

# Simulador de escenas forestales para la segmentación de bosques utilizando VANTs

Francisco Raverta Capua

Instituto de Ciencias de la Computación  
Universidad de Buenos Aires  
C1428EGA, CABA, Argentina  
Email: fraverta@icc.fcen.uba.ar

Juan Schandin

Departamento de Computación  
Universidad de Buenos Aires  
C1428EGA, CABA, Argentina  
Email: jschandin@dc.uba.ar

Pablo De Cristóforis

Departamento de Computación  
Universidad de Buenos Aires  
C1428EGA, CABA, Argentina  
Email: pdecris@dc.uba.ar

**Abstract**—El sensado remoto, mediante el uso de vehículos aéreos no tripulados (VANTs), ha crecido sostenidamente en los últimos años, junto con el uso del aprendizaje automático para el procesamiento de datos. La creación de *datasets* de ambientes naturales es costosa económica y temporalmente, ya que requiere de sensores y equipamiento de alta precisión, y de tareas de procesamiento tales como el etiquetado manual en imágenes o en nubes de puntos. Algunas áreas naturales, como los bosques, pueden resultar de difícil acceso o peligrosas para el ser humano. Para poder promover la investigación en tareas que requieran de *datasets* específicos de estos ambientes, en este trabajo desarrollamos un simulador que genera proceduralmente escenas forestales sintéticas realistas. A partir de este simulador, buscamos responder la pregunta de si es posible utilizar datos sintéticos para el entrenamiento de redes de aprendizaje profundo basadas en nubes de puntos para la segmentación de escenas forestales sin la necesidad de contar con grandes volúmenes de datos reales de estos ambientes. Tanto el simulador como el *dataset* generado se encuentran disponibles públicamente on-line como parte de este trabajo.

## I. INTRODUCCIÓN

Las técnicas de sensado remoto para el monitoreo ambiental utilizando VANTs han tomado gran relevancia, gracias a técnicas tales como el escaneo láser aéreo (ALS, del inglés *Aerial Laser Scanning*) y la fotogrametría aérea [1], [2], [3]. Sin embargo, crear *datasets* a partir de estos sensados resulta costoso, ya que se requiere de equipos complejos, incluyendo VANTs y sensores de alta precisión. Si el *dataset* consiste en nubes de puntos o imágenes segmentadas, resulta también temporalmente demandante, ya que requiere el etiquetado manual de los puntos o píxeles. Además, en el caso particular de las áreas forestales, pueden ser de difícil acceso o peligrosas para los seres humanos, lo que dificulta aún más las tareas de recolección de datos.

Este trabajo tiene como objetivo presentar un simulador de ambientes forestales de código abierto basado en Unity que permita generar datos sintéticos de escenas forestales de forma procedural que resulten fieles a ambientes forestales reales, con el fin de brindarlo como alternativa a la necesidad de adquisición de datos reales, al momento de entrenar o testear nuevas tecnologías. El simulador presentado cuenta con la capacidad de exportar las nubes de puntos de los objetos instanciados, y además simular una misión de vuelo para la toma de fotografías aéreas desde una vista superior.



Fig. 1: Ejemplo de una escena generada por el simulador. El terreno se genera usando ruido de Perlin, y cuenta con árboles y arbustos que siguen una distribución definida por el usuario.

El simulador es adaptable para agregar sensores, tales como LiDAR o diferentes tipos de cámaras. Un ejemplo de una escena generada por este simulador puede verse en la Fig. 1.

Como caso de prueba, utilizamos el simulador para responder si el uso de datos sintéticos en el entrenamiento de redes de aprendizaje profundo basadas en nubes de puntos es adecuada para la segmentación de nubes de puntos de escenas forestales reales capturadas mediante VANTs que poseen LiDAR y/o cámaras como sensores principales.

Hoy en día existen pocos *datasets* de nubes de puntos disponibles públicamente que permiten entrenar, validar y evaluar nuevas arquitecturas de redes, entre los que podemos nombrar ShapeNetPart [4] y SemanticKITTI [5] para tareas de clasificación, segmentación y completamiento. En general, ninguno de éstos está diseñado para ambientes específicos, por lo que si se necesitara entrenar una red de un tipo de ambiente particular, se debe generar un *dataset* nuevo en este ambiente. A modo de ejemplo para el caso de ambientes naturales, en [6] se desarrolló un *dataset* de las regiones de Southern Sierra Nevada Mountains, USA; en [7] de Australia y New Zealand, y en [8] de Evo, Finlandia. Estos trabajos se desarrollaron para llevar a cabo una segmentación forestal. De los tres, sólo el último *dataset* está disponible públicamente, limitando la repetibilidad de los experimentos y la comparación entre los trabajos citados. Por lo tanto, el uso del simulador forestal se presenta como alternativa a la recolección masiva de datos para poder realizar tanto el entrenamiento como la evaluación de las arquitecturas, restringiendo la recolección solamente a los datos necesarios para el estudio en cuestión. Las contribu-

ciones de este trabajo son:

- 1) El desarrollo de un simulador de ambientes forestales de código abierto basado en Unity que genera proceduralmente escenas forestales realistas.
- 2) *Datasets* de dominio público de nubes de puntos de escenas forestales sintéticas, que pueden ser utilizados para entrenamiento o evaluación de diferentes arquitecturas.
- 3) Un estudio comparativo de redes de aprendizaje profundo del estado del arte basadas en nubes de puntos para determinar si el uso de datos sintéticos es adecuado para la segmentación de nubes de puntos de ambientes forestales reales.

