The instruction of the 3DPhenoMVS

Acquiring images

A total of 8 tomato samples located inside a greenhouse were collected for the full growth period covering 7 time points, seven point cloud models were reconstructed for each tomato sample. After the fruit was ripe, 5 tomato fruits were randomly picked from each sampling plant, so we have 56 original point clouds and 40 tomato fruits point clouds for further process and analysis.

Challenges in the extraction of tomato phenotype

In terms of data collection, the high price of a LiDAR system limits its application in agriculture. In terms of phenotype extraction, most of previous work focused on tomato seedling stage, the steps were complicated, and three is no open source for point cloud data processing. Therefore, we developed a low-cost, open-access pipeline to extract 3D phenotypic traits covering the whole growth periods of tomato.

Reconstruction Solution and technology roadmap

Our goal is actually to solve the phenotypic trait extraction problem on the generated 3D point clouds of tomato plants during the whole growth period. The image acquisition is conducted by moving a camera around the tomato plant, and produce the point cloud through three-dimensional reconstruction. Based on the L_1-*Medial* Skeleton of stalk point cloud [1], the node detection and internode length calculation regarding the stalk of one individual tomato plant at different time points were completed in a fully automated way. In other words, it is not necessary to build the correspondences for certain organs, such as node and leaf, at different growth stages in order to investigate the temporal trait changes regarding one plant organ. The results showed that R^2 values between the phenotypic traits and the manually measurements of stem length, plant height, internode length and transverse diameter were more than 0.85. The detailed technical roadmap is as follows (Figure S1):

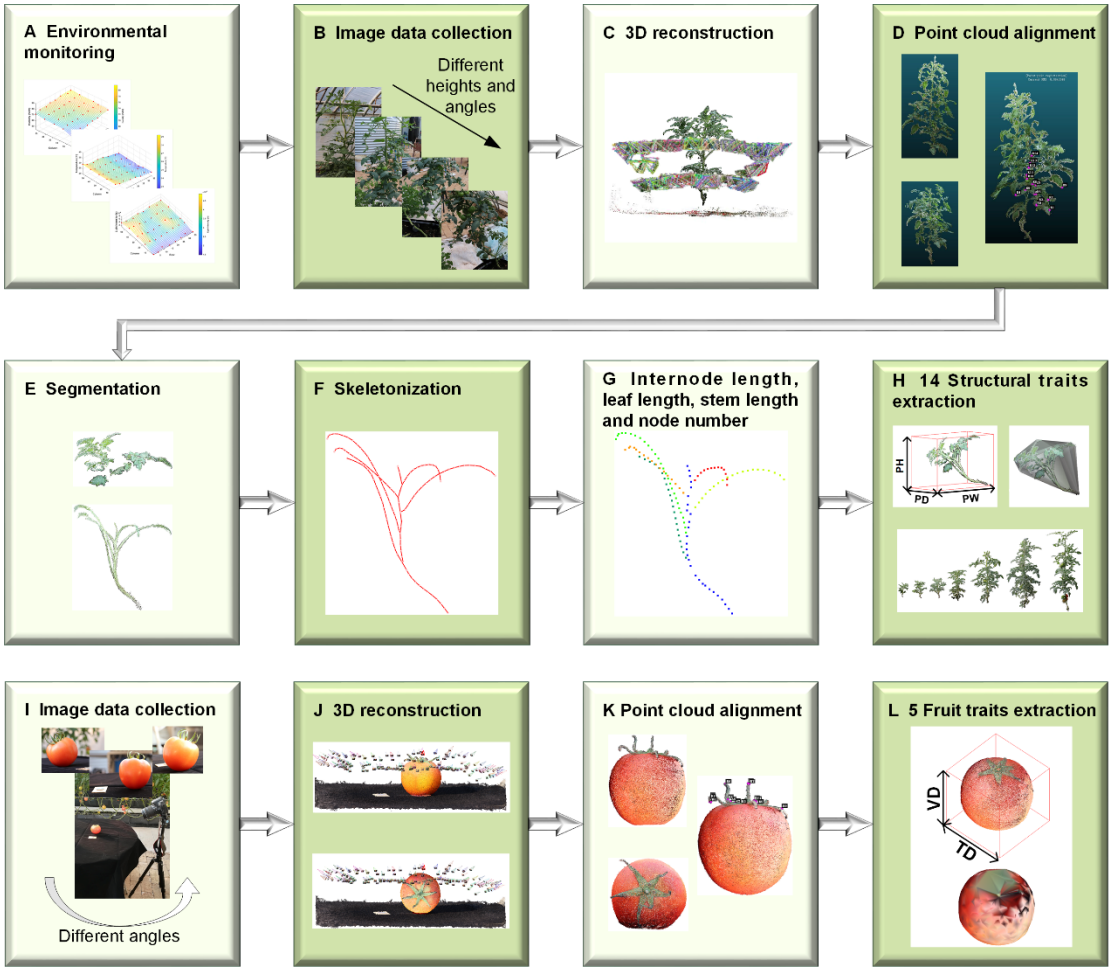


Figure S1. Technology roadmap of using 3DPhenoMVS

Detailed methodological information

Our project mainly consists of 12 parts: (1) Environmental monitoring. A distribution map of environmental differences is illustrated for tomato plant sample selection in Figure 1A. (2) Multiple-image data collection covering the whole life cycle of tomato plants, as shown in Figure 1B. (3) 3D point cloud generation through the combined SfM-MVS algorithms. (4) Alignment of point clouds regarding one tomato plant. For large tomato plants, image data are collected at two height levels, and the generated point clouds need to be aligned and registered together to produce a complete representation of the whole tomato plant. (5) Segmentation of stalk and leaf point clouds. (6) Skeletonization of stalk point clouds for structural phenotypic trait estimation. (7) Node detection on the skeletonized stem point clouds. Node detection is conducted on the skeletonized points; hence, the phenotypic traits, including node number, internode length, and stem length, are calculated automatically. (8) 3D structural phenotypic trait calculation and analysis. (9) Multiple-image data collection of tomato fruits. (10) 3D point

cloud generation through the combined SfM-MVS algorithms. (11) Alignment of point clouds regarding one tomato fruit. (12) Tomato fruit phenotypic trait calculation and analysis.

Part 1 Image Acquisition

A digital camera is used to take multi-angle and multi-layer shooting around a single plant. In the early stage of tomato plant, each image was taken at about every 4-6°. The overlap between two adjacent images was guaranteed to be more than 70%, and the whole plant accounted for more than 80% of each image, and a total of around 100 multi-view images were obtained. In the later stage of tomato growth, layered multi-angle shooting was adopted, and a total of more than 300 multi-view images were obtained. Similarly, high degree of overlap between two adjacent pictures in each layer, and at the same time, there should be more than 50% overlap between two upper layer and lower layers, which is helpful for subsequent point cloud alignment to generate complete tomato plant clouds (Figure S2A and S2B).

For each tomato fruit, it was picked when it was mature for image acquisition. The fruit was placed on the table in the forward and lateral directions (Figure S2C and S2D), and image acquisition as described above.

Figure S2. Image acquisition

Attention

1) Markers should be placed next to the plant, and both the image and maker should be taken

in the image, which will help for the subsequent ratio conversion between the estimated phenotypic trait parameters and the real ones. In this experiment, the planting trough was used as a marker.

2) The image quality affects the quality of the reconstructed point cloud directly. Please tune the camera parameters according to the actual situation.

3) The overlap between the images mentioned above is suggested as Figure S3.

Figure S3. Adjacent image sequence

Part 2 Three-dimensional reconstruction

Procedure

Use Visual SFM+CMVS/PMVS to perform 3D reconstruction to generate the original point cloud [2,3]. The general process is: The scale-invariant feature transformation（SIFT）algorithm is used for feature extraction [4], and the kd-tree model is used for feature point matching. For each image matching pair, the fundamental matrix is estimated, and the matching pair is optimized and improved by the RANSAC algorithm. Nonlinear least squares are used to calculate the camera parameters and scene geometric information, a sparse 3D point cloud is generated. Finally, a dense point cloud is generated for each image cluster acquired by CMVS through the PMVS patch reconstruction method. All these procedures were assembled in the free and academic software, VisualSFM.

Install

Visual SFM+CMVS/PMVS (http://ccwu.me/vsfm/)

Run

1) Import multi-view image sequences.

2) Perform feature extraction and feature matching.

3) Sparse reconstruction.

4) Dense reconstruction.

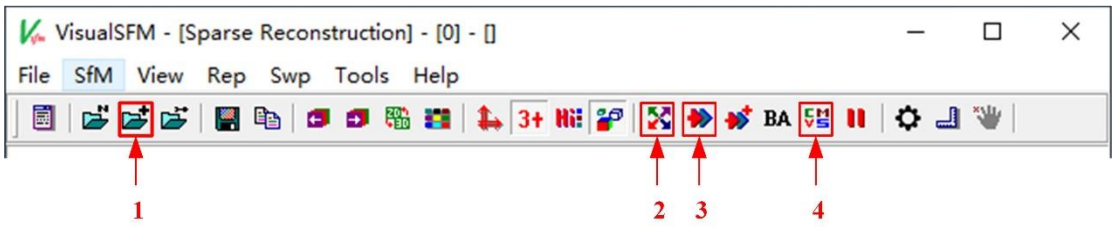


Figure S4. Software interface and its use process

Attention

1) In the later stage of tomato plants, two layers of the plants were reconstructed separately using the two sets of images.

2) The reconstruction efficiency of this step is closely related to the number of images and computer performance. Usually, more images would lead to more time consumption. However, reconstruction would fail if there were no enough overlap between adjacent images.

Part 3 Point cloud preprocessing

Install

CloudCompare (http://www.danielgm.net/cc/release/)

Run

1) Import the point cloud into CloudCompare, and segment the plant point cloud and the marker point cloud by using .

