Experiment 1 Object Overlay Using 3D Models and 2D RGB Images:

Files Utilized:

- 3D Object Model: obj_000008.obj
- Camera Scene Information: camera.py and scene_camera.json
- Scene Ground Truth: scene_gt.json
- Test RGB 2D Image: 000005.png

Methodology:

Initial Extraction and Overlay

- Steps Extracted pose information from scene_gt.json and camera parameters from scene_camera.json.
- Rendered the 3D object model and aimed to overlay it on test image 000005.png.

Results and Analysis (First Part)

- Extracted pose information specifically for object ID 8 (obj_000008.obj), including a rotation matrix and a translation vector.

Challenges (First Part)

- Encountered installation issues with the pywavefront library for OBJ file parsing.

Solutions and Modifications (First Part)

Utilized alternative parsing methods to successfully extract object vertices and faces.

Further Steps:

- Applied a transformation matrix, derived from the pose, to the object's vertices.
- Projected the transformed vertices onto the image plane using camera parameters.

Results and Analysis (Second Part)

- Extracted camera parameters, notably the intrinsic matrix (cam_K).
- Employed original image dimensions (640x480 pixels) due to the lack of this information in scene_camera.json.

Challenges (Second Part)

- Projection and Alignment Discrepancies in the intrinsic matrix dimensions resulted in incorrect projections.
- Had to adjust the y-coordinates and translation for better alignment.
- Experienced issues with object scaling and orientation.

Sub-Challenges and Fixes

- **Flipping Orientation:** Corrected the object's orientation by flipping it along the appropriate axis.
- **Positioning and Alignment:** Made iterative adjustments for proper alignment.
- **Scaling:** Applied a scaling factor to match the size of the original object in the image.

Solutions and Modifications (Second Part)

- Reshaped the intrinsic matrix (K) to a 3x3 matrix, suitable for the proj_from_K function.

Final Outcomes:

- Projected vertices were normalized.
- Mapped the normalized coordinates to the original image's pixel coordinates.
- Successfully rendered the object, completing the overlay process.

Experiment 2
Trimesh Library for Integrating 3D Models onto 2D RGB Visuals

Files Utilized

- 3D Object Models: obj_000008.obj and obj_000008.ply
- Texture: obj_000008.png
- Test RGB 2D Image: 000005.png
- Camera Information: camera.py and scene_camera.json
- Ground Truth Information: scene_gt_info.json and scene_gt.json

Initial Setup and Data Exploration

- Loaded the 3D object models and the 2D RGB test image.
- Reviewed the camera and ground truth files to understand their structure and content.

Summary of Camera and Ground Truth Data:

- **Camera Data:** Includes camera intrinsic parameters (cam_K), rotation (cam_R_w2c), and translation (cam_t_w2c) from world to camera coordinates.
- **Ground Truth Data:** Provides object-specific rotation (cam_R_m2c) and translation (cam_t_m2c), as well as object ID (obj_id).

Methodology:

- Created a projection matrix and attempted to align the 3D object with the 2D image using the Trimesh library.

Results and Analysis

Challenges

- The Trimesh library limitations prevented rendering the transformed mesh, making it incompatible with the 2D RGB image perspective.

Solutions and Modifications

- Added material files (material.mtl and material_0.png) to enhance the object's appearance during rendering.
- Reloaded the 3D object with the new material and repeated the rendering and alignment process.

Further Steps

- Transform the 3D object using the previously calculated projection matrix.

Results and Analysis

- The 3D object with the applied material was successfully transformed.

Final Overlay and Rendering Needed

- Need to utilize additional tools like PyTorch3D and Blender for material-based rendering.

Custom Approach for Overlay

- Created a 2D rendering of the transformed mesh and overlaid it onto the 2D RGB image, but unsuccessfully applied the texture.

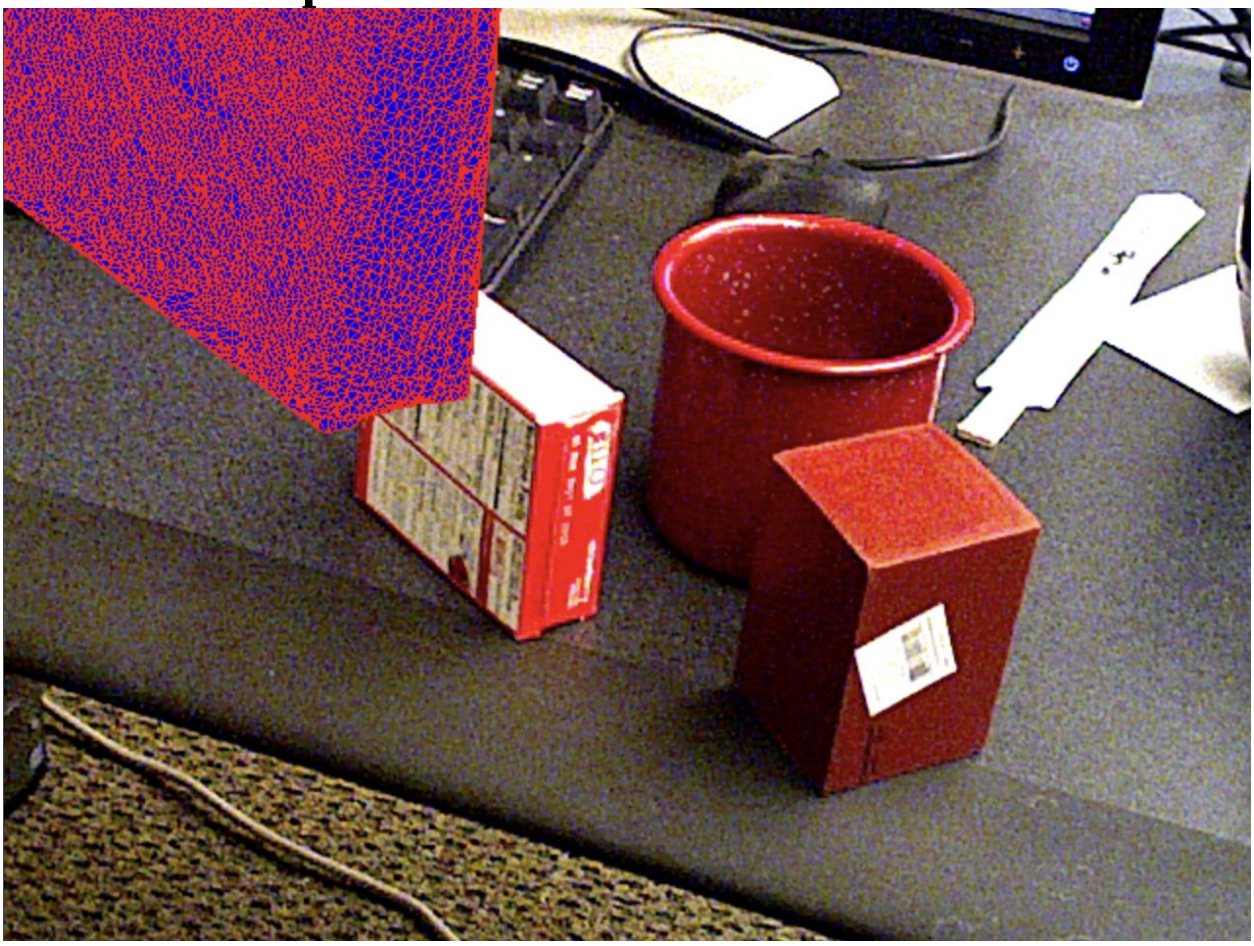
Final Outcomes:

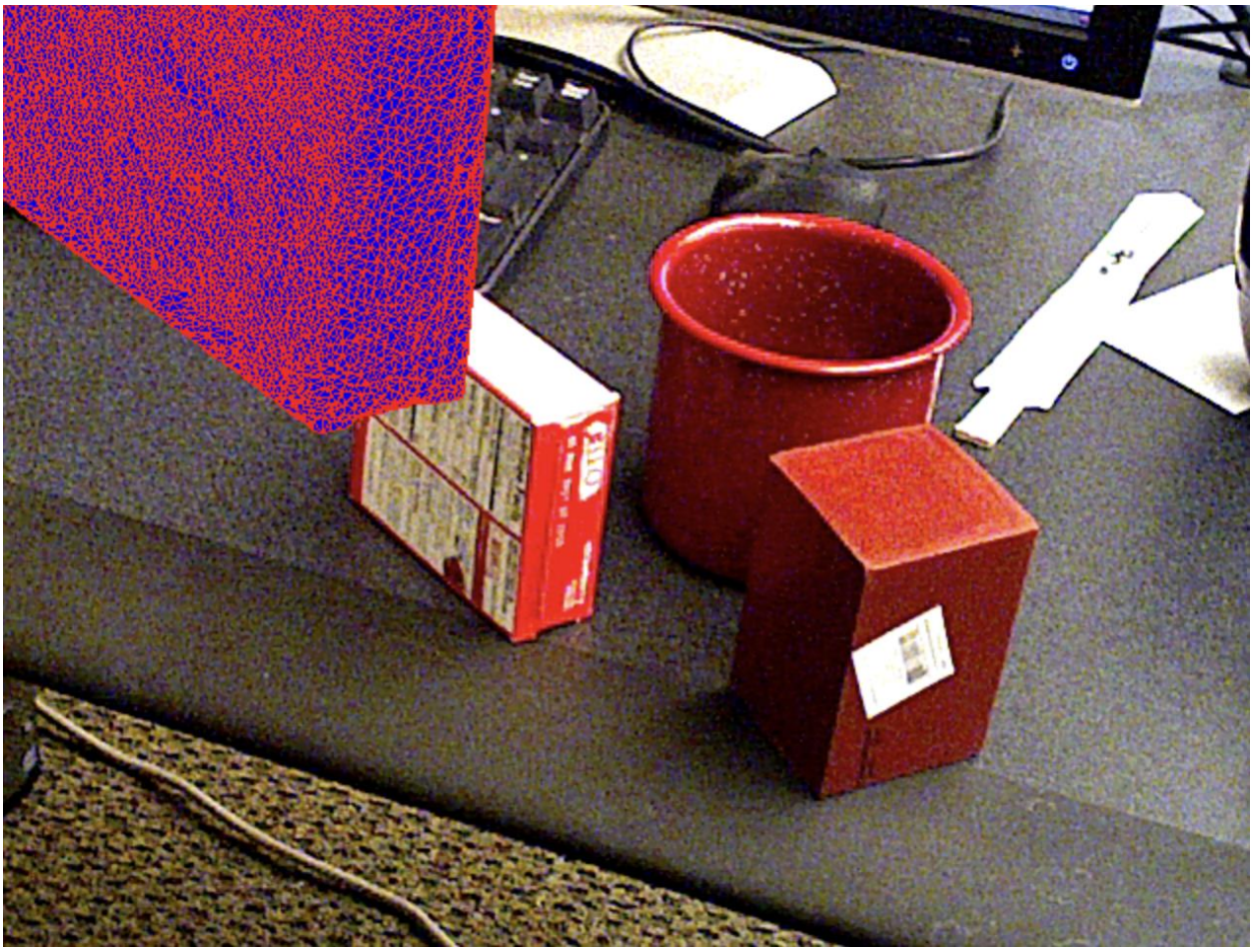
- The 2D rendering, represented by red dots, was successfully overlaid onto the RGB image, based on the transformed 3D object vertices, and aligned with the 2D RGB image perspective.

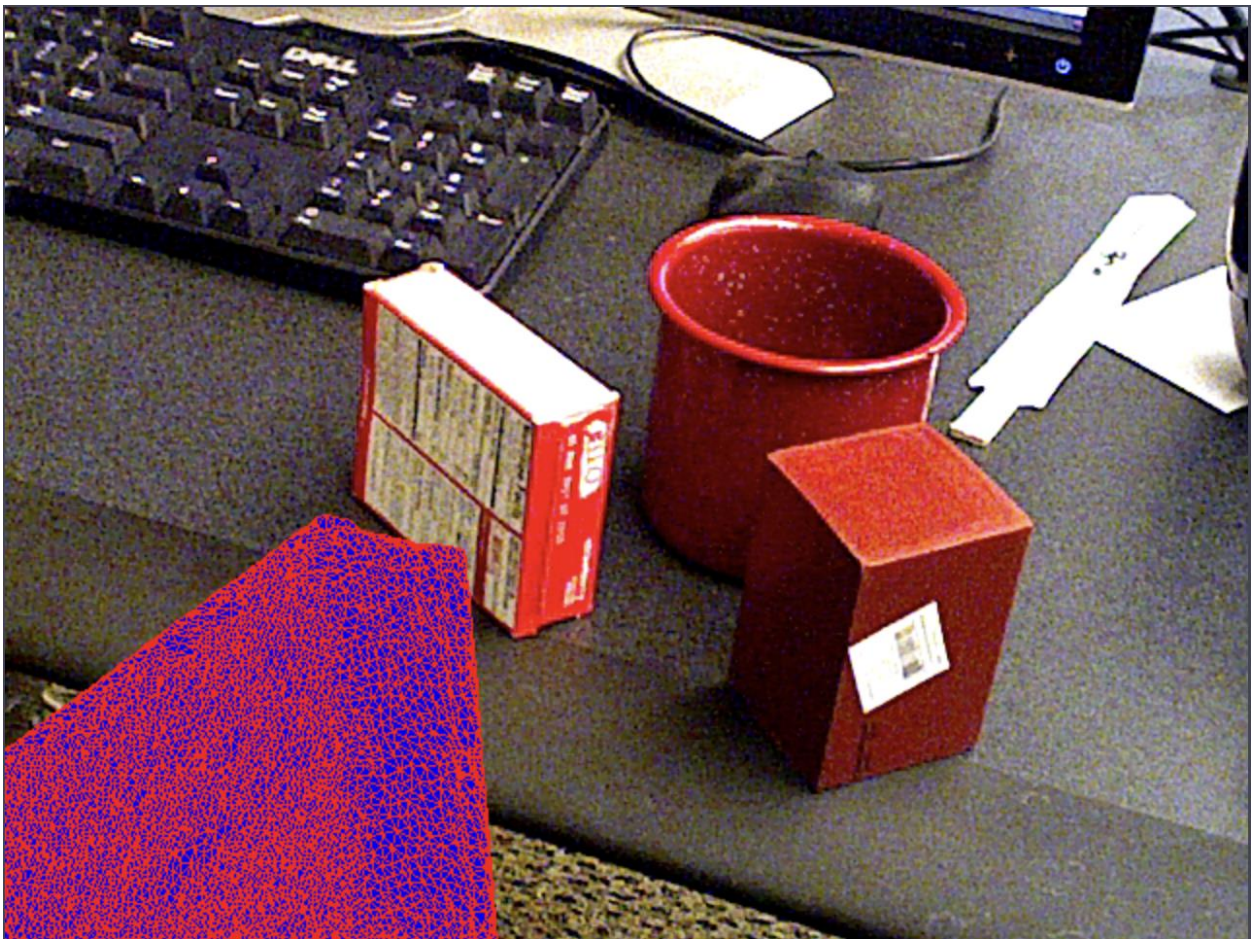
Next Steps:

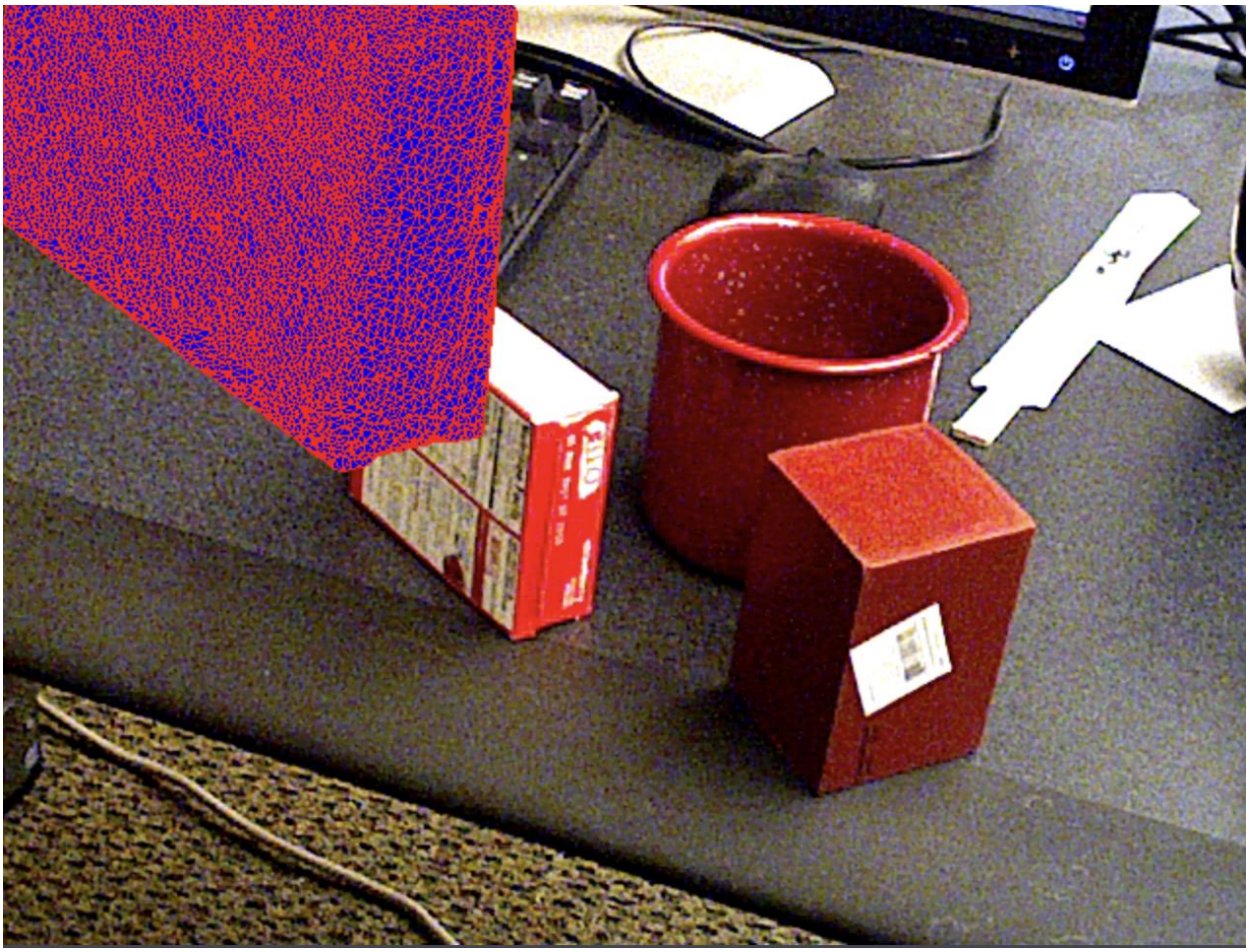
1- Learn Blender Python API for material-based rendering.
2- Continue Experiment 2 and complete a full rendering script for the 3D Models.
3- Keep following the Advice and Guidance of Dr. Tramblay and Professor Birchfield and learn from their feedback.

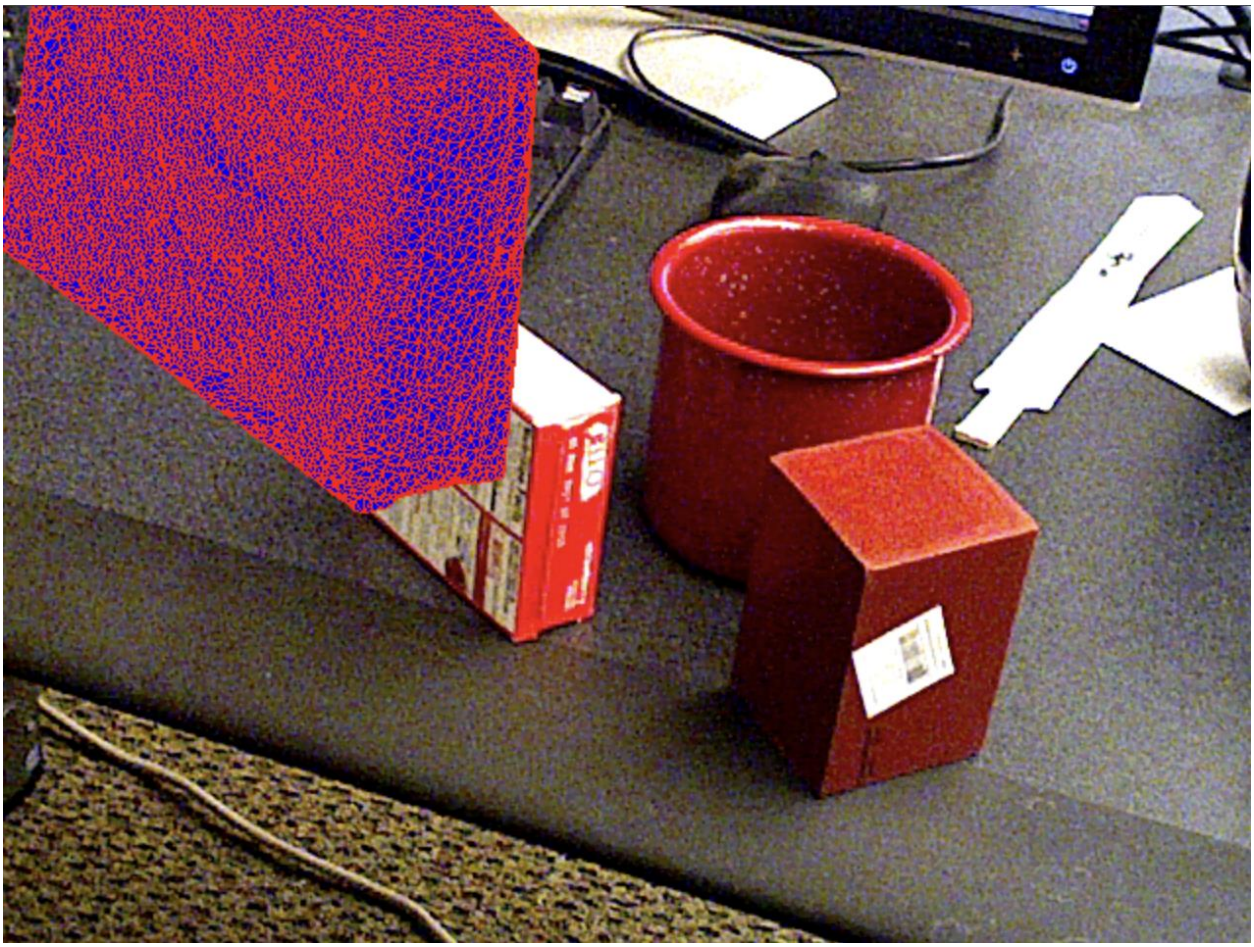
Results of Experiment 1:

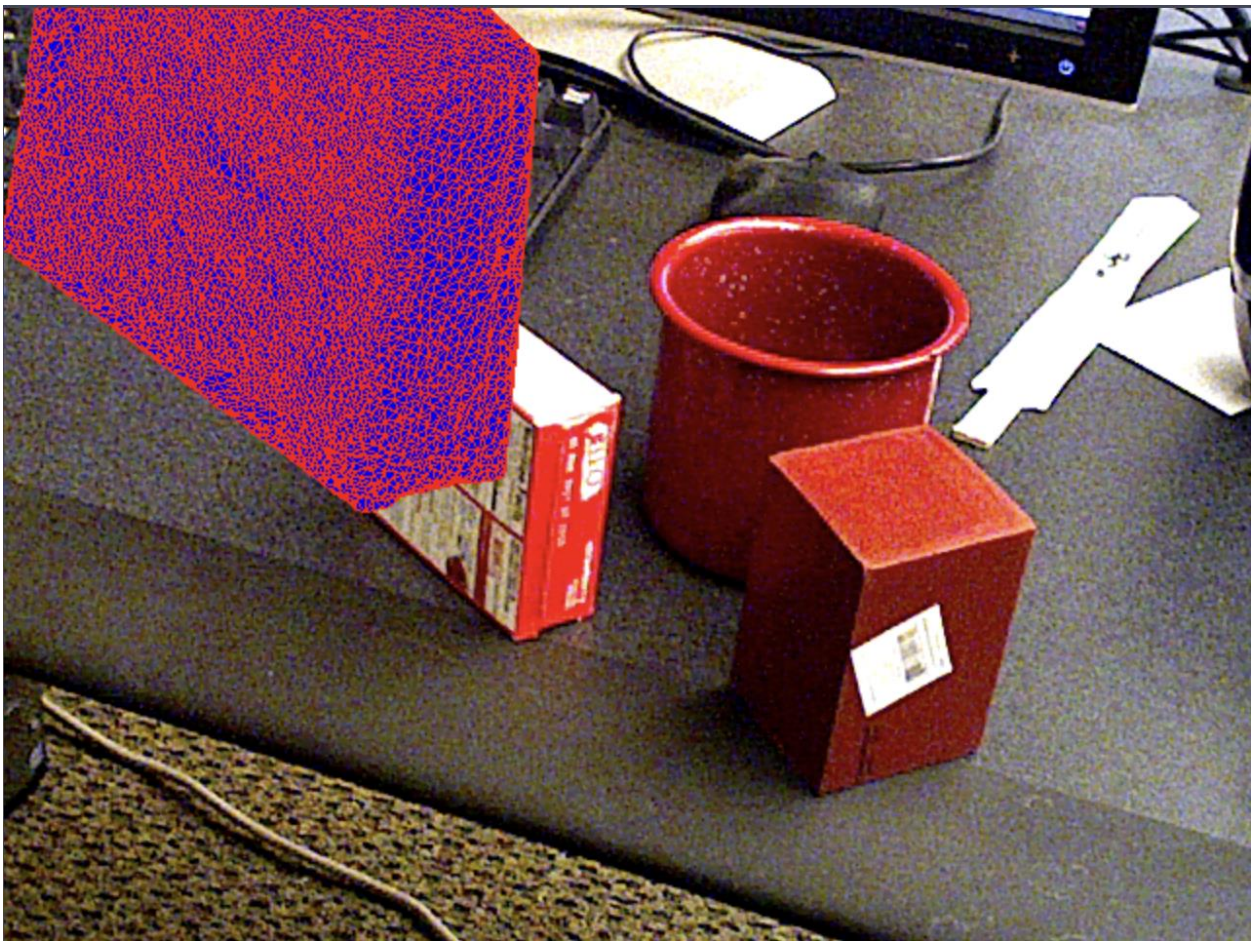


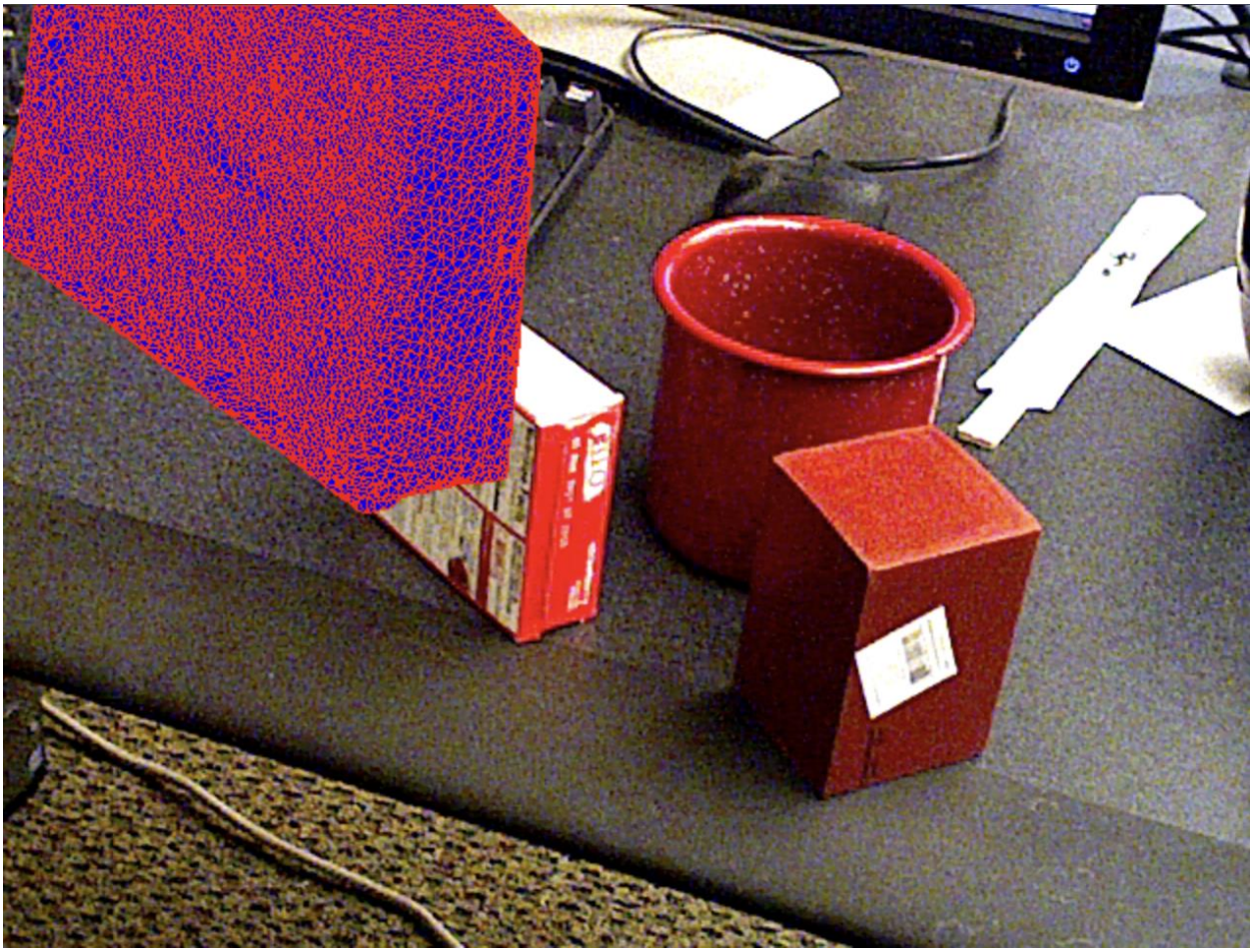


