A Scalable Pipeline to Create Synthetic Datasets from Functional-Structural Plant Models for Deep Learning

Dirk Norbert Helmrich[1,2], Felix Maximilian Bauer[3], Mona Giraud[3], Andrea Schnepf*[3], Jens Henrik Göbbert[2], Hanno Scharr[4], Ebba Þora Hvannberg[1], and Morris Riedel[1,2]

November 15, 2023

* Corresponding author, a.schnepf@fz-juelich.de
[1] School of Engineering and Natural Sciences, University of Iceland, Reykjavík, Iceland
[2] Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany
[3] Institute of Bio- and Geosciences 3 (Agrosphere), Forschungszentrum Jülich GmbH, Germany
[4] Institute for Advanced Simulation 8 (Computer Vision), Forschungszentrum Jülich GmbH, Germany

Keywords: FSPM, Deep Learning, Visualization, HPC, Synthetic Data, Computer Vision

Abstract

In plant science it is an established method to obtain structural parameters of crops using image analysis. In recent years, deep learning techniques have improved the underlying processes significantly. However, since data acquisition is time and resource consuming, reliable training data is currently limiting. To overcome this bottleneck, synthetic data is a promising option for not only enabling a higher order of correctness by offering more training data, but also for validation of results. However, the creation of synthetic data is complex and requires extensive knowledge in Computer Graphics, Visualization and High-Performance Computing. We address this by introducing *Synavis*, a framework that allows users to train networks on real-time generated data. We created a pipeline that integrates realistic plant structures, simulated by the functional-structural plant model framework CPlantBox, into the game engine Unreal Engine. For this purpose, we needed to extend CPlantBox by introducing a new leaf geometrization that results in realistic leafs. All parameterized geometries of the plant are directly provided by the plant model. In the Unreal Engine, it is possible to alter the environment. WebRTC enables the streaming of the final image composition, which in turn can then be directly used to train deep neural networks to increase parameter robustness, for further plant trait detection and validation of original parameters.

We enable user-friendly ready-to-use pipelines, providing virtual plant experiment and field visualizations, a python-binding library to access synthetic data, and a ready-to-run example to train models.

1 Introduction

Machine Learning (ML) algorithms usually perform well when trained on large quantities of data well covering the input space. Deep Learning (DL) techniques are a subset of ML and utilize the training of many-layered compute graphs. Pound et al. (2017) show that DL techniques have the highest performance on plant image analysis, which in turn has been established to be a significant bottleneck for plant phenotyping (Yang et al., 2020; Kamilaris and Prenafeta-Boldú, 2018; Tsaftaris et al., 2016; Scharr and Tsaftaris, 2022). Many image-based applications for plant phenotyping involve semantic or instance segmentation, i.e., associating each pixel with a class label such as organ type and/or organ numbering (Scharr et al., 2016; Yang et al., 2020; Tsaftaris et al., 2016). In terms of DL in biological image analysis, data is often rare and hard to extract from real-world measurements (Pound et al., 2017). This is due to the high variability of environmental conditions, including light conditions, rain, soil types, stresses influencing plant appearance even for the same genotype, and the high intrinsic variability of plants and their changing appearance over time due to growth or senescence. Even lab condition data cannot always be acquired in optimal circumstances and cannot easily be reproduced as plants would need to be regrown. Consequently, data sets acquired over years of intense measurement campaigns are often heterogeneous and can only cover small parts of a large data space. Therefore, developed plant image analysis solutions are typically highly specialized, e.g., for specific organs and/or plant types, and do not generalize broadly (Lobet et al., 2013). In most cases the full potential of the data remains unused. Analysis methods which exploit large quantities of heterogeneous data sets, covering a more significant part of the data space, would be highly beneficial for robustness and generalization.

DL frameworks and algorithms can address data analysis challenges but need a lot of data to perform well. Synthetic data generation can provide arbitrary amounts of well-annotated data, thus enabling the scaling of DL methods to a more powerful capacity for generalization and accuracy. Previous approaches showed that introducing synthetic data makes high-quality training examples available at low cost (Zhang et al., 2020; Pollok et al., 2019). The generation of versatile synthetic data, however, requires a lot of inter-domain expertise on plant biology as well as computer graphics and HPC. Solutions to generate more data for specific tasks often end up highly specialized, such as for 2D image generation approaches by Ubbens et al. (2018), who generate leaf images for counting tasks or 2D root rasterization with noise components as implemented by Lobet et al. (2017). Similar approaches for root systems have also been implemented by Benoit et al. (2014) or the 3D voxelization approach by Masson et al. (2021). Functional-Structural Plant Models (FSPMs) produce, using cultivar-specific input parameters, plant morphological and topological data that can then be used to produce synthetic images. To assist with making FSPM rendering more realistic, some approaches also include physics-based surface simulation and reaction to light spectra (Bailey, 2019). In contrast, Hartley and French (2021) address the potential synthetic-to-real data gap by applying domain adaptation using image generation, such as Gao et al. (2023). Other approaches additionally use few-shot learning, i.e., a limited number of observations/samples, and transfer learning to bridge the gap between synthetic and authentic images (Zhang et al., 2020). An example application where synthetic data can be very impactful are rhizotron experiments, which is a method for whole-plant measurement that requires plants to be grown in specialized containers, such as produced by Nagel et al. (2012). The measurements of this type of special set-up is resource consuming and produces accurate but little data. The significant overhead for a single plant measurement causes challenges for the DL models that need to work on limited data as well as a need to process the present data with as much accuracy as possible.

To use the full potential of synthetic data in the training of DL approaches, we developed

Synavis[1], a coupling framework that is versatile enough to allow different approaches and makes use of HPC to allow an interplay of FSPMs, visualization, and DL model training. We developed a pipeline using visualizations of the FSPM CPlantBox (Zhou et al., 2020) and the Synavis software, enabling the coupling of the FSPM with Unreal Engine (UE) [2] and the DL framework. Synavis handles dynamic workflows with UE and is scalable for HPC systems. The combination of CPlantBox with Synavis in UE also allows for the easy addition of wind, leaf diseases imposed onto the leaf texture, and other effects such as rain or degraded image quality. The primary goal of this work is to improve image analysis processes for plant phenotyping and model parameter extraction. To support the data generation for these difficult tasks, there are bottom-up approaches to directly simulate surface radiation (Bailey, 2019) and top-down approaches aimed at replicating measures (Bouvry and Lebeau, 2023). Synthetic data can also be generatively produced to assist with specific tasks, such as leaf segmentation (Ward et al., 2018).

This article highlights the central points of our contribution, which are centered around three main axes. We implemented a geometrization and texturization for FSPM Visualization with CPlantBox. Furthermore, we introduce our Synavis framework that enables the integration of CPlantBox models into UE in an HPC compatible method for the purpose of flexible synthetic data generation. We test our approach against experimental data by implementing the specific use case of rhizotron experiments to demonstrate feasibility and performance. Furthermore, we detail the field scale rendering in our virtual environments and what aspects of the pipeline give this method advantages over other approaches.

1.1 Description of the CPlantBox FSPM

FSPMs describe digitized versions of a phenotype (Soualiou et al., 2021) of a plant, providing means of assessing interventions, crop combinations, photosynthesis assessments, and even nutrient and soil interaction. They can provide insight into *in vivo* counterparts by providing access to more measures and modeled information, making them ideal digital twin candidates. The interaction between the *in silico* and *in vivo* versions of plants can provide valuable insight, especially concerning possible success from interventions, as shown by Perez et al. (2022).

CPlantBox (Zhou et al., 2020; Giraud et al., 2023) is a modeling framework for FSPMs based on the graph formalism as seen in Figure 1. Plants are described as a series of vertices linked by edges describing the abstract morphology of the plant. The plant object stores moreover a series of arrays which can contain any kind of necessary information for each point or edge, such as radius or age. CPlantBox is a stochastic model, where all parameters are defined via their truncated normal distribution. CPlantBox can be used to generate raw plant structural data mimicking various plant development dynamics under specific environmental conditions. Parameterization of CPlantBox is illustrated in Figure 1 and can be done in two ways: Calibration of the model are sometimes done directly from experimental data, such as in Bauer et al. (2023), who investigate the effect of phosphorus deficiency in *Zea mays* plants. Another approach is measuring a target distribution from experimental data, and running an estimator for the posterior distribution of the parameter space w.r.t. the target distribution, as done by Morandage et al. (2021) in their work on analysing how well model parameterizations can fit onto synthetic field data. Additionally, CPlantBox can be coupled with other models that simulate dynamic soil or atmospheric conditions. Such models can exhibit growth properties that are grounded in, for example, nutrient availability, such as developed by Giraud et al. (2023).

[1] Synavis Framework: https://github.com/dhelmrich/synavis
[2] Unreal Engine, https://unrealengine.com/

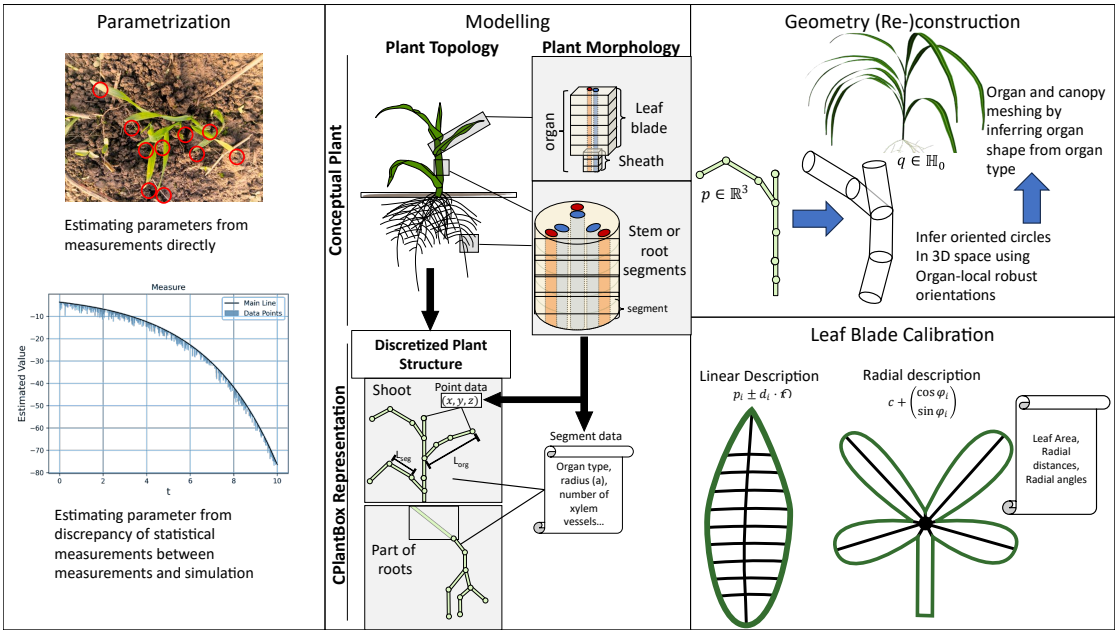

Figure 1: Overview of CPlantBox Parametrization, adapted from Giraud et al. (2023), Simulation of Structural and Functional Properties, and Visualization. Parametrization exemplifies different measuring techniques - direct and through stochastic optimization. Modelling shows our approximation model for the true plant shape, which is the discretized point/edge model. We further introduce two options for leaf-blade calibration. The visualization approach introduced with this work infers a 3D plant shape from the graph-like structure by assuming that by default, there are no drastic local changes to the rotation.

