

# SIMULACIÓN DE ESCANEADOS 3D

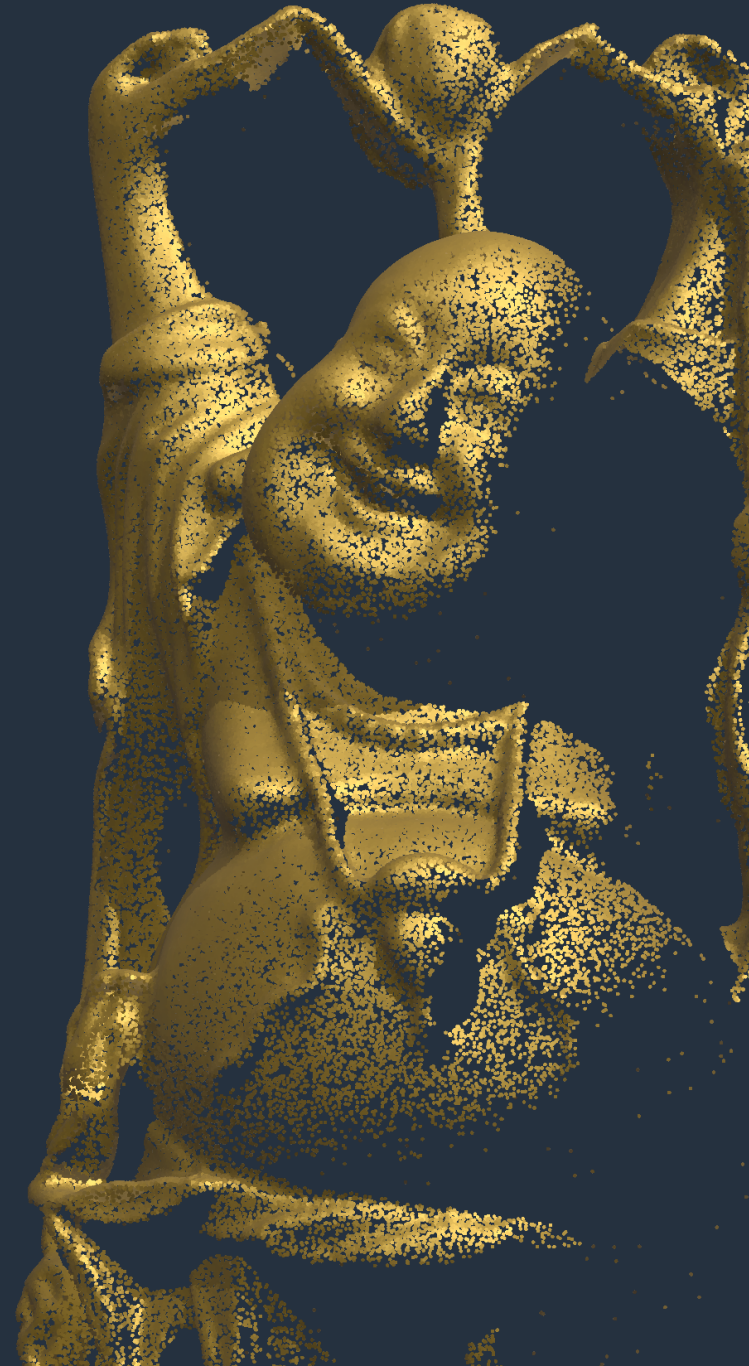
Programación avanzada en ingeniería geomática




Doctorado en Tecnologías de la  
Información y la Comunicación

Curso académico 2019-2020

Alfonso López Ruiz




- 1




Simular metodología de escaneo LiDAR

  - Simular comportamiento físico básico.
  - Simular errores más comunes del sensor.
- 2



Desarrollar múltiples tipos de dispositivos LiDAR

  - Terrestre, aéreo, etc.
  - Sensor parametrizado.
- 3



Simulación parametrizada

  - Interacción con interfaz gráfica.



4



Obtener un tiempo de respuesta reducido

- Reducir espera del usuario.
- Generación de un gran número de conjuntos de datos.

5



Representación de diferentes escenarios

- Escenario procedural.



## Reducción de costes

- Elimina la necesidad de disponer de un dispositivo.
- Disminuye tiempo empleado en la obtención de datos.
- Posibilidad de evaluar diversos sensores.



## Obtención de conjuntos de datos etiquetados

- Obtención de datasets para problemas de clasificación de nubes de puntos y segmentación semántica.
- Asignación de clases automática y sin errores.



## Conjuntos de datos sintéticos

- Escenarios 3D modelados. Flexibilidad de escenario.
- Generación de un gran número de nubes de puntos.



## Preparación y verificación de trabajos de campo

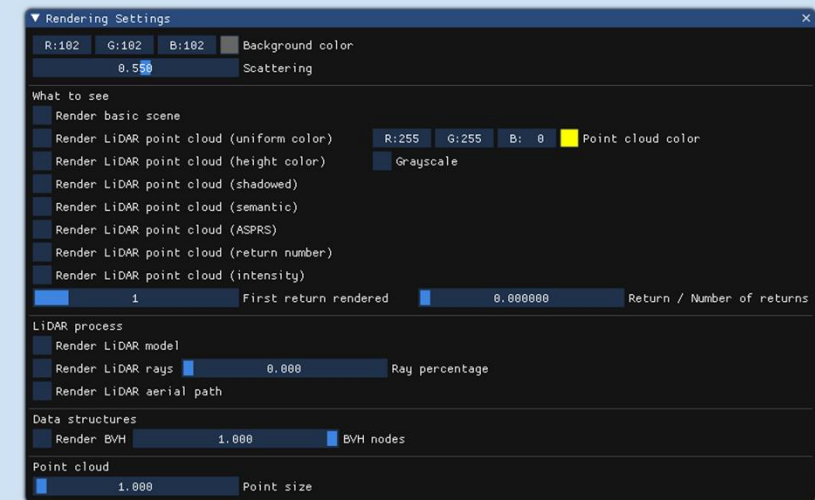
- Observación de comportamiento de LiDAR frente a un mismo escenario empleando diferentes configuraciones.



**Escena renderizada**

Mallas de triángulos y nubes de puntos

## Interfaz gráfica de usuario



# TECNOLOGÍA UTILIZADA

---

Lenguaje de programación

**C++**

API de rendering 2D/3D

**OpenGL (4.5)**

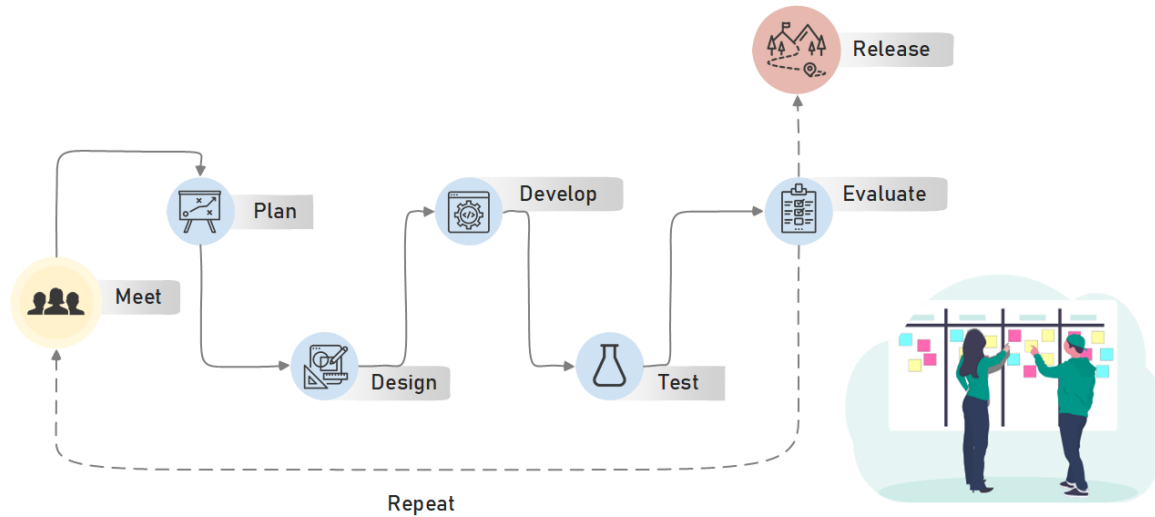
Computación paralela

**Compute shaders**

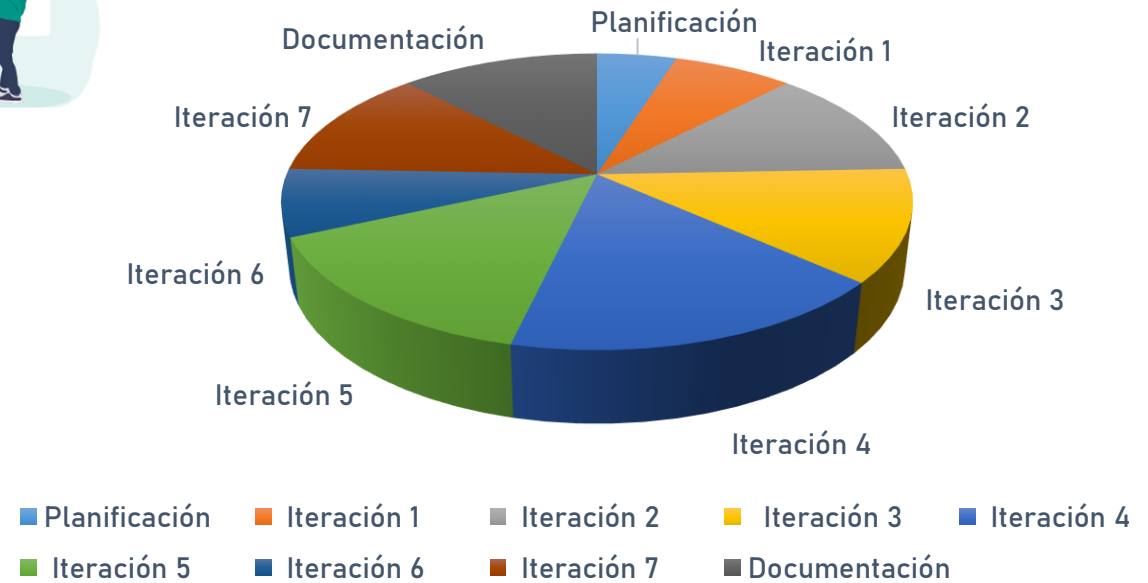
Interfaz gráfica de usuario

**Dear ImGui**





## Esfuerzo de iteraciones y tareas

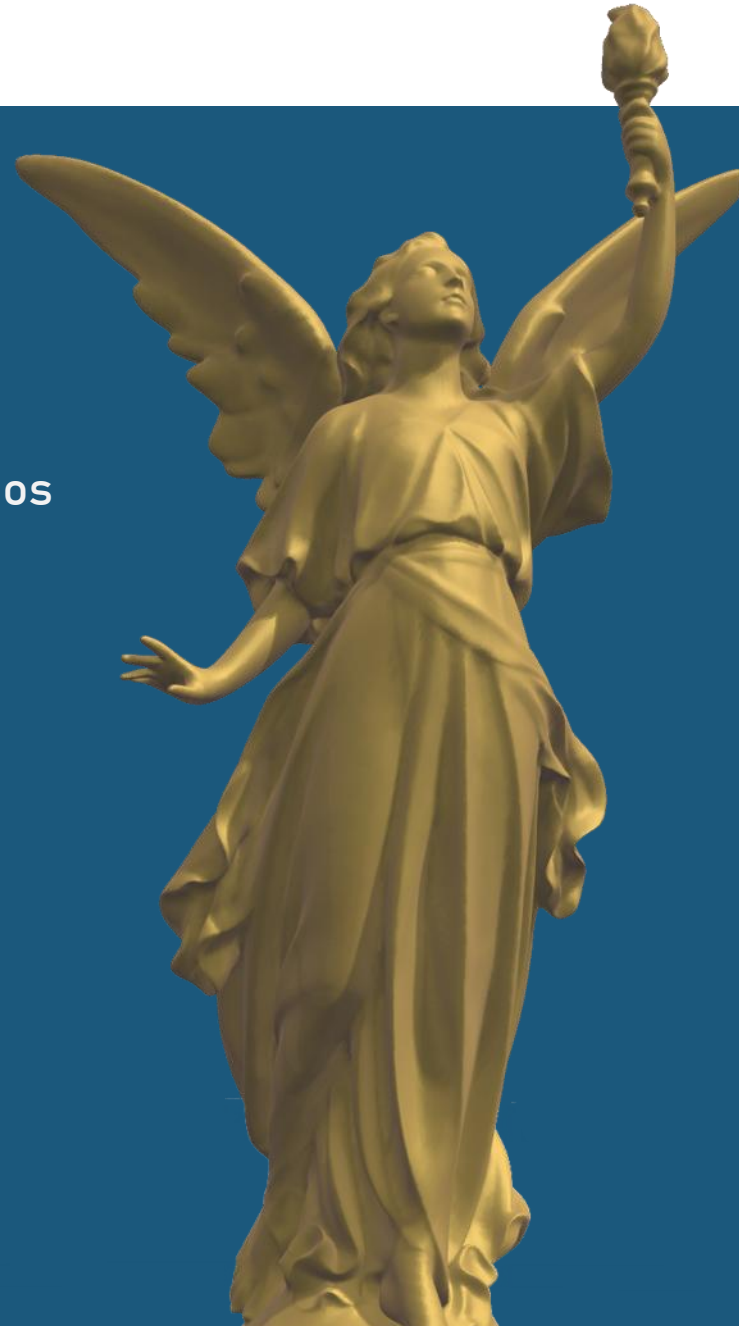




---

# PRIMERA ITERACIÓN

Desarrollo de una aplicación gráfica que conforma la base del proyecto. Sobre una base previa, se añaden conceptos observados en la bibliografía de esta área (Akenine-Möller 2018)





Key light



Fill light

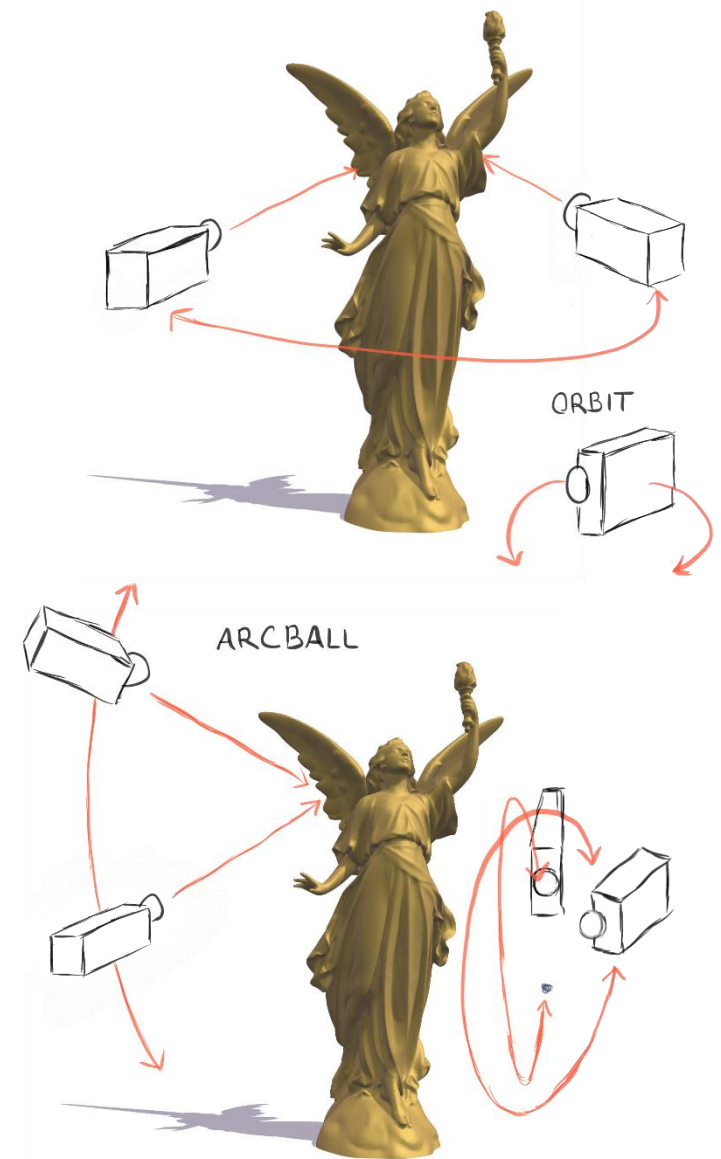
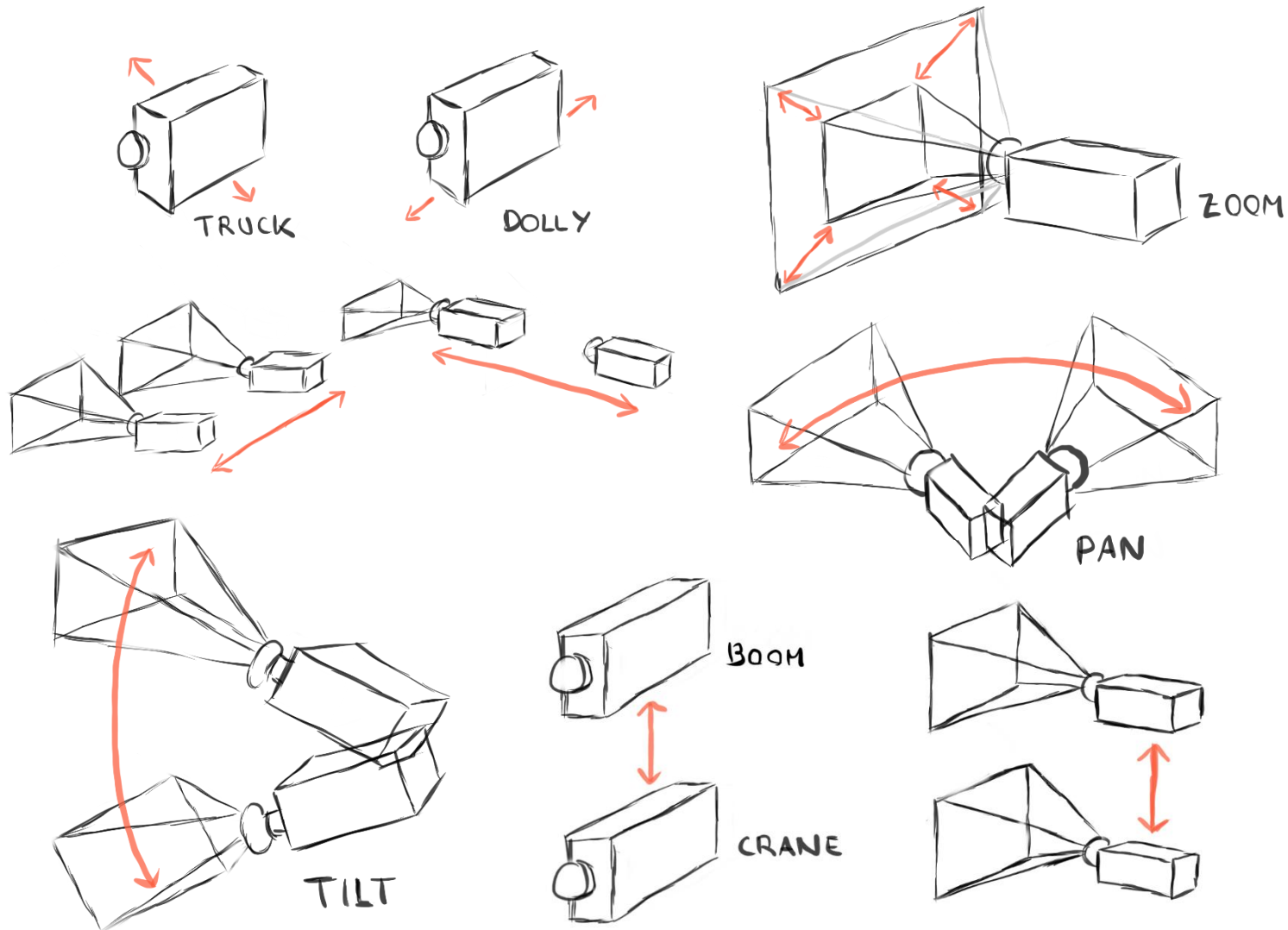


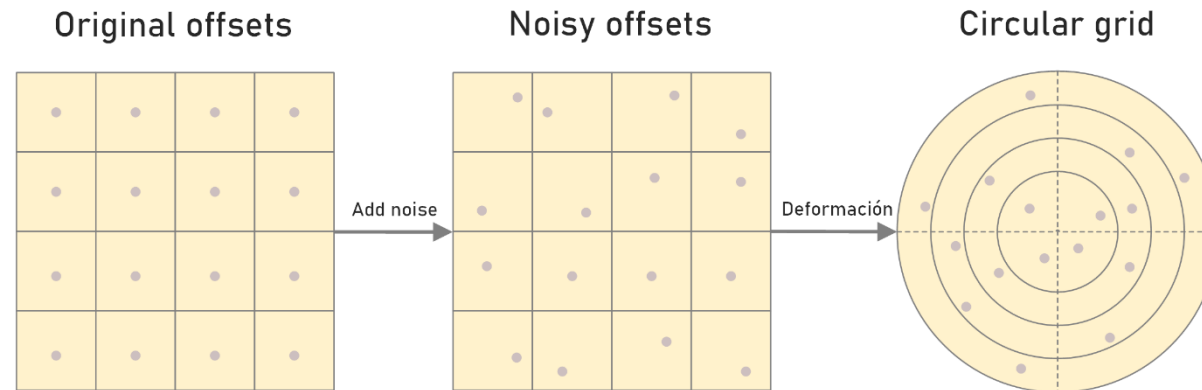
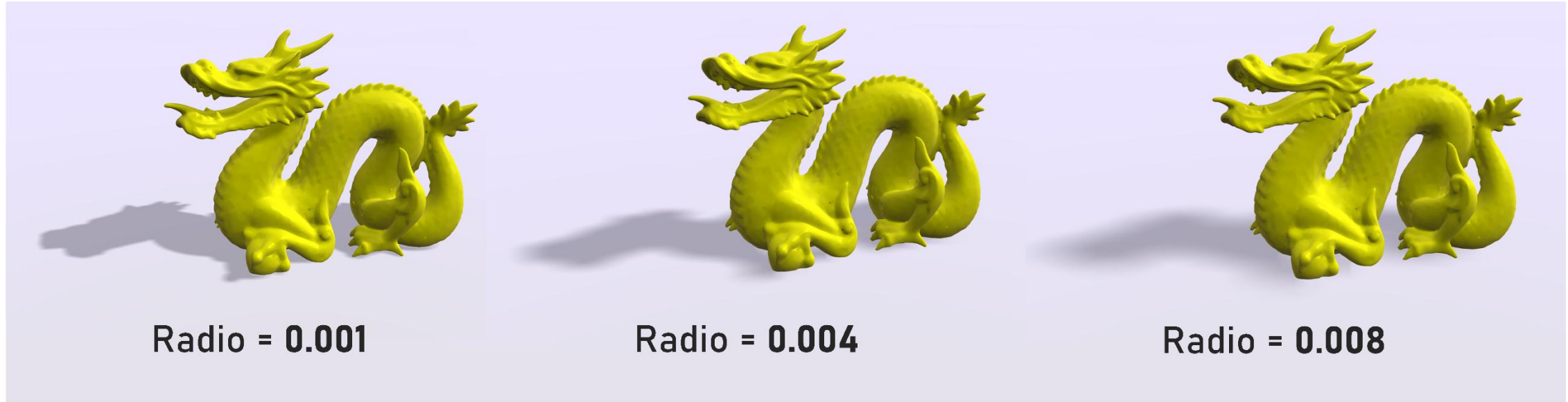
Rim light



# APLICACIÓN GRÁFICA. CÁMARA

11











$c_1 = c_2 = 0$   
 $c_3 = 0.022$

**Default model**



$r_{\text{start}} = 6$   
 $r_{\text{end}} = 14$

**Interval model**



$f_c = 0.9; f_{\text{max}} = 0.3$   
 $r_c = 8$   
 $s_e = 3$

**Pixar model**







Vértices

**2.288.011**

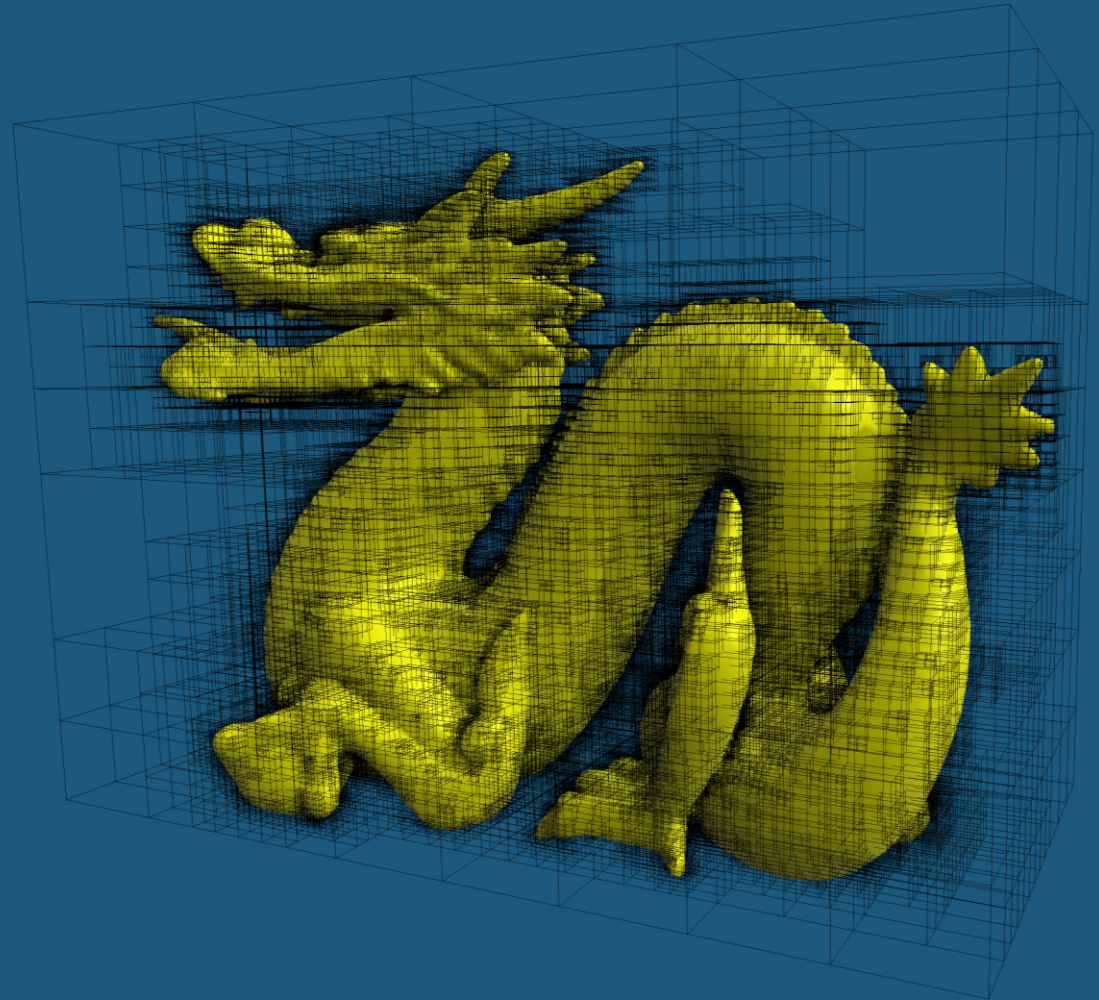
Triángulos

**6.683.902**

---

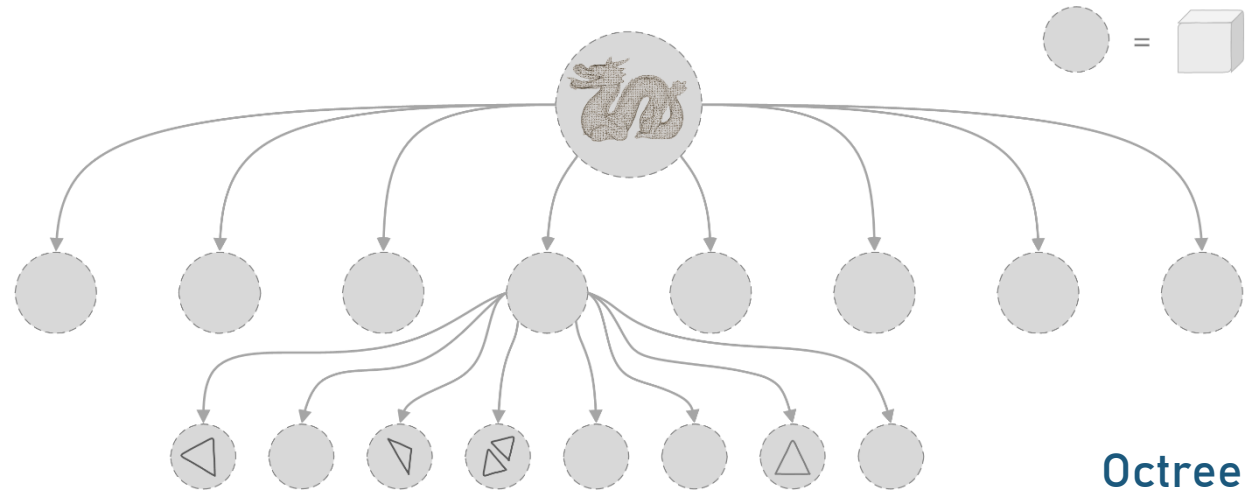
## SEGUNDA ITERACIÓN

Desarrollo de una primera simulación en CPU. Se construye sobre estructuras de datos y algoritmos conocidos de antemano, aunque también es necesario acudir a intersecciones geométricas desconocidas hasta este momento.

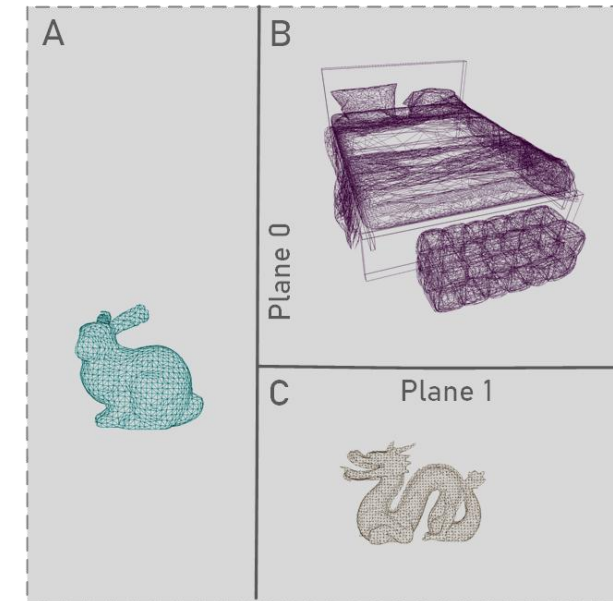
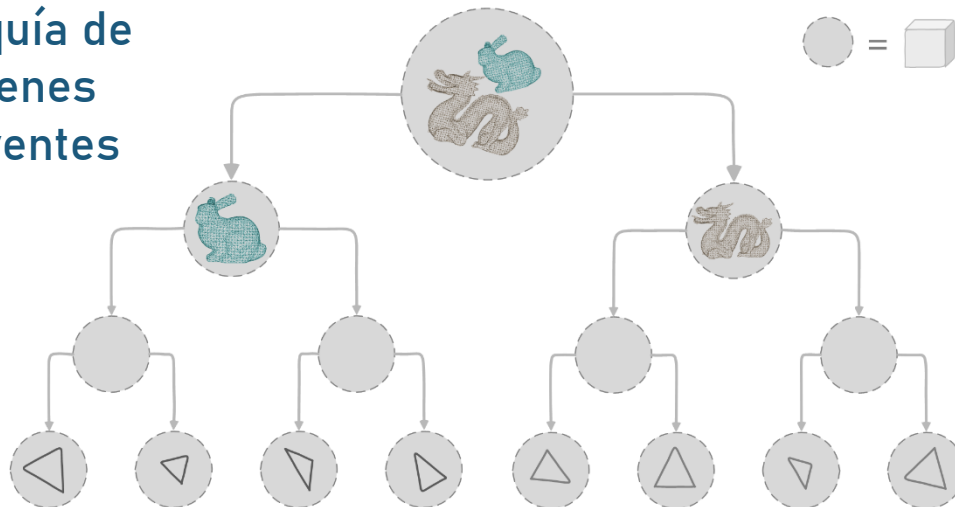


# SELECCIÓN DE ESTRUCTURA DE DATOS

17

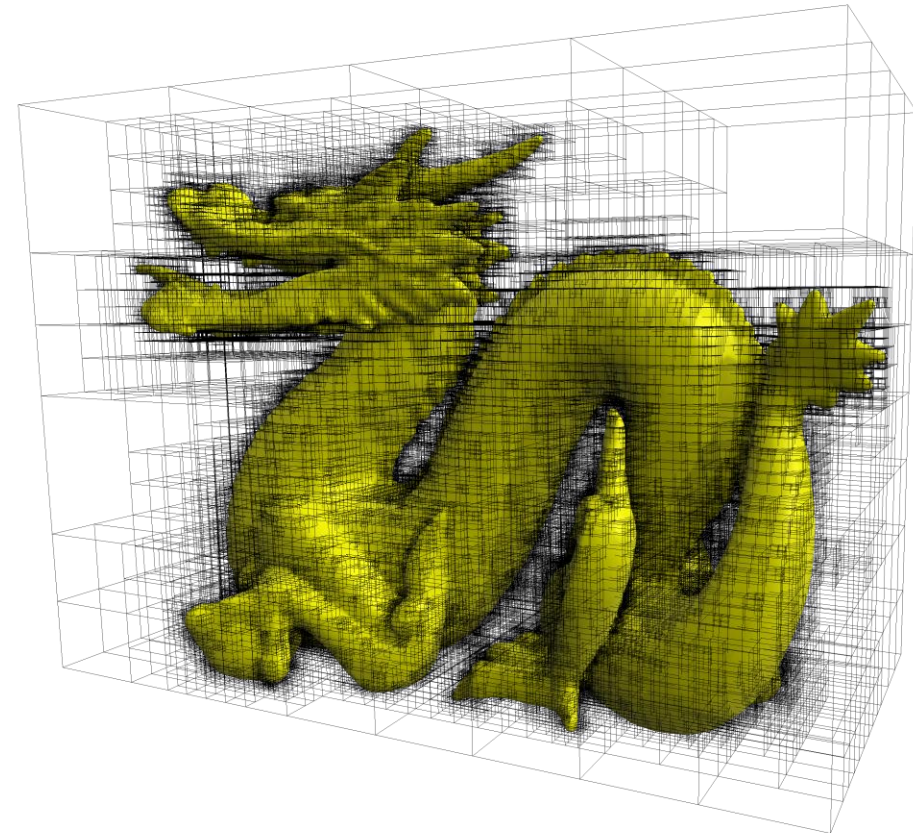
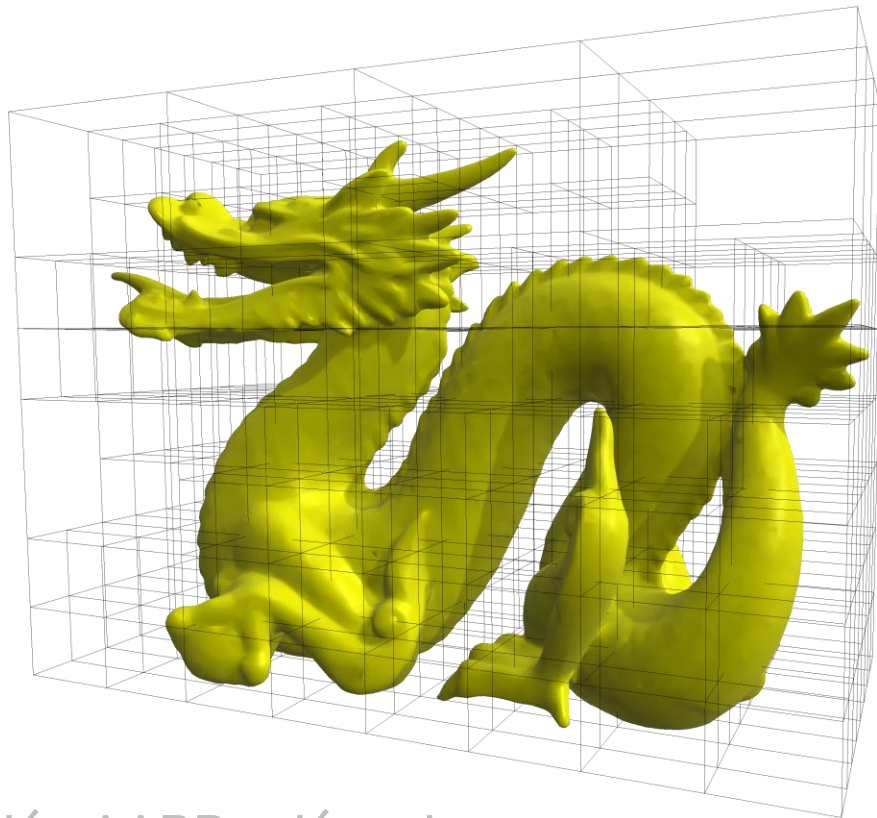


Jerarquía de volúmenes envolventes





Octree. Primitiva: triángulo



Intersección AABB-triángulo

Fast 3D Triangle-Box Overlap Testing. Akenine-Möller, Tomas

## Octree traversal

An Efficient Parametric Algorithm for Octree Traversal. Revelles, J.; Ureña, C.; Lastra, M.