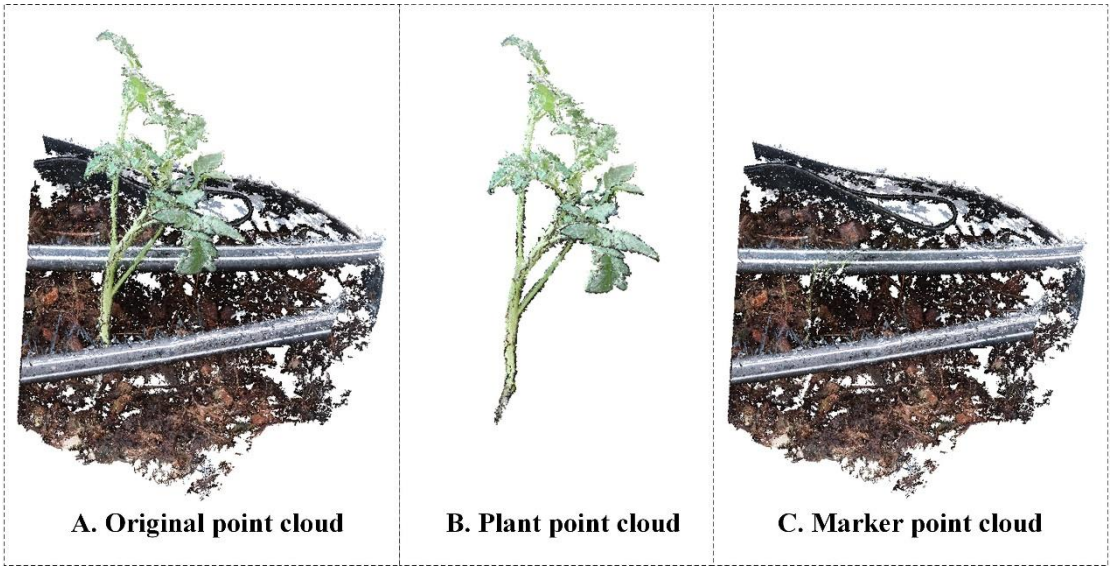


Figure S5. Original point cloud and the result of manual segmentation

2) Use the point list picking function, select the two endpoints of the marker separately to

calculate the Euclidean distance between two points (Figure S6).

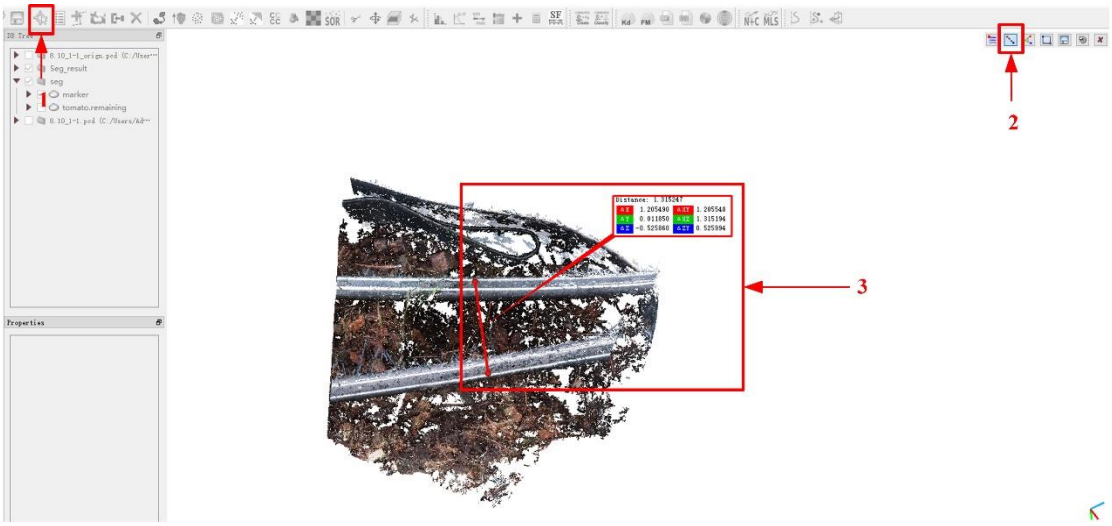


Figure S6. Calculate the Euclidean distance between two points

3) For two sets of point clouds, the alignment algorithm is used for point cloud registration to produce a complete point cloud for one individual tomato plant. Then merge point clouds and remove duplicate points.

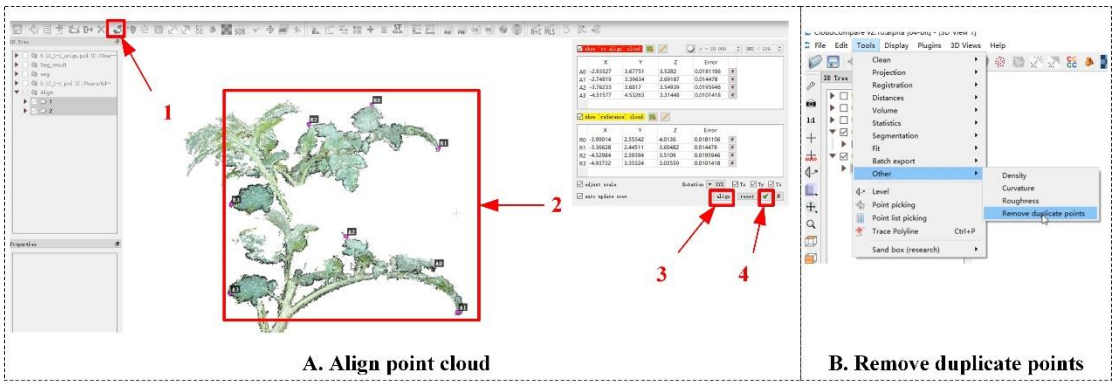


Figure S7. The alignment algorithm

4) Use the  tool to segment the point cloud into leaf point clouds (excluding petiole part) and stalk point clouds.

5) The stem point cloud (including the petiole) was down-sampled using  . This helps to improve skeletonization efficiency.

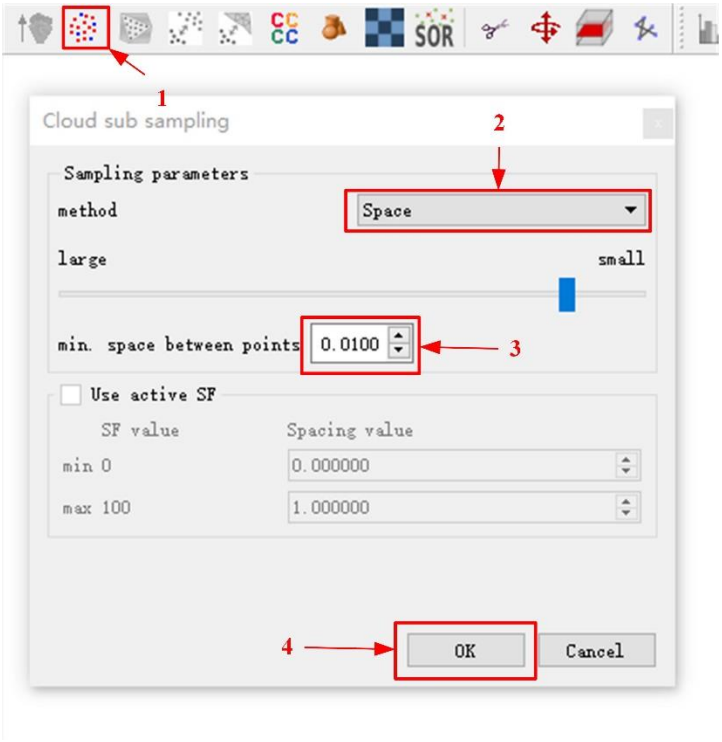


Figure S8. Point cloud down-sampled

Attention

1) In order to estimate the accuracy of leaf area and leaf length, all leaf points are also clustered into individual ones, which is very fast because almost one cluster of point clouds represent one individual leaf. The segmented stalk point cloud contains not only the petiole, but also a part of the leaf tip as shown in Figure S9C.

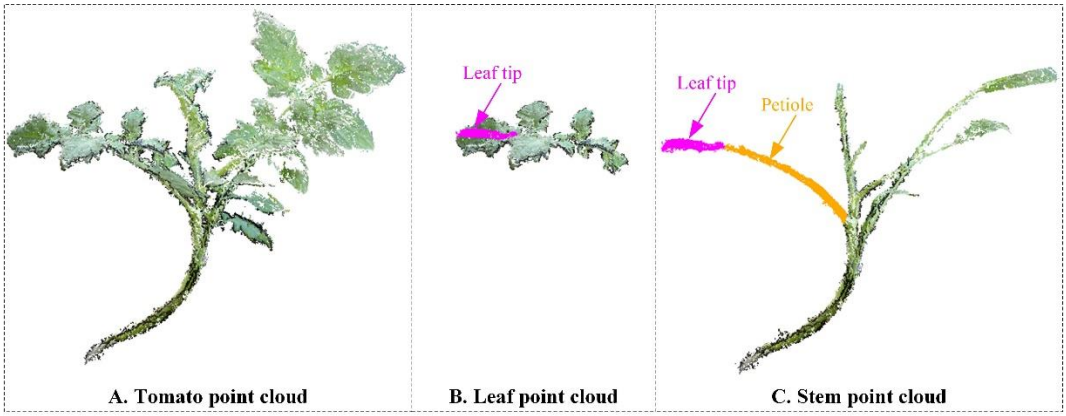


Figure S9. Tomato point cloud and its organ point cloud

2) In the later stage, due to self-occlusion and old leaf abscission, a small number of leaves will be missing. In order to record the phenotypic trait parameters accurately regarding one leaf at different growth time points, it is necessary to manually record the number of the missing leaf. The numbering sequence is 0, 1, 2…along the direction of the main stem

from bottom to top.

3) Alignment is implemented to produce a complete point cloud about the entire fruit. The point cloud registration of the reference plant, the process as shown in Figure S10.