2 Methods

2.1 Modelling CPlantBox in Unreal Engine

As one of the cornerstones of our synthetic plant data pipeline, this section will detail our 3D meshing and the modeling framework for texturing plants in SynavisUE.

For the generation of the CPlantBox geometry, as seen in Figure 1 on the right, we implemented a visualization module that produces triangulated meshes from the CPlantBox data. For this, we needed to write a geometrization scheme that could convert the point/edge graph structure into a mesh. The mesh is then rendered in UE as object that can be placed in the scene using Synavis.

Generally, models of the leaf surface should incorporate its morphological properties. It greatly enhances the realism and visual quality of the rendering, as was shown by Wang et al. (2006). While there are different established ways of producing and texturing leaves in graphics engines, one of the more common methods is masking. This technique displays a rectangle that contains a texture with an binary mask that describes the leaf area. As exemplified in Bailey (2019), masking the leaf surface as a texture is a good tool of fast creation of varying plant shapes as the masking is very cheap in both ray-tracing and rasterization renderers. It has drawbacks, however, as modern graphics engines such as UE might introduce additional geometric features, e.g. Nanite Hierarchical Culling (Karis, 2021), that might change masked geometries more than

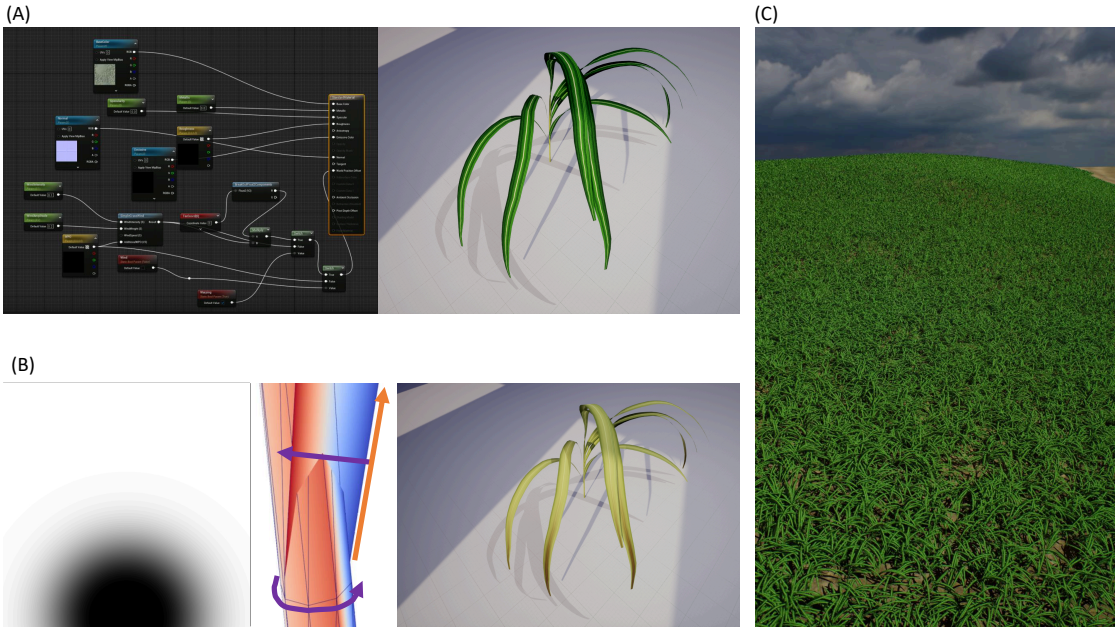

Figure 2: A. We model the material, i.e., the shader language graph representation, in UE for sample textures to make sure both dynamically set parameters and operations on textures are feasible. B. Starting from this graph, we dynamically instantiate not only surface color textures, but also additional effects, such as normal maps or opacity masks. Example shows use of composition, with discoloration masks and parameters set to 0 by default (such as in A), which can by used during runtime to generate additional effects, such as discoloration. C. UE is capable of handling a lot of geometries. While there are diverse geometry pipelines supported, Synavis uses procedural meshes as they directly accept geometry buffers. To differentiate between organs, their individual geometries have to be transmitted separately into mesh sections.

their unmasked counterparts, or their rendering does not accurately reflect the geometry of the plant as optimizations are made. This leads to a discrepancy between the masked leaf shape and what is visible in the scene. Another approach is to declare a full triangulated geometry for the shape of the plant, such as performed by Yun et al. (2019) for simulated Light Detection and Ranging (LiDAR) data (Behroozpour et al., 2017). In this work, we use the latter approach for producing leaf shapes. The advantage of this is that there exists a pre-rendering correct geometry that DL approaches can train against. Our approach infers an orientation from the point position by either assuming that the local forward vector (i.e., in growth direction) is upright or by branching off from the parent organ. For leaf meshing, this orientation must be robust, as the inferred orientation at the origin point of the leaf blade has more extreme effects at the sides of the leaf. The leaf blade is scaled using the CPlantBox parameter sets that also contain its description. We detailed the CPlantBox geometrization approach in Appendix 4.4. Figure 1 Right also shows how the leaf area templates are saved in the parameter file, either as linear description as distances from the midline or as radial distance-angle pairs. Leaf blade distances are generally shape-only parameters and are pending the scaling using the leaf area parameters or growth stage.

For the visualization of CPlantBox plants, textures are warped onto the geometry. This means that there is an internal relative coordinate system spanning the leaf surface. This projection of the surface onto the texture is similar to some world map projections. This allows for more

pixels at the leaf tip and the start of the leaf, where often more complex surface properties are exhibited as structures merge and leaf veins flow together. Leaf textures either stretch the whole leaf length or are automatically repeated by the UE. However, graphics engines also support the creation of rectangular textures that again provide more resolution along the leaf length. Masking the leaf texture comes at the cost of potentially not allowing for enough pixel space at potentially crucial parts of the leaf. With the warping of the leaf texture, we also make maximum use of the texture resolution and the texture buffers on the GPU. Moreover, there is an obvious midline on the leaf, which is exactly the $m : \mathbb{R} \ni x \mapsto (0.5, x) \in \mathbb{R}^2$ coordinate line in the texture. For masked textures in contrast, there always needs to be a fixed starting point and the midline of the leaf might not be obvious. Positioning leaf sickness effects or structural defects such as holes can be done by generating the appropriate discoloration and transmitting the texture as buffer to SynavisUE.

Stem textures are similarly warped, but the texture coordinates for cylindrical plant organs are wrapped around the axis counter-clockwise. Generally, this behavior of the system can be adapted by changing the geometrization, but mapping discolorations onto parts of the plant is easier when using a normalized coordinate space.

Figure 2 gives an overview over the individual steps involved. Users can achieve clearer data mapping and visualization when using warped data-containing textures as opposed to vertex colors. The surface modelling approach for the Synavis framework mainly rests on the material shaders in UE as shown in Figure 2A. The structure of the surface descriptions allows for dynamic parameters that can be set during runtime, such as in Figure 2B. Parameter ranges can be investigated within UE, such as how discolorations are projected onto the surface, while actual data augmentation is done through Synavis. Parameters introduced in shaders will be available for runtime adaptation. Visual editing modes in UE are helpful for validating value ranges and assessing how the material reacts to changes in values. This templated material increases the possible training data space significantly. Figure 2B shows this with the example of a simplistically generated texture mask that is superimposed onto the leaf, generating a discoloration of the leaf surface. Field settings, such as Figure 2C rely heavily on stochastic augmentations of either geometry or material, which are alterations of the basic parameters with random chance. Plant textures used in this example were generated using simple generative color filters and calibrated based on image data from experiments shown in Figure 5. Soil and environment texture choice depends on what is being trained. If it serves only secondary purpose, we refer to either free-to-use scanned textures[3], or paid custom asset collections[4].

The main difference of this work, seperating it from all previous approaches, are three main points. Synavis enables a *versatile* workflow that works well on plant data generation, but is not restricted to it, as it can be added to any existing project. Furthermore, our approach is provable to be *scalable*, with its primary design being catered towards its use on HPC systems. Lastly, we aimed to make it more *accessible* by using the default protocols and behaviors of UE, such as WebRTC (Jennings et al., 2021). An overview of the framework components can be seen in Figure 3.

As shown in Figure 3.A, Synavis handles the connection and communication between UE and the DL training frameworks. Furthermore, Synavis was designed as bridge service similar to the one described in Reddy et al. (2020), connecting compatible endpoints in a modular supercomputing architecture where software might run most optimally on specialized hardware (Suarez et al., 2019). An important feature is the steering of the virtual scene using JavaScript Object Notation (JSON) commands, as in Figure 3B. Synavis also integrates well with UE without requiring a direct coupling to it. The network interfacing used by this method follows a

[3]Examples include Quixel Megascans
[4]Unreal Engine Marketplace

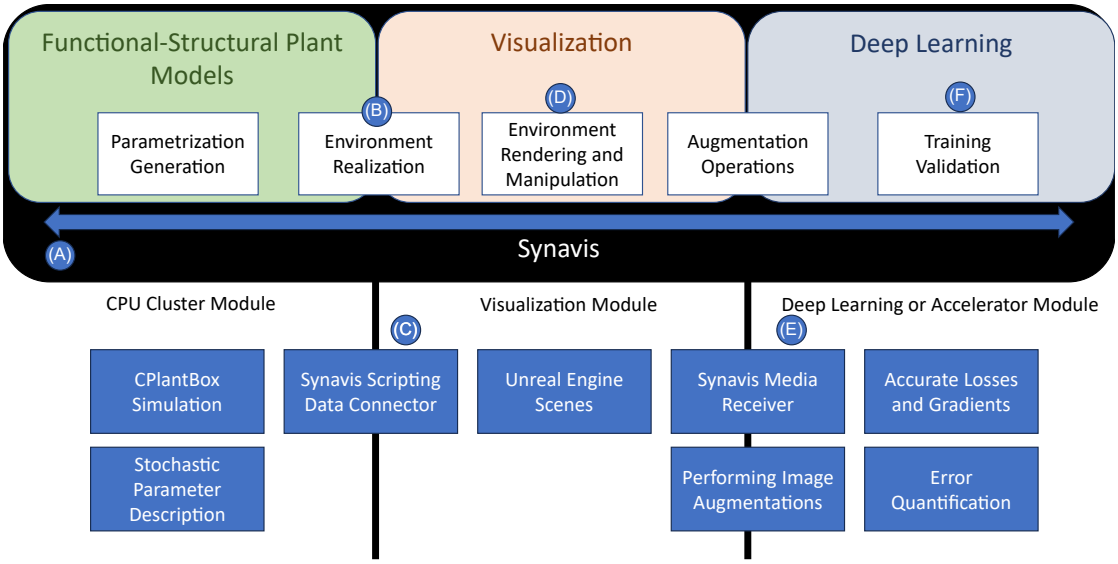

Figure 3: Overview of the total workflow setup. Synavis is a coupling framework that utilizes standardized communication across a supercomputing network. Oftentimes, different applications must run on specific architectures. This include highly parallelizable algorithms such as backpropagation for DL model training. In contrast do visualization, these can also run on accelerator compute nodes that mainly provide tensor cores, which in turn cannot provide rendering support.

standard and provides simple tools. Moreover, this framework allows the coupling of UE to many different endpoints - providing simulation data, geometry, textures, commands, information for training, or communication with the simulation.