## II. TRABAJOS RELACIONADOS

El desarrollo de ALS y fotogrametría aérea, sumada a técnicas de visión por computadora, han permitido adquirir reconstrucciones 3D precisas de los ambientes estudiados. Estas reconstrucciones permiten extraer información valiosa para la caracterización, monitoreo y conservación de bosques, tales como la estimación de parámetros estructurales, densidad de árboles, e incluso la identificación de áreas con tala selectiva y/o ilegal, entre otras aplicaciones [3].

En robótica, el uso de simuladores se ha constituido como una herramienta fundamental no sólo para el desarrollo de algoritmos y sistemas, sino también para el estudio de la interacción del robot con el medio. Entre ellos, algunos simuladores se han popularizado en los últimos años, tales como Gazebo y PyBullet, que permiten la simulación de diversos sensores, de fuerzas e incluso realizar tareas de teleoperación [9]. Sin embargo, estos simuladores tienen una capacidad de renderizado limitada frente al uso de motores gráficos, tales como Unity o Unreal Engine. A partir del uso de estos motores surgen otros simuladores, tales como AirSim [10], desarrollado en Unreal Engine y orientado al estudio de drones y vehículos terrestres no tripulados, con la capacidad de controlar su trayectoria sobre un ambiente urbano y semiurbano predefinido, y Flightmare [11], un simulador de cuadrotómetros desarrollado en Unity, con la capacidad de simular cientos de cuadrotómetros en paralelo en ambientes hiperrealistas. Si bien el primero presenta escenas semiurbanas arboladas, y el segundo cuenta con una escena forestal, éstas son estáticas en el sentido en que siempre se realizan los experimentos sobre la misma escena.

En cuanto a simuladores sintéticos de bosques, en [12] se presenta un simulador que utiliza técnicas procedimentales para generar ambientes boscosos, que cuenta con LiDAR, cámara RGB y cámara de profundidad, todos con la capacidad de segmentar la escena en las categorías: fondo, terreno, zonas transitables, troncos, follaje, arbustos, plantas herbáceas y rocas. Un *dataset* sintético generado con este simulador está disponible en [13]. Éste incluye imágenes RGB, mapas de segmentación semántica, mapas de profundidad y la proyección de dos nubes de puntos capturadas con LiDAR sobre el campo de visión RGB, pero no cuenta con la nube de puntos 3D del ambiente. Finalmente, [14] usa este *dataset* como parte

del entrenamiento para detectar áreas sensibles al fuego para ayudar a prevenir incendios forestales.

El estudio de distintos ambientes naturales se ha visto favorecido por el rápido desarrollo de las redes de aprendizaje profundo. Específicamente para el procesamiento de nubes de puntos, en varios trabajos se emplea aprendizaje profundo para la extracción de modelos digitales del terreno (DTM por sus siglas en inglés, *Digital Terrain Models*) a partir de datos capturados con LiDAR [6], [15], [16], [17]. En [7] se segmenta las nubes de puntos de ambientes forestales capturadas con LiDAR en diferentes categorías: terreno, vegetación, restos leñosos y troncos. De forma similar, en [8] se usan técnicas de segmentación para separar la nube de puntos capturada en terreno, sotobosque, troncos, follaje. Este último trabajo es uno de los pocos cuyo *dataset* se encuentra disponible públicamente.

Se han propuesto numerosas arquitecturas de redes para la segmentación, clasificación y completamiento de nubes de puntos, expandiendo los conceptos ya desarrollados para el procesamiento de lenguaje natural o el procesamiento de imágenes, tales como PointNet++ [18], construida mayoritariamente con perceptrones multi-capa, y PointNeXt [19], que actualiza PointNet++ usando estrategias de entrenamiento más modernas. Por otro lado, el desarrollo de la tecnología de *transformers*, basados en capas de *self-attention* [20], presentan algunas ventajas frente al uso más estandarizado de capas convolucionales y de perceptrones multi-capa, pudiendo lograr resultados más precisos con menores tiempos de entrenamiento, a costa de utilizar una mayor cantidad de parámetros, lo que requiere de un volumen mayor de datos de entrada para el entrenamiento. Algunas arquitecturas basadas en este enfoque son PointBERT [21], PointMAE [22] y PointGPT [23]. Estas redes dividen la nube de puntos en numerosas nubes locales más pequeñas, que se codifican en *tokens*, de los cuales se selecciona una proporción que son enmascarados y usados como entrada para el bloque de *transformers*, que finalmente es entrenado para recuperar los tokens originales de las nubes enmascaradas.