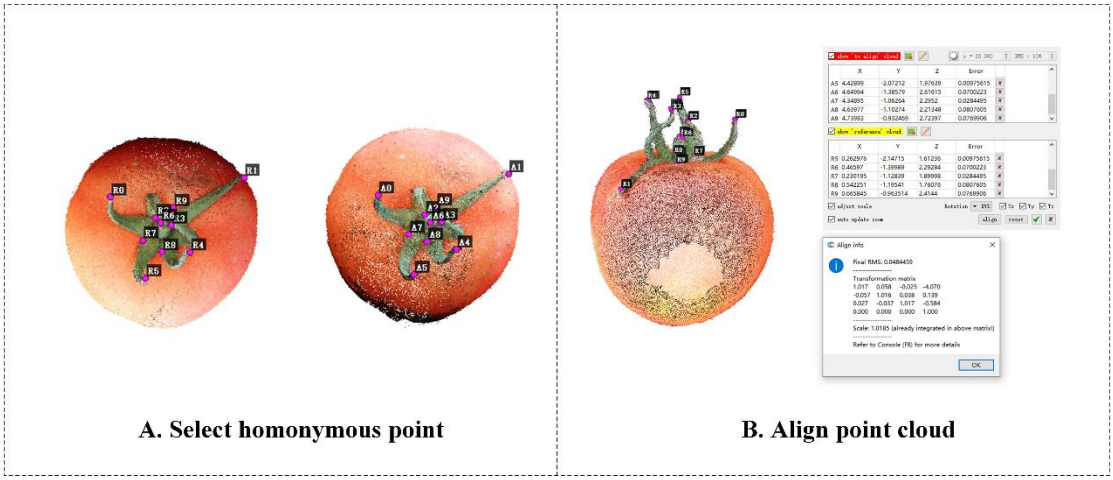


Figure S10. The registration of a tomato fruit

Part 4 Skeletonization

We use an *L1-medial* skeleton algorithm [1] to derive a skeleton for stalk. The code should be able to run without any setup, and it only takes 1-2 minutes to process 10k points. The code is implemented in MATLAB R2018a (https://www.mathworks.com/products/matlab.html). More details about the code can be found in:

https://github.com/ataiya/cloudcontr/blob/master/matlab/readme.txt .

Install

Download Zipped Source Code by the link as follows, including skeletonized source code and KDTree Library.

(https://github.com/ataiya/skeletonization/raw/master/Downloads/cloudcontr_2.0.1.zip)

(https://github.com/ataiya/skeletonization/raw/master/Downloads/kdtree.rar)

Run and result

1) Unzip kdtree.rar and copy it to cloudcontr_2.0.1\matlab\toolbox

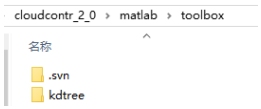


Figure S11. The file path of kdtree

2) Change the Current Folder to the cloudcontr_2.0.1\matlab.

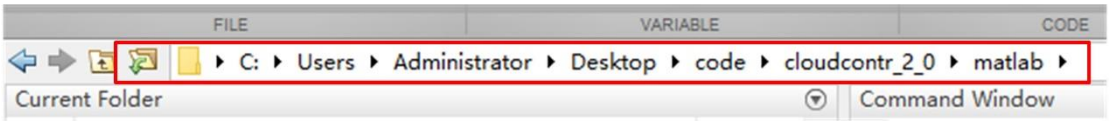


Figure S12. The result of current folder

3) Double-click the eg_skeleton_laplacian_rosa.mlx, change the point cloud reading path and the result saving path, and then click Run, the result will be displayed on the right side.

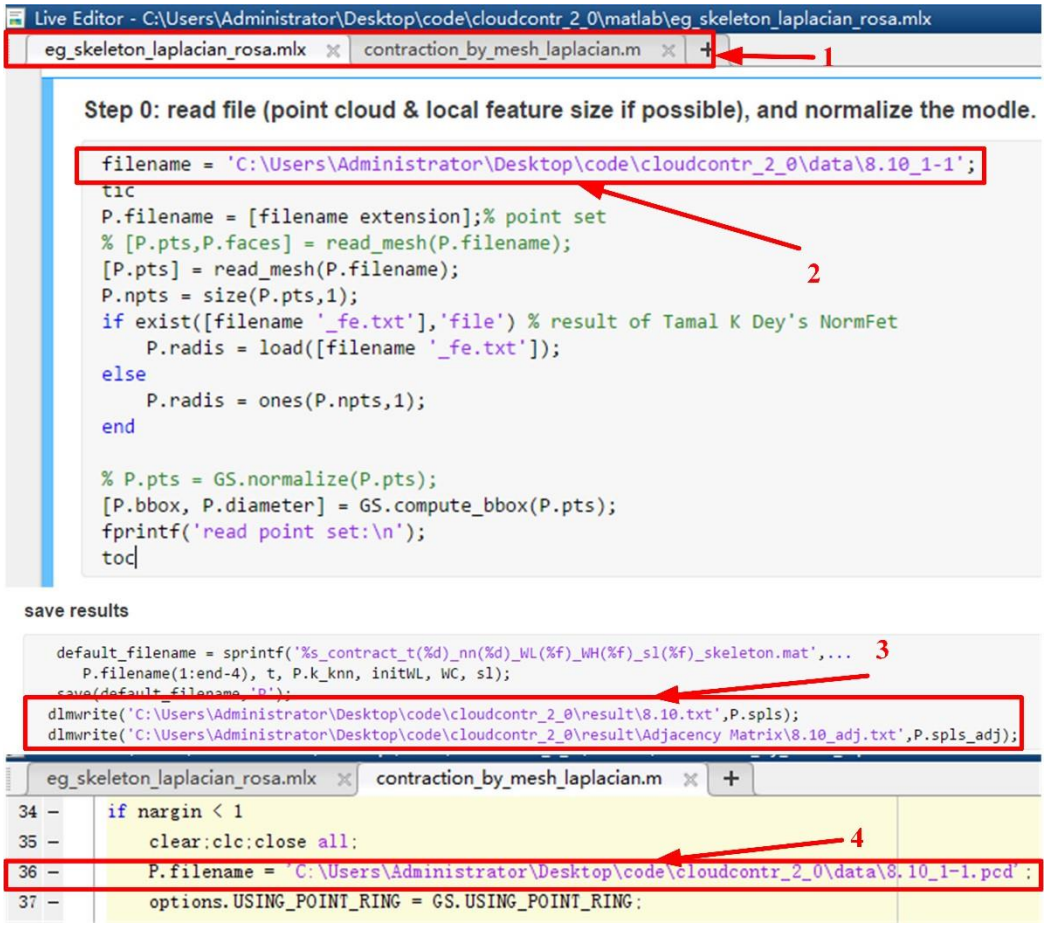


Figure S13. Change the file path

4) In the result, the generated skeleton nodes are saved in the 8.10.txt, and the adjacency matrix is saved in 8.10_adj.txt. (In the pop-up figure: Figure 4: Original point cloud (green) and skeleton (red); Figure 5: skeleton node (pink) and skeleton (red).)

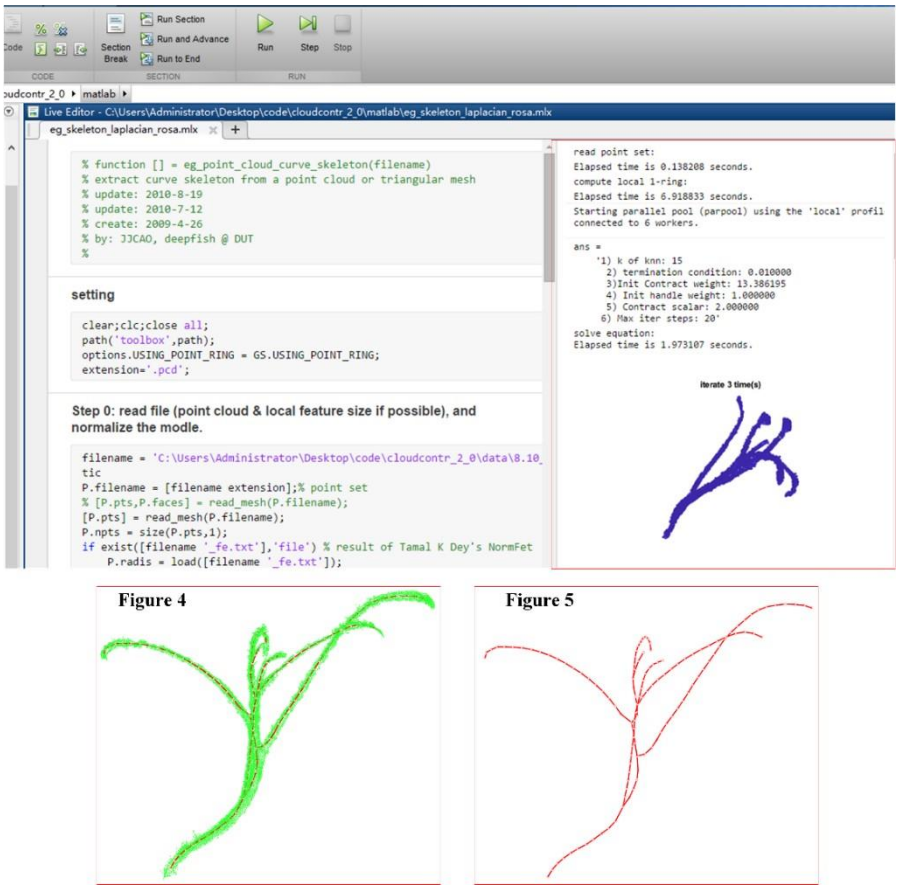


Figure S14. The result of *L*1*-medial* skeleton algorithm

5) Open the skeleton node in CloudCompare, and use the point list picking to view the number information (Index) of the two ends of the main stem as input for subsequent organ segmentation.

