Chapter Outline

Multimodal Sensors: From Scene Rendering to Usable Data

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.3.1 Multimodal Data and Visual Images

Textbook: Sec. 3.1.4.6 (Introduction, Visual Images)

3.3.2 Depth and Segmentation Maps

Textbook: Sec. 3.1.4.6 (Depth, Segmentation)

3.3.3 Thermal Infrared and LiDAR

Textbook: Sec. 3.1.4.6

3.3.4 Observation Validity and Temporal Pairing

Textbook: Sec. 3.2.1.3

3.3 Multimodal Sensor Data Generation

3.3.1 Multimodal Data and Visual Images: Generation Principles

Unreal Engine (UE): Sensor Data from Rendering, Raycasts, and Multiple Buffers

- Physically based rendering (PBR, introduced in Section 3.2) computes reflections from material parameters and lighting, making color images resemble real camera images.
- Raycasting sends rays from the sensor along specified directions and returns hit coordinates, distances, normals, and materials. This is the basis of LiDAR point clouds.
- Multiple render buffers can provide color, scene depth, custom depth, and stencil values for the same frame, producing color images, depth maps, and segmentation maps.
- Blueprint and C++ scripts configure sensors, read results, and send them to external programs, linking sensor data with AI algorithms and control systems in a closed loop.

Rendering provides appearance, raycasts provide distance, and multiple buffers provide depth and labels.

3.3 Multimodal Sensor Data Generation

3.3.1 Multimodal Data and Visual Images: Textbook Fig. 3.5

Six Outputs: Appearance, Distance, Classes, Thermal Contrast, and 3D Points

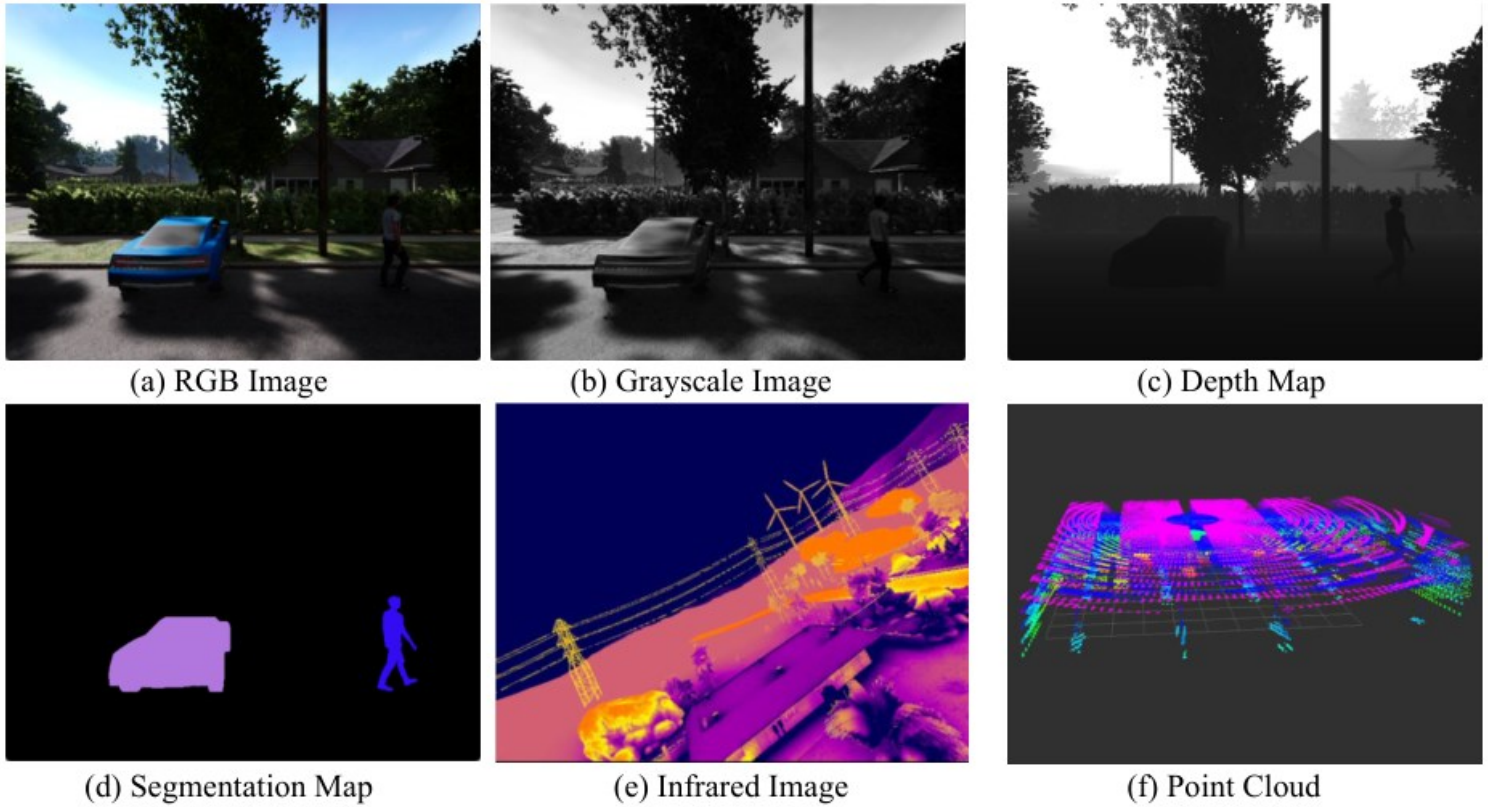


Fig. 3.5 RflySim Multimodal Sensor Data Preview

3.3.1 Multimodal Data and Visual Images: Observation Types

Observations Capture Appearance, Geometry, Labels, or Thermal Values

- **Appearance:** Color images preserve color and brightness; grayscale images describe edges and texture with one intensity channel.
- **Geometry:** Depth maps encode distance at each pixel; point clouds store spatial samples as lists of coordinates.
- **Labels:** Segmentation maps distinguish objects using scene assignments. A label table links colors to classes or instances.
- **Thermal values:** Infrared simulation maps assigned thermal values to grayscale or false color to reveal thermal contrast.
- **Task examples: Use color images to identify a car's color, depth to calculate its distance, and segmentation labels for pixel classification.**

Choose the sensor output that preserves the information your task needs.

3.3.1 Multimodal Data and Visual Images: Color and Grayscale

Color and Grayscale Preserve Different Appearance Cues



(a) RGB Image

Fig. 3.5(a) RGB Color Image

Materials and lighting set appearance.
RGB (red, green, blue) keeps colors distinct.



(b) Grayscale Image

Fig. 3.5(b) Grayscale Image

Grayscale keeps intensity edges and texture.
Color differences map to gray levels.

3.3.1 Multimodal Data and Visual Images: The Imaging Pipeline

Camera Images Pass Through Rendering, Capture, Buffering, and Output

1 Rendering Compute each pixel's color and brightness from materials and lighting using PBR.

↓

2 Capture A Camera Actor or SceneCaptureComponent2D (a 2D scene capture component) captures a frame using the configured pose, field of view, and resolution.

↓

3 Buffering Write the frame to RenderTarget2D (a 2D render target), forming a pixel array in graphics memory.

↓

4 Output Save PNG/JPEG files, or send data through shared memory, UDP/TCP networking, or video streaming for external programs to read as image arrays.

3.3.1 Multimodal Data and Visual Images: Configuration and Observation

Field of View, Resolution, Pose, and Time Define the Observation

Setting	Direct effect	Change with other conditions fixed
Field of view	Coverage on either side of the optical axis	A wider angle reduces the pixel size of a distant target
Resolution	Horizontal and vertical sample counts	More pixels at the same field of view sample detail more densely
Mounting pose	Camera origin and optical-axis direction	Translation changes viewpoint; rotation changes the target projection
Sampling time	The instant of the scene being recorded	A moving target's position changes with time

3.3.1 Multimodal Data and Visual Images: A Pinhole Example

How Field of View Changes a Target's Pixel Size

Assumptions: Ideal pinhole camera; a plane perpendicular to the optical axis at axial distance d = 10 m.

$$W = 2d\tan(\frac{\theta}{2}); \quad n = \frac{640w}{W}$$

Horizontal field of view	Width covered by 640 pixels	Pixel width of a 1 m target
90°	20.0 m	About 32 pixels
60°	About 11.55 m	About 55 pixels

θ is the horizontal field of view, W the covered width, w the target width, and n the target's pixel count.

At the same resolution and distance, a wider field of view gives the target fewer pixels.

3.3.1 Multimodal Data and Visual Images: Field of View and Pixels

Field of View: 90° → 60°; Target Width: 32 → About 55 Pixels

Narrower FOV: larger target, less surrounding context

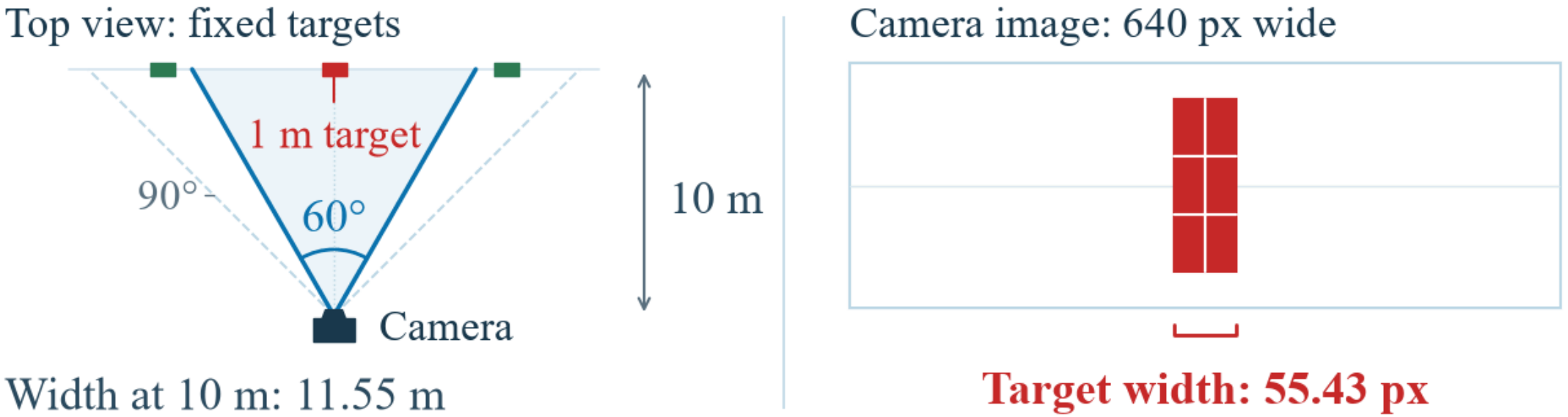


Figure: Field of view and target pixel width (ideal pinhole illustration)

Ideal pinhole model: A narrower view gives the target more pixels, while reducing visible coverage.

3.3.1 Multimodal Data and Visual Images: Varying One Setting at a Time

Compare Resolution and Field of View One Setting at a Time

Same 1 m target at 10 m	Expected pixel width	Reason for change
640 pixels wide, 90° field of view	About 32	Baseline
1280 pixels wide, 90° field of view	About 64	Twice as many samples
640 pixels wide, 60° field of view	About 55	Narrower view, larger projection scale

Assumptions: Target faces the camera; ignore distortion. Hold target, pose, and sampling time fixed.

Geometric predictions help distinguish pixel changes caused by field of view from those caused by resolution.

Chapter Outline

Multimodal Sensors: From Scene Rendering to Usable Data

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.3.1 Multimodal Data and Visual Images

Textbook: Sec. 3.1.4.6 (Introduction, Visual Images)

3.3.2 Depth and Segmentation Maps

Textbook: Sec. 3.1.4.6 (Depth, Segmentation)

3.3.3 Thermal Infrared and LiDAR

Textbook: Sec. 3.1.4.6

3.3.4 Observation Validity and Temporal Pairing

Textbook: Sec. 3.2.1.3

3.3.2 Depth and Segmentation Maps: Reading the Images

Depth Maps Encode Geometry; Segmentation Maps Encode Labels



(c) Depth Map



(d) Segmentation Map

Fig. 3.5(c) Depth Map

Grayscale is a display mapping of depth values.

Fig. 3.5(d) Segmentation Map

Scene label settings define color meanings.

Depth pixels support spatial calculations; segmentation pixels link to objects through a label table.

3.3.2 Depth and Segmentation Maps: Comparing the Same Scene

One Moving Scene Produces Color, Depth, and Labels at the Same Time

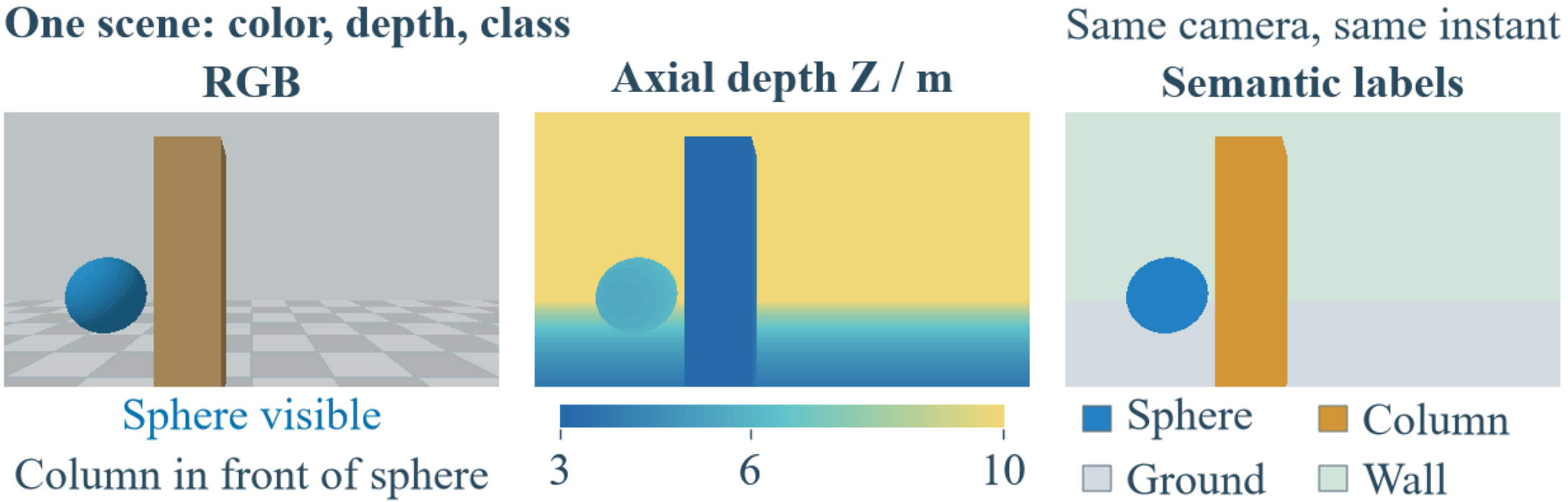


Figure: Three representations from the same camera at the same sampling time

Color describes appearance, depth records distance, and labels identify object classes.

3.3.2 Depth and Segmentation Maps: Generating Depth

Depth Maps Store Distance per Pixel; Grayscale Is for Display

- Generation: SceneCaptureComponent2D uses SceneDepth in R. Raw depth is in centimeters by default; convert to meters here.
- Display: A custom material maps depth linearly or logarithmically to grayscale. A new range or mapping changes brightness at the same depth.
- Storage: Keep floating-point depth for precise calculations and point-cloud reconstruction; export 32-bit EXR. Verify RflySim Type 2 encoding and scale in the example configuration.

- Example: z_min = 0.3 m, z_max = 12 m, z = 6 m gives g ≈ 124 (linear) or 207 (logarithmic). Here g is display grayscale; z is depth.

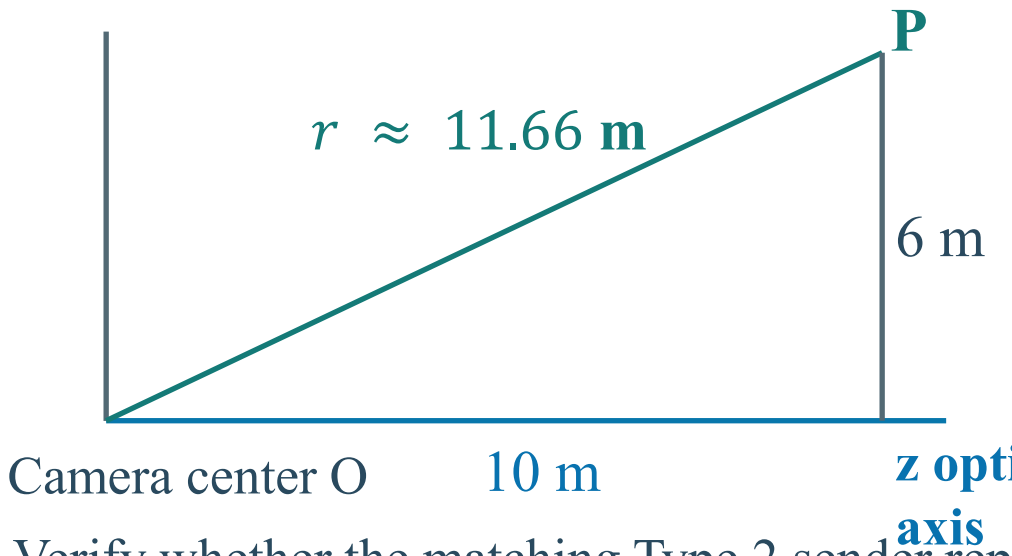
$$g_{\text{lin}} = 255\,\frac{z - z_{\min}}{z_{\max} - z_{\min}}$$

The depth array stores distance; grayscale only displays it. Different normalization changes brightness.

3.3.2 Depth and Segmentation Maps: Axial Depth and Ray Range

The Same 3D Point Has an Axial Depth and a Ray Range

Geometry: The camera's z-axis is its optical axis, and $P = (6, 0, 10)$ m.



Axial depth: $z = 10$ m
Use only the projection onto the optical axis.

Ray range: $r = \sqrt{6^2 + 10^2}$
Measure from the camera center to P.

Verify whether the matching Type 2 sender reports axial depth or ray range.

Axial depth is the optical-axis component; ray range is the straight-line distance from the camera center to the point.

3.3.2 Depth and Segmentation Maps: Decoding Raw Depth

Converting a 16-Bit Depth Array to Meters

Default: 0.001 m/code; 1200 → 1.2 m; 65535 is invalid. Check the scale against otherParams[2].

```
raw = vis.Img[i]
assert raw.dtype == np.uint16
d = raw.squeeze(2) if raw.ndim == 3 else raw
assert d.ndim == 2
meters = d.astype(np.float32) * 0.001
meters[d == 65535] = np.nan
```

The matching protocol defines the step and invalid codes; whether 0 is valid also depends on the protocol.

Quantization step is not measurement error.

Depth encoding defines both the distance scale and which codes are not valid measurements.

3.3.2 Depth and Segmentation Maps: Generating Segmentation

Generate Segmentation with Solid-Color Materials or Custom Stencil Values

- Solid-color material replacement | Temporarily replace each object class's materials with one solid color, then capture the scene | Intuitive, but requires material replacement for each class
- Custom depth and stencil | Objects write integer stencil values (8-bit, 0–255); a post-process material maps values to colors | Preserves the original materials; performance depends on the implementation
- RflySim Type 4 colors objects by CustomDepthStencilValue and outputs an H×W×3 color array in blue-green-red (BGR) channel order.

Segmentation colors come from assigned object labels. A different palette does not change the labels themselves.

3.3.2 Depth and Segmentation Maps: Classes and Instances

A Label Table Links Pixel Colors to Classes or Individual Objects

Scene object	Class label	Instance ID (example)
Vehicle A	Vehicle	17
Vehicle B	Vehicle	23
Pedestrian C	Pedestrian	31

Labeling: Type 4 maps each integer stencil label to a color.

Mapping: A class may contain several instances. Save label-to-object correspondences.

A class says what an object is; an instance says which one it is. Color only encodes the label for display.

Chapter Outline

Multimodal Sensors: From Scene Rendering to Usable Data

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.3.1 Multimodal Data and Visual Images

Textbook: Sec. 3.1.4.6 (Introduction, Visual Images)

3.3.2 Depth and Segmentation Maps

Textbook: Sec. 3.1.4.6 (Depth, Segmentation)

3.3.3 Thermal Infrared and LiDAR

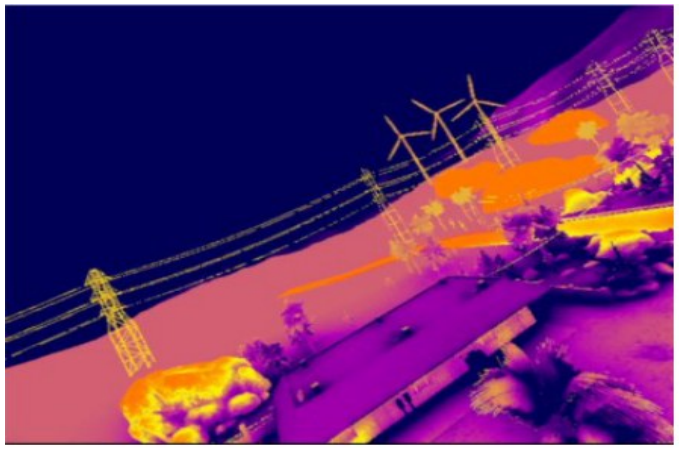
Textbook: Sec. 3.1.4.6

3.3.4 Observation Validity and Temporal Pairing

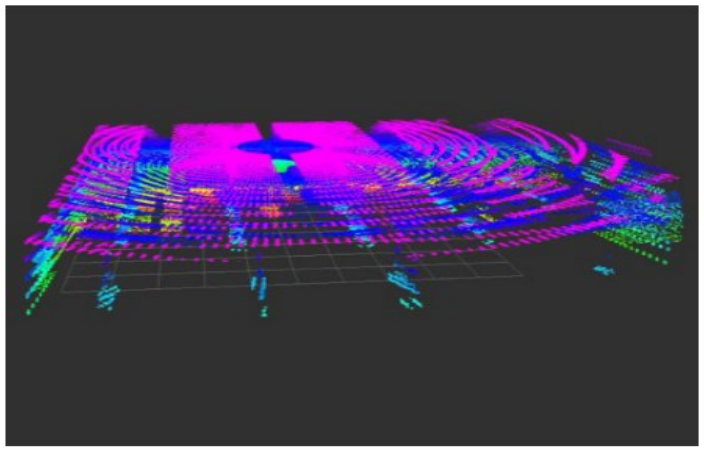
Textbook: Sec. 3.2.1.3

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: Reading the Images

Thermal Images Show Thermal Contrast; Point Clouds Record Spatial Samples



(e) Infrared Image



(f) Point Cloud

Fig. 3.5(e) Infrared Image

Assigned thermal values in gray or false color.

Fig. 3.5(f) Point Cloud

Valid points record spatial sampling results.

Thermal-value mapping supports appearance comparisons; spatial point coordinates support geometric analysis.

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: Mapping Thermal Values

Relating Thermal Values, Colors, and Temperature Measurements

- **Simplified generation:** Assign thermal values to objects or materials, then use a lookup table to map them to grayscale or false color.
- **Color-table role:** The same thermal value can appear in different colors. Changing the table changes the display, not the original thermal value.
- **Physical measurement:** Emissivity affects the energy a material emits; ambient reflections and the atmosphere affect propagation; detector response determines the output.
- **Conditions:** Infrared Types 40/41 require the corresponding paid license. Thermal contrast supports detection comparisons; quantitative thermometry also requires a radiometric model and device calibration.

A color table makes thermal values visible; temperature measurement also requires material and sensor-response models.

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: Raycasting

LiDAR from Batched Raycasts: Asynchronous Results Arrive One Tick Later

- Single trace: LineTraceSingleByChannel casts each LiDAR ray and returns its first hit's position, normal, material, and distance.
- Multi trace: LineTraceMultiByChannel returns overlaps before the first blocking hit, plus that hit. Multiple LiDAR returns need another model.
- Asynchronous trace: AsyncLineTraceByChannel submits ray queries without blocking the main thread, then retrieves results later.
- RflySim: Submit queries now and read them at the next Tick (engine update). One Tick of latency is 10 ms at 100 frames per second (FPS).
- Example: 0.2° steps give 360/0.2 = 1800 columns. With 16 channels: 28 800 rays/scan; at 10 scans/s: 288 000 rays/s.

Ray count determines point-cloud density; asynchronous queries improve efficiency but add one tick of latency.

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: The Nearest Hit

Each Ray Records the Nearest Surface; Occluded Surfaces Have No Points

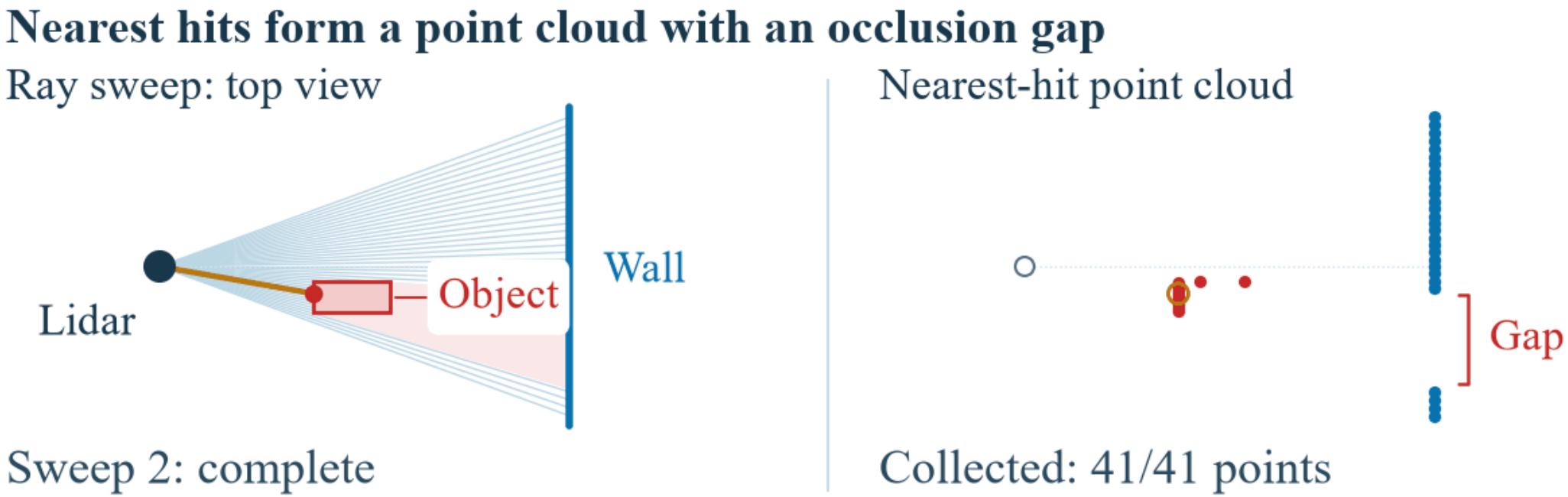


Figure: Nearest ray hits and occlusion (2D teaching illustration)

Nearest hits define the point-cloud outline; wall surfaces hidden by foreground objects produce no returns.

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: Two Point-Cloud Methods

Ray Scanning and Depth Back-Projection Produce Different Sampling Patterns

Generation method	How sample positions are determined	Missing points and density
Types 20/21 Ray scanning	Trace surfaces at horizontal and vertical angles	Angular spacing and hit results determine point density
Type 7 depth back-projection (paid edition)	Calculate 3D positions from pixels, depth, and camera intrinsics	Pixel distribution and valid depth determine point distribution

When a ray hits a foreground surface, that hit remains valid; the occluded surface behind it is not sampled.

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: Point-Cloud Encoding

Decode and Scale Point-Cloud Codes to Recover Coordinates

Stage	Data form	Processing
Encoded payload	int16; 3 or 4 values per point	Read XYZ and color code
Coordinate recovery	XYZ × (otherParams[0]/32767)	Columns 1–3: recover meters Column 4: do not scale color
Decoded array	N×4 floating-point array Check shape and dtype	Three channels: column 4 = 0 Four channels: keep color code

Coordinate reference: Type 20 uses the sensor frame; Type 21 uses the world frame.
Color encoding: Column 4 stores an RGB555 semantic color code, not an object ID.

Integer payloads reduce transfer size. Coordinate scaling restores physical values; column 4 uses a separate encoding.

3.3.3 Thermal Infrared Images and LiDAR Point Clouds: A Conversion Example

Convert Coordinate Codes to Meters and Color Codes to RGB Components

- Assumptions: Type 20 maximum range R_max = 200 m for this example. One point has x-coordinate code q = 16384 and color code c = 0x7C00 in column 4.

$$x = \frac{qR_{\max}}{32767} = \frac{16384 \times 200}{32767} \approx 100.0\ \text{m}$$

- Resolution: A code increase of 1 changes the coordinate by R_max/32767 ≈ 6.1 mm. A larger configured range gives coarser coordinates.
- Color code: The example uses (c>>10)&0x1F, (c>>5)&0x1F, and c&0x1F to extract red, green, and blue, then multiplies by 8 to restore 8-bit color. For 0x7C00, (31, 0, 0) becomes pure red (248, 0, 0).

A larger range makes each integer-code step represent a coarser spatial resolution.

Chapter Outline

Multimodal Sensors: From Scene Rendering to Usable Data

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.3.1 Multimodal Data and Visual Images

Textbook: Sec. 3.1.4.6 (Introduction, Visual Images)

3.3.2 Depth and Segmentation Maps

Textbook: Sec. 3.1.4.6 (Depth, Segmentation)

3.3.3 Thermal Infrared and LiDAR

Textbook: Sec. 3.1.4.6

3.3.4 Observation Validity and Temporal Pairing

Textbook: Sec. 3.2.1.3

3.3.4 Observation Validity and Temporal Pairing: Display and Data

Display Transformations Preserve the Raw Data's Physical Meaning

Display	Underlying data	Display versus calculation
Depth grayscale	Integer depth codes and the distance definition	Normalized brightness does not directly give meters
Segmentation color regions	Object assignments and the label table	The mapping links each color to a class or instance
Thermal false color	Thermal values and the color lookup table	Color changes may reflect only a different lookup table
Point-cloud coloring	XYZ and additional channels	Do not apply coordinate scaling to class/color codes

Save raw values and mappings to return from the display to data you can calculate with.

3.3.4 Observation Validity and Temporal Pairing: Validity

Where Occlusion, Invalid Pixels, and Missing Frames Occur

Situation	What you receive	Meaning for the algorithm
Foreground occludes background	Valid depth for the foreground surface	Only the nearest visible surface is observed
No hit / out of range	The protocol's invalid code or missing points	Mark valid pixels first, then calculate geometry
Frame lost during transfer	The current image is not fully updated	Do not treat an old buffered frame as a new sample

Occlusion changes the visible surface; invalid values mark failed measurements; a missing frame means the current data did not arrive in full.

3.3.4 Observation Validity and Temporal Pairing: Matching Times

Time Mismatch Becomes Position Error for Moving Targets

Assumptions: A car moves at a constant 5 m/s (18 km/h). The image is sampled 40 ms before the state paired with it.

$$\Delta x = v\Delta t = 5 \times 0.040 = 0.20\ \mathbf{m}$$

- **Source of error:** The image shows an earlier position. Projecting the later state shifts the target's predicted pixel position.
- **Pairing method:** In a common time domain, select or interpolate the state using the sampling timestamp;
  the receive time only indicates when data arrived.

At the same speed, a larger time mismatch causes a larger position discrepancy between the image and state.

Classroom Exercises

Four Exercises in Calculation and Data Interpretation

- Pixel size: For a camera 1280 pixels wide with a 60° field of view, how many pixels span a 1 m target at 20 m? What does comparison with 640 pixels, 60°, and 10 m show?
- Depth definition: A camera faces a flat wall 10 m away; an edge ray is 45° off the optical axis. Predict center/edge axial depth and ray range. Compare with Type 2 raw codes and configuration.
- Ray count: A LiDAR scans a full revolution at horizontal intervals of 0.1° with 32 channels. How many rays are in one scan? At 50 FPS, how late does the asynchronous result arrive?
- Time pairing: A pedestrian walks at 1.5 m/s. The image is sampled 100 ms before the paired state. What is the position error, and how can it be avoided?

Correct units alone do not ensure a correct result; depth and pixel problems also require the right geometry.

Reference Answers

Answers and Reasoning for the Four Exercises

$$W = 2 \times 20 \tan 30^\circ \approx 23.09\ \text{m}, \quad n = \frac{1280}{W} \approx 55$$

- Pixel size: Distance and resolution increase by the same factor, so the target still spans about 55 pixels. The field of view remains 60°.
- Depth definition: Axial depth stays about 10 m across a flat wall facing the camera; ray range is about 14.14 m at an edge 45° horizontally off axis. Check scale, invalid codes, and capture source first, then use measurements to identify the Type 2 definition.
- Ray count: 360/0.1×32 = 115 200 rays. Asynchronous results arrive one Tick later, which is 20 ms at 50 FPS.
- Time pairing: Δx = 1.5×0.1 = 0.15 m. Select or interpolate the state at the image's sampling timestamp; do not pair by arrival time.

The same pixel width can arise under different conditions; check both distance and resolution.

Section Summary

Four Observation Types: Generation, Reading, and Validation

- Generation: Color images use physically based rendering and scene capture; depth comes from scene depth; segmentation uses solid-color materials or stencil values; point clouds use raycasts or depth back-projection.
- Reading: Check type, shape, dtype, and units first. Convert raw depth codes using their scale and identify invalid codes. Point-cloud XYZ in vis.Img are already in meters; do not scale them again.
- Validation: Predict pixel width from field of view and resolution; use a wall facing the camera to distinguish axial depth from ray range; pair images and states by sampling timestamps.
- Common errors: Treating display brightness as distance, BGR as RGB, array index as SeqID (Unit 3.4.4), arrival time as sampling time, or thermal-image color as measured temperature.

First ask how sensor data are generated and encoded; then check them using geometry and timing.