Visibility of Point Clouds

Prakash Dhimal
Computational Geometry
George Mason University

Point Clouds

We formally define a point cloud P as follows:

$$P = P1, P2, P3 \ldots Pn$$

- A set of data points in space
  - Have X,Y,Z values
  - represent a 3D shape or feature
  - might contain additional information such as color about each point.
  - Is also called range data

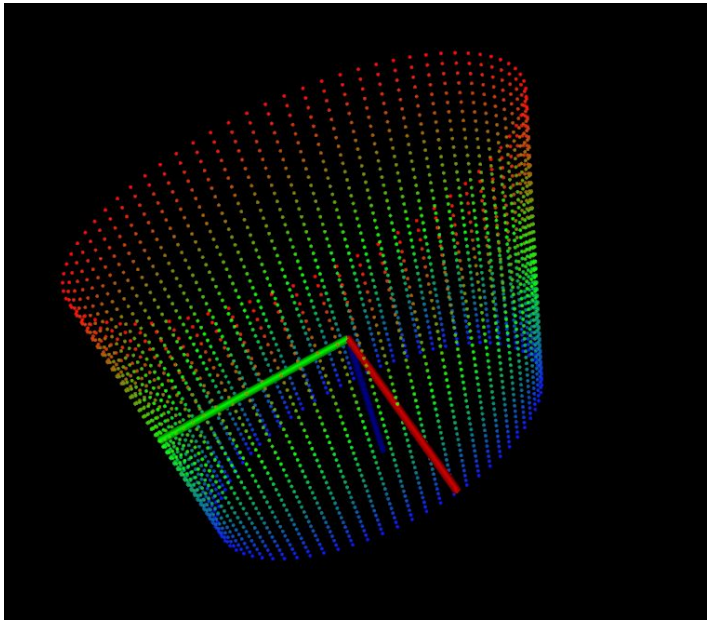


Figure: Simple point cloud generated using the Point Cloud Library

Sources of point cloud

- created by remote sensing and 3D scanning devices such as LiDAR
- The use of LiDAR and similar 3D scanning devices has been increasing.
- If the Cloud is dense enough, we can easily pick up shape of an object.

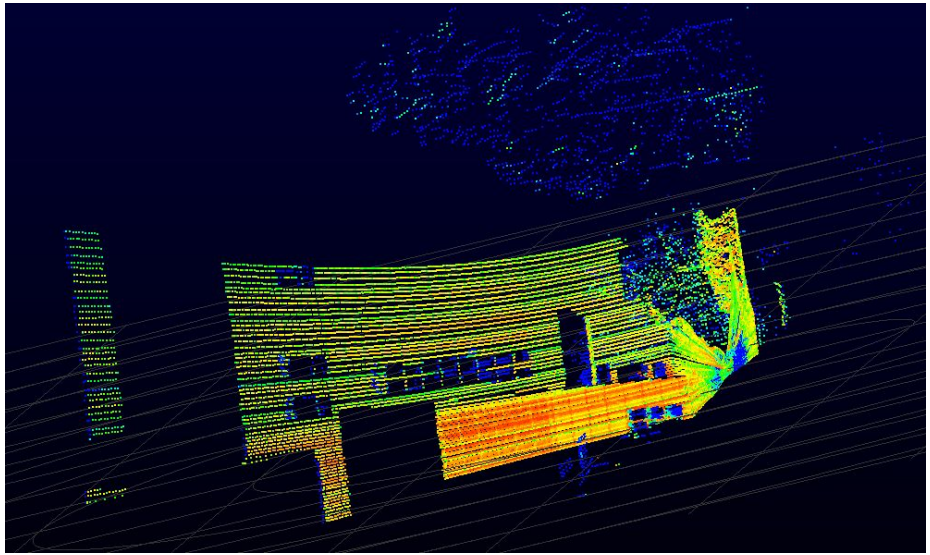


Figure: 3D scan of Johnson Center using Velodyne HDL-64E LIDAR scanner by the Motion and Shape Computing (MASC) Group at George Mason University [5]

Point Cloud Library

- http://pointclouds.org/
- Open project for 2D/3D image and point cloud processing
- PCL modules:
  - I/O,
  - Visualization,
  - Registration
- http://pointclouds.org/media/
  - Links to various 3D models

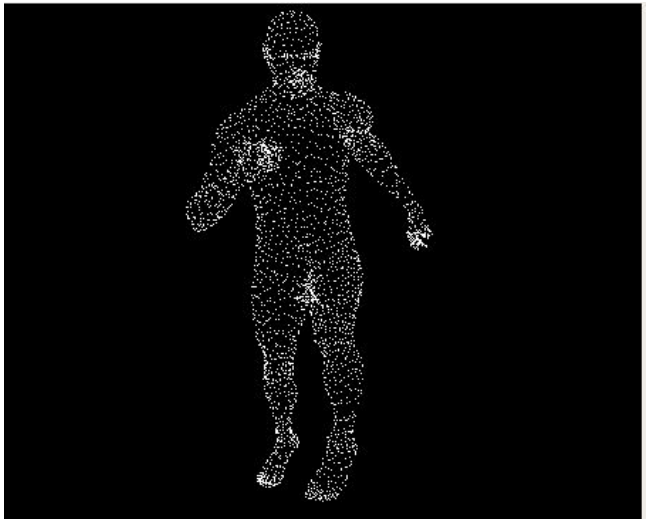


Figure: David visualized using the Point Cloud Library

The problem definition

Given a point cloud P = P1 , P2 , P3 ...Pn and a viewpoint C, is it possible to determine all the points visible from C directly from the point cloud?