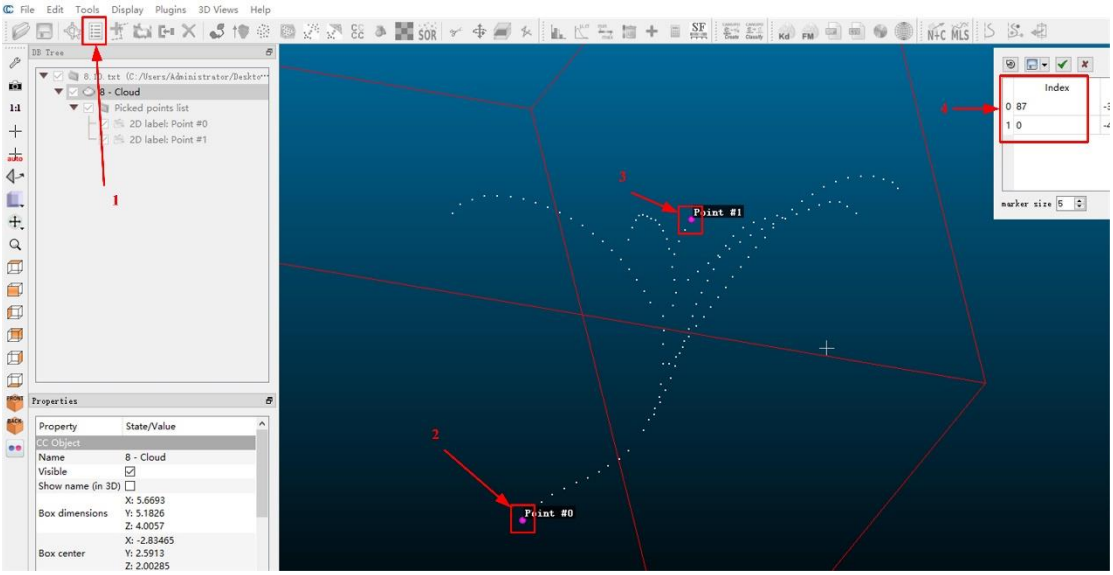


Figure S15. Obtain the number information of the two ends of the stem

Attention

When processing the index of stem skeleton node, the skeleton node near ground is set as the starting point of the stem (Point #0), and another point as the growth point (Point #1), shown as Figure S15.

Part 5 Organ segmentation

This algorithm uses the skeleton node and node adjacency matrix to count the number of adjacency points, and classifies, extracts and marks the adjacency points. Finally, it will realize the segmentation of each internode and each leaf, calculate internode length and leaf length, and rearrange and visualize the skeleton nodes according to the leaf growth law of tomato. Herein, the correspondence of one organ at different time points is built automatically.

The detailed information for the algorithm is as follows. Readers can also skip this part and run the program directly.

The inputs for this algorithm are skeleton points and its adjacency matrix, starting point, growth point, and missing leaf ID. Then the algorithm builds the adjacency relationship between points according to the input. The node types re listed as follows:

1) If the number of adjacent points is greater than three, it is considered as Leaf start point. (The point where the petiole and the stem intersect)
2) The point between leaf star point, stem start point and growth point is the stem node. (The stem point except for the two ends)
3) Except for the main stem node, the adjacent nodes of leaf end points and their subsequent nodes are leaf nodes. (The leaf point except for the two ends)

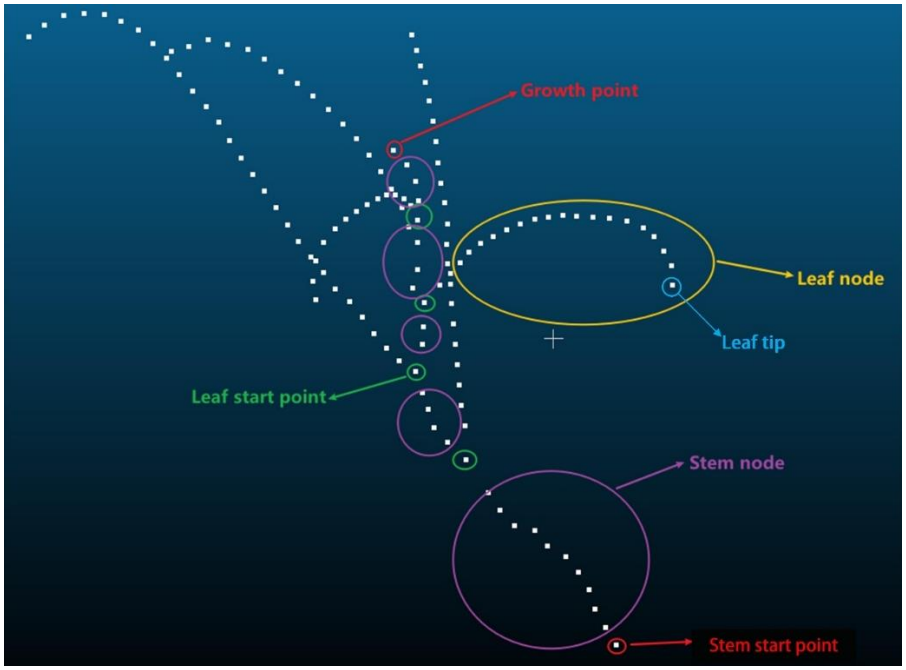


Figure S16. The type of each node

After classifying the points, the length of the stem can be calculated by summing up all the Euclidean distances between two adjacent stem points at the same time.

Then, the algorithm segments each leaf through the adjacency relationship between the leaf start points and the leaf nodes. The leaf length is calculated in the same way as stem length. Finally, reorder the ID of leaves.

The leaf ID starts with 1. Since the order of leaves regarding one plant was set from bottom to top along the tomato plant, except for artificial trimming and leaf abscission, remains unchanged, the leaf ID would keep unchanged in our algorithm. The algorithm can give a same ID to one leaf at different periods. Through the steps described above, the algorithm outputs the segmented stems, leaves and their length. After processing all the skeletonized points regarding one tomato plants at different time points, the correspondence relationship of one individual leaf at different time point were built as well.

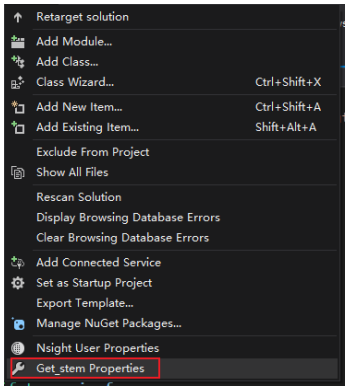
This software can only be used for academic purpose. For any questions on this software, please contact with author, the email address is: 574849348@qq.com.

Install

The program is a C + + program written with Visual Studio 2019 on Windows10-64bit. Some functions for automatic node detection are encapsulated in Io_ frame_ Matrix.lib, so you need to add this library as follow:

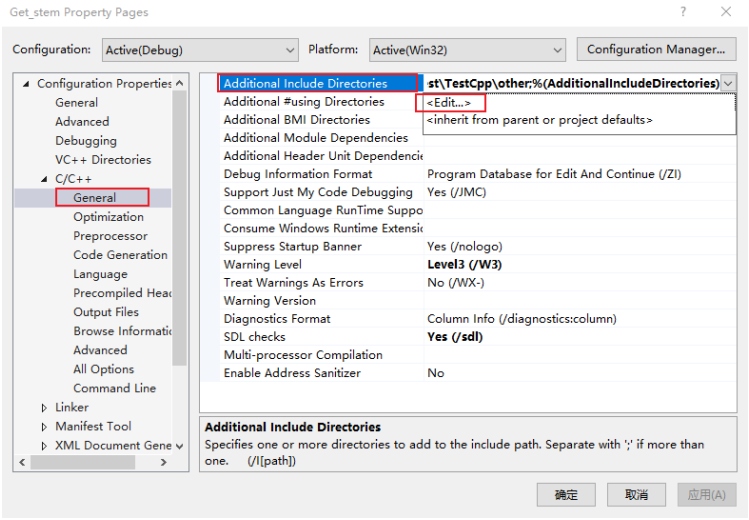
1) Double click Get_ Stem.sln open the vs project file;

2) Click Project and open Settings;

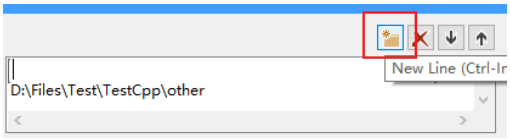


3) Open C/C++ in the Settings page and select General;

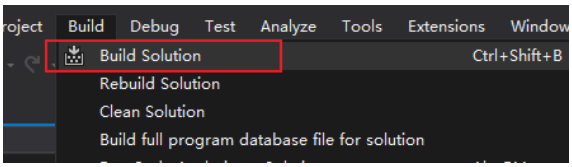
4) Find the Additional Include Directories, click Edit;



5) Click New Line to add a new Additional Include Directories and select the directory where the IO_frame_matrix resides;



6) Click Build to build the solution;



7) Open the Debug directory and run Get_ stem.exe.

Run and result

You only need to provide a couple of inputs in the console window to get the results:

1) The file path of the skeleton point

2) The file path of the adjacency matrix

3) The sequence numbers of the start point in the skeleton point file

4) The sequence numbers of the growth point in the skeleton point file

Due to leaf trimming and leaf abscission, some leaves would be missed in the later progress. To solve this problem, we record the leaf IDs to build leaf correspondences automatically. If there are missing leaves in the node, enter the leaf ID from bottom to top along tomato plants. If there are multiple leaves, enter multiple values, separated by spaces. If there are no missing blades, enter - 1.

Then press Enter key after input. If the input is correct, all the results will be printed out in the console window. (The console will print all the results.)