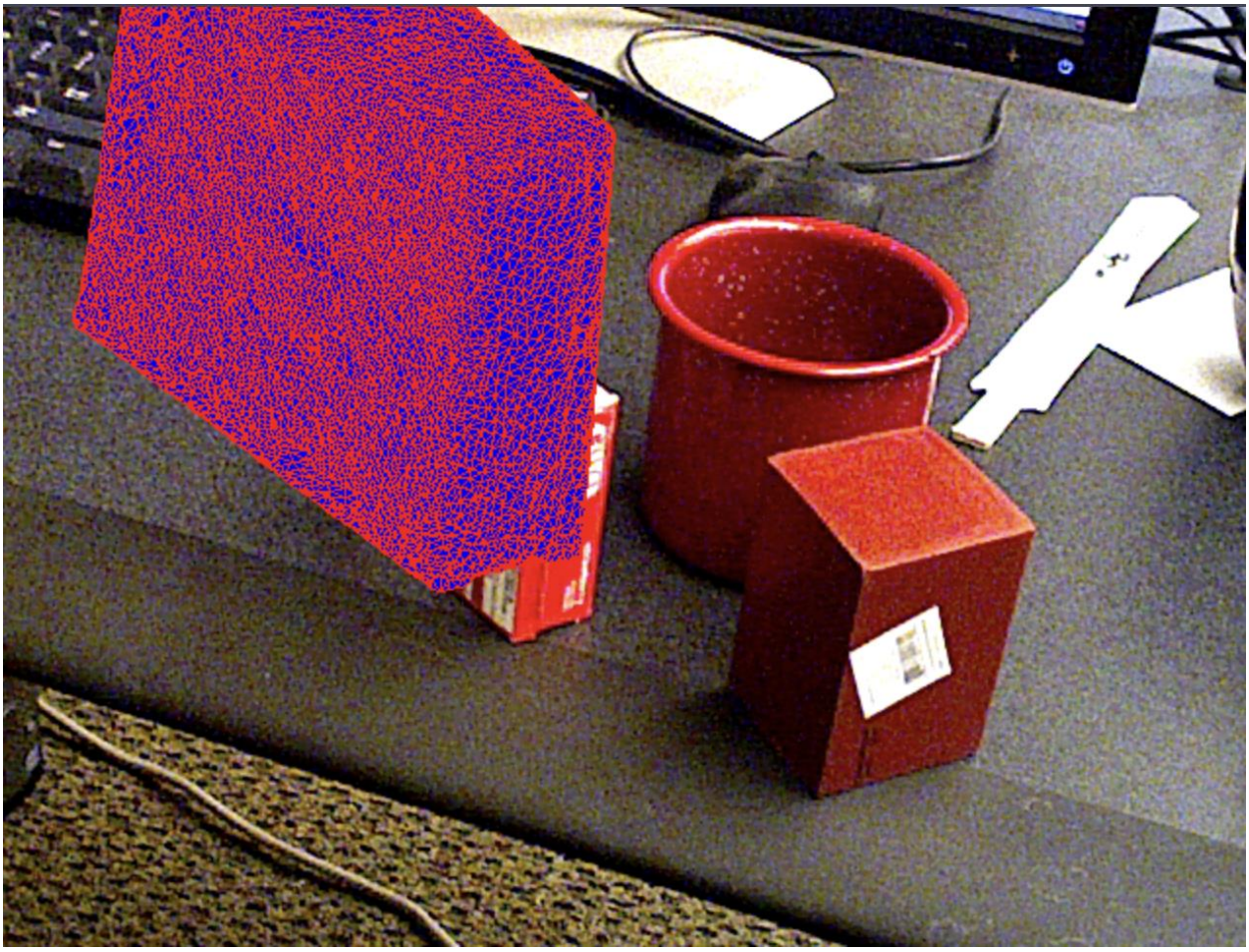


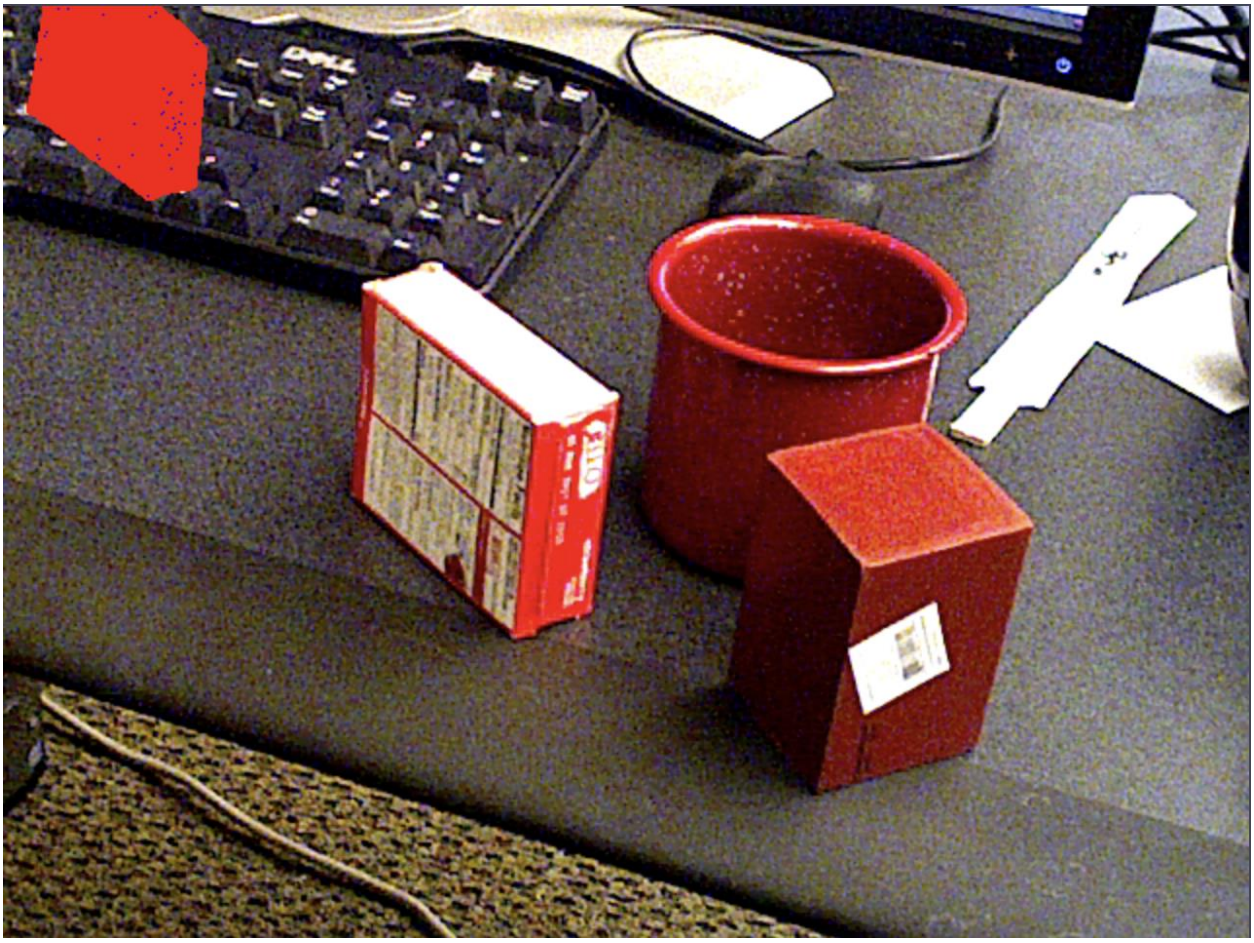


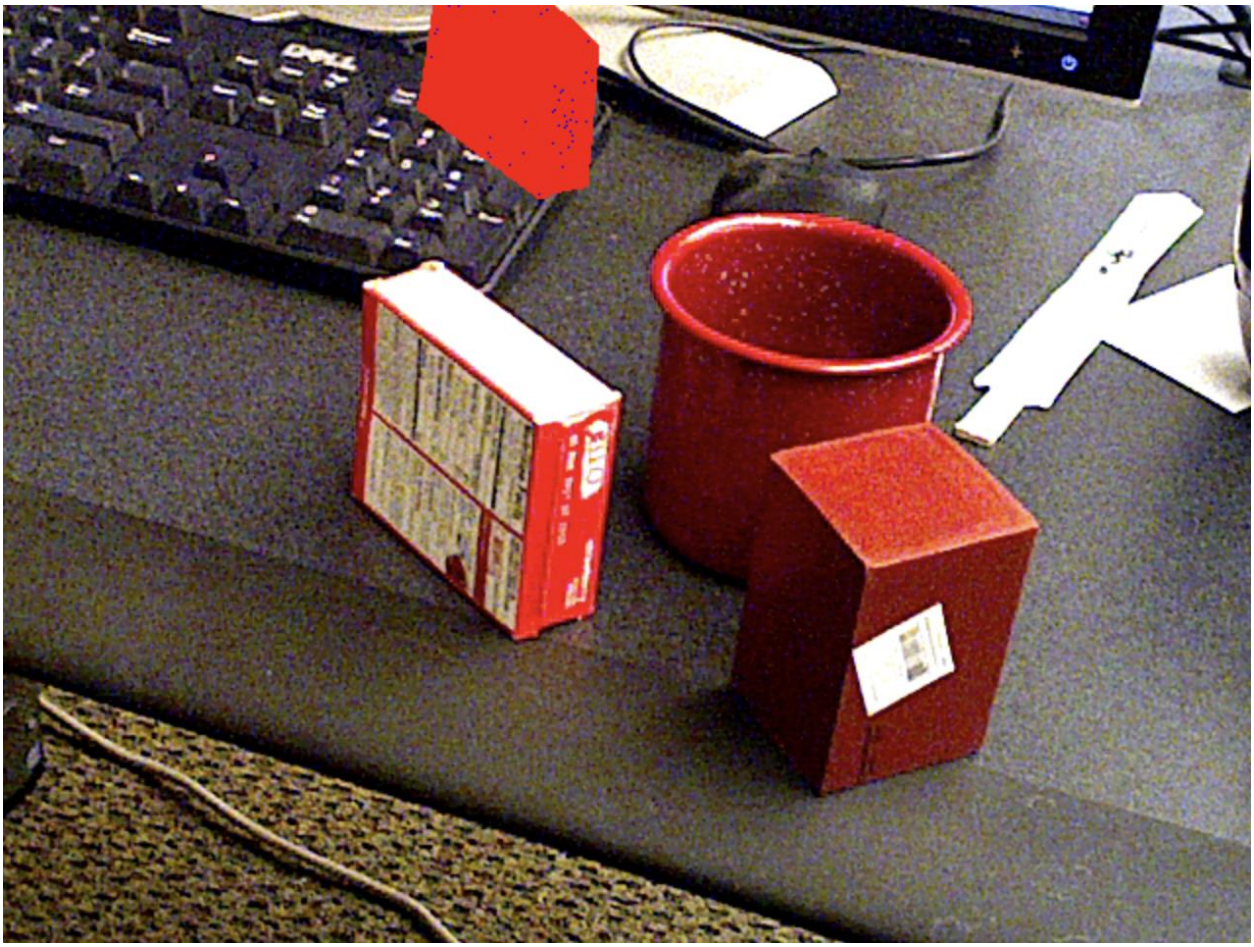


