độ thống nhất đa cấp (MPU) [OBA \* 03], tái tạo bề mặt từ các điểm chưa được tổ chức [HDD \* 92], xử lý ảnh khối lượng hình ảnh (VRIP) [CL96], và phương pháp dựa trên FFT của [Kaz05].



*Figure 2 Các cuộc tái thiết của thỏ Stanford sử dụng năng lượng Crust (a), Robont Cocone (b), RBF nhanh (c), MPU (d), tái thiết Hoppe và cộng sự (e), VRIP (f), tái tạo dựa trên FFT g), và tái thiết Poisson (h).*

Trường hợp thử nghiệm ban đầu của chúng tôi là bộ dữ liệu thỏ "Stanley" của Stanford với 362.000 điểm được lắp ráp từ 10 hình ảnh. Dữ liệu đã được xử lý để phù hợp với định dạng đầu vào của mỗi thuật toán. Ví dụ, khi chạy phương pháp của chúng tôi, chúng tôi ước tính một mẫu bình thường từ vị trí của hàng xóm; Chạy VRIP, chúng tôi đã sử dụng quét đã đăng ký làm đầu vào, duy trì tính chính xác của việc lấy mẫu và cung cấp các giá trị độ tin cậy.

Hình 2 so sánh các tái cấu trúc khác nhau. Kể từ khi dữ liệu có chứa tiếng ồn, các phương pháp nội suy như Power Crust (a) và Robust Cocone (b) tạo ra các bề mặt bị ồn. Các phương pháp như Fast RBF (c) và MPU (d), mà chỉ hạn chế hàm ẩn gần các điểm mẫu, kết quả tái thiết với tám bề mặt giả mạo. Các phương pháp không interpolatory, chẳng hạn như cách tiếp cận của [HDD \* 92] (e), có thể làm dịu tiếng ồn, mặc dù thường tốn kém chi phí mô hình. VRIP (f), cách tiếp cận FFT (g), và phương pháp Poisson (h) tất cả các bề mặt của thỏ đều được tái tạo một cách chính xác ngay cả khi có tiếng ồn và chúng tôi so sánh ba phương pháp này một cách chi tiết hơn.

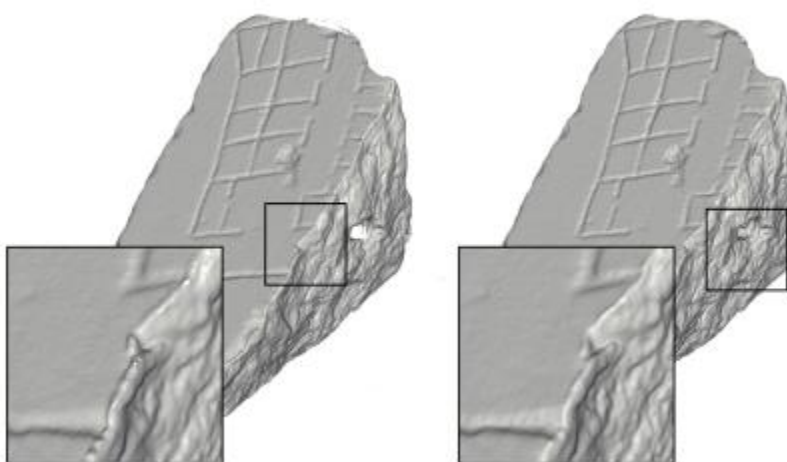


Figure 3 Tái tạo lại một đoạn của viên Forma Urbis Romae sử dụng VRIP (trái) và giải pháp Poisson (bên phải).

### 5.3 Hiệu quả và khả năng mở rộng

Bảng 1 tóm tắt hiệu quả thời gian và không gian của thuật toán của chúng tôi đối với mô hình "rỗng" và chỉ ra rằng yêu cầu bộ nhớ và thời gian của thuật toán của chúng tôi là khoảng phân hai trong độ phân giải. Do đó, khi chúng ta tăng độ sâu octree của một, chúng ta thấy rằng thời gian chạy, bộ nhớ trên cao, và số tam giác đầu ra tăng gần bằng một yếu tố bốn.

