Chapter Outline

From the Example Catalog to Three Practice Tasks

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

3.6.1 Entry Points and Task Selection

Textbook: Sec. 3.3.1

3.6.2 Interface Example Catalog

Textbook: Sec. 3.3.1

3.6.3 Basic and Advanced Example Catalogs

Textbook: Sec. 3.3.1

3.6.4 Representative Cases and Practice Roadmap

Textbook: Sec. 3.3.2

3.6.1 Entry Points and Task Selection: Example Resources

Ground Placement: Entry Point and Comparison Objects

- **Entry point:** 0.ApiExps/e5_UEMapCtrl/3.TargetPlace/Readme.pdf.
- **Script:** code/TargetPlaceDemo.py lists the standard and ground-placement calls.
- **Comparison objects:** MountainTerrain, character style 30, instance IDs 100 and 101.

Record the input coordinates, run command, and foot positions of both characters. Check each object's x and y first; a height difference between different locations cannot be attributed directly to the API.

The entry point, script, two application programming interface (API) calls, and placement records form a reproducible chain.

3.6.1 Entry Points and Task Selection: Directories and Learning Tasks

Three Directory Tiers, Three Starting Points

- **0.ApiExps:** learn one operation, one command, or one returned data structure.
- **1.BasicExps:** turn existing inputs into assets and scenes that can be created and saved.
- **2.AdvExps:** add asset processing, Blueprints, or geographic data; choose 3.CustExps when its requirements are met.

Example route: first run the ground-placement example at the API tier and compare character placement. Then save and reload a layout at the basic tier. Keep each input with its corresponding result.

Choose an example for the question; its dependencies define the scope of practice.

3.6.1 Entry Points and Task Selection: Reading the Readme

Follow 3.TargetPlace from Inputs to Observations

- **Inputs:** MountainTerrain, character style 30, and coordinates read from Click Point.

- **Prepare:** open the 3D application with ue4.bat, press M for the mountain map, and double-click the ground to read z.
- **Run:** edit TargetPlace.py as instructed in the Readme, then run it through Python38Run.bat.

Outputs: check the specified z for ID 100 and the ground-placement result for ID 101 separately. Record the map, coordinates, commands, and view; a height difference at different x positions is not the API effect.

Record object IDs, coordinates, and foot positions before comparing placement methods.

3.6.1 Entry Points and Task Selection: API Terminology

Ground Placement: Commands, Styles, and Instances

- **Command:** use sendUE4Cmd to switch to MountainTerrain; wait for the map to load before placement.
- **APIs:** sendUE4Pos supplies z; sendUE4Pos2Ground requests ground placement.

- **Style / instance:** vehicleType=30 selects the character style; IDs 100 and 101 identify the two characters.

Message check: match each call's ID and coordinates to its object in the 3D application. The 3D application handles the ground-placement flag; the standard call does not automatically query terrain.

The command selects the map, the style selects the character, the instance ID selects the object, and the API determines placement.

3.6.1 Entry Points and Task Selection: Operation Entry Points

Operations, Keyboard Shortcuts, and Commands

0.ApiExps

e0_DevToolsUsage	Tool roles, input formats, and entry points for creating assets.
e0_RflySim3DUsage	Starting, displaying, and operating the scene application.
e1_KeyboardAPI	Change views, selections, and scene objects with keyboard shortcuts.
e2_CommandAPI	Send scene commands through the console or a BAT file.

Manual operations support exploration; scripted calls make scene changes reproducible.

3.6.1 Entry Points and Task Selection: Initialization and Objects

Initialization Settings and Object Motion

0.ApiExps

e3_InitAPI	TXT commands execute at startup or map change. After editing, trigger the corresponding load again.
e4_UAVCtrl	Send inputs to an existing ID at runtime to update component actions or whole-object attachment.

Check: reload the map to confirm the initial state, then update a control input for the same ID and compare the motion.

Startup commands set initial conditions; control interfaces update objects.

3.6.1 Entry Points and Task Selection: Environment and Data

Map, State, Geographic Data, and Effects

0.ApiExps

e5_UEMapCtrl	Object placement, terrain data, and ground-following trajectories.
e6_RflySim3DCtrlAPI	Model state control, display, and structured data return.
e7_RflySim3DGIS	Geographic origins and Cesium scene integration.
e8_RflySim3DEffect	Environmental effects, communication links, and action effects.

Scene control and data return use different interfaces.

3.6.1 Entry Points and Task Selection: Collisions and Scene Saving

Collision Events and Scene Editing

0.ApiExps

e9_RflySim3DCollision	Collision events and returned data.
e10_BuildCusScene	Create and save a custom layout in the scene interface.

Check: save a layout and reload the same scene, then reproduce the contact sequence. Match collision records using event times and object IDs.

Event records and scene layouts are different outputs.

3.6.1 Entry Points and Task Selection: Three Classroom Tasks

Connect the Catalog through Three Tasks

- **Ground placement:** e5_UEMapCtrl/3.TargetPlace → character positions and foot placement.
- **Scene saving:** e0_StarterContent/5.UE_CustomImport → layout JSON (JavaScript Object Notation) and associated assets.
- **Data acquisition:** e6_RflySim3DCtrlAPI/10.GetCamObjDemo → state fields and two red-green-blue (RGB) image streams.

Each task has an entry point, parameters, and an observable output.

Chapter Outline

From the Example Catalog to Three Practice Tasks

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

3.6.1 Entry Points and Task Selection

Textbook: Sec. 3.3.1

3.6.2 Interface Example Catalog

Textbook: Sec. 3.3.1

3.6.3 Basic and Advanced Example Catalogs

Textbook: Sec. 3.3.1

3.6.4 Representative Cases and Practice Roadmap

Textbook: Sec. 3.3.2

3.6.2 Interface Example Catalog: Tools and Assets

Minimum Reproducible Outputs from Tool Examples

0.ApiExps/e0_DevToolsUsage

1.UEBlueprintUsage	Run the box Blueprint and record the rotation node and result.
2.3dsMaxUsage	Create a box, export box.fbx, and check its dimensions.
3.CesiumForUnrealUsage	Save the QuickStart level and geographic loading configuration.
4.SketchUpUsage	Complete a house model and save house.skp.

Keep a specific output from each tool to connect asset creation with scene execution.

3.6.2 Interface Example Catalog: Acquiring External Assets

Records for Scene Transfer and External Assets

0.ApiExps/e0_DevToolsUsage

5.TwinmotionUsage	Save house.tm; export the scene using the branch for your tutorial version.
6.EpicFabUsage	Select one asset and record its source, license, and target format.

- Deliver the source file, exported output, referenced textures, and the corresponding software and plugin versions.

Source assets, exported files, and version records allow external assets to be imported again.

3.6.2 Interface Example Catalog: Looking Up and Calling Commands

Commands, Enumerations, and BAT Calls

0.ApiExps/e2_CommandAPI

1.command	Send one command in the defined parameter order and observe the corresponding scene state change.
2.enumerate	Look up the field's enumeration before using it in a command; a number's meaning depends on the field.
3.BatAPI	Put verified parameters in a BAT file; check the map and view after restarting.

