# Photorealistic Arm Robot Simulation for 3D Plant Reconstruction and Automatic Annotation using Unreal Engine 5

Xingjian Li[1,2] Jeremy Park[3] Chris Reberg-Horton[4] Steven Mirsky[5]
Edgar Lobaton[2] Lirong Xiang[1*]
[1]Department of Biological and Agricultural Engineering, North Carolina State University
[2]Department of Electrical and Computer Engineering, North Carolina State University
[3]Department of Computer Science, North Carolina State University
[4]Department of Crop and Soil Science, North Carolina State University
[5]Sustainable Agricultural Systems Laboratory, USDA-ARS
{xli228, jipark, screberg}@ncsu.edu steven.mirsky@usda.gov
{ejlobato, lxiang3}@ncsu.edu

## Abstract

*Robotics paired with computer vision are widely used in precision agriculture. Simulations are critical for safety and performance estimation by verifying their routine in a virtual world before real-world testing and deployment. However, many simulators used in agricultural robots lack photorealism in their virtual worlds compared to the real world. We implemented Unreal Engine 5 (UE5) and the Robot Operating System (ROS) to develop a robot simulator tailored to agricultural tasks and synthetic data generation with RGB, segmentation, and depth images. We designed a method for assigning multiple segmentation labels within a single plant mesh. We experimented with a semi-spherical routine for two robot arms to perform 3D point cloud reconstruction across 10 plant assets. We showed our simulator produces much more accurate segmentation images and reconstruction compared to existing UE5 solutions. We extend our results with Neural Radiance Field (NeRF) reconstructions. The packaged simulator, UE5 project, and ROS package with the Python routine can be found at https://github.com/NCSU-BAE-ARLab/AgriRoboSimUE5.*

## 1. Introduction

Robotics paired with computer vision are widely used in precision agriculture. However, the development and verification of these systems are challenging due to dynamic and ever-changing agricultural environments. Similar issues exist across all robotics fields, and simulation is a common solution to reduce test waiting times. Gazebo [12] has been one of the most popular simulators because of its integration with the Robot Operating System (ROS). However, its rendering capabilities lag behind other modern approaches using Isaac Sim, Unity, or Unreal Engine (UE) which have seen applications in autonomous driving or aerial vehicles [7, 11, 22]. For agricultural robots with image sensors such as RGB and depth cameras, developing an agricultural-specific simulator could accelerate advancements in perception and decision-making specific to agricultural tasks [15].

There have been previous attempts to create photo-realistic agricultural scenes using UE4, but they only produce RGB and depth information [5] or ignored robot simulations and developed purely as a data generation tool [6, 9]. In agricultural simulations, primitive geometry (cubes and circles) were used as plant models [10, 13] because accurate plant and foliage models are difficult to obtain due to their complex geometry and textures. This issue has also restricted the digitization of plant models. While realistic digital plant models can be created using 3D reconstruction techniques such as LiDAR scans or NeRFs [19, 30], there has yet to be a public library hosting these formats with a good amount of variety.

An alternative to synthetic data generation from simulations is to use diffusion models. This technique enabled photo-realistic weed image generation [17] but was limited to the camera views from the training dataset. Robots should also consider various view angles for finding efficient workflows and optimal designs. Integrating a robot simulator into a 3D simulation would provide better clarity and perspective realism in generated synthetic images and also provide the execution time to reach these positions.

We used UE5.3 to develop our simulation environment. Compared to UE4 used in AirSim [22] and CARLA [7],

*corresponding author

UE5 offers many new features for better rendering and development tools. Lumen and Nanite are new features in UE5 enabling highly detailed meshes rendered with realistic dynamic lighting in real-time [23, 24]. UE5 also changed the default physics engine from PhysX to Chaos Engine for lightweight simulation. However, there is a limited amount of documentation or research about this engine's performance for robotics purposes.

We developed a UE5 robot simulator focused on photorealistic plant image generation for photogrammetry tasks. Using UE5 also licensed the unlimited use of Quixel MegaScan[1] assets for high-quality realistic 3D plant models optimized for video game purposes (artistically designed meshes). The simulated task would be appropriate for plant 3D reconstruction and phenotyping research in agriculture with stationary robot arms. In addition to mesh assets, we experimented the same task over point clouds, a data structure for 3D mapping commonly used in robotics.

Our main contributions to this project are:

- We present a multi-robot simulator with RGB, segmentation, and depth image data generation capabilities as a packaged application.
- The development of a texture-based segmentation technique for automatic annotations with multi-labels for a single mesh.
- We develop a novel ROS-UE5 pipeline for concurrent 3D point cloud reconstruction.

## 2. Related Works

### 2.1. Agricultural Synthetic Data

Generating synthetic data currently is a cost-efficient approach for creating the training dataset for neural networks. Cicco *et al.* [6] demonstrated synthetic data from UE4 were cheaper and competed well against networks trained on real images in weed detection tasks, even for networks trained exclusively using synthetic data. Similarly, Choi *et al.* [4] released many synthetic image datasets with instance segmentation on the fruits through the Helios renderer for multiple crops.