- We do not want to reconstruct the surface or estimate normals.
- Here, C is the camera location.

Determining all the points visible from C will give us the “visibility cloud.”

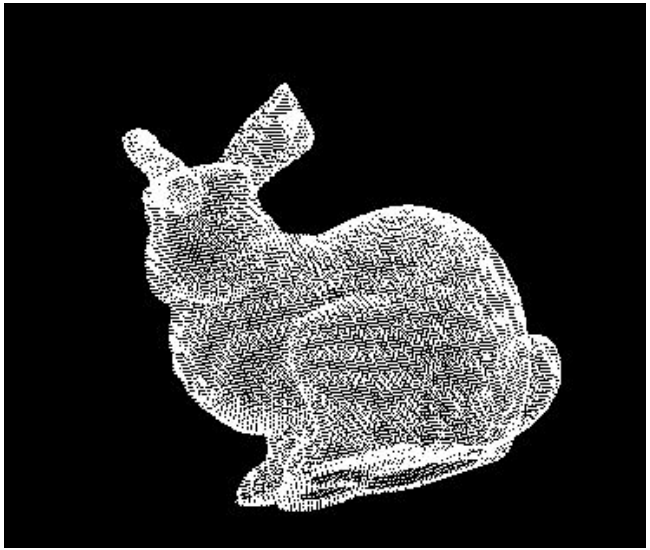


Figure: Is the bunny facing forward or backward?

Visibility

- Determining the visibility of point cloud is an interesting problem in itself
- Visibility can be useful in:
  - visualizing point clouds,
  - view-dependent reconstruction, and
  - shadow casting
- Determining visibility can lead to a more robust registration of the points from multiple point clouds.
- It can provide information about
  - spatial relations
  - potential obstacles in the line of sight between two points in space.

The Hidden Point Removal Operator

Katz et al. [2] introduced an operator called the Hidden Point Removal operator that can be used to determine all of the points that are visible from C.

The operator is simple and it consists of two steps:

1. Spherical Inversion
2. Convex hull construction

This is reducing the problem of visibility to the problem of convex hull construction. All we are doing is a simple transformation of the points and then computing the convex hull

Spherical Inversion

Is the first step in the HPR operator.

Consider a D-dimensional sphere with radius *R*, centered at the origin *C* and includes all points in *P*.

Spherical inversion reflects a point *Pi* ε *P* with respect to the sphere by applying the following equation:

$$f(P_i) = P_i + 2(R - ||P_i||)\frac{P_i}{||P_i|} = \widehat{P}_i$$

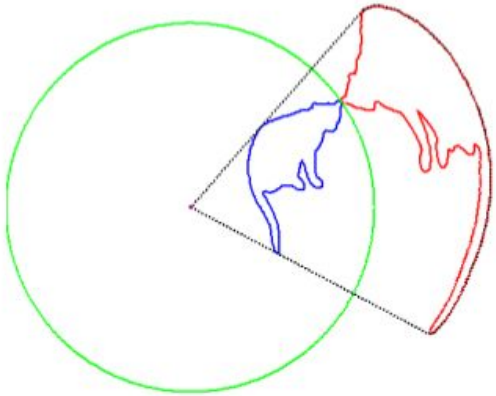


Figure: Spherical flipping, shown in red, of a 2D curve, shown in blue, using a sphere, shown in green [2]

Spherical Inversion

Spherical inversion

- Takes the blue curve (P) bounded in some green circle.
  - The green circle is centered at the origin, or the viewpoint C .
- Maps the curve to its image that you see in red
  - This is the result of the spherical inversion.

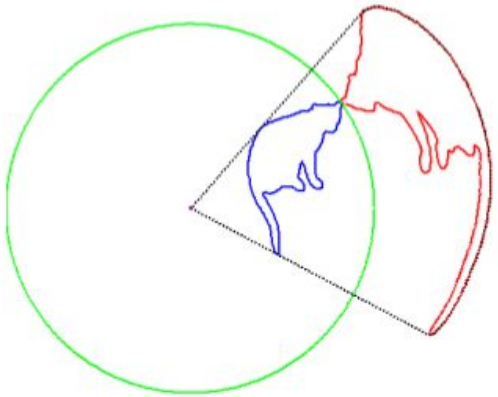


Figure: Spherical flipping, shown in red, of a 2D curve, shown in blue, using a sphere, shown in green [2].