## III. MÉTODOS Y MATERIALES

### A. Simulador Forestal

En este trabajo se desarrolló un simulador forestal basado en el motor gráfico Unity, del cual pueden extraerse datos sintéticos de apariencia similar a bosques reales, facilitando el desarrollo y evaluación de nuevas tecnologías sin necesidad de la recolección masiva de datos en ambientes reales. Generar datos sintéticos de forma procedural se hace aún más relevante al considerar los costos económicos y la demanda de tiempo que estas tareas de recolección implican. A continuación presentamos los módulos más importantes del simulador desarrollado.

1) *Generación del terreno*: El simulador genera la malla del terreno utilizando ruido fractal para construir un mapa de alturas para cada uno de sus vértices. Las muestras del ruido se toman de capas de ruido Perlin en diferentes escalas. Esto permite controlar el nivel de detalle y el aspecto general

del terreno. Se tomó ventaja de las funciones de ruido Perlin provistas por Unity para su ejecución en paralelo. Este método es notorio y muy utilizado por la comunidad de desarrollo de videojuegos por sus resultados de aspecto realista.

2) *Generación de árboles, arbustos y plantas:* La vegetación se genera a partir de *pipelines*. Cada *pipeline* es un Grafo Acíclico Dirigido que une *prefabs* (i.e. objetos pre-generados reutilizables, como árboles o arbustos individuales) con su ubicación sobre el terreno. Esto se lleva a cabo utilizando texturas que determinan tanto la probabilidad de instanciación como su densidad sobre la malla del terreno. Cada *pipeline* utiliza diferentes nodos:

- *Source:* importa una textura desde un archivo o desde otro *pipeline*, o bien genera una nueva a partir de un diagrama de Voronoi o de ruido aleatorio.
- *Logic:* aplica operaciones lógicas sobre texturas.
- *Sampling:* aplica variantes del método de sampleo de disco de Poisson.
- *Placement:* genera parámetros de instanciación para los *prefabs* asignados.

Respecto del proceso de sampleo, fueron implementados tanto el método original de sampleo de disco de Poisson de Bridson [24] como una variante que se describe más adelante. En el algoritmo de Bridson, dado un objeto  $a_0$  con un radio  $r$ , nuevos objetos con el mismo radio son añadidos en un anillo con radios  $[r, 2r]$ , hasta alcanzar una cantidad máxima o hasta que no pueda ser colocado ninguno más por superposición de radios, y luego se repite este proceso con un nuevo objeto. Usando un tamaño de celdas de  $r/\sqrt{n}$ , donde  $n$  es la dimensión de una grilla de fondo donde se guardan las muestras, se tiene que cada celda puede contener un único objeto instanciado dentro de su posición. Este proceso es rápido para colocar los objetos, pero produce una distribución de puntos que puede parecer equidistante, sobre todo para valores pequeños de  $r$ , resultando en una distribución poco realista. Como alternativa, se propuso una variante para este método: nuevos puntos son buscados dentro de un anillo de tamaño  $[r_{min}, r_{max}]$ , donde  $r_{min}$  y  $r_{max}$  son un radio mínimo y máximo respectivamente, interpolando la distancia linealmente mediante el valor de una textura en escala de grises en ese punto. Es decir, para valores cercanos a 0 en la textura se generarán puntos a una distancia  $r_{min}$  de  $a_0$ , y lo inverso para valores cercanos a 1. Para poder acomodar estos valores en la grilla, se tomaron celdas de tamaño  $r_{min}/\sqrt{n}$ , donde ahora se tiene una lista de índices para los objetos que pueden caer dentro de una misma celda, y cuya distancia entre ellos es aceptable.

La textura de probabilidad de instanciación actúa como la probabilidad efectiva de generar un objeto en un punto dado. Tener esta textura separada del nodo de sampleo es útil, ya que permite tener áreas enteras donde el número de objetos instanciados pueda reducirse a cero, pudiendo lograrse claros de cualquier forma posible.

El simulador cuenta con algunos *pipelines* básicos para árboles y arbustos, pero pueden generarse tantos como sea requerido. Los modelos de *prefabs* de árboles, arbustos y otras

plantas fueron generados usando el software libre TreeIt [25]. Una muestra de ellos puede verse en la Fig. 2. Más modelos pueden ser fácilmente añadidos de ser necesario. Antes de instanciar cada modelo, algunas transformaciones son aplicadas para agregar más variabilidad a la escena: una rotación aleatoria respecto del eje vertical, una desviación aleatoria del eje vertical respecto del eje vertical de la escena y un cambio de escala aleatorio, todos ellos customizables con valores máximos y mínimos.