Aside from using 3D renderers, other approaches like image stitching or generative models are used to create agricultural synthetic datasets. Toda *et al.* [27] demonstrated instance segmentation models trained on synthetic images stitched from a pool of seed images achieved high average precision (>95%) in real-world images in a controlled environment. Cap *et al.* [3] developed a generative adversarial network (GAN) using real-world data to transform images of healthy leaves into diseased leaves. Moreno *et al.* [17] proposed Stable Diffusion models to generate weed images for training object detection models, and Anagnostopoulou *et al.* [2] applied ControlNet, a diffusion model, to

[1]Quixel: https://quixel.com/megascans

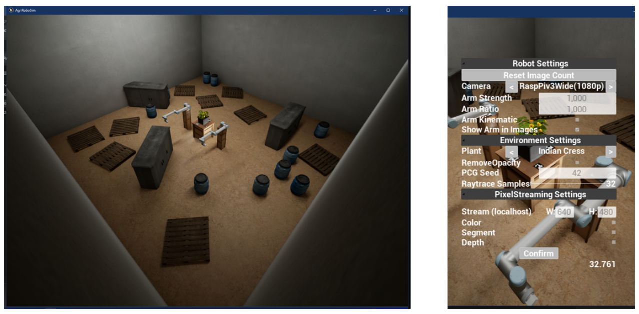

Robot Settings
Reset Image Count
Camera < RaspPiv3Wide(1080p) >
Arm Strength 1,000
Arm Ratio 1,000
Arm Kinematic
Show Arm in Images ✓
Environment Settings
Plant < Indian Cress >
RemoveOpacity
PCG Seed 42
Raytrace Samples 32
PixelStreaming Settings
Stream (localhost) W: 640 H: 480
Color
Segment
Depth
Confirm
32.761

Figure 1. Simulator and GUI.

create realism effects in rendered mushroom images using text prompts.

Depth sensing is important in robotics and precision agriculture to create a 3D mapping of the robot's surroundings. However, none of these synthetic agricultural data generation pipelines currently include synthetic depth generation. Furthermore, our tool provided more freedom in segmentation labeling, where multiple labels could be marked in a single mesh.

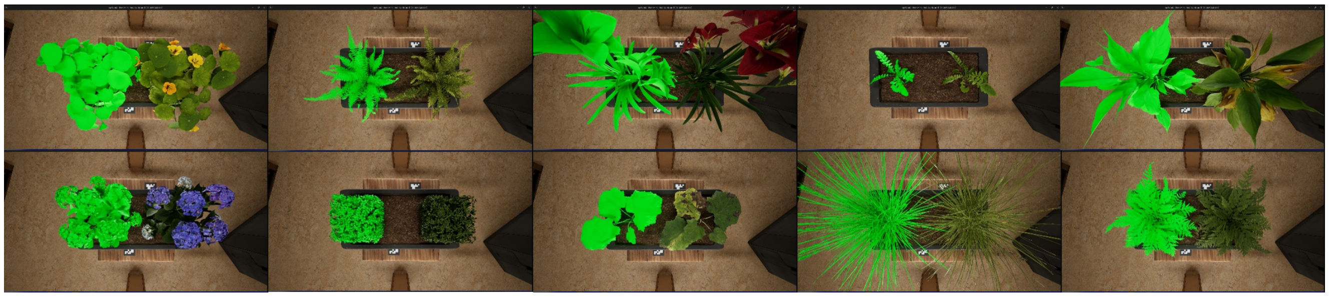
### 2.2. Photorealistic Robot Simulations

Vision-enabled robotics have seen adoption across various fields with advancements in computing hardware and computer vision. This also created the need for simulators with photorealistic features, *e.g.*, global illumination, dynamic materials, and complex scene creation. These features are difficult to implement in traditional robot simulators such as Gazebo [12]. Habitat 3.0 [18] is a recent human-robot simulator for home environments, this simulation trained end-to-end reinforcement learning models for simulated robots with RGB and depth cameras. FlightGoggles [8] is a Unity-based simulation for drones with various sensor outputs and integration with ROS. Similar simulators exist in autonomous driving with CARLA [7] and AirSim [22] built on UE4 to integrate computer vision and autonomous vehicles.

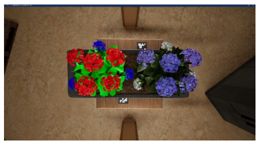
However, there is very little development toward this type of simulator for agricultural tasks. Cicco *et al.* [6] and Choi *et al.* [4] attempted photorealism through 3D rendering as mentioned before, but there is no agricultural-specific simulator known to us that connects both robotics and photorealism. Such simulators exist for home robots, Truong *et al.* [28] demonstrated visual robot navigation and successful sim-to-real transfers of end-to-end policies trained in both Habitat and iGibson simulators.

## 3. Simulation Environment

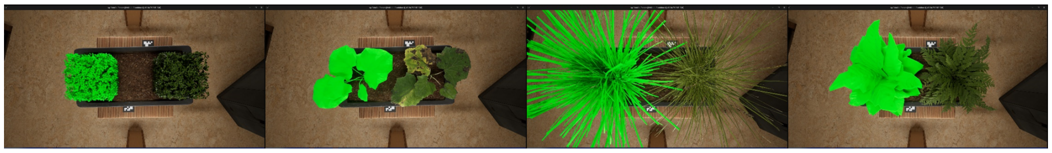
We designed a simulated indoor environment with four controlled lighting sources, two from the ceiling facing down and two on the robot stands facing the plant. We manually



(a) The 10 meshes selected with both segmentation and original view. From left to right, top to bottom are Indian Cress, Boston Fern, Amaryllis, Bog Marshcress, Chinese Evergreen, Hydrangea, Boxwood, Coltsfoot, Basket Grass, and Common Fern.



(b) Multi-label Hydrangea.



(c) The segmentation mesh using the RepOpa method.