Convex Hull Construction

From the spherical inversion above, we have a new point cloud:

$$\widehat{P} = \hat{P}_1, \hat{P}_2, \hat{P}_3 ... \hat{P}_n$$

The second, and the final step in our HPR operator is to construct the convex hull of the new point cloud *P^* union C, i.e:

$$C_{hull} = \widehat{P} \cup C$$

This takes O(nlogn)

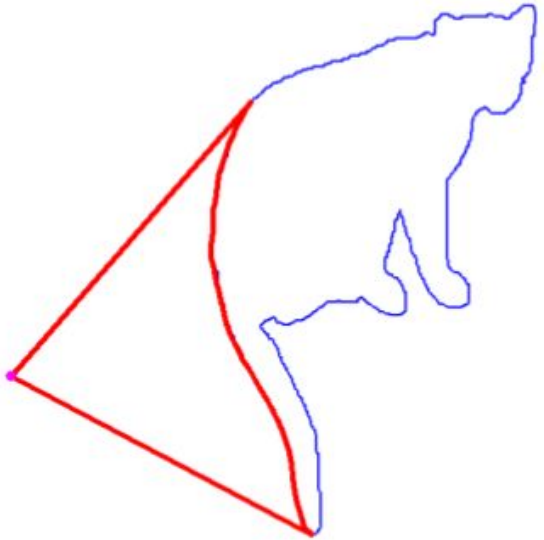


Figure: Convex hull of the result from spherical inversion [2].

The result of HPR operator

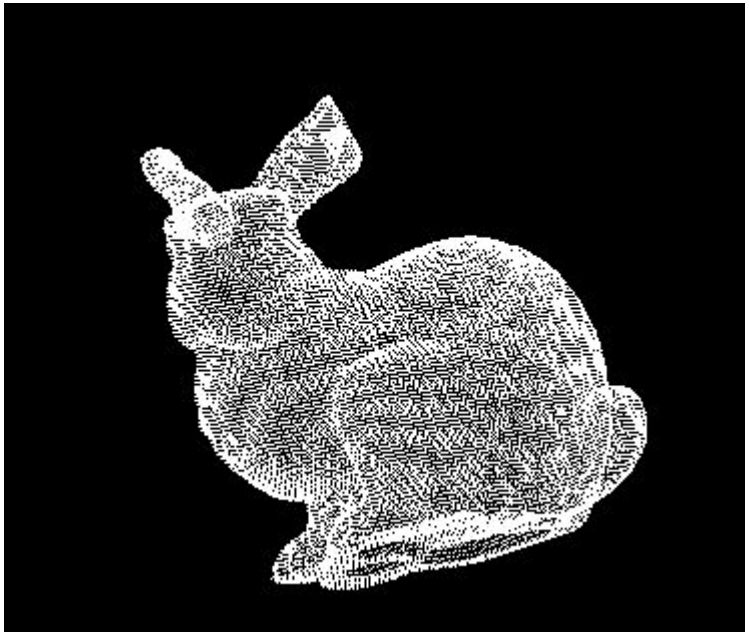


Figure: Which way is the bunny facing?

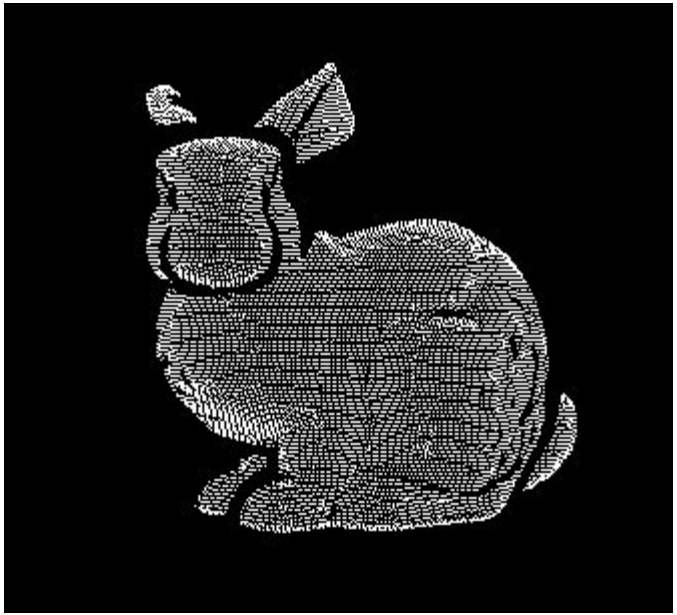


Figure: The bunny is facing forward

The data

The initial plan was to use data obtained from the Velodyne HDL-64E LIDAR scanner by the Motion and Shape Computing (MASC) Group at George Mason University to investigate the visibility of point clouds.

While the Point Cloud Library (PCL) provides capabilities to read and replay/visualize such data, it became clear that determining visibility of a moving point cloud is beyond the scope of this project, as the implementation would need to support more complex data structures such as kinetic data structures to handle the moving point cloud.

The Stanford 3D Scanning repository

- http://graphics.stanford.edu/data/3Dscanrep/
- The 3D models used in this project, including the Stanford Bunny, were scanned with a Cyberware 3030 MS scanner
- Range data
- The 3D models are stored in PLY files.
    - This format was developed at Stanford University.

Georgia Tech

- https://www.cc.gatech.edu/projects/large_models/

Implementation

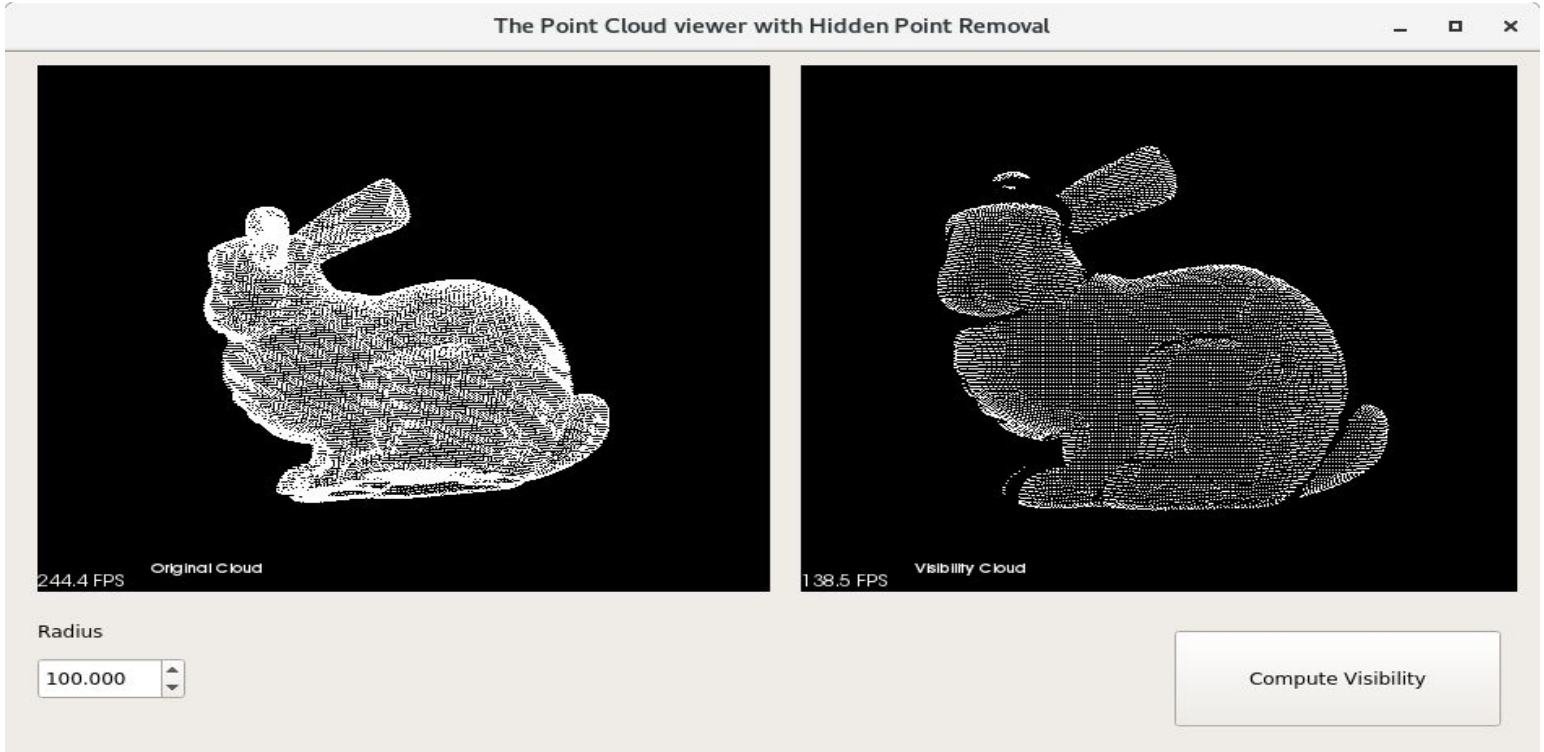
The goal was to implement this operator in C++ so that it can be used with the Point Cloud Library (PCL) for visualization and various other features that PCL provides.

Hidden Point Operator (HPR) was implemented in C++ using

- PCL
- Qt5
- VTK

The program takes as an input a point cloud in PCD or PLY format.
The implementation is available at https://github.com/pdhimal1/HPR

The Point cloud viewer with HPR operator



The Radius R

The radius R relaxes the visibility condition

- more points are considered visible.
- As R increases, more points become visible, until all (truly visible) points become visible by the Hidden Point Removal operator.

However, large R is suitable for dense point clouds, while a small R is suitable for sparse clouds.

What is the optimal radius of the sphere R that marks all the points that are visible as visible?

- There is no one size fit all solution
- Hence, the radius R is left as an input to the HPR operator.