During our investigation of current practice, we uncovered some key workflows that we wanted to support, with tangible applications either in demand, or already in use: Firstly, we provide access to otherwise inaccessible data for training workflows using synthetic data, such as depth estimation (McCormac et al., 2017) with UE, as was already shown to integrate well with the underlying rendering pipeline (Jansen et al., 2022). Secondly, we use UE to digitally mimic the environment for synthetic data such as light interception (Kim et al., 2020), or scalable workflows to meet the rising demand for high-quality data (Pound et al., 2017; Scharr et al., 2016; Qiu et al., 2017; Zhang et al., 2020).

The central theme in these use cases is the combination of complex frameworks and very domain-specific workflows. Especially in plant sciences, in combination with visualization and DL, it can be a challenge to overcome the technical requirements of different systems and users. Synavis is a pathway for collaborative use of these techniques. Incorporating a specific visualization for an FSPM, plant scientists can dictate the look and structure of the virtual scene at runtime.

2.2 Central Concepts of Synavis

Synavis uses the standardized WebRTC communication method as well as a command structure based on JSON. Generally, WebRTC is a framework to couple participants in real-time communication. This means that the communication is always non-blocking and reception, while

possibly assured through certain protocols, does not occur in any predictable order. This setup is preferable when messages are sent by peers at the same time and an ordering might not be possible.

Figure 3C shows that CPlantBox is connected to UE via a *Data Connector*, which is the main component that accommodates the possibility of transferring data to and from UE. The data connector also introduces the possibility of connecting models directly to each other or to the DL framework. For communication with UE, the data connector provides functionalities consisting of a base set of commands that are pre-integrated to enable many simple workflows. As a method of communication, the data connector uses a WebRTC concept called *data channel*, which is the direct peer-to-peer message passing connection that is the minimum required setup for a connection with the framework. Enhancing the data connector with a *media track* yields a setup similar to a video conference client receiving a video stream, which in Figure 3.D serves as a UE-DL connection. This type of coupling class is called *Media Receiver* in Synavis and its added functionality is the capturing of video frame packages (Schulzrinne et al., 2003). The media receiver also offers an easy coupling to decoding services that work with streams, such as OpenCV with GStreamer (Nimmi et al., 2014). The setup of such an example can be found in Appendix 4.4.

Lastly, communication through the Synavis framework uses JSON. This method of communication encapsulates data well without requiring a lot of meta text and is the default for the Pixel Streaming Plugin as well. An example of a simple prompt would be `{"type":"query"}` , which prompts UE to relay the list of objects within the virtual scene. All objects are accessible that are present within the reflection system of UE. This system is a separate data structure generation system that contains meta information on classes, properties, and objects of code definitions in UE.

2.3 SynavisUE Visualization

UE is a graphics engine that has been in development since 1995 (Sanders, 2016). Apart from game development, it has uses in scientific and industrial disciplines (Qiu et al., 2017; Zhang et al., 2020; Bondi et al., 2018). We are using UE in combination with Synavis to generate virtual scene data. Using a graphics engine such as UE has a number of advantages. Focusing on the ones that are directly relevant to our workflow, we want to highlight that (i) UE is capable of handling scripted runtime manipulation while offering a system for fast prototyping through editors. Whereas digital twin models require transference of measurements, the engine itself further enables the design and use of virtual environments that are designed rather than programmed - the user interface will make it much easier for most users to arrive at a visually realistic environment. Large scale environments produced by the engine have already seen some synthetic data uses, such as the Unmanned Aerial Vehicle (UAV) approach by Bondi et al. (2018).

(ii) UE furthermore uses a reflection system which this framework makes heavy use of, as highlighted in Figure 3.E. Reflection systems are generally helpful in counting or selection tasks, retrieving properties of objects, and tracking objects. UE also grants access to the GPU buffers that contribute to the rendering of the final image, such as distance buffers. These buffers provide an expedient way of accessing distance and segmentation information, which has already been taken advantage of by approaches such as by Jansen et al. (2022).

PixelStreaming is a plugin that also changes the render pipeline of UE to include an encoding step, providing a video stream of the virtual scene. Typically, the *final image* buffer is the last step of the rendering scene, preceeded by pixel shaders. This image is usually transmitted to the display and discarded. The plugin handles the final image in an another step, encoding a video stream on the graphics card using the H264 (ISO/IEC 14496-10:2022, 2022) or H265

(ITU-T H.265, 2023) encoding standards. There are alternate CPU-based encoders that have performance drawbacks but increased compatibility, such as an arbitrary number of encoders. The encoded image is transported using Real-Time Protocol (Schulzrinne et al., 2003) packages. The plugin enables easy and web-based access to a stream from a data center's backend. We are using this pipeline to enable our coupling between the visualization with UE and the DL pipeline. We developed a managed way of coupling video streams with DL frameworks through a framework akin to a kind of relay server (Reddy et al., 2020). This standard also provides us with means of inserting data into UE.

2.4 Training with Synavis

The central techniques we will illustrate in this work, that enable workflows driven by Synavis, are *tracking*, *segmentation* and *mapping*. This relates to Figure 3.F.

Tracking is a method of obtaining frame-wise information about elements of the virtual world at each point in time. The temporal resolution of this information is ultimately dependend on the frame time of the engine. However, the temporal resolution of the tracked information always matches the temporal resolution of the world. Prompting tracked information is done using a JSON prompt that triggers the transmission of the tracked information through the data channel. The information is sent every frame, and to compensate unordered arrival of the data channel messages relative to the video track frames, a timestamp is added to the messages to provide ordering information. As this timestamp is also present in the video track messages (Schulzrinne et al., 2003), the messages can be ordered. In our use-cases, we train through the use of an intermediate image buffer. This allows the training to progress through the data set at any speed, decoupled from the rate of image production. While this is not practically an issue, it does alleviate some scheduling and fetching delay concerns for supervised learning.

Segmentation and mapping are both techniques that require pixel-level information. UE can deliver semantic segmentation maps via multiple techniques, specifically a separate depth rendering pass for segmentation maps. This way, objects can be assigned an ID which will be rendered to ground truth by SynavisUE. Mapping on the other hand, is largely related to mapping properties onto the scene, on the geometry level or on population level. Plant surfaces can contain values mapped onto colors much like in standard visualization pipelines. In Synavis, these additional information carriers are then marked as visible only to the *information camera*. Alternatively, the information can also be mapped on patches of the image rather than the actual plant geometry. This way of generating ground truth is also possible remotely within Python.

Overall, Synavis offers direct callbacks for frame reception or alternatively, rerouting into decoding frameworks, such as GStreamer, which is commonly used for remote video platforms (Nimmi et al., 2014; Ahmadi et al., 2016). The base implementation to couple Synavis to a DL framework is implemented as a buffering of images, such that the reception and decoding does not limit throughput in the training framework. This handling, however, is implemented within the DL framework and while Synavis offers a template on how to handle this, there are many hyperparameters that limit generalization capacity of any single implementation, such as output resolution, or whether images or a video stream is needed. Image augmentations that are being considered best practice (Kuznichov et al., 2019) should still be used, albeit as a precursor to fetching the next image as opposed to an operation on the data set. In this way, the actual image is drawn from a selection of equally likely possibilities $I \in \left\{I, I^T, (-x, y)_{x,y\in I}, (x, -y)_{x,y\in I}, ...\right\}$. This augmentation should be random depending on rank, such that the individual instance will not usually receive the same image during training.

3 Results

The creation of synthetic data for model training is useful because it is scalable (Scharr et al., 2016; Qiu et al., 2017). The strongest advantage, even surpassing any potential synthetic-to-real data gap, as exemplified by Zhang et al. (2020), is the ability to generate varying data of different visual properties, while retaining exact information on training labels. A non-exhaustive list of randomization options can be found in the Appendix 4.4, where remarks are included on how and when to introduce stochasticity to virtual scenes.

We generally recommend making as much use of stochasticity as possible. A strong reason for this is the fact that previous investigations into learning behavior of DL models found that more general pre-training greatly improves later out-of-distribution performance (Jitsev, 2021).

We note that for entirely custom scenes, UE provides a full-featured 3D editor as well as integration of common architectural data types, such as computer-aided design (CAD) drawings. Our framework furthermore targets runtime-creation of scenes. Initializing and coupling Synavis to UE, or its respective SynavisUE plugin, the data channel connection provided is capable of handling operations such as spawning and scene manipulation. SynavisUE allows a combination of pre-made assets and scenes with runtime generation of objects, and anything on the scale of purely one or the other. Steering of the scene is done using the Python bindings of Synavis, which allows for the direct interaction of the DL framework with the virtual scene, including the training of agents using reinforcement learning.

3.1 Setting up Data Generation for Scalable Use Cases

Important aspects of the rendering should be changeable. It is important to note that in a lot of cases, such as UnrealPerson (Zhang et al., 2020), synthetic-to-real transfer learning yields better results than simple application of a synthetically trained model onto real-world data. As such, we setup HPC-targeted workflows and highlight how the different aspects of the scene can be changed to target scalability rather than exact replication. UE uses a game loop that constantly produces new images as part of the rendering.

Through the reflection system, any registered parameter and function can be accessed. Parameters can be changed, and Synavis can spawn geometries as well as textures. Synavis was further designed to make educated guesses as to the properties of objects, so that scene setup is fast while retaining versatility.

Users should change positions, materials, geometries if applicable as well as the camera properties, as this further introduces data diversity. More possibilities and descriptions can be seen in Appendix 4.4.

3.2 Pipeline Performance on the JUWELS Booster

In our measurements, we focused on the performance of the individual pipeline components that contribute to the total performance. As such, we measured the *field-filling* capacity of CPlantBox, meaning the timing of the generation of enough plants to realistically fill a plant field in Table 1.

Furthermore, we measured the performance of Synavis for different message passes. We measured the performance of Unreal Engine in dealing with the rendering of a large-scale plant field, as is also shown below in Section 3.3. The results of the frame time measurements can be seen in Figure 4.

Queries for objects or properties are delegated to either the reflection system or the object management system. Many modern game engines run such a system under-the-hood, and specifically in the case of UE, this allows for a functionality extension that is otherwise not possible.

An important factor of scalability is making use of the network topology, namely using the fastest network available on the supercomputer. This means that all services have registered their expected communication via the InfiniBand network interface (Pentakalos, 2002). We refer to literature (Kesselheim et al., 2021; Krause, 2019; Alvarez, 2021) for an in-depth overview of the supercomputing system that we are using. One the Jülich Wizard for European Leadership Science (JUWELS), one cluster compute node has two Intel Xeon Platinum 8168 CPUs. Booster compute nodes additionally have four NVIDIA V100 GPUs. Furthermore, the JUWELS supercomputer has, in addition to its Ethernet network, an InfiniBand high-performing high-throughput network that is entirely unrestricted. The signalling server included in the Synavis framework also contains methods of automatically detecting and using the InfiniBand network. Overhead is introduced by the layers of application management. These layers are the Pixel-Streaming plugin itself, the transport layer using WebRTC, and also the message passing management by Synavis and its handling on the Python side. This amounts to an average of 0.1 seconds per message ($\sigma = 7.79E-05, n = 1000$). The low standard deviation shows that the main delays are in fact the traversal of the communication layers. This average is also a total of diverse messages, but the individual categories are not very different, such as for a simple query in $\sim N(0.1003, 8.771e-05, n = 250)$ seconds in contrast to transmitting geometry in $\sim N(0.1004, 8.01e-05, n = 250)$ seconds.

For complex scenes and geometries, it is also important to understand the underlying processes that happen within UE during scene generation. To showcase UE performance on JUWELS, we ran a measurement of the complete engine tracing[5] on the JUWELS booster module. The setup for this tracing is simple and can be seen in the listing in Appendix 4.4. The timings of certain UE processes can be seen in Table 2.

The configuration of this test can be seen Appendix 4.4. UE performs on the JUWELS cluster using the installed compatible graphics interfaces and NVIDIA drivers. The program is packaged in Windows and only uploaded to the cluster. Alternatively, UE packages can also be generated within the cluster environment, using the shared memory as build directory.

For larger fields, plant generation might be an issue. We measured how long the FSPM CPlantBox, callibrated on lab-experiment grown plants, needs to generate one million individual geometries. This includes the loading of the parameters and full FSPM simulation, which also creates a single realization of the stochastic parameter set, as well as our geometry generation module. The measurements can be seen in Table 1 and were performed on the JUWELS CPU module.

3.3 Synavis for Augmentation of a Rhizotron Experiment

To validate the Synavis framework and showcase that it correctly replicates experimental[6] data, we first produced a virtual replica of a rhizotron experiment seen in Figure 5 and then scaled the CPlantBox model up to field scale in Figure 6. There is a strong bottleneck in data acquisition for rhizotron experiments as the effort to produce data is very high: Rhizotrons need to be filled, seeded, and stored during the measurements and most of the processes must still be done manually. This causes a large overhead for the data acquisition that is not comparable to field data, as field data can be seeded in bulk and the restrictions on how to grow the plants are less limiting. To repeat a rhizotron experiment many times, either a lot of resources are needed to run the experiments in parallel, or an long time is needed for the back-to-back repetitions of the individual experiments. Moreover, it is necessary to achieve the proper environmental conditions for each expermental run. Therefore, often only a very limited number of shoot and root images

[5] Unreal Insights Tracing
[6] Data Source: Repository of CPlantBox

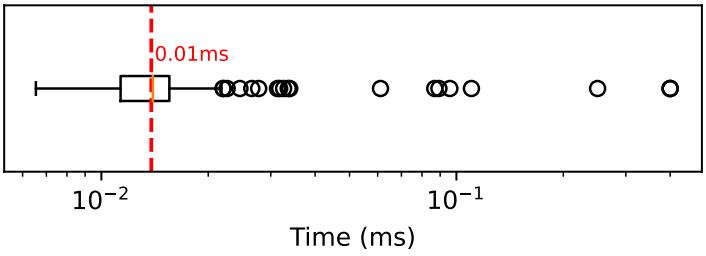

Figure 4: Performance measurement on field scene in Unreal Engine, see Figure 6. This measurement was done on the JUWELS Booster with about 340K plants and a total of 80M triangles. The average frametime is measured for each call of the field drone's update method. There are spikes in the framerate for initialization and geometry loading. The maximum frame time is 0.4 seconds.

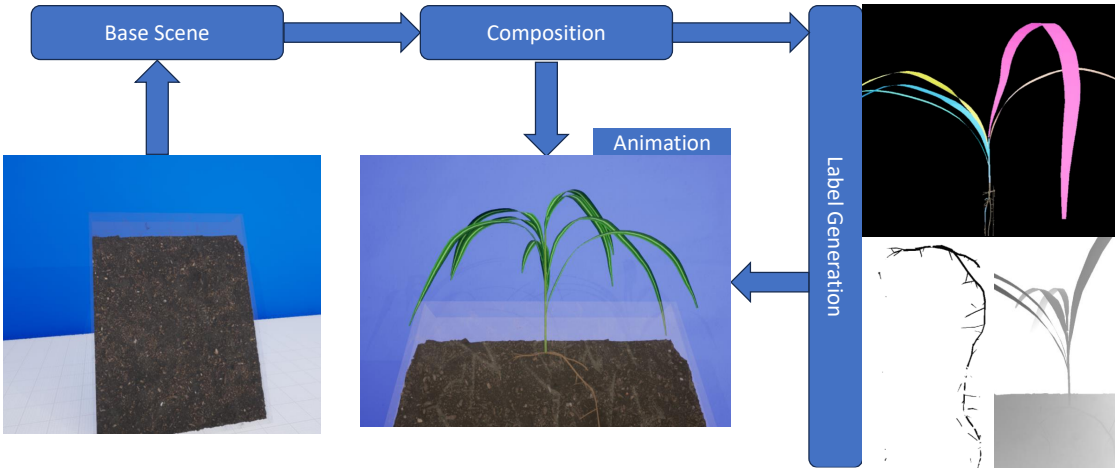

Figure 5: Overview of Experiment Setup. For this setup, we chose to model the soil within UE by making use of the modelling toolkit. From the Material Properties we can change this dynamically similarly to how the plant is positioned (middle). Synavis allows us to change properties of the plant as well as what measurements we make, whether pixel-based (right) or quantitative, as presented here by examples of leaf instances, root segmentation, and depth mask.

Task	**Simulation of one Plant**	**Parallel Total**	**Num Cores**
1M plants / 7 days	0.015 seconds $\mu = 0.005$	121.40 seconds	192
1M plants / 14 days	0.067 seconds $\mu = 2.6E\text{-}4$	353.34 seconds	192
1M plants / 28 days	0.1 seconds $\mu = 3E\text{-}4$	535.44 seconds	192

Table 1: Timing of CPlantBox generation of plant structures. For this experiment, the plant parameters from Section 3.3 were used. The visualizations were taken into account for the stem and leaf part, as for the field scene, the roots are not visible. Note that these measurement were done using a descriptive callibration of CPlantBox for experimental measurements and thus only capture the structure and morphology of a maize plant as observed in the laboratory. Coupled simulations *will* take longer. CPlantBox simulations were run on 4 CPU nodes with 192 MPI instances total (48 cores each). Timings were wallclock timings before simulation run until after geometrization was completed.

Task	**Average Time per Call** $[s]$	**Total seconds**
Time until Rendered Image is Encoded to Video	0.0366	19.3
Rendering of the Synthetic Data onto the Image	0.006	54.3
Raytracing Steps for the Image Composition	0.005	70
Cumulated two-sided rendering of the image	0.018	163
Full run of the application loop	0.052	360

Table 2: Timing statistics of UE runtime on a JUWELS Booster node, Total Time of Experiment was 6 minutes, and measurements were taken per-frame. This table shows the most significant timings that were relevant to the game performance. Lumen Screen Probes are used for dynamic shadowing and global illumination, whereas the scene captures are separate full render passes. Timings were taken using Unreal Insights, an application shipped with Unreal Engine.

can be obtained from rhizotron experiments. Since it is of huge interest to have data of shoot and root from the same plant, a precise segmentation of all organs is required. Consequently, labelled image data from rhizotron experiments are both needed and scarce. The scarcity of the data makes training a segmentation network based on this kind of data very hard. As such, it is necessary to implement methods which make optimal use of rhizotron images. To approach these challenges, we design a virtual rhizotron experiment using the FSPM CPlantBox, coupled via Synavis to the UE. In the course of this section we will highlight how the setup works, how data is produced, how to fetch the data and furthermore, present a variety of test scenarios to use for training. The experiment setup for this example is publicly available and we encourage the tuning and customization of all content we produce.

3.4 Setting up CPlantBox for the Laboratory Scenario

The rhizotron used as baseline has an interior measurement of $(20 \times 60 \times 2)\,\text{cm}$ in $w \times h \times d$ as well as a distance of 138 cm to the camera. In the UE scene, for the sake of generating synthetic data, some of these measurements might be varied. While the orientation of the camera to the rhizotron matters, as it of course must be directed at the object, the measurements are not impactful and, in fact, should be changed for the training. The base setup can be seen in

Figure 5 Left.

However, root growth must happen within the constraints of the rhizotron, which can be done using signed distance functions to limit root growth (Zhou et al., 2020). This restriction helps with the visualization in this instance, but does not detract from the stochasticity of the simulation. The seed position of the plant is put on top of the rhizotron box, and the simulation time is 25 days in steps of 0.5 days.

3.5 Integrating the Geometry into UE

Simulation of the CPlantBox plants is done sequentially: individual plant geometries are inserted into UE uniquely and regularly to exchange the data. This is shown in Figure 5 in the middle, where a plant is rendered within the otherwise empty rhizotron. For this implementation, we automatically add certain textures to the geometry, by using a container class in UE that can separately receive a stem, leaf, and root geometry group. This separation of the organ parts allows the use of different materials for the individual organs. This is not strictly necessary, but it is much easier to separate the individual components both on the Python endpoint as well as in UE.

Geometry generation is handled in the same script as the coupling, as the generation is done in regular intervals. This is a hyperparameter for training: more regular exchanges of the geometry might help in some circumstances. For feature extraction tasks where the orientation of the shoot to the root does not matter, we can further randomize shoot orientation for more images.

We note that the segmentation is done by using an alternate depth rendering pass within UE, which makes sure that we have a separation of the roots from the rest. However, it is also important to only show the root system that is actually visible, which is being taken into account during soil rendering. In Figure 5, we highlight now this part of the rendering pipeline works by contrasting the root rendering. There is a possibility of estimating the whole biomass or leaf area from a drone perspective, but there is a strong distinction between the tasks *estimate total leaf area* and *segment and calculate visible leaf area*, which depends on the label sets and the measurements of the accompanying real-world data, if applicable.

3.6 Randomizing the Laboratory Scene

Scene randomization in laboratory settings is challenging, but not impossible. A lot of robustness also stems from changing camera properties. Lens properties, such as shutter speed, ISO factor, and aperture, can be configured in UE through Synavis to allow for some more physically-based randomization. UE also offers features such as film grain to allow for more augmentation. This is a filter that would introduce noise that does not correspond to what the information rendering is seeing, similar to depth of field. Distance measures through UE can be altered but will be exact by default. Users might choose to artificially decrease data quality through selection or filtering algorithms, as LiDAR data can be imperfect as well (Zhao et al., 2022). At this point, image-based augmentation usually done on image datasets should still be employed, even on streams. We calibrated leaf surface colors in the direct comparison using average color measurements.

3.7 Training Concepts

In this framework, the active camera in the scene continuously emits an encoded video stream, much like a webcam would. The other data transport method is the data channel - a direct messaging pathway with low latency and minimal overhead. The data channel is also the minimum required communication channel to setup a WebRTC connection.

Figure 5 on the top-right shows an example of leaf instance segmentation that was handled via the data camera. Note that this is not necessary if the training algorithm also has access to the plant data, from CPlantBox, along with its orientation and position on the map. There, leaf instances on regions of the image can also be inferred directly from the plant data. However, ultimately it depends on how the real-world data is labelled, and the synthetic data should be labeled the same. In this case, pixel labels can be acquired much faster than any other type.

3.8 Upscaling the Simulation to Field Level

Large scale examples of the Synavis framework are possible, but require a lot of resources. In our case, the best way to utilize the scalable aspects of the framework is to make use of the modular supercomputing architecture. We present a large-scale example of synthetic field data generation using Synavis and the FSPM CPlantBox. The setup is shown in Figure 6. For the setup of the field, usually a boundary definition is sufficient. The addition of geometries for the soil can be done premeditated in the editor or at runtime through Synavis. When generating plants using CPlantBox dynamically, we refer to the measurements of the associated processes in Section 3.2.

Randomization between compute nodes is done using JSON descriptions that are generated based on local rank, as exemplified in Figure 6. This can be included in the runscript, or explicitly in the python environment. Generally, it is possible to have both varying[7] as well as static-property scenes for comparison between compute nodes of model robustness. This is especially shown in Figure 6.

The setup is aimed at producing high stochastic variability while allowing for cross-comparison between the individual data extractions. As such, we utilize a large field that is centrally populated by the FSPM with plant geometries. UE uses primarily GPU resources, but will also have a lot of CPU work to do, as the virtual world has to be simulated and UE needs to respond to inputs through the Synavis framework. As such, UE can run either on the booster module or the visualization module of a supercomputer. This is only true for booster modules which allow rendering, such as JUWELS. Training frameworks and extraction tasks are best run on the booster module and connected to UE via a common network interface or through tunneling. The CPlantBox generation is best run on pure CPU nodes, as the generation of plant models can be run in parallel or, in the case of coupled simulations, would use too many resources to be used together with UE on the same node. In our approach, we do not compute the CPlantBox geometries locally on the UE node. One bottleneck, however, is that the spawning of geometries occurs in batches for each frame, resulting in a warm-up time until the scene is loaded and present.

4 Discussion

We developed a novel framework for a direct and scalable coupling of an FSPM visualization with UE. This framework, Synavis, and its plugins SynavisUE, are primarily designed to allow for a loose coupling of data generation with synthetic data training on HPC systems. The framework is designed to allow the transport of FSPMs to UE, while also allowing easy access to objects in the scene as well as their properties. Synavis automatically conveys information from UE and is extremely suitable for people who develop their own applications in UE, especially building on top of existing projects.

[7] SynavisUE Lighting Management or Dynamic Volumentric Sky Lighting Management

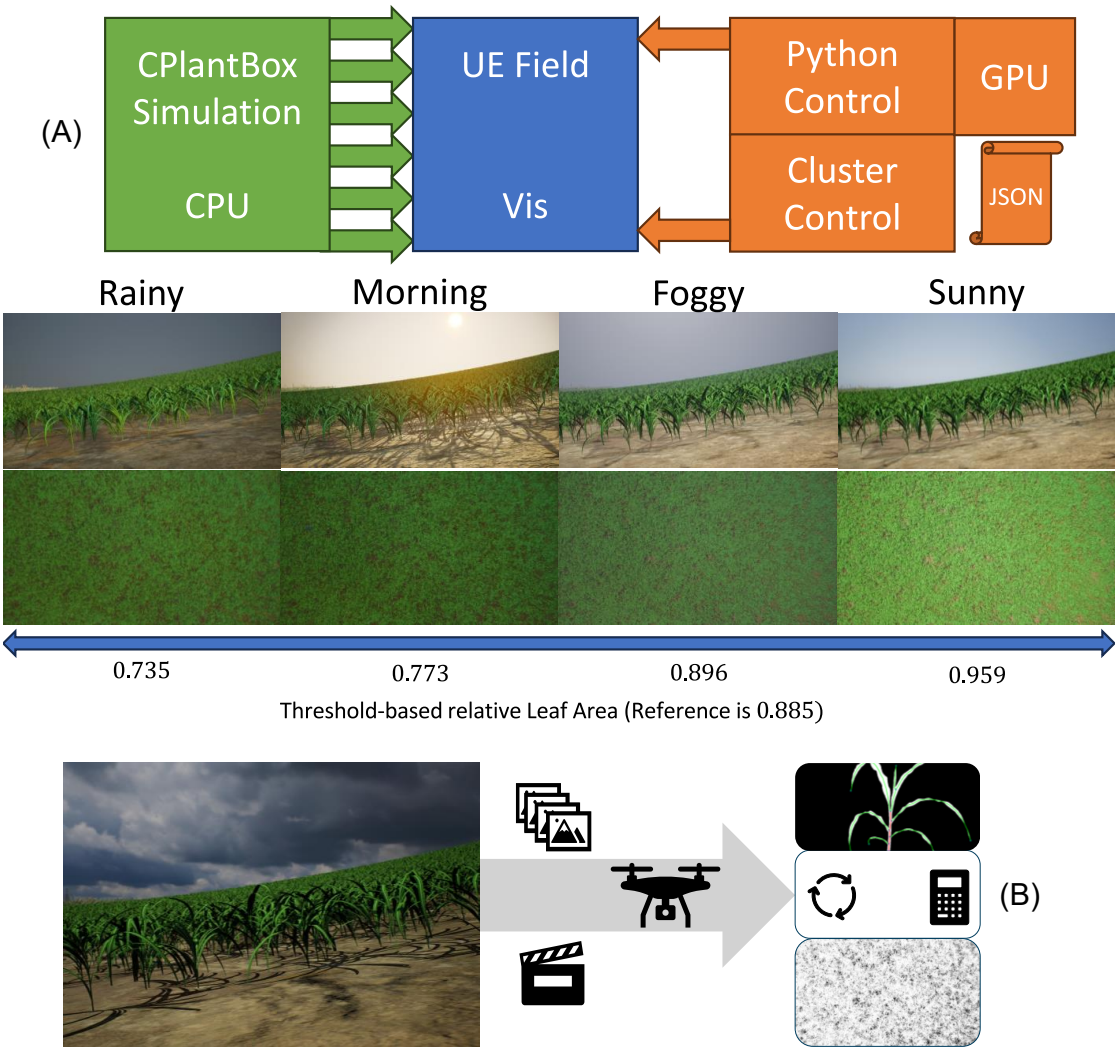

Figure 6: A. Overview of the field data generation pipeline. We also show scene property change can lead to significant changes in acquired data. The setup for a field visualization with a cluster is used to assess robustness of models against influences of the image. In this example, the field is not randomized, to allow for a comparable result in all compute nodes. Influences on the image can be numerous, and here we primarily showcase the change of weather that impacts image rendering. For more direct management, the SynavisUE plugin also contains methods of quickly altering lighting effects. Since all assets have UE accessible properties, the JSON description for the rank automatically manages the condition. This can be done preemptively using a JSON file, or through Synavis dynamically. The prime advantage of using the pipeline is versatility, as measurements as well as data labels can be generated very easily. The field scene is especially impactful for a number of aspects, such as conditional instance segmentation by using prior knowledge on which parameter set was used for the plant generation. Rule for calculating the relative leaf area is extracting by channel and then by threshold.

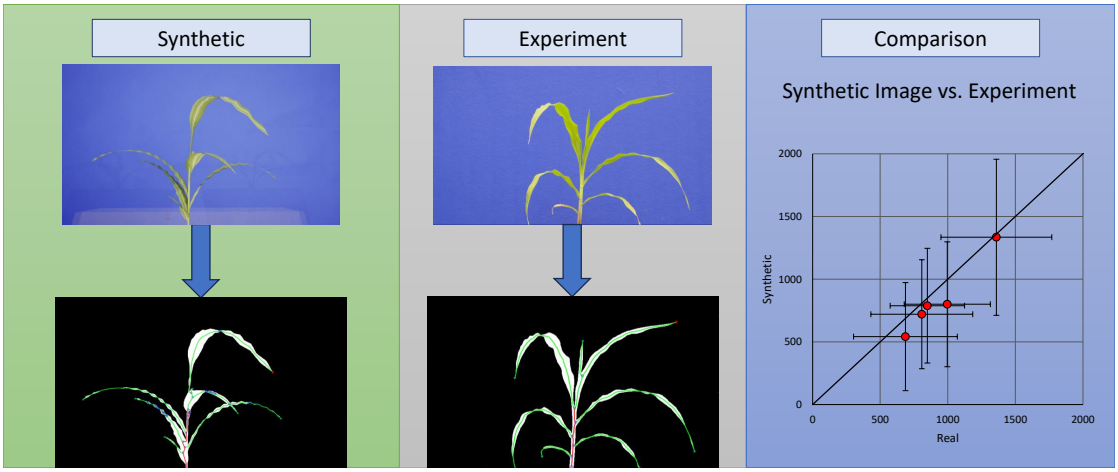

Figure 7: Comparison of parameter extraction pipeline between synthetic and real-world data. Real-world data, a total of 23 plant images at this growth stage and angle, was acquired in controlled rhizotron experiments. Bottom: Analyzed skeletons of shoot organs, starting with the pseudo-stem. Right: Comparison of blade lengths in mm, compared across samples sorted by longest first. The error bars indicate the standard deviation on each axis.

4.1 Analysis of the Synthetic Data

Synthetic data must, most of all, convey the information and retain the measurable properties that the experimental data also exhibits. To analyse the performance of SynavisUE in the example use-case, the rhizotron experiment, we ran the data through the extraction pipeline that is also used for the model calibration on rhizotron experiments (Bauer et al., 2023). Using the skeletonization and subsequent topological analysis, we extracted leaf blade length measures from UE as well as the experiment, as seen in Figure 7.

We want to highlight that in general, it is more beneficial to create diversity rather than exact replicas. Nonetheless, the synthetic data should be targeted at the use-case. Scalability can be targeted while training problems that are closely related to the target measurements. We see in Figure 7 that the synthetic measurements are lower in general, with a higher standard deviation. Lower measurements might be caused by the flatness of the leafs. The comparison of the synthetically generated data and the experimental data in Figure 7 right shows that relevant plant measures are replicated and are not distorted by the synthetic data generation pipeline.

4.2 Comparison to other Approaches

For comparison, we highlight the use of UnrealCV (Qiu et al., 2017) in science. An example of this is the data set generation for UnrealStereo by Zhang et al. (2018). Conceptually, UnrealCV is not dissimilar to SynavisUE, with the distinction that SynavisUE is based on PixelStreaming and loose coupling rather than a direct interaction. Furthermore, we put more emphasis on real-time data generation than writing data sets. The loose coupling on our end also enables us to separate technologies and make use of a system where interaction of the DL models with the scene mimic a robotics environment with all its latencies, as the world continues to simulate and does not wait for input. Moreover, we expand on the concept of synthetic data generation and present a whole-plant rendering approach together with data generation in UE as well as a coupling to training algorithms.

We especially highlight the embedding of Synavis into a modular supercomputing system (Suarez et al., 2019). Both JUWELS (Kesselheim et al., 2021; Krause, 2019; Alvarez, 2021) and JURECA supercomputers (Krause and Thörnig, 2018; Thörnig, 2021) are modular supercomputers. Newer systems, such as LUMI, further developed this concept to focus their accelerator systems on general compute-based programming paradigms (Markomanolis et al., 2022). For these systems, it is more optimal to separate the visualization and training component onto different modules, which for modern tensor core GPUs is even necessary as they do not provide rendering infrastructure. We designed Synavis with this concept in mind.

Moreover, we would like to make the important distinction between the generation of a virtual environment and algorithms such as by Gao et al. (2023). These approaches make the images more realistic and the training more robust, but ultimately flatten the virtually generated environment to images by augmenting them unpredictably. These approaches have also been reported to be unable to retain the geometric qualities of the virtual scene as described as CycleGAN failure cases by Hartley et al. (2021) and Zhu et al. (2017). SynavisUE is a framework that lets training models interact with the virtual scene by steering and manipulation. Furthermore, our scalability setup utilizes the fact that manipulating settings in the scene creates augmented images whose main advantage is the fact that one can interpolate settings between a success and a failure case while always retaining a coherent and well-annotated image.

4.3 Synavis: Possibilities and Limitations

Synthetic data is not the final training step, if fine-tuning has to be performed. We believe that the main strength in synthetic data training is its scalability rather than exactness.

Transfer learning approaches such as ImageNet (Deng et al., 2009) based encoders have exhibited good performance in many use-cases, as analyzed by Huh et al. (2016) and Morid et al. (2021). The reason behind this is that a lot of tasks in computer vision are re-usable and generalize well. This concerns individual feature maps as well as activation weights that are transferable. One example of this is presented by Chen et al. (2020), who analyze transfer learning performance in image-based plant disease detection.

Synavis and SynavisUE are most comfortably usable between a pre-trained network and the final tuning. Synthetic data provides a lot of variety when given proper stochasticity commands, and its use in training DL models is potentially very impactful. Synavis is created with strong data augmentation in mind, as scene properties can be changed at runtime, changing the visual properties of the data.

WebRTC is a video real-time communication experience standard. For our user-less framework this means that the video information might not be in lossless format. On HPC machines, we usually circumvent this by scaling up the image size and downsample on the DL side. Formats like H264 (ISO/IEC 14496-10:2022, 2022) are very light in network usage, as shown by Van der Auwera et al. (2008), but might impact the customizability of the framework because the numbers of encoding sessions are limited, or by the aspect ratio of supported image formats.

4.4 Outlook

In this article, we showed the promise and practical use of our coupling framework, together with a visualization of the FSPM CPlantBox. Visualization of more morphological features is needed in the future to push the limits of the visualization. We also aim for calibration measurements such as light metering in comparison to UE lights, as currently we can make only relative but not absolute statements about how plants would behave in the simulated scenes.

WebRTC is inherently interactive, which means that data is sent both ways (Jennings et al., 2021). We will build on this concept further by allowing the steering of the data generation during training.

There are possibilities in advancing algorithm robustness that are both depended on scene variety as well as training strategies. We will explore both those options in the near future.

Incorporation of different data sources is also possible within Synavis, but more fine-tuning should be done to arrive at a successful scene visualization.

Model and Data Availability

The code is open source and available under the Synavis and SynavisUE repositories with an example avaliable under SynavisUEexample. The CPlantBox official code can be found at on the institute's GitHub page. The branch associated with this article has been forked to this page.

We have prepared introduction videos to the subjects. An overview of the framework and pipelines that are possible can be found here:
Helmrich, Dirk Norbert (2023). Supplemental Video for our Manuscript ISPLANTS-2023-026. figshare. Media. https://doi.org/10.6084/m9.figshare.24179136.v2
A step-by-step guide can be found here:
Helmrich, Dirk Norbert (2023). Step by Step guide to setup Synavis with CPlantBox and Unreal Engine. figshare. Media. https://doi.org/10.6084/m9.figshare.24182592.v1

Contribution

DH has implemented the FSPM visualization, the coupling framework, the Unreal Engine implementation, and authored this paper primarily. FB has contributed to the scientific workflow, the calibration of the FSPM model to plant data, the implementation of the analysis pipeline used in this paper, and contributed input about field measurements. He has contributed to the text of this paper. AS supervises the CPlantBox FSPM development, has contributed to the text of this paper, and has given input on scientific workflows. MG has implemented large parts of features and analysis pipelines used within the FSPM used in this paper. She has implemented calibration workflows for the FSPM and contributed scientific counsel to this paper. She has contributed to the text of this paper. JHG has contributed to the text of this paper, implemented functions and methodology used within the HPC environment, is responsible for the implementation of new visualization packages and technologies on JURECA and JUWELS supercomputers, and has contributed scientific counsel to this paper. EH has contributed to the text of this paper, given scientific counsel, access to the LUMI supercomputer for testing, and is supervisor to this project for the University of Iceland. HS has contributed to the text of this paper, given scientific counsel, and is supervisor to this project for the University of Iceland. MR has contributed to the text of this paper, given scientific counsel, provided access to compute systems and information about methodology. He is primary supervisor of this project.

Funding

The authors would like to acknowledge funding provided by the German government to the Gauss Centre for Supercomputing via the InHPC-DE project (01—H17001).

This work has partly been funded by the EUROCC2 project funded by the European High-Performance Computing Joint Undertaking (JU) and EU/EEA states under grant agreement No

101101903.

This work has partly been funded by the German Research Foundation under Germany's Excellence Strategy, EXC-2070 - 390732324 - PhenoRob and by the German Federal Ministry of Education and Research (BMBF) in the framework of the funding initiative "Plant roots and soil ecosystems, significance of the rhizosphere for the bio-economy" (Rhizo4Bio), subproject CROP (ref. FKZ 031B0909A).

References

Ahmadi, M., Gross, W. J., and Kadoury, S., (2016). A real-time remote video streaming platform for ultrasound imaging in *Proceedings of the 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* DOI: 10.1109/EMBC.2016.7591698

Alvarez, D. (2021). JUWELS cluster and booster: Exascale pathfinder with modular supercomputing architecture at Jülich Supercomputing Centre. *Journal of Large-Scale Research Facilities JLSRF*, 7. DOI: 10.17815/jlsrf-7-183

Bailey, B. N. (2019). Helios: A scalable 3D plant and environmental biophysical modeling framework. *Frontiers in Plant Science*, 10. DOI: 10.3389/fpls.2019.01185

Bauer, F. M., Lobet, G., Helmrich, D. N., Galinski, A., Kahlilova, Z., Zaner, L., Kuczkowska, M., Yu, P., Dörmann, P., Schaaf, G., and Schnepf, A. (2023). In silico investigation on phosphorus efficiency of zea mays: An experimental whole plant model parametrization approach. *FSPM 2023* DOI: 10.34734/FZJ-2023-04031

Behroozpour, B., Sandborn, P. A. M., Wu, M. C., and Boser, B. E. (2017). LiDAR system architectures and circuits. *IEEE Communications Magazine*, 55(10):135–142. DOI: 10.1109/MCOM.2017.1700030

Benoit, L., Rousseau, D., Belin, É., Demilly, D., and Chapeau-Blondeau, F. (2014). Simulation of image acquisition in machine vision dedicated to seedling elongation to validate image processing root segmentation algorithms. *Computers and Electronics in Agriculture*, 104:84–92. DOI: 10.1016/j.compag.2014.04.001

Bondi, E., Dey, D., Kapoor, A., Piavis, J., Shah, S., Fang, F., Dilkina, B., Hannaford, R., Iyer, A., Joppa, L., and Tambe, M. (2018). AirSim-W: A Simulation Environment for Wildlife Conservation with UAVs. COMPASS '18, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3209811.3209880

Bouvry, A. and Lebeau, F. (29 March 2023). Digital twin of a smart plant factory for plant phenotyping: Data assimilation between measured and simulated 3D point cloud data in the CPlantBox FSPM. *FSPM 2023*, Berlin, Germany

Chen, J., Chen, J., Zhang, D., Sun, Y., and Nanehkaran, Y. (2020). Using deep transfer learning for image-based plant disease identification. *Computers and Electronics in Agriculture*, 173:105393. DOI: 10.1016/j.compag.2020.105393

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNET: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2009.5206848

Gao, Y., Li, Y., Jiang, R., Zhan, X., Lu, H., Guo, W., Yang, W., Ding, Y., and Liu, S. (2023). Enhancing green fraction estimation in rice and wheat crops: A self-supervised deep learning semantic segmentation approach. *Plant Phenomics*, 5:0064. DOI: 10.34133/plantphenomics.0064.

Giraud, M., Gall, S. L., Harings, M., Javaux, M., Leitner, D., Meunier, F., Rothfuss, Y., van Dusschoten, D., Vanderborght, J., Vereecken, H., Lobet, G., and Schnepf, A. (2023). Development and calibration of the FSPM CPlantBox to represent the interactions between water and carbon fluxes in the soil-plant-atmosphere continuum. DOI: 10.1101/2023.04.18.537289.

Hartley, Z. K. J. and French, A. P. (2021). Domain adaptation of synthetic images for wheat head detection. *Plants*, 10(12). DOI: 10.3390/plants10122633

Hartley, Z. K. J., Jackson, A. S., Pound, M., and French, A. P. (2021). GANana: Unsupervised domain adaptation for volumetric regression of fruit. *Plant Phenomics*, 2021. DOI: 10.34133/2021/9874597

Hughes, J. F., Van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S. K., and Akeley, K. (2014). Introduction to fixed-function 3d graphics and hierarchical modeling. *Computer Graphics, Principles and Practice*, pages 117–148. ISBN: 978-0-201-84840-3

Huh, M., Agrawal, P., and Efros, A. A. (2016). What makes ImageNet good for transfer learning? In *2016 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.48550/arXiv.1608.08614

ISO/IEC 14496-10:2022 (2022). H.264 : Advanced video coding for generic audiovisual services. Standard, International Telecommunication Union, Geneva, CH. ID: T-REC-H.264-202108-I

ITU-T H.265 (2023). H.265 : High efficiency video coding. Standard, International Telecommunication Union, Geneva, CH. ID: T-REC-H.265-202309-P

Jansen, W., Huebel, N., and Steckel, J. (2022). Physical LiDAR simulation in real-time engine. In *2022 IEEE Sensors*. DOI: 10.1109/SENSORS52175.2022.9967197

Jennings, C., Castelli, F., Boström, H., and Bruaroey, J.-I. (2021). WebRTC 1.0: Real-Time Communication Between Browsers. *Recommendation of the World Wide Web Consortium*. ID: REC-webrtc-20230306

Jitsev, J. (2021). Impact of large-scale pre-training on intra- and inter-domain transfer learning in full and few-shot regimes. In *Workshop Machine Learning on HPC Systems, International Supercomputing Conference*, Frankfurt, Germany, Jul 2021.

Kamilaris, A. and Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70–90. DOI: 10.1016/j.compag.2018.02.016

Karis, B., Stubbe, R., and Wihlidal, G. (2021). A deep dive into Nanite Virtualized Geometry. In *SIGGRAPH* 2021.

Kesselheim, S., Herten, A., Krajsek, K., Ebert, J., Jitsev, J., Cherti, M., Langguth, M., Gong, B., Stadtler, S., Mozaffari, A., Cavallaro, G., Sedona, R., Schug, A., Strube, A., Kamath, R., Schultz, M. G., Riedel, M., and Lippert, T. (2021). JUWELS booster – a supercomputer for large-scale AI research. In *High Performance Computing*, Cham. Springer International Publishing. DOI: 10.48550/arXiv.2108.11976

Kim, D., Kang, W. H., Hwang, I., Kim, J., Kim, J. H., Park, K. S., and Son, J. E. (2020). Use of structurally-accurate 3D plant models for estimating light interception and photosynthesis of sweet pepper (capsicum annuum) plants. *Computers and Electronics in Agriculture*, 177:105689. DOI: 10.1016/j.compag.2020.105689

Krause, D. (2019). JUWELS: Modular tier-0/1 supercomputer at the Jülich Supercomputing Centre. *Journal of Large-Scale Research Facilities JLSRF*, 5. DOI: 10.17815/jlsrf-5-171

Krause, D. and Thörnig, P. (2018). JURECA: modular supercomputer at Jülich Supercomputing Centre. *Journal of large-scale research facilities JLSRF*. DOI: 10.17815/jlsrf-4-121-1

Kuznichov, D., Zvirin, A., Honen, Y., and Kimmel, R. (2019). Data augmentation for leaf segmentation and counting tasks in rosette plants. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2580–2589.

Lobet, G., Draye, X., and Périlleux, C. (2013). An online database for plant image analysis software tools. *Plant Methods*, 9(1):38. DOI: 10.1186/1746-4811-9-38

Lobet, G., Koevoets, I. T., Noll, M., Meyer, P. E., Tocquin, P., Pagès, L., and Périlleux, C. (2017). Using a structural root system model to evaluate and improve the accuracy of root image analysis pipelines. *Frontiers in Plant Science*, 8. DOI: 10.3389/fpls.2017.00447

Markomanolis, G. S., Alpay, A., Young, J., Klemm, M., Malaya, N., Esposito, A., Heikonen, J., Bastrakov, S., Debus, A., Kluge, T., Steiniger, K., Stephan, J., Widera, R., and Bussmann, M. (2022). Evaluating GPU programming models for the LUMI supercomputer. In *Supercomputing Frontiers*. Springer International Publishing. DOI: 10.1007/978-3-031-10419-0_6

Masson, A. L., Caraglio, Y., Nicolini, E., Borianne, P., and Barczi, J.-F. (2021). Modelling the functional dependency between root and shoot compartments to predict the impact of the environment on the architecture of the whole plant: methodology for model fitting on simulated data using Deep Learning techniques. *in silico Plants*, 4(1):diab036. DOI: 10.1093/insilicoplants/diab036

McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. (2017). SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation? In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/ICCV.2017.292

Morandage, S., Laloy, E., Schnepf, A., Vereecken, H., and Vanderborght, J. (2021). Bayesian inference of root architectural model parameters from synthetic field data. *Plant and Soil*, 467(1):67–89. DOI: 10.1007/s11104-021-05026-4

Morid, M. A., Borjali, A., and Del Fiol, G. (2021). A scoping review of transfer learning research on medical image analysis using ImageNet. *Computers in Biology and Medicine*, 128:104115. DOI: j.compbiomed.2020.104115

Nagel, K. A., Putz, A., Gilmer, F., Heinz, K., Fischbach, A., Pfeifer, J., Faget, M., Blossfeld, S., Ernst, M., Dimaki, C., Kastenholz, B., Kleinert, A.-K., Galinski, A., Scharr, H., Fiorani, F., and Schurr, U. (2012). GROWSCREEN-Rhizo is a novel phenotyping robot enabling simultaneous measurements of root and shoot growth for plants grown in soil-filled rhizotrons. *Functional Plant Biology*, (39):891–904. DOI: 10.1071/fp12023

Nimmi, S., Saranya, V., Theerthadas, and Gandhiraj, R. (2014). Real-time video streaming using GStreamer in GNU radio platform. In *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*. DOI: 10.1109/ICGCCEE.2014.6922233

Pentakalos, O. I. (2002). An introduction to the infiniband architecture. In *High Performance Mass Storage and Parallel I/O: Technologies and Applications*. DOI: 10.1109/9780470544839.ch42

Perez, R. P. A., Vezy, R., Brancheriau, L., Boudon, F., Grand, F., Ramel, M., Artanto Raharjo, D., Caliman, J.-P., and Dauzat, J. (2022). When architectural plasticity fails to counter the light competition imposed by planting design: an in silico approach using a functional–structural model of oil palm. *in silico Plants*, 4(1). DOI: 10.1093/insilicoplants/diac009

Pollok, T., Junglas, L., Ruf, B., and Schumann, A. (2019). UnrealGT: Using unreal engine to generate ground truth datasets. *Advances in Visual Computing*. DOI: 10.1007/978-3-030-33720-9_52

Pound, M. P., Atkinson, J. A., Townsend, A. J., Wilson, M. H., Griffiths, M., Jackson, A. S., Bulat, A., Tzimiropoulos, G., Wells, D. M., Murchie, E. H., Pridmore, T. P., and French, A. P. (2017). Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *GigaScience*, 6(10). DOI: 10.1007/978-3-030-33720-9_52

Qiu, W., Zhong, F., Zhang, Y., Qiao, S., Xiao, Z., Kim, T. S., and Wang, Y. (2017). UnrealCV: Virtual worlds for computer vision. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1221–1224. DOI: 10.1145/3123266.3129396

Reddy, T., Johnston, A., Matthews, P., and Rosenberg, J. (2020). Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). *Proposed Standard of the Internet Engineering Task Force (IETF)*. DOI: 10.17487/RFC8656

Sanders, A. (2016). *An introduction to Unreal engine 4*. CRC Press. ISBN 978-1138427440

Scharr, H., Minervini, M., French, A. P., Klukas, C., Kramer, D. M., Liu, X., Luengo, I., Pape, J.-M., Polder, G., Vukadinovic, D., et al. (2016). Leaf segmentation in plant phenotyping: a collation study. *Machine vision and applications*, 27(4):585–606. DOI: 10.1007/s00138-015-0737-3

Scharr, H. and Tsaftaris, S. A. (2022). Meeting computer vision and machine learning challenges in crop phenotyping. *Advances in plant phenotyping for more sustainable crop production*. Burleigh Dodds Science Publishing. DOI: 10.19103/AS.2022.0102.11

Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. (2003). RTP: A Transport Protocol for Real-Time Applications. *Standard of the Internet Engineering Task Force*. DOI: 10.17487/RFC3550

Soualiou, S., Wang, Z., Sun, W., de Reffye, P., Collins, B., Louarn, G., and Song, Y. (2021). Functional–structural plant models mission in advancing crop science: Opportunities and prospects. *Frontiers in Plant Science*, 12. DOI: 10.3389/fpls.2021.747142

Suarez, E., Eicker, N., and Lippert, T. (2019). Modular supercomputing architecture: from idea to production. In *Contemporary high performance computing*, pages 223–255. CRC Press. ID: FZJ-2019-03055

Thörnig, P. (2021). JURECA: Data Centric and Booster Modules implementing the Modular Supercomputing Architecture at Jülich Supercomputing Centre. *Journal of large-scale research facilities JLSRF*. DOI: 10.17815/jlsrf-7-182

Tsaftaris, S. A., Minervini, M., and Scharr, H. (2016). Machine learning for plant phenotyping needs image processing. *Trends in plant science*, 21(12):989–991. DOI: 10.1016/j.tplants.2016.10.002

Ubbens, J., Cieslak, M., Prusinkiewicz, P., and Stavness, I. (2018). The use of plant models in deep learning: an application to leaf counting in rosette plants *Plant methods*, 14 (6). DOI: 10.1186/s13007-018-0273-z

Van der Auwera, G., David, P. T., and Reisslein, M. (2008). Traffic characteristics of H264/AVC variable bit rate video. *IEEE Communications Magazine*, 46(11):164–174. DOI: 10.1109/M-COM.2008.4689260

Wang, L., Wang, W., Dorsey, J., Yang, X., Guo, B., and Shum, H.-Y. (2006). Real-time rendering of plant leaves. In *ACM SIGGRAPH 2006 Courses*. DOI: 10.1145/1185657.1185725

Ward, D., Moghadam, P., and Hudson, N. (2018). Deep leaf segmentation using synthetic data. In *Proceedings of the British Machine Vision Conference (BMVC) Workshop on Computer Vision Problems in Plant Phenotyping (CVPPP)*. DOI: 10.48550/arXiv.1807.10931

Yang, W., Feng, H., Zhang, X., Zhang, J., Doonan, J. H., Batchelor, W. D., Xiong, L., and Yan, J. (2020). Crop phenomics and high-throughput phenotyping: Past decades, current challenges, and future perspectives. *Molecular Plant*, 13(2):187–214. DOI: 10.1016/j.molp.2020.01.008

Yun, T., Cao, L., An, F., Chen, B., Xue, L., Li, W., Pincebourde, S., Smith, M. J., and Eichhorn, M. P. (2019). Simulation of multi-platform lidar for assessing total leaf area in tree crowns. *Agricultural and Forest Meteorology*, 276-277:107610. DOI: 10.1016/j.agrformet.2019.06.009

Zhang, T., Xie, L., Wei, L., Zhuang, Z., Zhang, Y., Li, B., and Tian, Q. (2020). UnrealPerson: An Adaptive Pipeline towards Costless Person Re-identification. In *2020 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR46437.2021.01134

Zhang, Y., Qiu, W., Chen, Q., Hu, X., and Yuille, A. (2018). UnrealStereo: Controlling hazardous factors to analyze stereo vision. In *International Conference on 3D Vision (3DV)*. DOI: 10.48550/arXiv.1612.04647

Zhao, X., Su, Y., Hu, T., Cao, M., Liu, X., Yang, Q., Guan, H., Liu, L., and Guo, Q. (2022). Analysis of UAV lidar information loss and its influence on the estimation accuracy of structural and functional traits in a meadow steppe. *Ecological Indicators*, 135:108515. DOI: 10.1016/j.ecolind.2021.108515

Zhou, X.-R., Schnepf, A., Vanderborght, J., Leitner, D., Lacointe, A., Vereecken, H., and Lobet, G. (2020). CPlantBox, a whole-plant modelling framework for the simulation of water- and carbon-related processes. *in silico Plants*, 2(1). DOI: 10.1093/insilicoplants/diaa001

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/ICCV.2017.244

Appendices

Variation of Virtual Scenes

Object	Property	Description
Sun	Position	2 DoF $\phi, \eta \in [0, \pi]$
Sun	Strength	[Lumen]
Sun	Ambient Light	Intensity [Lumen]
Field	Plant Position	Varying position 2 DoF, dependend on scene and placing, interior experiment scenes have less potential variability here
Field	Plant Density	Handled via position, this would be a good way of targeting different DL models that also react sensitively towards changes in complexity
Plant	Plant Age	Determined by parametrization, usually within $T \in [0, 30]$
Plant	Plant Leaf Bending	Partly visual feature - leaf bending is influenced by environmental conditions. When rendering a wet scene, leaf bending should be increased. For accuracy, it should be determined how large the impact of additional weight is on the leaf structure. If it can be assured that the leaf bending is not taken into account to solve the target problem, this can be estimated.
Room	Room Lights and Diffusion	For digitized experiments, lights should be calibrated to actual brightness values as stated by the bulb manufacturer.
Room	Room Wall Material	RGB Texture $X \times Y \times R \times B \times G$, Specularity, Roughness
Camera	Camera Lens Properties	A collection of scalars describing the cameras lense and depth-of-field effects in UE
Camera	Camera Position	Camera paths throughout the scene might further reveal critical insight into algorithm performance in specific and extraordinary circumstances.
Post Processing	Colour and Screen Effects	Film Grain, Movement effects, and more effects can be introduced using Post Process Materials. Not that encoding effects would be captures using the style of encoding and if finetuning towards those effects should be done, the reference should produce effects that might be affected by the encoding the strongest.

Table 3: Selection of scene variation possibilities to improve larger-scale data augmentation using Unreal Engine and Synavis. The list is not exhaustive, but should give a good impression that stochastic augmentation should be applied wherever possible.

Unreal Engine Cluster Configuration

```
m = syn.MediaReceiver()
...
tracefile = "trace_" + str(int(time.time())) + ".utrace"
m.SendJSON({"type":"console", "command":"trace.File "+ tracefile})
start_time = time.time()
runtime = 360
while time.time() < start_time + runtime :
  time.sleep(0.05)
  pos = np.random.rand(3) * 100 + np.array([0, 0, 0])
  size = np.random.rand() * 0.9 + 0.1
  v, i, n = cube_geometry(size, pos)
  v = v.flatten(); i = i.flatten(); n = n.flatten()
  m.SendGeometry(v, i, "cube", n, None, None, True)
m.SendJSON({"type":"console", "command":"trace.stop"})
m.SendJSON({"type":"console", "command":"quit"})
```

Listing 1: Setup for the scalability test performed with an increasing number of cube geometries within the scene. These were randomly spawned through Synavis

Property	**Value**	**Info**
Encoder	H264	GPU-Encoding
Resolution	3860×2160	4k UHD
Keyframe Interval	1	Triggers transmission of the full image every second
Scene Capture Virtual Shadow Map Size	4096	This allows the scene capture components, the InfoCam and SceneCam of the Synavis Drone, to have larger shadow maps than in a classical setting (default 512)
Render Offscreen	true	Needed for Cluster Headless Mode
Max GPU Count	4	Allows Unreal to make use of NVLink
PixelStreaming Degradation Preference	Maintain Quality	Prevents UE from subsampling the encoding

Table 4: Configuration for Unreal Engine runtime on JUWELS Booster

Media Receiver in Synavis

```
1  import Synavis as sv
2
3  media = sv.MediaReceiver()
4  media.Initialize()
5  media.SetConfig({"SignallingIP":"625.174.856.321", \
6      "SignallingPort": 8080})
7  media.SetTakeFirstStep(False)
8  media.StartSignalling()
9  media.SetMessageCallback( \
10     lambda message : message_buffer.append(message) \
11 )
12
13 while media.GetState() != sv.EConnectionState.CONNECTED :
14     time.sleep(0.1)
```

Listing 2: Simple Coupling with Media Receiver

Overview of Pipeline Components

Pipeline Component	Function	Futher Information
CPlantBox	• Generation of plant model • Provision of geometry	see Giraud et al., 2023, https://github.com/Plant-Root-Soil-Interactions-Modelling/CPlantBox
Unreal Engine	• Visualization of the plant in a virtual scene • Scene variation • Streaming of images	www.unrealengine.com
Synavis UE	• Reception and integration with PixelStreaming • Dynamic command handling • Integration with UE system	gh/dhelmrich/synavisue
Synavis	• Coupling between frameworks/software • Connection handling and signalling server • JSON format parsing and python binding	gh/dhelmrich/synavis
OpenCV/Pytorch	• Reception of images • Random image augmentation • Data analysis	pytorch.org, opencv.org

Table 5: Short overview of all pipeline component mentioned in this article, their function, and where to find out more and/or download them.

Geometrization

Generally, for a complete geometry as seen in Figure 8, we compute vertices $V \subset \mathbb{R}^3$, vertex-assigned normal vectors $N : V \ni x \mapsto n(x) \in \mathbb{R}^3$, triangles (with an ordered set V): $T \subset V \times V \times V$, as well as texture coordinates $A : V \ni x \mapsto a(x) \in \mathbb{R}^2$ and tangents. For a full introduction to geometry modelling, we refer to Hughes et al. (2014).

We extended CPlantBox with a geometrization pipeline based on its graph formalism and common assumptions. In this section, we describe how the plant data is used to create the plant morphology. Moreover, some characteristics of the plant morphology are not simulated by CPlantBox. Default geometrization schemes have therefore been setup while keeping the resulting geometry mainly sensitive to the CPlantBox outputs, as seen in Figure 1 and further in Figure 8.

Stem geometrization is based on generating a continuous space of ordered circles in a 3D-space that are connected by triangles. Their orientation has to be inferred from the graph structure, which means that their rotation Quaternion $q^{o,i,i-1} \in \mathbb{H}$ is defined by the angle between two consecutive segments $i-1, i$ on the organ o. The rotation operator is defined as $\mathrm{Rot}(q, x) := q \cdot (0, x) \cdot q^{-1}$ where $q \in \mathbb{H}$ and $x \in \mathbb{R}^3$. To prevent ambiguity of the local geometry, we generate the circle in *local space* $P_L^{o,i}$ and transfer it to *world space* by using the new direction and the axis of the previous segment. The transformation from local to world space can be described as

$$P_W^{(o,i)} = d^{(o,i)} \cdot \mathrm{Rot}\left(P_L^{(o,i)}, q^{(o,i,i-1)}\right) + C^{(o,i)} \tag{1}$$

where P_W, P_L are the positions in world and local space respectively, d is the local diameter, q is the world orientation quaternion and C is the position of the segment i on organ o. We compute a quaternion by the vector of two subsequent points i and $i+1$ as well as the last coordinate system vectors u (forward), v (right), w (up). When using splines, i.e., smooth curves in 3D space, to interpolate points, we can also compute rotation using the curve derivative. Within our rotation space $\mathbb{H}$, we can use spherical interpolation by applying $q_\alpha(q_0, q_1, \alpha) := q_1 \cdot (q_1^{-1} \cdot q_2)^\alpha$ with $\alpha \in [0, 1]$ being the interpolation factor, which we use for angular smoothing. Triangles are computed by connecting subsequent pairs of vertices with their predecessors. To update the coordinate system with a new forward vector $u^{(i)}$, we compute the new up vector $w^{(i)} = v^{(i-1)} \times u^{(i)}$ and the new right vector $v^{(i)} = w^{(i)} \times u^{(i)}$.

For leaf geometrization, we interpolate the graph structure of the underlying FSPM by using a series of splines. For the leaf structure specifically, we employ two distinct techniques to describe its surface: Linear and radial description. A linear leaf description is a geometrization that assigned each point p_i on the midline of the leaf a distance to the edge of the leaf blade in each direction. In contrast, radial geometrization is a kind of template that describes a shape by the distances d_i at specific angles φ_i from a center point of the leaf, such as the end of the petiole. For the leaf structure, we are rendering the leaf as two-sided flat surface. This means that all leaf variants are rendered into a series of triangles spanning the top and bottom part of the plant. Within UE, we can also simulate this by not culling the backface triangles of the geometry, and instead rendering the backfaces like frontfaces.

Additional structural features of the plant geometry can be inferred from texture as well: UE can render plants with additional world position offset, which changes where the vertices are rendered in the scene, regarding a displacement vector given in the material shader. This is especially useful when simulating wind, especially in conjunction with our texture warping technique as we can prescribe forces at specific positions relative to the leaf surface. Masked leafs might behave somewhat unpredictably with this addition, as it is not always clear where the first leaf point connects to the stem.

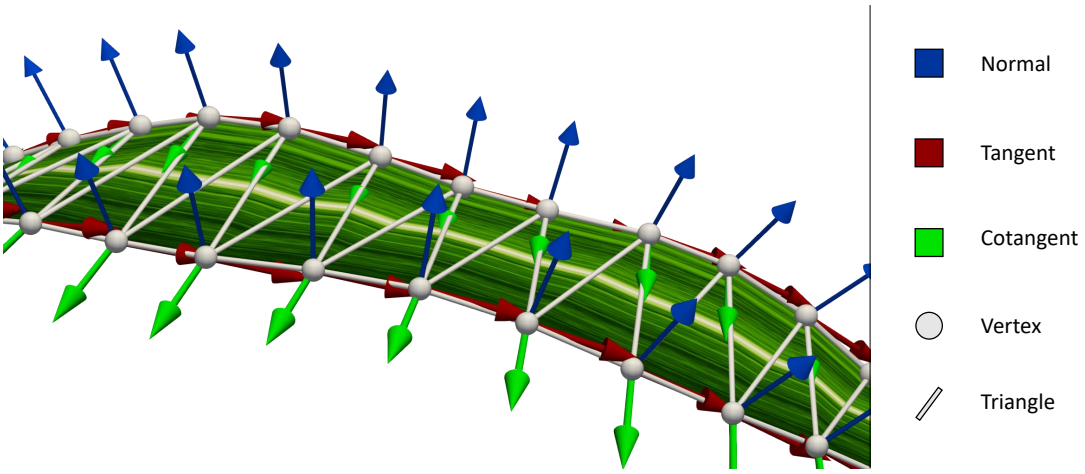

Figure 8: Leaf-level geometry, applied texture onto texture coordinate map, as well as local coordinate spaces. Texture image axes corresponds to tangent directions. A local coordinate system is essential for a robust geometrization of the leaf; otherwise instabilities cause significant visual artifacts. Local coordinates are not always guaranteed to be right-handed but uniform across the leaf surface. The blue arrow represents the local z-axis while the red arrow represents the local x-axis and green represents the local y-axis. By default, we heavily punish the z component of the cotangent vector while updating with each new CPlantBox point information.