Figure 2. Top-down view of plant assets.

recreated Universal Robot 10 (UR10) for our simulator. The robots were constructed with meshes from ur_description with kinematics and dynamics defined from their corresponding config files. Physics constraint components were used to represent the revolute joints between static meshes. The robot was constructed as a blueprint for multi-arm simulation and can be controlled using ROS topics detailed in Sec. 3.4. Two UR10 were placed 1.5 meters from the center of the pot as shown in Fig. 1. We also created a graphical user interface (GUI) for any runtime adjustments on the robot, camera, and environment settings without a UE5 editor.

### 3.1. Robot Cameras

We attached two different camera models, namely *RobotCam* and *CineCamera*, to the robot end effector, where *RobotCam* represents an ideal camera defined simply on horizontal field of view (FOV) and aspect ratio, and *CineCamera* represents a more realistic model that can be defined by settings such as sensor dimensions, focal length, and aperture. We used three Capture Scene 2D to load the one of the camera settings to render RGB, segmentation, and depth textures respectively. These textures can be saved as images asynchronously or streamed over WebRTC using the PixelStreaming[2] plugin.

UE5.3 offers several rendering options including various color spaces, HDR/LDR, base color, depth, and normal. Specifically, we used *BaseColor in RGB* to obtain the segmentation masks under SegmCaptureScene2D. For regular RGB rendering, we used *Final Color (HDR)* to match the player view in the application. We used *SceneDepth in R* to save the ground truth depth as EXR images with 16-bit information at 1 cm. Furthermore, these Scene Capture 2D would only render *Actors* tagged with *ColoredImageGen* or *SegmentImageGen*. This allows control over which objects to render which is useful in generating plant segmentation masks using the technique described in Sec. 3.2.

We enabled hardware raytracing for the RGB camera for realistic soft-shadow calculations. UE5 offers other shadow map options but those have visually inaccurate shadows for these plant assets. Enabling raytracing costs the simulator's frames per second (FPS) but still enough to generate static images for individual runs.

### 3.2. Plants and Automatic Annotation

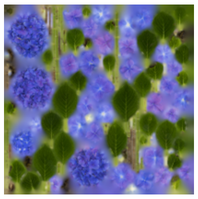
We used 10 Quixel Megascan plants with 4K textures as listed in Tab. 1. These plants covered a wide variety of features in leaf shape, texture, and colors. We also created a blueprint for changing the plant assets' appearances and adding new plants in the editor. A data table was used to store mesh references and manage their segmentation texture and scale in the world. Figure 2a shows the top-down view of two blueprints (one segmentation and one RGB) of each plant asset used in our work.

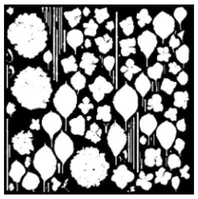
Our segmentation method operates on the textures and UV mappings instead of existing, free, UE5 solutions like EasySynth [29] which replaces every material within the mesh with a single color, leading to fully opaque meshes. We only replaced the base-color (Albedo) textures with a segmentation texture and thresholded the opacity mask as shown in Fig. 3d. This technique produced precise segmentation masks for assets with opacity/transparent textures which is common in artist-designed 3D plant assets. We thresholded the opacity texture to fully opaque or fully

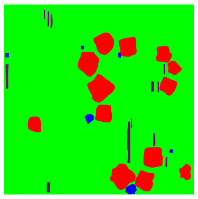
[2]PixelStreaming Plugin: https://docs.unrealengine.com/5.3/en-US/pixel-streaming-in-unreal-engine/

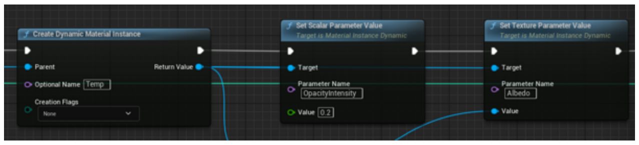
| Plants | Vertices | Triangles |
|---|---|---|
| Hydrangea | 72409 | 77836 |
| Indian Cress | 1790 | 1463 |
| Amaryllis | 2129 | 2352 |
| Boston Fern | 2773 | 2816 |
| Common Fern | 621 | 752 |
| Bog Marshcress | 708 | 896 |
| Boxwood | 485871 | 697847 |
| Coltsfoot | 1078 | 1648 |
| Chinese Evergreen | 1688 | 1968 |
| Basket Grass | 25180 | 23850 |

Table 1. Plant assets used for the simulator.


(a) *Albedo*


(b) *Opacity*


(c) *Segmentation*


Create Dynamic Material Instance
Parent
Return Value
Optional Name: Temp
Creation Flags: None

Set Scalar Parameter Value
Target is Material Instance Dynamic
Target
Parameter Name: OpacityIntensity
Value: 0.2

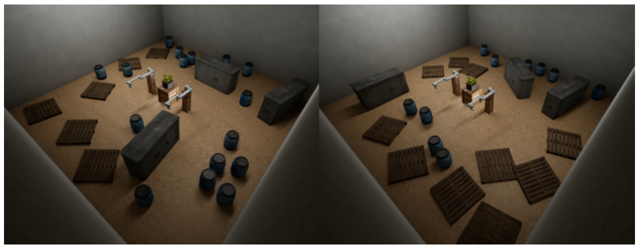
Set Texture Parameter Value
Target is Material Instance Dynamic
Target
Parameter Name: Albedo
Value

(d) Blueprint calls to adjust opacity threshold and segmentation parameters. Segmentation is done by replacing Fig. 3a with Fig. 3c for the material during runtime.