3) *Generación de césped:* Ninguno de los métodos de sampleo utilizados para la distribución de la vegetación resulta escalable para generar millones de puntos, manteniendo un *frame rate* adecuado. Por lo tanto, se desarrolló un método paralelizable para la colocación del césped. Éste se genera mediante una instanciación indirecta, donde la geometría de los vértices se construye mediante un *compute shader* (programa que corre en la placa gráfica, por fuera del *pipeline* normal de renderización gráfico) y se le envía al *pipeline* gráfico a través de un *buffer* de memoria compartido. El pseudo-código de muestreo para las hojas de césped se presenta en el Algoritmo 1. Como las ventanas usadas no son solapables, entonces el procesamiento sobre cada ventana puede ser paralelizable. Los puntos generados son luego transformados al sistema de coordenadas del mundo, mediante *raycasting* paralelo. A modo de ejemplo, para una textura de  $256 \times 256$  pixeles, una ventana de  $4 \times 4$  pixeles y un máximo de 1024 puntos por ventana, al algoritmo le toma menos de un segundo generar aproximadamente 4 millones de puntos, corriendo en un procesador Intel Core i9-10900, con 32 GB RAM y una placa NVIDIA GTX 1060.

Cada punto luego pasa a un *compute shader* que genera la geometría para una única hoja de césped. El número de segmentos de la hoja es customizable. Para agregar un efecto realista al césped, las siguientes transformaciones se aplican de forma independiente para cada hoja: un *jitter* sobre la posición de la base de la hoja, para romper con el patrón de grilla que resulta de la etapa anterior; una rotación aleatoria; una curvatura aleatoria sobre la punta de la hoja y una escala aleatoria. Luego, los puntos se regresan al marco de coordenadas global para ser posicionados sobre el terreno, y se aplica una textura a cada hoja para agregar volumen y color.

Aproximadamente 4 millones de hojas de césped, cada una compuesta por 9 puntos, puede ser generada y actualizada cada  $20 \sim 30$  fps con el hardware seleccionado, mostrándose todas simultáneamente. Es importante mencionar que este método de instanciación no puede ser usado con otros tipos de vegetación,



Fig. 2: Modelos de árboles de muestra utilizados en el simulador, generados con el software TreeIt [25]

---

**Algorithm 1** Sampleo de césped

**Input:** textura  $T$ , tamaño de ventana  $t_s$ , número máximo de puntos por ventana  $P$

**Output:** posiciones de las hojas de césped  $G$

- 1: Dividir  $T$  en ventanas no superpuestas según  $t_s$
- 2: Definir una lista vacía  $G$  para las posiciones de las hojas de césped
- 3: **for** cada ventana  $t$  **do**
- 4:   Definir densidad  $d = \frac{1}{t_s^2} \sum_{(i,j) \in t} p_{i,j}$ , donde  $p_{i,j}$  es el valor de la textura en el pixel  $(i,j)$
- 5:   Definir el número de puntos de sampleo en  $t$  según  $p = d \times P$
- 6:   Definir los puntos  $G_t$  distribuyendo  $p$  en un patrón tipo grilla en la ventana
- 7:   Aregar  $G_t$  a  $G$
- 8: **end for**
- 9: **return** Puntos generados  $G$

---

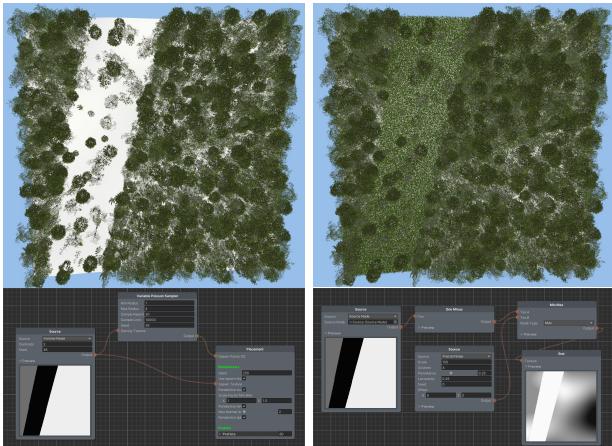


Fig. 3: Arriba: Escena forestal generada con árboles (izquierda), y árboles, arbustos y césped (derecha). Abajo: los *pipelines* correspondientes para instanciar árboles (izquierda), y césped (derecha).

ya que el *jitter* aleatorio impide conservar el radio mínimo de distancia entre objetos instanciados, previniendo colisiones.

4) *Repetibilidad*: Cada escena es generada con una semilla para asegurar su repetibilidad, la cual se transmite a cada *pipeline* de vegetación, así como a los algoritmos de generación de terreno y césped. La Fig. 3 muestra un ejemplo de los *pipelines* usados para generar árboles y césped, junto con una vista superior de la escena obtenida, y la Fig. 4 muestra la misma escena desde una vista frontal y una vista más cercana.

5) *Planificador de misiones de vuelo*: El simulador cuenta con un módulo para planificar trayectorias de vuelo que recorran el terreno construido, generando una grilla con nodos desde los cuales se capturan imágenes que se guardan automáticamente en formato jpg. Permite configurar el tipo de grilla (simple o doble), la altura y velocidad de vuelo, el porcentaje de solapamiento vertical y horizontal de las

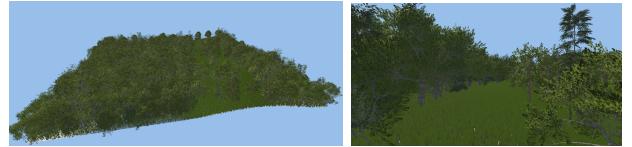


Fig. 4: Izquierda: Vista frontal de una escena generada. Derecha: Vista cercana de la misma escena.

imágenes, el ángulo de la cámara respecto del eje vertical y los parámetros intrínsecos de la cámara.