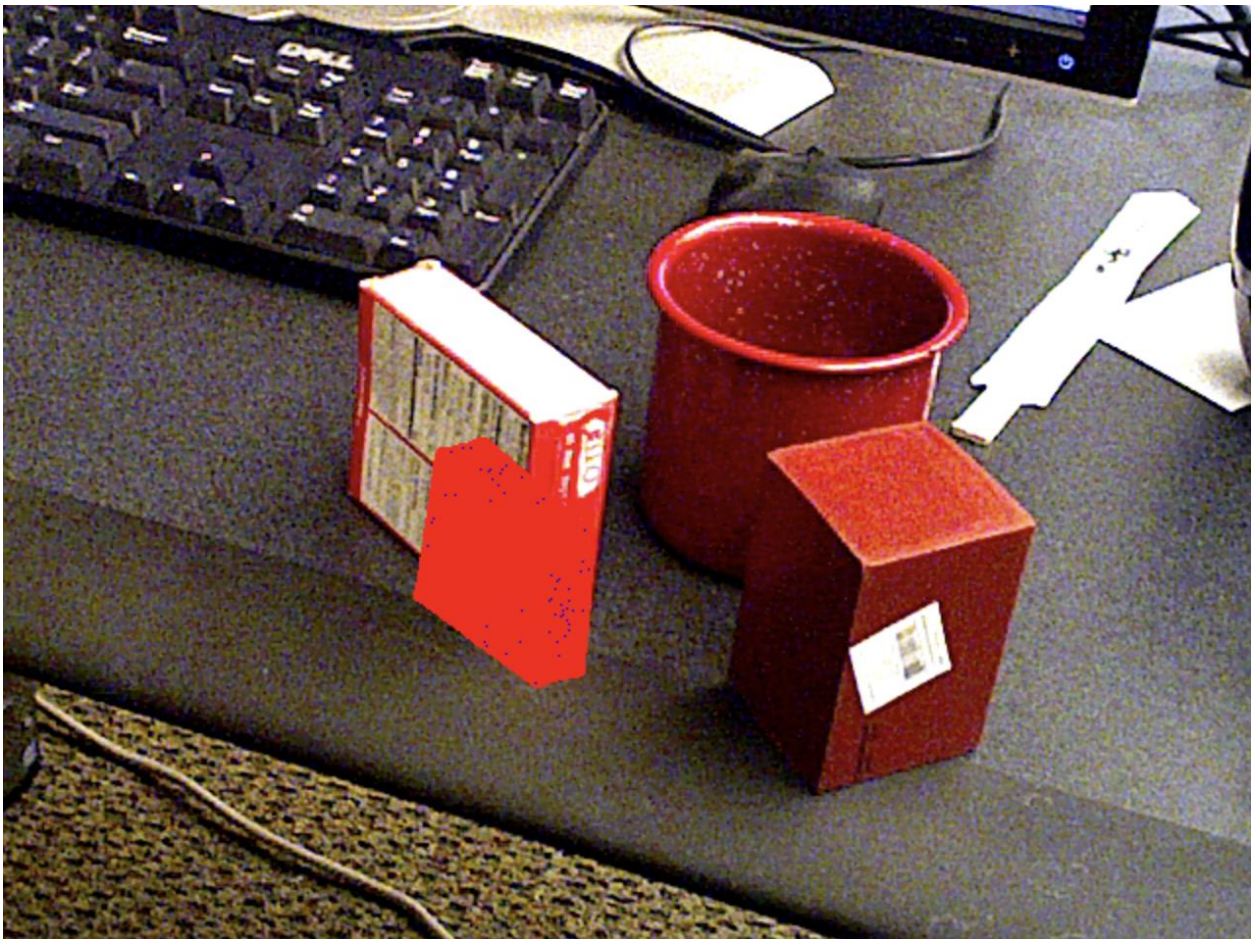


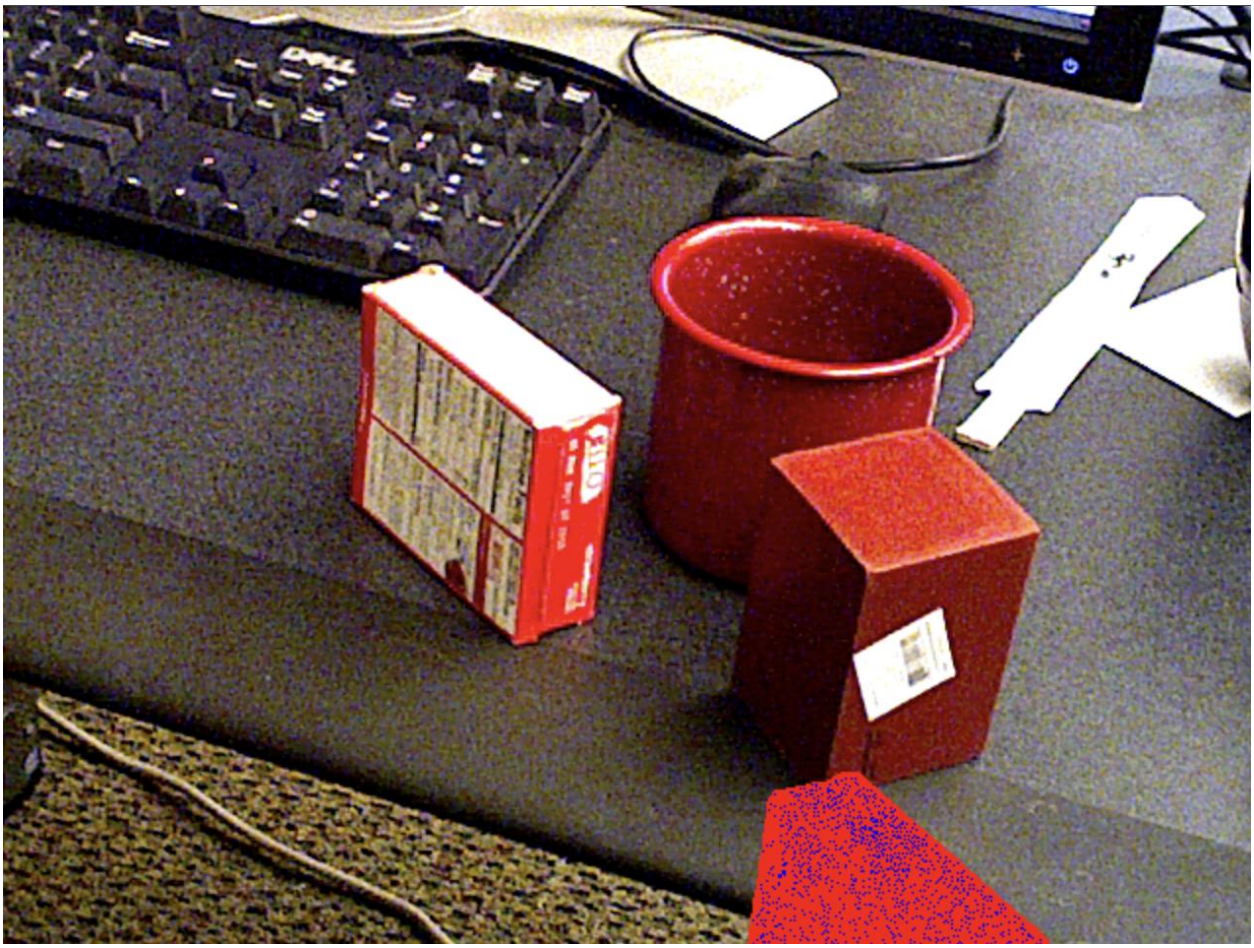


