Appendix- 3

Poisson Surface Reconstruction

Introduction

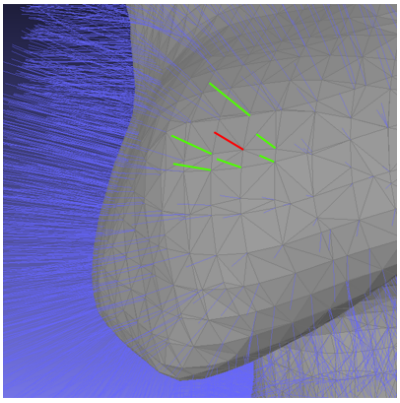
After scanning an object into a point cloud, we want to create a mesh from this points. There are different approaches for creating a mesh, for example:

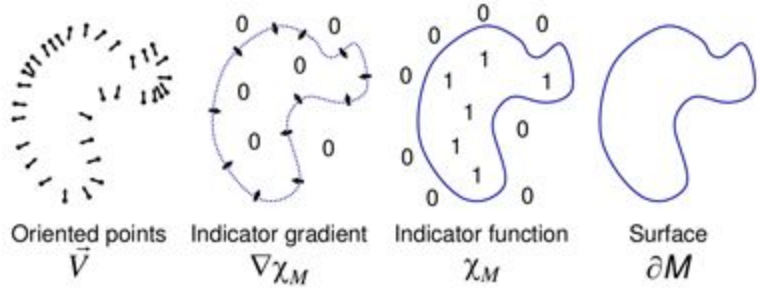
- Poisson Surface Reconstruction
- Ball Pivoting
- Power Crust

Because we want to get a watertight and smooth surface we decided to use the “Poisson Surface Reconstruction”. This approach was introduced by Michael Kazhdan and is used for example in the mesh processing software “MeshLab” as well as in the CGAL (Computational Geometry Algorithms Library).

Poisson Surface Reconstruction: How it Works

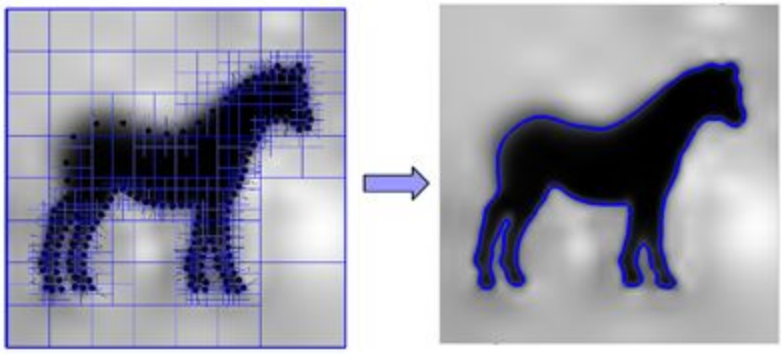
Picture 1: Extrapolate the normals





Picture 2: Extracting the iso surface

Source: “Poisson Surface Reconstruction”,Kazhdan et.al. SIGGRAPH 2006



Picture 3: Extraction of the Surface using "Marching Cubes"

Source: "Unconstrained Isosurface Extraction on Arbitrary Octrees",Kazhdan et.al. SIGGRAPH 2007

At first we need oriented normals for all of the points included in the scanned point cloud.

Therefor we use the "vcglib" that provides a nice algorithm for the extrapolation of the normals. The algorithm itself can be found in the file "normal_extrapolation.h" located in the folder "\vcglib\vcg\space".

The algorithm requires 3 steps:

- Find the tangential planes for approximating the local surface:
- To achieve this the centroid of every vertex is computed as the average of all k-nearest neighbors. Then the corresponding normal is computed by its eigenvector.
- Building the Riemannian graph:
- The Riemannian graph is needed because it creates a structure where for every vertex the edges to his k-nearest neighbors are included what we need for the next step.
- Using the "minimum spanning tree" algorithm from Kruskal on this Riemannian graph to weight the normals what results in different length for each of them.

Or in other words: the algorithm computes the normals as a weighted product over the k-nearest neighbors. So we get a very smooth model. In picture 1 we will try to illustrate this, where the red marked normal is computed using the six nearest neighbors of it.

Assuming that the scanned points form the surface of the scanned object, the isosurface of the model is approximated from the normalfield that is in relation to the gradient of the indicator function describing the isosurface.

This problem is considered as the solution of a poisson problem.

Picture 2 illustrates this approach for an 2D example:

At first the points with the oriented normals are taken and the gradient of the indicator function is approximated through the normalfield. After that the indicator function, which is zero everywhere except near the surface, is derived from this gradient and the surface of the object can be extracted.

For this extraction the “marching cubes algorithm” is used to build an octree data structure for the representation of the surface. As seen in picture 3 the marching cubes algorithm divides the point cloud into a voxel grid by marching through the point cloud and analyzing which points define the isosurface of our object. By detecting which edges of the voxel are intersected by the model’s isosurface the algorithm creates the triangles of the mesh.

The Poisson Surface Reconstruction Parameters

There are several parameters available that affect the result of the remeshing:

- **Depth:** tree-depth that is used for the reconstruction. default: 8
- **SolverDivide:** specifies depth at which a block Gauss-Seidel solver is used to solve Laplacian equation. default: 8
- **IsoDivide:** specifies the depth at which a block iso-surface extractor should be used to extract the iso-surface. default: 8
- **SamplesPerNode:** specifies the minimum number of sample points that should fall within an octree node as the octree construction is adapted to sampling density. For noise-free data: [1.0, 5.0], noisy data: [15.0, 20.0]
- **Scale:** ratio between the diameter of the cube used for reconstruction and the diameter of the samples’ bounding cube. default: 1.25
- **Offset:** an hacked offset value. if set to 1 there is no offset.

- **Confidence:** enabling this flag tells the reconstructor to use the size of the normals as confidence information. When the flag is not enabled, all normals are normalized to have unit-length prior to reconstruction.

Especially the parameters **depth** and **samples per node** have a great influence on the generated mesh.

- The higher the value for the octree-depth is chosen the more detailed results you will get. This is because the deeper the marching cubes algorithm goes the finer gets the granularity of the voxelgrid. On the other hand with noisy data (like our scanned point clouds) you keep vertices in the generated mesh that are outliners but the algorithm doesn't detect them as such. So a low value (maybe between 5 and 7) provides a smoothing but you will lose detail. The higher the depth-value the higher is the resulting amount of vertices of the generated mesh.
- The samples per node parameter defines how many points the marching cubes algorithm puts into one node of the resulting octree. A high value like 10 means that the algorithm takes 10 points and puts them into the node of the octree. If you have noisy data a high sample per node value provides a smoothing with loss of detail while a low value (between 1.0 and 5.0) keeps the detail level high. A high value reduces the resulting count of vertices while a low value remains them high.