```
#include <CGAL/trace.h>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/Surface_mesh_default_triangulation_3.h>
#include <CGAL/make_surface_mesh.h>
#include <CGAL/Implicit_surface_3.h>
#include <CGAL/IO/facets_in_complex_2_to_triangle_mesh.h>
#include <CGAL/Poisson_reconstruction_function.h>
#include <CGAL/Point_with_normal_3.h>
#include <CGAL/property_map.h>
#include <CGAL/IO/read_xyz_points.h>
#include <CGAL/compute_average_spacing.h>
#include <iostream>
#include <CGAL/Polygon_mesh_processing/distance.h>
#include <vector>
#include <fstream>
using namespace std;
// Types
typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef Kernel::FT FT;
typedef Kernel::Point_3 Point;
typedef CGAL::Point_with_normal_3<Kernel> Point_with_normal;
typedef Kernel::Sphere_3 Sphere;
typedef std::vector<Point_with_normal> PointList;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;
typedef CGAL::Poisson_reconstruction_function<Kernel> Poisson_reconstruction_function;
typedef CGAL::Surface_mesh_default_triangulation_3 STr;
typedef CGAL::Surface_mesh_complex_2_in_triangulation_3<STr> C2t3;
typedef CGAL::Implicit_surface_3<Kernel, Poisson_reconstruction_function> Surface_3;
int main(void)
{
// Poisson options
FT sm_angle = 20.0; // Min triangle angle in degrees.
FT sm_radius = 30; // Max triangle size w.r.t. point set average spacing.
FT sm_distance = 0.375; // Surface Approximation error w.r.t. point set average spacing.
```

```
// Reads the point set file in points[].
// Note: read_xyz_points_and_normals() requires an iterator over points
// + property maps to access each point's position and normal.
// The position property map can be omitted here as we use iterators over Point_3 elements.
PointList points;
std::ifstream stream("data/kitten.xyz");
if (!stream ||
!CGAL::read_xyz_points(
stream,
std::back_inserter(points),
CGAL::parameters::normal_map(CGAL::make_normal_of_point_with_normal_map(PointList::
value_type()))))
{
std::cerr << "Error: cannot read file data/kitten.xyz" << std::endl;
std::system("pause");
return EXIT_FAILURE;
}
// Creates implicit function from the read points using the default solver.
// Note: this method requires an iterator over points
// + property maps to access each point's position and normal.
// The position property map can be omitted here as we use iterators over Point_3 elements.
cout << "starting the poisson reconstruction function !!\n";
getchar();
Poisson_reconstruction_function function(points.begin(), points.end(),
CGAL::make_normal_of_point_with_normal_map(PointList::value_type()) );
// Computes the Poisson indicator function f()
// at each vertex of the triangulation.
if ( ! function.compute_implicit_function() )
return EXIT_FAILURE;
// Computes average spacing
FT average_spacing = CGAL::compute_average_spacing<CGAL::Sequential_tag>(points, 6 /*
knn = 1 ring */);
// Gets one point inside the implicit surface
// and computes implicit function bounding sphere radius.
Point inner_point = function.get_inner_point();
Sphere bsphere = function.bounding_sphere();
FT radius = std::sqrt(bsphere.squared_radius());
// Defines the implicit surface: requires defining a
// conservative bounding sphere centered at inner point.
```

```
FT sm_sphere_radius = 5.0 * radius;
FT sm_dichotomy_error = sm_distance*average_spacing/1000.0; // Dichotomy error must be
<< sm_distance

cout << "starting surface reconstruction\n";
getchar();
Surface_3 surface(function,
Sphere(inner_point,sm_sphere_radius*sm_sphere_radius),
sm_dichotomy_error/sm_sphere_radius);
// Defines surface mesh generation criteria

cout << "surface mesh default ceteria ....\n";
getchar();
CGAL::Surface_mesh_default_criteria_3<STr> criteria(sm_angle, // Min triangle angle
(degrees)
sm_radius*average_spacing, // Max triangle size
sm_distance*average_spacing); // Approximation error
// Generates surface mesh with manifold option
STr tr; // 3D Delaunay triangulation for surface mesh generation
C2t3 c2t3(tr); // 2D complex in 3D Delaunay triangulation

cout << "starting mesh surface mesh\n";
getchar();
CGAL::make_surface_mesh(c2t3, // reconstructed mesh
surface, // implicit surface
criteria, // meshing criteria
CGAL::Manifold_with_boundary_tag()); // require manifold mesh
if(tr.number_of_vertices() == 0)
return EXIT_FAILURE;
// saves reconstructed surface mesh

std::ofstream out("kitten_poisson-20-30-0.375.off");
Polyhedron output_mesh;
CGAL::facets_in_complex_2_to_triangle_mesh(c2t3, output_mesh);

cout << "starting saving mesh!!\n";
getchar();
out << output_mesh;
```

```
/// [PMP_distance_snippet]
// computes the approximation error of the reconstruction

cout << "starting saving polygon mesh processing\n\n";
getchar();
double max_dist =
CGAL::Polygon_mesh_processing::approximate_max_distance_to_point_set(output_mesh,
points,
4000);
std::cout << "Max distance to point_set: " << max_dist << std::endl;
/// [PMP_distance_snippet]

return EXIT_SUCCESS;
}
```