### 1 Selección de plano de entrada: YZ, XZ, XY

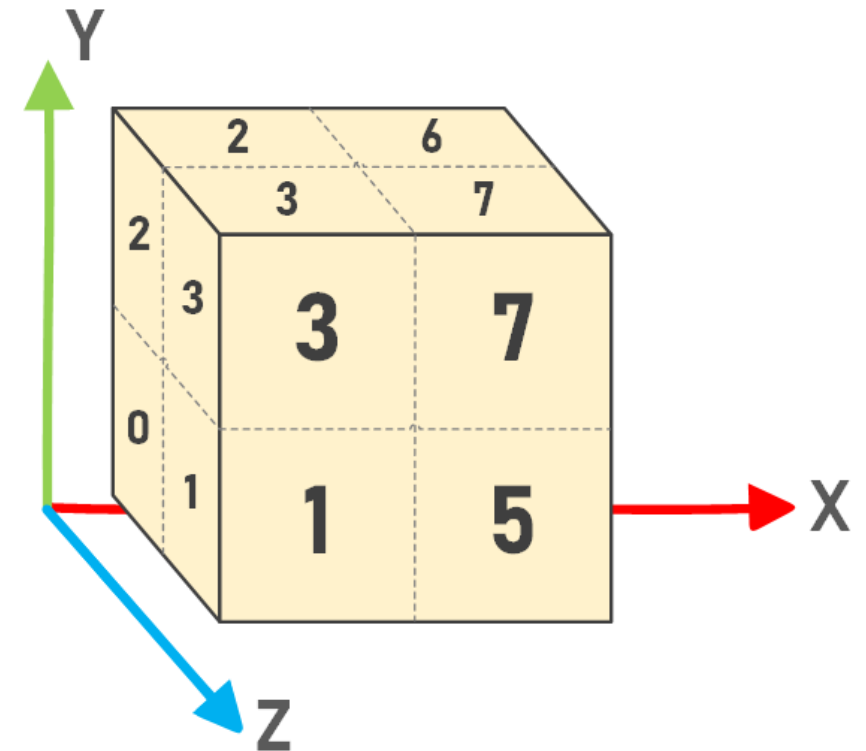
Máximo valor paramétrico:  $t_{0x}$ ,  $t_{0y}$ ,  $t_{0z}$

### 2 Selección de nodo

Modificar bits en función de  $t_0$ ,  $t_m$

### 3 Selección de siguiente nodo

En función de nodo actual y plano de salida



Octree traversal

An Efficient Parametric Algorithm for Octree Traversal. Revelles, J.; Ureña, C.; Lastra, M.





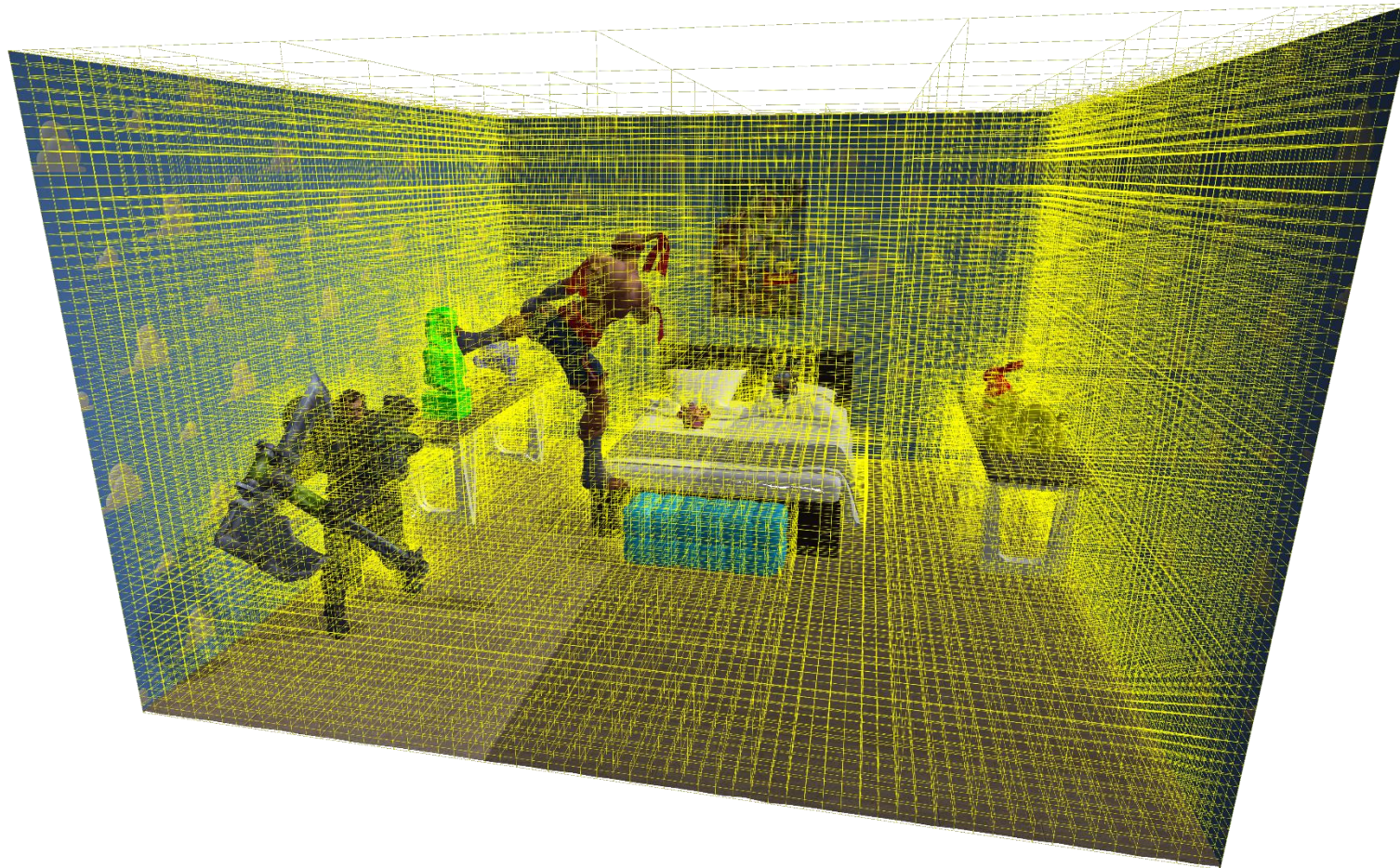
## 1 Octree de AABBs (escena)



## Octree de triángulos (modelo)



## 2 Octree de triángulos (escena)





## Otros algoritmos de intersección

### 1 Algoritmo de intersección rayo-triángulo

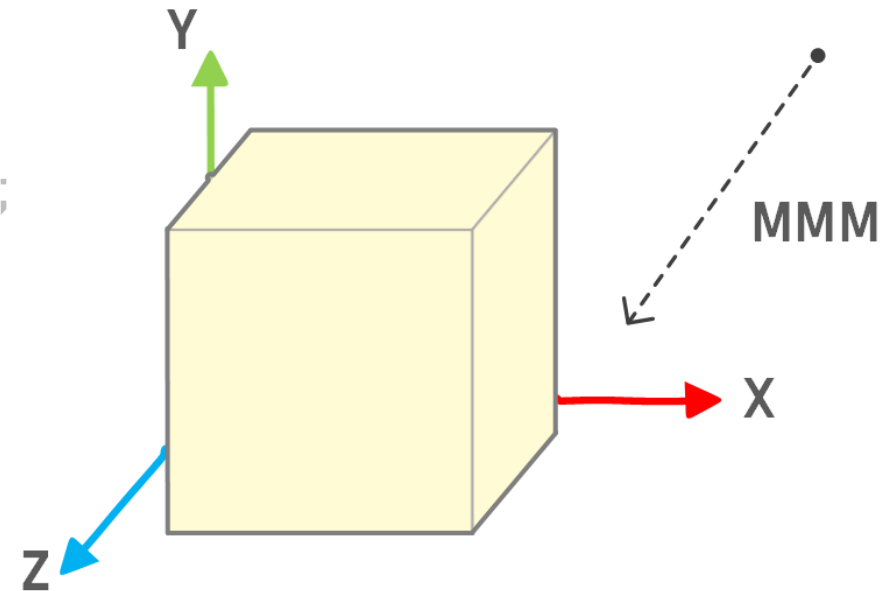
Fast, minimum storage ray-triangle intersection. Möller, Tomas; Trumbore, Ben

### 2 Algoritmo de intersección rayo-AABB

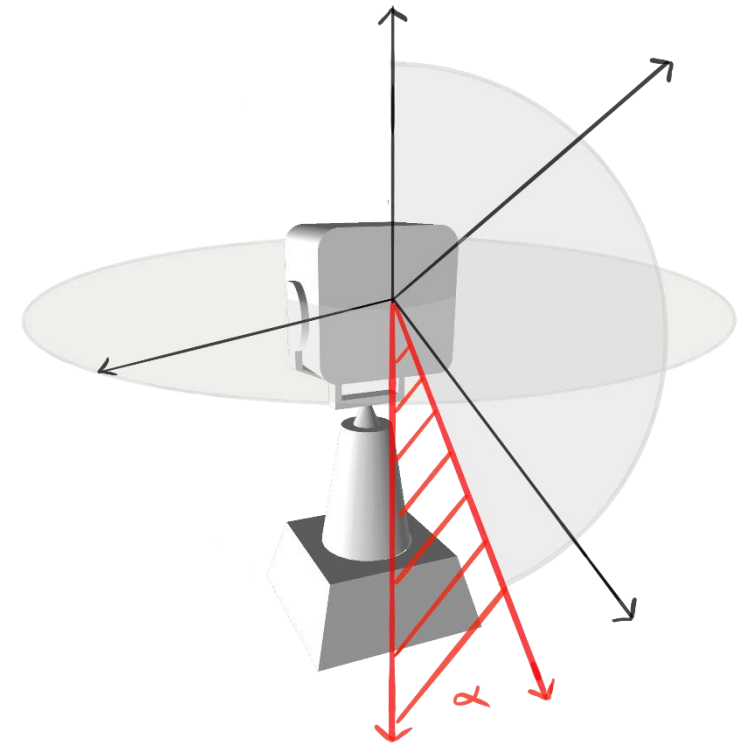
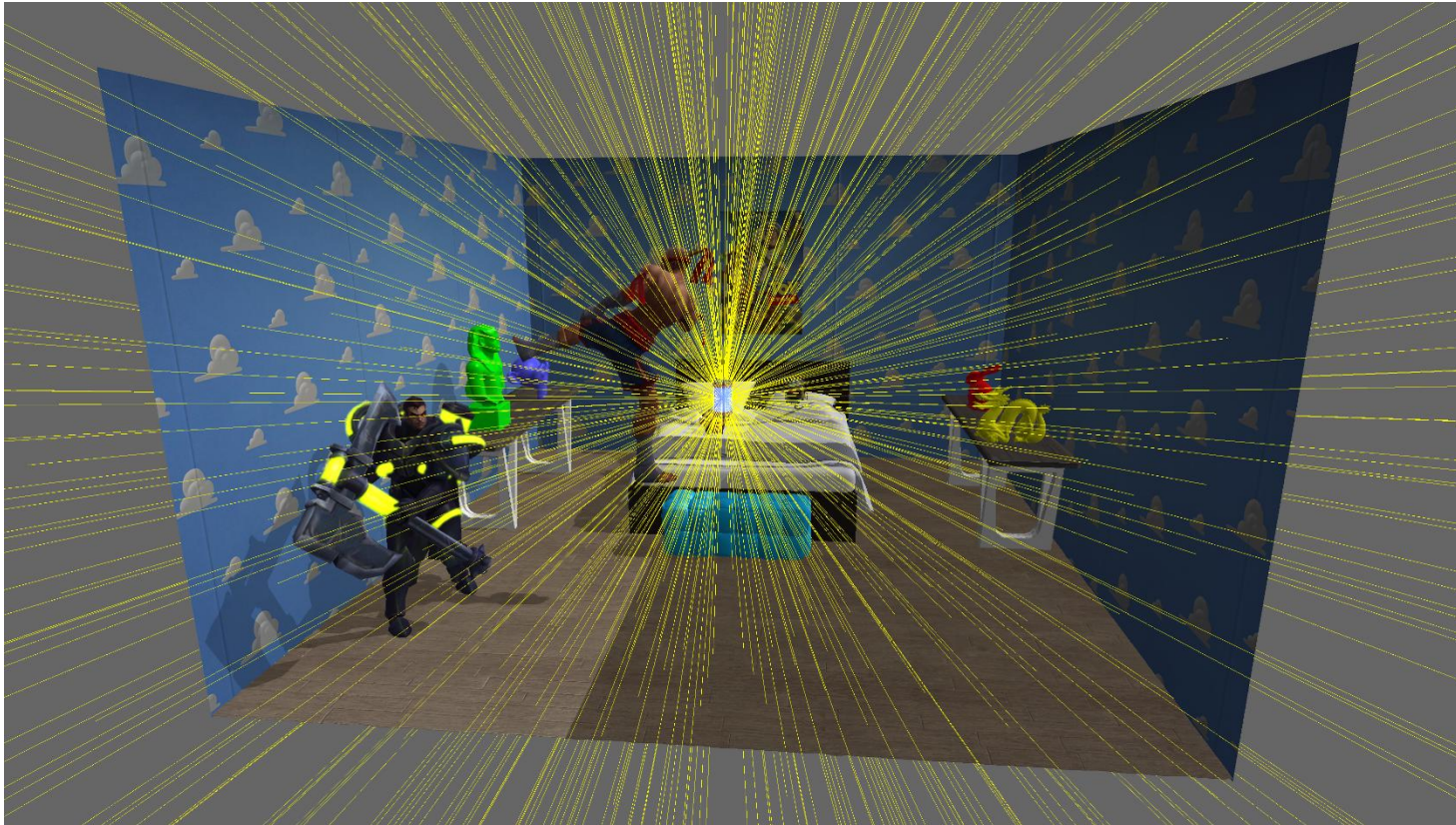
Fast Ray/Axis-Aligned Bounding Box Overlap Tests using Ray Slopes. Eisemann, Martin; Marcus Magnor; Thorsten Grosch; Stefan Müller

### 3 Algoritmo de intersección AABB-AABB

Descrito en Real-Time Collision Detection; Ericson, Christer.



Instanciación de rayos en función de  $rays_x$  y  $rays_y$



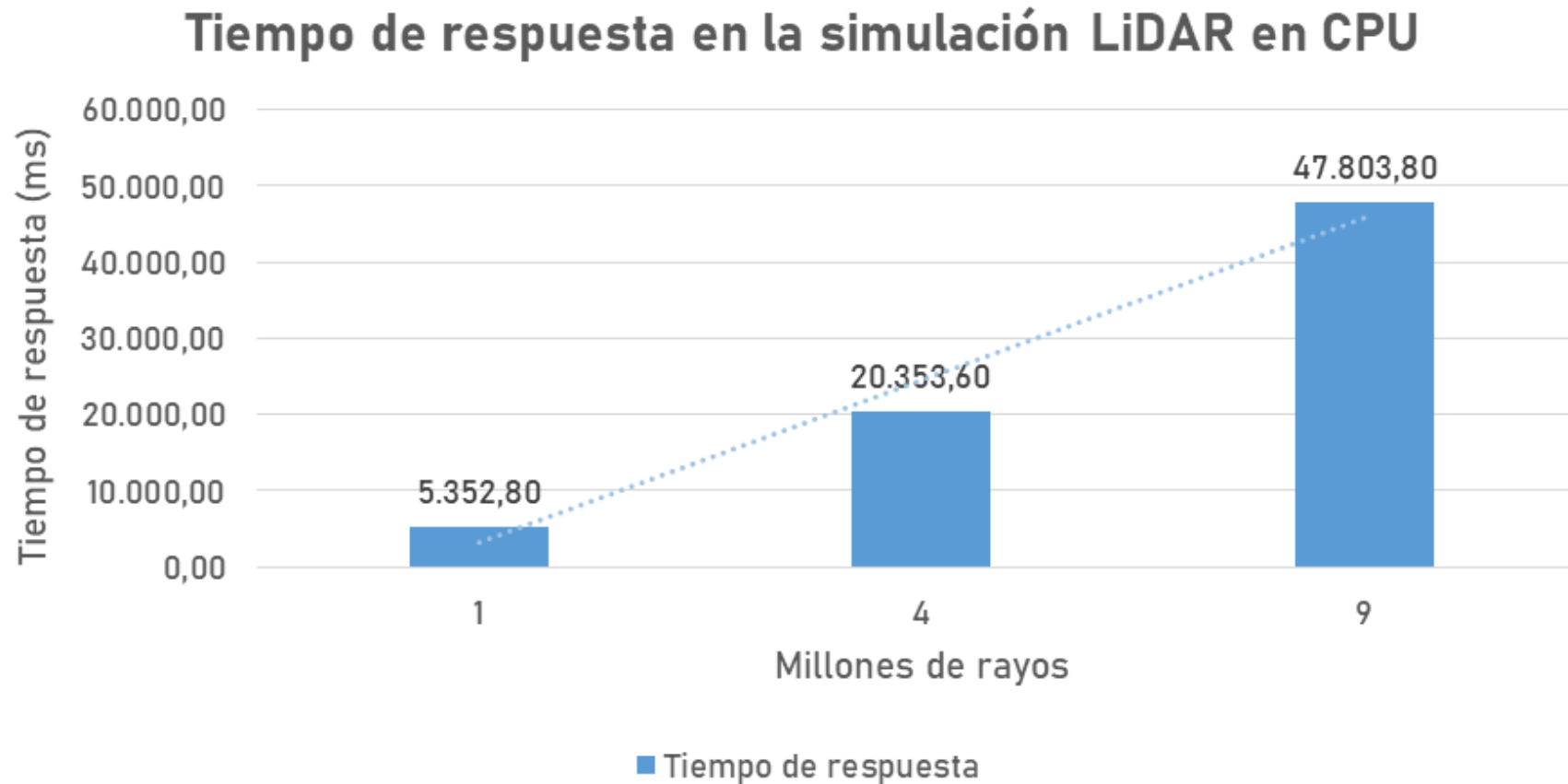




Rayos emitidos  
**250.000**

Offset  
 **$\pi/9$**

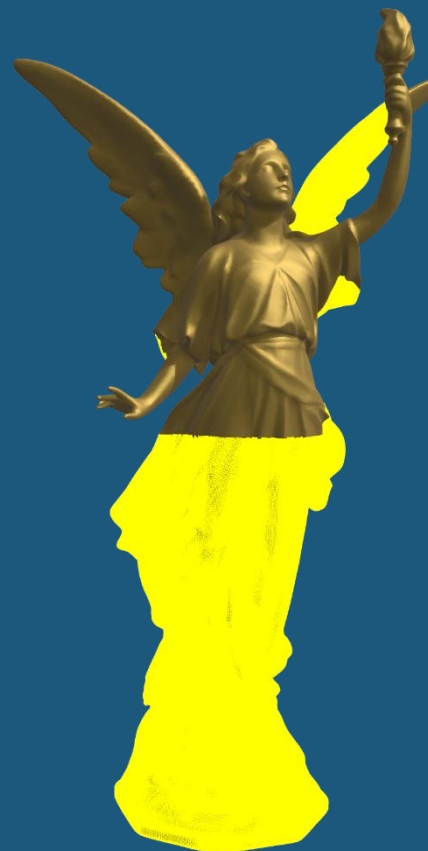
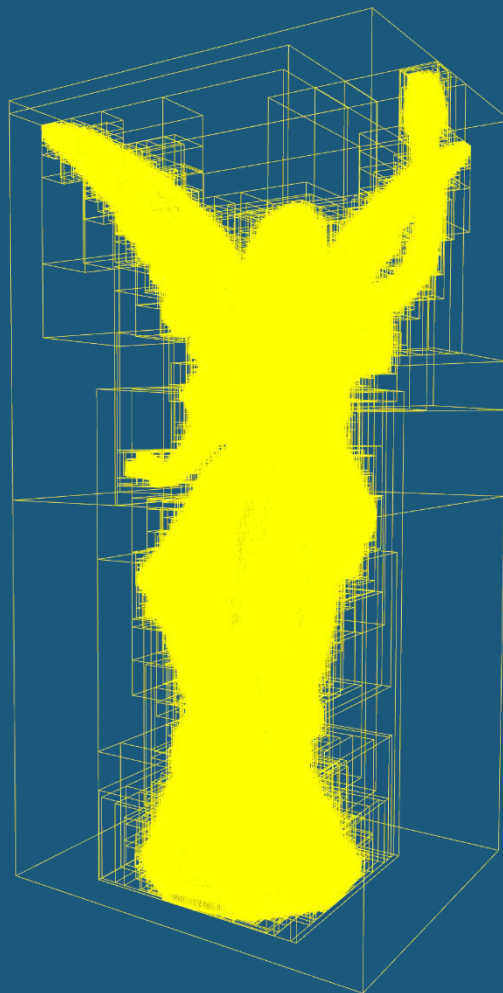
## Tiempo de respuesta de simulación para diferentes millones de rayos





## TERCERA ITERACIÓN

Transición de algoritmos en CPU hacia GPU (unidad de procesamiento gráfico) una vez se observa el elevado tiempo de respuesta de la versión anterior. Desarrollo de algoritmos en GPU para la generación de la geometría y la topología de los modelos 3D.



# ACELERACIÓN DE CARGA DE MODELOS

28

*Datos obtenidos de una escena inicial de 300.973 vértices y 587.351 triángulos y una escena final de 3.338.346 vértices y 6.660.500 triángulos.*

## Carga de modelos OBJ

Ficheros con estructura muy limitada, pero la implementación es mínima

Tiempo de lectura de ficheros (ms)

1.695,22

Tiempo de lectura de ficheros + cálculos (ms)

1.741,99

## Almacenamiento de ficheros binarios

Es necesaria al menos una carga con el método previo

Tiempo de lectura de ficheros (ms)

67,49

Tiempo de lectura de ficheros (ms)

1.059,705



## 1 Transformación de vértices

A partir de matriz de modelado y displacement mapping

## 2 Cálculo de tangentes

- 1) Contribución de triángulos en vértices
- 2) Ortogonalización de Grand-Schmidt (base ortogonal desde tres vectores linealmente independientes)

## 3 Composición de topología

Construcción de datos derivados de la topología cargada

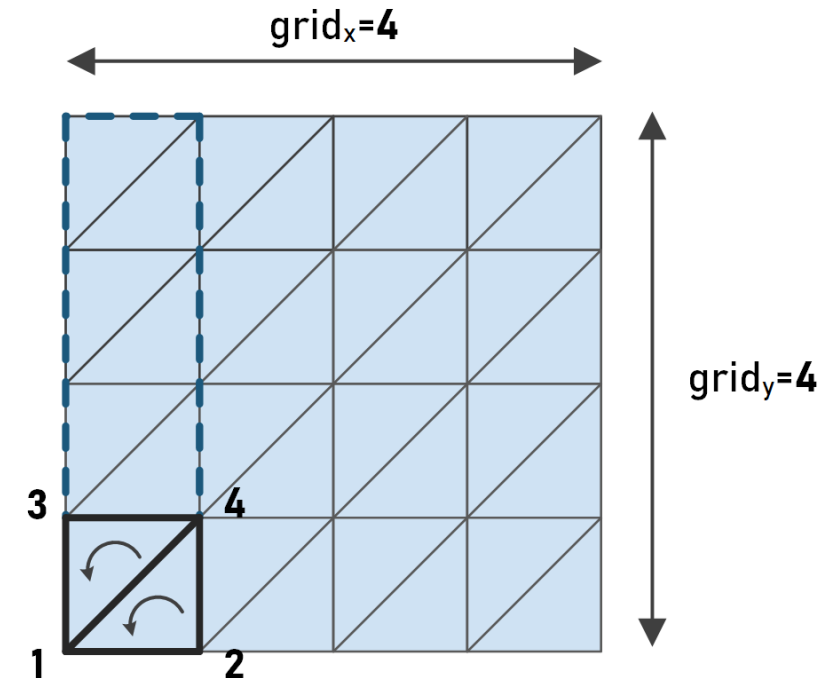
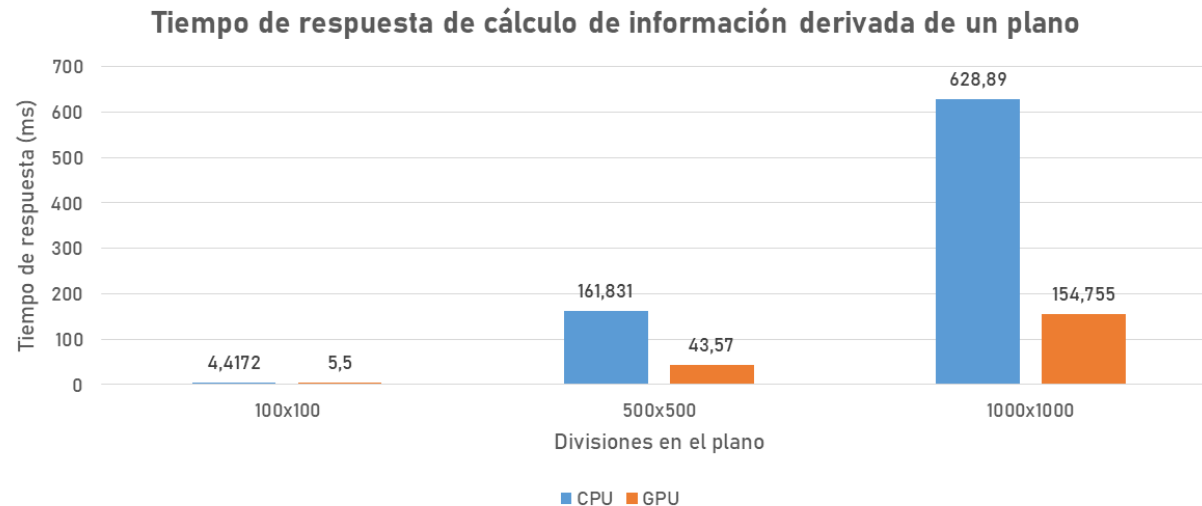


## 1 Generación de geometría y topología (parcialmente)

Información asociada a vértices e índices de la malla (rendering)

## 2 Definición de topología extendida

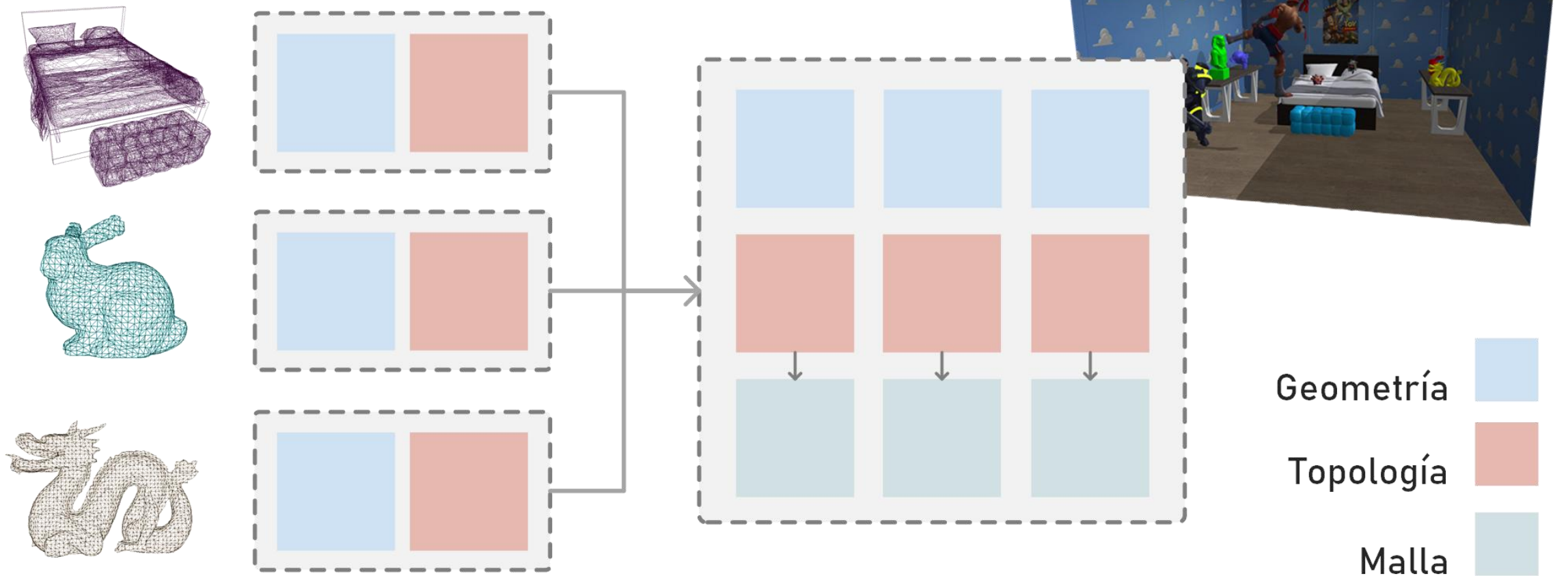
Información de topología para compute shaders



# CONSTRUCCIÓN DE BVH

31

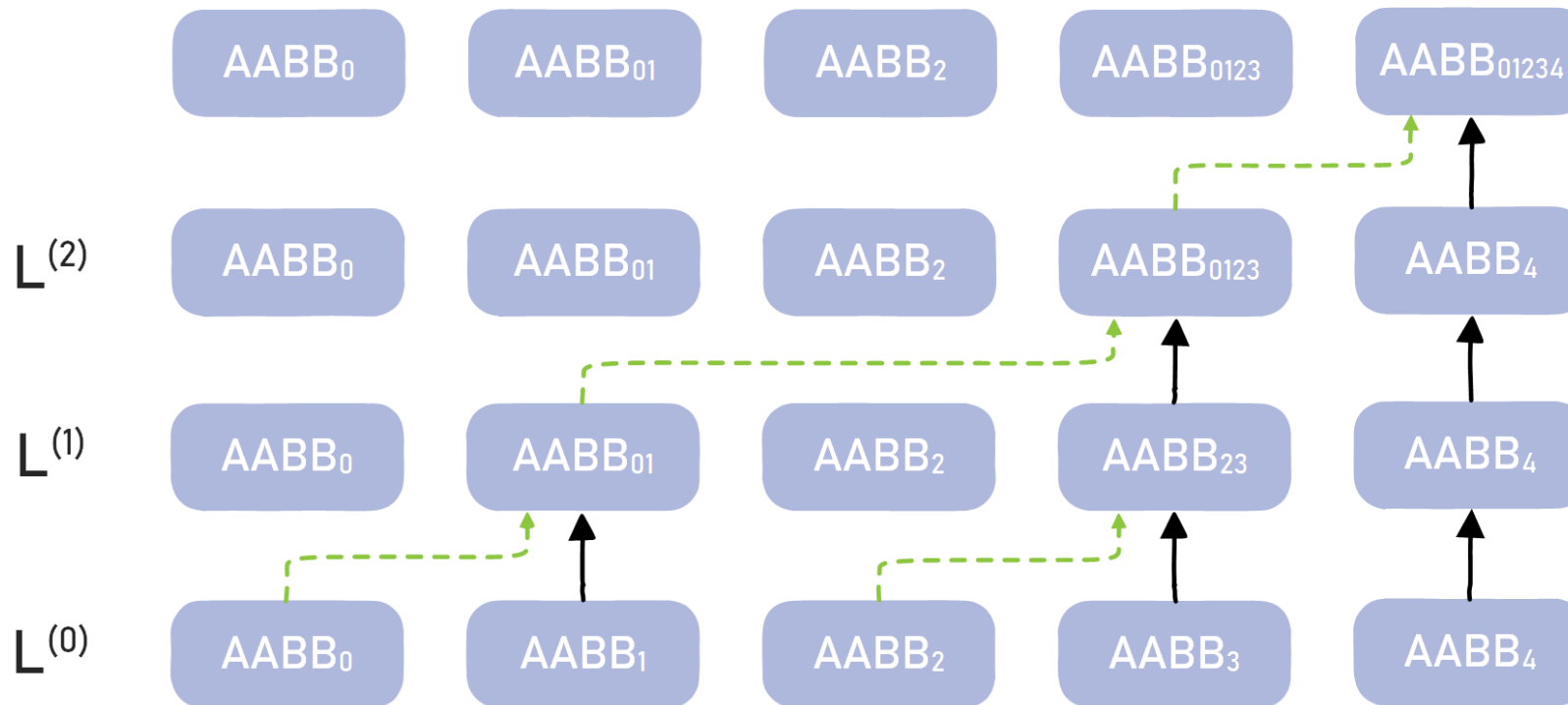
## Preparación de construcción



# CONSTRUCCIÓN DE BVH

32

## Cálculo de caja envolvente de la escena



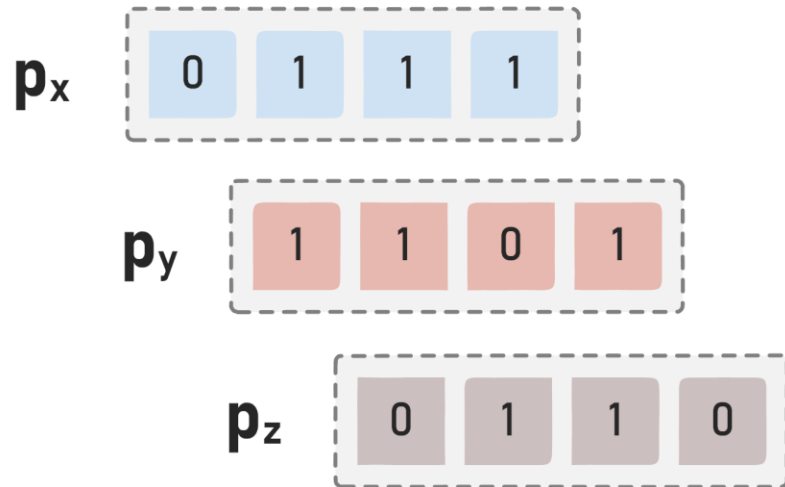
*Escena evaluada*

**3.350.433 vértices y  
6.683.902 triángulos**

**Tiempo en CPU**  
37,93 ms

**Tiempo en GPU (kernel)**  
11,63 ms

## Cálculo de códigos Morton



**Morton code**

Tiempo en CPU

118,87 ms

Tiempo en GPU

29,03 ms

*Escena evaluada*  
**3.350.433 vértices y**  
**6.683.902 triángulos**

$$point_{expanded_i} = (point_{expanded_i} * 0x00010001u) \& 0xFF0000FFu$$

$$point_{expanded_i} = (point_{expanded_i} * 0x00000101u) \& 0x0F00F00Fu$$

$$point_{expanded_i} = (point_{expanded_i} * 0x00000011u) \& 0xC30C30C3u$$

$$point_{expanded_i} = (point_{expanded_i} * 0x00000005u) \& 0x49249249u$$

## Ordenación de códigos Morton: Radix Sort

### 1 Comprobación del bit $i$ de cada código

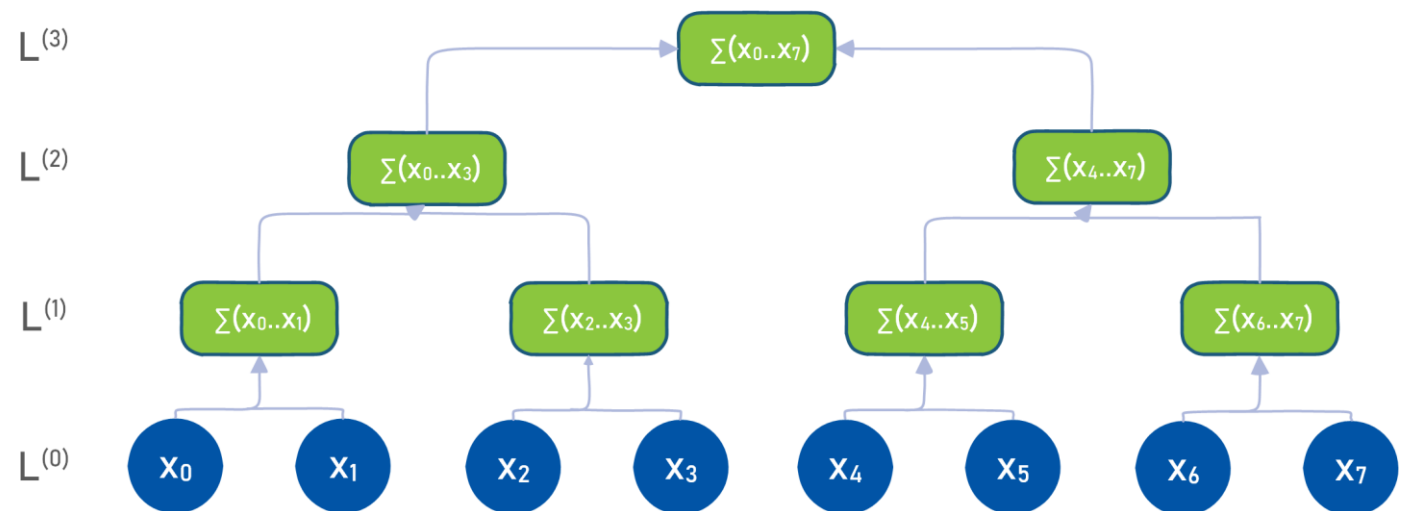
Dados 30 bits, serán necesarias 30 iteraciones

### 2 Prefix scan #1. REDUCE

Conteo de códigos con el bit actual a cero antes de una posición  $i$

### 3 Prefix scan #2

Modificación de sumatoria en  $N - 1$ . Pasar a ser 0.