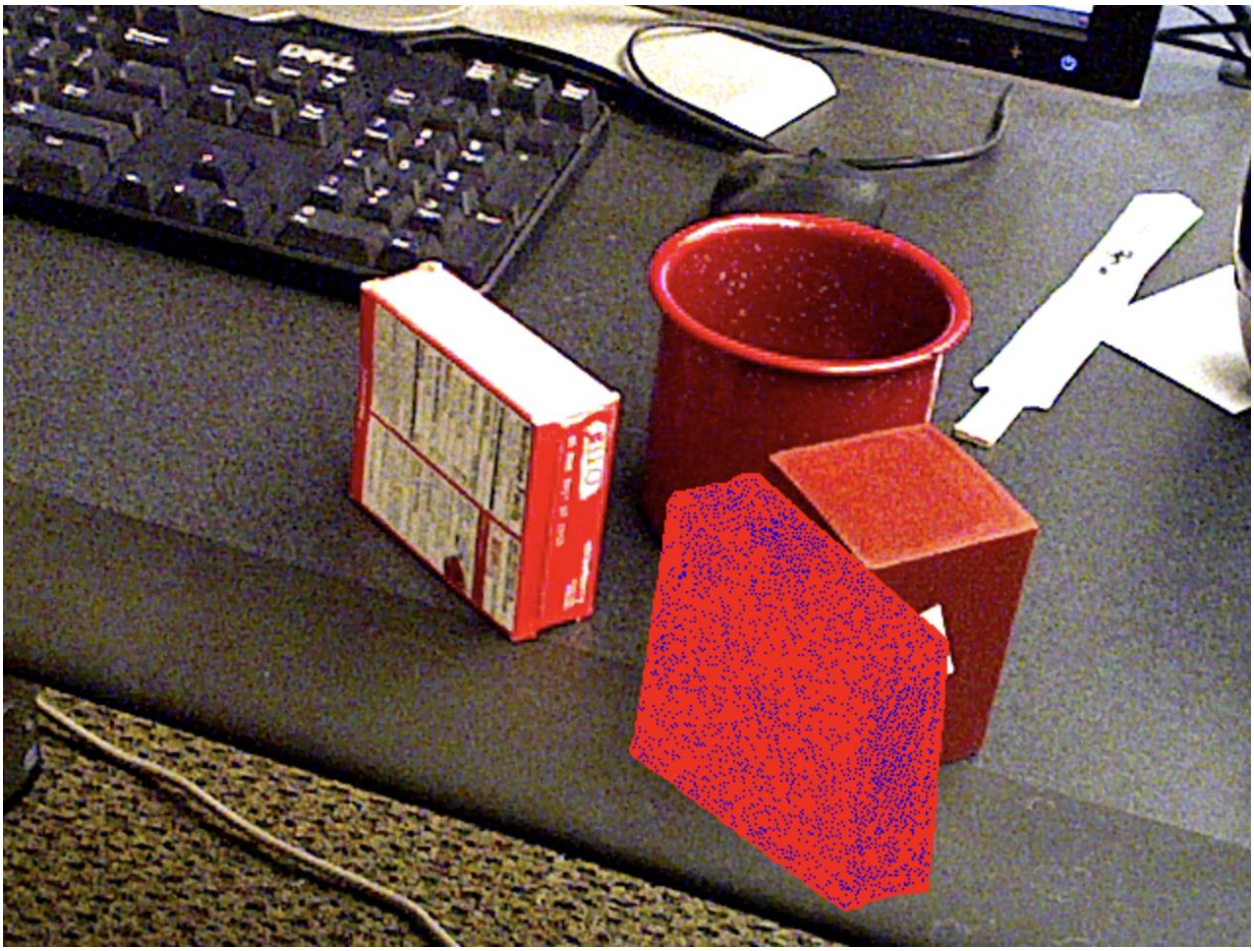


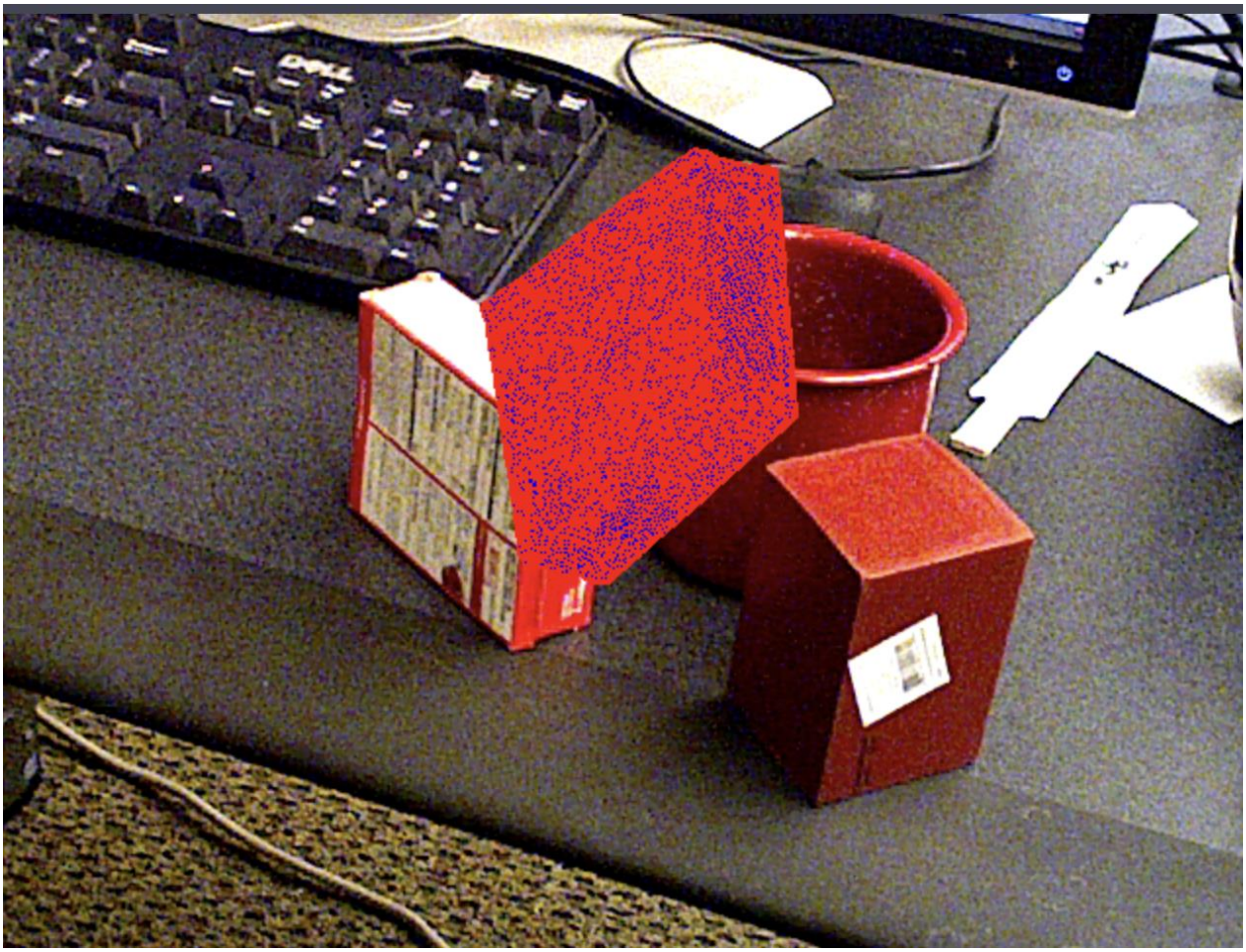


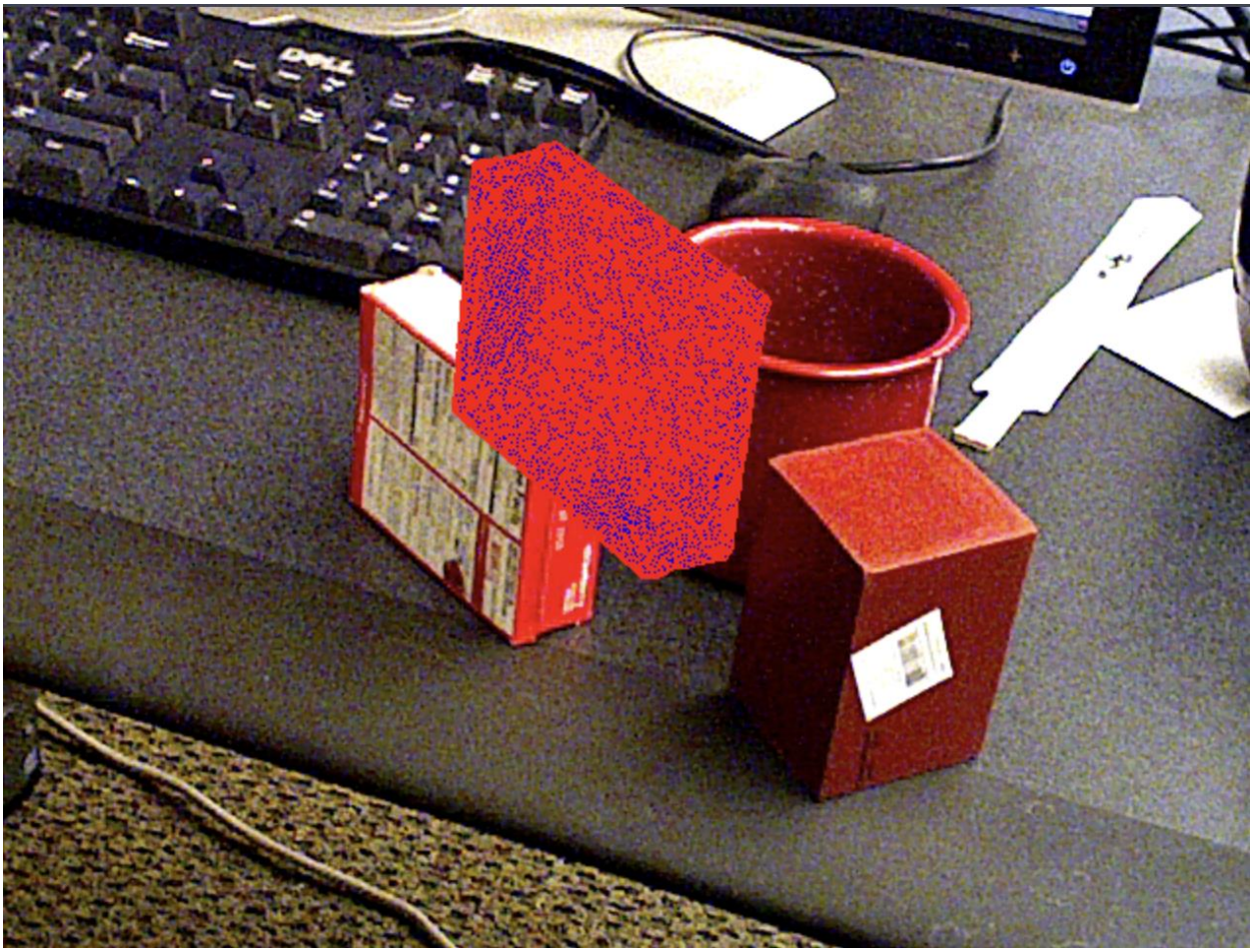