Radius R as an input

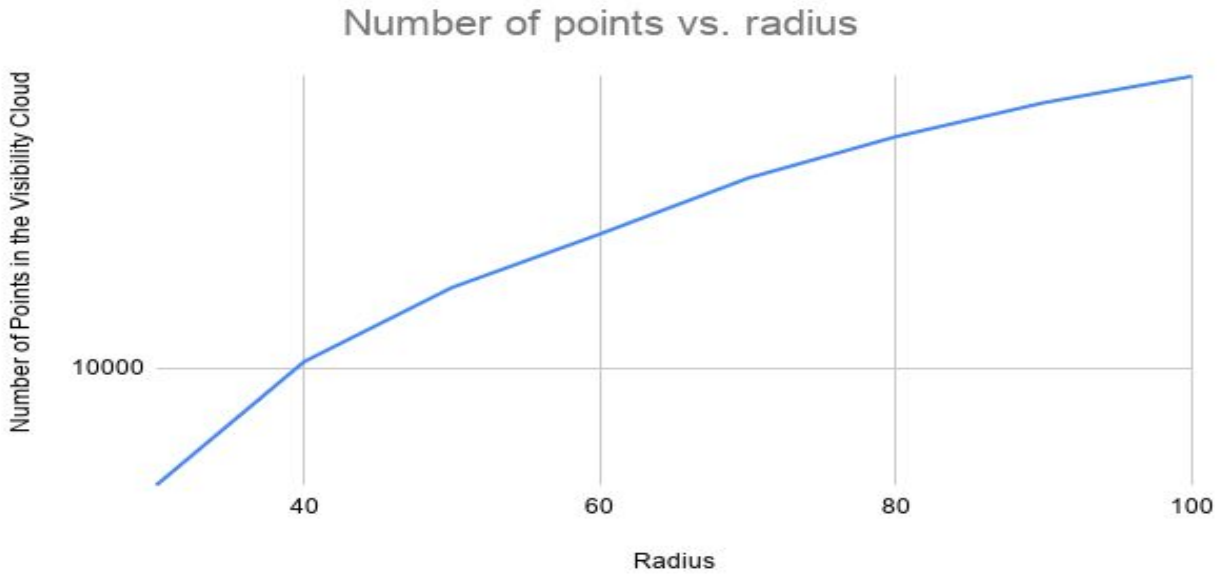


Figure: Number of points marked visible vs. radius on a point cloud of 35947 points. As the radius increases, the number of points marked visible increased. The viewpoint C was not changed.

HPR with radius $R = 10$

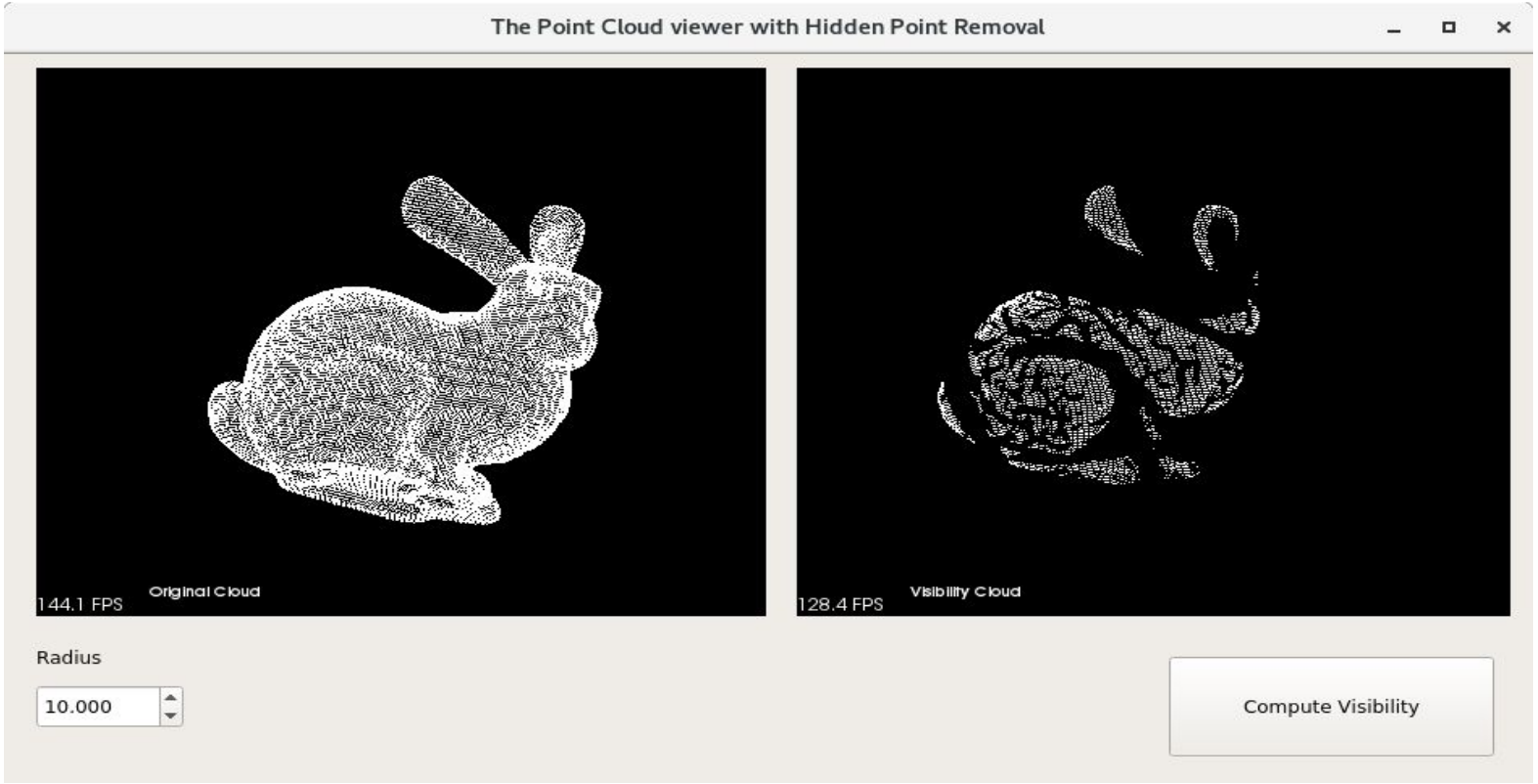


Figure: HPR with radius $R = 10$, Out of 35947, only 4597 points were marked visible

