Simple Occlusion Culling from 3D Wavefront File (.obj)

Ickbum Kim

Department of Mechanical, Aerospace, and Nuclear Engineering Rensselaer Polytechnic Institute, 110 8th St. Troy, NY 12180

Abstract

This work presents a method for projecting 3D point clouds onto 2D image planes using camera projection matrices composed of intrinsic and extrinsic parameters. The method enables synthetic image generation from mesh-based datasets, such as those in the Wavefront `.obj` format, including the Google Scanned Objects dataset. Since standard projection does not account for occlusion, the approach includes an occlusion-handling algorithm that estimates the average distance from each triangular mesh to the camera and uses sorting and `cv::fillpoly` to resolve visibility. Camera parameters such as focal length, principal point, position, and viewing direction are used to define the full projection pipeline. The implementation allows users to simulate realistic camera images from 3D assets while improving efficiency by skipping occluded surfaces. The result is a lightweight and customizable framework for 3D-to-2D projection that is useful in synthetic vision datasets, robotics, and graphics applications.

Keywords: Occlusion culling, mesh, computer vision

Simple Occlusion Culling from 3D Wavefront File (.obj):
Developer's repository link: https://github.com/ickbumk/cas_project
Licensing provisions(please choose one): BSD 3-clause
Programming language: C++
Supplementary material: OpenCV[1], Google Scanned Objects[2]

1. Summary

This project focuses on projecting 3D point clouds onto a 2D camera image plane using a projection matrix composed of intrinsic and extrinsic parameters. Point clouds, defined as $P_i = (x_i, y_i, z_i)$, are transformed into image coordinates through a projection matrix that combines camera calibration data and camera pose. However, due to the homogeneous nature of the projection matrix, occlusions cannot be automatically resolved. Therefore, an occlusion-detection algorithm is needed to:

1. Improve computational efficiency by avoiding rendering hidden surfaces.
2. Enhance visual realism by accurately displaying visible surfaces.

The project uses datasets in Wavefront `.obj` format, such as those from the Google Scanned Objects dataset. These files describe mesh geometry through vertex (`v`), texture coordinate (`vt`), normal (`vn`), and face (`f`) entries. To detect occlusions, triangle meshes are sorted by average distance to the camera. Occluded regions are overwritten using `cv::fillpoly`.

Simple Occlusion Culling from 3D Wavefront File (.obj)

Ickbum Kim

Department of Mechanical, Aerospace, and Nuclear Engineering Rensselaer Polytechnic Institute, 110 8th St. Troy, NY 12180

2. Statement of need

Consider a Point Cloud, $P_i = (x_i, y_i, z_i), i = 1, 2, 3, ..., n$, a set of points that defines a target object in 3D space. An example of a point cloud dataset can be demonstrated as:

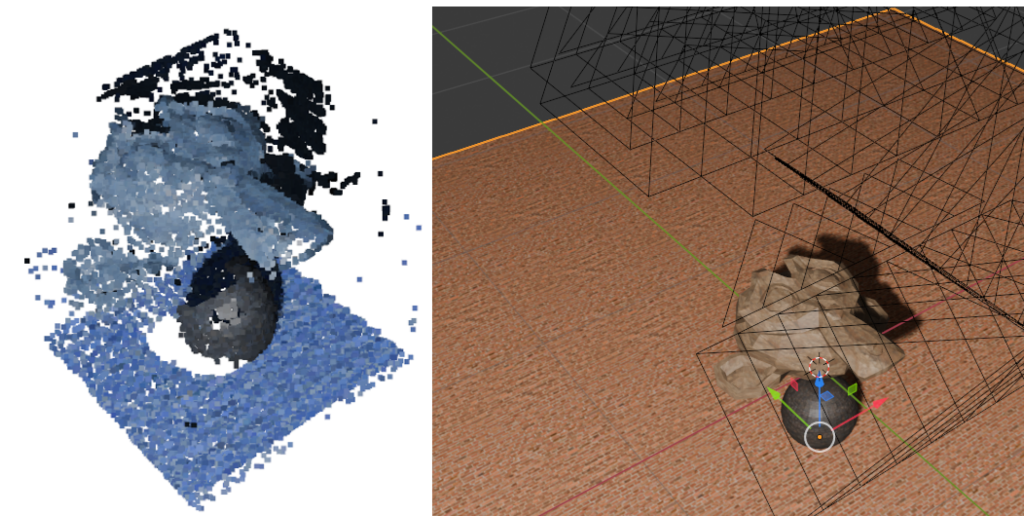

Figure 1: Example of point cloud reconstructed from images sequences simulated in 3D modeling software "Blender"