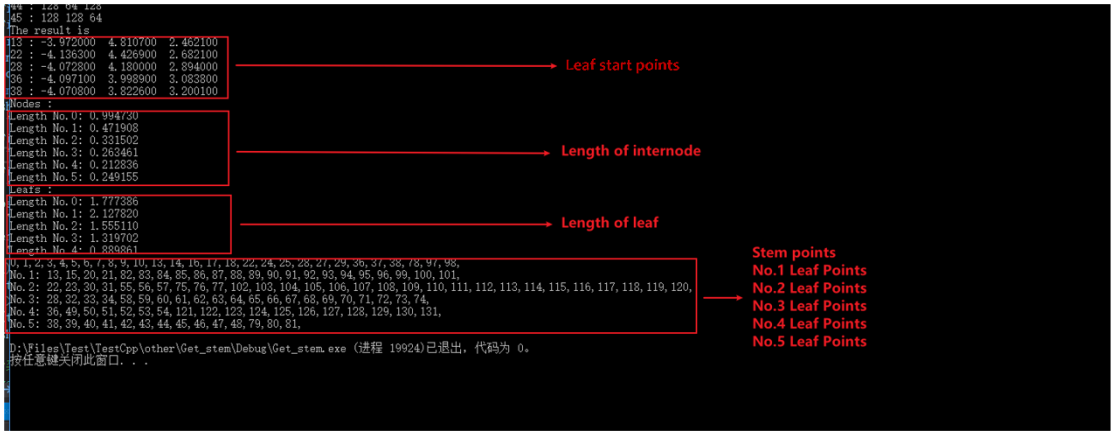


Figure S17. The result of organ segmentation

1) The result is: output the main stem node and its coordinates

2) Nodes: internode lengths along main stem

3) Leaf: leaf length

Finally, the main stem points and all leaf points are printed out in the console window.

You can check the output files in your project. This file stores the reordered skeleton points.

You can use any point cloud software that supports out.txt for visualization, or you can directly open it to see whether the data itself meets your expectations. In out.txt, each point is affiliated with nine values, which are listed as follows:

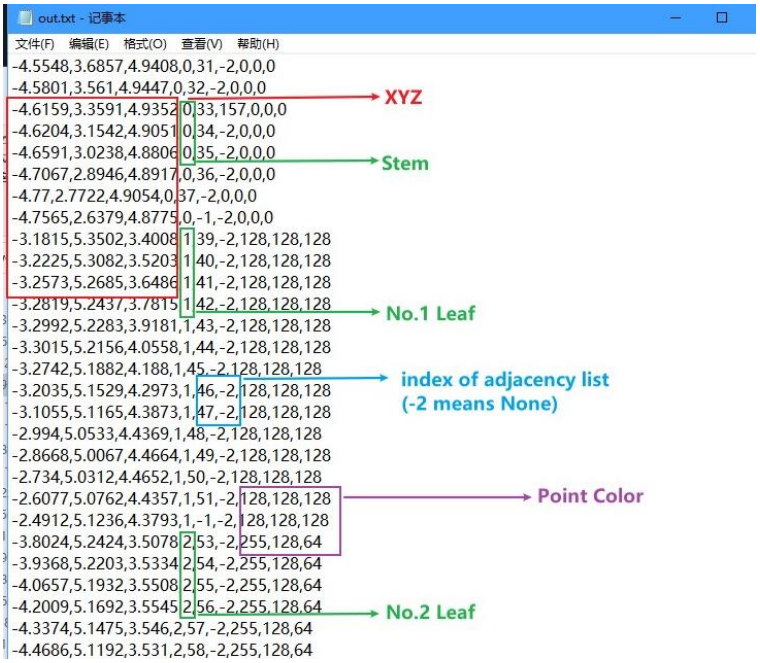


Figure S18. The result of reordered skeleton points

Current point information:

1) Value 1: point cloud coordinate X

2) Value 2: point cloud coordinate Y

3) Value 3: point cloud coordinate Z

4) Value 4: point type (0 for main stem, y ~ x for y ~ x leaf)

The next point information:

If the current point is the stem node or leaf node:

5) Value 5: the index of the next point (-1 represents the leaf end point or growth point)

6) Value 6: -2

If the current point is the leaf start point:

5) Value 5: the index of the next point on the stem (-1 represents growth point)

6) Value 6: the index of the next point on the petiole (-1 represents the leaf end point)

Color information of current point:

7) Value 7: Red value ranges from 0 to 255

8) Value 8: Green value ranges from 0 to 255

9) Value 9: Glue value ranges from 0 to 255

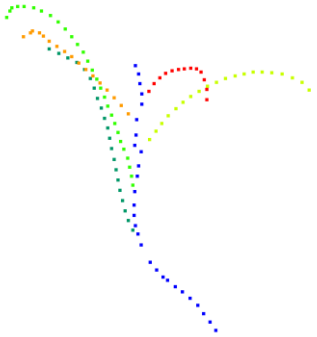


Figure S19. The visualization of reordered skeleton points

We tested the skeletons of 56 tomato plants, and the average number of skeletons was about 120. The results show that visualization results of the extracted stems and leaves regarding one plant at different time points.

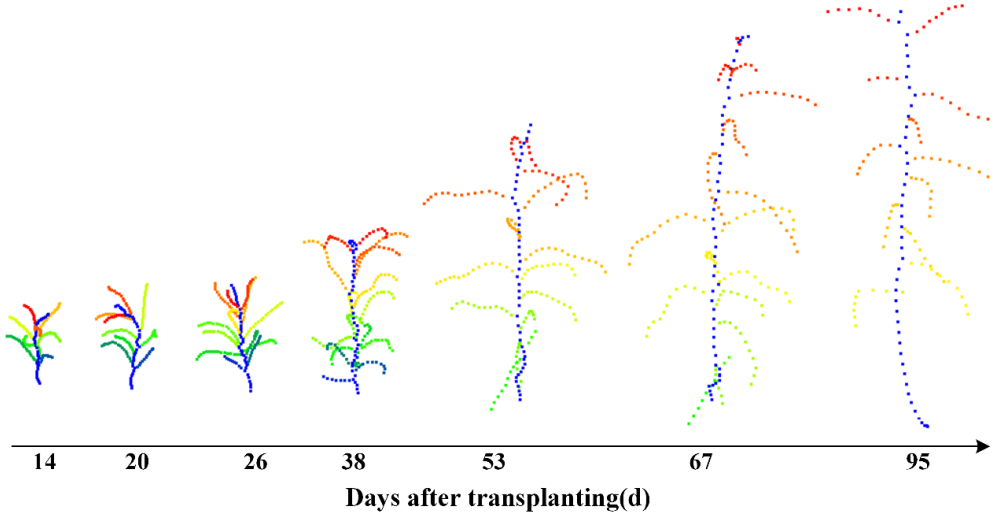


Figure S20. The visualization results of organ segmentation in the whole growth stages

Part 6 Phenotypic parameter extraction

Algorithm details

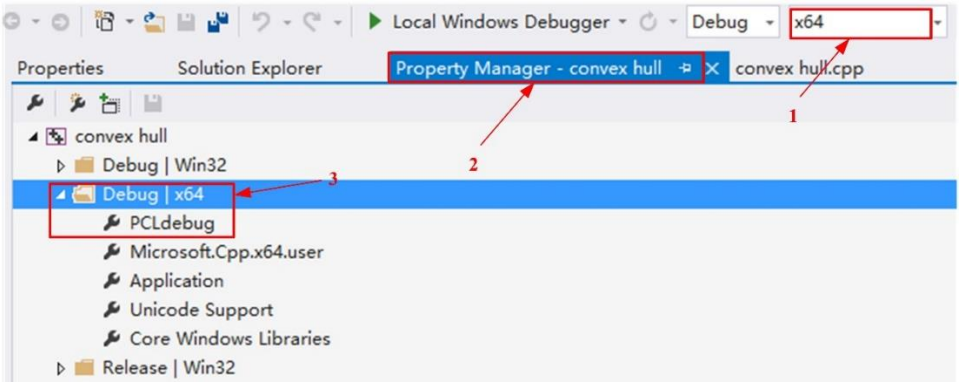
1) Volume estimation

The smallest convex polygon that generates a three-dimensional point cloud is the convex hull. The spatial distribution of the three-dimensional point cloud directly affects the generation of the convex hull, so this study uses the volume of the convex hull as the plant volume. At the same time, the functions for batch calculation, saving and output are realized. The whole algorithm is realized in Visual Studio 2019 (https://visualstudio.microsoft.com/downloads/)

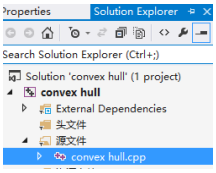
and PCL-1.12.0- win64 (https://github.com/PointCloudLibrary/pcl/releases/). Before running this algorithm, the users need to complete the PCL configuration in Visual Studio.

Prepare

i) Open convex hull.sln. Change the solution platforms to x64. Open Property Manager, add PCLdebug.props to Debug | x64.



ii) Open solution explorer, add convex hull.cpp to source file.



Run and result

i) Modify the path, including the input path of the point cloud, the out path of the saved convex hull.

ii) Runs. The convex hull model in the result can be viewed in CloudCompare, and the output of the convex hull volume (area) is in volume.txt (area.txt). Since this experiment is a reconstructed point cloud, it needs to be scaled to real size by the scale conversion.