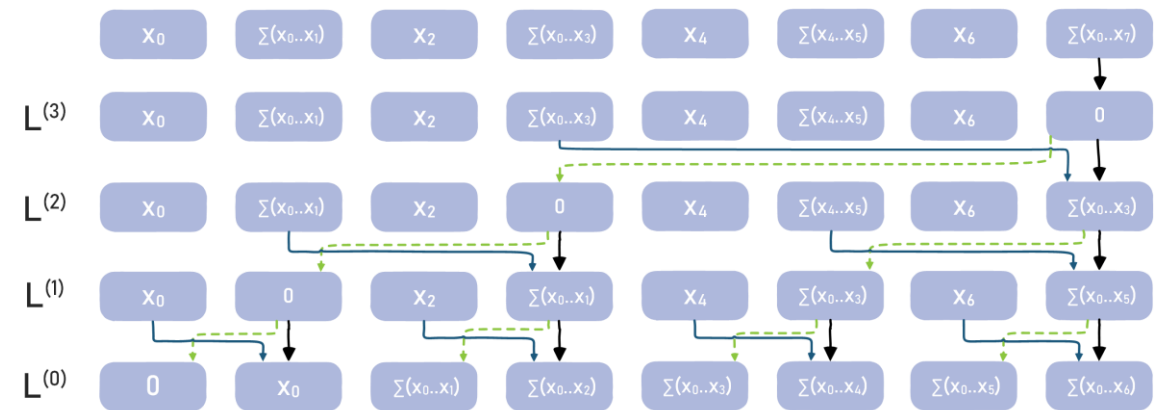




## Ordenación de códigos Morton: Radix Sort

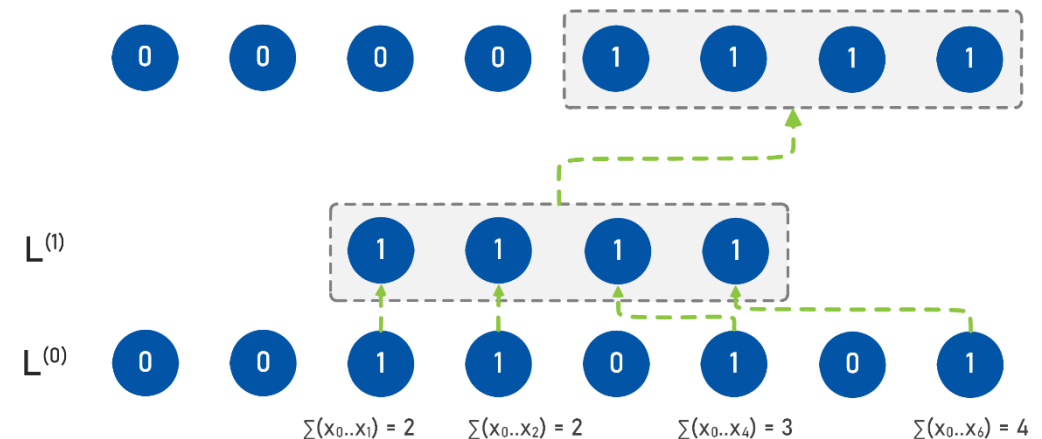
### 4 Prefix scan #3. DOWN-SWEEP

Cálculo final de conteo

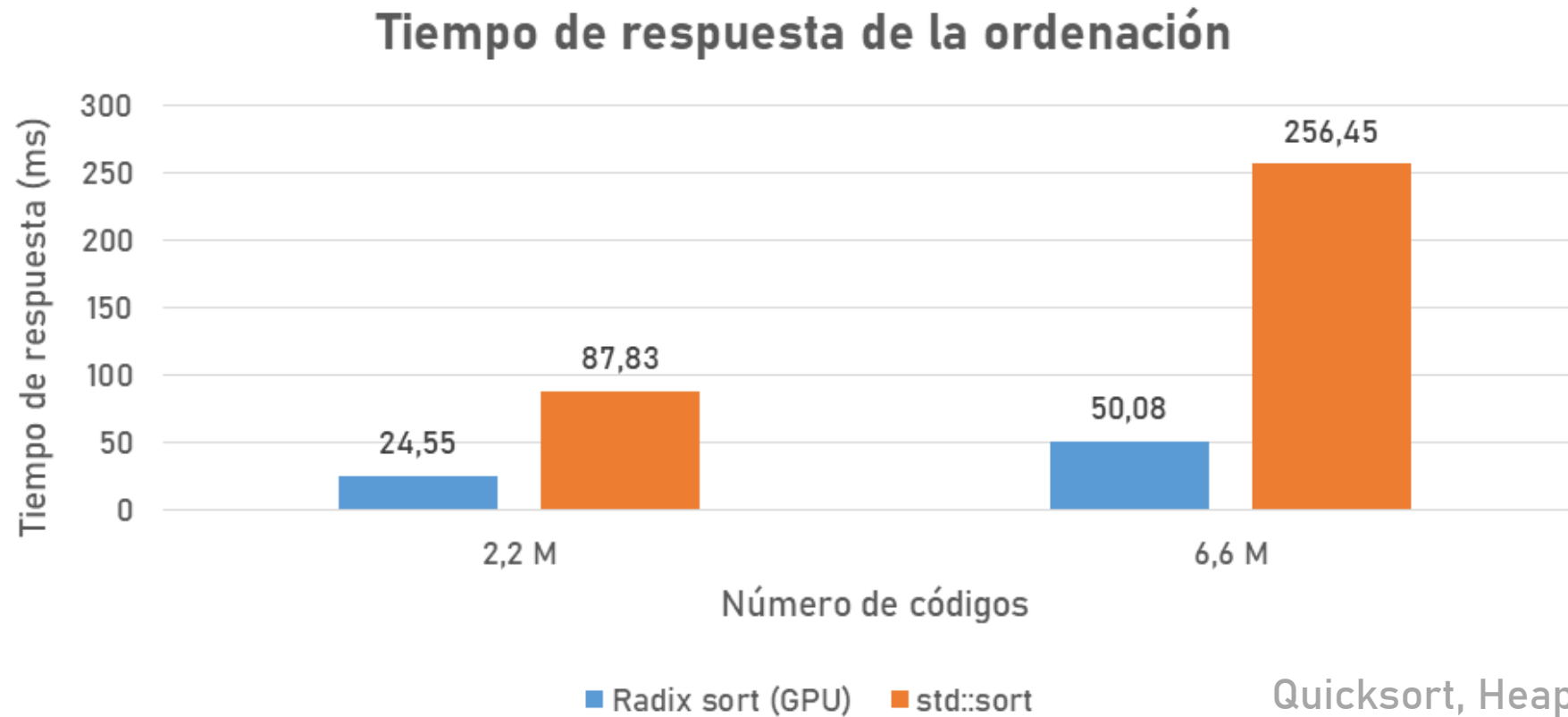


### 5 Modificación de posiciones

$index - nBits[index] + nBits[N - 1]$   
 $+ (pBits[N - 1] \oplus 1)$



## Ordenación de códigos Morton: Radix Sort



**std::sort: IntroSort**

Quicksort, Heapsort e Insertion Sort

## Núcleo de construcción de BVH

### 1 Búsqueda del mejor vecino en un radio

Heurística SAH

### 2 Combinación de clústeres

¿Soy el mejor vecino de mi mejor vecino? Sólo uno debe actualizarse

### 3 Prefix scan

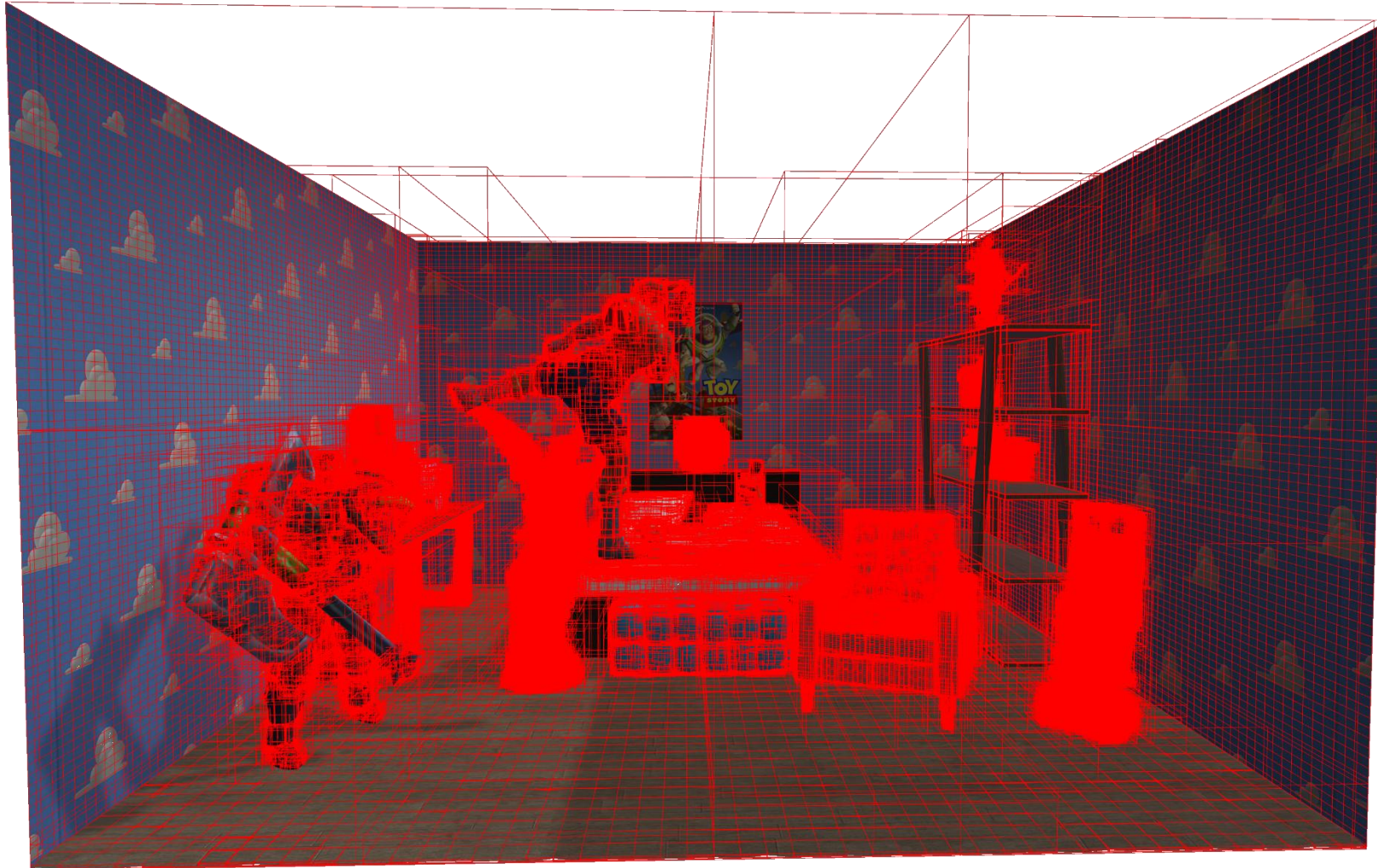
Recuento de nodos válidos

### 4 Actualización de buffer de nodos válidos

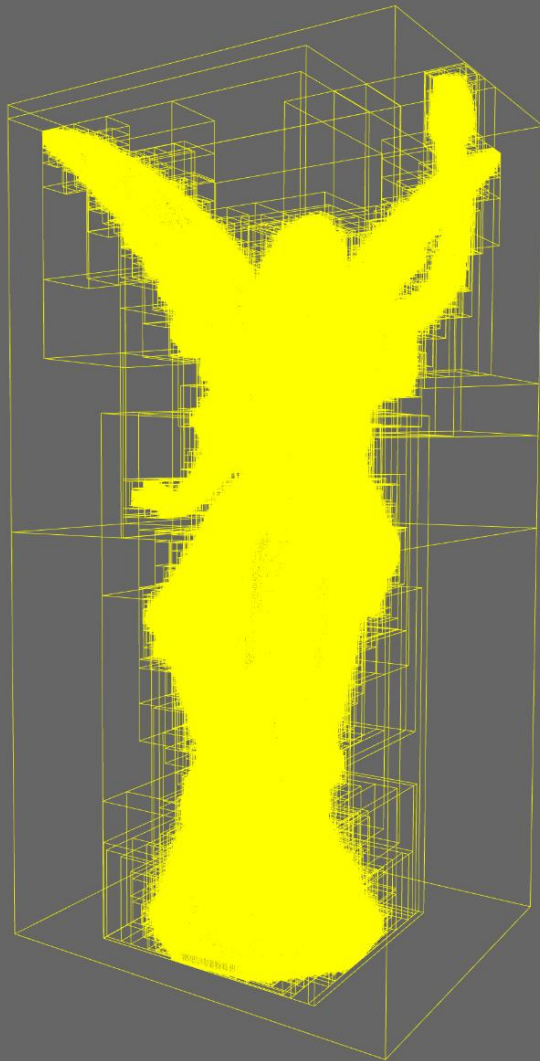
Actualización y compactación de nodos restantes

### 5 Cálculo de nodos restantes

Se realiza en GPU por eficiencia

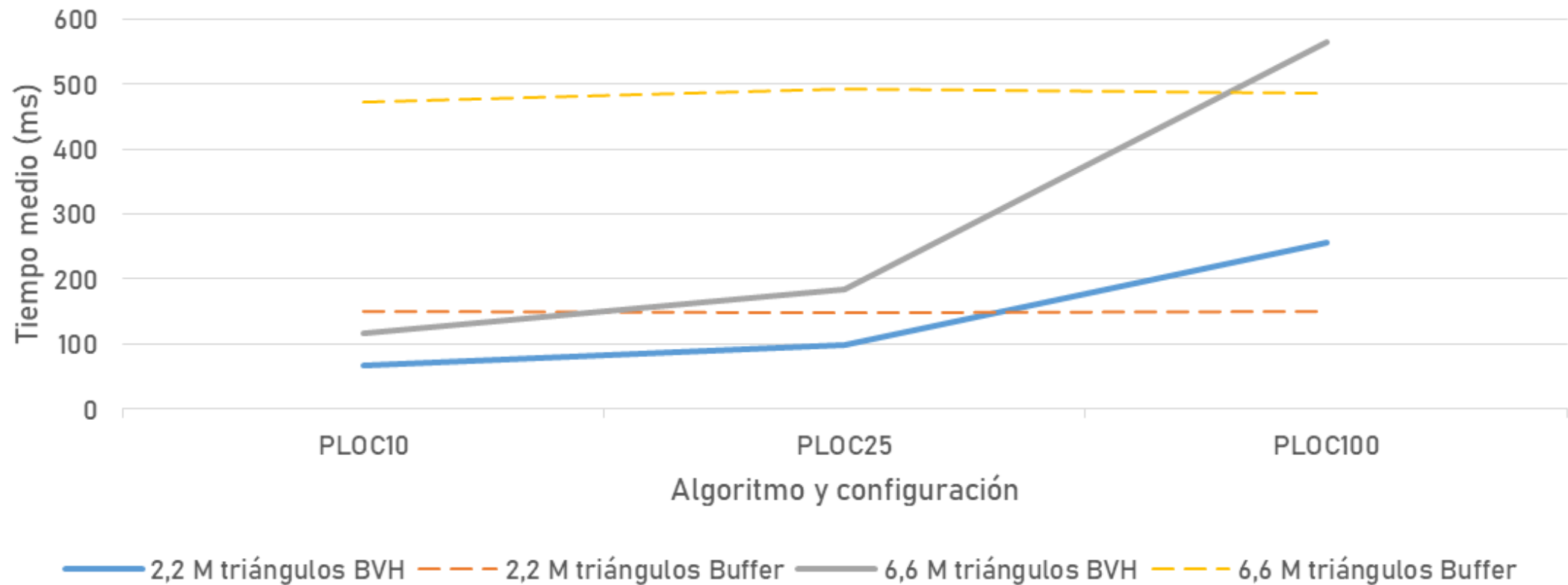






## Núcleo de construcción de BVH

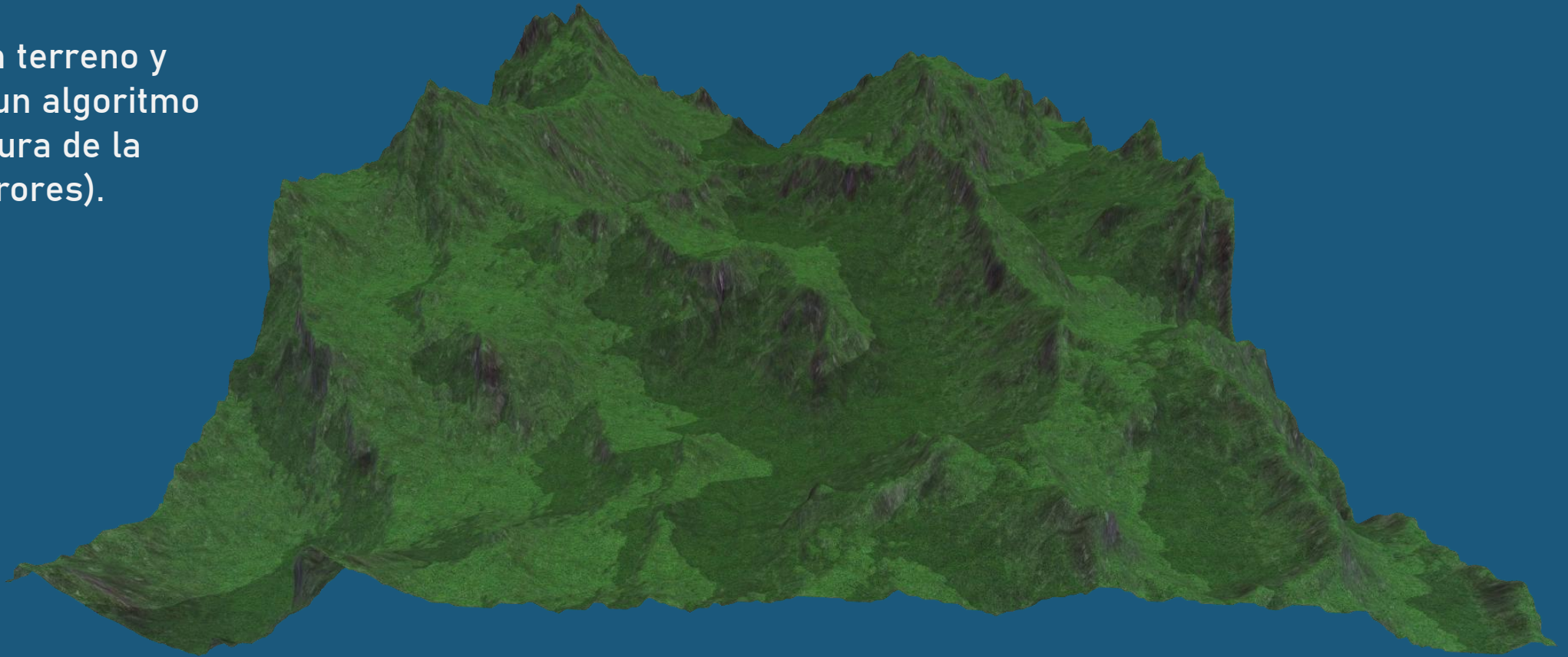
Tiempo medio de respuesta en construcción de un BVH



---

## CUARTA ITERACIÓN

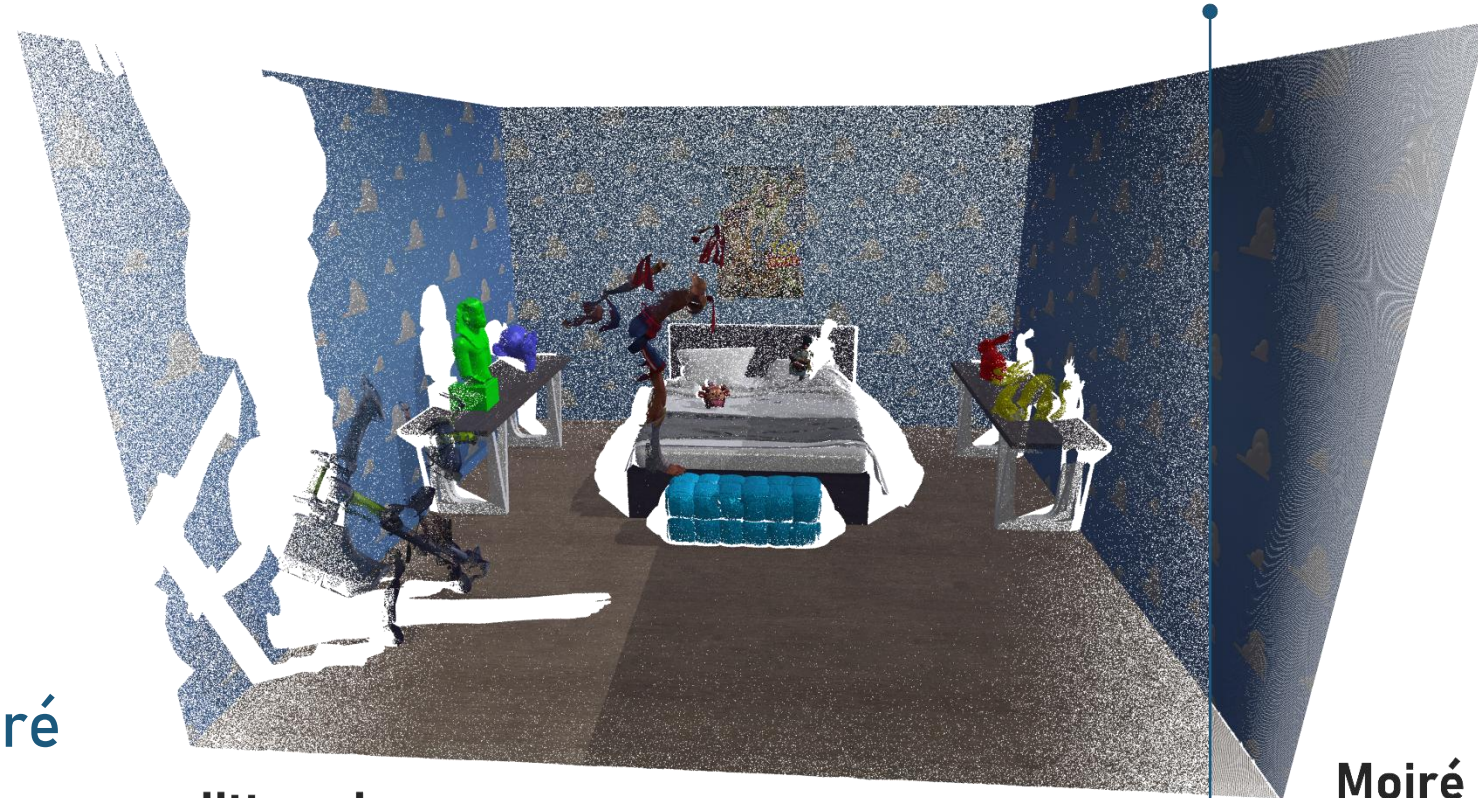
Creación de un terreno y desarrollo de un algoritmo básico de captura de la escena (sin errores).





Distribución uniforme  
Fix de patrón de Moiré

**Jittered rays**



**Moiré pattern**



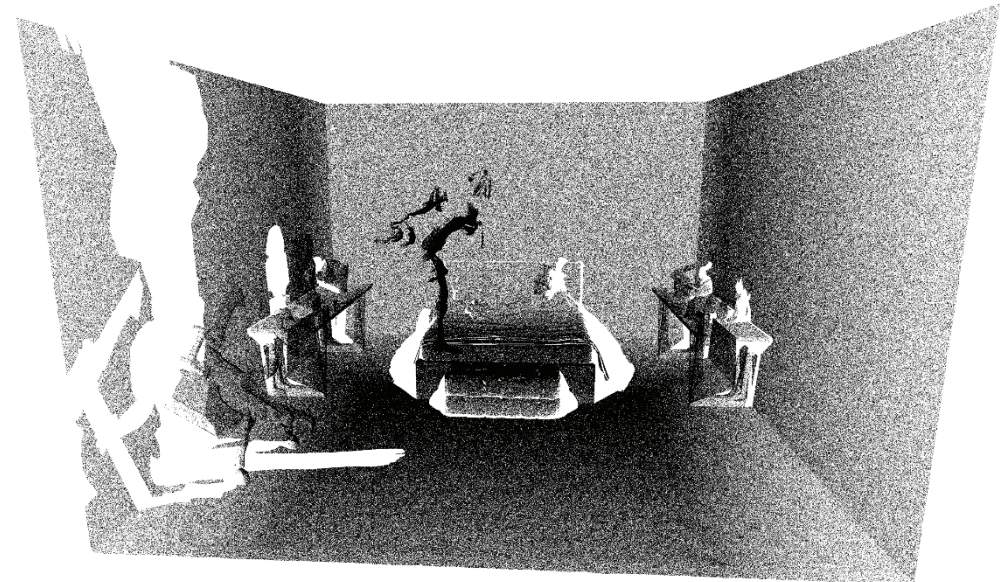
# SIMULACIÓN BÁSICA EN GPU

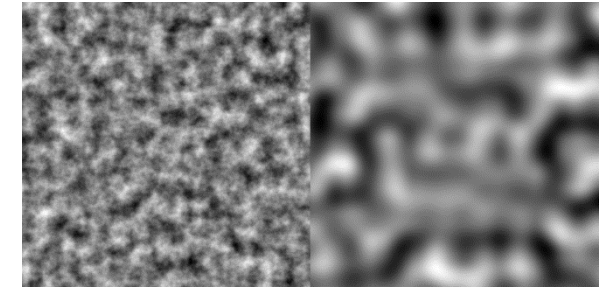
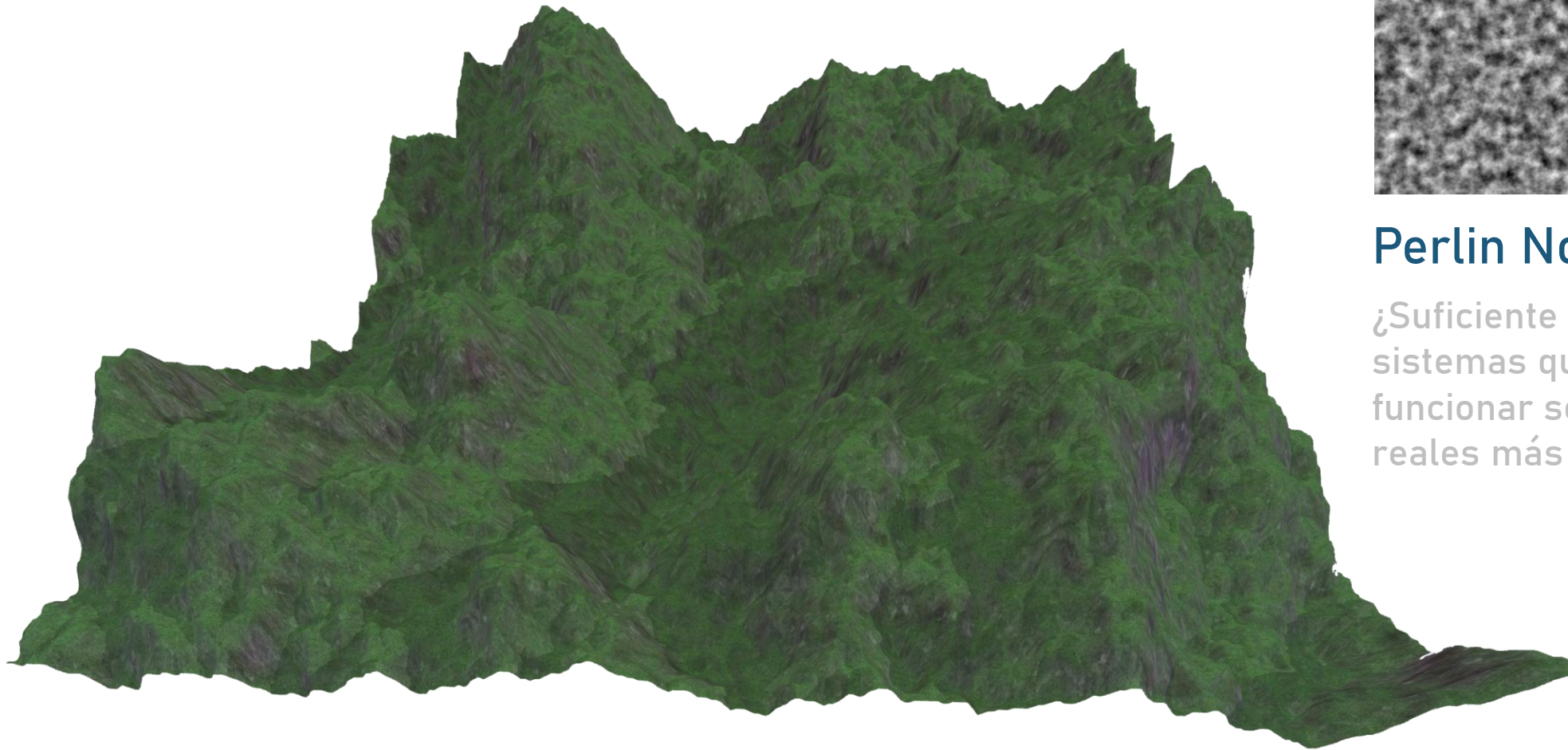
## Algoritmo de intersección rayo-AABB

A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering. Majercik, Alexander; Crassin, Cyril; Shirley, Peter; McGuire, Morgan

## Algoritmo de intersección rayo-triángulo

Fast, minimum storage ray-triangle intersection. Möller, Tomas; Trumbore, Ben





## Perlin Noise

¿Suficiente realismo para sistemas que deben funcionar sobre datos reales más tarde?

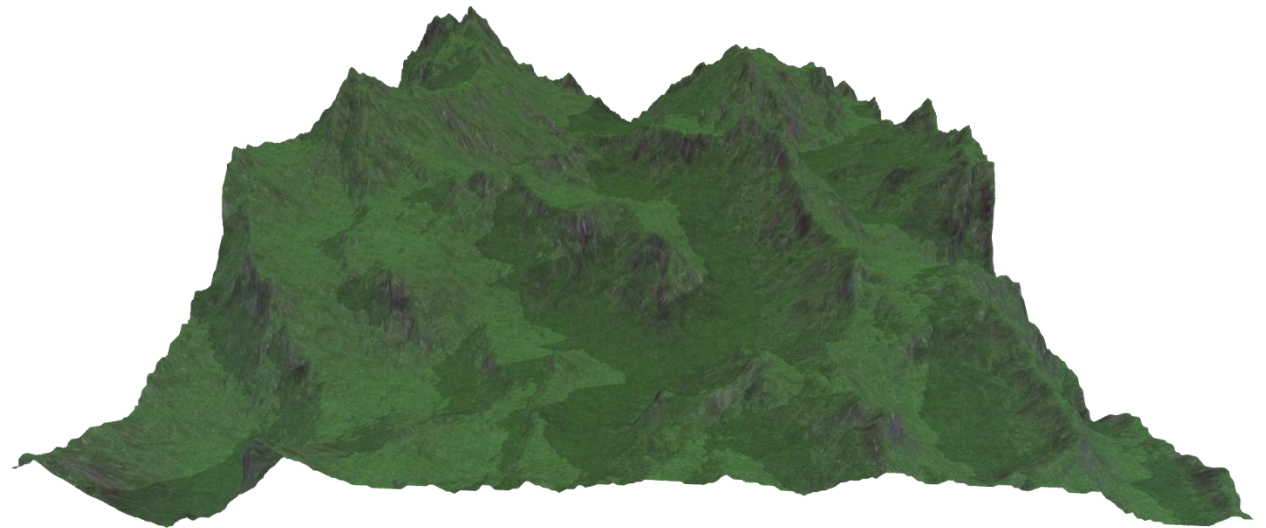
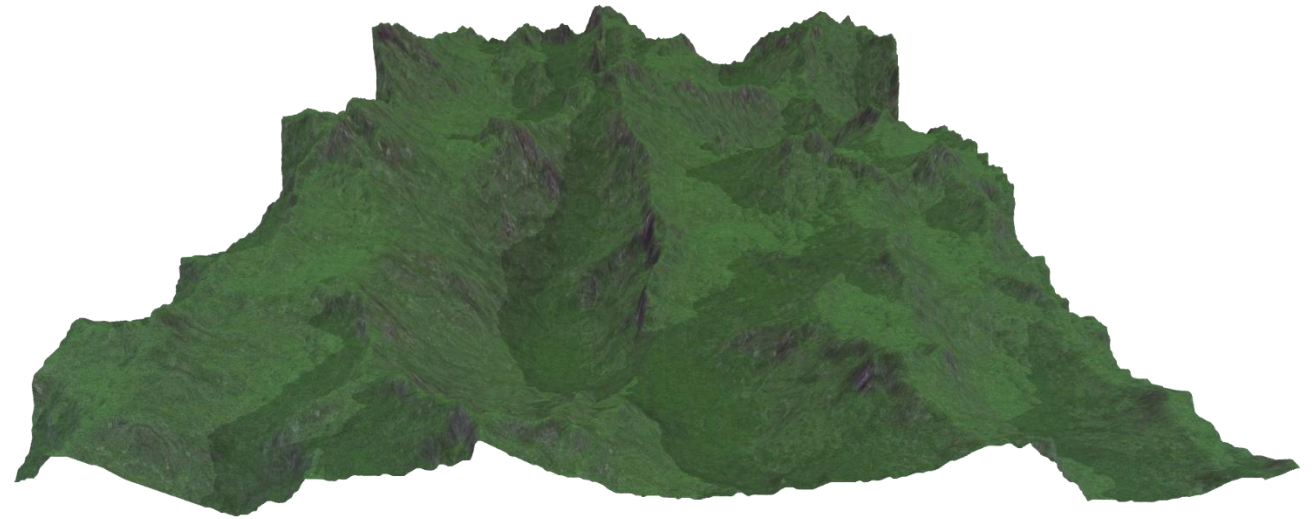


Erosión hidráulica

Erosión fluvial

Erosión térmica

Métodos híbridos



Implementation of a method for hydraulic erosion. Bayer, Hans Theobald

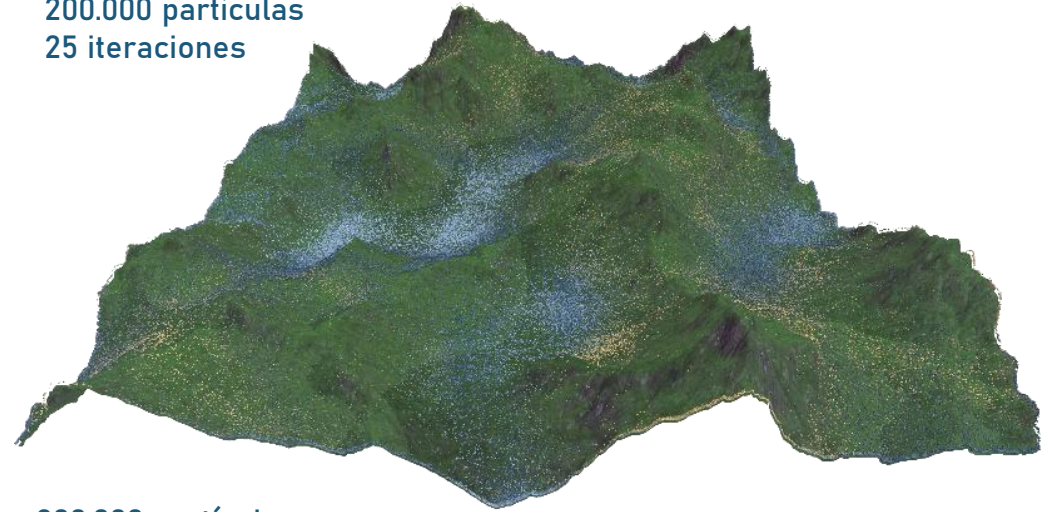
# GENERACIÓN DE ESCENARIO ALTERNATIVO

46

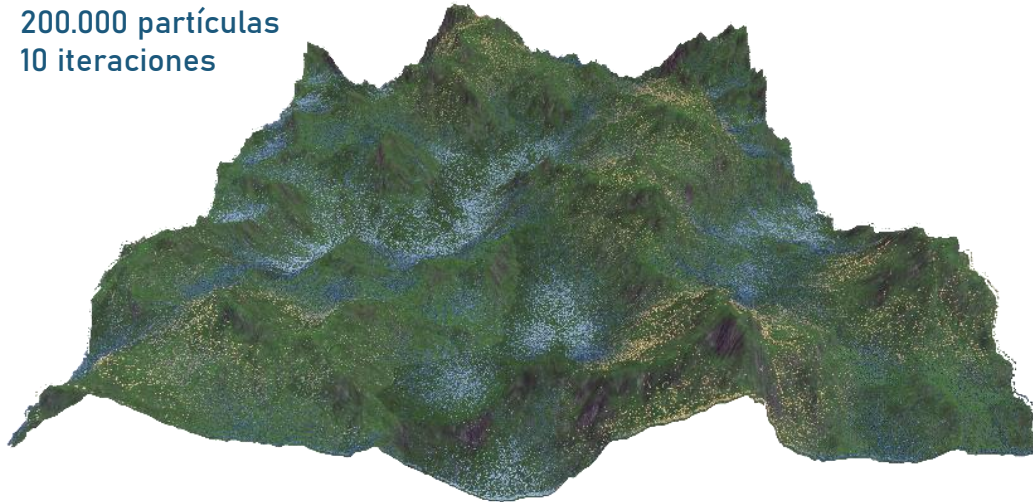
200.000 partículas  
1 iteración



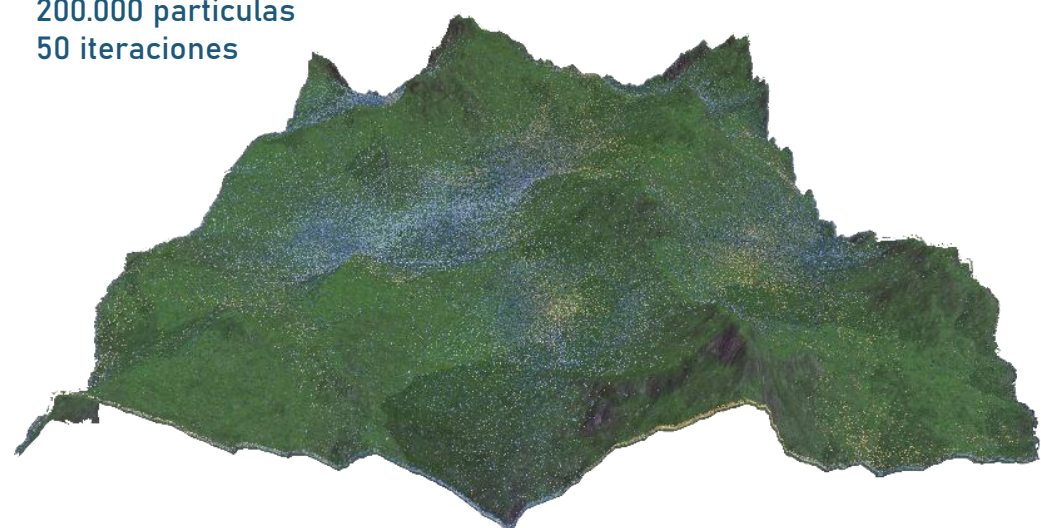
200.000 partículas  
25 iteraciones



200.000 partículas  
10 iteraciones

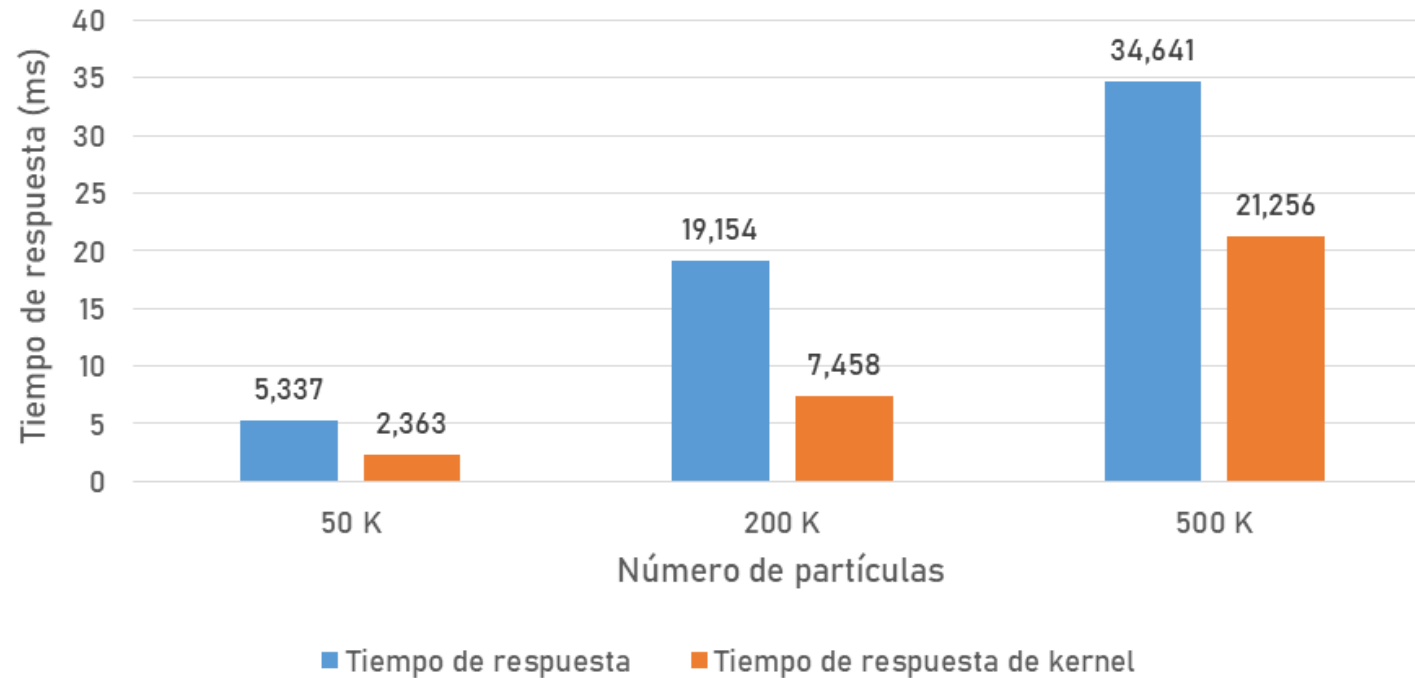


200.000 partículas  
50 iteraciones





## Tiempo de respuesta de erosión



## Tiempo de referencia (artículo)

~30 segundos para un tamaño de mapa 256x256

## 500.000 partículas; 64 iteraciones

48,684 ms (global), 38,665 ms (kernel)

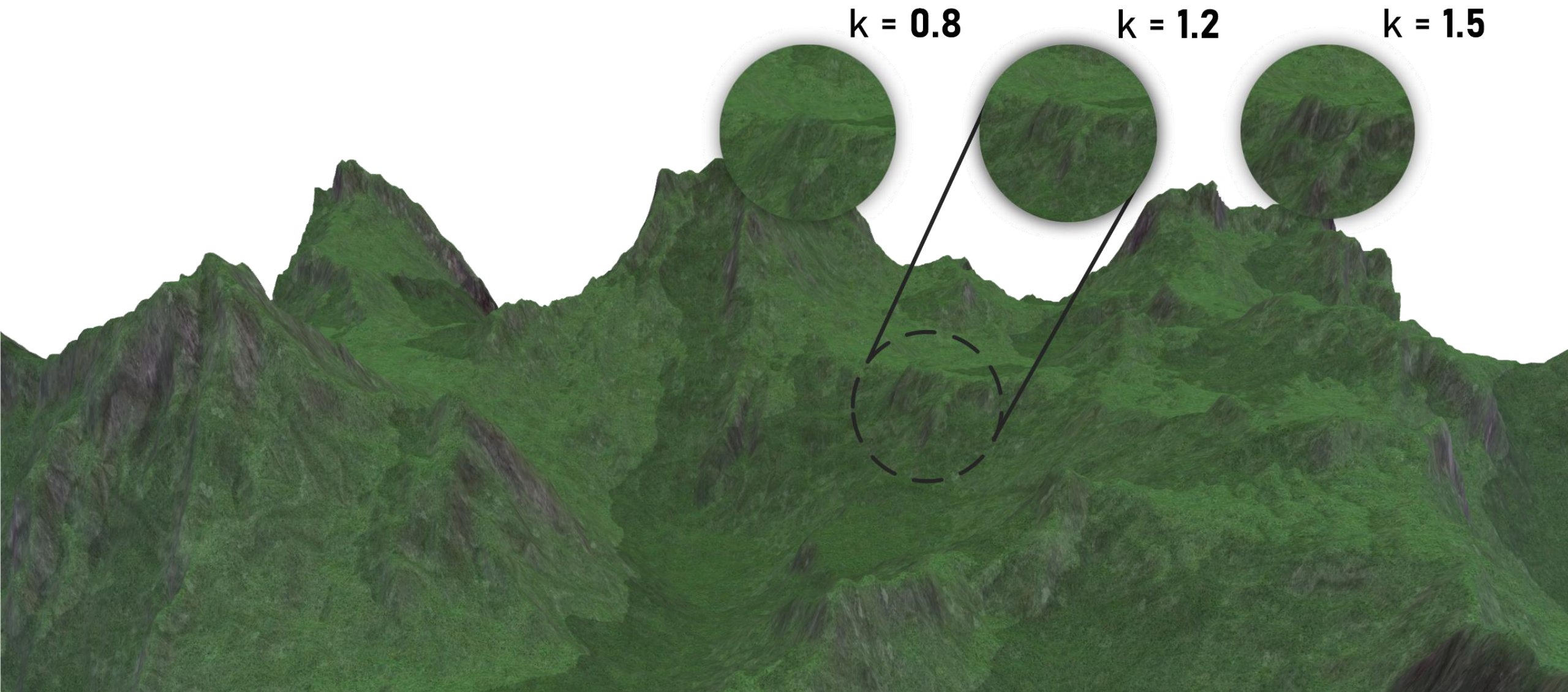
*Tamaño de mapa evaluado:* 320x320

*Número de iteraciones:* 30

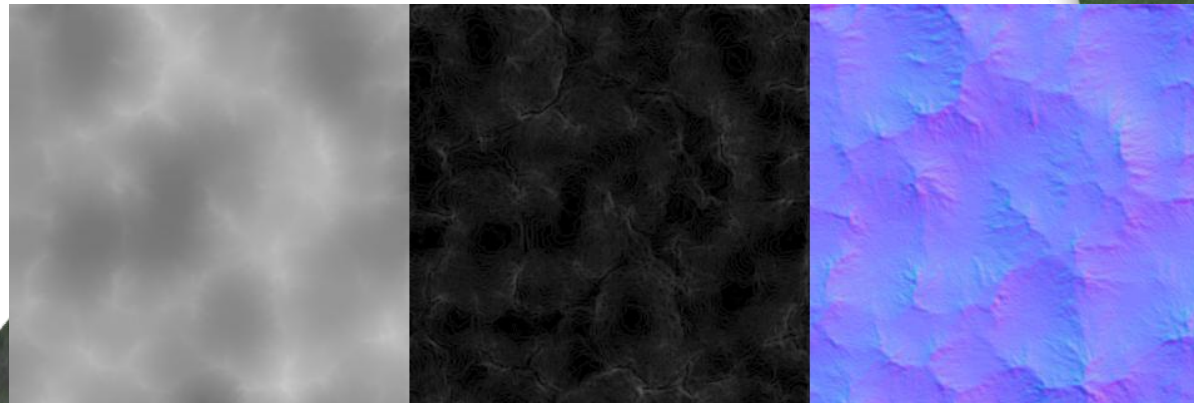
# GENERACIÓN DE ESCENARIO ALTERNATIVO

---

48







Extensión de operador de Sobel

$k = 0.8$

$k = 1.2$

$k = 1.5$



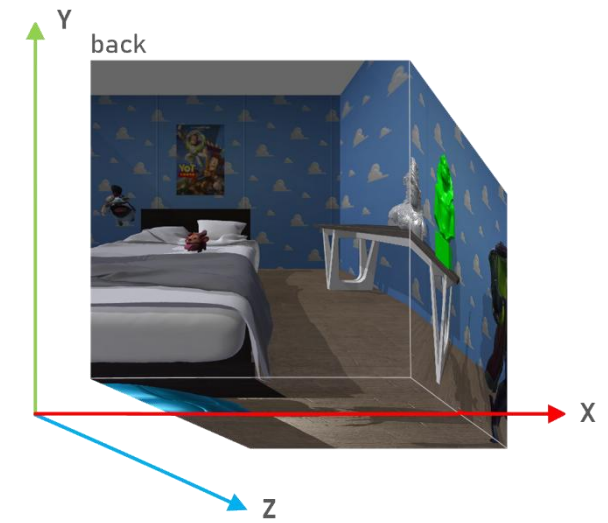
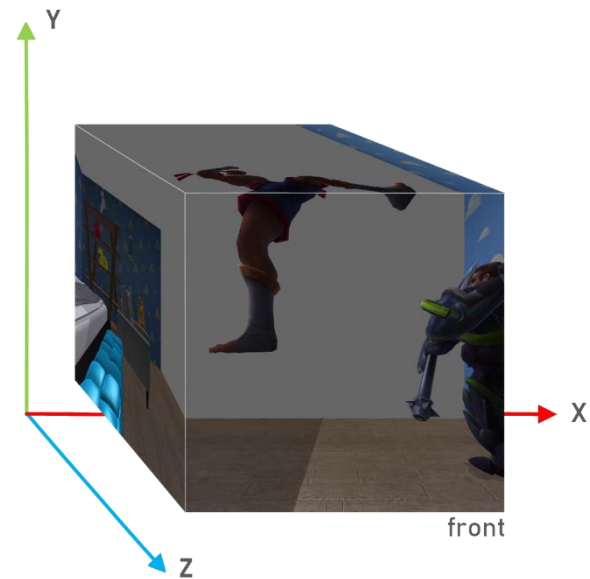
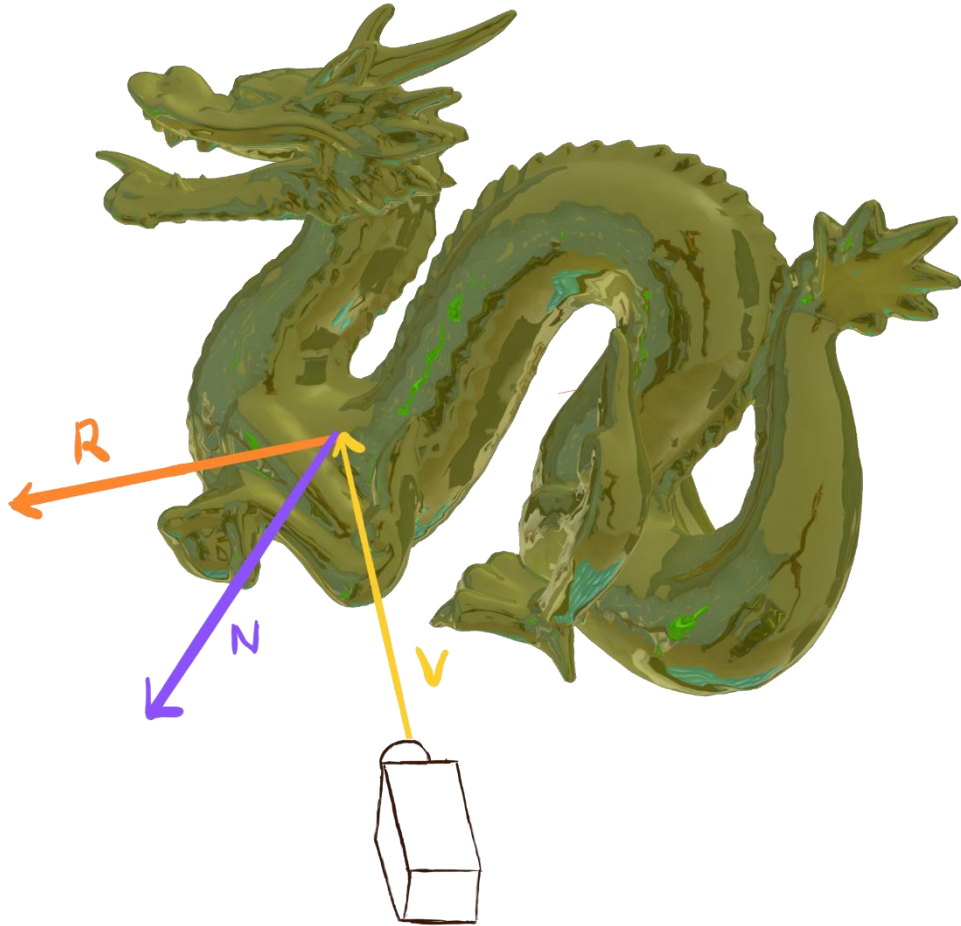
---

## QUINTA ITERACIÓN

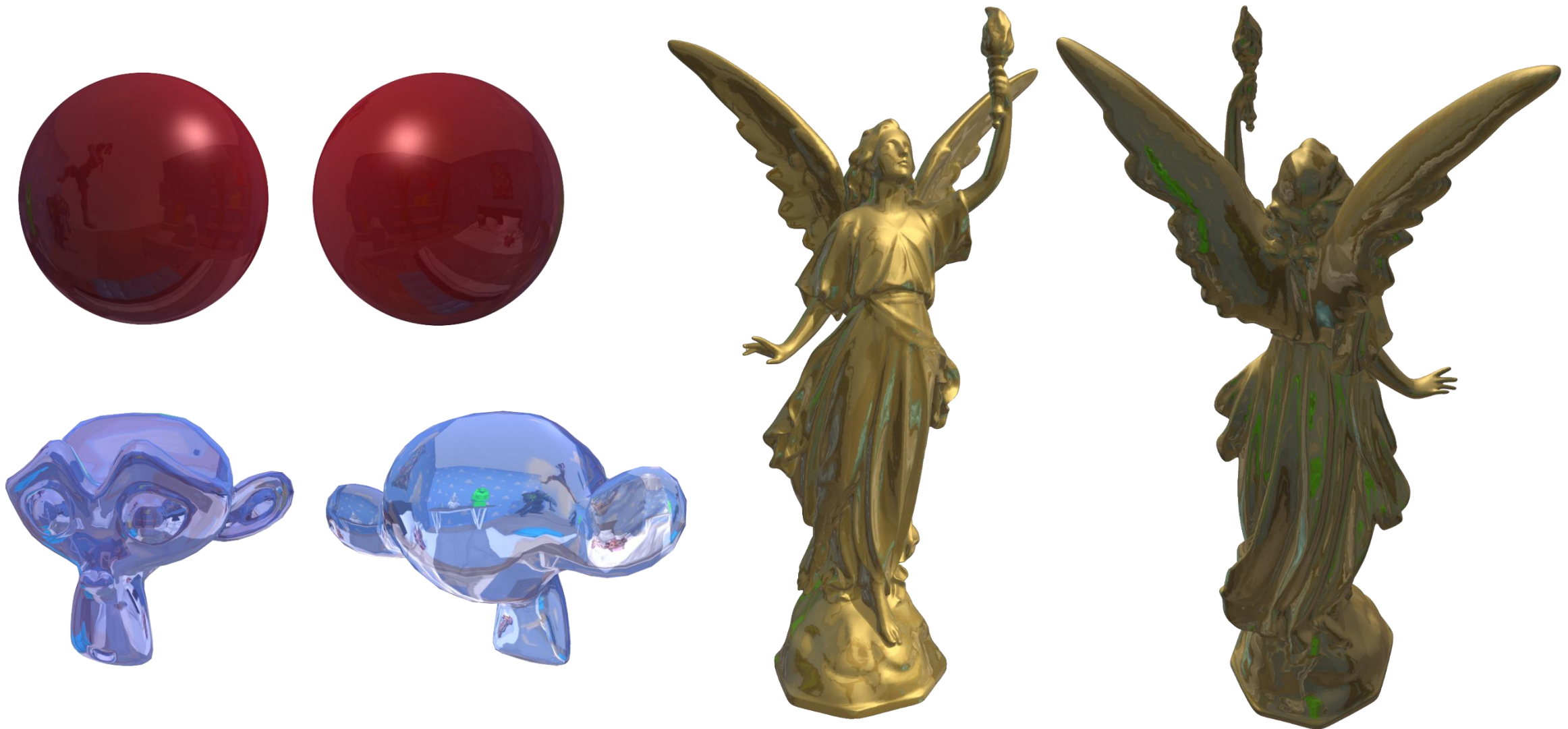
Finalización de la aplicación gráfica: reflejos y refracciones en superficies, lagos, etc. Se crea un escenario alternativo con animaciones.



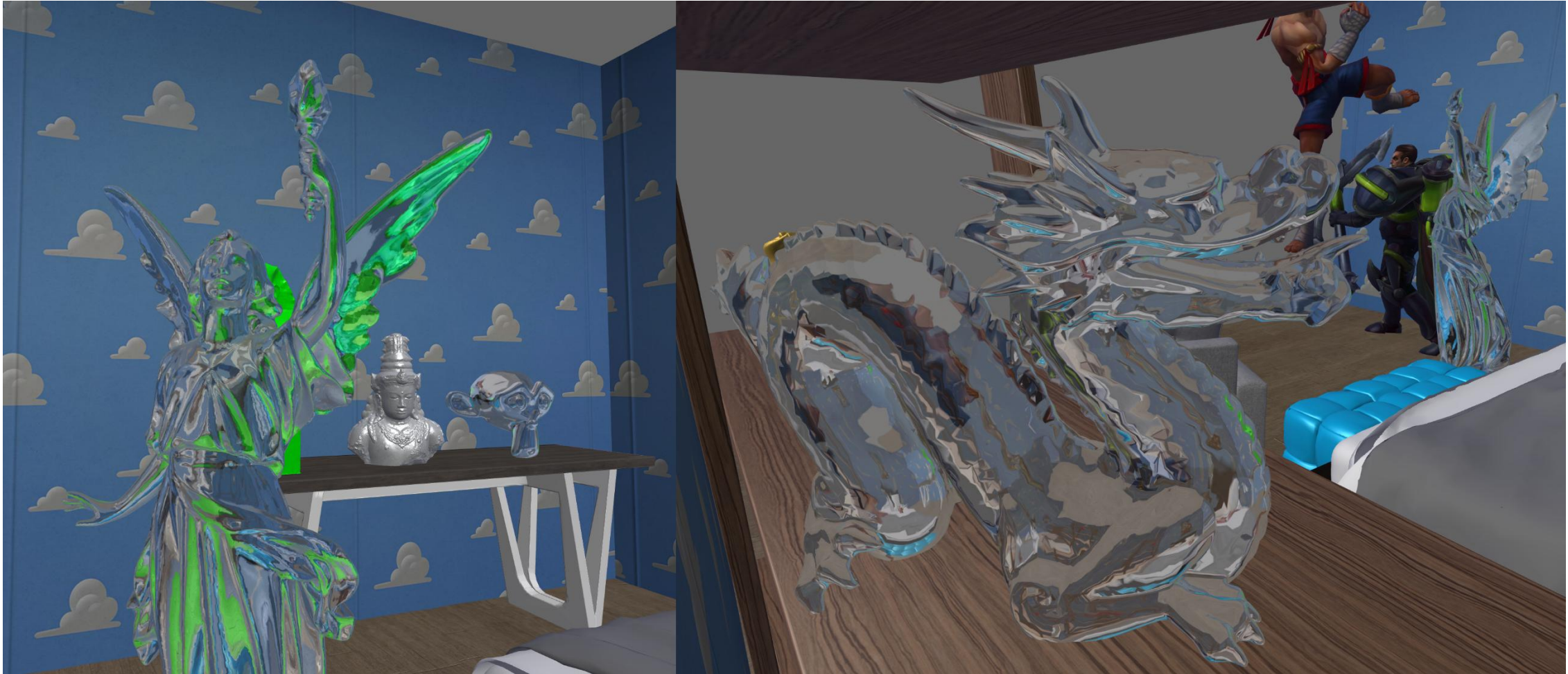




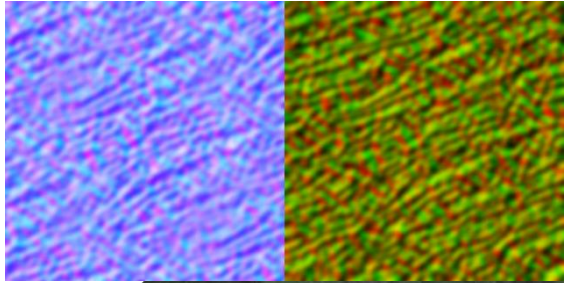




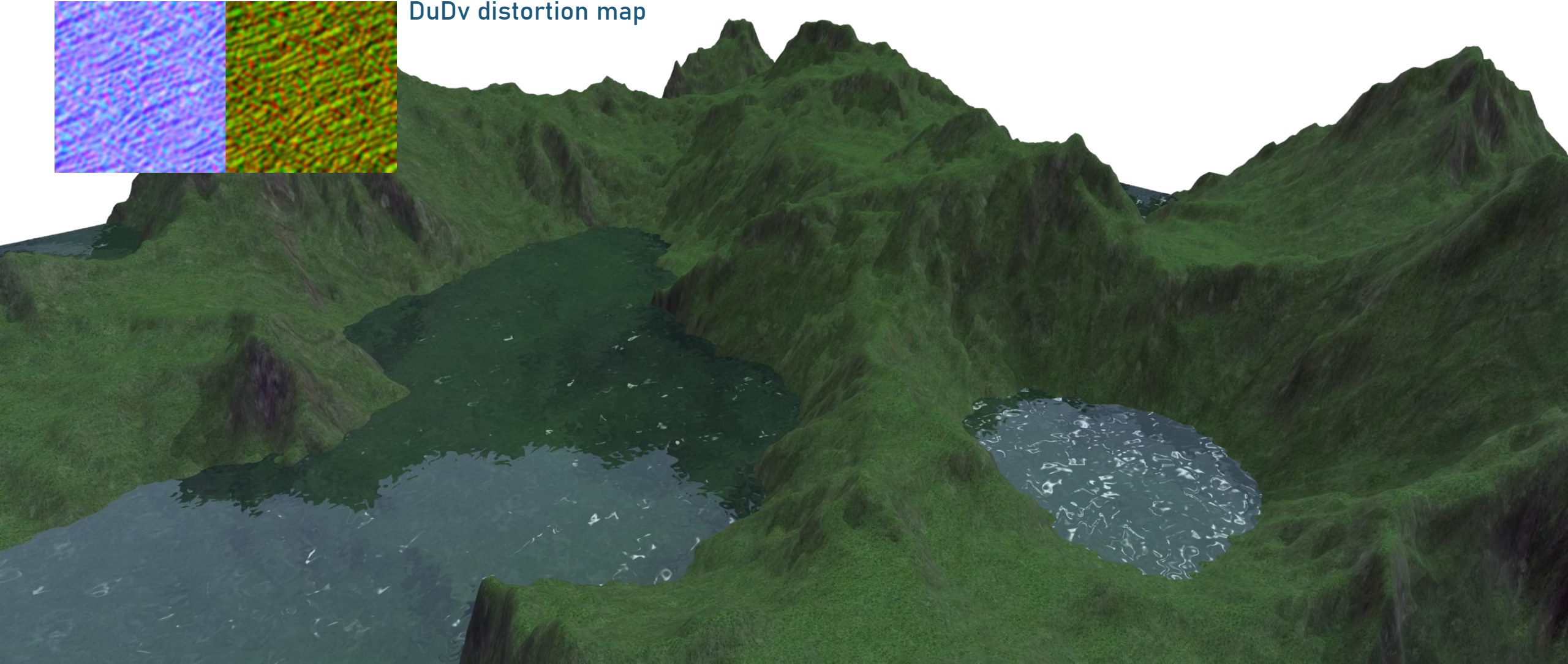


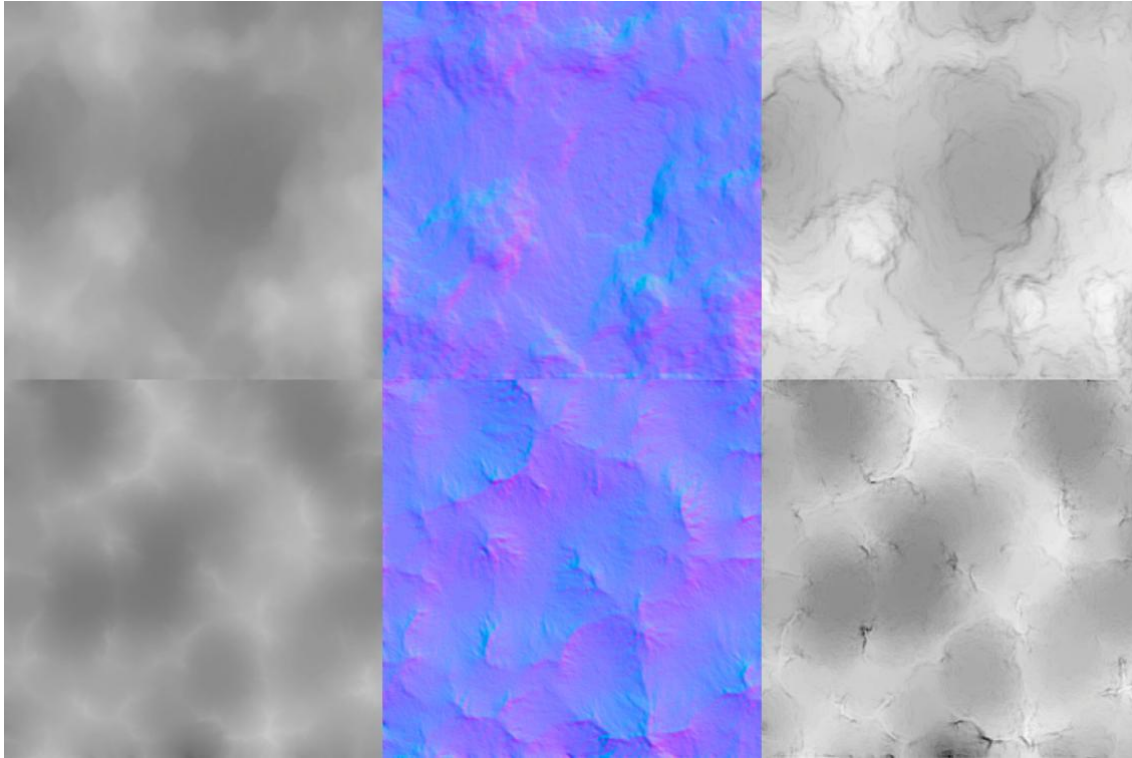




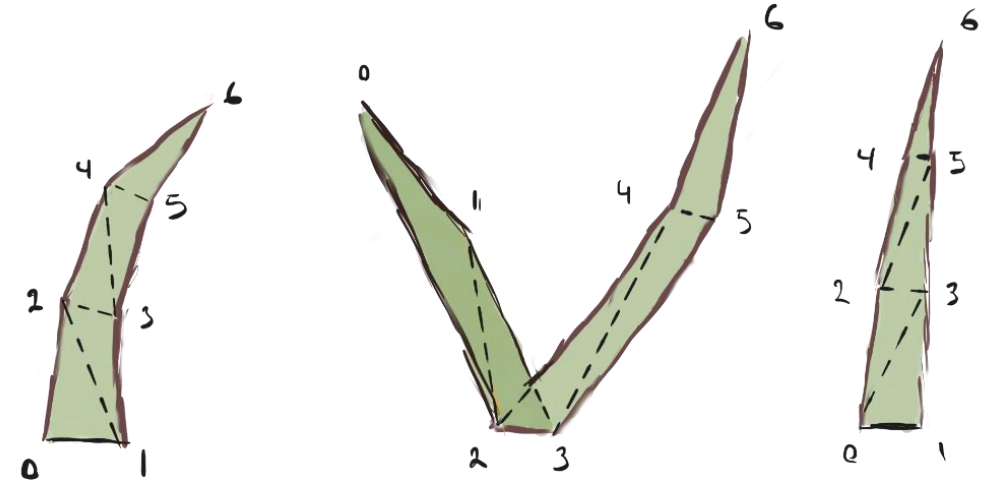


DuDv distortion map



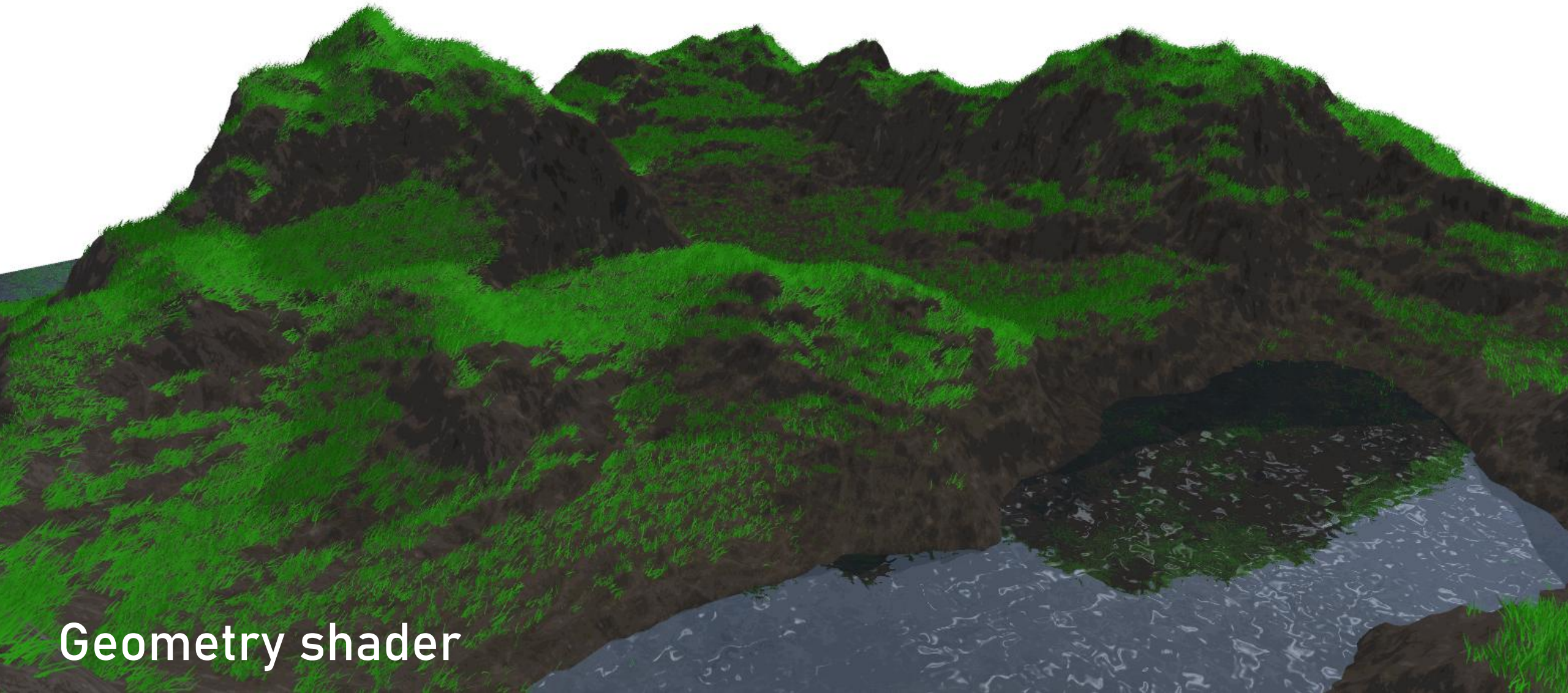


Cálculo de mapa de vegetación



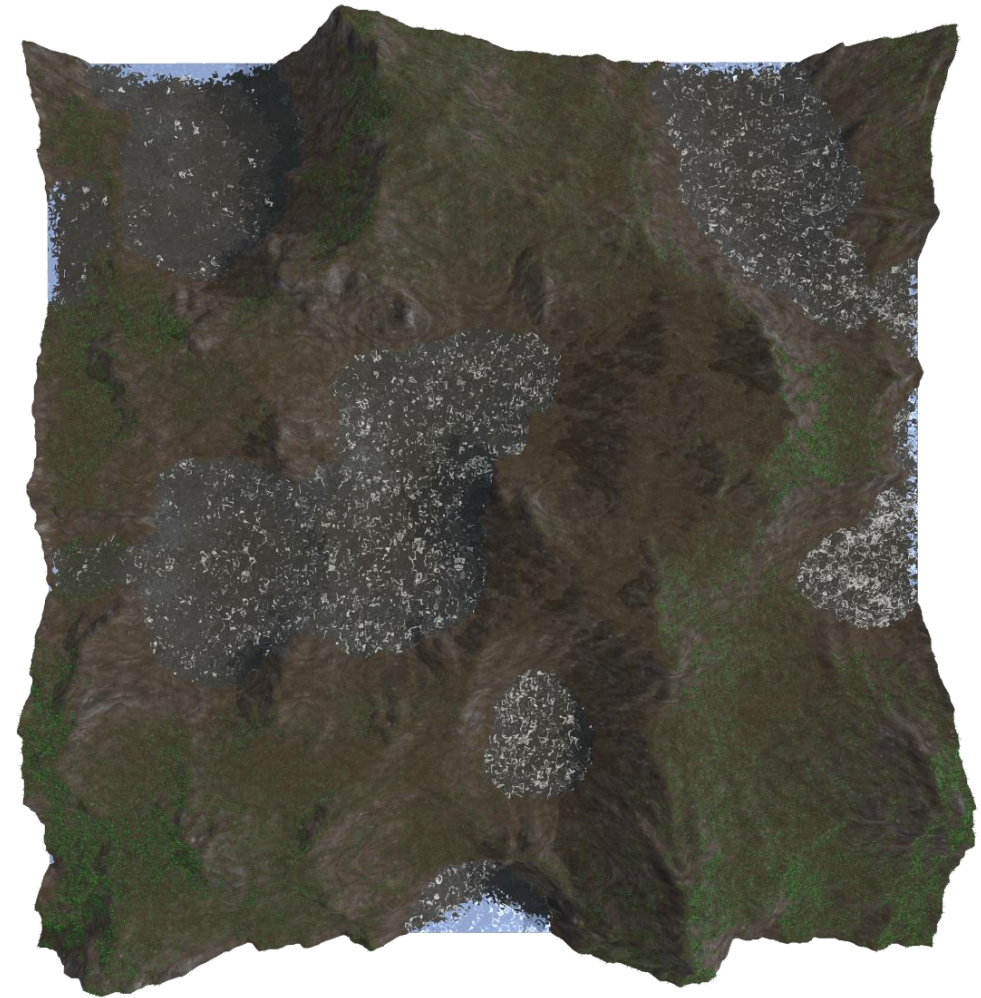
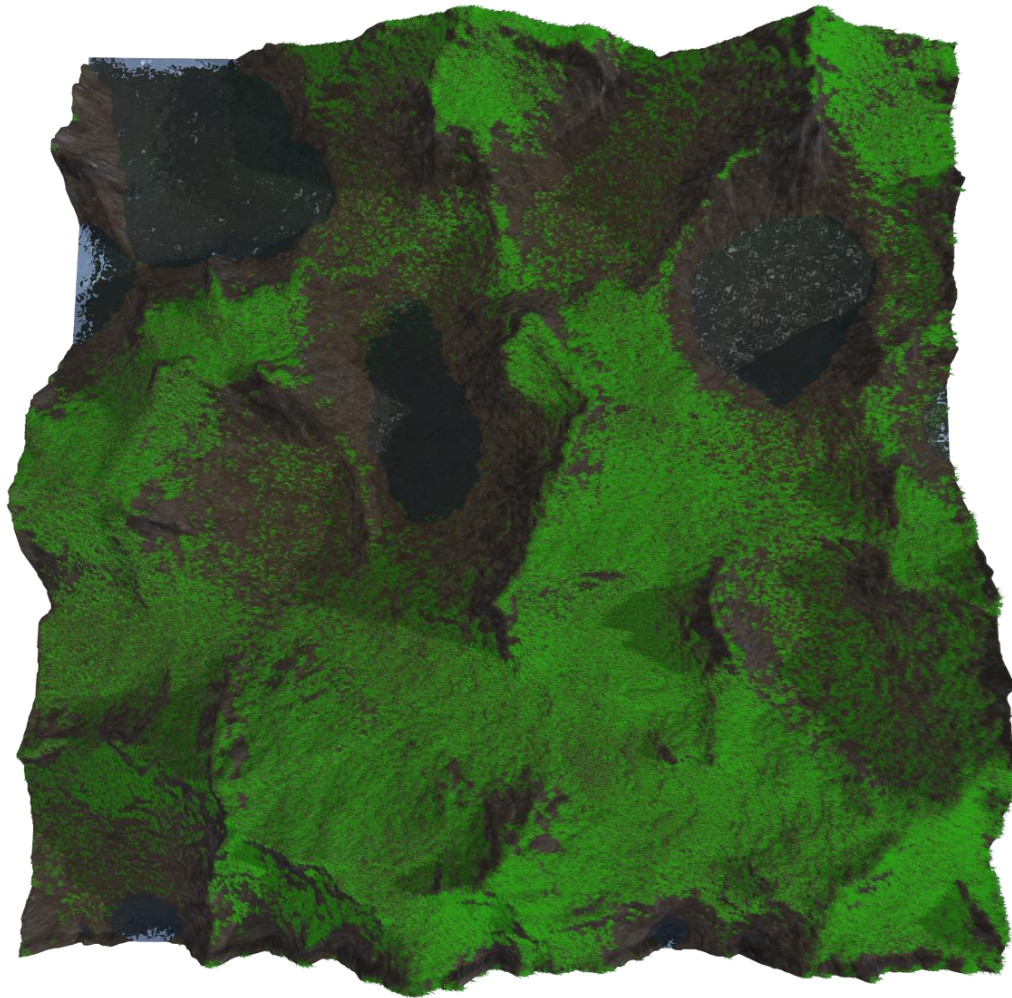
Modelos de vegetación



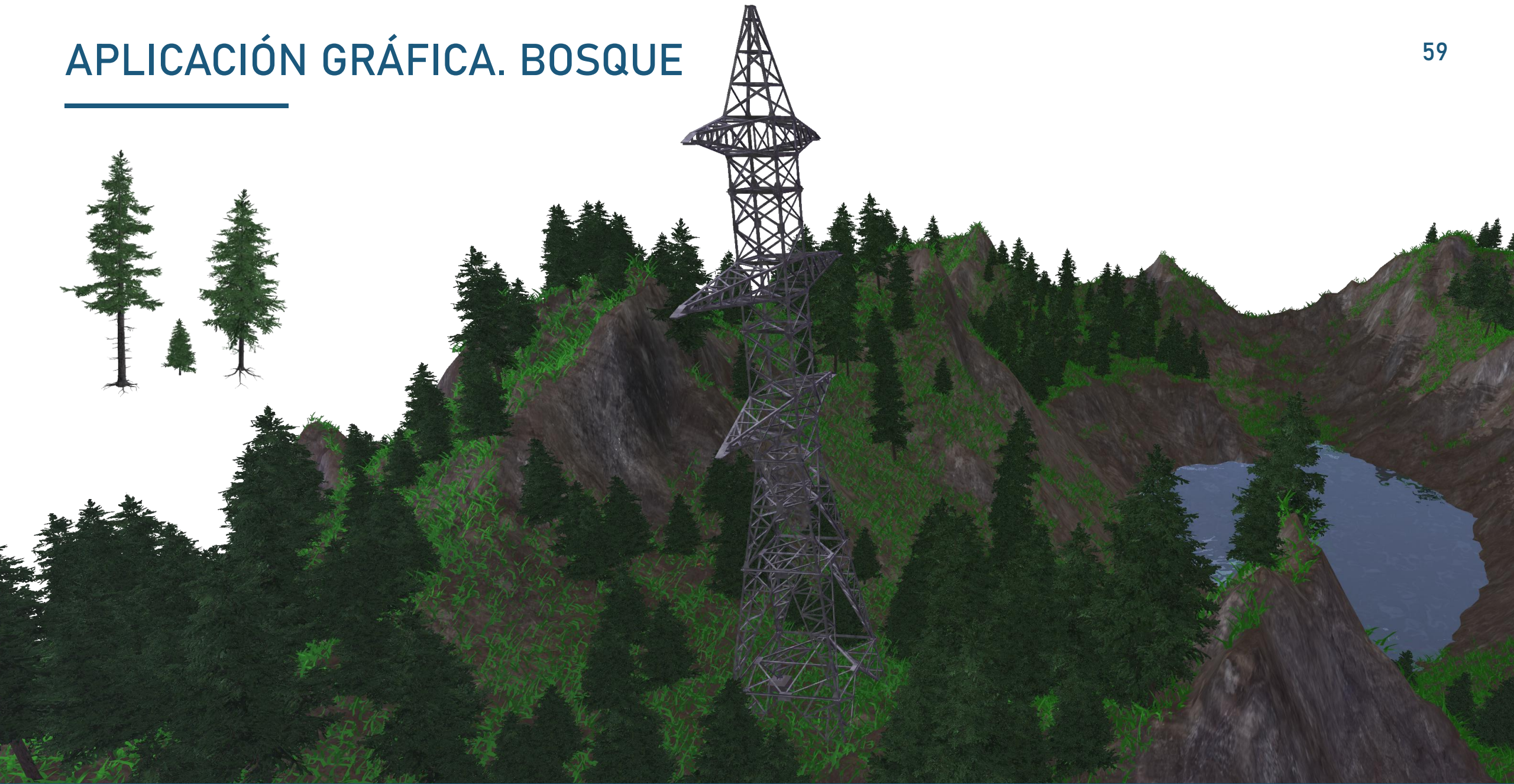


Geometry shader











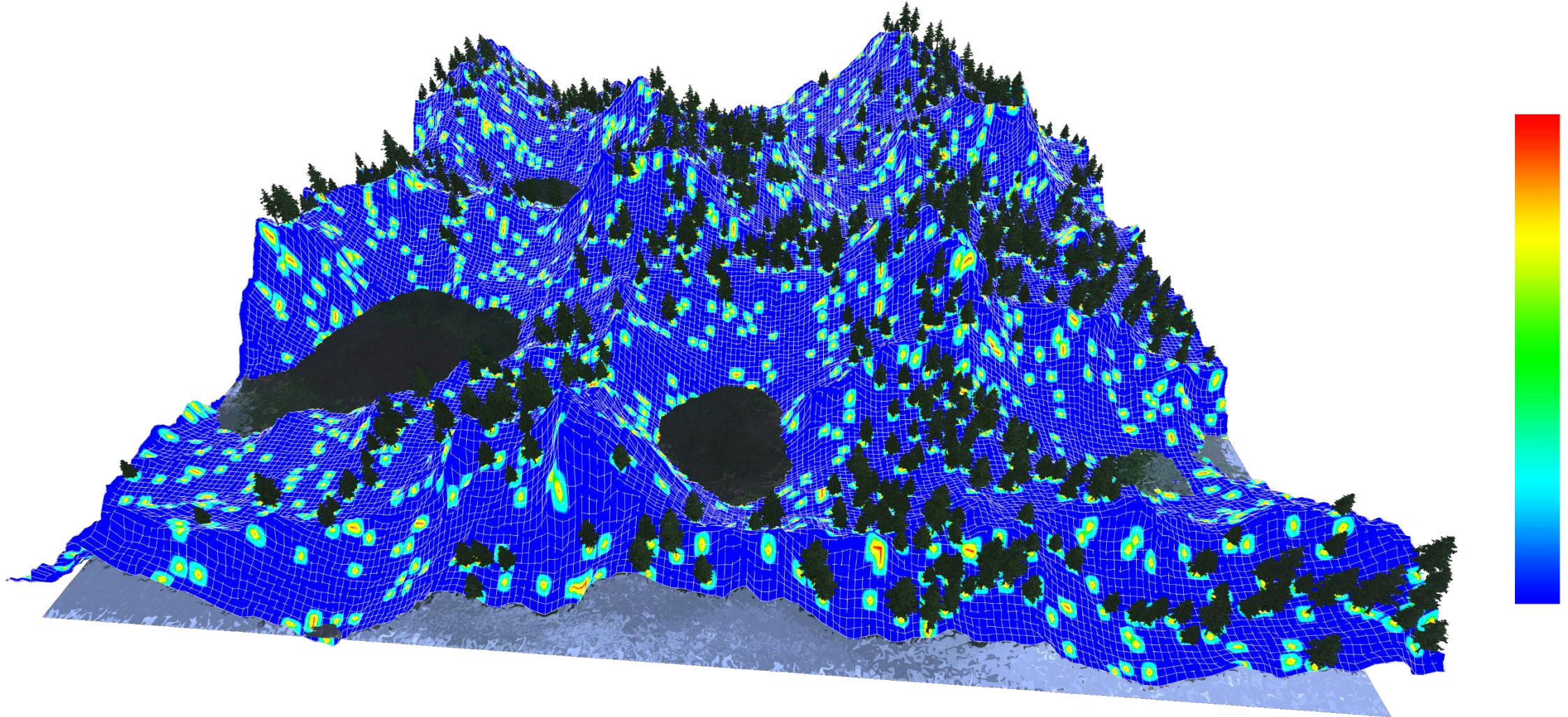


Posiciones generadas: **20.000**



Posiciones generadas: **2.500**



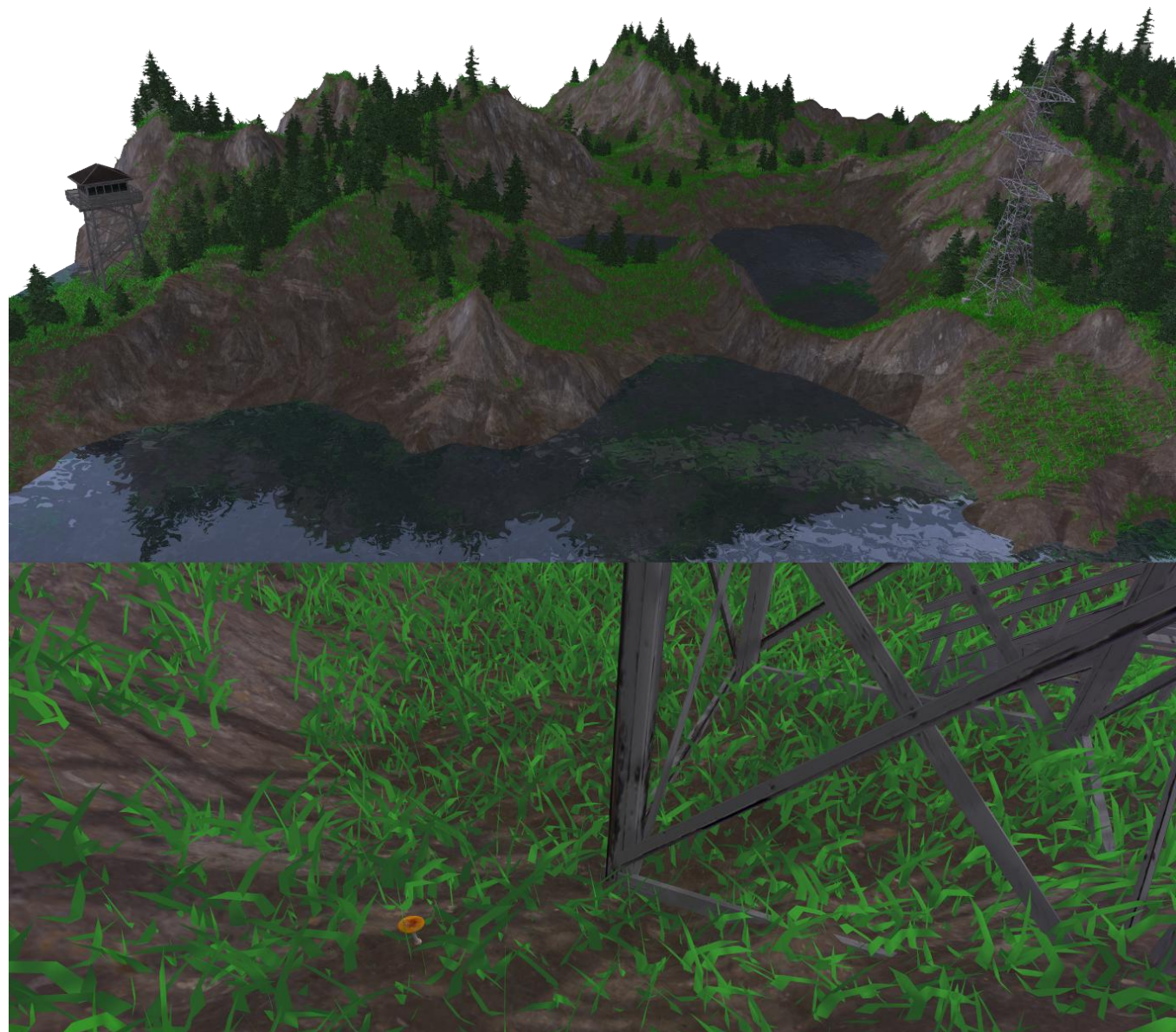
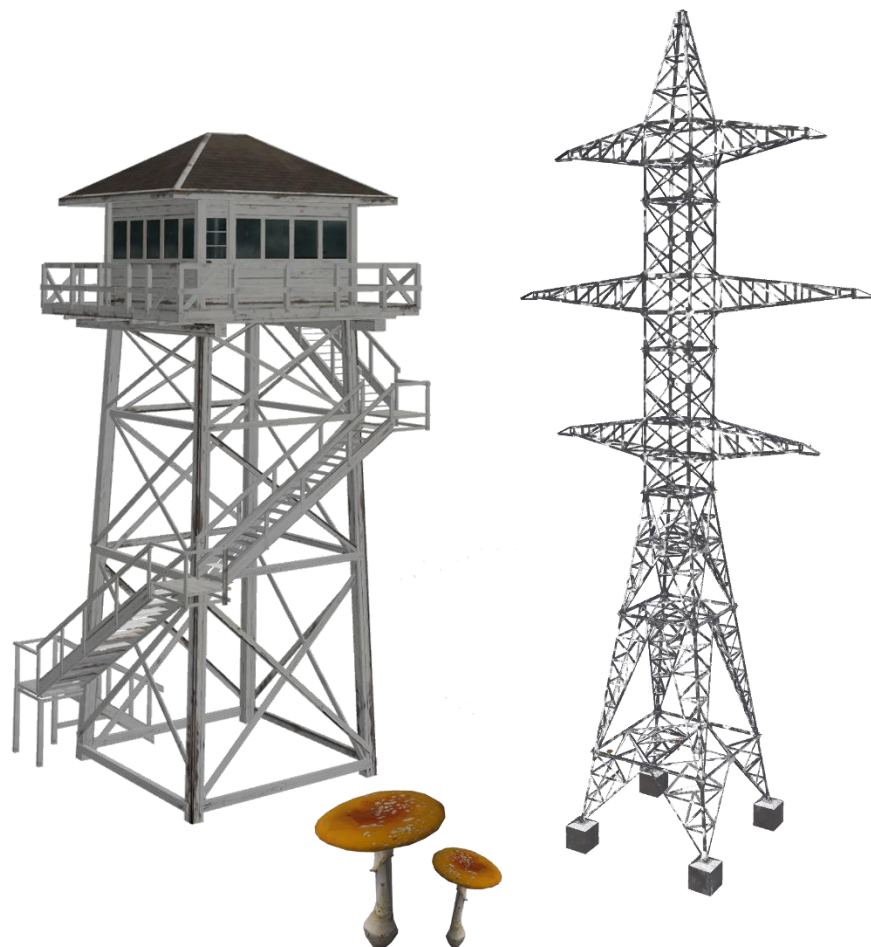




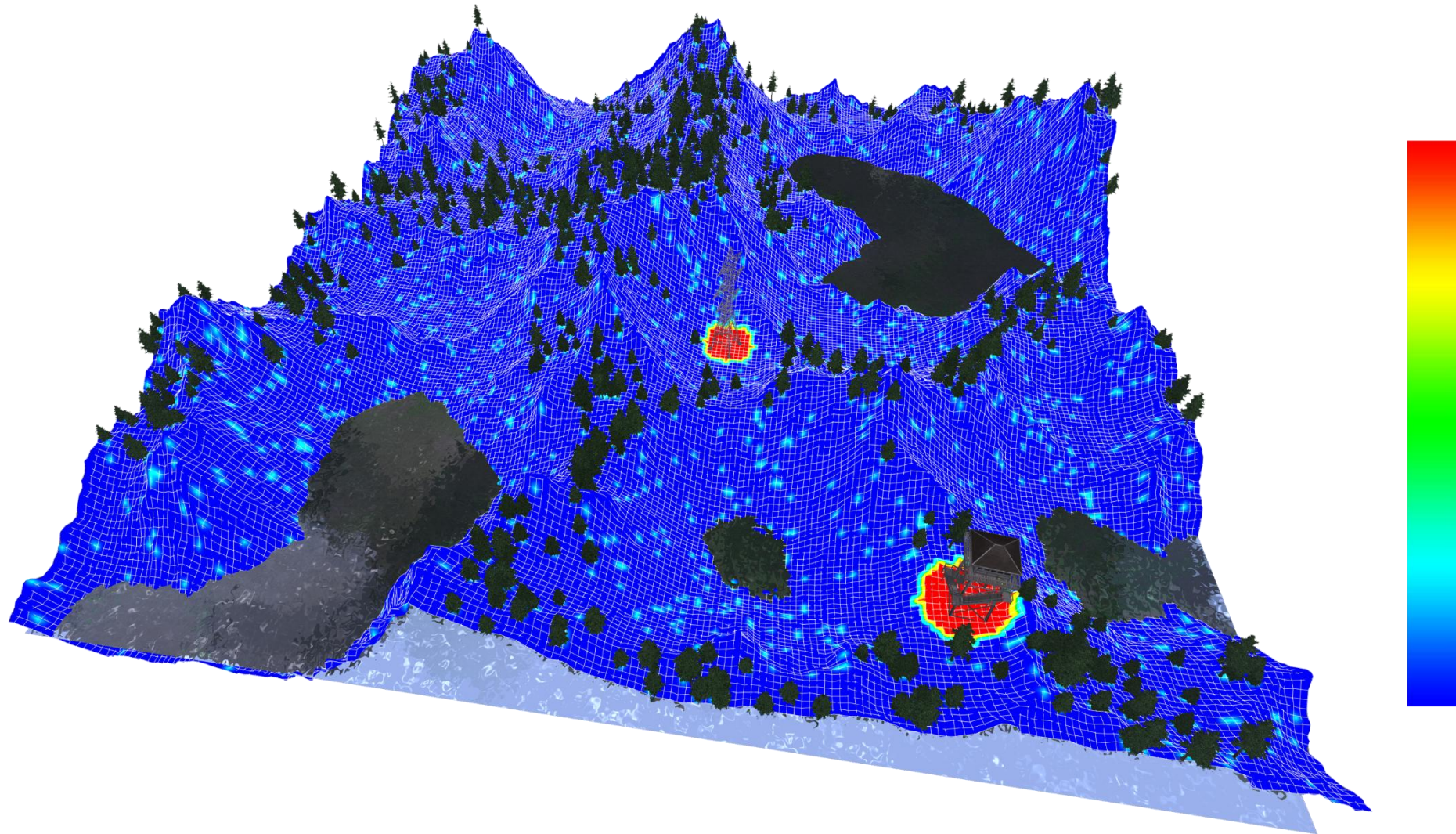
# APLICACIÓN GRÁFICA. OTROS MODELOS

---

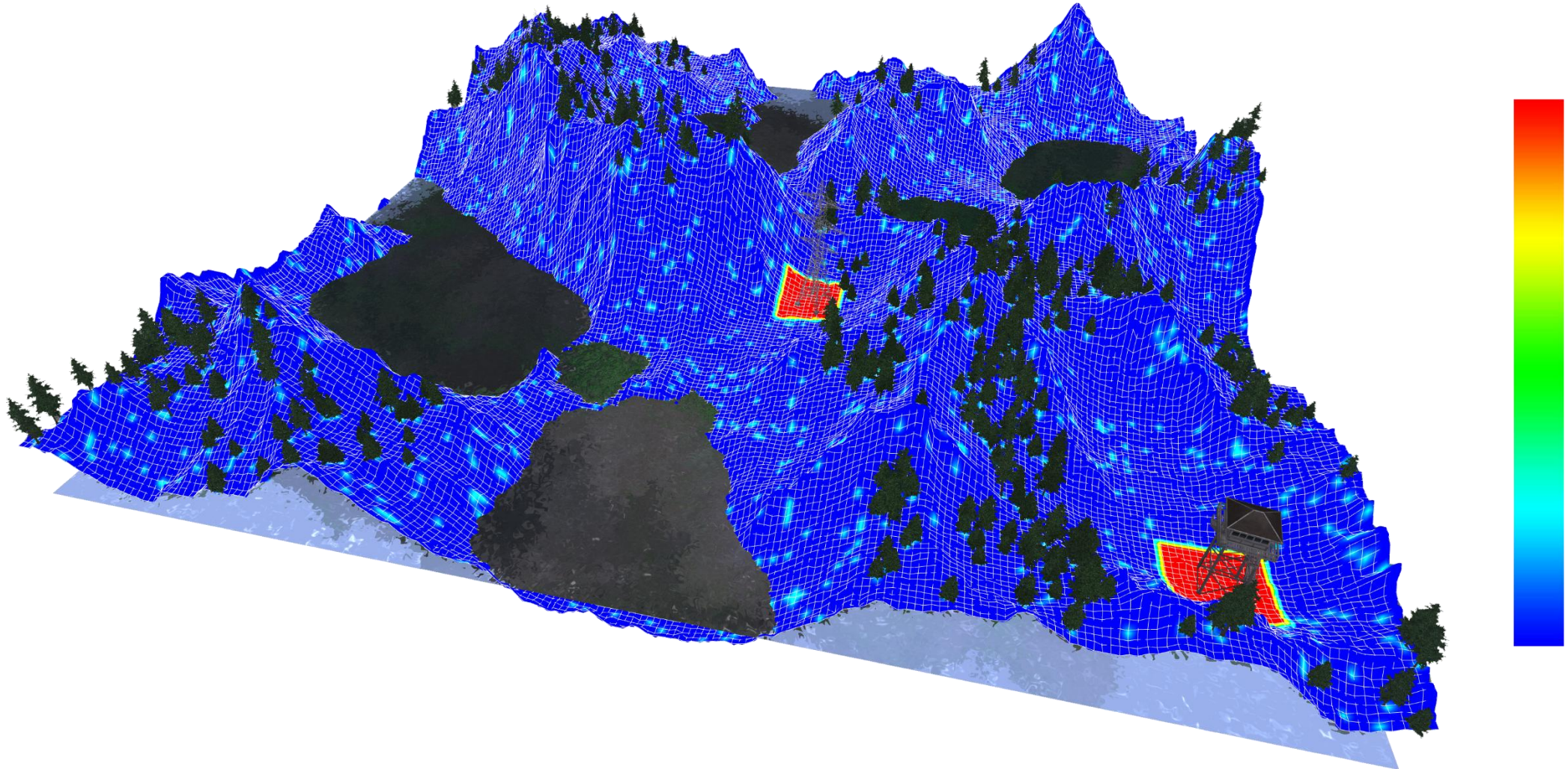
62

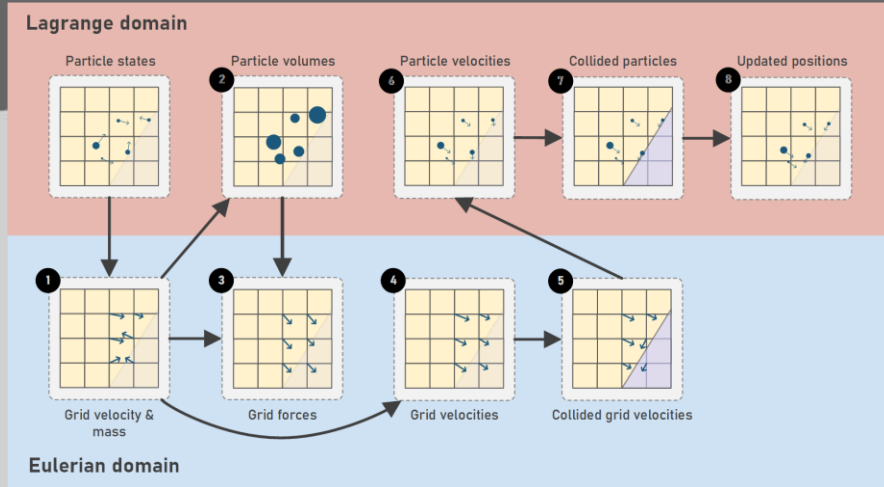




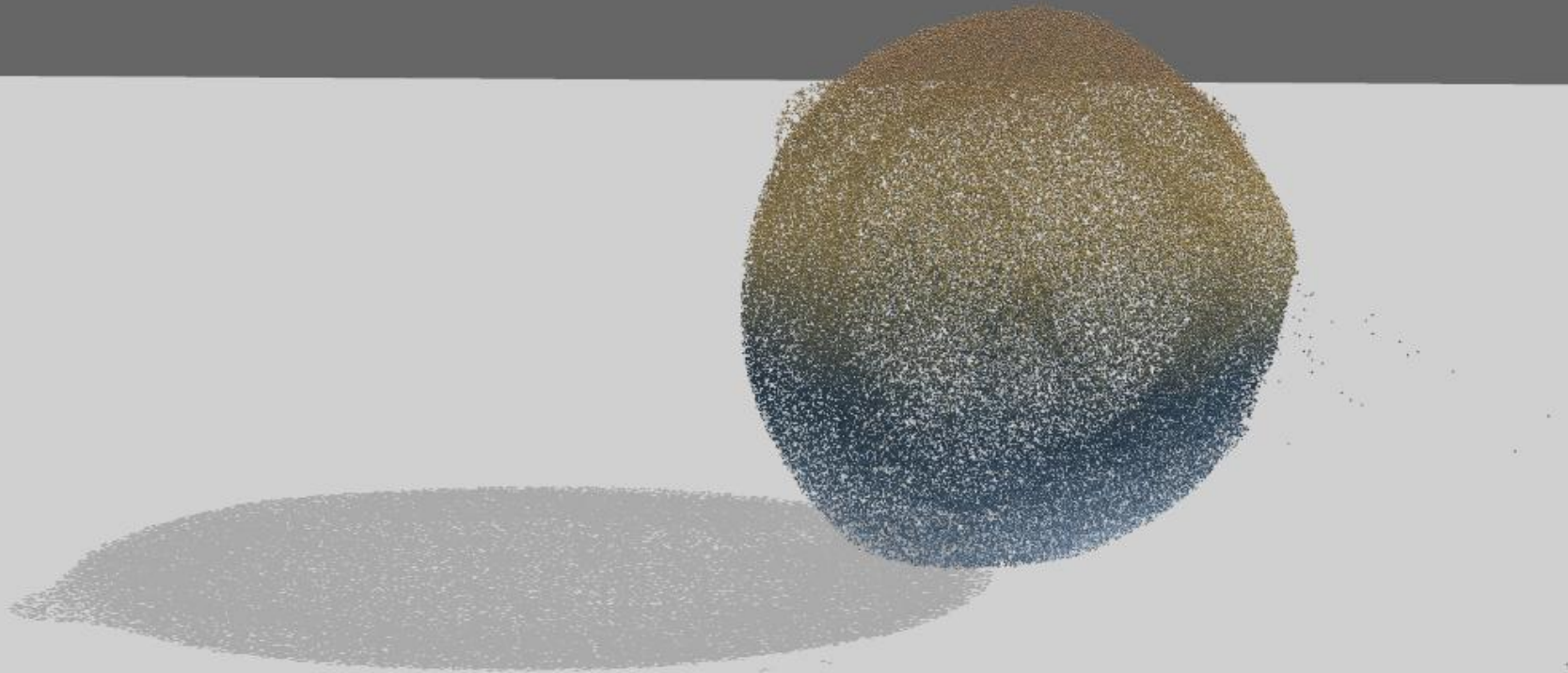










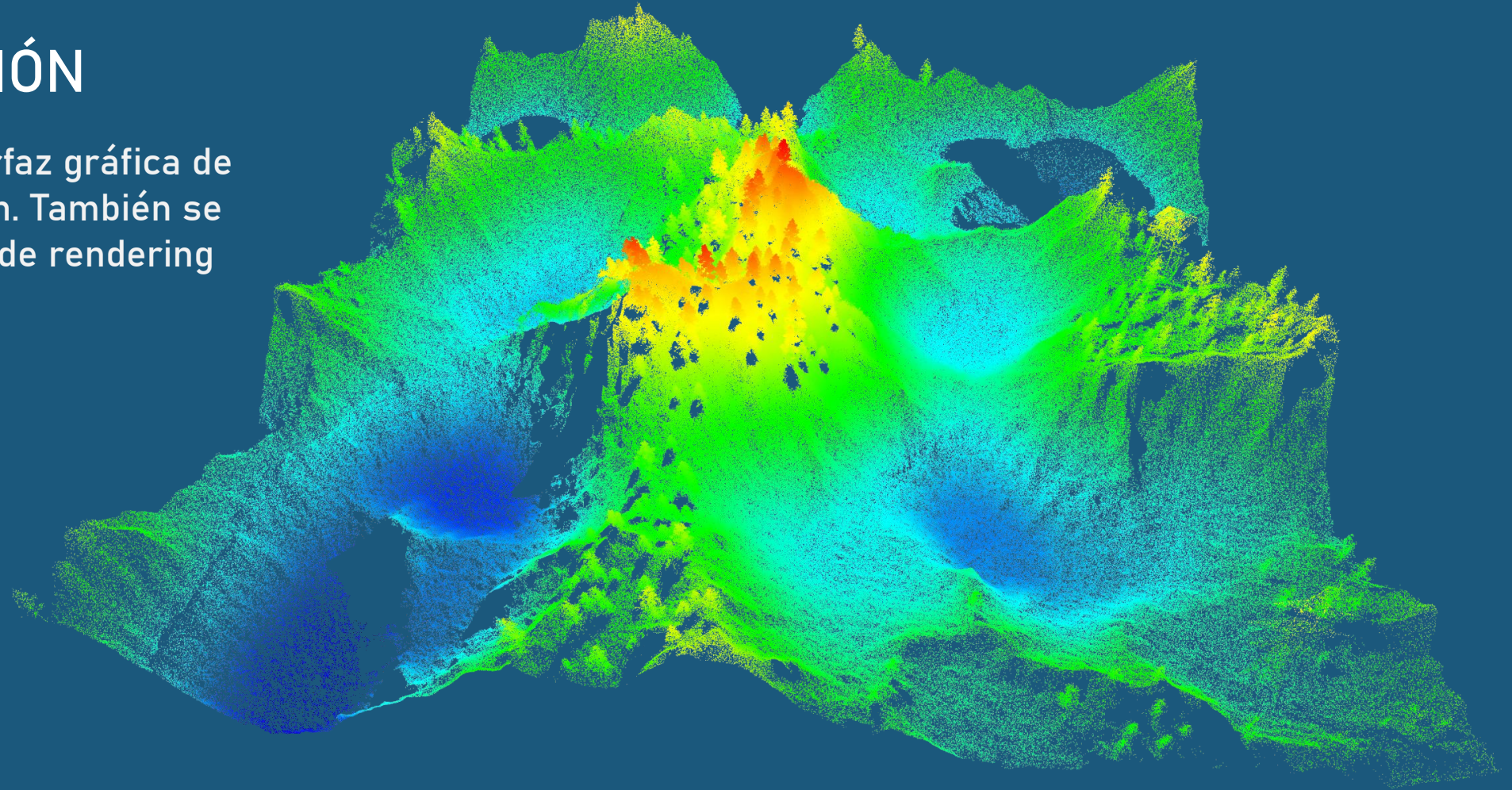


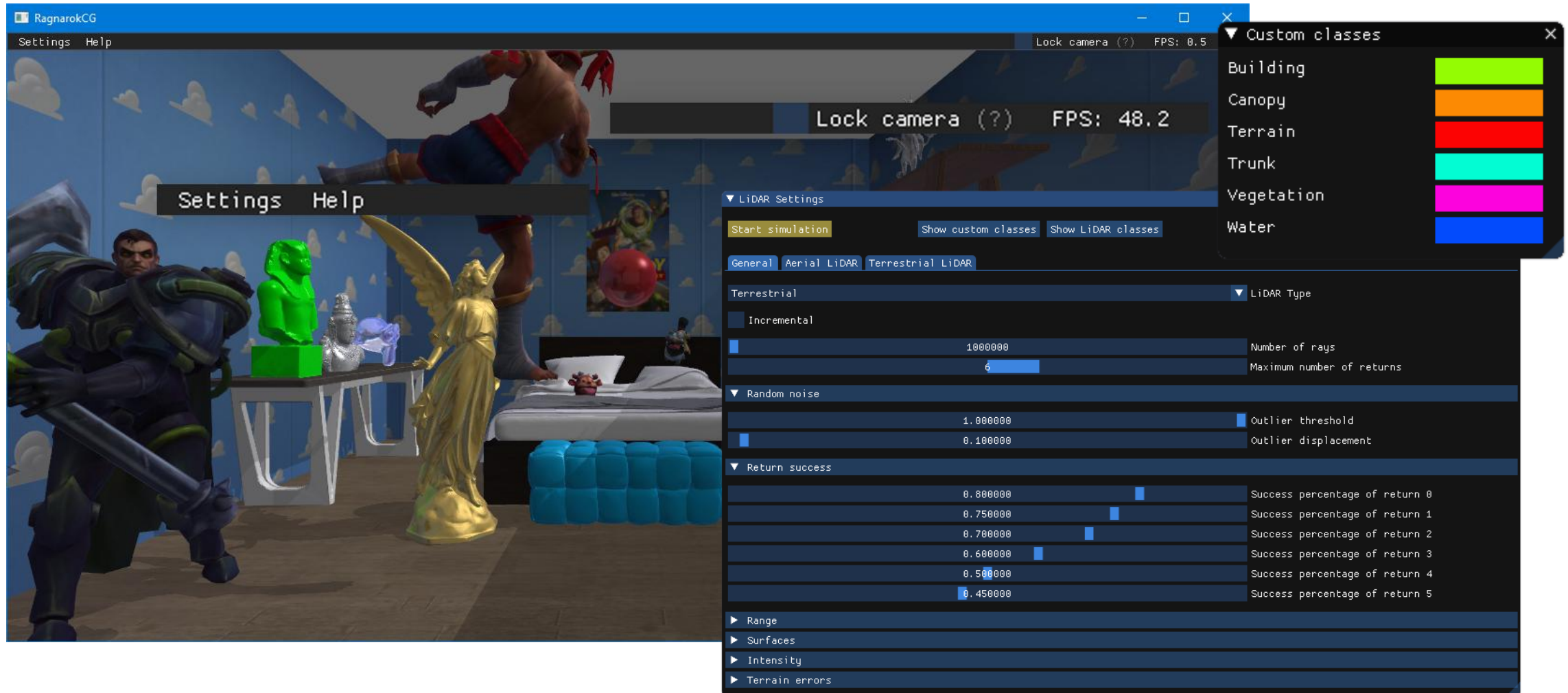


---

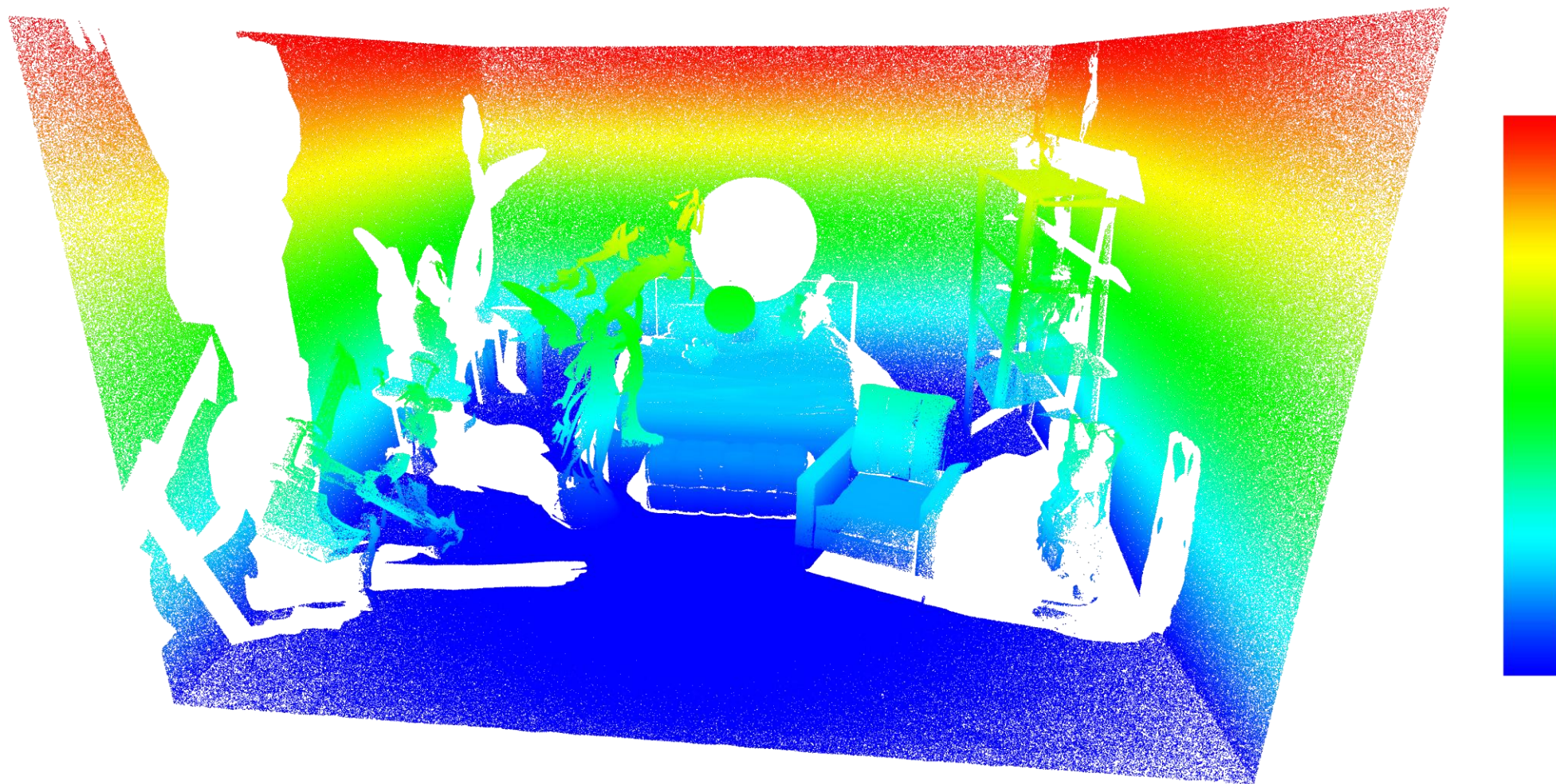
## SEXTA ITERACIÓN

Introducción de la interfaz gráfica de usuario en la aplicación. También se incluyen algunos tipos de rendering de nubes de puntos.

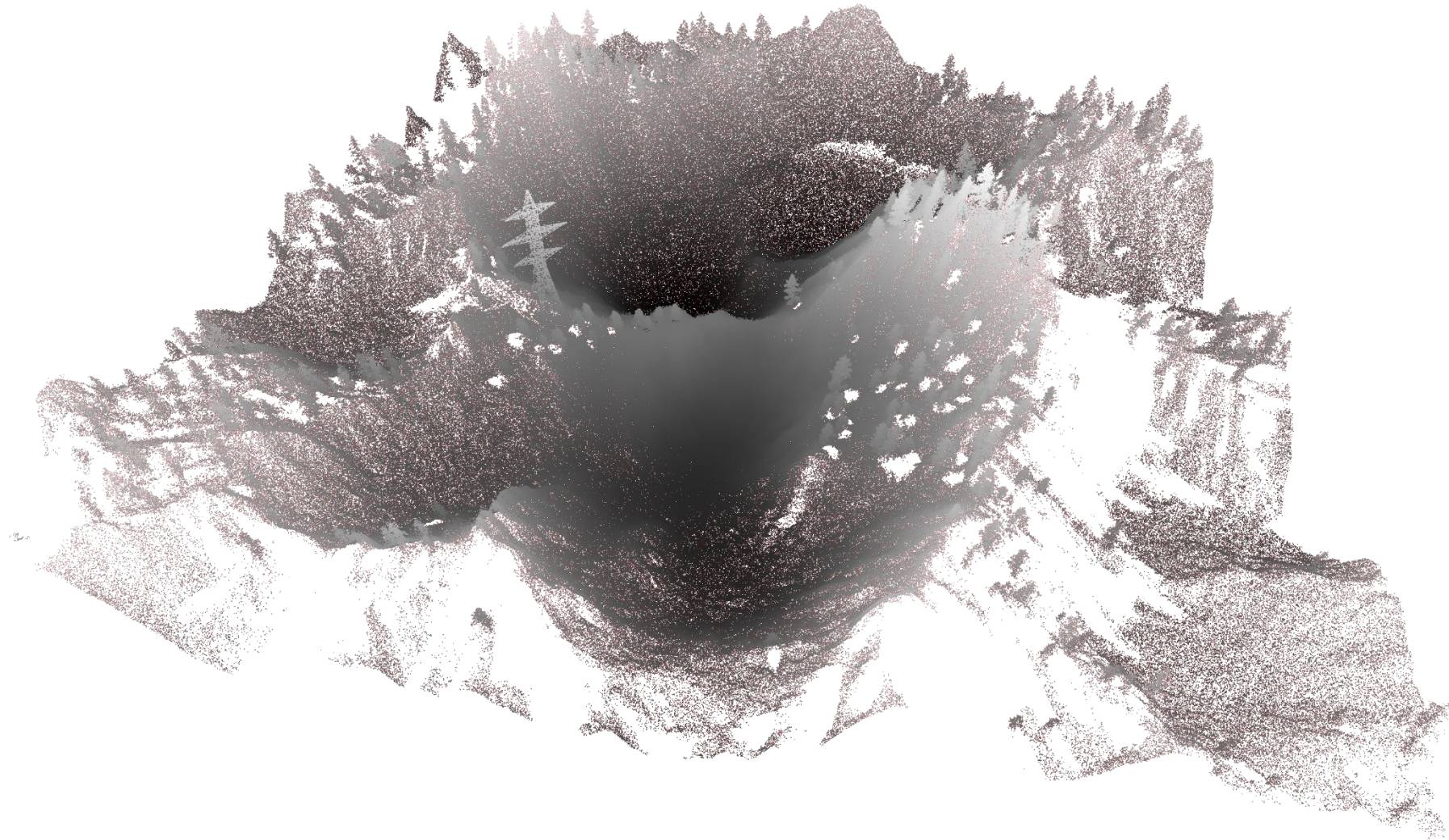
















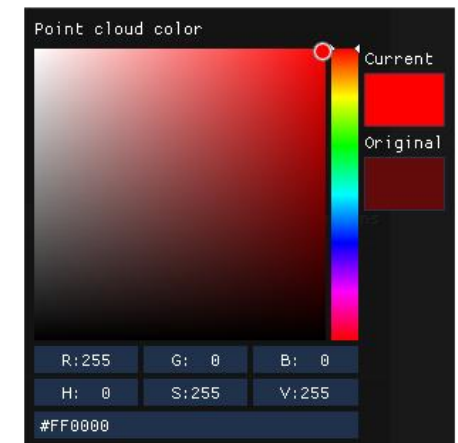
● #fd0202



● #ffffff

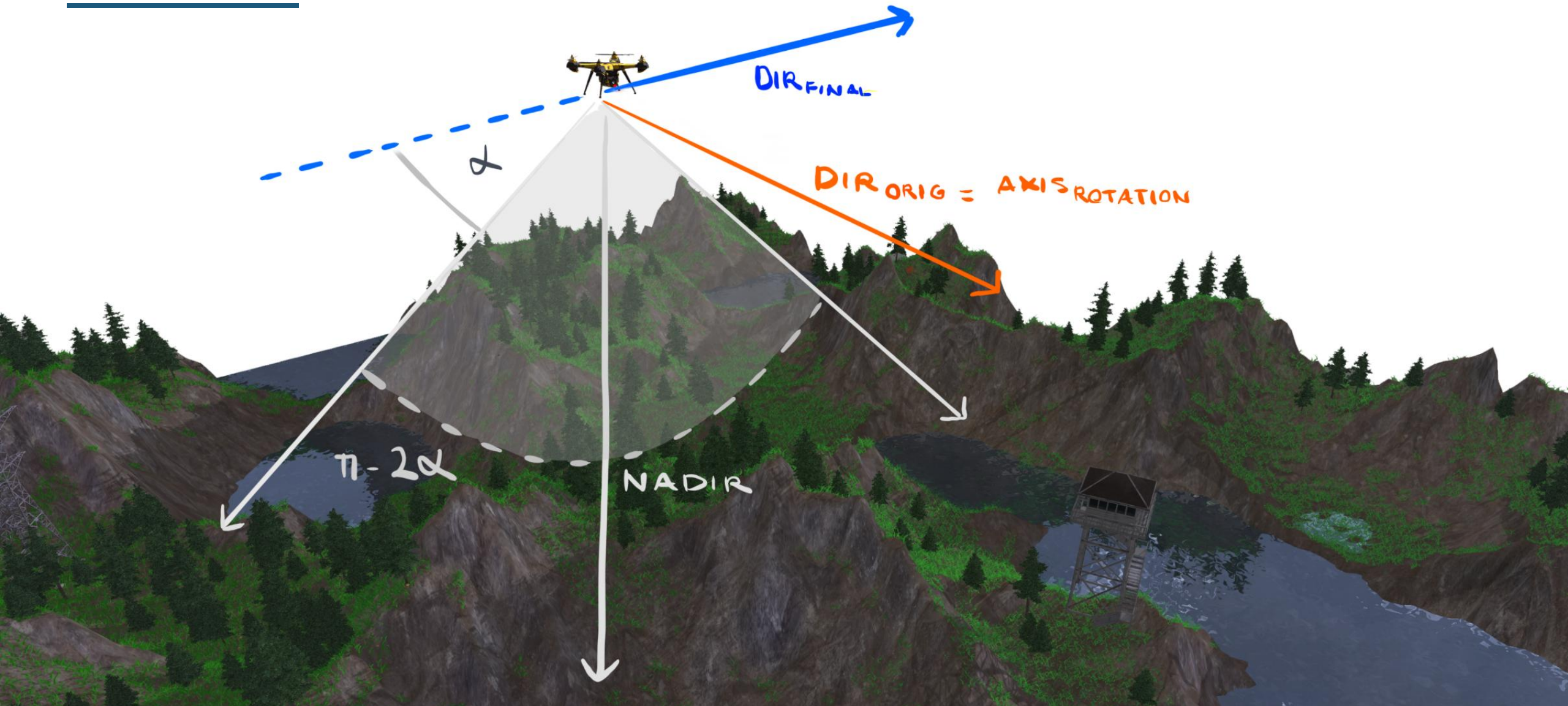


● #630b57

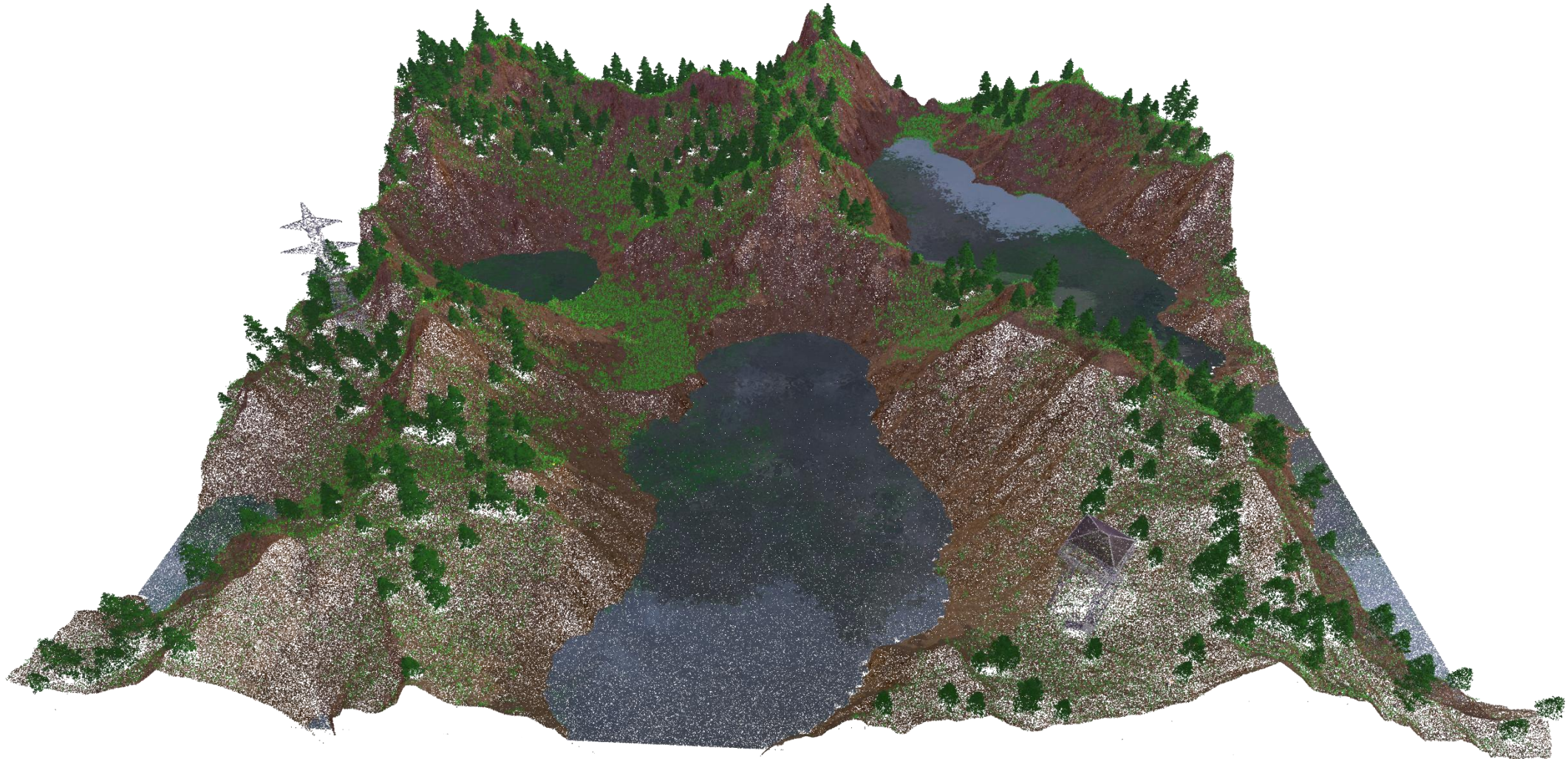


# SIMULACIÓN LiDAR. ESCANEEO AÉREO

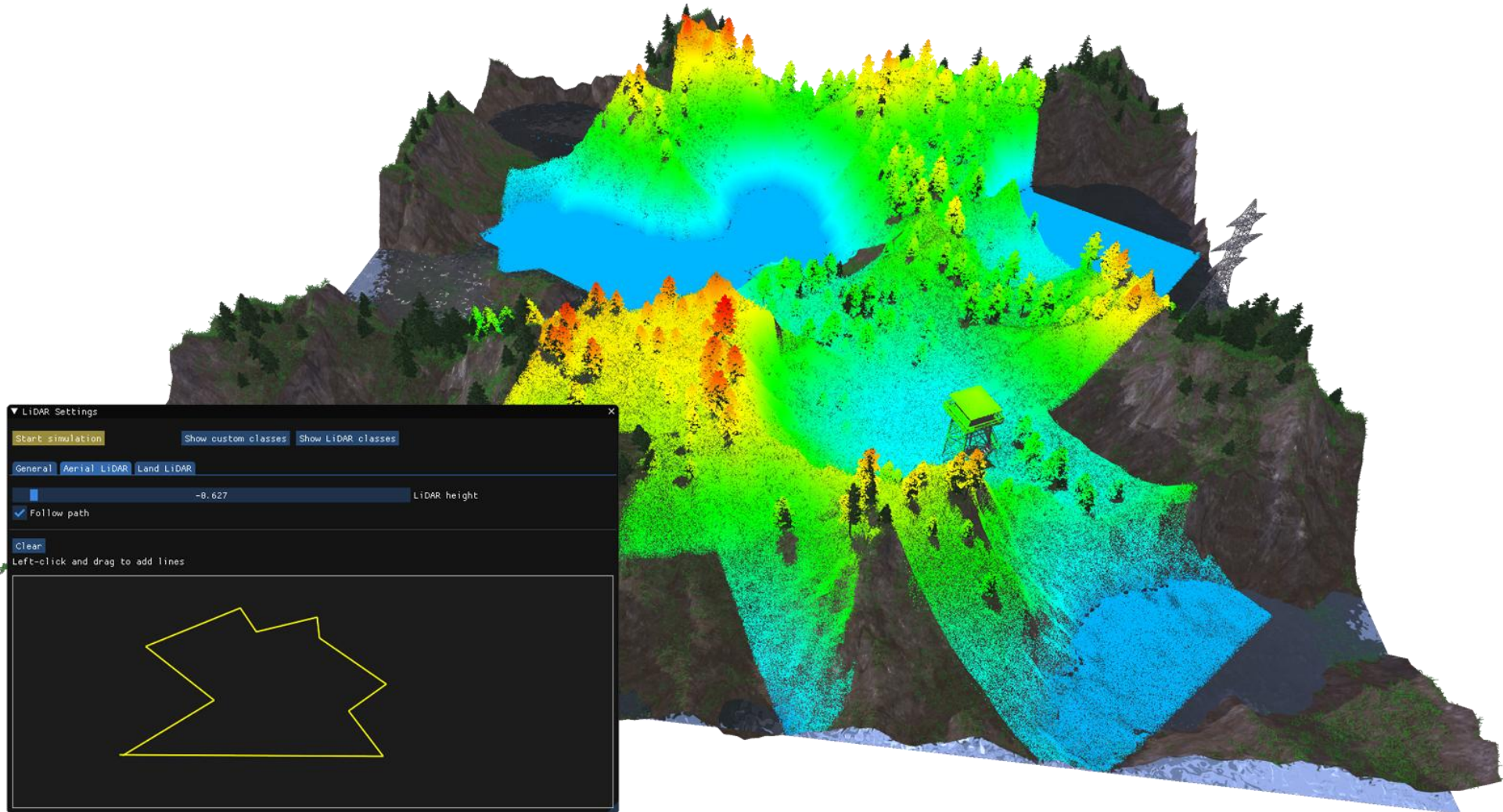
72

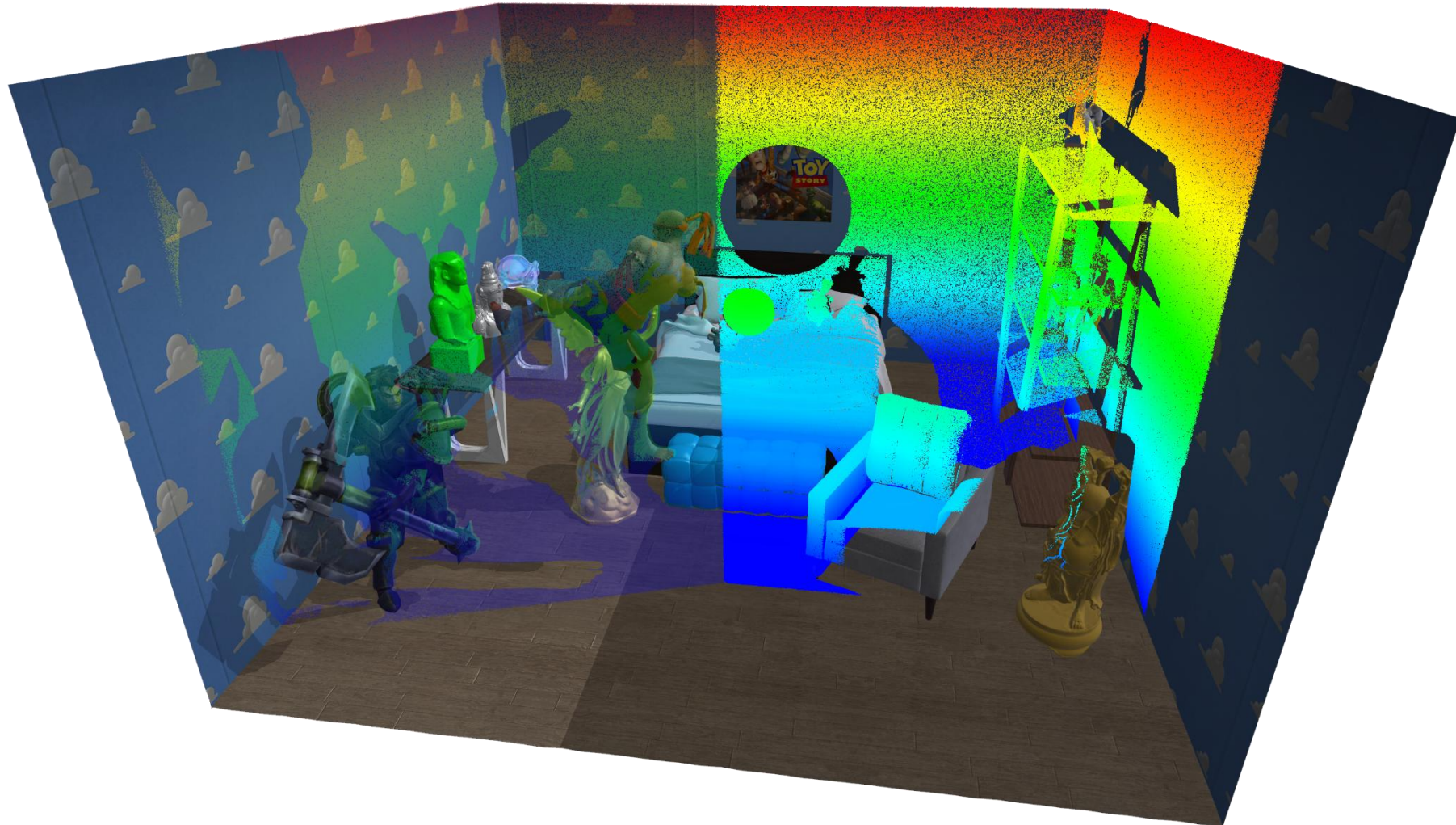




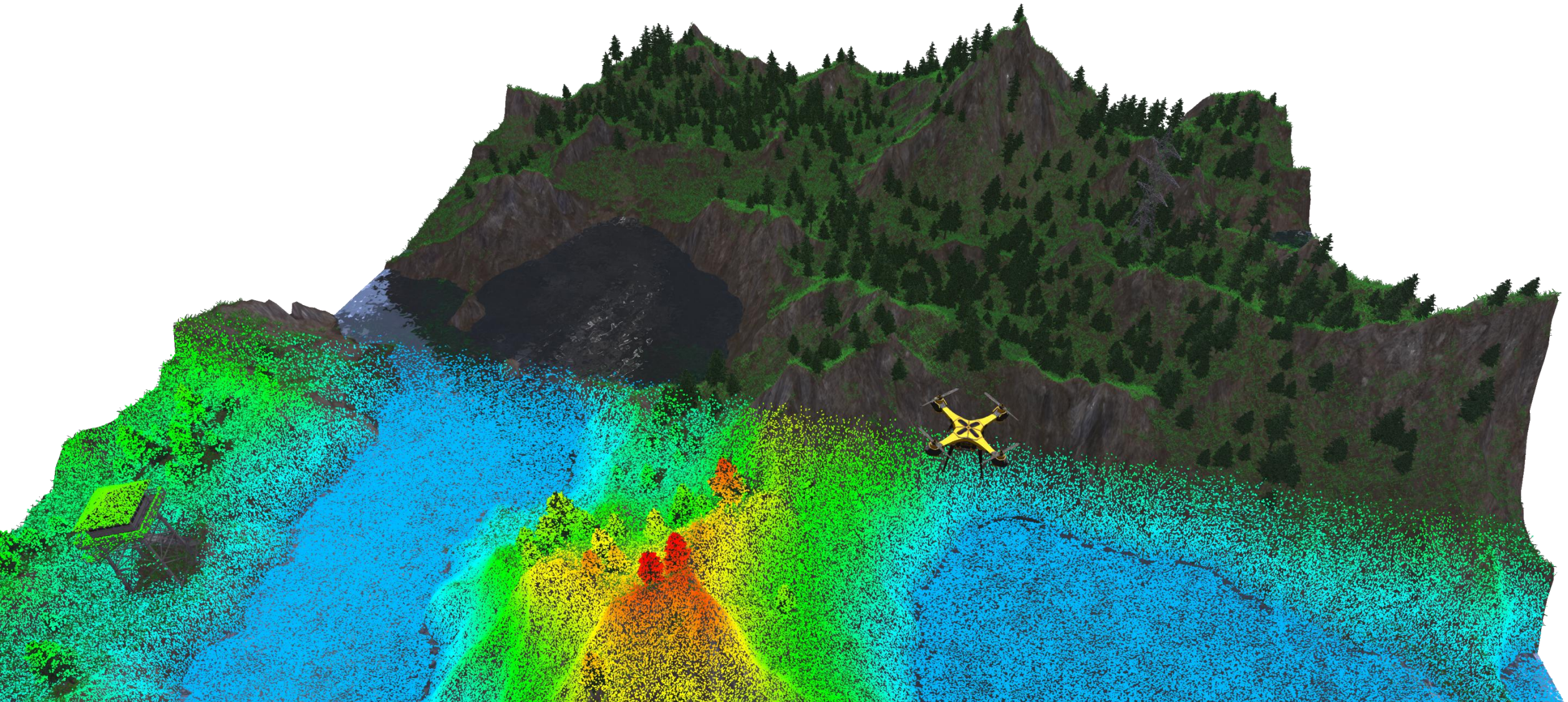














---

## SÉPTIMA ITERACIÓN

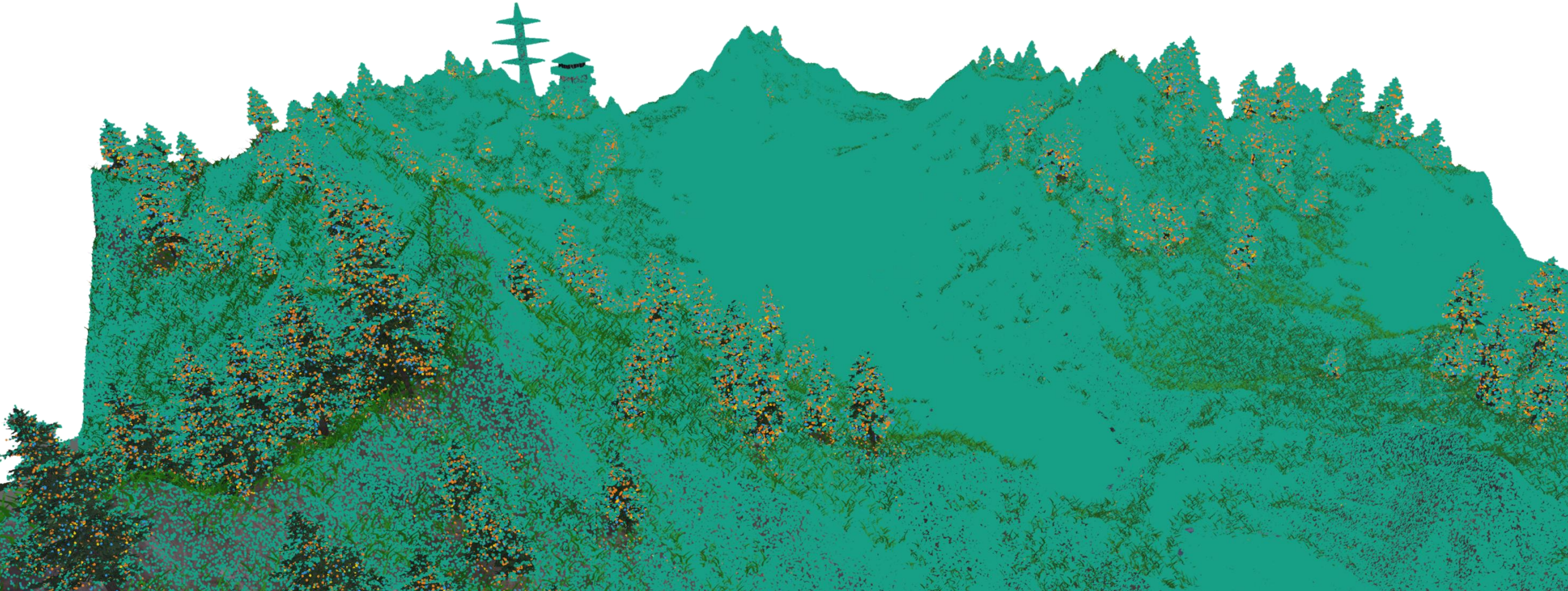
Simulación de errores en el escaneo LiDAR. Revisión de bibliografía donde se describen los principales errores observados y la magnitud de los mismos.



# SIMULACIÓN LiDAR. MÚLTIPLES RETORNOS

78

Color asignado en función del  
valor de retorno





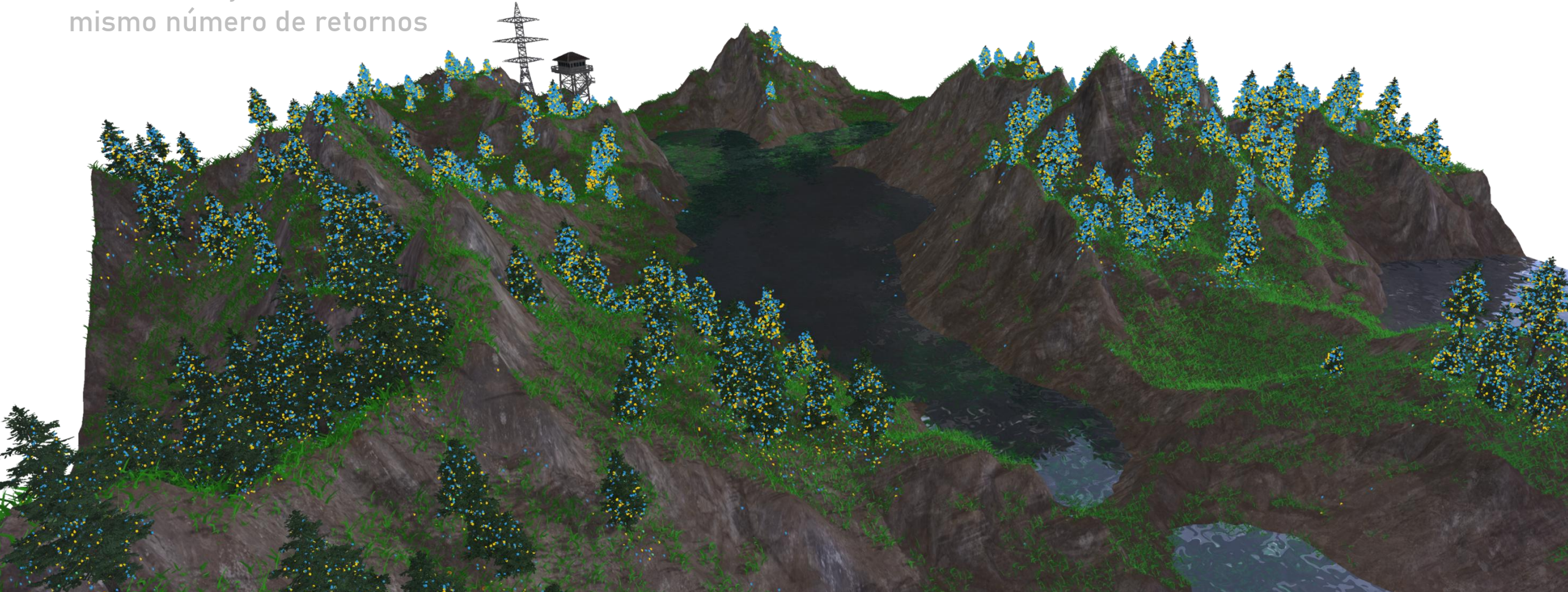
# SIMULACIÓN LiDAR. MÚLTIPLES RETORNOS

---

79

## Filtrado por valor de retorno

Todos los rayos no tienen el mismo número de retornos





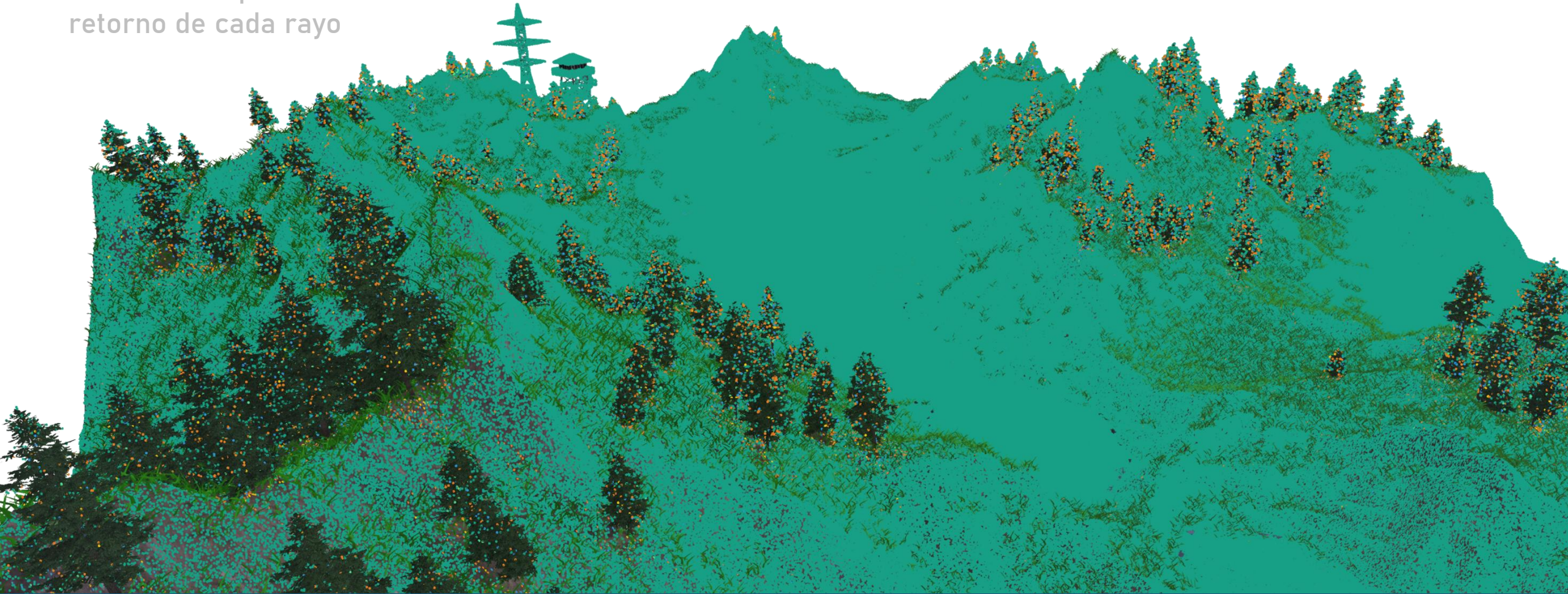
# SIMULACIÓN LiDAR. MÚLTIPLES RETORNOS

---

80

## Filtrado por valor de retorno normalizado

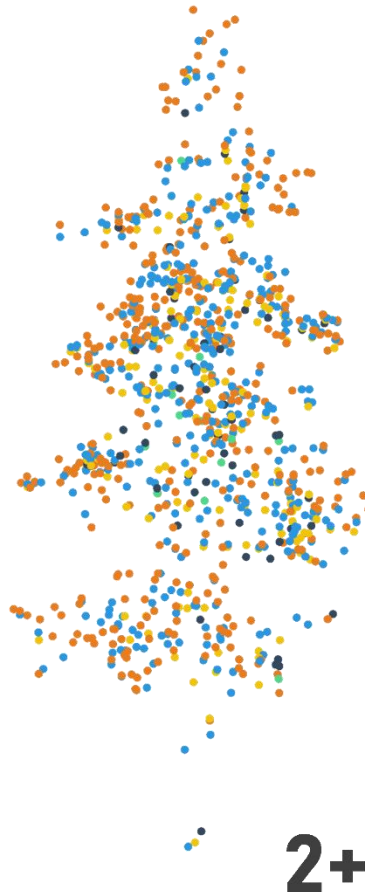
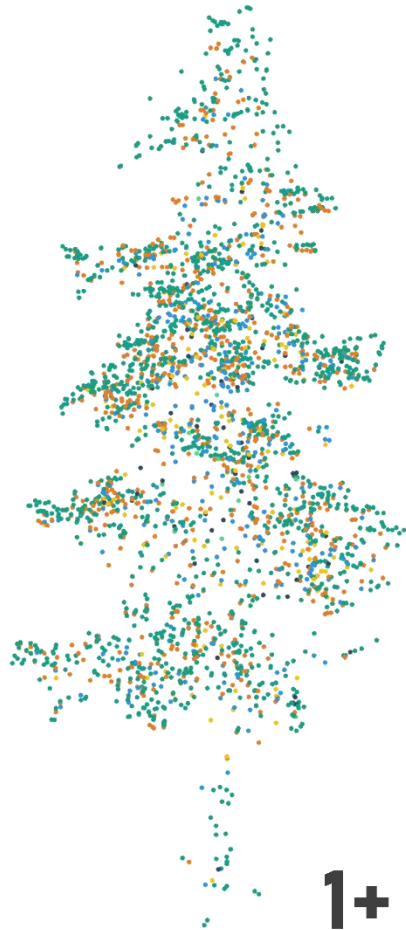
Permite recuperar el último  
retorno de cada rayo





# SIMULACIÓN LiDAR. MÚLTIPLES RETORNOS

81

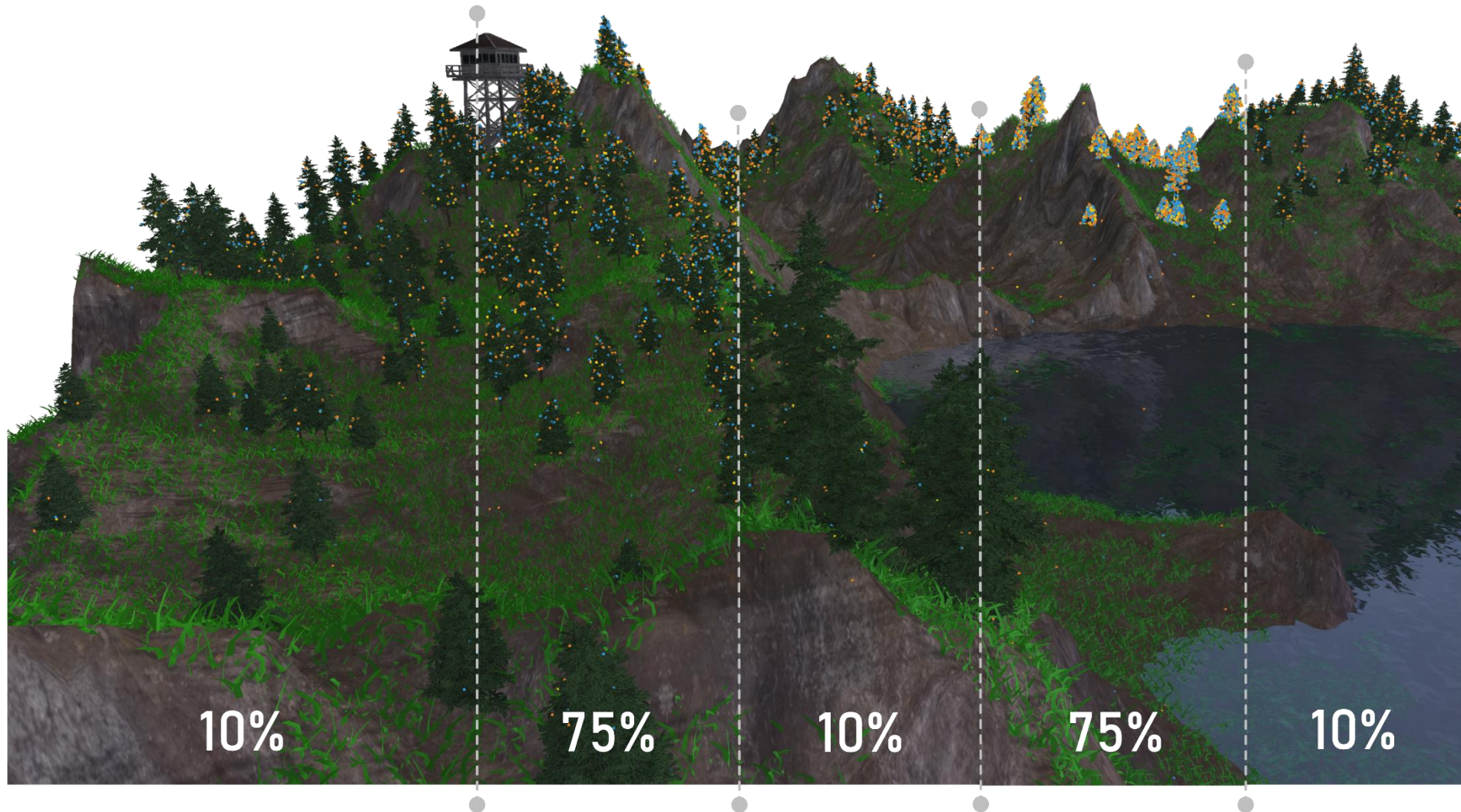


Aplicación de  
retornos sobre  
superficies con  
opacidad  $< 1$



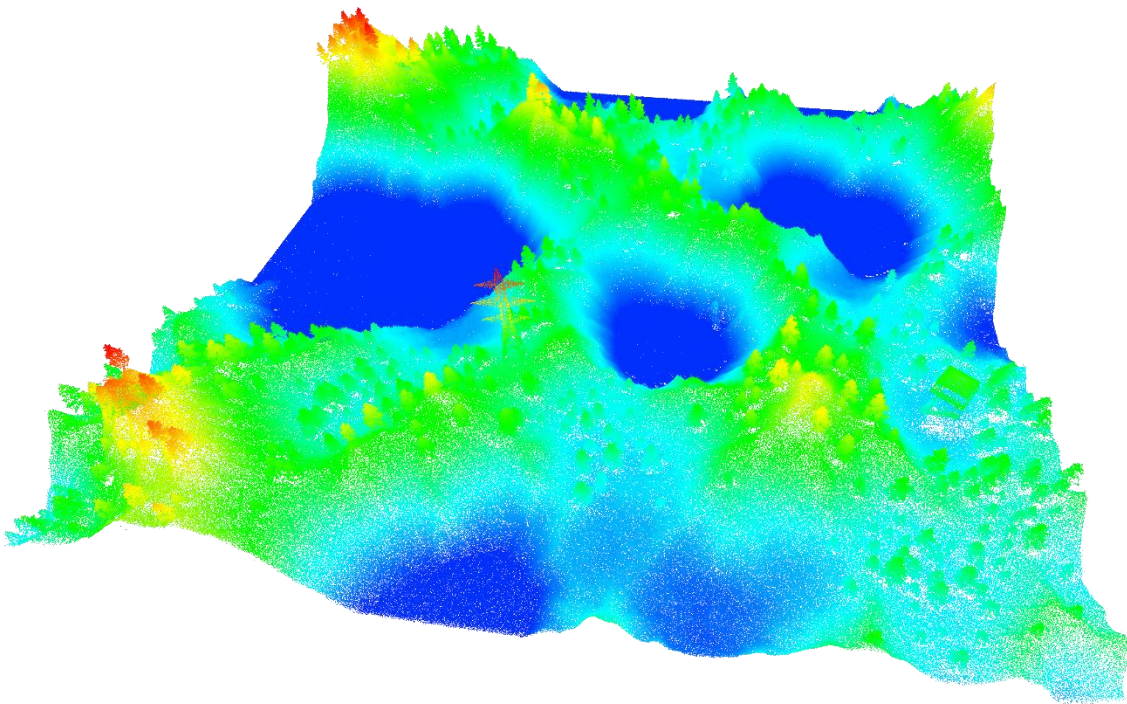
# SIMULACIÓN LiDAR. MÚLTIPLES RETORNOS

82

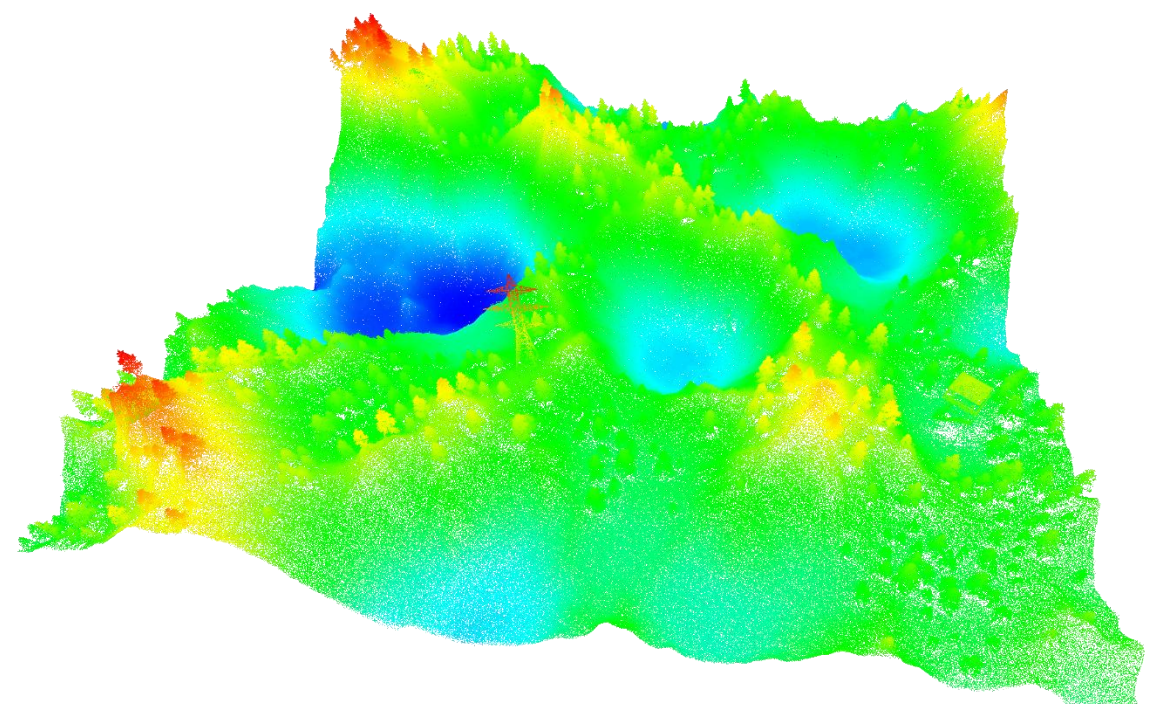




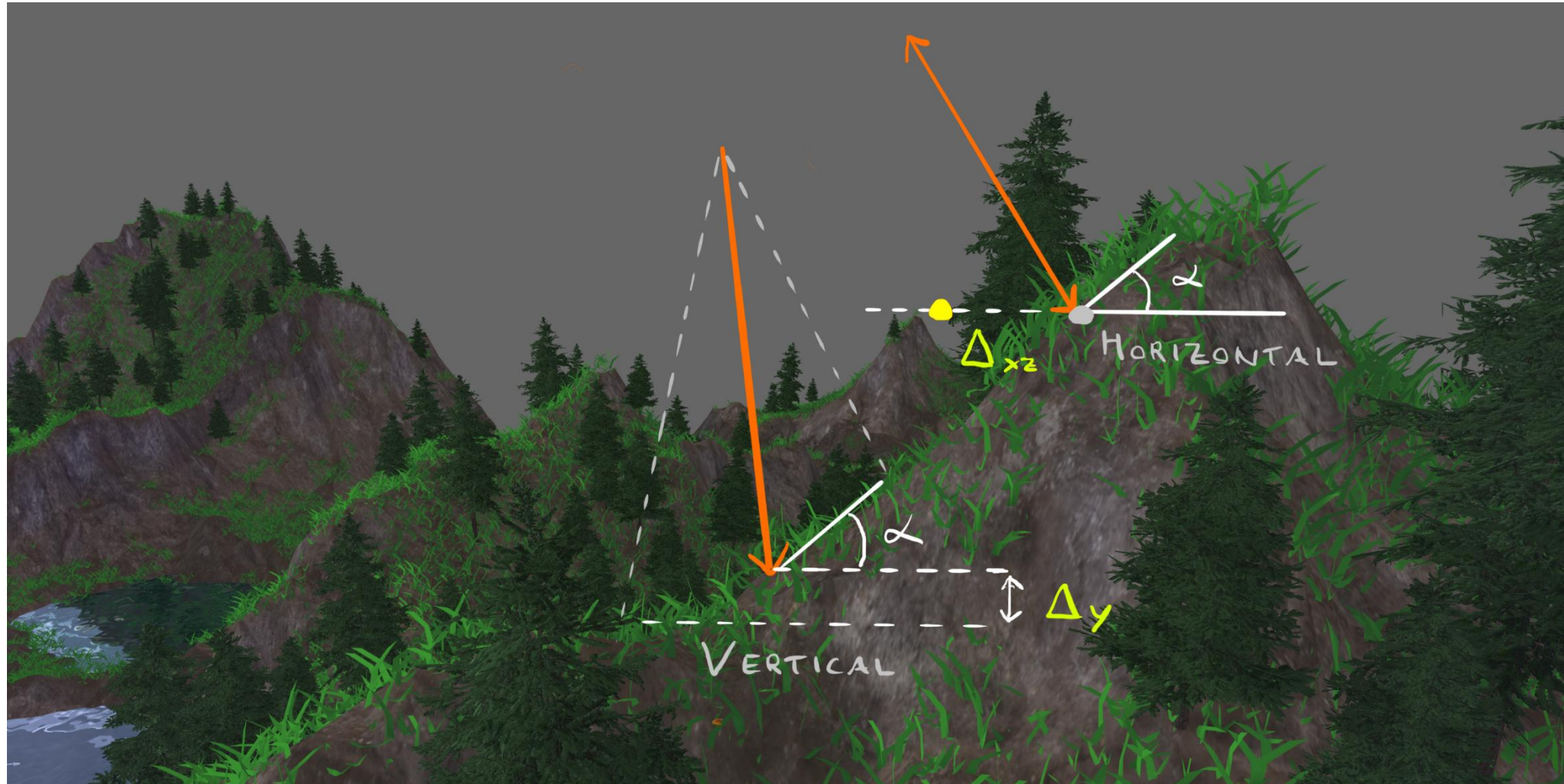
LiDAR no batimétrico



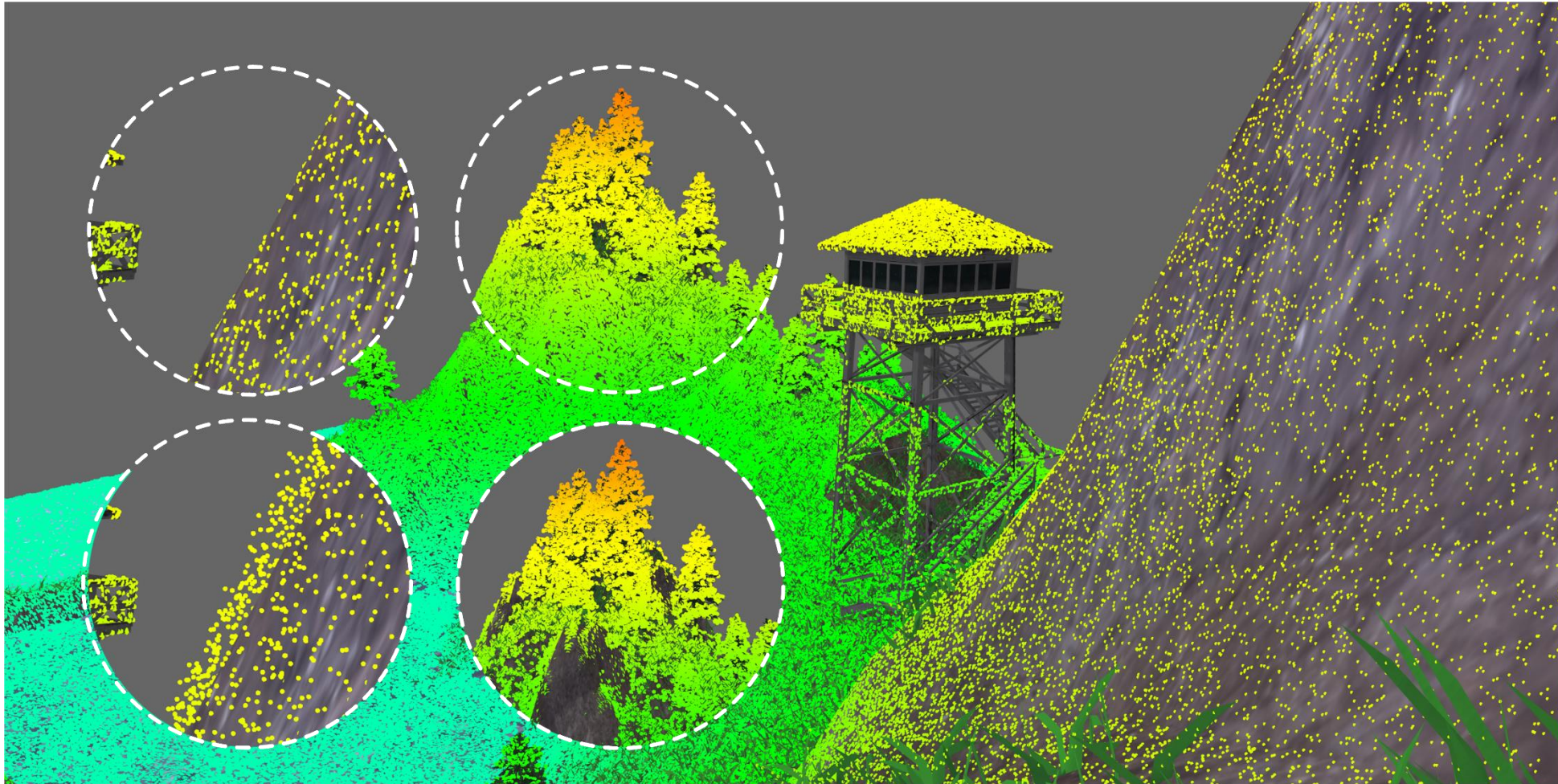
LiDAR batimétrico







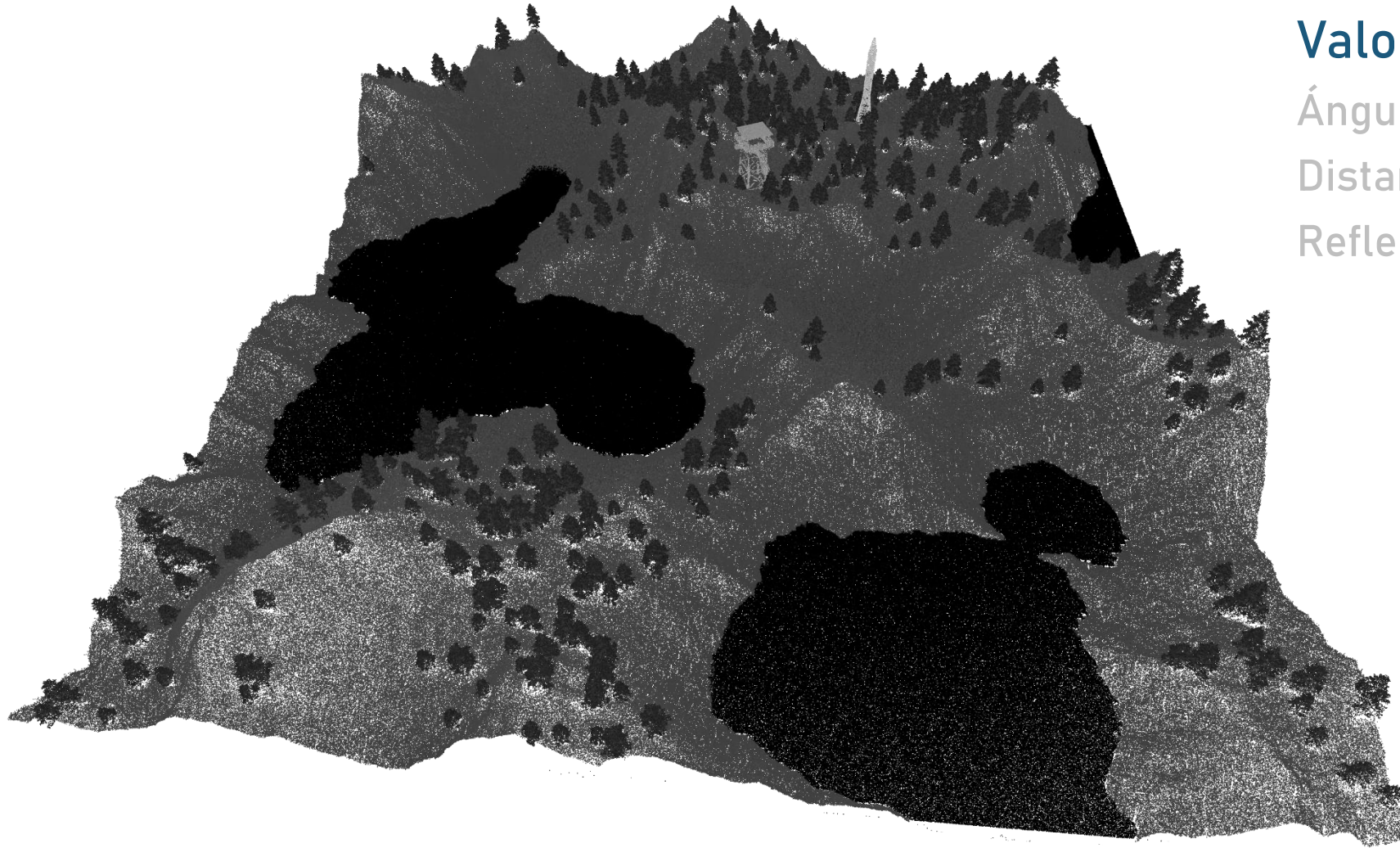












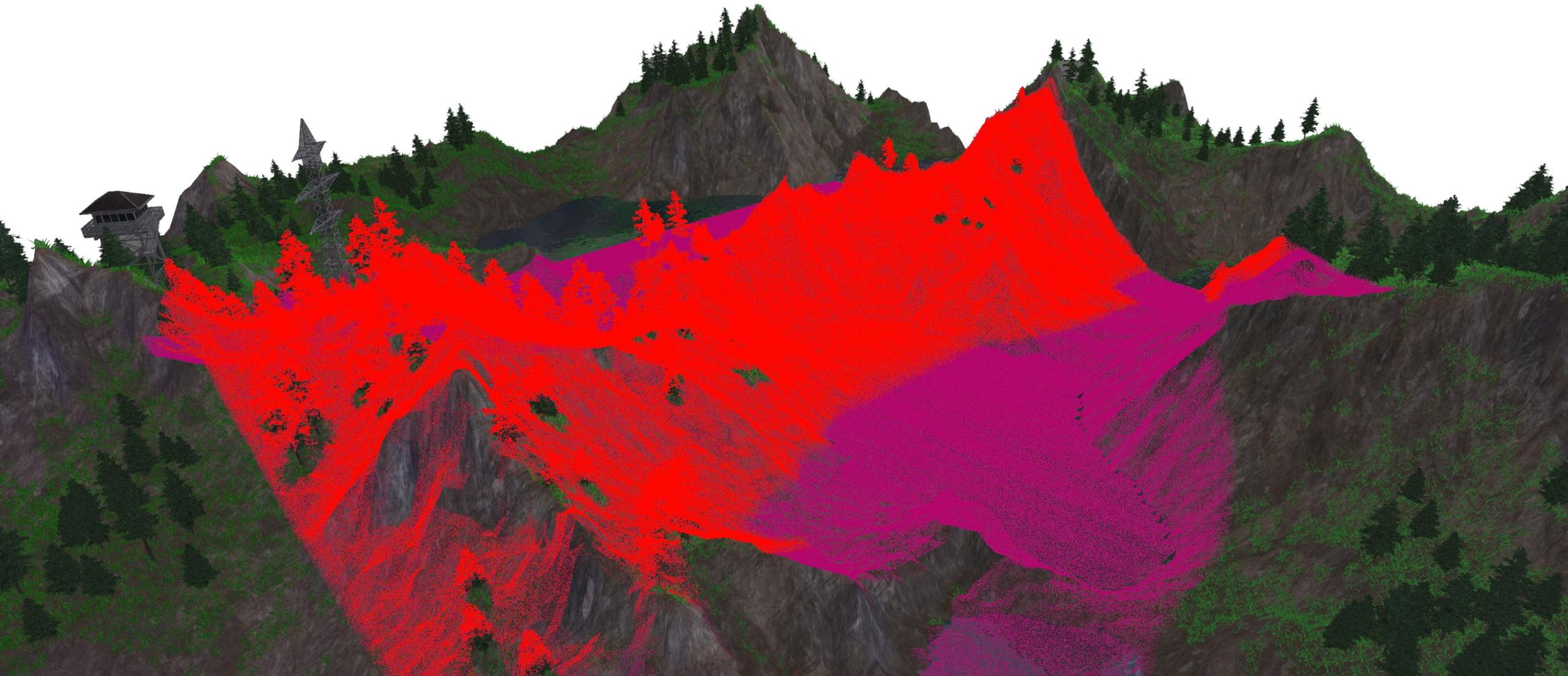
Valor de intensidad

Ángulo de incidencia

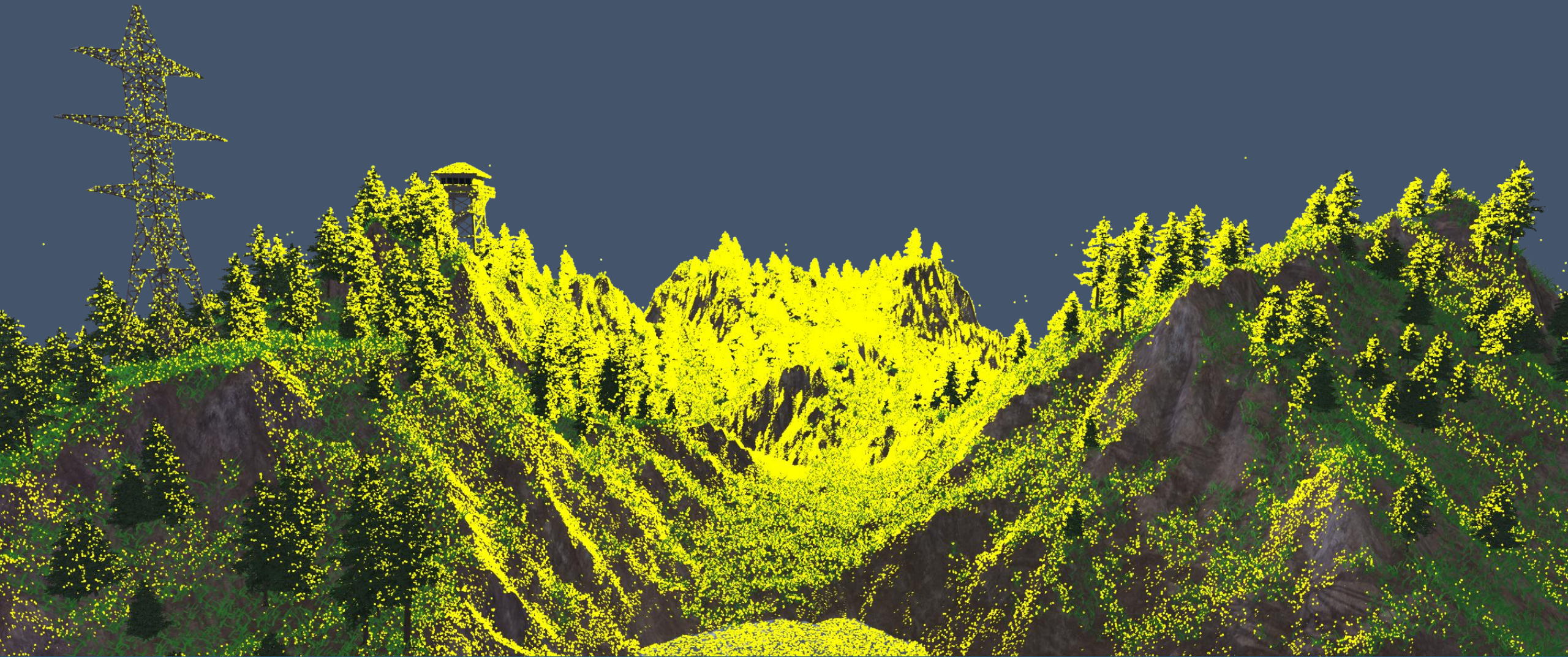
Distancia

Reflectancia de superficie

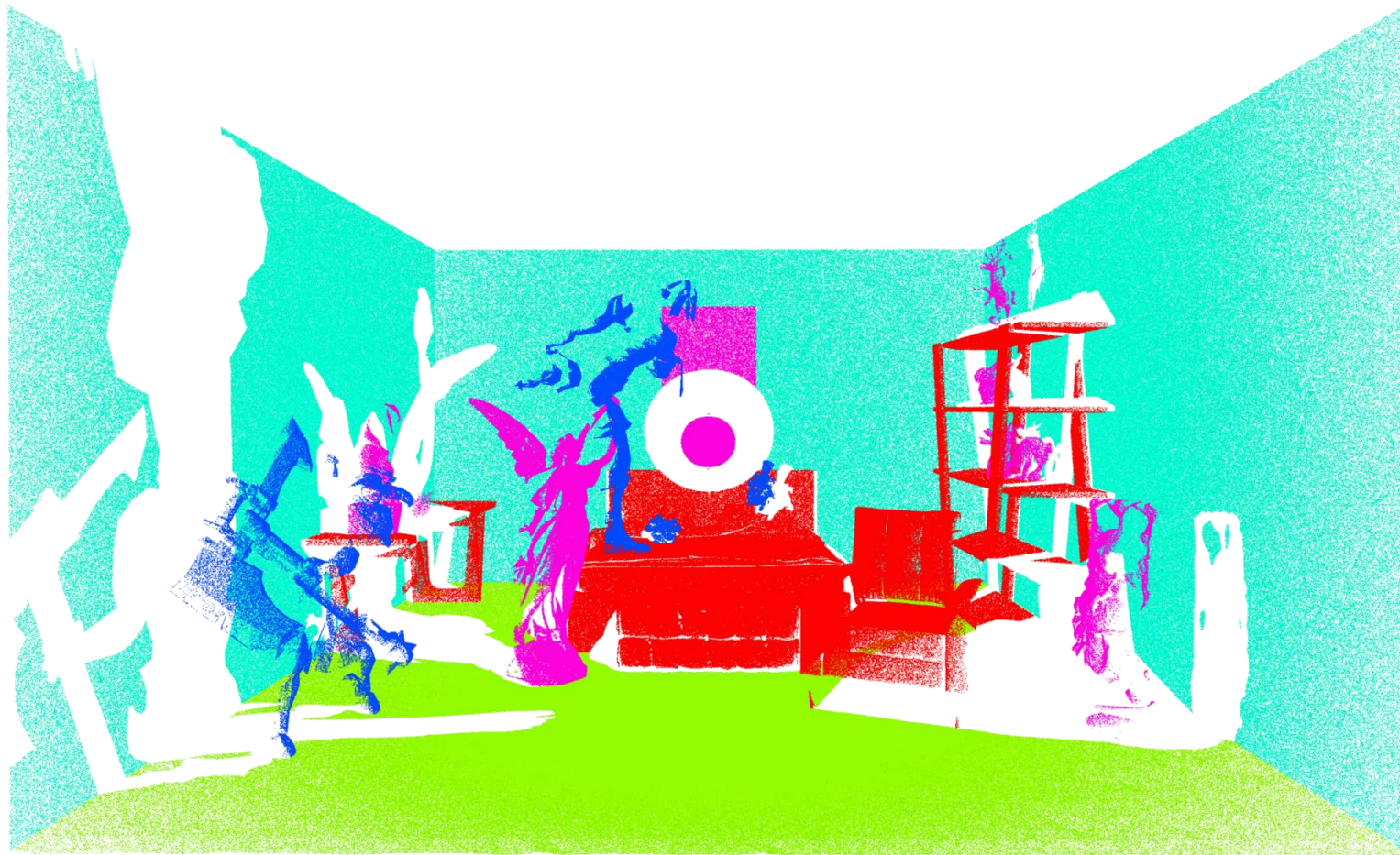












Muebles

Ornamentación

Personajes

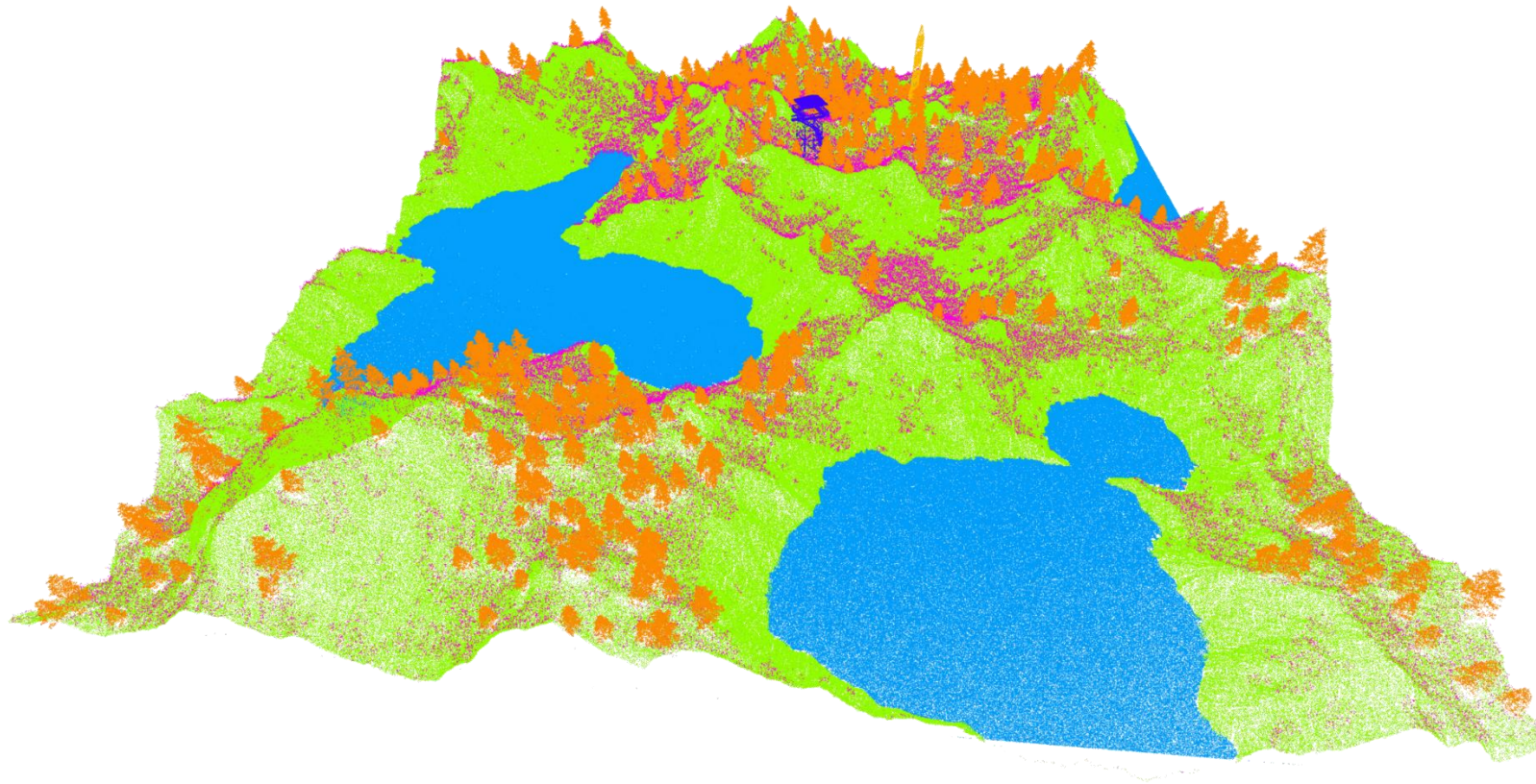
Suelo

Pared

# APLICACIÓN GRÁFICA. RENDERING DE NUBE DE PUNTOS

---

91



Ground

Low vegetation

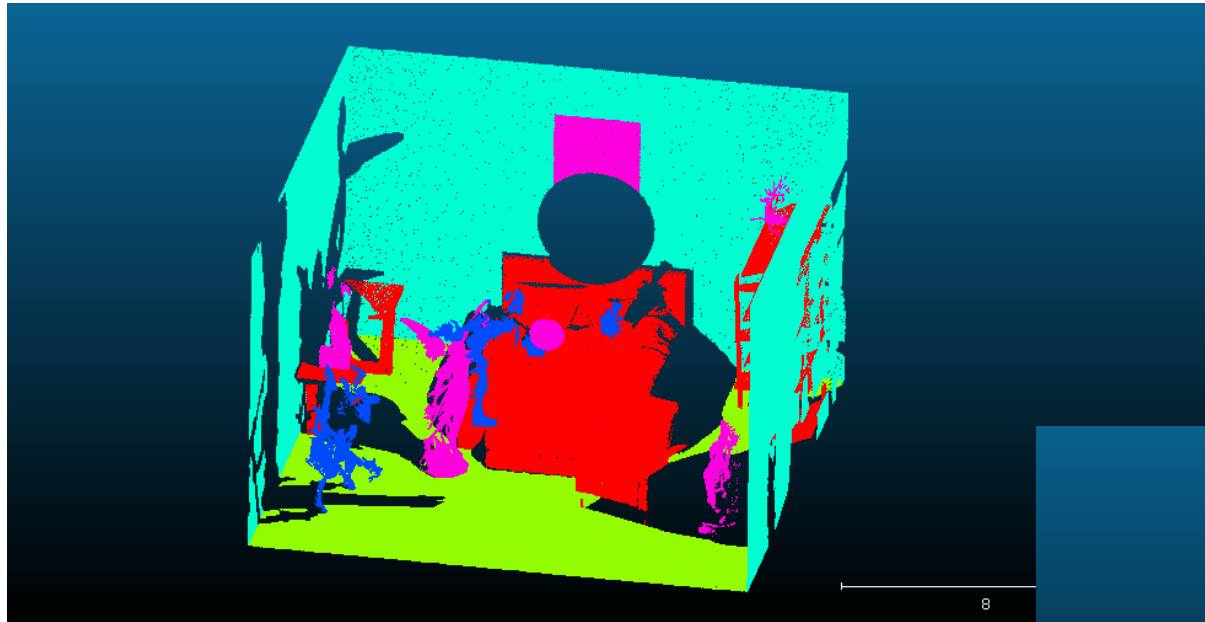
Water

Building

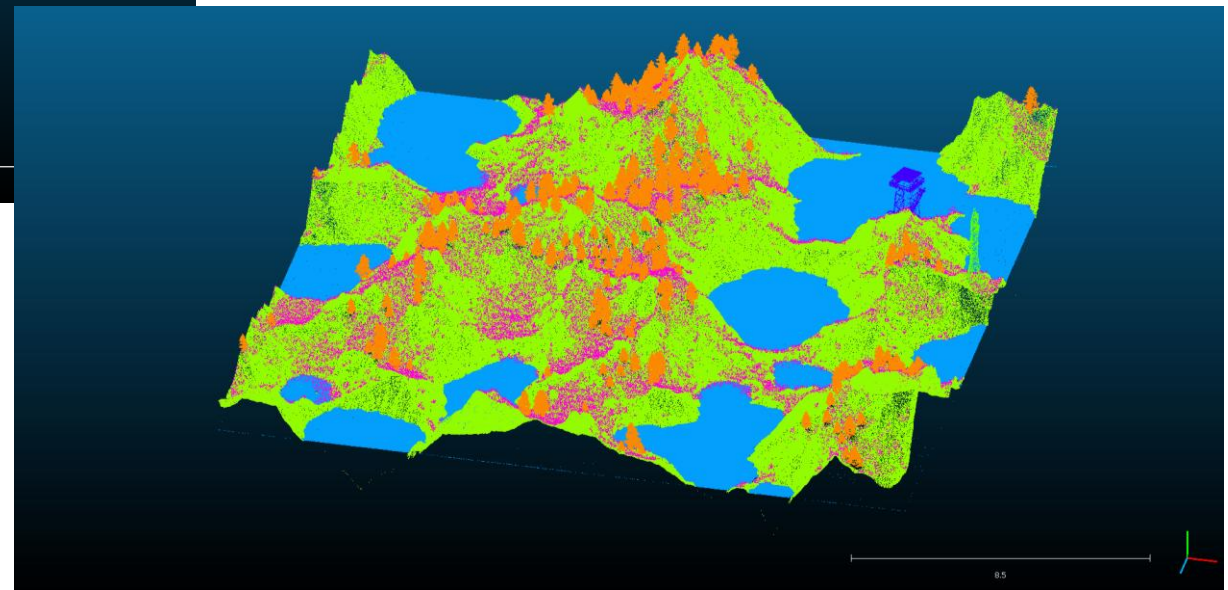
High vegetation

Transmission Tower





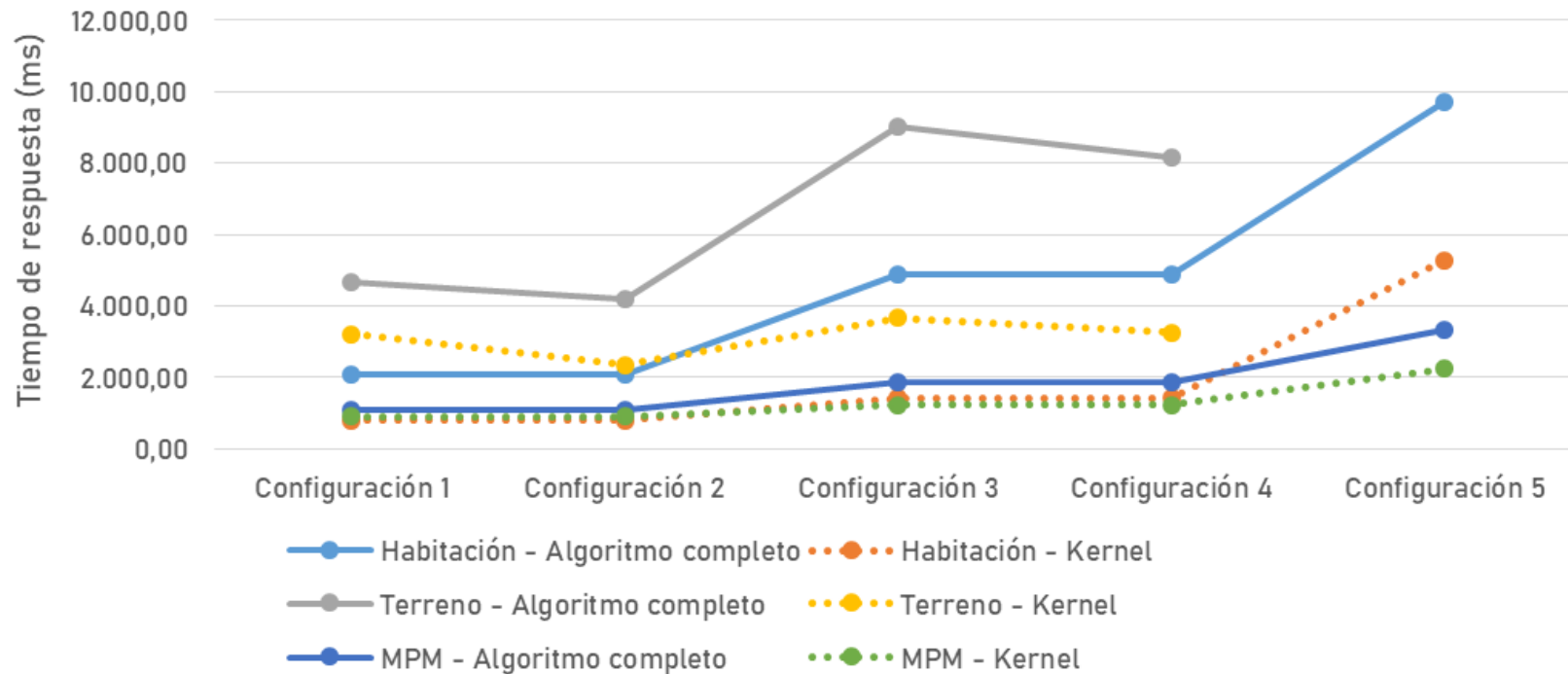
*Rendering en CloudCompare*



## Formato PLY

Flexibilidad de datos asociados  
a la nube

## Tiempo de respuesta de simulación LiDAR



**Habitación**

6.683.902 triángulos

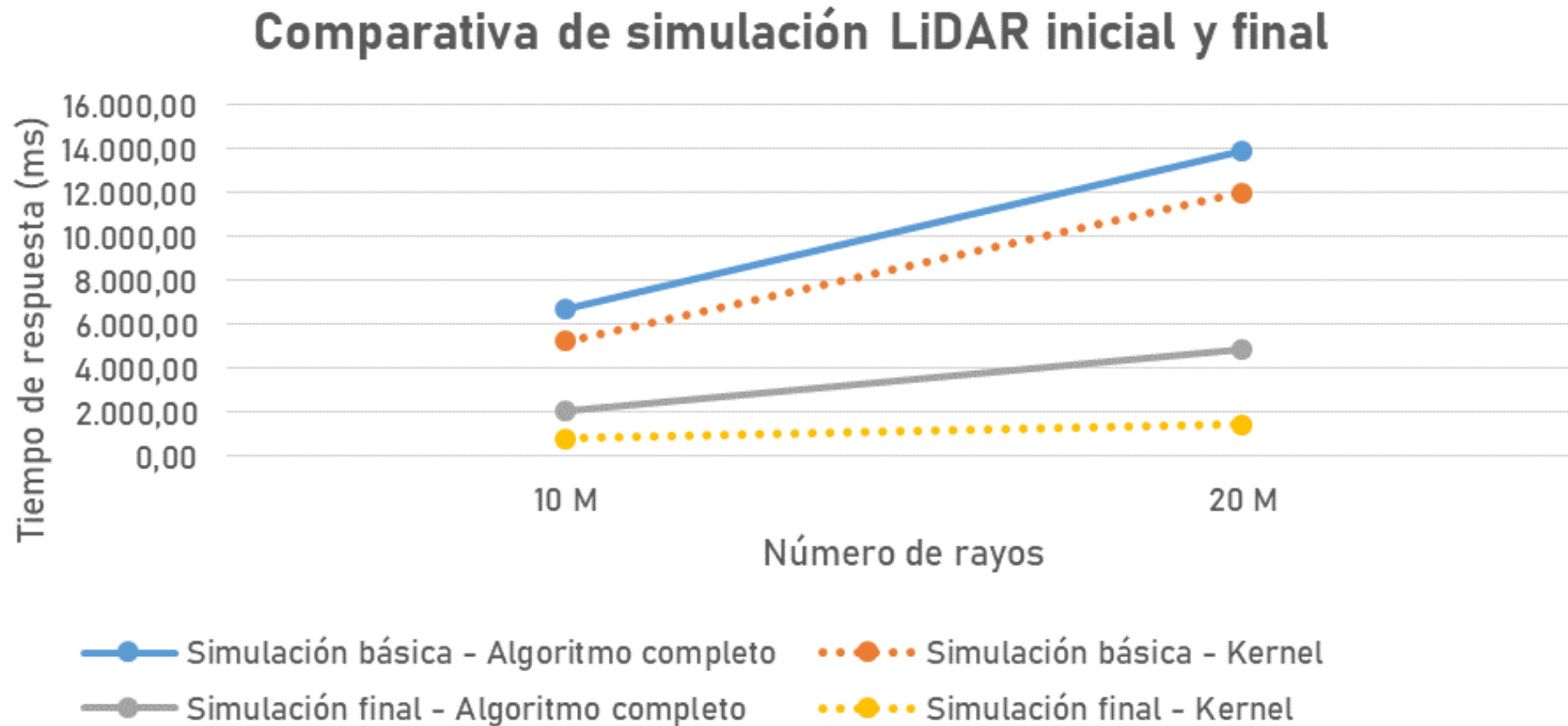
**Terreno**

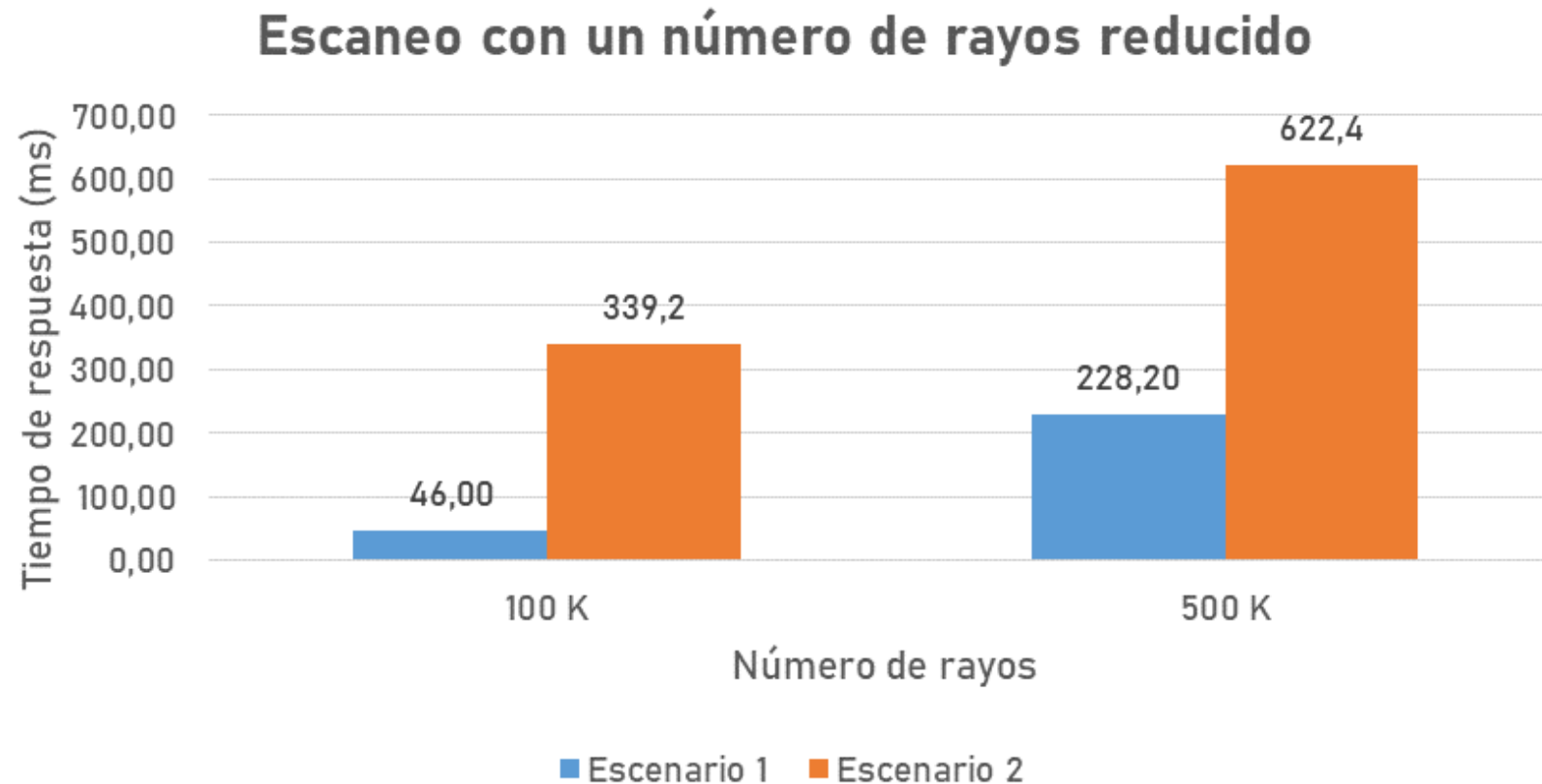
4.624.832 triángulos

**MPM**

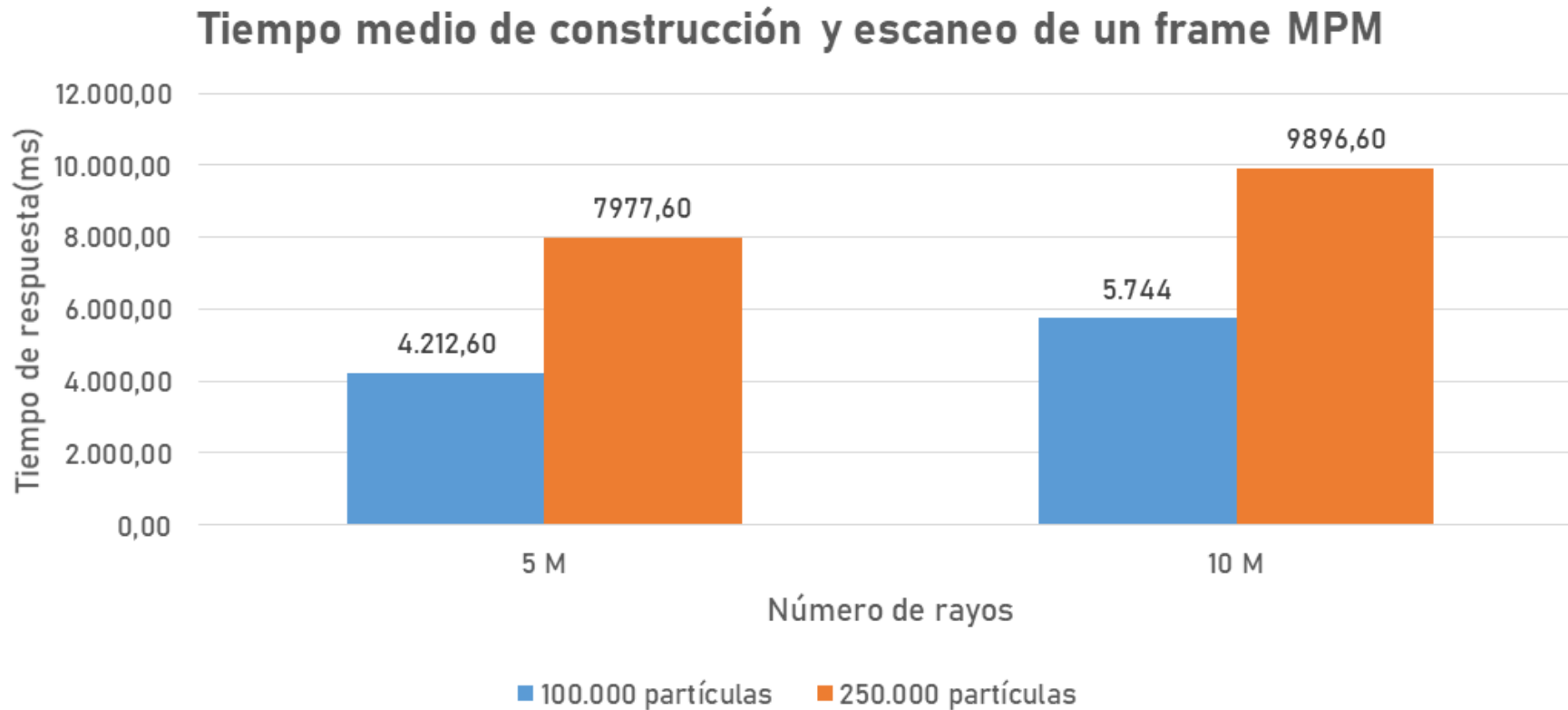
6.000.000 triángulos











**Este trabajo desarrolla un enfoque experimental orientado a solucionar un problema de gran complejidad computacional.**

- Introducción de computación paralela.
- Introducción de errores. Sea esta la principal razón del enfoque utilizado.
- Nubes etiquetadas de tamaño considerable.
- Resolución de escaneo LiDAR con un gran número de rayos y con escenas complejas.
- Posibilidad de introducir el algoritmo en un sistema en tiempo real (conducción autónoma), especialmente en escenario estáticos.

**El principal problema se encuentra aún en escenarios dinámicos.**

- ¿Actualización de BVH? ¿Consideración independiente de entidades dinámicas?



## Framework de escaneo LiDAR

- Orientado a una aplicación específica
- Herramientas de control de la escena

## Combinación de múltiples nubes de puntos

- Extensión del comportamiento de este trabajo

## Reducción de memoria empleada

- ¿Aplicabilidad de pack, unpack?
- ¿Reducción de precisión? ¿Aumento de carga de trabajo?

## Evaluación del sistema

- Mediante parámetros estadísticos (error cuadrático medio, desviación, etc)
- Medir diferencias entre el escaneo implementado y el escaneo generado por un sensor real
- Dirigir solución, con el fin de minimizar el error

## Evaluación de impacto en sistemas de Inteligencia Artificial

- ¿Las nubes de puntos recuperadas de un escaneo sintético pueden ayudar a mejorar los resultados que habitualmente se obtienen?
- Segmentación semántica