HPR with radius = 100

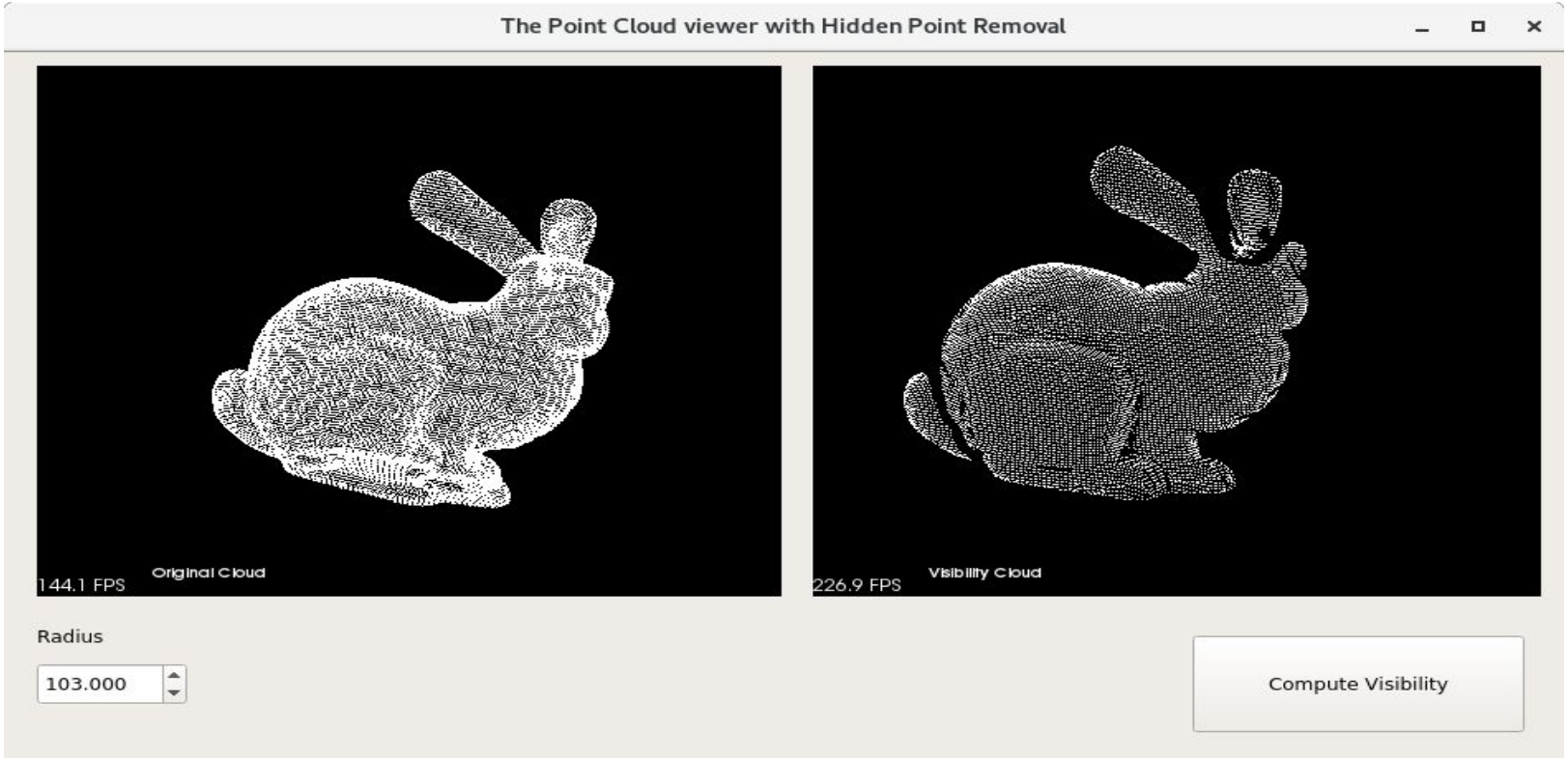


Figure: HPR with radius *R* = *100*, Out of 35947, 10667 points were marked visible