Tree Depth	Time	Peak Memory	# of Tris
7	6	19	21 000
8	26	75	90 244

9	126	155	374 686
19	633	699	1 516 806

**Table 1:** Thời gian chạy (tính bằng giây), mức sử dụng bộ nhớ cao điểm (tính bằng megabyte) và số hình tam giác trong mô hình dựng lại để tái tạo độ sâu của mô hình rỗng. Một chiều sâu hạt là 6 đã được sử dụng để ước lượng mật độ.

Thời gian chạy và hiệu suất bộ nhớ của phương pháp của chúng tôi trong việc tái tạo Bunny Stanford ở độ sâu 9 được so sánh với hiệu suất của các phương pháp liên quan trong Bảng 2. Mặc dù trong thí nghiệm này, phương pháp của chúng tôi không nhanh nhất cũng không phải là bộ nhớ hiệu quả nhất, nó có thể mở rộng để tái cấu trúc độ phân giải cao hơn. Ví dụ, Hình 8 cho thấy một sự tái thiết của người đứng đầu của David Michelangelo ở độ sâu 11 từ một bộ 215.613.477 mẫu. Việc tái thiết được tính bằng 1,9 giờ và 5,2 GB bộ nhớ RAM, tạo ra mô hình tam giác 16.328.329. Cố gắng để tính toán xây dựng lại tương đương với các phương pháp như phương pháp tiếp cận FFT sẽ yêu cầu xây dựng hai voxel lưới ở độ phân giải 20483 và sẽ yêu cầu vượt quá 100GB bộ nhớ.



*Figure 4 Một số hình ảnh của việc tái thiết đầu của Michelangelo David, đã thu được thuật toán của chúng tôi với độ sâu cây tối đa là 11. Khả năng tái tạo lại đầu ở độ phân giải cao như vậy cho phép chúng ta tạo ra các tính năng tốt trong mô hình như phần lót mồn*

## 6. Kết Luận

Chúng tôi đã chỉ ra rằng tái thiết bề mặt có thể được thể hiện dưới dạng một vấn đề Poisson, tìm kiếm chức năng chỉ thị nhất quán với một tập hợp các quan sát ồn ào, không đồng nhất và chúng tôi đã chứng minh rằng phương pháp này có thể phục hồi mạnh mẽ các chi tiết tốt từ ồn ào trong thế giới thực quét.

Có một số con đường cho công việc trong tương lai:

- Mở rộng cách tiếp cận để khai thác các giá trị độ tin cậy mẫu.
- Kết hợp thông tin tuyến đường từ quá trình quét vào quá trình giải quyết.
- Mở rộng hệ thống để cho phép xử lý ngoài lõi cho các bộ dữ liệu lớn.

## **XII. Những vấn đề gặp phải**

1. Mất rất nhiều thời gian tìm hiểu, cài đặt thư viện PCL, sử dụng CMake trên nền tảng windows, tìm hiểu các thư viện và cách sử dụng(số lượng hàm, class của PCL rất lớn)
2. Rất khó tìm ra hướng đi và thuật toán ngay từ đầu phải thử nhiều cách khác nhau
3. Đã cố gắng cài đặt PCL hỗ trợ Qt(lập trình giao diện) nhưng bị gặp lỗi trong quá trình compile thư viện PCL(mất 2 tuần)

## **XIII. Ứng dụng**

1. Tái tạo lại mô hình 3D được áp dụng rộng rãi trong các lĩnh vực như y học, thực tế ảo vv.
2. Đây là một công việc tiềm năng và có thể phát triển mạnh trong tương lai

## **XIV. Tài liệu tham khảo**

1. Trang WebSite PCL <http://pointclouds.org/documentation/>
2. Thuật toán Fast Greedy Triangle Algorithm  
<https://www.cs.dartmouth.edu/~trdata/reports/TR94-215.pdf>