Given a camera and its properties, the projection of 3D coordinates onto the image plane of the specified camera can be found using the projection matrix:

$$\begin{bmatrix} c_i \\ r_i \\ \lambda_i \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{1}$$

where c_i, r_i, λ_i represent column, row, and normalizing scalar respectively, and p_{ij} represents the projection matrix.

The projection matrix can be further decomposed into:

$$\begin{bmatrix} c_i \\ r_i \\ \lambda_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2}$$

where f_x, f_y, c_x, c_y are the camera intrinsic properties like focal length and principal axis and r_{ij}, t_i are the rotation and translation matrices of the camera in the world coordinate. Position and viewing angle of the camera will fully define **Rotation** and **Translation** matrices. Once the position and viewing angle of the camera are known, we can project the point cloud dataset onto camera image plane to simulate the sequence of images taken from the camera. However, the projection matrix cannot determine what points are occluded because of the normalizing scalar (λ_i). Therefore, there is the need for an algorithm to determine which mesh surfaces are occluded by another, thus providing the following:

1. Computation efficiency (No need to render the occluded surfaces)
2. Accurate demonstration

3. Dataset

The proposed implementation requires wavefront files (.obj). An example dataset provided in the Github Repository is the Google Scanned Objects (GSO) [2]. The wavefront file has the following format, :

- `v` – Geometric vertex. Specifies a vertex with x, y, z coordinates:

  ```
  v 1.000000 2.000000 3.000000
  ```

- `vt` – Texture coordinate. Specifies a 2D or 3D texture coordinate:

  ```
  vt 0.500000 1.000000
  ```

- `vn` – Vertex normal. Specifies a normal vector with x, y, z components:

  ```
  vn 0.000000 1.000000 0.000000
  ```

- `f` – Face. Defines a polygonal face by indexing vertices, texture coordinates, and normals:

  ```
  f 1/1/1 2/2/2 3/3/3
  ```

  Each face entry is of the form `v/vt/vn`, where:

  – `v` is the vertex index
  – `vt` is the texture coordinate index
  – `vn` is the vertex normal index

read_obj.cpp file details the approach to reading the wavefront file.

4. Determining Occlusion

Iterative *cv::fillpoly* has been used to overwrite the occlusions. Provided a triangular meshes from the wavefront file, the average distance can be determined by averaging the distance to each vertices:

$$D_{avg} = \frac{\sum_{i=1}^{n} ||v_i - v_c||_2^2}{n} \tag{3}$$

where $v_i = \{x_i, y_i, z_i\}$ is the 3D vertex and v_c is the camera location. The average distance to mesh faces can then be sorted in the descending order to determine the projection order.

5. Example camera properties used

Camera Intrinsic (I) & Extrinsic (E)	Value	Explanation
f_x, f_y	600	Focal length in x and y direction
c_x, c_y	320, 240	Principal axis in x and y direction
v_c	{0,0,2}	Camera position in world coordinates

Table 1: Camera Intrinsic and Extrinsic Properties

Camera position does not fully define the camera rotation matrix. Therefore, the normal vector from the camera image plane (principal axis) needs to be specified. In this project, it is simplified by normalizing the given dataset and redirecting the camera to directly look at the origin.

6. Acknowledgment

I would like to thank Professor Jacob Merson who has done an unbelievable job teaching this course. It has been a great learning opportunity.

References

[1] Gary Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Lucas Downs, Jaesik Sung, Yu Xiang, Jonathan Tremblay, Juston Chien, Stan Birchfield, Dieter Fox, and Ankur Handa. Google scanned objects: A high-quality dataset of 3d scanned household items. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14446–14455, 2022.