David back view

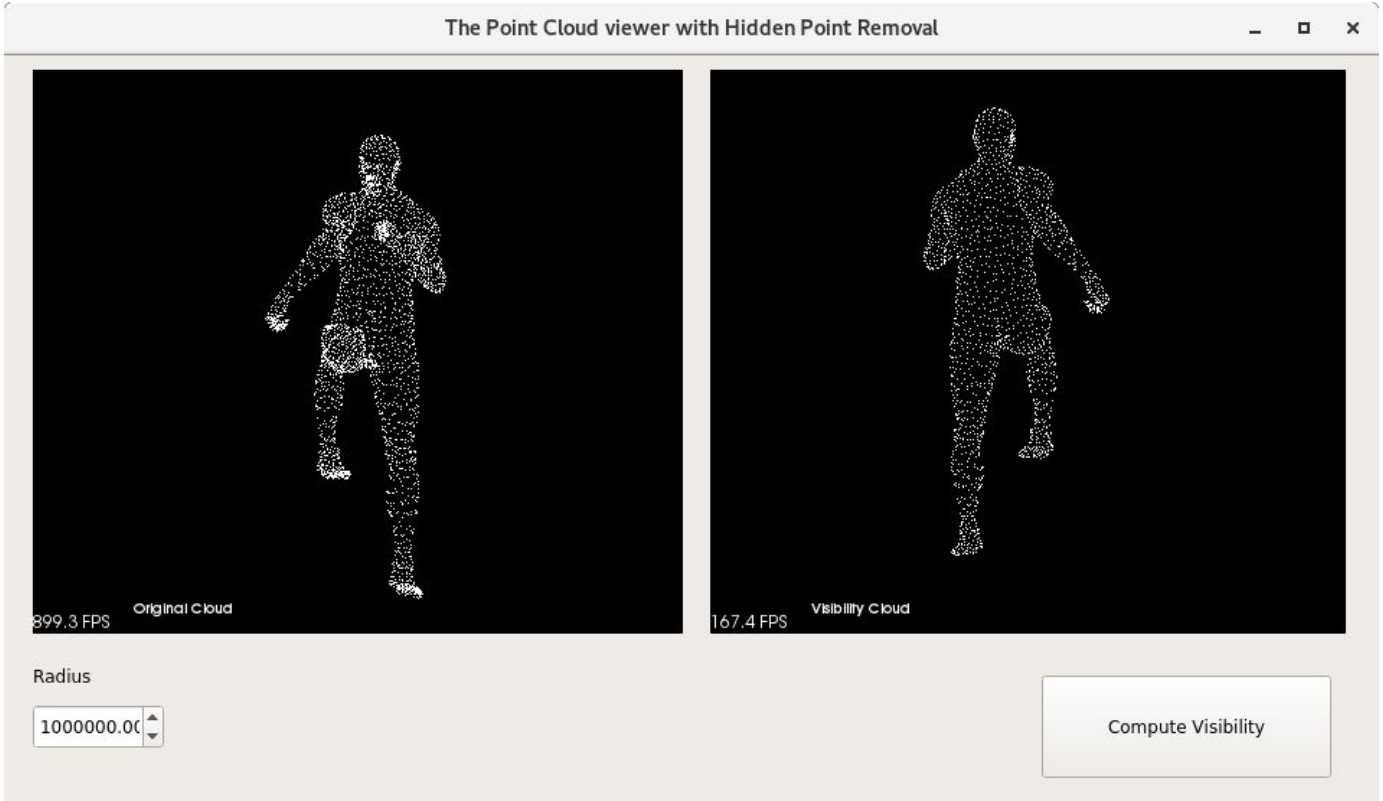


Figure: David back view

David side view

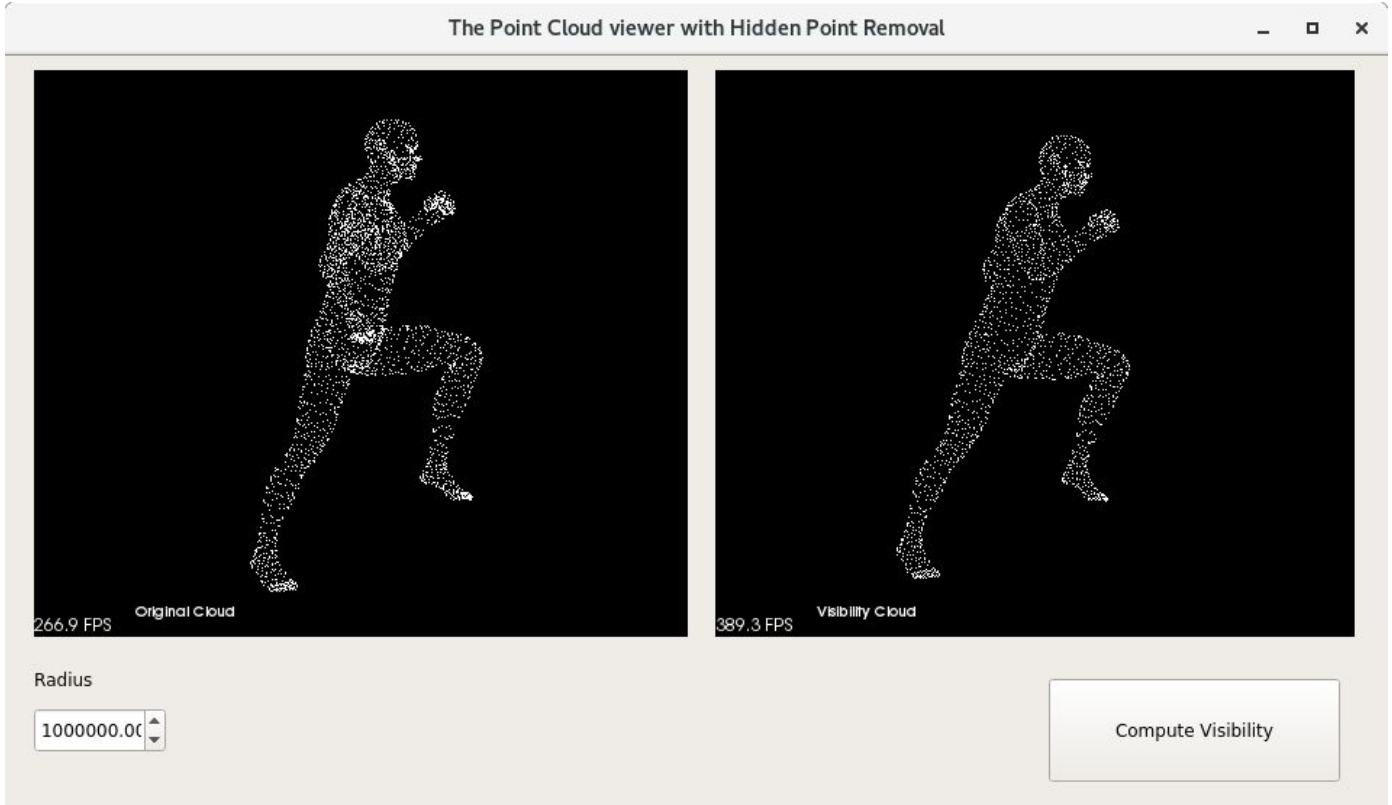


Figure: David side view

C++ Implementation

Listing 2: C++ Implementation

```
std::vector <pcl::visualization::Camera> cam;
viewer->getCameras(cam);
Eigen::Vector3d camera_location(
        cam[0].pos[0],
        cam[0].pos[1],
        cam[0].pos[2]);

// ------ 1. Spherical Inversion ------------------
std::vector <Eigen::Vector3d> spherical_projection;
pcl::PointCloud<pcl::PointXYZ>::Ptr newCloud(
        new pcl::PointCloud <pcl::PointXYZ>);
for (size_t pidx = 0; pidx < cloud->points.size(); ++pidx) {
    pcl::PointXYZ currentPoint = cloud->points[pidx];
    Eigen::Vector3d currentVector(
                currentPoint.x, currentPoint.y, currentPoint.z);
    Eigen::Vector3d projected_point = currentVector - camera_location;

    double norm = projected_point.norm();
    if (norm == 0)
        norm = 0.0001;

    // radius is set by the user using the radius editor
    // f(P_{i}) = P_{i} + 2 (R - ||P_{i}||) \dfrac{P_{i}}{||P_{i}||} = {P_{i}}
    spherical_projection.push_back(
        projected_point + 2 * (radius - norm) * projected_point / norm);
}

size_t origin_pidx = spherical_projection.size();

spherical_projection.push_back(Eigen::Vector3d(0, 0, 0));

for (std::size_t i = 0; i < spherical_projection.size(); ++i) {
    Eigen::Vector3d currentVector = spherical_projection.at(i);
    pcl::PointXYZ currentPoint(
                currentVector.x(), currentVector.y(), currentVector.z());