Enumerations define values; commands and scripts execute actions.

3.6.2 Interface Example Catalog: TXT Trigger Timing

Startup TXT and Map TXT Files

0.ApiExps/e3_InitAPI

1.TXTAllCrtlScript	Read RflySim3D.txt at startup.
2.TXTMapCrtlScript	Execute the corresponding TXT when switching to the specified map.

Compare: restart to observe the global TXT, then switch to the specified map to observe its TXT. Record the map and display state each time.

The trigger determines when preset commands take effect.

3.6.2 Interface Example Catalog: Startup Configuration Example

Set Startup Conditions Line by Line in RflySim3D.txt



Figure: Scene started from TXT presets

- **File:** RflySim3D.txt, one command per line.
- **Map:** RflyChangeMapbyName Changsha.
- **Display:** 1280×720 window; frame-rate limit 25.

Preset commands set the map and display parameters at startup.

3.6.2 Interface Example Catalog: Actual TXT Parameters

Store Commands and Display Parameters in a Preset File

```
RflyChangeMapbyName Changsha
r.setRes 1280x720w
t.maxFPS 25
stat fps
```

Map: Changsha must be an asset that the current application can load.
Display: 1280×720; w selects windowed mode; 25 is the frame-rate limit; stat fps displays statistics.
File: 1.TXTAllCrtlScript/RflySim3D.txt, with one command per line.

Map names and display parameters can define the initial conditions of an experiment.

3.6.2 Interface Example Catalog: Components and Object Attachment

Actuator Control and Whole-Object Attachment

0.ApiExps/e4_UAVCtrl

1.ActuatorBinding	XML (Extensible Markup Language) defines parent-driven child motion; see the tilt example in 3.5.
2.ActuatorCtrl	Assign extended actuator inputs according to the model's conventions.
3.VehicleAttachSim	Define whole-object attachment in Simulink.
4.VehicleAttachPy	Specify the child object, parent object, and mode in Python.

Internal component indices and scene object IDs identify different targets.

3.6.2 Interface Example Catalog: Whole-Object Attachment

Define Whole-Object Attachment with Scene IDs



Figure: Earlier attachment demo with two unmanned aerial vehicles (UAVs); the current example uses a character and a UAV

- **Call:** sendUE4Attatch(2,1,3).
- **Objects:** attach object 2 to object 1; mode 3 attaches both position and attitude.
- **Entry point:** e4_UAVCtrl/4.VehicleAttachPy.

Object attachment uses instance IDs; internal binding uses actuator indices.

3.6.2 Interface Example Catalog: Placement and Terrain

Object Placement and Terrain Data Entry Points

0.ApiExps/e5_UEMapCtrl

1.TargetCreateKey	Create targets with keyboard shortcuts; XML defines the center-to-ground height.
2.TargetCreatePy	Create a target at a specified position and update its height.
3.TargetPlace	Two calls: standard positioning and engine-based ground placement.
4.GetTerrainMAT	Read a height PNG and query terrain in MATLAB.

Ground placement and height-matrix reading are separate processes.

3.6.2 Interface Example Catalog: Ground-Following Trajectories

Single-Vehicle, Multi-Vehicle, and Heterogeneous Trajectories

0.ApiExps/e5_UEMapCtrl

5.TrajGen	Use terrain elevations to generate a ground-following trajectory for one model.
6.TrajGenMulti	Homogeneous vehicles share a trajectory-generation mechanism.
7.HeterTrajGenMulti	Heterogeneous models receive User Datagram Protocol (UDP) states by their own IDs.
8.TenCarCircleCtrl	Combine a ten-vehicle circular trajectory with terrain heights.

Send trajectory data by instance ID; terrain supplies height conditions.

3.6.2 Interface Example Catalog: Creation and Data Return

Model Creation and State Data Entry Points

0.ApiExps/e6_RflySim3DCtrlAPI

1.UECtrlPy	Switch maps, create models, and apply additional motion.
10.GetCamObjDemo	Read returned states and acquired images separately.
2.LoadModelsOnBat	Specify models and the initial scene in a startup script.
3.LoadModelsByTxt	Use RflyLoad3DFile to load model control commands.

Creation sends states; return-data requests obtain structured data.

3.6.2 Interface Example Catalog: Trajectories, Viewports, and Feedback

Trajectories, Data Return, Viewports, and Damage Display

0.ApiExps/e6_RflySim3DCtrlAPI

4.TrajDemo	Update model position and attitude along a trajectory.
5.UE4RecUE4APITest	Request and receive scene return messages.
6.RflySim3DViewPortDemo	Control the viewing viewport and display region.
7.DamageModel	Model damage states and their visible effects.

Object state and the viewing window can be controlled separately.

3.6.2 Interface Example Catalog: Messages and Positions

Message Display and Position Reading

0.ApiExps/e6_RflySim3DCtrlAPI

8.RflySim3DMsgDispDemo	Display specified message content and styles.
9.RflySim3DPosGet	Read position data from the scene.

Check: move an object with the same ID and verify that its returned position updates. Screen labels aid observation; numerical records support calculations.

On-screen text and position structures carry different information.

3.6.2 Interface Example Catalog: Geographic Origins

Cesium and Oblique Model Entry Points

0.ApiExps/e7_RflySim3DGIS

1.CesiumPlugin	Load geographic tiles and set the map reference origin.
2.ObliModelMap	Convert oblique models and configure their geographic placement.

Application: record the source data reference, map origin, and loading address together.

Geographic assets require a common coordinate reference.

3.6.2 Interface Example Catalog: Connections and Vehicle Effects

Communication Links, Tunnels, and Vehicle Effects

0.ApiExps/e8_RflySim3DEffect

1.Comm	Link style and duration represent a connection.
2.Tunnel	Generate segmented tunnel geometry from a coordinate sequence.
3.EastBomberSU24	Extended model inputs trigger visible effects.
4.RoadRollerDemo	Create a vehicle and send ground-following motion states.

Coordinates, styles, or event parameters drive effect geometry.

3.6.2 Interface Example Catalog: Environment and Model Actions

Weather, Model Actions, and Event Effects

0.ApiExps/e8_RflySim3DEffect

5.StandardVtol	Update a vertical takeoff and landing model from its state.
6.Weather	Weather parameters change the appearance of the sky and precipitation.
7.3DlasersBomb	Custom assets map hit events to visual effects.
8.RobotDog	Drive joints with extended inputs according to the model protocol.

A model input takes its meaning from the protocol for that asset.

3.6.2 Interface Example Catalog: RflySim3D Collision Example

RflySim3D Ray Queries Report Collisions to CopterSim

Example: [InstallationDirectory]\RflySimAPIs\3.RflySim3DUE\0.ApiExps\e9_RflySim3DCollision

Detection: query distances in multiple directions at a set frequency. A collision is detected when the distance between the vehicle and an object is below the threshold.

RflySim3D ray queries → **Distance below threshold collision information** → **CopterSim receives results**

Step 1: double-click ue4.bat to open RflySim3D, then double-click Python38Run.bat. Enter python collsion.py and observe collision messages for the ground, static objects, and vehicles.

Step 2: Run CopterSimColl.bat to start software-in-the-loop (SITL; SIL in the textbook) for two vehicles, then run python CopterSimColl.py. Hold Ctrl and scroll the mouse wheel to enlarge them until they collide and are pushed apart.

Edition limit: Press P to enable the physics collision engine (Full Edition or higher only).

3.6.2 Interface Example Catalog: Current API Reference

Trace an Example Call to Its SDK Definition

- **Task entry point:** Readme.pdf in the installation package gives the steps; scripts and configuration in the same directory provide the actual inputs.
- **Documentation:** the UE4CtrlAPI reference in 3.4, Distributed Simulation Architecture and Data Interfaces.
- **Source entry point:** RflySimAPIs/RflySimSDK/ue/UE4CtrlAPI.py.
- **Call trace:** TargetPlaceDemo.py → sendUE4Pos2Ground → build packet → send via UDP.

Version differences: sendUE4PosFull explicitly uses north-east-down (NED) coordinates in meters. Axis and time descriptions for sendUE4PosNew conflict; retain the supplied example values initially.

The example gives the task sequence; the software development kit (SDK) defines current parameters and return data.

Chapter Outline

From the Example Catalog to Three Practice Tasks

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

3.6.1 Entry Points and Task Selection

Textbook: Sec. 3.3.1

3.6.2 Interface Example Catalog

Textbook: Sec. 3.3.1

3.6.3 Basic and Advanced Example Catalogs

Textbook: Sec. 3.3.1

3.6.4 Representative Cases and Practice Roadmap

Textbook: Sec. 3.3.2

3.6.3 Basic and Advanced Example Catalogs: X680 Asset Chain

From X680 FBX Files to a Loadable 3D Model

- **Inputs:** VehicleModel/DroneyeeX680Body.FBX and rotor-blade FBX files.

- **Process:** check hierarchy, pivots, and materials in Unreal Engine (UE), then Cook the runtime assets.
- **Outputs:** DroneyeeX680Body.uasset/.uexp and DroneyeeX680.xml.

Use: Copy.bat deploys to RflySim3D/Content. MeshPath in the XML points to the body asset; check the rotors after creation.

Match the FBX files, cooked assets, XML paths, and deployment directory.

3.6.3 Basic and Advanced Example Catalogs: Basic Asset Overview

Four Basic Asset Groups

1.BasicExps

e0_StarterContent	Scene creation, cooking, origins, and saving.
e1_CusLoadDroneyeeX680	Ready-made FBX files, cooked assets, and XML templates.
e3_RflySim3DTerrainPcd	Terrain grid sampling, queries, and PLY output.
e4_CusLoadObj	Import images, runtime meshes, and XML objects.

Obtain one loadable asset first, then change its parameters.

3.6.3 Basic and Advanced Example Catalogs: Scene Version Branches

UE4 and Current UE5 Scene Workflows

1.BasicExps/e0_StarterContent

1.UE4StarterContent	UE4.27 scene cooking and terrain-file delivery.
2.UE5StarterContent	The current UE5.8 workflow uses the RflyDemo base scene.
3.MaterialContent	Trace material references, cooking, and display issues.
4.SetSceneOrigin	Distinguish asset pivots, world origins, and geographic references.

The same directory topic may have different workflows for different versions.

3.6.3 Basic and Advanced Example Catalogs: Scene Generation and Saving

Connect Python Creation with Saving in the Interface

1.BasicExps/e0_StarterContent

5.UE_CustomImport TargetCreate.py creates objects; save JSON in the interface.

- **Inputs:** ChallengeMap and the pos and ang lists in TargetCreate.py.
- **Process:** send object states in a loop, then save this generated result in the interface.
- **Outputs:** layout JSON and associated files with the same base name; retain this run's coordinates for the randomized square frames.

The creation script sends object states; saving writes the layout file.

3.6.3 Basic and Advanced Example Catalogs: Object Format Branches

Images, Runtime Meshes, and XML Objects

1.BasicExps/e4_CusLoadObj

1.loadPng	Use a calibration pattern as a scene asset with defined dimensions.
2.loadSimpleMesh	Import a local mesh through Unreal Motion Graphics (UMG) and save the asset repository configuration.
3.loadMeshXML	Register FBX assets with XML after UE4 cooking.

Three input types produce different loading configurations.

3.6.3 Basic and Advanced Example Catalogs: Advanced Asset Workflows

Workflows, Scenes, Blueprints, and Geographic Assets

2.AdvExps

e0_DevWorkflow	Development workflow for assets, XML, and runtime loading.
e1_TwinmotionContent	Transfer scenes and referenced assets with Datasmith.
e2_BlueprintModel	Blueprints for Rfly events, component actions, and effects.
e3_CusGIS	OSGB conversion and geographic scene configuration.

Each advanced entry point adds a specific asset dependency.

3.6.3 Basic and Advanced Example Catalogs: Point Clouds and Scanning

Point-Cloud Assets and Reconstruction from Scans

2.AdvExps

e4_LidarPointClound	Import LAS point clouds, build collision geometry, and Cook assets.
e5_LiDAR-UE	Scan objects or interiors with a phone, repair the result, and import it into UE.

For terrain elevation sampling, see 3.5. Physical scanning requires suitable acquisition equipment.

Choose separately between capturing scans and importing existing point clouds.

3.6.3 Basic and Advanced Example Catalogs: Connecting Asset Outputs

From Cooked Assets through XML to Runtime Loading

2.AdvExps/e0_DevWorkflow

1.3DMap_Model_Import_Dev — Connect cooked assets, XML indices, and scene loading.

- **Inputs:** UE assets for the target engine and referenced dependencies such as materials and textures.
- **Process:** Cook creates runtime assets; XML registers styles and MeshPath.
- **Outputs:** Content assets, XML, and an instance-creation call, with matching paths across all three.

Connect runtime assets, index configuration, and instance states layer by layer.

3.6.3 Basic and Advanced Example Catalogs: Blueprint Implementation Routes

Rotors, Fixed Wings, Animation, and Collision Events

2.AdvExps/e2_BlueprintModel

1.BlueprintBuild	Store the input in a speed variable; Tick drives the X680 rotors.
2.BPModelLoad	Configure extended inputs according to the fixed-wing Blueprint definition.
3.CusLoadBP	Create the Tello skeleton and Animation Blueprint.
4.BombEfffect	Collision events trigger particle and sound effects.

Trace a specific event from input to model action.

3.6.3 Basic and Advanced Example Catalogs: Extended-Input Effects

Extended Inputs and Niagara Responses

2.AdvExps/e2_BlueprintModel

5.LasersEffect

Niagara is UE's visual effects system; this example uses it to display a hit effect.

- **Input:** ActuatorInputsExt carries the target Copter ID.
- **Process:** match the Actor tag; LineTraceByChannel returns the hit position.
- **Output:** a Niagara effect at that position; after a delay, destroy the target and the emitter itself.

Extended events map inputs to specified visual effects.

3.6.3 Basic and Advanced Example Catalogs: OSGB Data Chain

Conversion Outputs for Oblique Photogrammetry Models

2.AdvExps/e3_CusGIS

1.ObliModelMap — Convert OSGB to 3D Tiles and configure the geographic origin.

- **Inputs:** Data/Tile_* and coordinate information in metadata.xml.
- **Process:** convert OSGB to 3D Tiles; configure the source reference and map origin.
- **Outputs:** tileset.json and child tiles; record the model height offset separately.

Deliver tileset.json and its child tiles with the coordinate reference.

3.6.3 Basic and Advanced Example Catalogs: Scanning Objects and Spaces

Scanning a Single Object or an Interior

2.AdvExps/e5_LiDAR-UE

1.3DScannerModel	Scan an object with a phone and repair the mesh in MeshLab.
2.3DScannerScene	Scan an interior with a phone, prepare the mesh, and import it into UE.

Check: verify mesh scale against a known length, inspect holes and orientation, and check geometry and materials after import. Detection tasks also require a collision check.

The capture scope differs; both routes require a usable mesh.

3.6.3 Basic and Advanced Example Catalogs: Further Work and Requirements

Prepare Existing Templates and External Data Separately

- **Inspect existing templates first:** X680 XML and cooked assets; source assets and Blueprints in BlueprintBuild.
- **Prepare input data first:** FBX for loadMeshXML; metadata.xml and models for OSGB; offline terrain / imagery TIF files.

- Further work may require scanning equipment, geographic information system (GIS) services, or a custom edition. Keep the course entry point consistent with the target application.

Template routes reuse existing assets; scanning and GIS routes add acquisition, conversion, and service dependencies.

3.6.3 Basic and Advanced Example Catalogs: Extension Requirements

Inputs and Acceptance Evidence for Three Custom Examples

- **Collision:** select P1/P3 in e3_CollisionMode and record WorldTick with stat game.
- **Geographic data:** tile TIF files in e5_CesiumOffline; serve the tiles through Nginx:8080.
- **Networking:** e1_CtrlAllUEinLAN requires communication between two computers; press I1 at the receiver.

Scope: all three are in 3.CustExps and require the Full Edition or higher. Check Nanite/Lumen/Chaos availability separately for the UE project version.

Provide frame timing, tile loading, and remote command-response evidence for the respective tasks.

Chapter Outline

From the Example Catalog to Three Practice Tasks

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

3.6.1 Entry Points and Task Selection

Textbook: Sec. 3.3.1

3.6.2 Interface Example Catalog

Textbook: Sec. 3.3.1

3.6.3 Basic and Advanced Example Catalogs

Textbook: Sec. 3.3.1

3.6.4 Representative Cases and Practice Roadmap

Textbook: Sec. 3.3.2

3.6.4 Representative Cases and Practice Roadmap: Integrated Applications

Four Integrated Applications and Their Data Chains

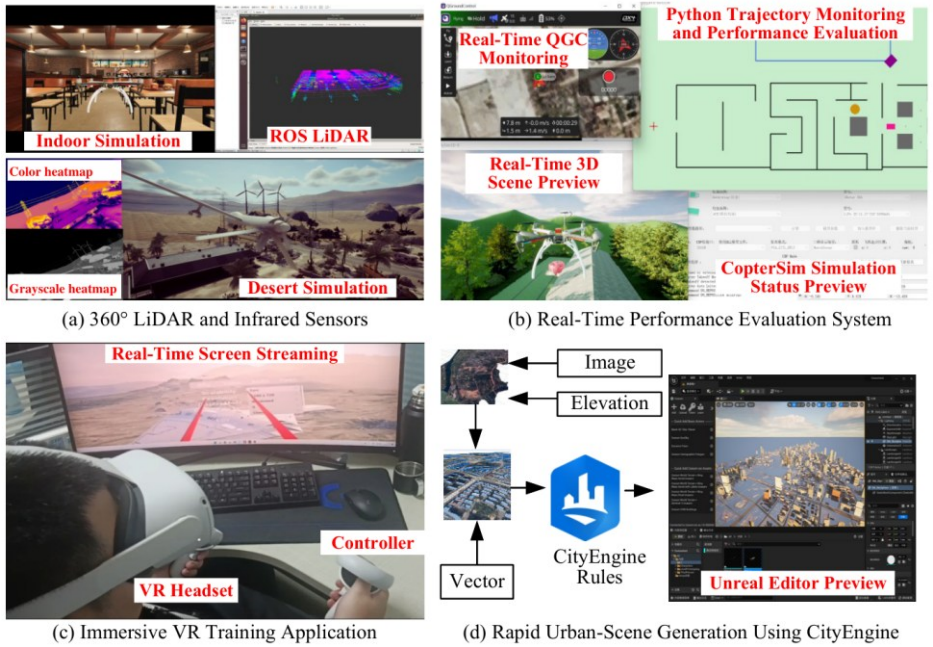


Figure 3.9 Representative Advanced Simulation Scenarios

Integrated tasks must keep environmental changes, state updates, and sensor observations consistent.

3.6.4 Representative Cases and Practice Roadmap: Sensor Application Chains

LiDAR and Infrared Use Separate Observation Chains

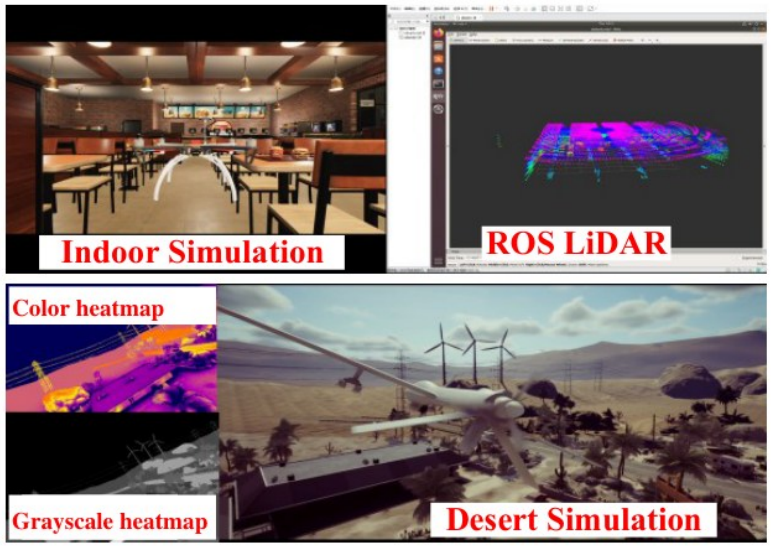


Figure 3.9(a) 360-Degree LiDAR and Infrared Sensors

- **LiDAR (light detection and ranging):** room point clouds and a Robot Operating System (ROS) display.
- **Infrared:** desert environment and two heatmap types.
- **Integration:** record the configuration, sampled data, and display method.

Sensor configuration determines the observation form; acquisition interfaces deliver the data.

3.6.4 Representative Cases and Practice Roadmap: States and Evaluation

Performance Evaluation Uses Trajectories and Task Rules



- **Inputs:** vehicle states and trajectories.
- **Rules:** task regions, paths, or evaluation conditions.
- **Outputs:** monitoring views and computed metrics.

Figure 3.9(b) Real-Time Performance Evaluation System

State data enters the evaluation program; the display supports simultaneous observation.

3.6.4 Representative Cases and Practice Roadmap: VR Interaction Requirements

VR Training Connects Headsets, Controllers, and Views



- **Equipment:** virtual reality (VR) headset and handheld controllers.
- **Display:** immersive view and live screen sharing.
- **Requirements:** matching hardware, drivers, and a supported version.

Figure 3.9(c) Immersive VR Training Application

Interaction-device inputs and visual outputs together form the training interface.

3.6.4 Representative Cases and Practice Roadmap: Procedural City Generation

Inputs, Comparisons, and Delivery for Procedural Cities

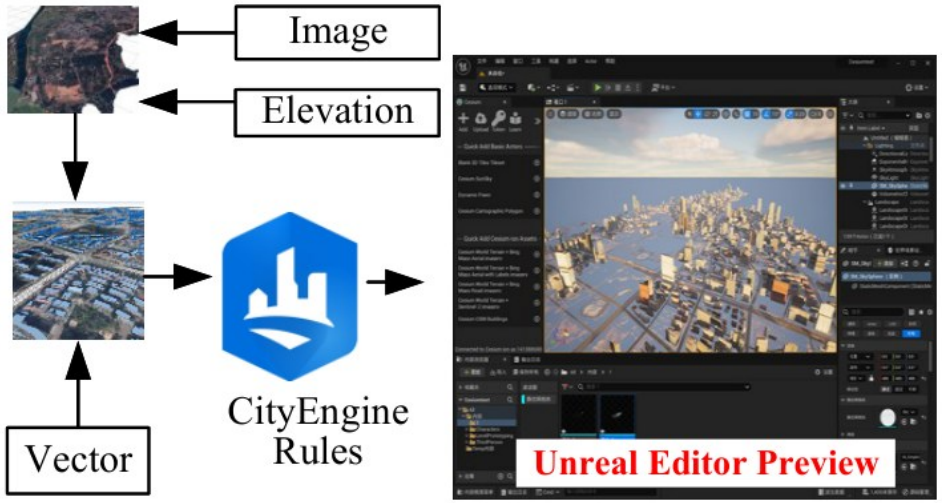


Figure 3.9(d) Rapid City Scene Generation with CityEngine Rules

- **Input roles:** vectors organize roads; elevations define terrain; imagery supplies the background.
- **Rule comparison:** Keep geographic inputs fixed; change building rules and save the results.
- **Scene delivery:** Save models, textures, the geographic origin, and versions together.

Keep geographic inputs fixed and save generation rules to reproduce the same city assets.

3.6.4 Representative Cases and Practice Roadmap: Ground-Placement Entry Point

Two Placement Methods on the Same Map

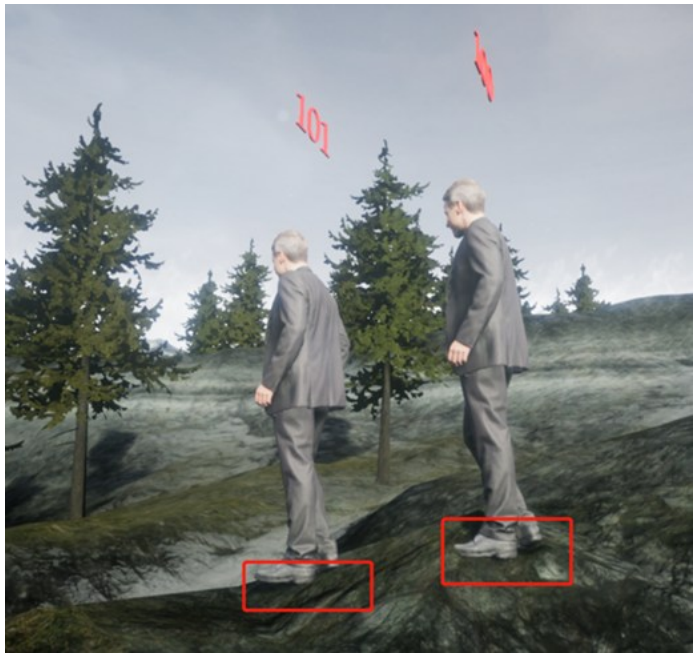


Figure: Character placement in MountainTerrain

- **Start:** run ue4.bat in code and select MountainTerrain.
- **Run:** python TargetPlaceDemo.py
- **Expected:** characters 100/101 appear; compare their feet with the terrain.

Environment: Python38Run.bat

The standard call uses the supplied height; the ground-placement call lets the engine place the model.

3.6.4 Representative Cases and Practice Roadmap: The Two Actual Calls

The Original Template Demonstrates Adjacent Positions

```
ue.sendUE4Pos(
  100,30,0,[-16.479,1.45,-177.181],[0,0,math.pi])

ue.sendUE4Pos2Ground(
  101,30,0,[-17.479,1.45,0],[0,0,math.pi])
```

Objects: IDs 100/101, style 30; x differs by 1 meter and y is the same.
Units: NED position in meters; attitude is roll/pitch/yaw in radians, with yaw π.

The characters differ in x by 1 meter, so both position and method change.

3.6.4 Representative Cases and Practice Roadmap: Ground-Placement Messages

sendUE4Pos2Ground Sends a Ground-Placement Flag

- **Message fields:** ground-placement flag 1234567891, object ID, model style, rotation speed, position, and attitude.
- **Process:** UDP state → RflySim3D recognizes the ground-placement flag → current terrain and model offset determine placement.
- **Data direction:** this call sends state. Python height queries separately read map PNG/TXT files and their calibration.

Interpretation: the standard call places the model origin at the supplied z. The ground-placement call combines the terrain surface with the model's center-to-ground offset.

Python sends the request; the scene engine determines model placement.

3.6.4 Representative Cases and Practice Roadmap: Controlled Comparison at One Point

Fix the Position Before Comparing Placement Methods

- **Fix:** MountainTerrain, model style 30, x/y, and attitude. Create objects in separate runs to avoid overlap.
- **Change:** standard positioning versus ground placement. Record input z, object ID, and visible placement.
- **Interpretation:** terrain location, collision / surface geometry, and the center-to-ground offset jointly affect foot position.

Expected: with correct terrain and offset settings, the ground-placed character's feet meet the surface. If it floats or penetrates the terrain, check surface collision and the model origin separately.

Hold terrain and horizontal position fixed to explain differences in height handling.

3.6.4 Representative Cases and Practice Roadmap: Foot-Clearance Example

Calculate Foot Clearance from NED Heights

- **Given:** in one NED frame, ground Down coordinate 177.0 m and character reference-point coordinate 176.0 m.
- **Model:** the feet are offset +0.8 m in the Down direction from the reference point.

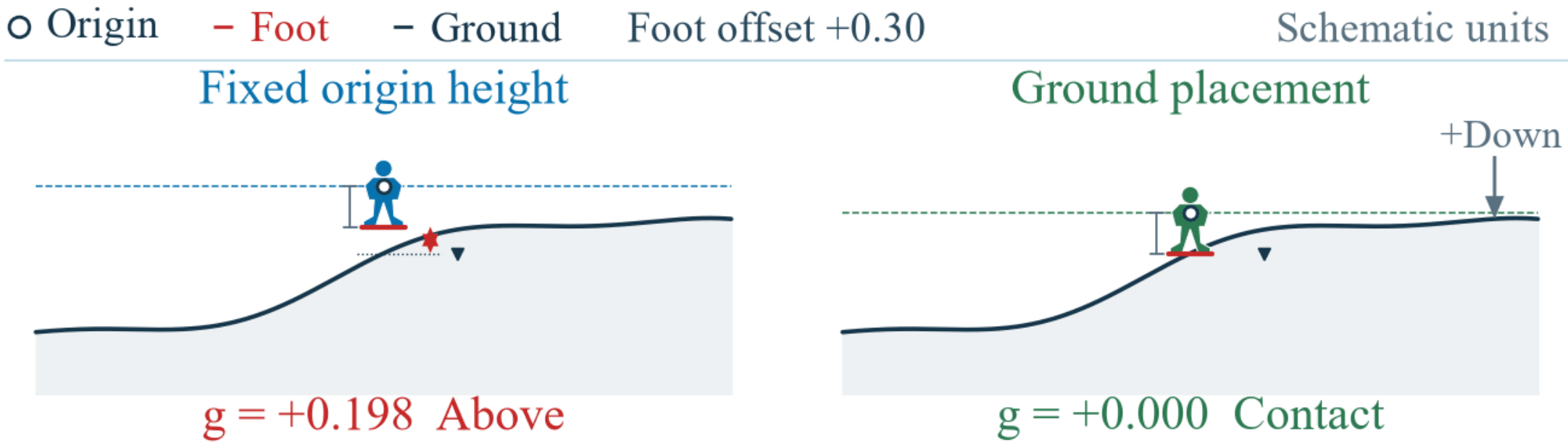
- **Substitution:** $g = 177.0 - (176.0 + 0.8) = 0.2\,\mathrm{m}$

Interpretation: g>0 means the feet are above the ground; g=0 means geometric contact. These are teaching-example values. Check actual ground placement in the 3D application.

The feet are 0.2 m above the ground; ground placement still requires checking the model origin.

3.6.4 Representative Cases and Practice Roadmap: Ground-Placement Comparison

Reference-Height and Ground Placement on One Terrain



$$g = z_{ground} - (z_{ref} + 0.30)$$

Fix the terrain and model origin before interpreting differences in foot placement.

3.6.4 Representative Cases and Practice Roadmap: Scene-Generation Code

TargetCreate.py Creates Nine Elements from Lists

```
for i in range(len(pos)):
    ue.sendUE4PosNew(
        copterID=i+10002, vehicleType=pos[i][0],
        PosE=pos[i][1:4], AngEuler=ang[i],
        runnedTime=-1)
```

IDs: nine entries map to 10002–10010; vehicleType comes from each row.
State: PosE is [x,y,z] in meters; AngEuler is [roll,pitch,yaw] in radians.
Time: retain this example's default runnedTime=-1; the client sends it unchanged.

The loop assigns instance IDs; the corresponding list entries supply types and positions.

3.6.4 Representative Cases and Practice Roadmap: Random Offsets and Output

A Randomized Layout Belongs to That Run

```
pos[i][1] += random.uniform(-0.1,0.1)
pos[i][2] += random.uniform(-0.2,0.2)

print(x[2],y[2])
```

Offsets: for the two square frames of style 815, x changes by ±0.1 meters and y by ±0.2 meters.
Record: print the complete pos list before creation, or save this run's layout JSON afterward.
Printout: 1 1 belongs to the unchanged post of style 824; it does not give the two frame positions.

Rerunning the generation script changes some frame positions.

3.6.4 Representative Cases and Practice Roadmap: Saving a Layout in the Interface

Save Scene Writes This Run's Actor Information to JSON



- **Create:** run TargetCreate.py once in ChallengeMap.
- **Operation:** Save Scene → enter a name → confirm.
- **Outputs:** JSON for this layout and its associated files.

Figure: Naming and saving a custom scene

Save the current layout so that reloading can read the same description.

3.6.4 Representative Cases and Practice Roadmap: Two Save Directories

Layout and Terrain Files Have Different Locations

Layout files:
RflySim3D/RflySim3D/Saved/AllActorSave/<SceneName>.json and .png.
JSON stores the object layout; PNG provides the preview.

Terrain files:
CopterSim/external/map/<SceneName>.txt and .png.
TXT provides calibration; PNG stores elevations.

Steps: stop the creation script → switch to another map → select the saved scene → check each item.

- **Reload:** select the saved scene and check type, position, and attitude by the same ID. Do not rerun random generation.

3.6.4 Representative Cases and Practice Roadmap: Reload Comparison Outputs

Compare Layouts Before Saving and After Reloading

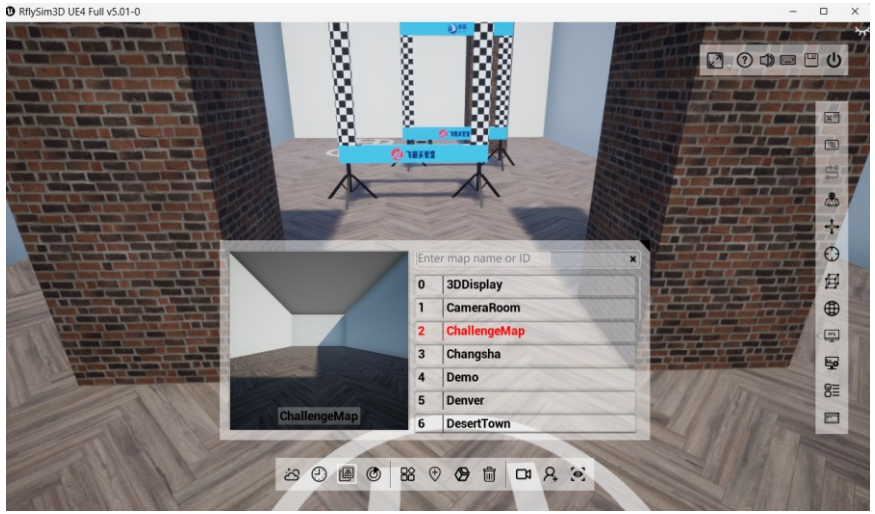


Figure: Generated layout and map selector

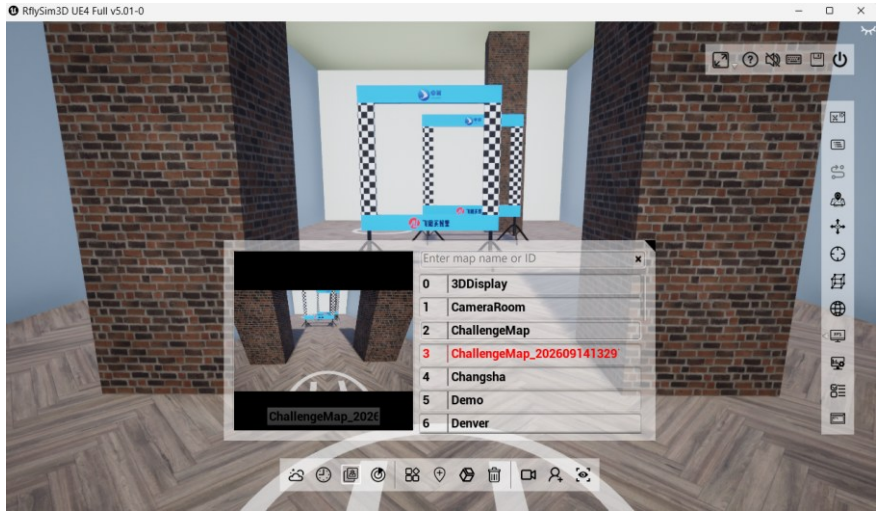


Figure: Layout after reloading the saved scene

Compare: use the brick posts as references for the frames' front–back and left–right relationships.

Reload check: In JSON, compare the same frame's ID, position, and attitude; do not rerun generation.

Saving makes this layout reloadable; frames and posts provide clear spatial references.

3.6.4 Representative Cases and Practice Roadmap: Saving and Rerunning

Rerunning and Reloading Answer Different Questions

Rerun the generation script:
random entries are sampled again, producing a new layout.

Save this layout:
JSON records the current generated result.

Reload the old JSON:
read the saved JSON and restore the positions at save time.

Criterion: compare position and attitude for each instance ID.

3.6.4 Representative Cases and Practice Roadmap: Separate State and Pixel Chains

Start State Queries and Image Acquisition Separately



Figure: Vehicle 1 and character targets (UE4 Full v5.01)

Scene: vehicle 1 is in the foreground; the character target is labeled 100, and the distant character is enlarged.

State chain: query camera and object structures for position, bounding boxes, and other fields.

Pixel chain: the WithCam version separately requests images and reads frames from vis.Img[i].

State return provides fields; image acquisition provides pixel arrays.

3.6.4 Representative Cases and Practice Roadmap: Request Types and Identifiers

One Query Interface Uses Types to Select Targets

```
ue.reqCamCoptObj(0,0)
ue.reqCamCoptObj(1,[1,100])
ue.reqCamCoptObj(2,'Landscape_1')
ue.initUE4MsgRec()
```

Start: GetCamObjDemo.bat; wait for Global Positioning System (GPS) and extended Kalman filter (EKF) initialization.
Types: 0 uses camera SeqID; 1 uses instance ID; 2 uses the scene object name.
Conditions: the first camera request is for WithCam; instances 1/100 and Landscape_1 must already exist.
Reception: initUE4MsgRec starts listening; getCamCoptObj reads the cache.

The type and identifier together determine which structure is requested.

3.6.4 Representative Cases and Practice Roadmap: Reading State Fields

hasUpdate Indicates a New State in the Cache

```
Copt = ue.getCamCoptObj(1,1)
if isinstance(Copt,UE4CtrlAPI.CoptReqData):
    if Copt.hasUpdate:
        print(Copt.CopterID,Copt.PosUE,Copt.BoxExtent)
        Copt.hasUpdate = False
```

New state: when hasUpdate is true, process it once, then set it to False and wait for the next message.

Run: enter python GetCamObjDemo.py in the terminal opened by Python38Run.bat.

Position: PosUE is the model origin; boxOrigin is the geometric bounding-box center.

Size: BoxExtent gives half-extents; multiply by 2 for full length, width, and height.

3.6.4 Representative Cases and Practice Roadmap: State Updates

hasUpdate Marks New State Since the Last Clear

- **Request:** subscribe to object state by type and target identifier.
- **Arrival:** the SDK caches a new structure and sets the update flag.
- Read: when hasUpdate is true, process once and clear the flag.

Criterion: stationary objects may still send periodic updates; count observed cache updates.

Poll count is not arrival count; repeated reads are not new data.

3.6.4 Representative Cases and Practice Roadmap: State Updates

State Messages Can Keep Updating after the Object Stops

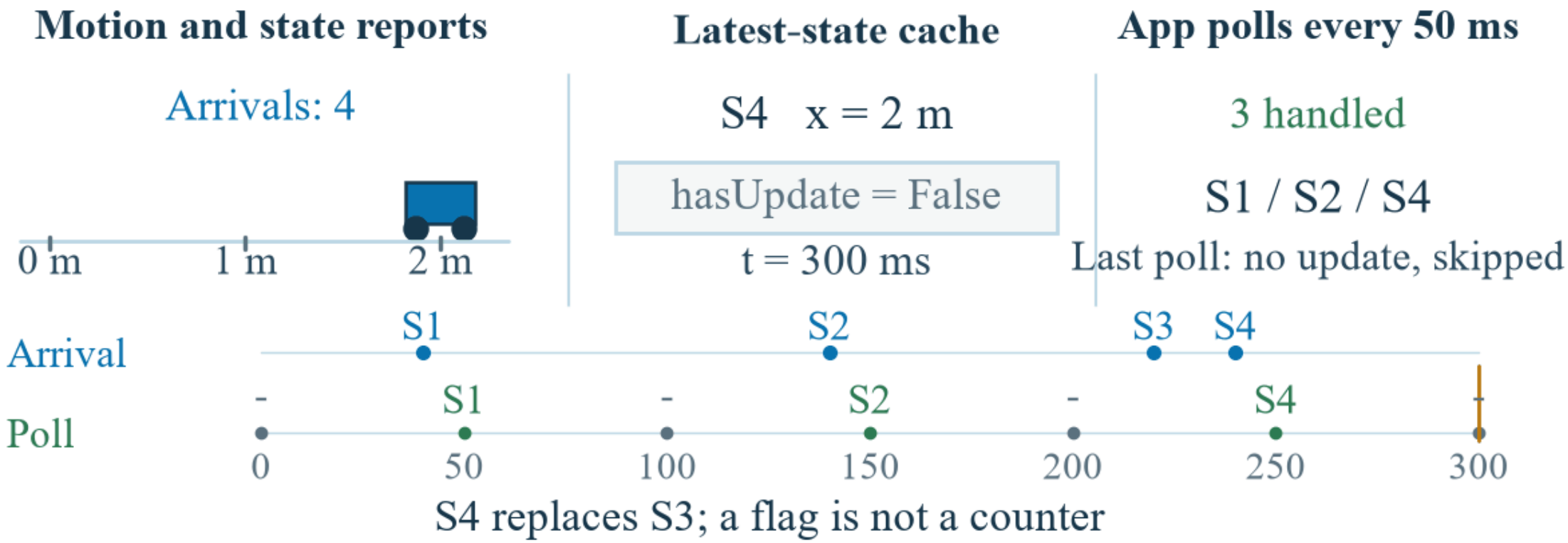


Figure: horizontal axis in ms; S labels messages in this illustration. The Boolean flag is not a message counter.

3.6.4 Representative Cases and Practice Roadmap: Image Configuration

Config.json Defines Two 640×480 RGB Streams

"SeqID": 0, "TypeID": 1, "TargetCopter": 1
"DataWidth": 640, "DataHeight": 480
"DataCheckFreq": 30
"SensorPosXYZ": [0.3,-0.15,0]
"SeqID": 1, "SensorPosXYZ": [0.3,0.15,0]

Type: two RGB streams, 640×480, with 0.3 meters of lateral camera separation.
Frequency: DataCheckFreq=30 requests 30 Hz updates; measure the actual receive rate separately.
Transport: the first SendProtocol item is 0, selecting local shared memory.

Camera sequence ID, attachment target, and image dimensions together define the acquisition request.

3.6.4 Representative Cases and Practice Roadmap: Dual-Camera Timing

Check Config, Request Sensors, Receive Two Pixel Streams

- **Configure:** give each camera its sequence ID, attachment target, and resolution.
- **Start:** run the supplied acquisition script after SITL initialization.
- **Receive:** check new frames, array shape, and timestamps for each stream.

Finish: stop the script, close the windows, and keep the configuration and records.

The two windows aid observation; array shape is the dimension check.

3.6.4 Representative Cases and Practice Roadmap: Starting Pixel Reception

View Camera Pixels and State Output Together

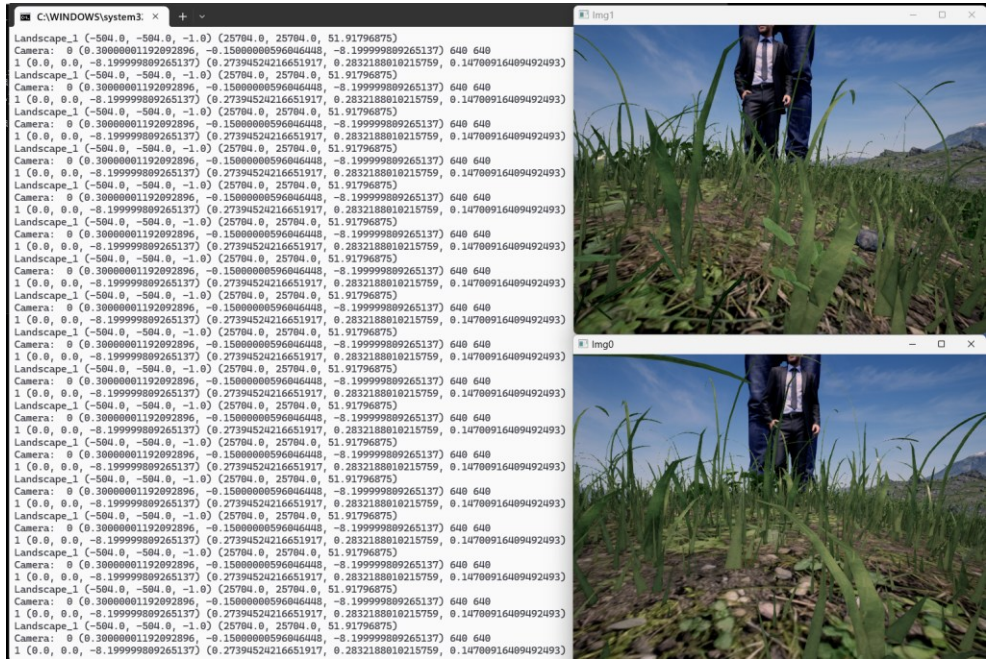


Figure: Terminal states and two camera images

Run: python GetCamObjDemoWithCam.py
Receive: read vis.Img[i] when hasData[i] is true.

Read the figure: Camera 0 and object 1 provide positions; the two images show characters and grass.
Dimensions: the next page explains why 640 appears twice.

States and pixels use separate paths; check object identity and new frames separately.

3.6.4 Representative Cases and Practice Roadmap: Dimensions and a Print Error

Printing Width Twice Does Not Give Image Height

Cam.DataWidth, Cam.DataWidth # Original script's final line

Original printout: both values are width, not width and height.

Cam.DataWidth, Cam.DataHeight # Classroom correction

Fields: DataWidth is 640; DataHeight is 480.

vis.Img[i].shape # Pixel array shape

Array order: height, width, channels; expected shape here is (480,640,3).

3.6.4 Representative Cases and Practice Roadmap: Practice Records and Interpretation

Results and Troubleshooting for the Three Tasks

- **Ground placement:** the standard call uses the supplied z; the engine combines the terrain and center-to-ground offset for ground placement.
- **Saving:** JSON retains this run's layout. Check position and attitude by the same ID; rerunning random generation changes the inputs.
- **Acquisition:** receive states and pixels separately. If images are missing, check configuration, requests, and reception first. Check dimensions in array height–width order.

Match each output to its input to explain differences.

3.6.4 Representative Cases and Practice Roadmap: Configuration and Acquisition Resources

Follow Configuration into State and Pixel Source Code

0.ApiExps/e6_RflySim3DCtrlAPI/10.GetCamObjDemo

- **Configuration:** Config.json; change one camera position and keep the other as a reference.
- **States:** GetCamObjDemo.py; trace request types, target identifiers, and hasUpdate.
- **Pixels:** GetCamObjDemoWithCam.py; trace requests, reception, and vis.Img.
- **Documentation:** visual configuration and acquisition in 3.4.4; UE4CtrlAPI definitions in 3.4.

Compare configuration, state scripts, and image scripts from the same example.

Classroom Exercises

Change One Input, Then Predict the Observable Difference

- Ground placement: fix the map and x,y; compare supplied z with ground placement.
- Saving: save one layout, rerun the script, then reload the old JSON.
- **Acquisition:** change only camera 1's attachment offset; keep camera 0 as the reference.

Record inputs, predictions, results, and reasons for any failed checks in each task.

Use the same conditions for prediction, execution, and recording.

Exercise Reference Points

Check Height Differences, Layout Files, and New Frames

- **Ground placement:** NED z is positive downward; compare terrain and input reference-point coordinates at the same location.
- Saving: random entries may change on rerun; reloading old JSON should restore the old layout.
- **Acquisition:** keep camera 0 as the reference; assess camera 1 using new-frame timestamps and array shape.

$$\Delta z = z_{\text{ground}} - z_{\text{input}}$$

Rule out changed inputs and stale data to identify which step caused a difference.

Section Summary

Close the Loop from Example Selection to Result Interpretation

- **Ground placement:** fix the map and horizontal position; compare foot placement from standard and ground-placement calls.
- **Saving:** record this run's randomized layout and JSON; compare reloading the saved scene separately from rerunning generation.
- **Acquisition:** identify new data by state-update flags and image timestamps; check array shapes and processing time.

Keep each task's entry point, inputs, outputs, and criteria.