6) *Extracción de nubes de puntos*: La nube de puntos de la escena generada puede ser exportada de Unity como un archivo .csv. Mediante el etiquetado de los objetos con mallas, éstos pueden ser exportados como distintas categorías, incluyendo pero no limitadas a terreno, follaje, troncos, ramas, arbustos, sotobosque, césped, cactus, madera muerta, entre otros, asignándoles la etiqueta correspondiente a cada punto de la nube de puntos. El tamaño de la nube de puntos de cada escena puede ser alterado agregando más puntos a la malla del terreno, agregando más hojas de césped o cambiando el número de segmentos, o bien importando nuevos *prefabs* con mallas del tamaño deseado. Esto permite a los usuarios generar escenas donde las nubes de puntos varíen de tamaño, y por lo tanto se adecúen a sus respectivas necesidades.

Como una de las contribuciones de este trabajo, el código del simulador presentado se encuentra disponible públicamente en <https://github.com/lmse/forest-simulator>.

#### B. Caso de estudio: segmentación forestal

Como caso de estudio, entrenamos distintas redes de aprendizaje profundo basadas en nubes de puntos con datos sintéticos generados con el simulador presentado, y evaluamos su desempeño con un *dataset* real. El etiquetado manual de las nubes de puntos es una tarea muy demandante y tediosa, donde en muchos casos resulta imposible discernir a qué categoría corresponde cada punto [8]. El uso de un simulador para la generación del *dataset* soluciona este problema, ya que permite etiquetar a cada punto de forma automática. Además, como la tecnología *transformer* requiere grandes volúmenes de datos para el entrenamiento, generar datos sintéticos de forma procedimental se hace aún más relevante.

En este trabajo nos proponemos entrenar distintas redes de aprendizaje profundo para poder segmentar nubes de puntos de ambientes forestales en las mismas categorías que [8]: troncos, follaje, sotobosque (incluyendo césped, arbustos y toda otra vegetación no arbórea) y terreno. Esto nos permite además diferenciar los estratos arbóreos del DTM, que se corresponde con los puntos del terreno. Seleccionamos 4 arquitecturas de redes del estado del arte: PointNeXt, PointBERT, PointMAP y PointGPT, a las que entrenamos con datos sintéticos de ambientes forestales generados por nuestro simulador. De estas cuatro, las últimas tres están construidas utilizando *transformers*, y están disponibles versiones preentrenadas con el *dataset* ShapeNetPart, por lo que pueden ser especializadas en el ambiente de interés usando una etapa de *fine-tuning*. En

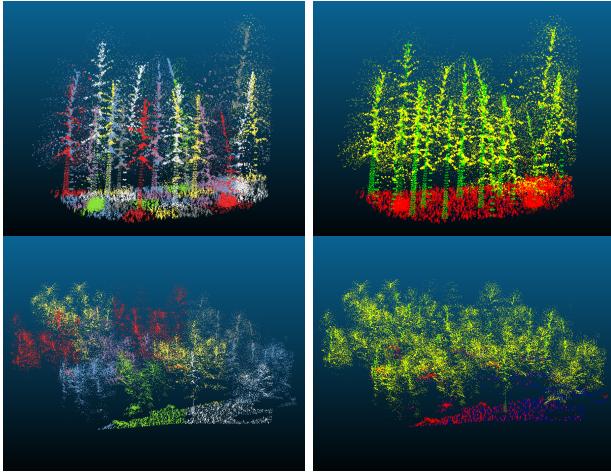


Fig. 5: Arriba: Escena de ejemplo sin oclusión, segmentada utilizando K-means (izquierda) y segmentada según etiquetado: azul, verde, amarillo y rojo correspondiendo a terreno, troncos, follaje y sotobosque, respectivamente (derecha). Abajo: Mismo ejemplo para otra escena, pero simulando además oclusión desde una vista superior.

cambio, PointNeXt se basa en el uso de perceptrones multicapa, y dado que no contamos con una versión previamente preentrenada, realizamos el entrenamiento de la red de cero.

Se crearon dos *datasets*: **Lidar-like** y **Camera-like**, para entrenar las arquitecturas seleccionadas, simulando tanto nubes de puntos obtenidas por LiDAR como aplicando algoritmos de *Structure from Motion* a imágenes tomadas por una cámara. Para el segundo se utilizó un método para incluir oclusión [26], dado que numerosos puntos no son visibles desde la vista superior de la escena. Las cuatro categorías antes mencionadas fueron extraídas de manera automática desde el simulador. Se añadió ruido aleatorio de media cero a las nubes de puntos para agregar variabilidad, y luego se las particionó utilizando los algoritmos de *clustering K-means*, generando así las nubes de puntos con las que se entrenaron las redes. En la Fig. 5 puede observarse un ejemplo de ambos *datasets* particionados utilizando *K-means*, y segmentados en las categorías mencionadas. Ambos *datasets* se encuentran disponibles públicamente en <https://github.com/lirse/synthetic-forest-datasets>. De esta forma garantizamos la repetibilidad de los resultados, e incrementamos la cantidad de *datasets* de nubes de puntos disponibles on-line públicamente.

Para la evaluación de las arquitecturas entrenadas se utilizó el *dataset* Evo, publicado en [8]. Se utilizó una versión con oclusión añadida de este *dataset* para testear las arquitecturas entrenadas con el *dataset* sintético Camera-like.

#### IV. RESULTADOS Y DISCUSIONES

Las cuatro arquitecturas fueron entrenadas con un procesador AMD Phenom II X6 1075T, con 32GB RAM y dos NVIDIA RTX 3090 conectadas con un puente SLI. Se utilizaron 50 epochs en cada red, con un tiempo promedio

TABLE I: Resultados obtenidos al testear sobre el dataset sintético forestal y sobre el dataset Evo, al entrenar con el dataset LiDAR-like.

Network	LiDAR-like Dataset			Evo Dataset		
	Overall Accuracy	Class Avg. Accuracy	Class Avg. mIoU	Overall Accuracy	Class Avg. Accuracy	Class Avg. mIoU
PointNeXt	0.9348	0.8339	0.7314	<b>0.7695</b>	0.6436	<b>0.5153</b>
PointBERT	0.9551	0.8719	0.8021	0.7195	<b>0.6462</b>	0.4206
PointMAE	<b>0.9575</b>	<b>0.8743</b>	<b>0.8029</b>	0.72788	0.61103	0.4278
PointGPT	0.9414	0.8257	0.7516	0.6417	0.5271	0.3620

TABLE II: Matriz de confusión para las cuatro redes estudiadas.

	PointNeXt				PointBERT			
	Terreno	Tronco	Follaje	Sotobosque	Terreno	Tronco	Follaje	Sotobosque
Terreno	<b>1055469</b>	4234	423	386498	756515	16905	6451	<b>1314363</b>
Tronco	468	<b>191869</b>	135275	59160	3386	<b>401490</b>	184272	67868
Follaje	10201	254302	<b>8310787</b>	1220544	11031	1088940	<b>6703359</b>	765463
Sotobosque	1105198	21323	1067	<b>1128622</b>	373703	38963	22493	<b>2129238</b>

	PointMAE				PointGPT			
	Terreno	Tronco	Follaje	Sotobosque	Terreno	Tronco	Follaje	Sotobosque
Terreno	315040	2506	18805	<b>1756467</b>	363932	8427	13425	<b>1709711</b>
Tronco	193	<b>340098</b>	187716	129111	593	219103	206613	<b>229875</b>
Canopy	6316	779671	<b>6986603</b>	7969519	11898	1049754	<b>5682647</b>	1824946
Sotobosque	49187	13302	38252	<b>2465254</b>	112227	23866	31023	<b>2387380</b>

de entrenamiento de 472 segundos por epoch en el dataset LiDAR-like, y de 239 segundos para el dataset Camera-like.

#### A. Experimento con dataset LiDAR-like

La Tabla I muestra los resultados de la evaluación de las redes entrenadas con el dataset LiDAR-like sobre el dataset Evo. La matriz de confusión resultante para cada red se presenta en la Tabla II. Se puede observar que, a pesar de tener una precisión general buena, las redes tienen dificultad para diferenciar sotobosque de terreno, especialmente cuando la vegetación tiene un nivel cercano al suelo. Pese a esto, puede apreciarse que las redes tienen un buen desempeño segmentando los árboles del resto de las categorías, e incluso segmentando troncos del follaje. PointNeXt obtiene un mejor resultado clasificando los puntos del *dataset* Evo, por lo que parece más adecuado para utilizarse en ambientes forestales.

#### B. Experimento con dataset Camera-like

La Tabla III muestra los resultados de la evaluación de las redes entrenadas con el dataset Camera-like sobre el dataset Evo con oclusión. Las matrices de confusión de cada red pueden observarse en la Tabla IV. De forma similar al caso anterior, las arquitecturas tienen dificultades para diferenciar puntos del terreno de los de sotobosque, y como solo una porción chica de puntos de troncos quedan visibles, especialmente los de la base de los troncos, también son confundidos frecuentemente con el sotobosque. Podemos ver que el desempeño es significativamente más bajo que en el caso sin oclusión. En este caso, PointMAE obtiene un resultado ligeramente mejor que el resto de las redes, aunque todas tienen respuestas similares.

TABLE V: Resultados obtenidos considerando únicamente las categorías: árbol y no-árbol, al testear con el Dataset Evo, al entrenar con los datasets LiDAR-like y Camera-like.

Network	LiDAR-like Dataset			Camera-like Dataset		
	Overall Accuracy	Class Avg.	Class Avg. mIoU	Overall Accuracy	Class Avg.	Class Avg. mIoU
PointNeXt	0.9051	0.9330	0.8035	0.9414	<b>0.9180</b>	0.8765
PointBERT	<b>0.9328</b>	<b>0.9449</b>	<b>0.8652</b>	<b>0.9487</b>	0.9108	<b>0.8773</b>
PointMAE	0.9276	0.9416	0.8561	0.9408	0.8982	0.8596
PointGPT	0.8454	0.8797	0.7252	0.9372	0.8906	0.8497

TABLE III: Resultados obtenidos al testear sobre el dataset sintético forestal y sobre el dataset Evo, al entrenar con el dataset Camera-like.

Network	Camera-like Dataset			Evo Dataset		
	Overall Accuracy	Class Avg.	Class Avg. mIoU	Overall Accuracy	Class Avg.	Class Avg. mIoU
PointNeXt	<b>0.9497</b>	0.6875	0.5926	0.6236	0.4868	0.4120
PointBERT	0.9369	0.7246	0.6303	0.6549	0.5085	<b>0.4533</b>
PointMAE	0.9401	<b>0.7289</b>	<b>0.6419</b>	<b>0.6822</b>	<b>0.5448</b>	0.4248
PointGPT	0.9336	0.7180	0.6175	0.5794	0.5445	0.3909

TABLE IV: Matriz de confusión para las cuatro redes estudiadas.

PointNeXt					PointBERT				
Terreno	Terreno	Tronco	Follaje	Sotobosque	Terreno	Terreno	Tronco	Follaje	Sotobosque
Terreno	<b>6566</b>	0	0	6229	<b>8923</b>	0	0	5388	
Tronco	68	0	24	<b>299</b>	102	23	40	<b>484</b>	
Follaje	100	0	<b>6990</b>	909	140	2	<b>6226</b>	639	
Sotobosque	2391	0	0	<b>3048</b>	<b>2392</b>	0	0	2265	

PointMAE					PointGPT				
Terreno	Terreno	Tronco	Follaje	Sotobosque	Terreno	Terreno	Tronco	Follaje	Sotobosque
Terreno	<b>9508</b>	0	0	4653	<b>7283</b>	3	0	7069	
Tronco	71	75	38	<b>445</b>	80	219	25	<b>308</b>	
Follaje	108	18	<b>6028</b>	951	137	415	<b>5278</b>	1126	
Sotobosque	2176	0	0	<b>2553</b>	2018	17	0	<b>2646</b>	

### C. Segmentación de árbol y no-árbol

La Tabla V refleja los resultados alcanzados si se consideran las categorías: árbol (que engloba tanto troncos como follaje) y no-árbol (que incluye el terreno y el sotobosque). Puede verse que la segmentación de los árboles del resto del ambiente alcanza un alto nivel de precisión en todas las redes, siendo PointBERT la cual alcanza los mejores resultados.

### V. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se desarrolló un simulador de código abierto basado en Unity, que genera escenas forestales realistas de manera procedimental, y que permite la extracción de datos para su utilización en el desarrollo, comparación y evaluación de nuevas tecnologías. Como caso de estudio, se crearon datasets sintéticos de nubes de puntos utilizando el simulador y etiquetado automático de los puntos. Éstos fueron empleados para entrenar cuatro redes de aprendizaje profundo basadas en nubes de puntos del estado del arte. Las redes ya entrenadas fueron evaluadas utilizando el dataset forestal real Evo. Los resultados muestran que es posible utilizar datos sintéticos para entrenar redes de aprendizaje profundo para la segmentación de nubes de puntos en ambientes forestales. Entre las redes utilizadas, PointNeXt dió mejores resultados cuando se la entrenó con el dataset LiDAR-like, mientras que PointMAE obtuvo ligeramente mejores resultados al entrenar con Camera-like. Al considerar sólo las categorías árbol y no-árbol, vemos que se alcanza una muy buena precisión al segmentar los árboles del resto del ambiente, logrando

PointBERT los mejores resultados. Como trabajo futuro, se incorporará un sensor LiDAR para poder realizar estudios similares a partir de las nubes de puntos capturadas por éste. También se desea implementar un módulo de ejecución de trayectoria de vehículos terrestres, que permitiría la aplicación de algoritmos de SLAM en este tipo de ambiente.