    newCloud->push_back(currentPoint);
}
// ------ 2. Convex hull construction ------------------
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_hull(
        new pcl::PointCloud <pcl::PointXYZ>);
pcl::ConvexHull <pcl::PointXYZ> chull;
chull.setInputCloud(newCloud);
chull.reconstruct(*cloud_hull);
```

Matlab implementation

Listing 1: Matlab Implementation [2]

```
function visiblePtInds=HPR(p,C,param)
        dim=size(p,2);
        numPts=size(p,1);
        % Move the points s.t. C is the origin
        p=p-repmat(C,[numPts 1]);
        % Calculate ||p||
        normp=sqrt(dot(p,p,2));
        % Sphere radius
        R=repmat(max(normp)*(10 param),[numPts 1]);
        %Spherical flipping
        P=p+2*repmat(R-normp,[1 dim]).*p./repmat(normp,[1 dim]);
        %convex hull
        visiblePtInds=unique(convhulln([P;zeros(1,dim)]));
        visiblePtInds(visiblePtInds==numPts+1)=[];
```

References

[1] S. Katz and A. Tal. On the visibility of point clouds. 2015 IEEE International Conference on Computer Vision (ICCV), pages 1350–1358, Dec 2015.

[2] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. ACM Trans. Graph., 26(3), July 2007.

[3] Ravish Mehra, Pushkar Tripathi, Alla Sheffer, and Niloy J. Mitra. Visibility of noisy point cloud data. Computers and Graphics, In Press, Accepted Manuscript:–, 2010.

[4] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 9-13 2011.

[5] Motion and Shape Computing (MASC) Group: Map GMU. Retrieved October 14, 2019 from http://masc.cs.gmu.edu

Questions?