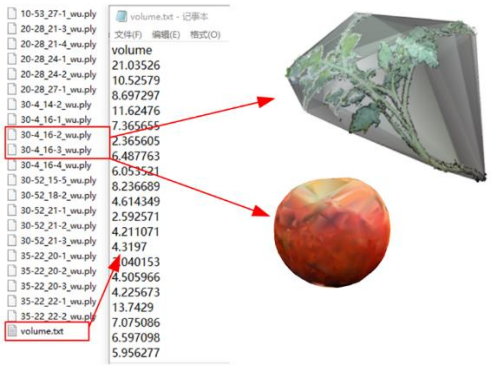
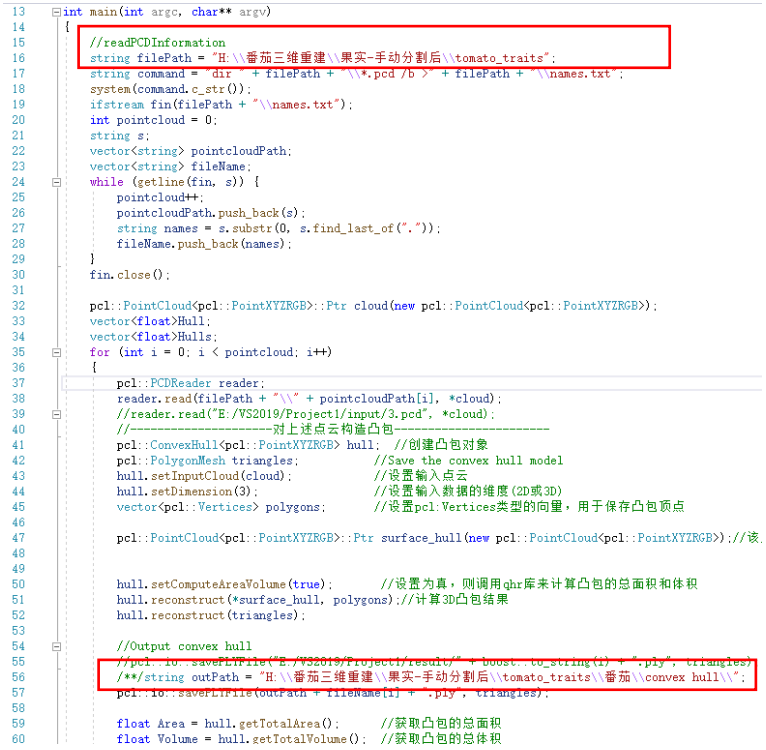


Figure S21. The result of volume and area estimation

The core source code is:



```
int main(int argc, char** argv)
{
    //readPCDInformation
    string filePath = "H:\\番茄三维重建\\果实-手动分割后\\tomato_traits";
    string command = "dir " + filePath + "\\*.pcd /b > " + filePath + "\\names.txt";
    system(command.c_str());
    ifstream fin(filePath + "\\names.txt");
    int pointcloud = 0;
    string s;
    vector<string> pointcloudPath;
    vector<string> fileName;
    while (getline(fin, s)) {
        pointcloud++;
        pointcloudPath.push_back(s);
        string names = s.substr(0, s.find_last_of("."));
        fileName.push_back(names);
    }
    fin.close();

    pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZRGB>);
    vector<float>Hull;
    vector<float>Hulls;
    for (int i = 0; i < pointcloud; i++)
    {
        pcl::PCDReader reader;
        reader.read(filePath + "\\" + pointcloudPath[i], *cloud);
        //reader.read("E:/VS2019/Project1/input/3.pcd", *cloud);
        //----------------------对上述点云构造凸包----------------------
        pcl::ConvexHull<pcl::PointXYZRGB> hull;  //创建凸包对象
        pcl::PolygonMesh triangles;             //Save the convex hull model
        hull.setInputCloud(cloud);              //设置输入点云
        hull.setDimension(3);                   //设置输入数据的维度(2D或3D)
        vector<pcl::Vertices> polygons;         //设置pcl::Vertices类型的向量，用于保存凸包顶点

        pcl::PointCloud<pcl::PointXYZRGB>::Ptr surface_hull(new pcl::PointCloud<pcl::PointXYZRGB>);//该


        hull.setComputeAreaVolume(true);        //设置为真，则调用qhr库来计算凸包的总面积和体积
        hull.reconstruct(*surface_hull, polygons);//计算3D凸包结果
        hull.reconstruct(triangles);

        //Output convex hull
        //pcl::io::savePLYFile("E:/VS2019/Project1/result/" + boost::to_string(i) + ".ply", triangles);
        /**/string outPath = "H:\\番茄三维重建\\果实-手动分割后\\tomato_traits\\番茄\\convex hull\\";
        pcl::io::savePLYFile(outPath + fileName[i] + ".ply", triangles);

        float Area = hull.getTotalArea();       //获取凸包的总面积
        float Volume = hull.getTotalVolume();   //获取凸包的总体积
```

Figure S22. The source code of volume and area estimation

Attention

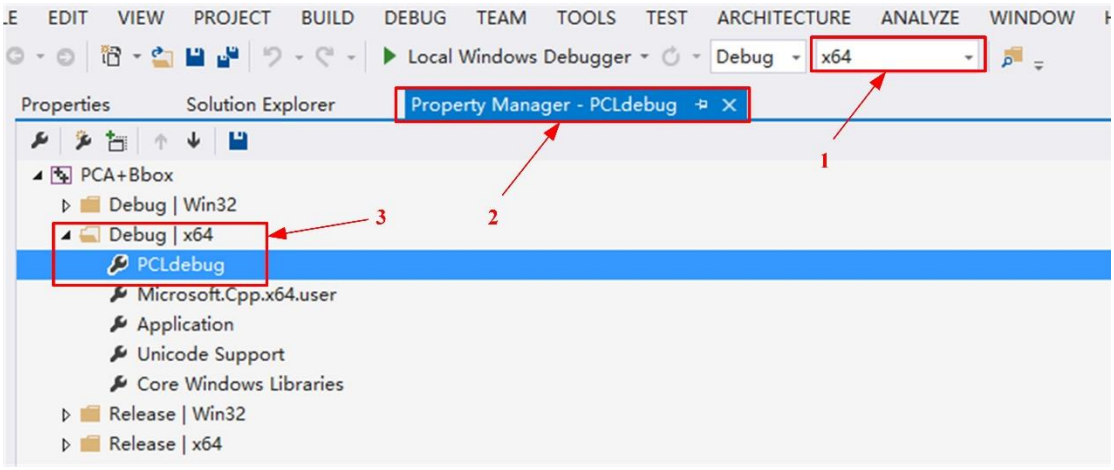
The imported leaf point cloud format must be PCD format. If not, please perform format conversion before this procedure.

2) Calculation of plant length, plant width and plant height

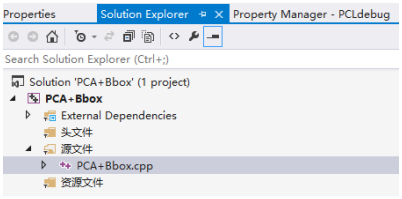
Use PCA (Principal Component Analysis) to analyze the main direction of the point cloud to generate the bounding box of the point cloud. The length of the side parallel to the stem in the bounding box is taken as the plant height, the larger side of the remaining two sides is taken as the plant length, and the smaller one is taken as the plant width. The entire algorithm is implemented by Visual Studio 2019 (https://visualstudio.microsoft.com/downloads/) and PCL-1.12.0-win64 (https://github.com/PointCloudLibrary/pcl/releases), so users need complete the PCL configuration before running. After that, users only need to change the input path of point cloud.

Prepare

i) Open PCA+Bbox.sln. Change the solution platforms to x64. Open Property Manager, add PCLdebug.props to Debug | x64.



ii) Open solution explorer, add PCA+Bbox.cpp to source file.



Run and result

i) Load the point cloud files. Use PCA to generate the centroid and covariance matrix of the point cloud, calculate its eigenvalues and eigenvectors, where the eigenvector is the main direction. The core source code is:

```
pcl::PointCloud<PointType>::Ptr cloud(new pcl::PointCloud<PointType>());
//Input point cloud
pcl::io::loadPCDFile("C:\\Users\\Administrator\\Desktop\\data\\1-1 volume\\input\\2.pcd", *cloud);

//The PCA principal component analysis method is used to obtain the three main directions of the point cloud, obtain the centroid, calculate the covariance,
//obtain the covariance matrix, and obtain the eigenvalue and special feature vector of the covariance matrix. The eigenvector is the main direction.
Eigen::Vector4f pcaCentroid;  //centre of mass
pcl::compute3DCentroid(*cloud, pcaCentroid);
Eigen::Matrix3f covariance;
pcl::computeCovarianceMatrixNormalized(*cloud, pcaCentroid, covariance);//Calculate the normalized covariance matrix

//Calculate the main direction: eigenvectors and eigenvalues
Eigen::SelfAdjointEigenSolver<Eigen::Matrix3f> eigen_solver(covariance, Eigen::ComputeEigenvectors);
Eigen::Matrix3f eigenVectorsPCA = eigen_solver.eigenvectors();
Eigen::Vector3f eigenValuesPCA = eigen_solver.eigenvalues();
//Correct the vertical between the main directions (the feature vector direction:  (e0, e1, e0 x e1) --- note: e0 x e1 = +/- e2)
eigenVectorsPCA.col(2) = eigenVectorsPCA.col(0).cross(eigenVectorsPCA.col(1));
eigenVectorsPCA.col(0) = eigenVectorsPCA.col(1).cross(eigenVectorsPCA.col(2));
eigenVectorsPCA.col(1) = eigenVectorsPCA.col(2).cross(eigenVectorsPCA.col(0));

std::cout << "eigenvectors va(3x1):\n" << eigenValuesPCA << std::endl;
std::cout << "eigenvalues ve(3x3):\n" << eigenVectorsPCA << std::endl;
std::cout << "centre of mass(4x1):\n" << pcaCentroid << std::endl;
```

Change the input path of point cloud

Figure S23. The core source code of calculating eigenvalues and eigenvectors

ii) Point cloud transformation. Using the main direction and center of mass, the input point cloud is transformed to the origin of the reference coordinate system, and the main direction is aligned with the coordinate axis of the reference coordinate system to establish a bounding box of the point cloud. The core source code is:

```
//Go to the reference coordinate system and align the main direction of the point cloud with the coordinate axis of the reference coordinate system
Eigen::Matrix4f tm = Eigen::Matrix4f::Identity();
Eigen::Matrix4f tm_inv = Eigen::Matrix4f::Identity();
tm.block<3, 3>(0, 0) = eigenVectorsPCA.transpose();   //R. [R^(-1) = R^T]
tm.block<3, 1>(0, 3) = -1.0f * (eigenVectorsPCA.transpose()) *(pcaCentroid.head<3>());//  -R*t  [t^(-1) = -R^T * t]
tm_inv = tm.inverse();

std::cout << "Transformation matrix tm(4x4):\n" << tm << std::endl;
std::cout << "Inverse transformation matrix tm'(4x4):\n" << tm_inv << std::endl;

pcl::PointCloud<PointType>::Ptr transformedCloud(new pcl::PointCloud<PointType>);//Transformed point cloud
pcl::transformPointCloud(*cloud, *transformedCloud, tm);
```

Figure S24. The core source code of point cloud transformation

iii) Visualization of the bounding box. Call the getMinMax3D function to obtain the maximum and minimum values of the transformed point cloud along the coordinate axis, and calculate the bounding box size, which are the length, width, and height of the tomato plant, respectively. The core source code is：

```
PointType min_p1, max_p1;
Eigen::Vector3f c1, c;
pcl::getMinMax3D(*transformedCloud, min_p1, max_p1);//Boundary value along the coordinate axis of the reference coordinate system
c1 = 0.5f*(min_p1.getVector3fMap() + max_p1.getVector3fMap());//centre

std::cout << "centre c1(3x1):\n" << c1 << std::endl;

Eigen::Affine3f tm_inv_aff(tm_inv);
pcl::transformPoint(c1, c, tm_inv_aff);

Eigen::Vector3f whd, whd1;
whd1 = max_p1.getVector3fMap() - min_p1.getVector3fMap();
whd = whd1;
float sc1 = (whd1(0) + whd1(1) + whd1(2)) / 3;  //Point cloud average scale, used to set the size of the main direction arrow

std::cout << "distance_x=" << whd1(0) << endl;
std::cout << "distance_y=" << whd1(1) << endl;
std::cout << "distance_z=" << whd1(2) << endl;
std::cout << "scale1=" << sc1 << endl;

//The transformation relationship from the reference coordinate system to the main direction coordinate system
const Eigen::Quaternionf bboxQ1(Eigen::Quaternionf::Identity());
const Eigen::Vector3f    bboxT1(c1);

//The main direction of the point cloud transformed to the origin
PointType op;
op.x = 0.0;
op.y = 0.0;
op.z = 0.0;
Eigen::Vector3f px, py, pz;
Eigen::Affine3f tm_aff(tm);
pcl::transformVector(eigenVectorsPCA.col(0), px, tm_aff);
pcl::transformVector(eigenVectorsPCA.col(1), py, tm_aff);
pcl::transformVector(eigenVectorsPCA.col(2), pz, tm_aff);
PointType pcaX;
pcaX.x = sc1 * px(0);
pcaX.y = sc1 * px(1);
pcaX.z = sc1 * px(2);
PointType pcaY;
pcaY.x = sc1 * py(0);
pcaY.y = sc1 * py(1);
pcaY.z = sc1 * py(2);

PointType pcaZ;
pcaZ.x = sc1 * pz(0);
pcaZ.y = sc1 * pz(1);
pcaZ.z = sc1 * pz(2);


//Visualization transformenCloud
boost::shared_ptr< pcl::visualization::PCLVisualizer > viewer(new pcl::visualization::PCLVisualizer("3D Viewer"));
viewer->setBackgroundColor(1, 1, 1);

viewer->addPointCloud(transformedCloud);
viewer->addCube(bboxT1, bboxQ1, whd1(0), whd1(1), whd1(2), "cube");
viewer->setShapeRenderingProperties(pcl::visualization::PCL_VISUALIZER_REPRESENTATION, pcl::visualization::PCL_VISUALIZER_REPRESENTATION_WIREFRAME, "cube");
viewer->setShapeRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR, 1.0, 0.0, 0.0, "cube");

viewer->addArrow(pcaX, op, 1.0, 0.0, 0.0, false, "arrow_X");
viewer->addArrow(pcaY, op, 0.0, 1.0, 0.0, false, "arrow_Y");
viewer->addArrow(pcaZ, op, 0.0, 0.0, 1.0, false, "arrow_Z");

viewer->addCoordinateSystem(0.5f*sc1);
viewer->setBackgroundColor(1.0, 1.0, 1.0);
while (!viewer->wasStopped())
{
    viewer->spinOnce(100);
}
viewer->removeAllPointClouds();
viewer->removeShape("cube");
viewer->removeAllShapes();
viewer->removeAllCoordinateSystems();
return 0;
```

Figure S25. The source code of visualization of the bounding box

The result is:

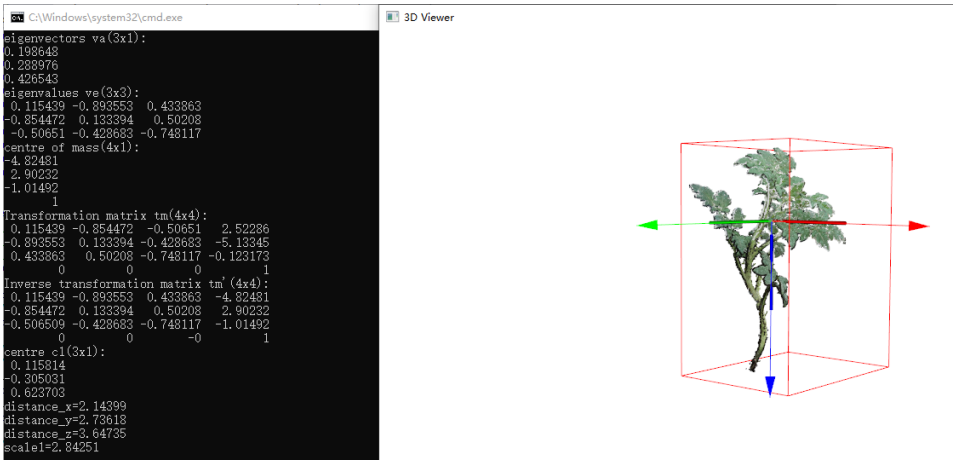


Figure S26. The result of Calculation of plant length, plant width and plant height

Attention

i) In order to ensure the normal operation for the next visualization, users need to close the 3D Viewer first, and then press any key to close the command window.

ii) Perspective projection is used in visualization instead of orthogonal projection, and its characteristic is "near big, far small", which is shown in Figure S27.

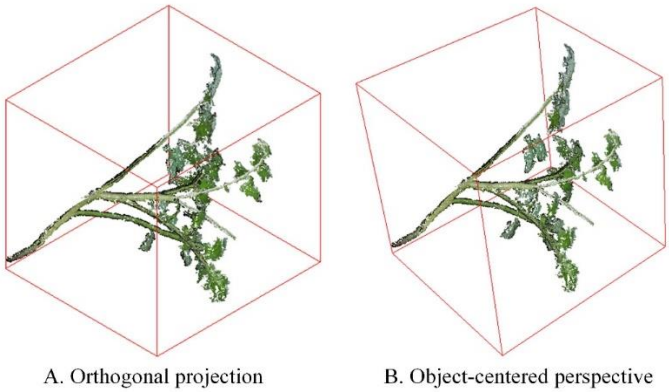


Figure S27. The difference between orthogonal and perspective projections

3) Leaf area and width calculation

Calculate the leaf width by generating the minimum bounding box of the blade, refer to the above steps. Use Wrap command in Geomagic Studio 2013 (64bit) (www.geomagic.com) for point cloud encapsulation to produce the meshed model, which are represented by connected triangles, and then sum up the total area of the triangles to represent the leaf area of each blade. In order to improve efficiency, this experiment uses the recording macro command to record the processing steps, and saves the command as Point cloud slicing.py. Users only need to import this command for batch processing of point clouds.

Run and result

i) Load Point cloud slicing.py to realize the encapsulation of the point cloud and generate the corresponding mesh model.

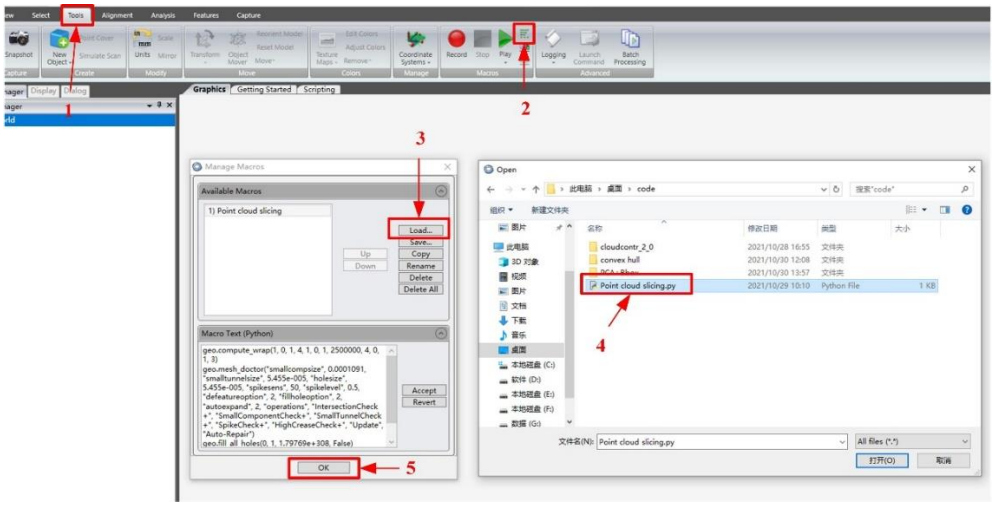


Figure S28. The step of loading Point cloud slicing.py

ii) Batch Processing. Change the input path, select the Import command. Users can choose whether to save the mesh model, the default format is WRP (format can be changed).

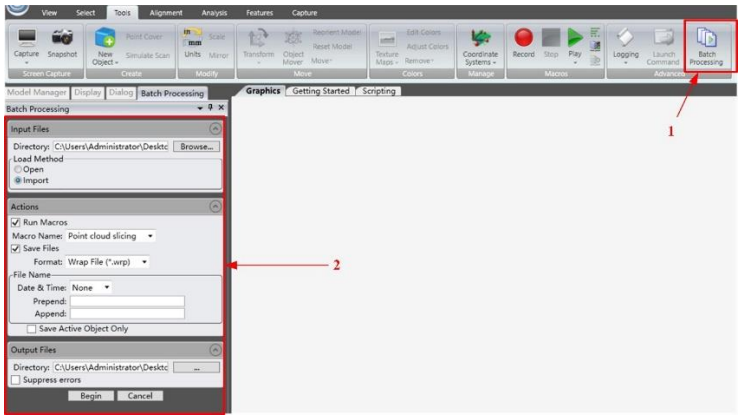


Figure S29. The step of batch processing

iii) Select the mesh model, compute surface area by Compute in Analysis command. The right figure is the encapsulated grid and zoom in for visualization.

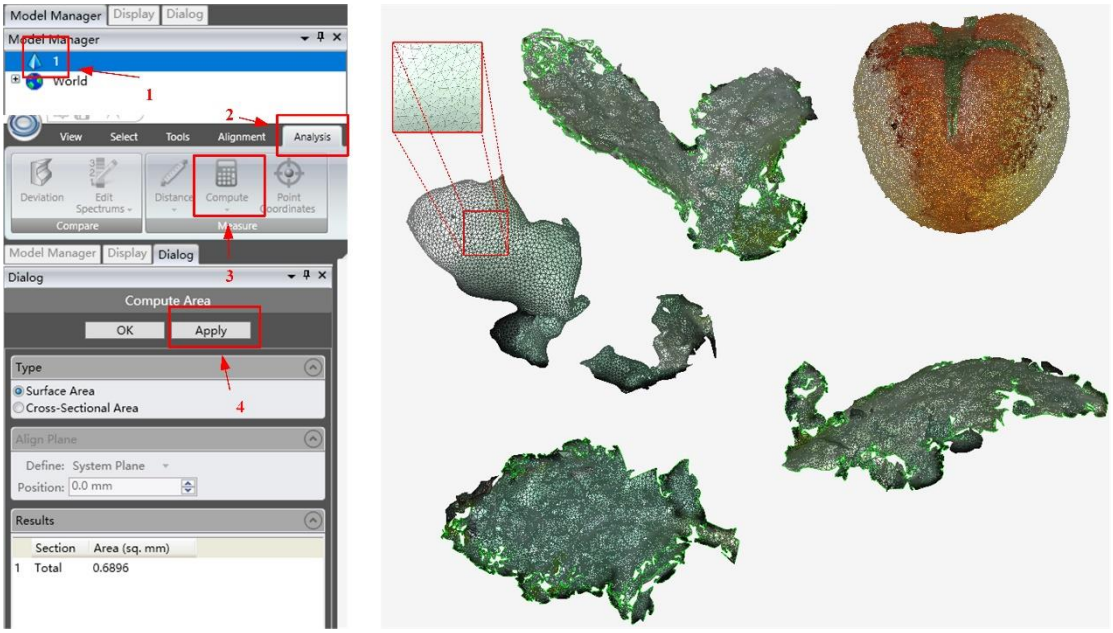


Figure S30. The step of computing leaf area

Attention

i) The imported leaf point cloud format must be PLY format. If not, please perform format conversion before this procedure.

ii) To estimate the leaf area accurately, a mesh model for the 3D points representing one leaf is generated, and the summation of the area of all the meshes account for leaf area.

4) Transverse and vertical diameters calculation

Firstly, the RANSAC [5] algorithm is used to fit the tabletop on which the tomato fruit is placed, and the plane equation of the tabletop is obtained. Then use the Rodrigue rotation formula to rotate the normal vector of the plane equation to the x-axis direction to obtain the rotation matrix R. Finally, the rotation matrix R is applied to the tomato fruit point cloud, take the maximum distance on the x-axis as the vertical diameter, take the maximum distance on the YOZ plane as the transverse diameter. The entire algorithm is implemented by Visual Studio 2019 and PCL-1.10.0-win64, so users need complete the PCL configuration before running.

Run and result

i) Open random_sample_consensus.sln, and modify the number of loops n and the paths. Including the input path of the point cloud, the output paths of the point cloud and the normal vector of the plane equation.

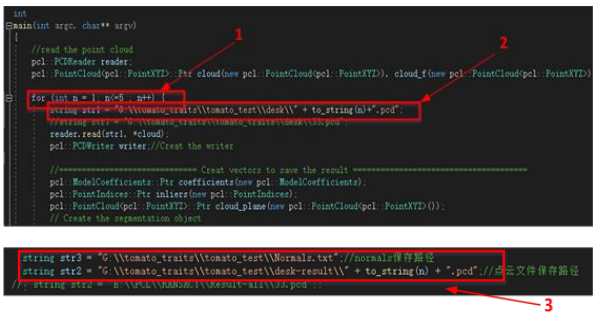


Figure S31. The step of modifying

ii) Run random_sample_consensus.sln. Obtain the PCD file of the fitted plane (Figure S32B, the blue points) and the coefficient file Normals.txt of the plane equation.

A. The coefficient of the plane equation B. The normal vector (n) of the fitting plane

Figure S32. The steps of the plane fitting and normal vector calculation

iii) Open rotationMatrix.sln and modify the paths. Including the input path of the Normals.txt, the output path of the rotation matrix R.

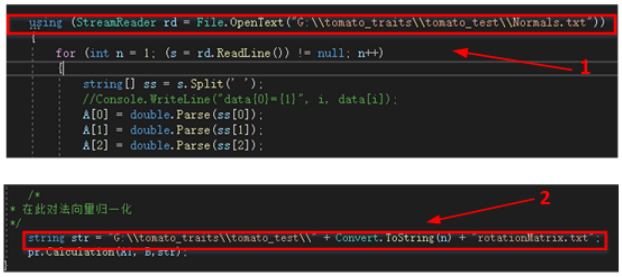
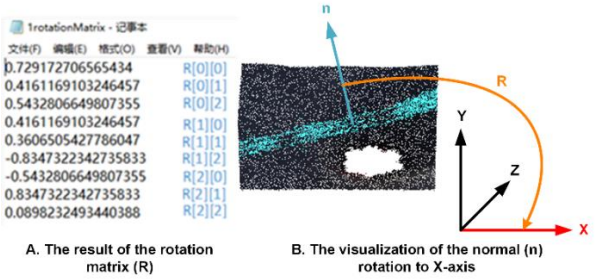


Figure S33. The step of modifying

iv) Run rotationMatrix.sln. Obtain the rotation matrix file rotationMatrix.txt.



A. The result of the rotation matrix (R) B. The visualization of the normal (n) rotation to X-axis

Figure S34. The step of the rotation

v) Open Transform.sln, and modify the number of loops and the paths. The paths including

the input path of the tomato fruit point clouds and the rotationMatrix.txt. The output path of rotated tomato fruit point clouds.

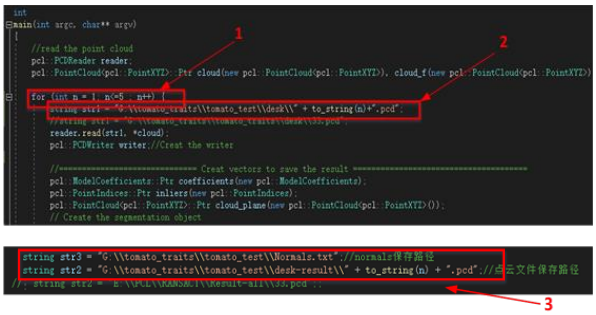
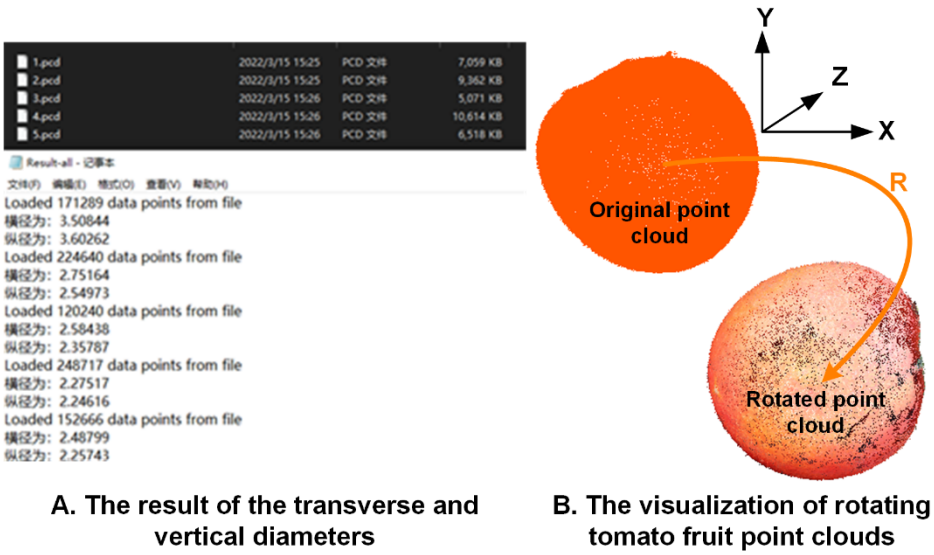


Figure S35. The step of modifying

vi) Run Transform.sln. Obtain the rotated tomato fruit point clouds, and the result of transverse and vertical diameters.



A. The result of the transverse and vertical diameters

B. The visualization of rotating tomato fruit point clouds

Figure S36. The steps of fruit rotation and traits extraction

Attention

The number of loops involved in the algorithm: Consistent with the number of point clouds to be processed.

Reference

1. Huang, H.; Wu, S.H.; Cohen-Or, D.; Gong, M.L.; Zhang, H.; Li, G.Q.; Chen, B.Q. L-1-Medial Skeleton of Point Cloud. *ACM Transactions on Graphics (TOG)* **2013**, *32*, 1-8, doi:10.1145/2461912.2461913.
2. Wu, C.C. Towards Linear-Time Incremental Structure from Motion. In Proceedings of the 3DV-Conference, 2013 International Conference on, Seattle, WA, USA, 29 June 2013 - 01 July 2013; pp. 127-134.
3. Wu, C.C.; Agarwal, S.; Curless, B.; Seitz, S.M. Multicore bundle adjustment. In Proceedings of the Computer Vision & Pattern Recognition, Colorado Springs, CO, USA, 20-25 June 2011; pp. 3057-3064.
4. Lowe, D. Distinctive image features from scaleinvariant keypoints. *International Journal of Computer Vision* **2004**, *60*, 91-110.
5. Fischler, M.A.; Bolles, R.C. Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography. *Commun. ACM* **1981**, *24*, 381-395, doi:10.1145/358669.358692.