### REFERENCES

- [1] A. Murtiyoso, S. Holm, H. Riihimäki, A. Krucher, H. Griess, V. C. Griess, J. Schweier, *Virtual forests: a review on emerging questions in the use and application of 3D data in forestry*, International Journal of Forest Engineering **35**(1), 29–42, 2024.
- [2] F. Pessacg, F. Gómez-Fernández, M. Nitsche, N. Cham, S. Torrella, R. Ginzburg, P. De Cristóforis, *Simplifying UAV-based photogrammetry in forestry: How to generate accurate digital terrain model and assess flight mission settings*, Forests **13**(2), 173, 2022
- [3] N. Guimarães, L. Pádua, P. Marques, N. Silva, E. Peres,, J. J. Sousa, *Forestry remote sensing from unmanned aerial vehicles: A review focusing on the data, processing and potentialities*, Remote Sensing **12**(6), 2020
- [4] L. Yi, V. G. Kim, D. Ceylan, I. C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, L. Guibas, *A scalable active framework for region annotation in 3d shape collections*, ACM Transactions on Graphics **35**(6), 1–12, 2016
- [5] J. Behley, M. Garbade, A. Milioti, J. Quenzel, S. Behnke, C. Stachniss, J. Gall, *Semanticitti: A dataset for semantic scene understanding of lidar sequences*, In: Proceedings of the IEEE/CVF international conference on computer vision, pp. 9297–9307. IEEE Xplore, Seoul, Korea, 2019
- [6] S. Jin, Y. Su, X. Zhao, T. Hu, Q. Guo, *A point-based fully convolutional neural network for airborne lidar ground point filtering in forested environments*, IEEE journal of selected topics in applied earth observations and remote sensing **13**, 3958–3974, 2020
- [7] S. Krisanski, M. S. Taskhiri, S. Gonzalez Aracil, D. Herries, P. Turner, *Sensor agnostic semantic segmentation of structurally diverse and complex forest point clouds using deep learning*, Remote Sensing **13**(8), 2021
- [8] R. Kaijaluoto, A. Kukko, A. El Issaoui, J. Hyypää, H. Kaartinen, *Semantic segmentation of point cloud data using raw laser scanner measurements and deep neural networks*, ISPRS Open Journal of Photogrammetry and Remote Sensing **3**, 100011, 2022
- [9] J. Collins, S. Chand, A. Vanderkop, D. Howard, *A review of physics simulators for robotic applications*, IEEE Access **9**, 51416–51431, 2022
- [10] S. Shah, D. Dey, C. Lovett, A. Kapoor, *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*, In Field and Service Robotics: Results of the 11th International Conference, 621–635, Springer International Publishing, 2018
- [11] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, D. Scaramuzza, *Flightmare: A flexible quadrotor simulator*, In Conference on Robot Learning, 1147–1157, 2018
- [12] R. Nunes, J. F. Ferreira, P. Peixoto, *Procedural generation of synthetic forest environments to train machine learning algorithms*, In: ICRA 2022 Workshop in Innovation in Forestry Robotics: Research and Industry Adoption, 2022
- [13] SynPhoRest Dataset Homepage, <https://zenodo.org/records/6369446>. Last accessed 21 Feb 2024
- [14] D. J. Russell, T. Arevalo-Ramirez, C. Garg, W. Kuang, F. Yandun, D. Wettergreen, G. Kantor, *UAV Mapping with Semantic and Traversability Metrics for Forest Fire Mitigation*, In: ICRA 2022 Workshop in Innovation in Forestry Robotics: Research and Industry Adoption, 2022
- [15] X. Hu, Y. Yuan, *Deep-learning-based classification for DTM extraction from ALS point cloud*, Remote sensing **8**(9), 730, 2016
- [16] H. A. Lê, F. Guiotte, M. T. Pham, S. Lefèvre, T. Corpetti, *Learning Digital Terrain Models From Point Clouds: ALS2DTM Dataset and Rasterization-Based GAN*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **15**, 4980–4989, 2022
- [17] B. Li, H. Lu, H. Wang, J. Qi, G. Yang, Y. Pang, H. Dong, Y. Lian, *Terrain-Net: A Highly-Efficient, Parameter-Free, and Easy-to-Use Deep Neural Network for Ground Filtering of UAV LiDAR Data in Forested Environments*, Remote Sensing **14**(22), 5798, 2022
- [18] C.R. Qi, L. Yi, H. Su, L. J. Guibas, *Pointnet++: Deep hierarchical feature learning on point sets in a metric space*, Advances in neural information processing systems **30**, 2017

- [19] G. Qian, Y. Li, H. Peng, J. Mai, H. Hammoud, M. Elhoseiny, B. Ghanem, *Pointnext: Revisiting pointnet++ with improved training and scaling strategies*, Advances in Neural Information Processing Systems **35**, 23192–23204, 2022
- [20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems **30**, 2017
- [21] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, J. Lu, *Point-bert: Pre-training 3d point cloud transformers with masked point modeling*, In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 19313–19322. IEEE Xplore, New Orleans, United States of America, 2022
- [22] Y. Pang, W. Wang, F. E. Tay, W. Liu, Y. Tian, L. Yuan, *Masked autoencoders for point cloud self-supervised learning*, In: European Conference on Computer Vision–ECCV, pp. 604–621. Springer, 2022
- [23] G. Chen, M. Wang, Y. Yang, K. Yu, L. Yuan, Y. Yue, *PointGPT: Auto-regressively Generative Pre-training from Point Clouds*, Advances in Neural Information Processing Systems **36**, 2024
- [24] R. Bridson, *Fast Poisson disk sampling in arbitrary dimensions*, SIGGRAPH sketches **10**(1), 1, 2007
- [25] TreeIt Homepage, <http://www.evolved-software.com/treeit/treeit>. Last accessed 27 Feb 2024
- [26] S. Katz, A. Tal, R. Basri, *Direct visibility of point sets*, ACM Transactions on Graphics **26**(3), 24, 2007