Figure 3. Plant textures and automatic segmentation. Notice the red color in the Segmentation texture corresponds to some of the purple flowers in the Albedo texture. Other purple sections were not labeled because they were outside of the UV map for the Hydrangea and thus unnecessary to edit.

transparent instead of a gradient using a step function because any transparency would lead to color blending in the final renders. Based on this technique, we also assigned multiple labels for a single mesh instance with the texture as shown in Fig. 3c. The end appearance is shown in Fig. 2b. Our method requires a similar amount of annotation effort in the singular label (creating a flat color texture) compared to Cicco *et al*. [6], and does not require additional modifications (turning off lights) in the environment when capturing segmentation images.

We added two of these plant blueprint actors into the scene, one for the RGB view tagged with *ColoredImageGen* and the other for the segmentation view tagged with *SegmentImageGen*. The latter actor was only visible for the segmentation capture without affecting RGB and depth captures, and vice versa for the former actor on the segmentation capture.


(a) *PCG Seed* variations. Left: 0. Right: 1

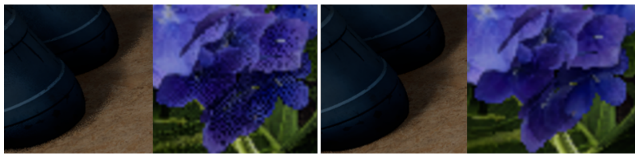

(b) *Raytrace Samples* variations on shadow quality in rendered images. Left: 1. Right: 32

Figure 4. Simulation variations.

### 3.3. Addition Parameters

Many robot simulators take advantage of randomization to increase the environment coverage in their simulations. Data generation using these techniques has been successful in Sim2Real transfer for object detection tasks [26]. We included a rule-based lab environment randomizer using Procedural Content Generation (PCG)[3] in our simulator for deterministic randomization noted as *PCG Seed*. We select cabinet, pallet, and drums from Quixel for indoor background randomization as added image features in our experiments. The rules are as follows:

- Random z-axis (up) rotation with $U(0, 180)$.
- Sampled with [0.1, 0.5, 0.75] per $m^2$ for the cabinet, pallet, and drum mesh.
- Removed mesh overlaps with priority of cabinet, pallet, and last drum.
- The center of the mesh cannot be generated within the middle 6x4m of the room.

We also implemented a *Raytrace Samples* slider to adjust the number of light raytraces between 0-32. This led to variations in shadow quality at the exchange of FPS. We implemented additional arm dynamics options for adjusting the response of joints. Figure 4 highlights some of these variations.

### 3.4. ROS Communication

ROS is commonly used in research on agricultural robot applications, it provides a Pub-Sub architecture to connect and control various sensors in robotics. We used the ROSIntegration plugin [14] to enable ROS topics within the UE5 environment as shown in Fig. 5. We also used ROS Bridge over a local WebSocket to expose UE5 topics in Windows

[3]PCG Plugin: https://docs.unrealengine.com/5.3/en-US/procedural-content-generation-overview/

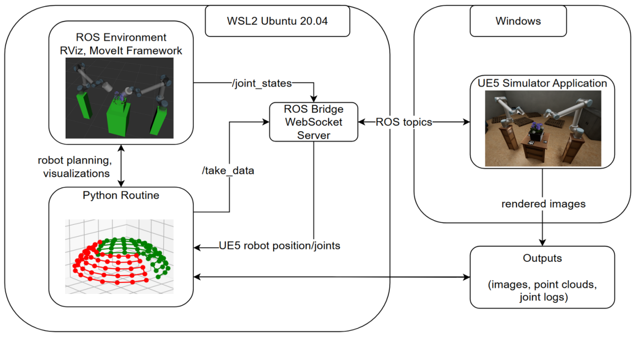

WSL2 Ubuntu 20.04
ROS Environment
RViz, MoveIt Framework
/joint_states
ROS Bridge WebSocket Server
ROS topics
Windows
UE5 Simulator Application
robot planning, visualizations
/take_data
Python Routine
rendered images
UE5 robot position/joints
Outputs
(images, point clouds, joint logs)

Figure 5. System flowchart. The setpoints were predetermined, in this experiment there were 100 setpoints, 50 for each arm with 5 levels using spherical coordinates with $r = 0.5$m, $\phi = \text{linspace}(\frac{\pi}{5}, \frac{3\pi}{7}, 5)$ radians and $\theta = \text{linspace}(\frac{\pi}{19}, \frac{18\pi}{19}, 10)$ radians.

11 to ROS-Noetic topics in WSL2. ROS Bridge is an additional ROS package launched with the custom ROS environment.

The simulator directly subscribed to the */joint_states* topic with additional header parsing to control the joint positions. Each UE5 robot also subscribed to a unique */take-data* topic as a trigger for image generation. To keep track of the UE5 robot position, the simulator also published the current robot position in both cartesian and joint spaces. These topics were started immediately upon running the simulator.

## 4. Experiments

We packaged the simulation into a Windows application to take synthetic images around the plant to generate RGB, segmentation, and depth images. The images were rendered and saved at 1920x1080 pixels for later processing and visualization. The 100 setpoints, including positions and orientations, were predetermined with semi-spherical motion at a 0.5m radius around the plant as shown in Fig. 5. The robot trajectories were then planned during the simulation using the *RRTConnect* planner. The world coordinates of both arms' end effectors in UE5 and ROS were recorded for trajectory comparison. The packaged simulator is around 2.8 GB for Windows build. The simulations were done on an i7-12700k, RTX3060, and 96GB memory desktop.

The simulation used similar parameters from the Raspberry Pi v3 Wide camera model *RaspPiv3Wide(1080p)* with 32 for the *Raytrace Samples* and 42 for the *PCG Seed*. The camera configuration was set with 6.45mm sensor width and 3.63mm sensor height, with a fixed 2.75mm focal length and 2.2 f-stops. However, this gave a horizontal FOV of 99.090668 degrees in simulation, slightly less than the 102 degrees horizontal FOV in the hardware specifications [1].

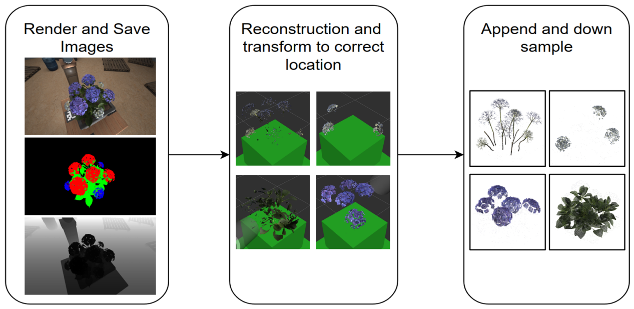


Figure 6. Reconstruction flowchart. Left shows the RGB, Segmentation, and Depth images at one setpoint. Middle shows the intermediate reconstruction $P_{W_i}$ for the four labels. Right shows the corresponding final reconstruction $P_W$.

The Python routine will trigger the cameras to render and save for 1 second upon reaching each setpoint within a 0.05 mm L2 distance between ROS and the simulator. If the robot fails to reach the set point after 5 seconds, we restarted the script with a manual reset for the image count in the simulator to ensure matching image names. We clarify how the dataset quality are evaluated in Secs. 4.1 and 4.2

### 4.1. Semantic Segmentation

To measure the quality of our masking method, we first complete the trajectories in Fig. 5 for each plant model with our segmentation method and the replace opacity (RepOpa) method by checking the *RemoveOpacity* option in the GUI. RepOpa method replaces the opacity mask with a completely white texture (fully opaque) to mimic the EasySynth's technique of replacing the material. We show some end-appearances of RepOpa in Fig. 2c. At the end of each trajectory, the total number of labeled pixels from 100 segmentation images was measured. Since EasySynth cannot handle multi-label segmentations, we only compare the single-label of the ten plants.

Generally, we used a simple green mask for the plants and black for every other object for the entire plant reconstruction. For the Hydrangea plant, we also create a multi-label mask. We used GIMP to create the multi-label mask with the white flowers labeled as blue, purple flowers as red, stems as deep purple, and the leaves as green as shown in Figs. 3 and 6. We only modified sections corresponding to the UV map of the mesh, so not all purple flowers in the texture were labeled.

### 4.2. 3D Reconstruction

We used RGB, segmentation, and depth images collected from Sec. 4.1 runs to create the point cloud for 3D reconstruction. The reconstruction process shown in Fig. 6 first removes any pixels with depth >1m, then masks the depth image using the segmentation image. Using this filtered depth image we reprojected the RGB information to create a point cloud from each setpoint, the camera intrinsic was modeled as a pinhole camera with Eq. (1) and horizontal $FOV = 99.090668$.

We merge the point clouds $P_{E_i} \in \mathbb{R}^{3\times N_i}$ at each location $i$ into one overall reconstruction $P_W \in \mathbb{R}^{3\times N}$, we transform $P_{E_i}$ to the world $P_{W_i}$ with respect to the current pose of the ROS robot's end effector $E$ to the world $W$ as 4x4 transformations $^W_E T$ using Eq. (2). We visualize each $P_{W_i}$ in RViz and append it to an overall reconstruction. Since we use high-resolution captures, we also downsample using 1mm voxels if there are >100k points to obtain $P_W$ as noted by Eq. (3). We compare the total number of points $N$ that exist in the overall point cloud $P_W$. We do not perform any additional denoising beyond the downsampling mentioned above.

$$C_{intrinsics} = \begin{bmatrix} f & 0 & 960 \\ 0 & f & 540 \\ 0 & 0 & 1 \end{bmatrix}, f = \frac{1920}{2\tan(\frac{FOV\pi}{360})} \quad (1)$$

$$\begin{bmatrix} P_{W_i} \\ 1 \end{bmatrix} = {}^W_E T \cdot \begin{bmatrix} P_{E_i} \\ 1 \end{bmatrix} \quad (2)$$

$$P_W = Downsample(Concat(P_{W_i})) \quad (3)$$

#### 4.2.1 Point Cloud Visualization

Since we can edit and visualize point clouds in the UE5 editor through its LiDAR plugin, we repeated the 3D reconstruction procedure on one of the point clouds generated by the simulator. This experiment provides additional insights when using point clouds instead of artistically designed meshes, which can be difficult to obtain in some agricultural applications. However, point cloud assets caused packaging to fail so we used the editor mode to run our experiments in this section.

We used the Coltsfoot point cloud obtained from our method as it has a relatively simple geometry for visualization and clear results. We first import the point cloud in LAS format, then delete the obvious incorrect points using UE5. To enable auto-annotation for point clouds, we place two point clouds in the same location following the tagging procedure in Sec. 3.2. We then change the color of the segmentation point cloud to green. We used the square shape and the *PerPoint* scaling method for the plugin, which sets equal sizes for all points.

### 4.3. Neural Radiance Fields

In addition, we collected RGB images without arm rendering using the same environment and setpoints for Neural Radiance Fields (NeRF) reconstruction implemented by Nerfstudio using the nerfacto model [16, 25]. As a prerequisite step for Nerfstudio, COLMAP [20, 21] is run on all 10 datasets with exhaustive preselection, where matches for 100% of the images were found. To train a nerfacto model on each plant, 90% of the 100 images are randomly selected for training, and 10% randomly for testing. We provide standard metrics including the peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM) for all ten plants. The multi-label Hydrangea was not reconstructed because we used the full RGB images without segmentation and thus unnecessary.

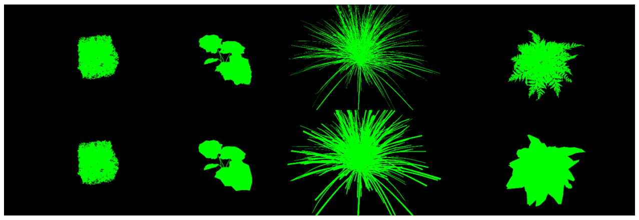
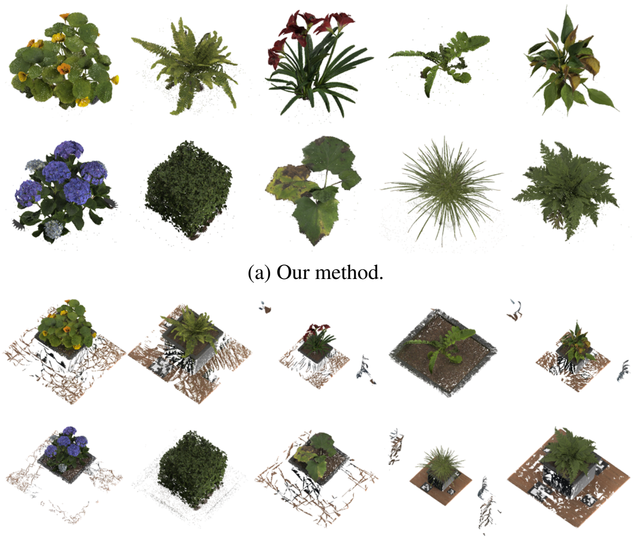


Figure 7. Segmentation images for Boxwood, Coltsfoot, Basket Grass, and Common Fern. Top: Our method. Bottom: RepOpa method.


(a) Our method.

(b) RepOpa method.

Figure 8. Reconstructed point clouds, same ordering as Fig. 2a. See Fig. 6 for multi-label reconstruction of the hydrangea plant using our method.

## 5. Results and Discussion

With our experiment settings from Sec. 4, the simulator runs at around 20 FPS with the application running at a 960x540 window. This is largely due to 32 ray-trace samples, with 0 samples the application could run at the capped 60 FPS. The frame rate drops further (8 FPS) while the robot cameras

|  | Segmentation Image ($10^6$ Pixels) |  |  | Reconstructed Point Cloud (Points) |  |  | NeRF/Nerfacto |  |
|---|---|---|---|---|---|---|---|---|
| Plant | Ours | RepOpa | % | Ours | RepOpa | % | PSNR ↑ | SSIM ↑ |
| Hydrangea | 24.224 | 24.990 | 3.16 | 686045 | 851382 | 24.10 | 26.197 | 0.799 |
| Indian Cress | 28.017 | 29.822 | 6.44 | 495769 | 699976 | 41.19 | 27.021 | 0.816 |
| Amaryllis | 27.483 | 30.713 | 11.75 | 439466 | 718772 | 63.56 | 28.234 | 0.819 |
| Boston Fern | 12.515 | 16.710 | 33.52 | 318746 | 714887 | 124.28 | 17.616 | 0.680 |
| Common Fern | 19.808 | 28.212 | 42.43 | 449209 | 1137598 | 153.24 | 28.632 | 0.828 |
| Bog Marshcress | 3.523 | 4.759 | 35.08 | 93898 | 191003 | 103.42 | 29.007 | 0.842 |
| Boxwood | 14.629 | 14.651 | 0.15 | 810085 | 812073 | 0.25 | 27.710 | 0.810 |
| Coltsfoot | 13.050 | 14.054 | 7.69 | 133002 | 329628 | 147.84 | 28.785 | 0.828 |
| Chinese Evergreen | 30.213 | 34.518 | 14.25 | 356625 | 836139 | 134.46 | 28.392 | 0.820 |
| Basket Grass | 26.029 | 43.561 | 67.36 | 622526 | 1144729 | 83.88 | 26.321 | 0.736 |

Table 2. Comparison of qualities in segmentation, reconstruction using ground truths between two methods. Right also shows the NeRF reconstruction results. Images were collected from the same environment and positions with the Plant as the only variable. ↑ in NeRF columns indicate that higher values show better test set performance [25].

are active with 32 ray traces, which is expected with the additional rendering cost from the 1080p cameras.

We show the visual results for the segmentation images in Fig. 7 and point cloud in Fig. 8, with tabular results in Tab. 2. In addition to raw results, we provide the percentage increase of the RepOpa method to our method using $100(\text{RepOpa}/\text{Ours} - 1)$. The RepOpa results in an overall increase in both categories compared to ours. Combined with the visual results, our method produces more precise segmentation labels than the RepOpa method offered in existing plugin.

We notice there is a significant percentage variation among different plant assets, specifically ranging from 0.15% to 67.36% for the segmentation mask. Boxwood showed the least amount of discrepancy between the two methods at 0.15%. This is likely due to Boxwood's complex mesh compared to others as shown in Tab. 1, thus less dependence on opacity mask for this plant. This is supported by Common Fern and Bog Marshcress having high discrepancies (42.43% and 35.08%) and low triangle numbers (752 and 896), where opacity is used for the details between the leaves. Interestingly, Basket Grass has the highest discrepancy at 67.36% while being a detailed mesh (23850 triangles), which is likely because of the quantity and the shape of the plant structure as shown in Fig. 7.

Table 2 also showed larger percentage differences in point clouds than segmentation across all plant assets. The RepOpa method's imprecise segmentation and voxel downsampling likely amplified this difference in 3D. Figure 8b showed the varying levels of inaccurate reconstructions of these plants compared to Fig. 8a. Boxwood's reconstruction is visually similar while the Basket Grass reconstruction included the entire pot, table surface, and robot arm parts. Generally, the final point cloud is a fraction of the total pixels collected, where none of the point clouds from our method exceeded 1 million points with many plants having >10 million pixel labels. This observation also indicates inefficiency in our robot routine because most ground truth points were discarded for reconstructions at $10^{-3}$ voxels. In our experiment, the number of pixels and points would be similar to indicate an efficient reconstruction process.

| Size | Pixels | Points | Size | Pixels | Points |
|---|---|---|---|---|---|
| 0.01 | 0.079 | 78576 | 0.8 | 15.11 | 371619 |
| 0.1 | 5.84 | 207594 | 1.0 | 15.49 | 440251 |
| 0.2 | 12.87 | 217163 | 1.5 | 16.37 | 493712 |
| 0.4 | 14.27 | 271858 | 2.0 | 17.19 | 785191 |
| 0.6 | 14.72 | 308115 | 5.0 | 21.49 | 1924660 |

Table 3. Reconstruction of Coltsfoot point cloud with varying point sizes. Pixels measured in $10^6$.

## 5.1. UE5 Point Clouds

We also show the Coltsfoot's cleaned point cloud in the UE5 editor using the LiDAR plugin in Fig. 9. We then performed reconstruction using this point cloud with results shown in Fig. 9 and Tab. 3. We notice that except for very tiny point sizes (0.01) where most points were invisible in segmentation images, this type of reconstruction led to an increased number of points in the new reconstruction (>200000) compared to the original (132874). We also notice the thickness of the leaves and stem increases as the point size increases, which likely led to the increased number of points to the original. Leaves and stems are frequently modeled as 2D planes in meshes which typically have no thickness, we believe this technique of using point clouds with varying point sizes could bring depth and thickness traits in synthetic plant datasets generated by a 3D rendering engine.

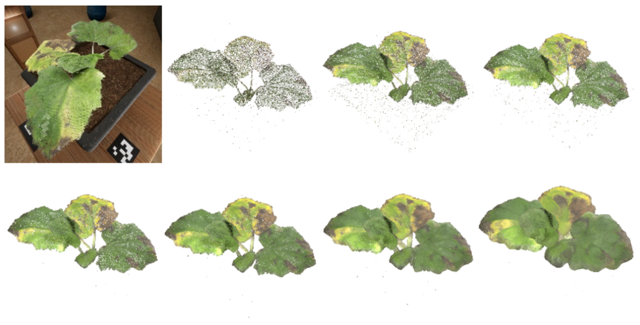


Figure 9. Coltsfoot point cloud reconstruction. Notice the changes in stem thickness. Top left: cleaned point cloud with point size 0.2. From left to right, top to bottom are reconstructions of point sizes: 0.01, 0.1 0.2, 0.6, 1.0, 2.0, and 5.0.

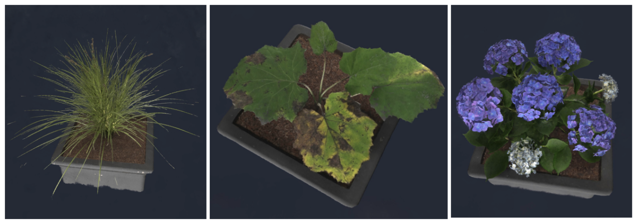


Figure 10. Example NeRF renderings of Basket Grass, Coltsfoot, and Hydrangea, respectively. Plant models were rendered in Nerfstudio at full resolution and cropped to a 1x1x1 box in the viewport. The images shown are screenshots from an arbitrary perspective within the viewer.

### 5.2. NeRF

Example NeRF renderings are shown in Figure 10. Table 2 shows similar metrics across all plants, except Boston Fern, for which COLMAP could not find the correct camera poses for a majority of its training images. The slight differences between the other plants could be attributed to the plant's complexity. We observe the lowest PSNR in high-density leaf structures with Hydrangea (26.197) and Basket Grass (26.321), compared to the highest PSNR among low-density leaf structures with Bog Marshcress (29.007) and Coltsfoot (28.785). With a similar trend in SSIM metric, it is likely that higher leaf density led to lower metrics because of less reference images for individual leaves.

## 6. Future Work and Conclusion

This work presented a dual robotic arm workflow simulated in UE5 and ROS to perform photogrammetry tasks across a variety of ten plants. The simulation environment is packaged as a Windows application with a GUI for runtime adjustments without downloading the UE5 editor and assets. The simulator generated high-quality images with corresponding segmentation labels and depths. We demonstrated the accuracy of the synthetic data through 3D reconstruction experiments with 100 setpoints. We believe the results demonstrated our simulator's potential in various agricultural applications such as plant phenotyping, disease detection, generating synthetic datasets, and robotic simulation.

In future work, we plan to adopt the simulator to other agricultural robotics tasks or optimize the workflow presented in this study to minimize the necessary number of views for reconstruction. Furthermore, we plan to develop an outdoor environment with extensive randomization to simulate more complicated field tasks. It is also in our interest to explore the performance of the simulation routines in the real world.

## 7. Acknowledgements

We thank the funding support from the USDA NIFA Hatch project (7005224).

## References

[1] Raspberry Pi Documentation - Camera. 5

[2] Dafni Anagnostopoulou, George Retsinas, Niki Efthymiou, Panagiotis Filntisis, and Petros Maragos. A Realistic Synthetic Mushroom Scenes Dataset. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 6282–6289, Vancouver, BC, Canada, 2023. IEEE. 2

[3] Quan Huu Cap, Hiroyuki Uga, Satoshi Kagiwada, and Hitoshi Iyatomi. LeafGAN: An Effective Data Augmentation Method for Practical Plant Disease Diagnosis, 2020. arXiv:2002.10100. 2

[4] Taeyeong Choi, Dario Guevara, Grisha Bandodkar, Zifei Cheng, Chonghan Wang, Brian N. Bailey, Mason Earles, and Xin Liu. DAVIS-Ag: A Synthetic Plant Dataset for Developing Domain-Inspired Active Vision in Agricultural Robots, 2023. arXiv:2303.05764. 2

[5] Tim Dellmann and Karsten Berns. Toward a Realistic Simulation for Agricultural Robots. In *Agriculture Digitalization and Organic Production*, pages 3–13, Singapore, 2022. Springer Nature. 1

[6] Maurilio Di Cicco, Ciro Potena, Giorgio Grisetti, and Alberto Pretto. Automatic model based dataset generation for fast and accurate crop and weeds detection. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5188–5195, Vancouver, BC, 2017. IEEE. 1, 2, 4

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 1, 2

[8] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. FlightGoggles: A Modular Framework for Photorealistic Camera, Exteroceptive Sensor, and Dynamics Simulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6941–6948, 2019. arXiv:1905.11377. 2

[9] Dirk Norbert Helmrich, Felix Maximilian Bauer, Mona Giraud, Andrea Schnepf, Jens Henrik Göbbert, Hanno Scharr, Ebba ora Hvannberg, and Morris Riedel. A scalable pipeline to create synthetic datasets from functional–structural plant models for deep learning. *in silico Plants*, 6(1):diad022, 2024. 1

[10] Chengsong Hu, Shuangyu Xie, Dezhen Song, J. Alex Thomasson, Robert G. Hardin IV, and Muthukumar Bagavathiannan. Algorithm and System Development for Robotic Micro-Volume Herbicide Spray Towards Precision Weed Management. *IEEE Robotics and Automation Letters*, 7(4): 11633–11640, 2022. 1

[11] Marcelo Jacinto, João Pinto, Jay Patrikar, John Keller, Rita Cunha, Sebastian Scherer, and António Pascoal. Pegasus Simulator: An Isaac Sim Framework for Multiple Aerial Vehicles Simulation, 2023. arXiv:2307.05263. 1

[12] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, pages 2149–2154 vol.3, 2004. 1, 2

[13] Yajun Li, Qingchun Feng, Yifan Zhang, Chuanlang Peng, Yuhang Ma, Cheng Liu, Mengfei Ru, Jiahui Sun, and Chunjiang Zhao. Peduncle collision-free grasping based on deep reinforcement learning for tomato harvesting robot. *Computers and Electronics in Agriculture*, 216:108488, 2024. 1

[14] Patrick Mania and Michael Beetz. A Framework for Self-Training Perceptual Agents in Simulated Photorealistic Environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4396–4402, 2019. 4

[15] Hasib Mansur, Stephen Welch, Luke Dempsey, and Daniel Flippo. Importance of Photo-Realistic and Dedicated Simulator in Agricultural Robotics. *Engineering*, 15(5):318–327, 2023. Publisher: Scientific Research Publishing. 1

[16] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 6

[17] Hugo Moreno, Adrià Gómez, Sergio Altares-López, Angela Ribeiro, and Dionisio Andújar. Analysis of Stable Diffusion-derived fake weeds performance for training Convolutional Neural Networks. *Computers and Electronics in Agriculture*, 214:108324, 2023. 1, 2

[18] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondruš, Theophile Gervet, Vincent-Pierre Berges, John M. Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A Co-Habitat for Humans, Avatars and Robots, 2023. arXiv:2310.13724. 2

[19] Farah Saeed, Jin Sun, Peggy Ozias-Akins, Ye Juliet Chu, and Changying Charlie Li. PeanutNeRF: 3D Radiance Field for Peanuts. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 6254–6263, Vancouver, BC, Canada, 2023. IEEE. 1

[20] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6

[21] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 6

[22] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, 2017. arXiv:1705.05065. 1, 2

[23] SIGGRAPH Advances in Real-Time Rendering. A Deep Dive into Nanite Virtualized Geometry, 2021. 2

[24] SIGGRAPH Advances in Real-Time Rendering. Radiance Caching for Real-Time Global Illumination, 2021. 2

[25] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–12, 2023. 6, 7

[26] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, 2017. arXiv:1703.06907. 4

[27] Yosuke Toda, Fumio Okura, Jun Ito, Satoshi Okada, Toshinori Kinoshita, Hiroyuki Tsuji, and Daisuke Saisho. Training instance segmentation neural network with synthetic datasets for crop seed phenotyping. *Commun Biol*, 3(1):1–12, 2020. Publisher: Nature Publishing Group. 2

[28] Joanne Truong, Max Rudolph, Naoki Yokoyama, Sonia Chernova, Dhruv Batra, and Akshara Rai. Rethinking Sim2Real: Lower Fidelity Simulation Leads to Higher Sim2Real Transfer in Navigation, 2022. arXiv:2207.10821. 2

[29] YDrive. Easysynth. https://github.com/ydrive/EasySynth, 2024. 3

[30] Jing Zhang, Xin Wang, Xindong Ni, Fangru Dong, Longrunmiao Tang, Jiahui Sun, and Ye Wang. Neural radiance fields for multi-scale constraint-free 3D reconstruction and rendering in orchard scenes. *Computers and Electronics in Agriculture*, 217:108629, 2024. 1