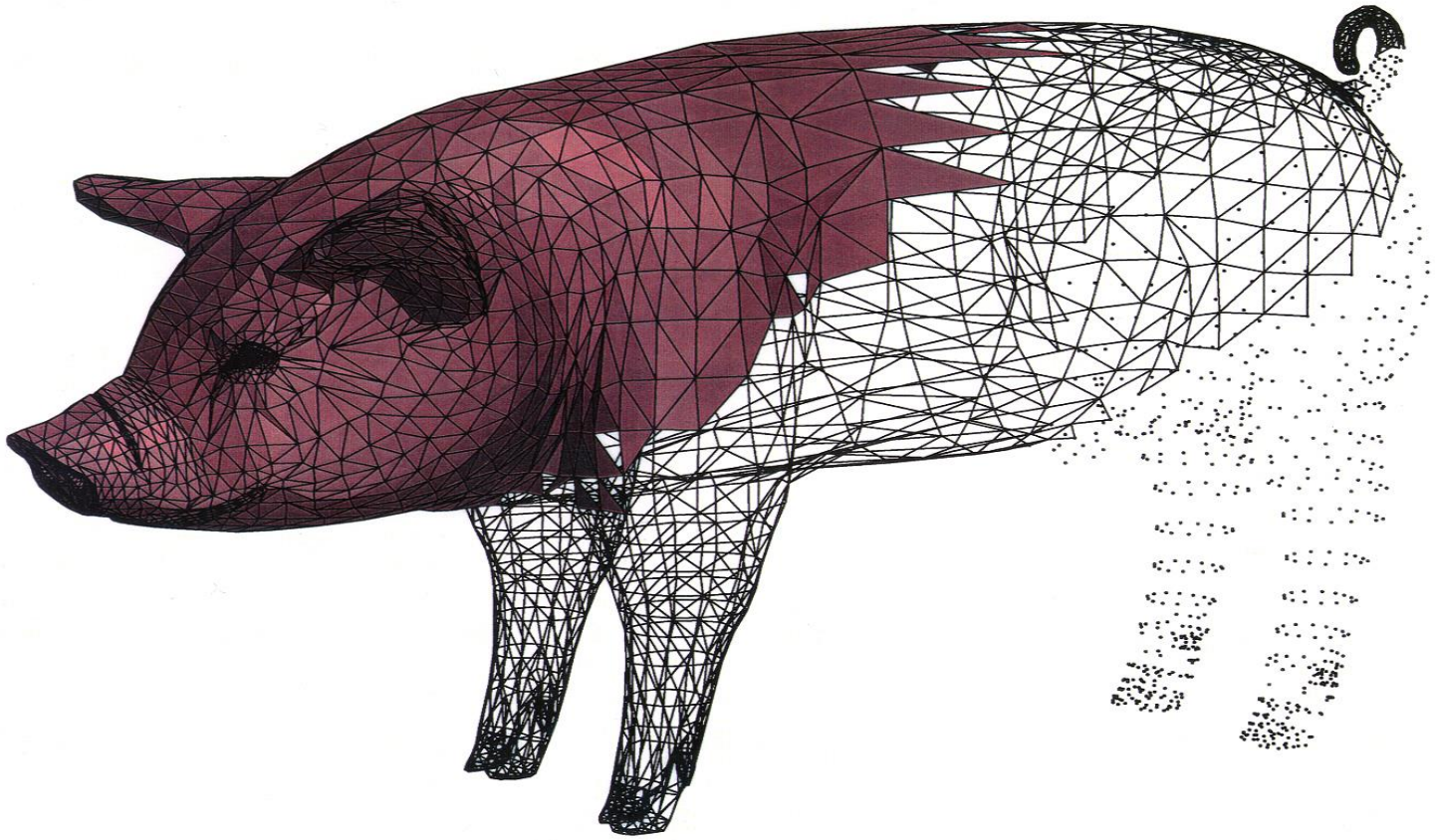
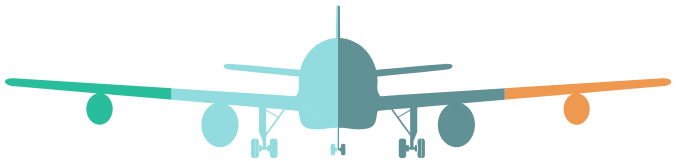
Surface Reconstruction

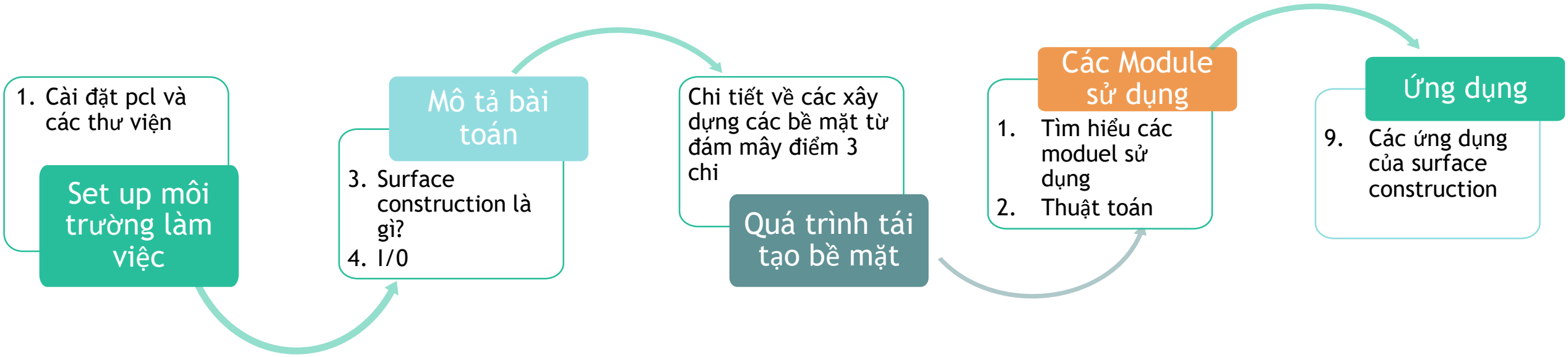



Welcome



Content

Surface construction



Cài đặt PCL và các thư viện liên quan

- PCL hỗ trợ những môi trường Linux, Windows, MacOS X ..vv.
- Hỗ trợ phát triển sử dụng ngôn ngữ C++, python
- Những thư viện bên thứ 3 mà pcl yêu cầu
- Boost (C ++ Semi Standard Library)
- Eigen (Matrix Library)
- FLANN (Nearest Neighbor Search Library)
- VTK (Visualization Library)
- QHull (Computational Geometry Library)
- OpenNI/OpenNI2 (RGB-D Sensor Library)

Cài đặt Qt

- Cài đặt Qt
- Tạo ra ứng dụng chạy đa nền tảng, phần cứng khác nhau mà không phải thay đổi code nhiều

Cài đặt CMake

- Sử dụng Cmake để tự tạo project chạy trên visual studio từ file source code
- Sử dụng Cmake giúp ta nhanh chóng thêm các thư viện cần thiết vào project

Cài đặt CMake

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(MY_GRAND_PROJECT)
find_package(PCL 1.3 REQUIRED COMPONENTS common io
)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(pcd_write_test pcd_write.cpp)
target_link_libraries(pcd_write_test ${PCL_COMMON_
LIBRARIES} ${PCL_IO_LIBRARIES})
```

Cài đặt CMake

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(MY_GRAND_PROJECT)
find_package(PCL 1.3 REQUIRED COMPONENTS common io
)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(pcd_write_test pcd_write.cpp)
target_link_libraries(pcd_write_test ${PCL_COMMON_
LIBRARIES} ${PCL_IO_LIBRARIES})
```

Cài đặt CMake

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(MY_GRAND_PROJECT)
find_package(PCL 1.3 REQUIRED COMPONENTS common io
)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(pcd_write_test pcd_write.cpp)
target_link_libraries(pcd_write_test ${PCL_COMMON_
LIBRARIES} ${PCL_IO_LIBRARIES})
```

Cài đặt CMake

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(MY_GRAND_PROJECT)
find_package(PCL 1.3 REQUIRED COMPONENTS common io
)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(pcd_write_test pcd_write.cpp)
target_link_libraries(pcd_write_test ${PCL_COMMON_
LIBRARIES} ${PCL_IO_LIBRARIES})
```

Cài đặt CMake

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(MY_GRAND_PROJECT)
find_package(PCL 1.3 REQUIRED COMPONENTS common io
)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(pcd_write_test pcd_write.cpp)
target_link_libraries(pcd_write_test ${PCL_COMMON_
LIBRARIES} ${PCL_IO_LIBRARIES})
```

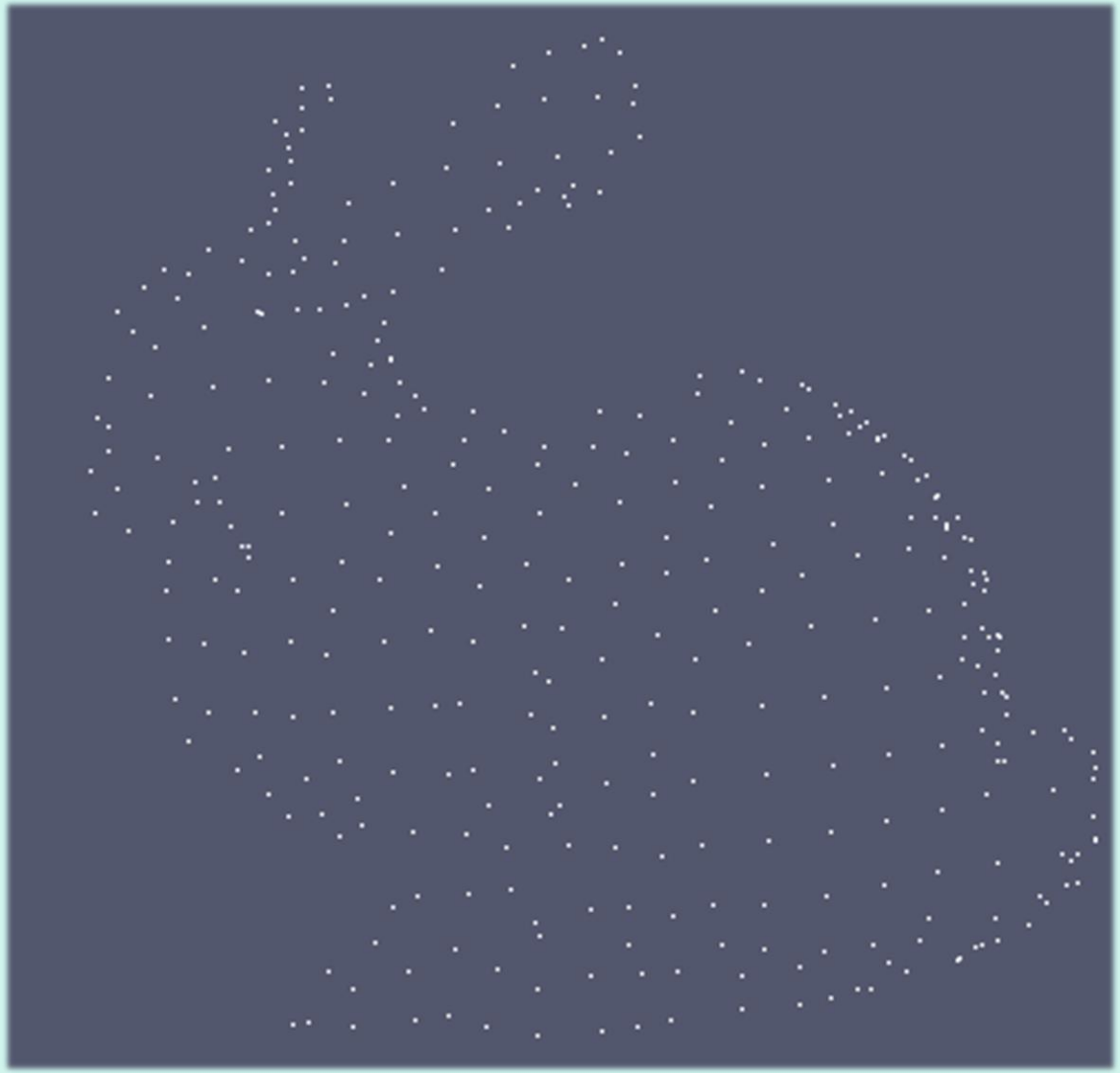
Cài đặt CMake

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(MY_GRAND_PROJECT)
find_package(PCL 1.3 REQUIRED COMPONENTS common io
)
include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
add_executable(pcd_write_test pcd_write.cpp)
target_link_libraries(pcd_write_test ${PCL_COMMON_
LIBRARIES} ${PCL_IO_LIBRARIES})
```

Cài đặt PCL và các thư viện liên quan

- PCL hỗ trợ những môi trường Linux, Windows, MacOS X ..vv.
- Hỗ trợ phát triển sử dụng ngôn ngữ C++, python
- Những thư viện bên thứ 3 mà pcl yêu cầu
- Boost (C ++ Semi Standard Library)
- Eigen (Matrix Library)
- FLANN (Nearest Neighbor Search Library)
- VTK (Visualization Library)
- QHull (Computational Geometry Library)
- OpenNI/OpenNI2 (RGB-D Sensor Library)

Làm tạo lại bề mặt từ đám mây điểm?



Xử lý input

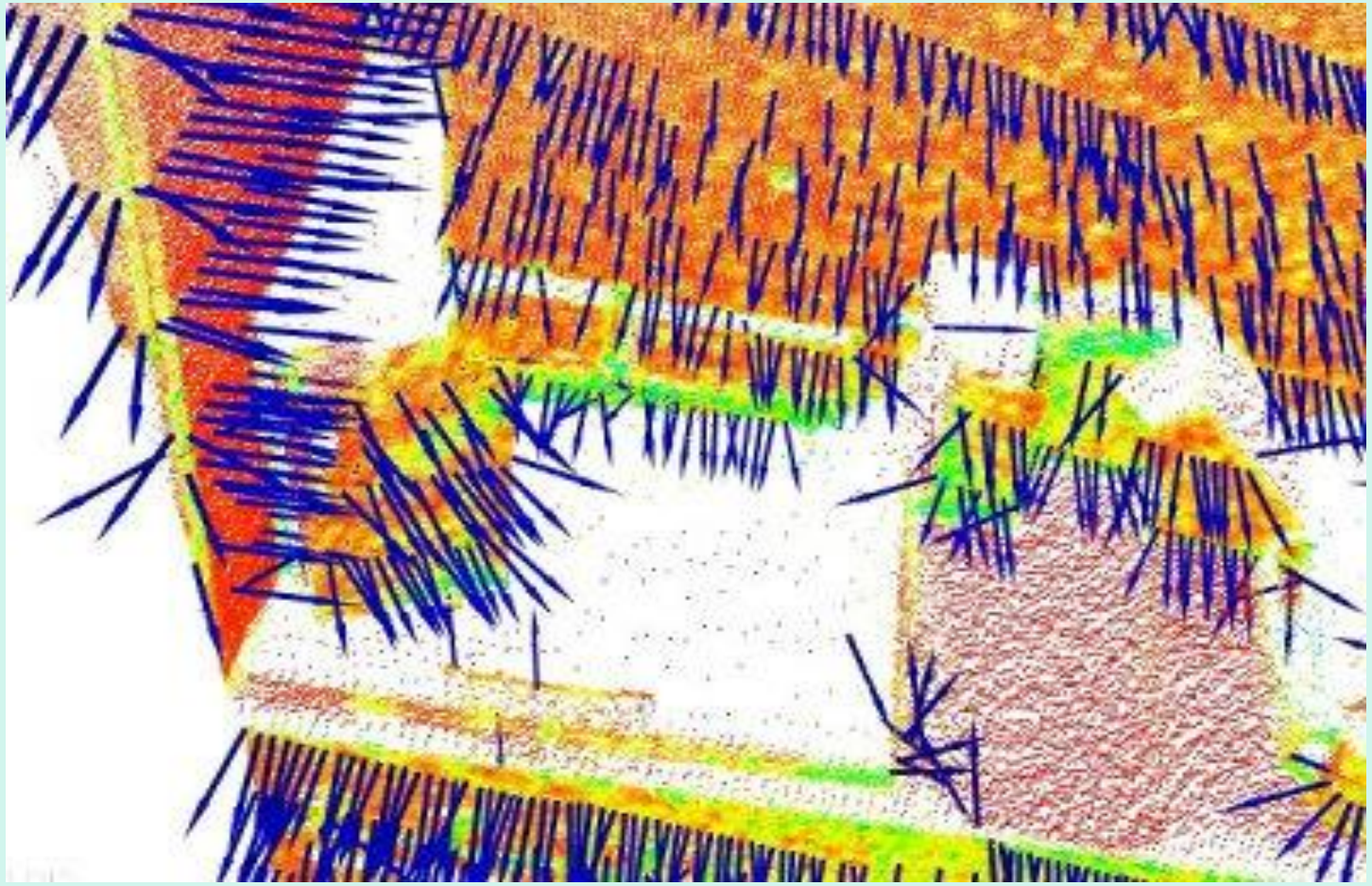
- Đám mây điểm có thể được tạo ra từ các thiết bị scanner, với sự phát triển của các thiết bị scanner thì đám mây điểm có thể được tạo ra trong khoảng thời gian ngắn (được lưu dưới dạng file *pcd ..vvv.)
- Làm mịn (smoothing) và lấy mẫu lại(resampling) nếu đám mây điểm nhiễu(noisy) hoặc đám mây điểm được hình thành từ nhiều lần quét dẫn tới những điểm trong point cloud lộn xộn. Dẫn tới độ phức tạp của việc dự đoán bề mặt có thể bị thay đổi
- Normal estimation(ước lượng pháp tuyến)
- Surface recontruction

Xử lý input

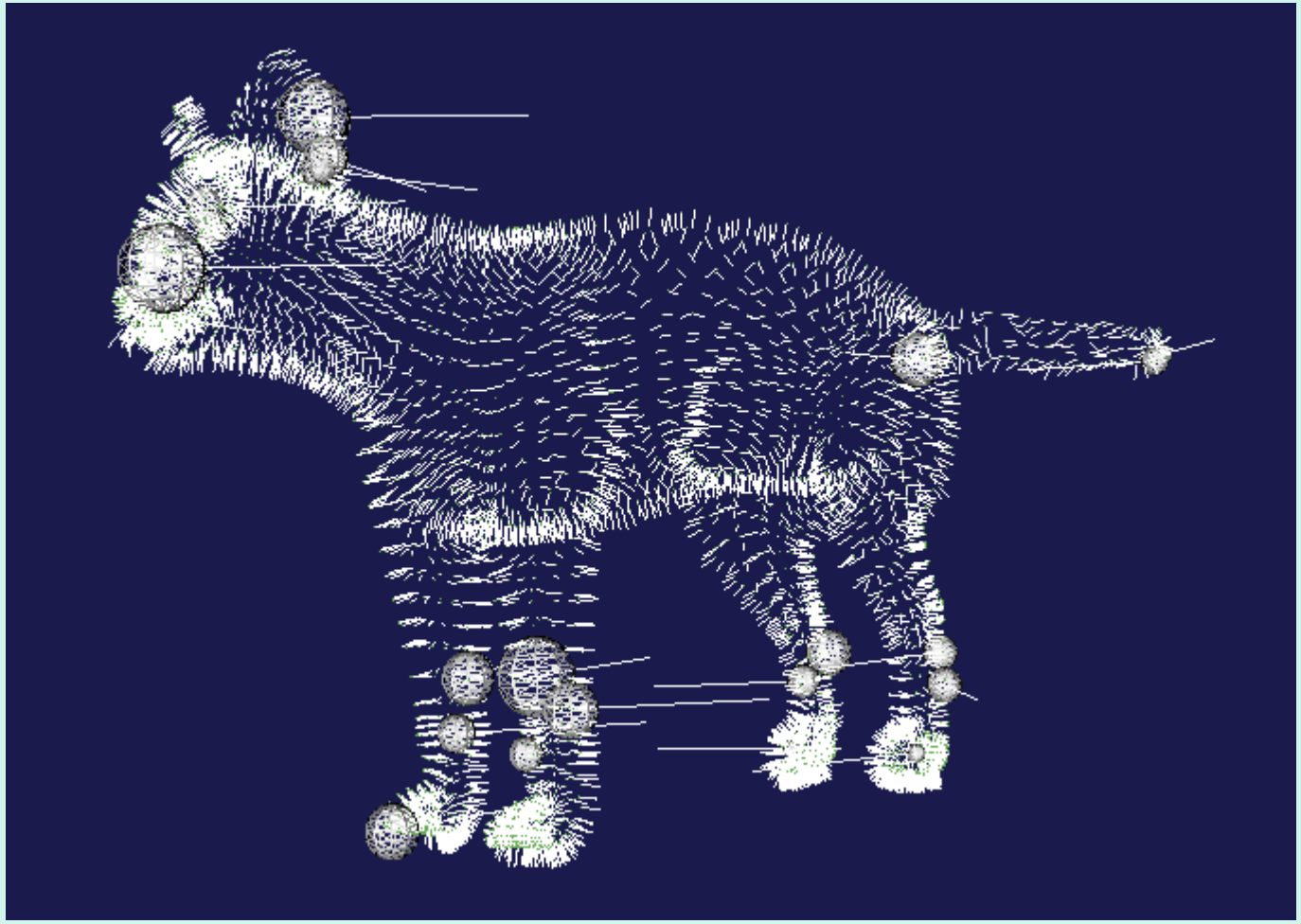
- Đám mây điểm có thể được tạo ra từ các thiết bị scanner, với sự phát triển của các thiết bị scanner thì đám mây điểm có thể được tạo ra trong khoảng thời gian ngắn (được lưu dưới dạng file *pcd ..vvv.)
- Làm mịn (smoothing) và lấy mẫu lại(resampling) nếu đám mây điểm nhiễu(noisy) hoặc đám mây điểm được hình thành từ nhiều lần quét dẫn tới những điểm trong point cloud lộn xộn. Dẫn tới độ phức tạp của việc dự đoán bề mặt có thể bị thay đổi
- Normal estimation(ước lượng pháp tuyến)
- Surface recontruction

Normal Estimation(ước lượng pháp tuyến)

- Làm sao để biết những điểm nào thuộc cùng bề mặt?
- Làm sao từ những điểm trong đám mấy điểm suy ra được hướng của bề mặt?



Point cloud + normals = clouds normals

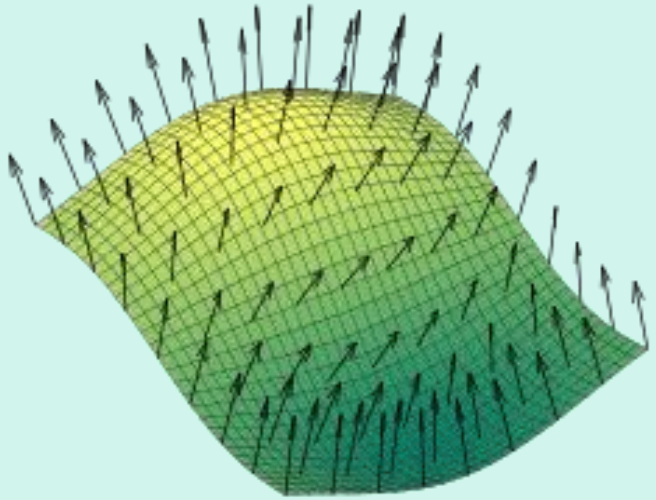


Ước lượng normals cho các điểm lân cận

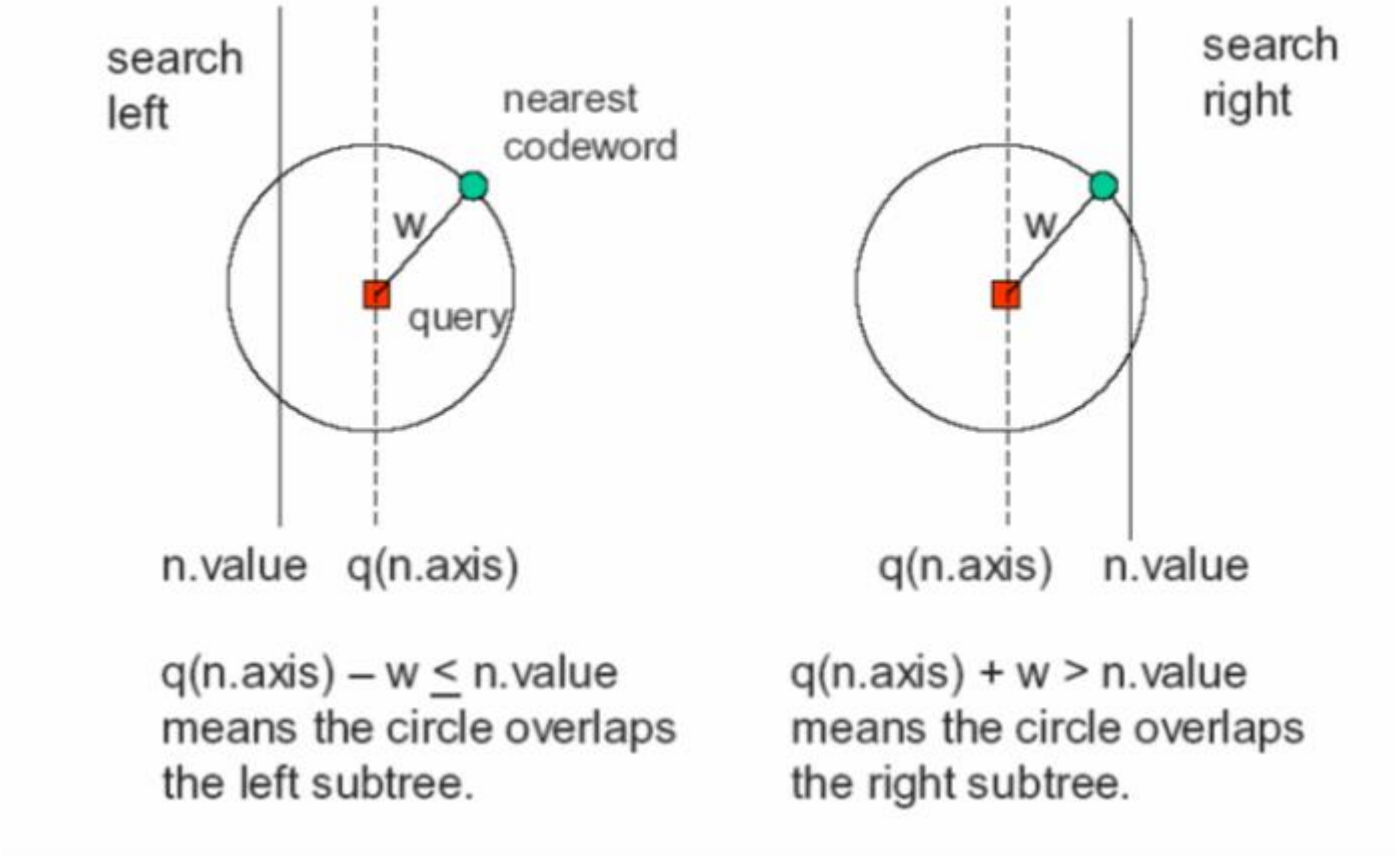
- Tìm kiếm các điểm lân cận gần nhất dung thuật toán k- nearest neight boirs
- Sự dụng cấu trúc kd tree để lưu những điểm 3D trong point cloud hỗ trợ việc tìm kiếm nhanh hơn

Ước lượng normals cho các điểm lân cận

- không thể tính toán trực tiếp pháp tuyến mặt phẳng từ tập đám mây điểm => Cần phải ước lượng pháp tuyến



Thuật toán K-Nearest neighbor



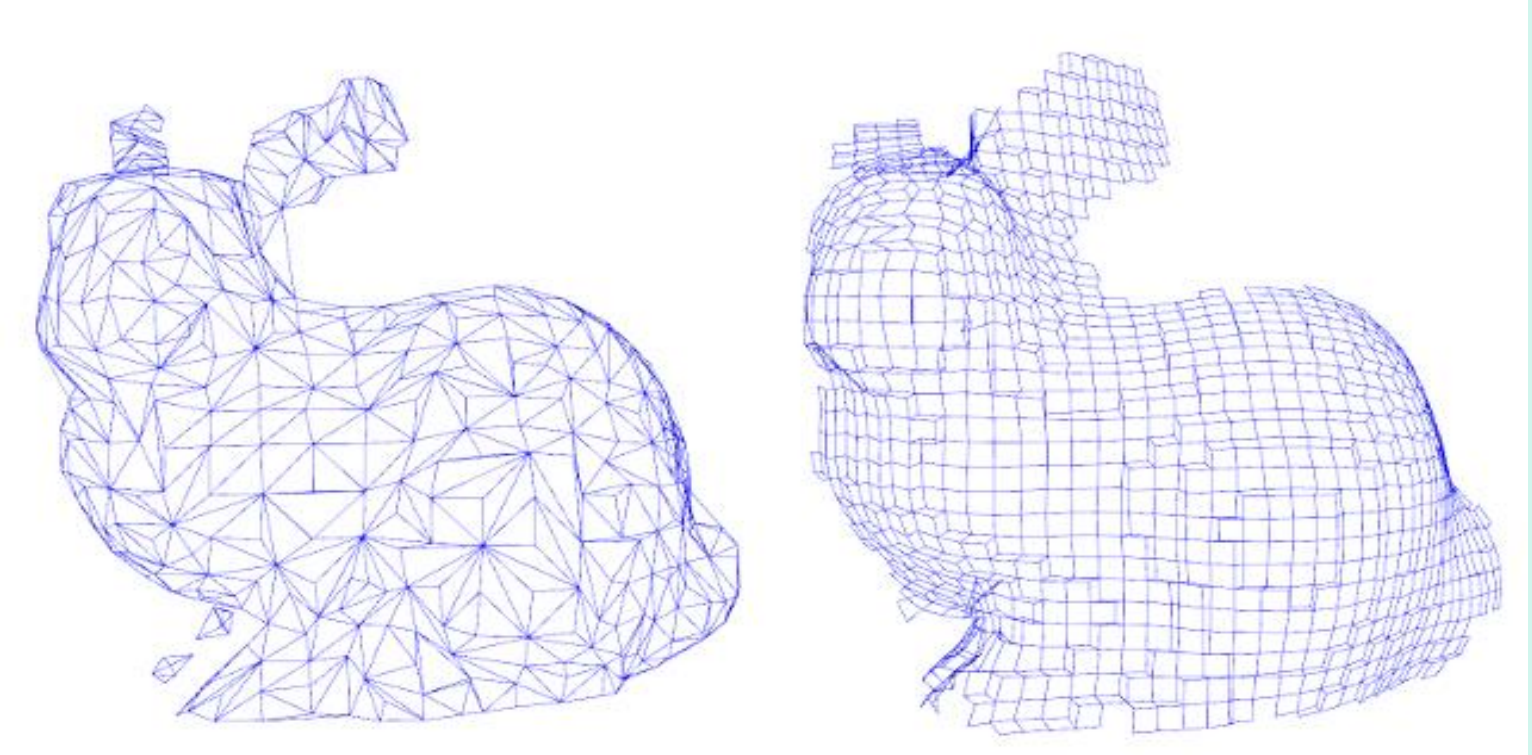
Thuật toán K-Nearest neighbor

```
NNS(q: point, n: node, p: ref point w: ref distance)
if n.left = n.right = null then {leaf case}
    w' := ||q - n.point||;
    if w' < w then w := w'; p := n.point;
else
    if q(n.axis) ≤ n.value then
        search_first := left;
    else
        search_first := right;
    if (search_first == left)
        if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w);
        if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
    else // search_first == right
        if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
        if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w);
```

initial call | NNS(q, root, p, infinity)

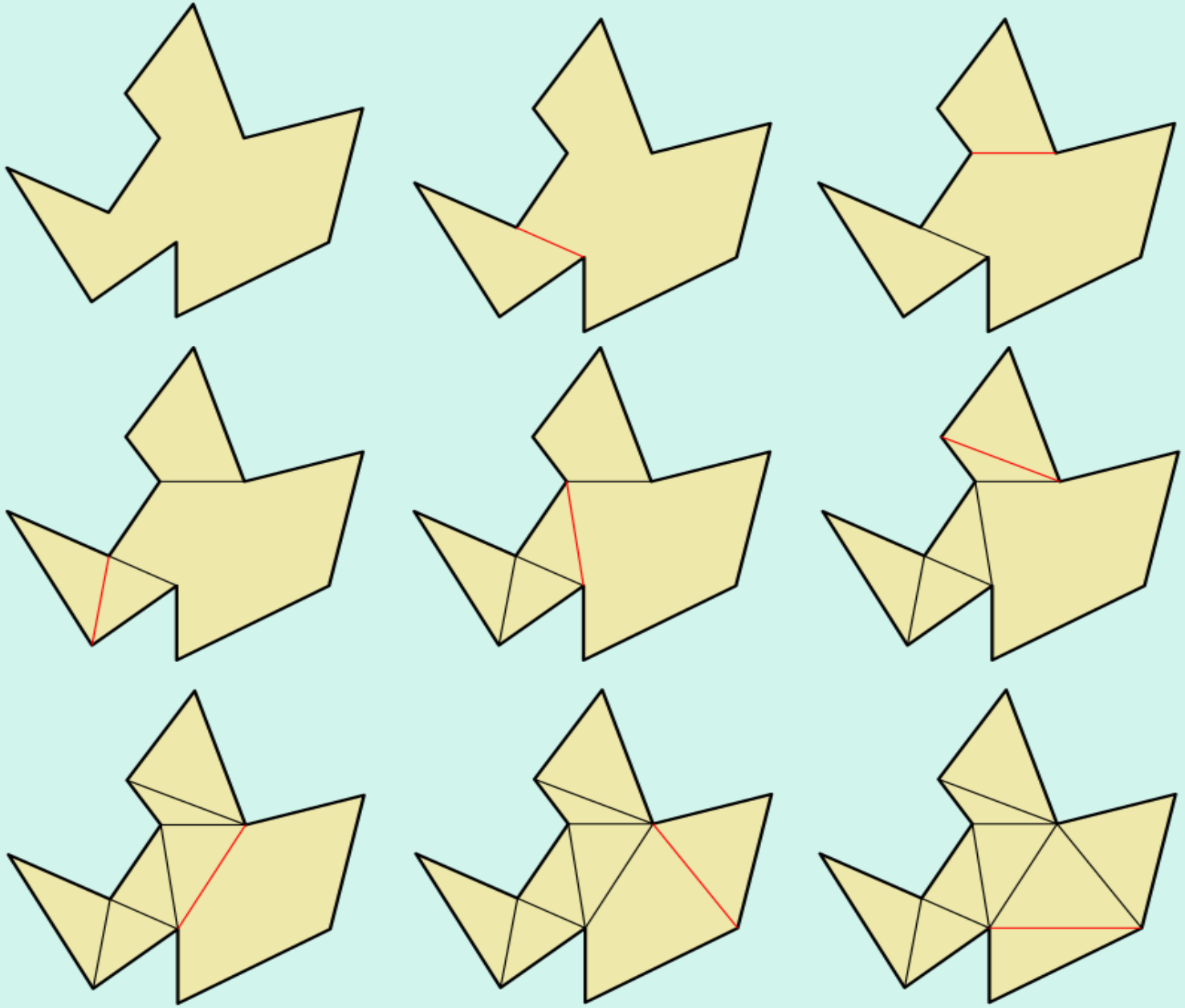
Tạo bề mặt từ các điểm

- Thuật toán **Fast triangulation of unordered point clouds O(nlogn) và Một phương pháp tạo lưới chậm hơn nhưng nó làm mịn và lấp các lỗ tốt hơn**



Thuật toán Greedy Projection Triangulation

- Thuật toán hoạt động bằng cách duy trì 1 list các điểm mà các điểm đó được kết nối bằng lưới các đường viền tạo thành các polygon, sau đó mở rộng dần cho đến khi tất cả các điểm có thể kết nối với nhau
- Kết nối các điểm lân cận dựa trên nguyên tắc tham lam
- Trường hợp tốt nhất O(|V|)
- Trường hợp xấu nhất O(|V|log|V|)



III. Cách sử dụng

- Thư viện: #include <**pcl/surface/gp3.h**>
- Ta có ví dụ sau về cách sử dụng thư viện:
  - Thiết làm các thông số cho point cloud

```
// Set the maximum distance between connected points (maximum
edge length)
  gp3.setSearchRadius (0.025);

  // Set typical values for the parameters
  gp3.setMu (2.5);
  gp3.setMaximumNearestNeighbors (100);
  gp3.setMaximumSurfaceAngle(M_PI/4); // 45 degrees
  gp3.setMinimumAngle(M_PI/18); // 10 degrees
  gp3.setMaximumAngle(2*M_PI/3); // 120 degrees
  gp3.setNormalConsistency(false);
```