DELL

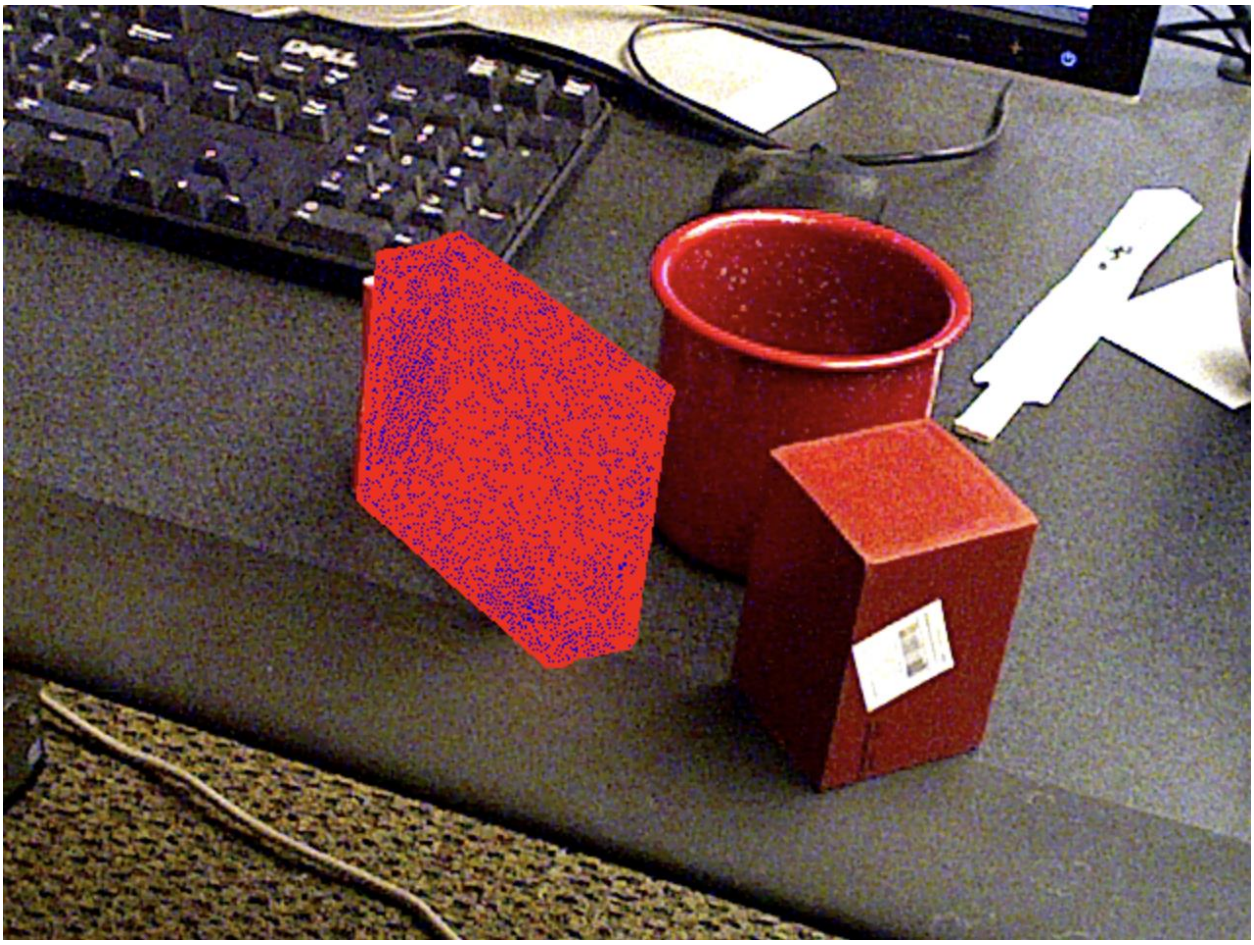

DELL


DELL


DELL


DELL





Results of Experiment 2: