Intelligent Unmanned System Control Practice

Chapter 3 3D Environment Modeling and Simulation

Author: Xunhua Dai, Central South University

Website: https://faculty.csu.edu.cn/daixunhua

Email: dai.xh@csu.edu.cn

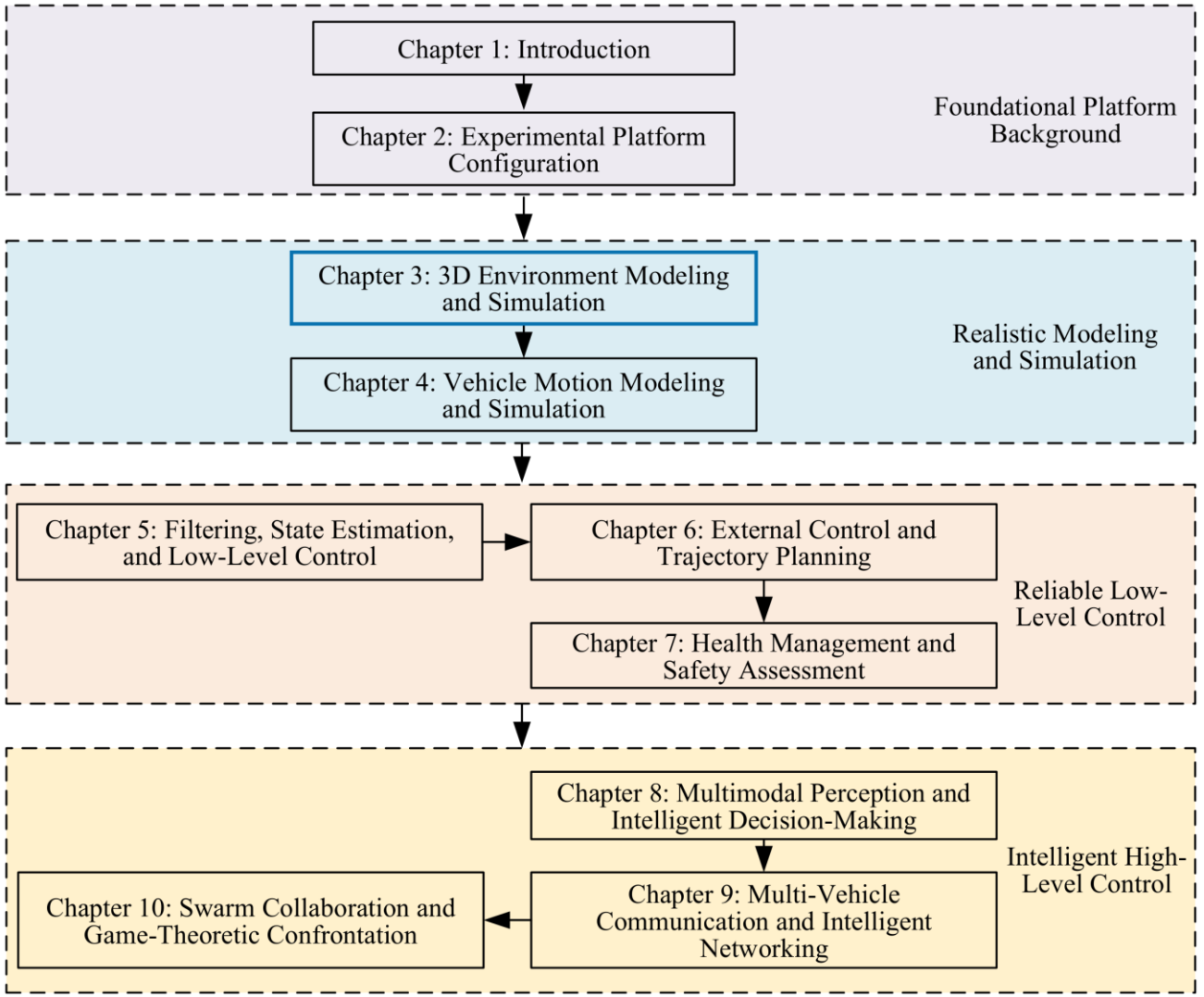
RflySim: http://rflysim.com

Companion Paper



Dai, Xunhua, Jinhu Tu, and Quan Quan. "RflySim ToolChain: a rapid development and validation toolchain for intelligent unmanned swarm systems." Journal of Systems Engineering and Electronics 36.4 (2025): 1077-1093.

Textbook is coming soon...



Chapter Learning Objectives

What You Can Do After Chapter 3

1. Explain how geometry, materials, scenes, and observations relate
2. Choose modeling and scene tools for the available resources
3. Configure objects and interfaces with defined coordinates and timing
4. Reproduce an example, save the scene, and explain a change

The value of 3D simulation lies in changing the environment repeatably and observing how algorithms respond.

Chapter Outline

3D Simulation Fundamentals and Tool Selection

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

5 Units in This Section

3.1.1 Background and 3D Simulation Workflow
Textbook: Secs. 3.1.1–3.1.2

3.1.2 Model Creation and Editing
Textbook: Sec. 3.1.2 · Model Creation and Editing

3.1.3 Modeling and Scene Construction Tools
Textbook: Secs. 3.1.3.1 and 3.1.3.3

3.1.4 Auxiliary Tools
Textbook: Sec. 3.1.3.3; data collection: Sec. 3.1.2

3.1.5 Platform Comparison and Selection
Textbook: Secs. 3.1.3.1–3.1.3.2

3.1.1 Background and 3D Simulation Workflow: Assets and Observations

Assets, Scenes, States, and Observations

- **3D assets:** Meshes, materials, textures, or robot structures that can be reused in different scenes.
- **Scene:** Terrain, buildings, vehicles, lights, and sensors organized in a shared space.
- **Runtime state:** Object positions, orientations, and component motion, updated continuously by motion models or scripts.
- **Sensor observations:** Images, depth, point clouds, and other algorithm inputs generated for a specified state and configuration.
- **Example:** Place the same house model twice beside a road. As the vehicle moves, its state changes and the camera sees different occlusions.

Assets describe objects; scenes organize them; states and sensors together determine observations.

3.1.1 Background and 3D Simulation Workflow: Environment and Observation Conditions

How Environment Differences Affect Unmanned-System Tasks

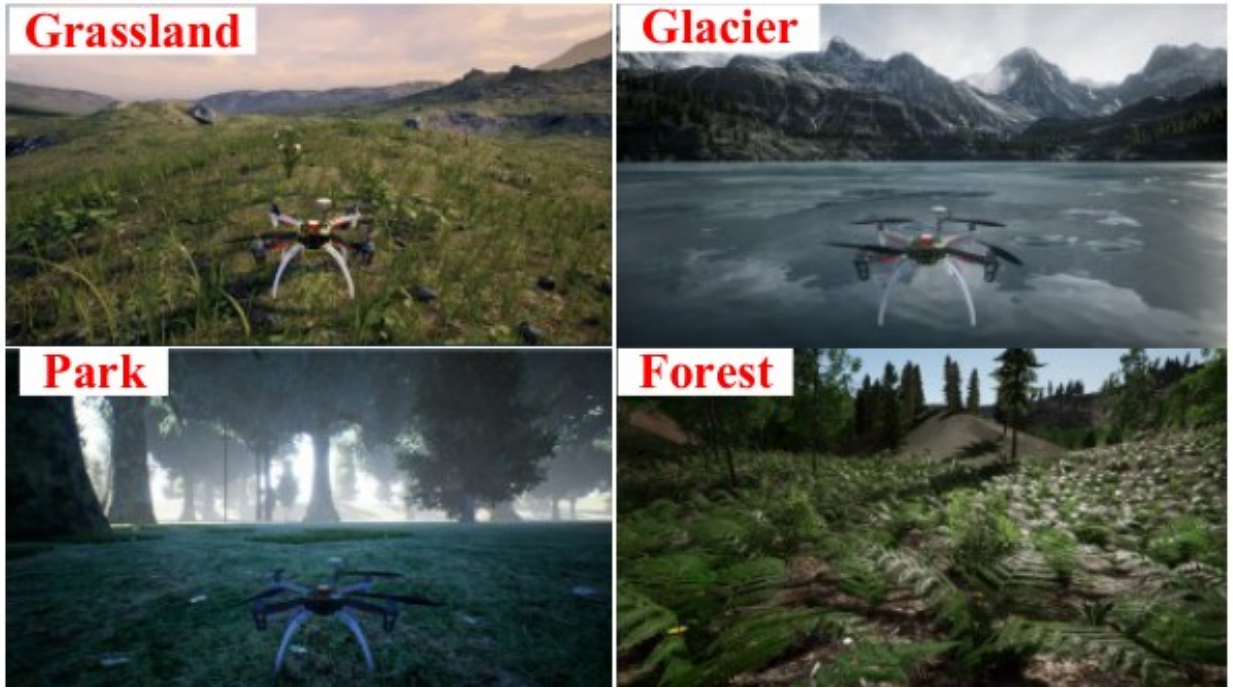


Figure: 3D scenes in different environments

- **Spatial conditions:** Terrain, building spacing, and obstacle locations determine feasible paths.
- **Observation conditions:** Texture, reflections, and shadows affect recognition and localization.
- **Task conditions:** One scene can support aerial, ground, and other vehicle types.

Terrain and occlusion affect traversability; surfaces and lighting affect visual cues.

3.1.1 Background and 3D Simulation Workflow: Simulation and the AI Control Loop

3D Environments in the Perception–Control Loop

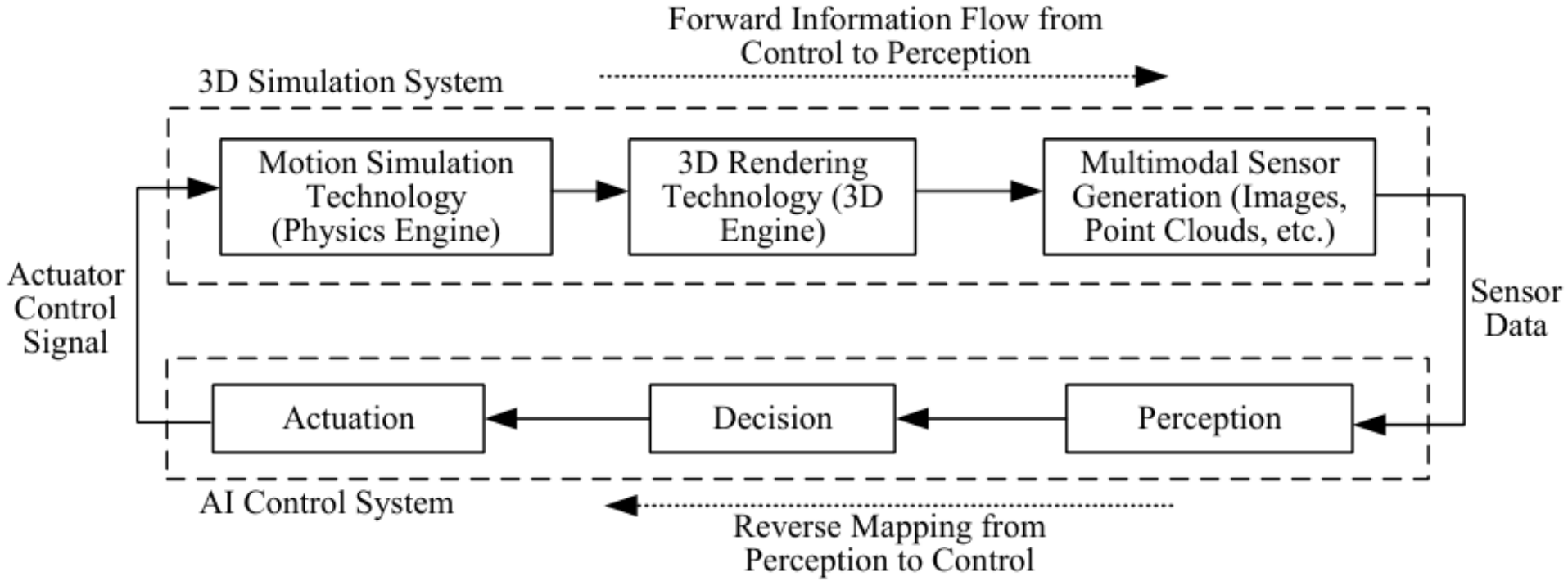


Figure 3.1 3D Simulation and Artificial Intelligence (AI) Control Systems

States enter the scene; the scene produces observations for perception, decision-making, and control.

3.1.1 Background and 3D Simulation Workflow: Task Applications

One Scene for Perception, Planning, and Training

- **Perception and fusion:** Fix camera and light detection and ranging (LiDAR) configurations; compare observations as occlusion changes.
- **Planning and coordination:** Replay the same obstacle layout; compare routes and multi-vehicle collision avoidance.
- **Training and evaluation:** Change only the weather or lighting; observe whether algorithm performance remains stable.
- **Human–machine interaction:** Display states and events so operators can inspect, intervene, and review decisions.

3D simulation provides repeatable environment, observation, and interaction conditions for a shared task.

3.1.1 Background and 3D Simulation Workflow: Rendering, Motion, and Sensors

Roles of Rendering, Physics, and Sensors

Stage	Inputs and Outputs	Role in the Task
Rendering	Scene, lighting, and camera → Image	Produces visible appearance
Physics and motion	Forces, control, and constraints → Motion state	Computes motion and contact responses
Sensors	Scene and sampling configuration → Observation data	Produces image, range, and other observations
Interaction and interfaces	Keys, scripts, and messages → Runtime operations	Connects algorithms and experimental conditions

Images, motion responses, and sensor data come from different computational stages.

3.1.1 Background and 3D Simulation Workflow: The Simulation Pipeline

From Assets and Scenes to Observations

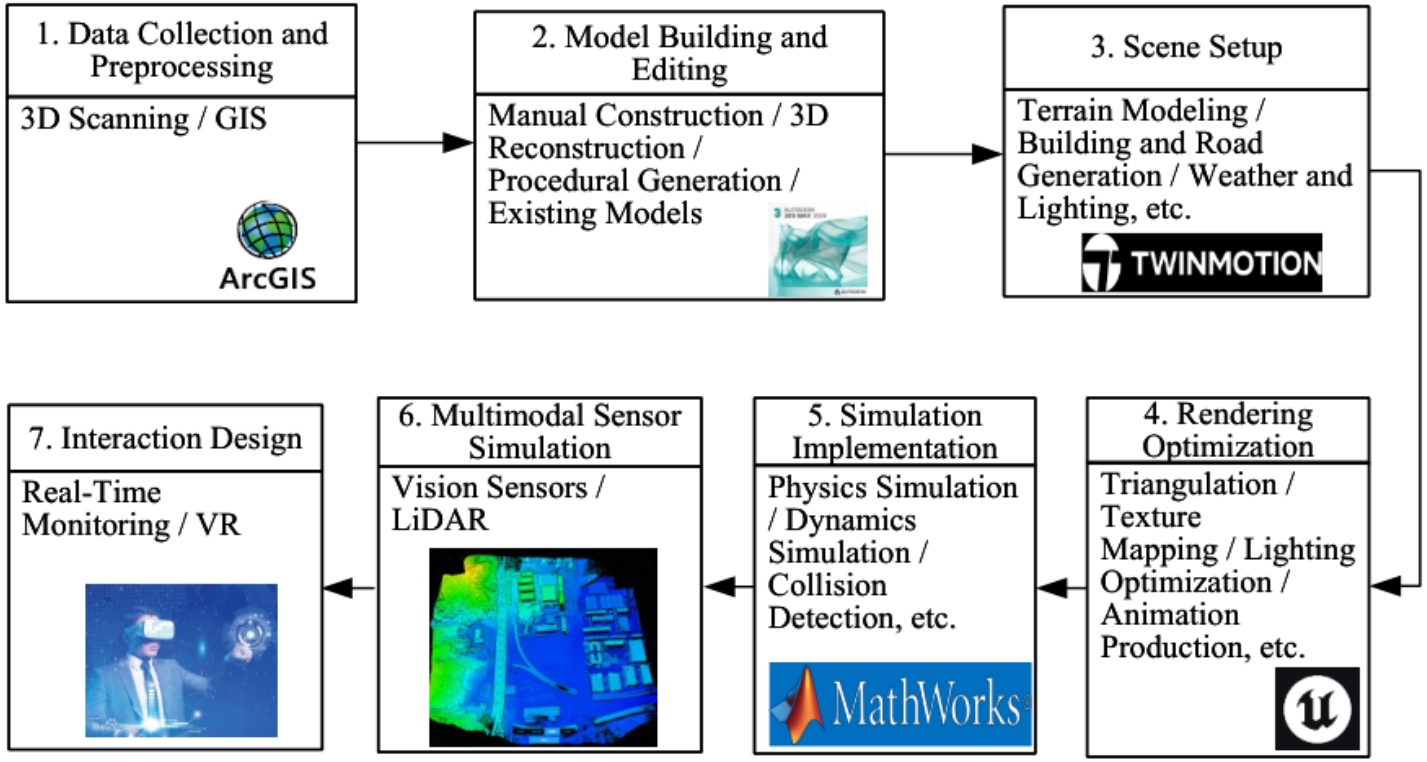


Figure 3.2 3D Simulation Workflow

Enter at the stage that matches the available inputs; static backgrounds can skip motion design.

3.1.1 Background and 3D Simulation Workflow: The First Three Stages

From Real-World Data to a Runnable Scene

- **Data collection:** Clean scans, imagery, or geographic data and check coordinates.
- **Model creation:** Organize shape, appearance, and motion into reusable object assets.
- **Scene construction:** Place terrain, buildings, vehicles, and lights with consistent scale and coordinates.
- **Handoff checks:** Confirm origins, units, and materials before execution and observation.

Each stage delivers assets we can check; existing assets can eliminate the need for new data collection.

3.1.1 Background and 3D Simulation Workflow: Data Sources and Asset Reuse

Three Asset Starting Points, Different Workloads

Starting Point	Additional Work	Examples in This Chapter
Existing model or scene	Import, placement, and configuration	loadSimpleMesh / StarterContent
Designed geometry	Modeling, component organization, materials	3dsMaxUsage / SketchUpUsage
Real-environment data	Capture: scan and clean Reuse: load and georeference	Own capture: 3DScannerScene Reuse: Cesium

Reuse assets first; create geometry or capture real-world data when needed.

3.1.1 Background and 3D Simulation Workflow: Runtime Speed and Rendering Quality

Balancing Visual Fidelity and Runtime Speed

- **Visual recognition:** Retain target textures, edges, and important lighting changes so targets remain distinguishable in images.
- **Traversability and obstacle avoidance:** Preserve obstacle outlines, distances, and passage widths; simplify tiny distant decorations.
- **Computational cost:** Visible geometry, lighting, and multiple camera views increase rendering load and affect data arrival rates.
- **Comparison:** Fix the layout and camera trajectory, then change one factor at a time to explain differences in results.

Preserve details that affect task decisions; spend computation on useful observations.

3.1.1 Background and 3D Simulation Workflow: Technical Challenges

Why Realistic Scenes Need Timing and Compute Budgets

- **Real-time operation:** At 30 camera frames/s, the total budget is about 33 milliseconds per frame.
- **Multiple sensors:** Images and point clouds have different sampling rates; check timestamps and coordinates.
- **Large scenes:** More detail, greater extent, and more moving objects increase loading and update demands.
- **Compute contention:** Rendering, physics, and algorithms share the central processing unit (CPU) and graphics processing unit (GPU); allocate capacity for the required observations.

Balance visual fidelity, timing consistency, and runtime speed for the task.

Chapter Outline

3D Simulation Fundamentals and Tool Selection

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

5 Units in This Section

3.1.1 Background and 3D Simulation Workflow
Textbook: Secs. 3.1.1–3.1.2

3.1.2 Model Creation and Editing
Textbook: Sec. 3.1.2 · Model Creation and Editing

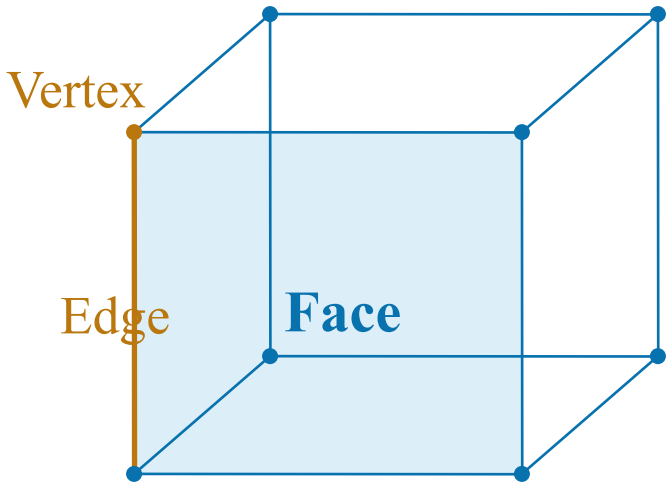
3.1.3 Modeling and Scene Construction Tools
Textbook: Secs. 3.1.3.1 and 3.1.3.3

3.1.4 Auxiliary Tools
Textbook: Sec. 3.1.3.3; data collection: Sec. 3.1.2

3.1.5 Platform Comparison and Selection
Textbook: Secs. 3.1.3.1–3.1.3.2

3.1.2 Model Creation and Editing: Meshes and Topology

Meshes: Vertices, Edges, Faces, and Connectivity



Schematic: The light-blue region is one face

- **Vertices and edges:** Vertices store 3D positions; edges connect them
to outline surface boundaries.

- **Faces and topology:** Faces represent surface regions;
their connections determine openings and continuity.

- **Editing effects:** Moving a vertex changes the shape;
deleting a face opens the surface.

Vertices specify positions; topology specifies connectivity; together they define the object surface.

3.1.2 Model Creation and Editing: Geometry and Appearance

Geometric Detail and Surface Appearance

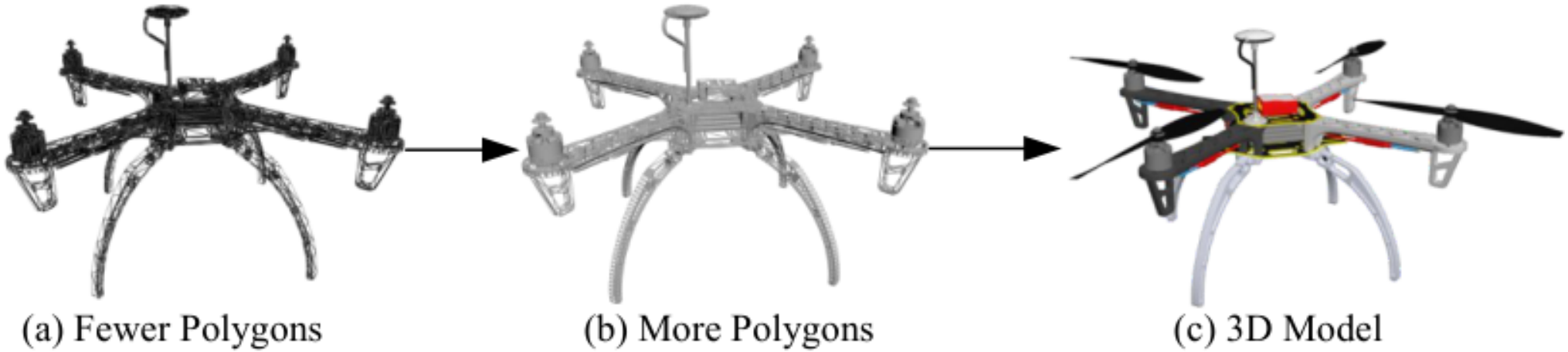
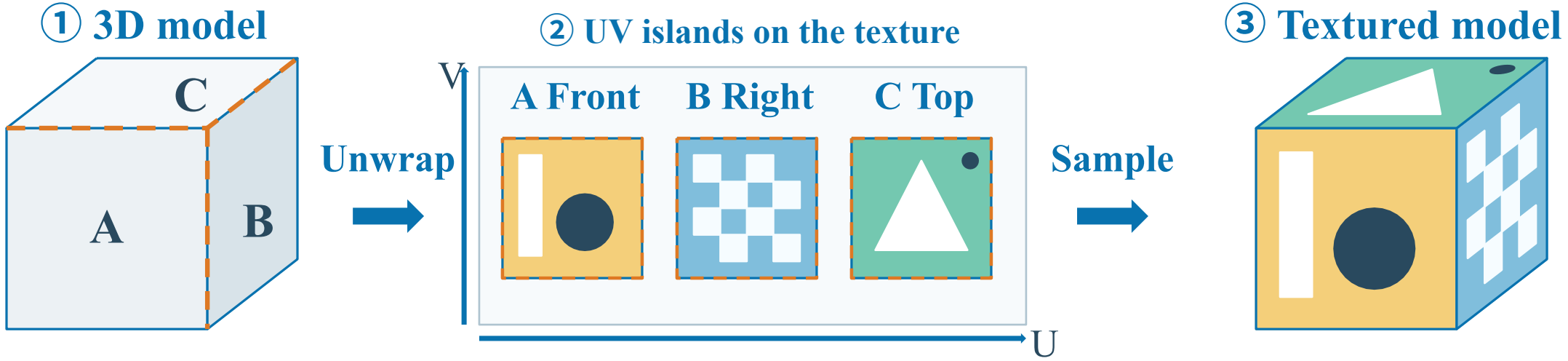


Figure 3.3 Geometric Shape Definitions and Topological Relationships (schematic)

- **Outline:** Too few polygons make curved surfaces and edges appear angular.
- **Appearance:** Textures and materials also determine color and lighting response; polygon count alone does not.

3.1.2 Model Creation and Editing: UVs and Materials

UV Unwrapping: Mapping 3D Surfaces to 2D Textures



Front / right / top faces

Orange dashes: unwrap seams

Texture mapped to each face

- **UV unwrapping:** Cut seams into 2D UV islands and assign texture coordinates (U, V).
- **Texture sampling:** U (horizontal) and V (vertical) locate each image color on the surface.
- **Material:** Textures and roughness define the surface response to light.

Geometry locates the surface, UVs locate texture samples, and materials define the lighting response.

3.1.2 Model Creation and Editing: UVs and Materials

Unwrap Along Seams, Then Map Textures Back with UVs

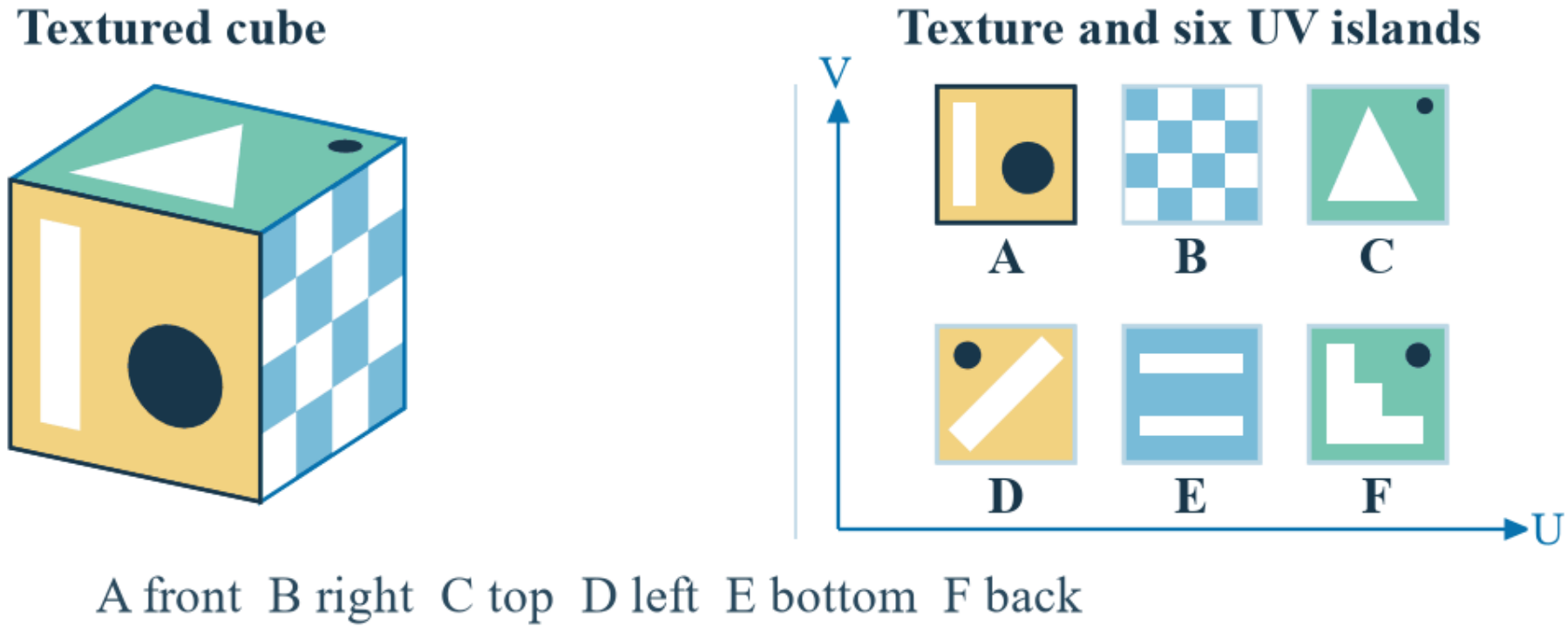


Figure: Unfolding six faces is a cutting-pattern analogy; UV editing maps coordinates without changing the model shape

3.1.2 Model Creation and Editing: Assets and Instances

Model Files, Engine Assets, and Scene Instances

box.fbx Interchange file	➡	Mesh asset Reusable resource	➡	Two scene instances Placed separately

- **Unreal Engine (UE) assets:** Imported meshes and materials can be shared by multiple objects, avoiding duplicate production.
- **Independent transforms:** Each instance stores position, rotation, and scale; repositioning one does not change the source file.

Multiple instances can reference one asset; each instance has its own position and state.

3.1.2 Model Creation and Editing: Dimensions and Pivots

Dimensions, Origins, and Pivots Define Imported Geometry

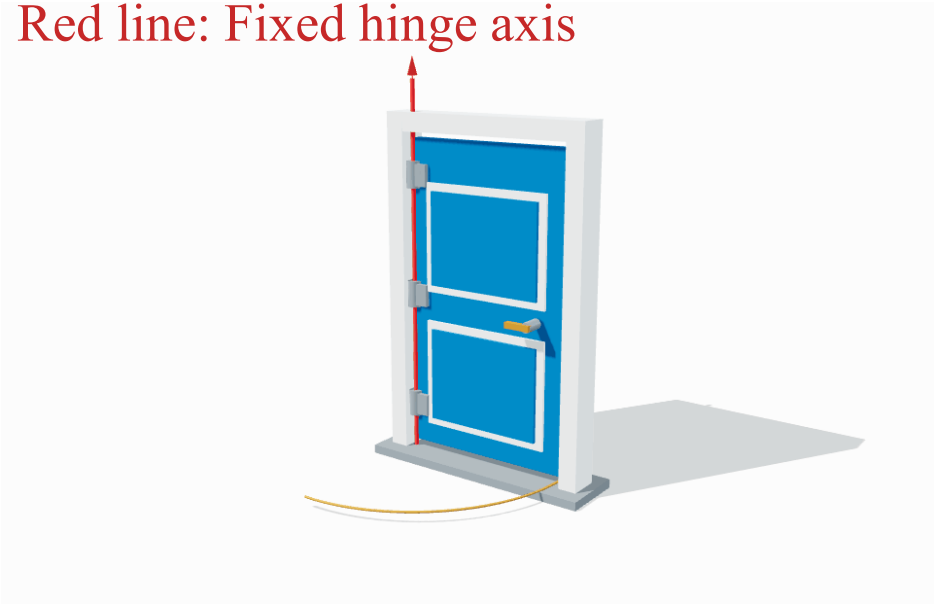


Figure: A 2 m door panel opens about a side hinge (schematic)

- **Dimensions:** 2 m and 2 cm differ by a factor of 100; import units affect size relative to the vehicle.
- **Origin:** A base origin helps place an object on the ground; a center origin helps with symmetric rotation.
- **Component pivots:** A door turns at its hinge; a rotor turns about its motor shaft.

Check scale, origins, and pivots before import so objects are placed and move correctly.

3.1.2 Model Creation and Editing: File Purposes

File Formats Serve Different Data Purposes

Data Type	Common Formats	Information Preserved
Mesh interchange	FBX / OBJ / STL	Geometry; appearance and motion depend on the format
Runtime assets	glTF / GLB	Hierarchy, meshes, surface data
Robot description	Unified Robot Description Format (URDF), etc.	Links, joints, inertia
Authoring projects / scenes	SketchUp (SKP) Twinmotion (TM)	Editable tool-specific organization

Choose a format for the handoff: shape, robot structure, and scene organization require different data.

3.1.2 Model Creation and Editing: Mesh Interchange Formats

What FBX, OBJ, and STL Transfer

Format	Main Data Transferred	Selection and Conditions
FBX	Meshes, hierarchy, animation; convertible material information	For components and motion; complex materials may need rebuilding
OBJ	Static geometry and UVs; appearance often uses MTL and textures	For static textured meshes; keep companion files together
STL	Surface geometry as triangles	For shape-only handoff; not complete appearance or animation

3.1.2 Model Creation and Editing: URDF Structure

Describe a Robot with Links and Joints

Continuous joint: No angular limits

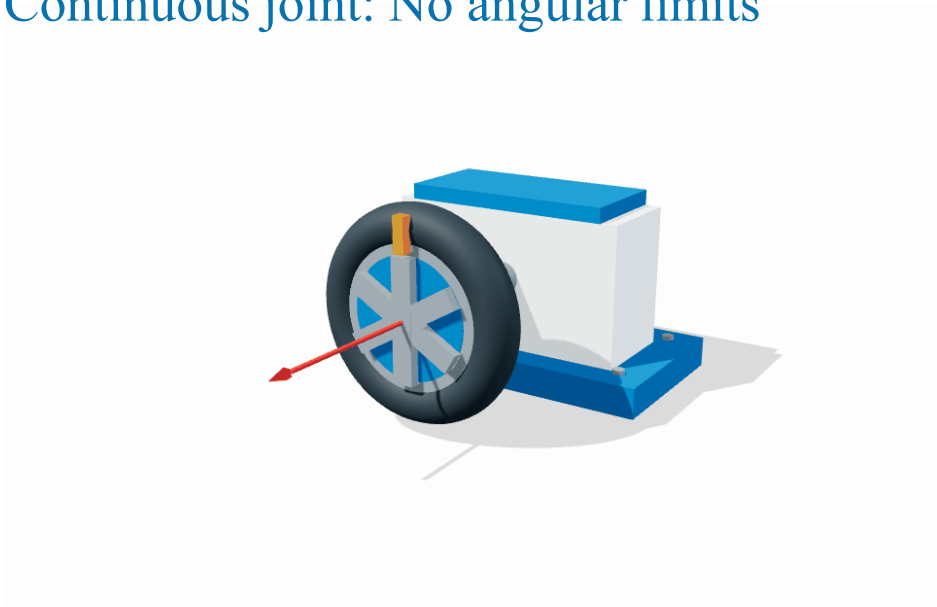


Figure: Base–wheel joint

- **Structure file:** Extensible Markup Language (XML) tags
record components and connections.

- **link:** A rigid component that may reference a mesh for its appearance.

- **joint:** Connects parent and child links, defining their relative
pose, motion type, and axis.

Introductory URDF models and source: https://github.com/ros/urdf_tutorial

URDF connects components in a tree; mesh files describe each component's shape.

3.1.2 Model Creation and Editing: URDF Properties

Appearance, Collision, and Inertia in URDF

wheel_link: The same wheel link

- **visual: Detailed mesh and textures**
- **collision: Simplified cylinder**
- **inertial: Mass, center of mass, inertia**

Three data groups for one link

- **visual:** Visible geometry, materials, and relative pose
  determine how the component appears.
- **collision:** Geometry for contact detection, often using
  simpler shapes.
- **inertial:** Mass, center of mass, and rotational inertia
  enter the dynamics calculations.

One link can combine detailed visuals, simplified collision geometry, and separate inertial parameters.

3.1.2 Model Creation and Editing: Limits of URDF Reuse

What URDF Can Reuse Across Platforms

Platform	How URDF Is Used	Further Configuration
RflySim	Import URDF robot assets	Platform control, observations, and experiments
Isaac Sim	Convert to Universal Scene Description (USD) assets	Drive, collision, and sensor properties
MuJoCo	Read URDF; use MuJoCo's native MJCF model format.	Actuators, contact, and simulation options

MuJoCo models and URDF extensions: https://mujoco.readthedocs.io/en/stable/modeling.html

Reuse robot structures and meshes; configure drives, contact, and sensors for the target platform.

Chapter Outline

3D Simulation Fundamentals and Tool Selection

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

5 Units in This Section

3.1.1 Background and 3D Simulation Workflow
Textbook: Secs. 3.1.1–3.1.2

3.1.2 Model Creation and Editing
Textbook: Sec. 3.1.2 · Model Creation and Editing

3.1.3 Modeling and Scene Construction Tools
Textbook: Secs. 3.1.3.1 and 3.1.3.3

3.1.4 Auxiliary Tools
Textbook: Sec. 3.1.3.3; data collection: Sec. 3.1.2

3.1.5 Platform Comparison and Selection
Textbook: Secs. 3.1.3.1–3.1.3.2

3.1.3 Modeling and Scene Construction Tools: Tool Roles

From Geometry to Execution: Choose Tools by Task

Geometry creation
Choose tools as needed
➡
Scene environment
Twinmotion optional
➡
UE Editor
Build a runnable scene
➡
RflySim3D
Execution and observations

- **Geometry:** Use 3ds Max, Blender, or SketchUp to create meshes, materials, and textures.
- **Asset reuse:** Import existing models or Fab assets directly into UE, skipping modeling from scratch.
- **Environment:** Arrange the scene in UE; Twinmotion can add materials, vegetation, and lighting.
- **Execution:** UE organizes objects and behavior; RflySim3D connects motion states and observations.

Geometry, environment construction, and execution produce different outputs; reuse existing assets where possible.

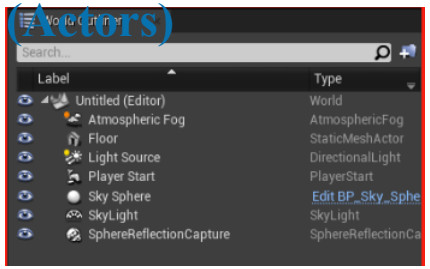
3.1.3 Modeling and Scene Construction Tools: UE Editing and Execution

The UE Editor and RflySim3D Runtime

Viewport
Preview the 3D scene



World Outliner
Scene objects
(Actors)



Content Browser: model and material assets

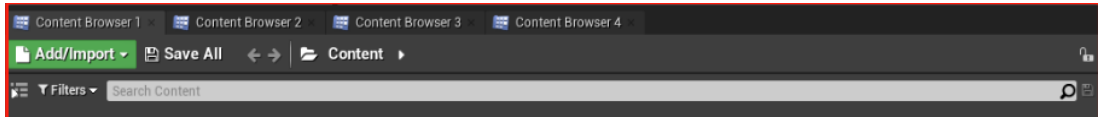


Figure: Three UE Editor regions (detail)

- **Unreal Engine (UE):** Organizes scenes, rendering, interaction, and extensible behavior.
- **Editing:** The box example uses UE4.27.2; create a blank project, then develop the level and Blueprints.
- **Execution:** RflySim3D loads assets and connects object states and sensors.

The editor creates scenes and behavior; the runtime loads assets and responds to experimental inputs.

3.1.3 Modeling and Scene Construction Tools: Blueprint and C++

How Blueprint and C++ Share the Work

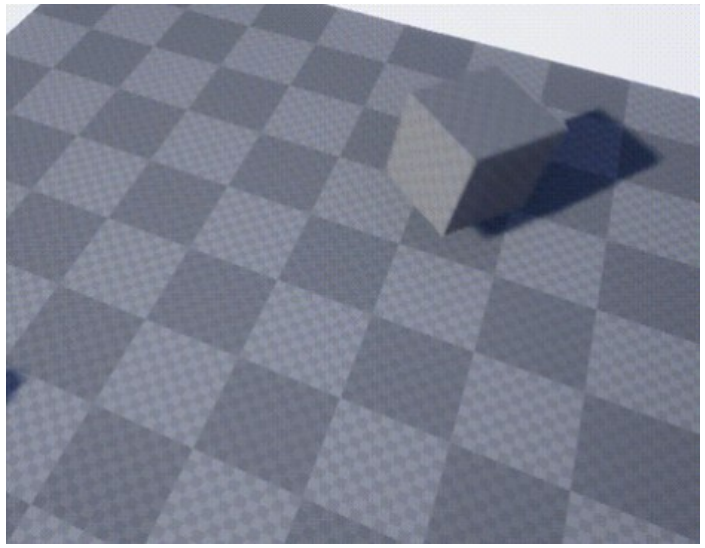


Figure: The running scene in the Blueprint box example

- **Blueprint:** Visual scripts link events, variables, and operations so behavior is easy to inspect.
- **C++:** Extend components, plugins, and interfaces; combine native code with Blueprint.
- **Runtime check:** Compile and save the Blueprint, place it in the level, click Play, and check box rotation.

UE levels, assets, and Blueprint documentation: https://dev.epicgames.com/documentation/en-us/unreal-engine

Blueprint connects scene behavior; C++ extends lower-level capabilities and native interfaces.

3.1.3 Modeling and Scene Construction Tools: Modeling in 3ds Max

Editing Detailed Meshes and Components

Create panel: Standard Primitives → Box

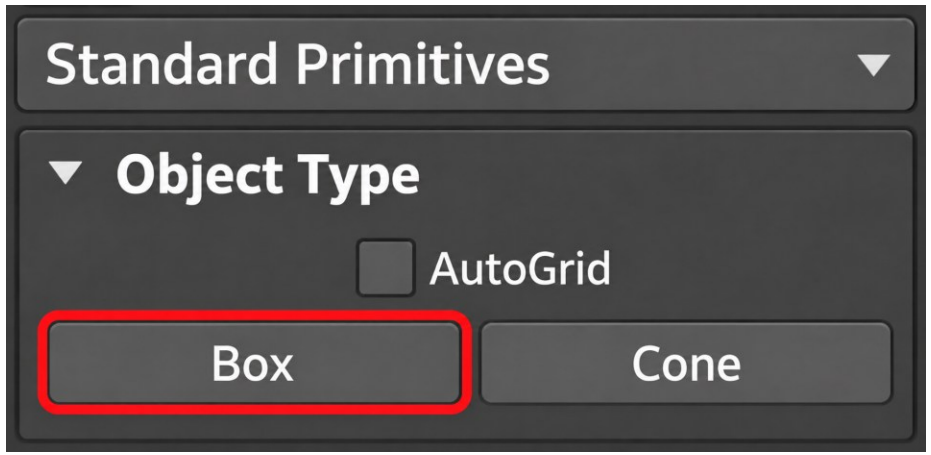


Figure: Box creation options (detail)

- **Role:** Create and modify meshes; organize separate components and animation.
- **Strength:** Fine polygon editing and transform control for vehicle components.
- **Output:** Keep the authoring project and export interchange assets such as box.fbx.

3ds Max modeling and export help: https://help.autodesk.com/view/3DSMAX/2024/ENU/

3ds Max turns dimension and component requirements into model assets for interchange.

3.1.3 Modeling and Scene Construction Tools: Object Organization in 3ds Max

Separate Objects and Combined Meshes

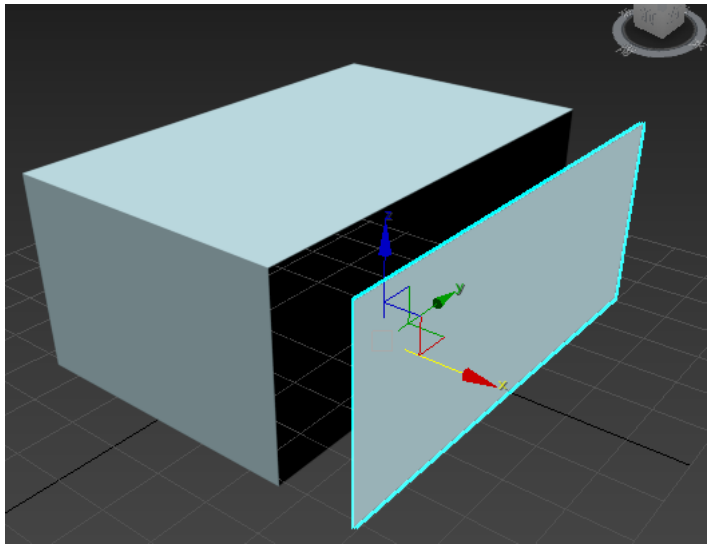


Figure: A box panel detached as a separate object

- **Detach:** Detaching as an object lets selected faces
move or rotate independently.

- **Attach:** Combine several parts into one mesh
for unified management.

- **Vehicle application:** Separate the fuselage and control surfaces,
then configure pivots and control.

Keep independently moving parts as separate objects; combine static parts when appropriate.

3.1.3 Modeling and Scene Construction Tools: Model Handoff

3ds Max Handoff: Save Geometry and Dependencies

Authoring project	→	FBX and dependencies	→	Target engine
Dimensions and components		Interchange data		Assets and materials

- **Geometry handoff:** Preserve units, orientation, and component names; check imported scale against a known dimension.
- **Appearance handoff:** Textures may be linked or embedded; rebuild complex materials for the target engine.

Keep the editable project and texture dependencies so the model can be repaired after transfer to another platform.

3.1.3 Modeling and Scene Construction Tools: Buildings in SketchUp

Build Architecture Quickly from Planar Outlines

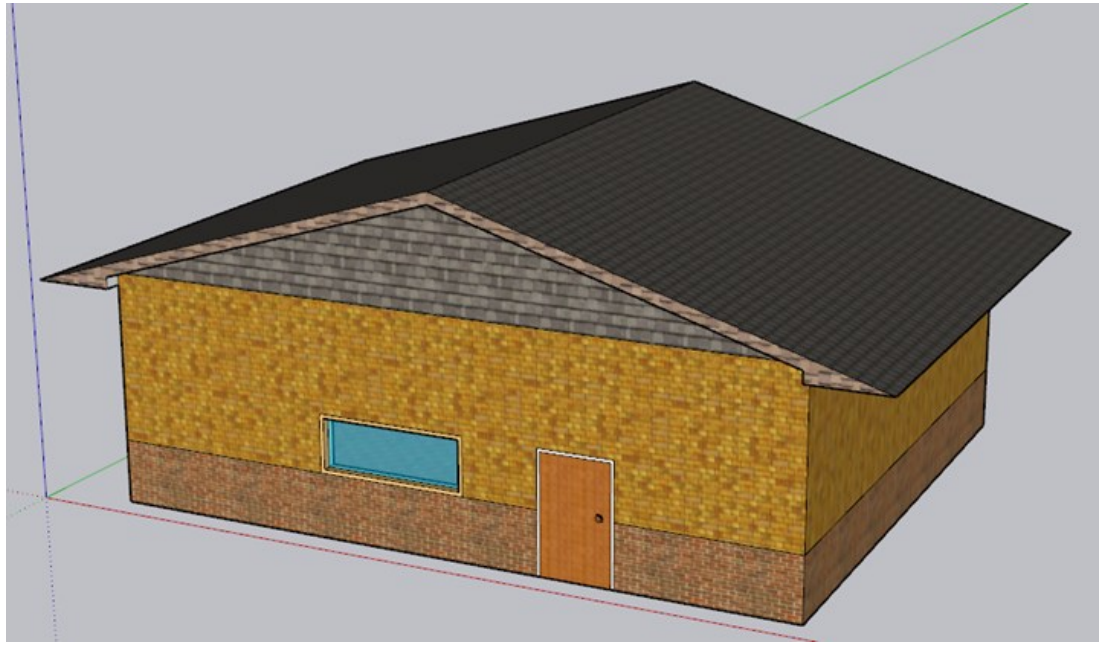


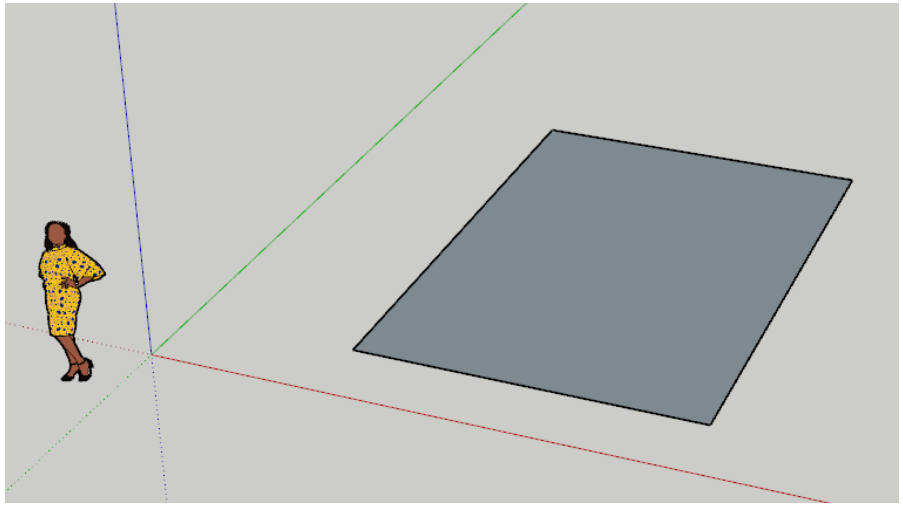
Figure: house.skp from SketchUpUsage

- **Method:** Push/Pull planar outlines into 3D volumes.
- **Suitable tasks:** Houses, rooms, and building layouts
whose dimensions need quick adjustment.
- **Handoff:** house.skp preserves model organization
for later scene decoration.

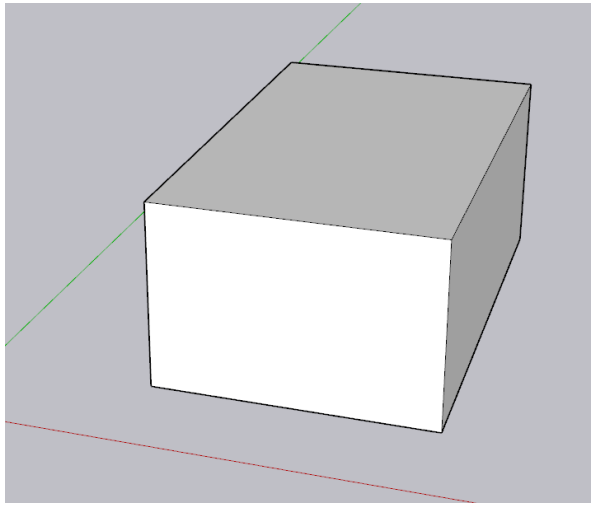
SketchUp Push/Pull tutorial: https://help.sketchup.com/en/sketchup/pushing-and-pulling-shapes-3d

3.1.3 Modeling and Scene Construction Tools: SketchUp Push/Pull

From a Planar Face to a Solid with Push/Pull



Rectangle: Create a planar outline



After Push/Pull: A box

- **Basic sequence:** Closed outline → Create face → Push/Pull depth → Adjust dimensions.
- **Building application:** Define wall, opening, and roof geometry before adding textures and the environment.

3.1.3 Modeling and Scene Construction Tools: Reuse in SketchUp

Organize Buildings with SketchUp Groups and Components

Walls and roof
Isolate edits with groups

Repeated doors and windows
Reuse component definitions

→

One building model
Preserve the object hierarchy

- **Groups:** Enclose geometry to reduce unintended interactions between neighboring edges and faces.
- **Components:** Instances share a definition; changing that definition updates the corresponding instances.

SketchUp components and instances: https://help.sketchup.com/en/sketchup/components

Repeated doors and windows can share a component definition; one edit updates its instances.

3.1.3 Modeling and Scene Construction Tools: The Blender Alternative

Open-Source Mesh Modeling and Asset Creation

- **Capabilities:** Create meshes, UVs, materials, rigs, and animation in one application.
- **Box example:** Object Mode moves the entire box; Edit Mode changes vertices or faces to alter its shape.
- **Course use:** Create a box in Blender and export FBX to continue with this chapter's model import workflow.
- **Save the project:** .blend keeps editable objects; FBX is for interchange. Deliver texture files with it.

Blender mesh editing manual: https://docs.blender.org/manual/en/4.2/modeling/meshes/index.html

Blender combines modeling, materials, and animation to deliver equivalent asset types with a different tool.

3.1.3 Modeling and Scene Construction Tools: Environment Setup in Twinmotion

Add an Environment to Existing Geometry



Figure: The house.skp scene in Twinmotion

- **Input:** Import house.skp while preserving the building's object hierarchy.
- **Processing:** Replace wall materials, add vegetation and lights, and observe appearance changes.
- **Output:** Save house.tm; export UE assets with the example's compatible versions.

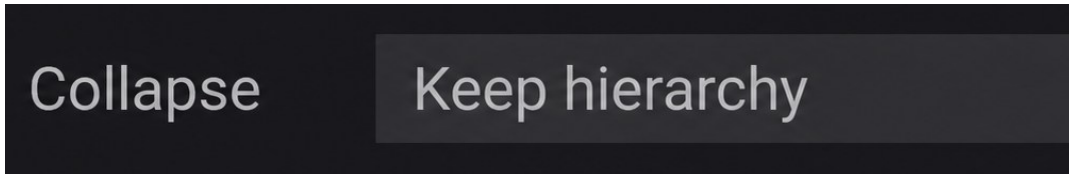
Twinmotion scene creation courses: https://www.twinmotion.com/learning-resources

Modeling tools define geometry; Twinmotion quickly organizes appearance, vegetation, and lighting.

3.1.3 Modeling and Scene Construction Tools: Object Organization in Twinmotion

Choose Object Organization During Twinmotion Import

Geometry import: Collapse options



"Keep hierarchy" preserves the source model's object hierarchy.

Figure: Twinmotion import options

- **Keep hierarchy:** Preserve the source object relationships for editing individual components.
- **Collapse by material:** Combine geometry with the same material to reduce the number of managed objects.
- **Control needs:** Keep independently moved doors, control surfaces, and similar parts separate.

Preserve hierarchy for object-by-object editing; collapse parts that do not need independent control.

3.1.3 Modeling and Scene Construction Tools: Datasmith Scene Handoff

Transfer a Design Scene to UE

Datasmith: Tools and data interchange methods for importing design scenes into UE.

SketchUp / TM
Design and decoration
➡
Datasmith
Scene and companion assets
➡
UE project
Add behavior

- **Handoff files:** .udatasmith stores the scene description; the companion asset folder stores meshes and textures. Copy both together.

- **Version matching:** Use compatible Twinmotion, Datasmith plugin, and UE versions for the example; configure behavior in UE after import.

Transfer the scene description with its companion assets to preserve usable object organization.

3.1.3 Modeling and Scene Construction Tools: The House Toolchain

Tool Roles in the House Example

SketchUp house.skp ➡ **Twinmotion house.tm** ➡ **UE Scene project** ➡ **RflySim Execution and observations**

- **SketchUp:** Geometry and object hierarchy.
- **UE:** Scene objects and runtime behavior.
- **Twinmotion:** Materials, vegetation, and lighting.
- **RflySim:** Motion states, control, and observations.

Datasmith handoff from newer Twinmotion versions to UE:
https://dev.epicgames.com/documentation/en-us/twinmotion/twinmotion-to-unreal-engine-workflow

Tool handoffs transfer object hierarchy, texture dependencies, and runtime interfaces as well as files.

3.1.3 Modeling and Scene Construction Tools: Getting Assets from Fab

Find Reusable Models and Engine Assets

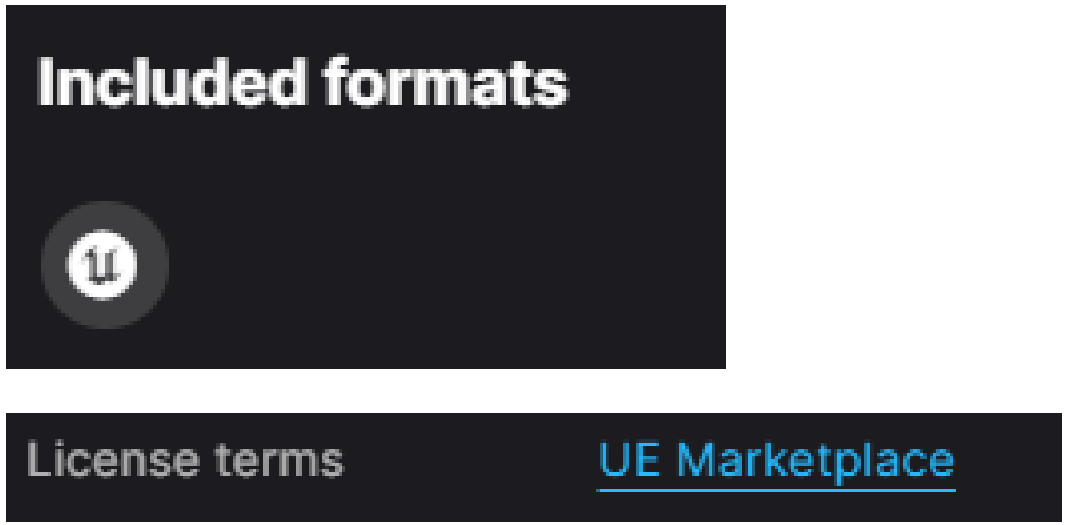


Figure: Formats and licensing on Fab (detail)

Find and download Fab assets: https://www.fab.com/

- **Asset types:** Models, materials, scene content, and plugins serve different purposes.
- **Technical requirements:** Match formats and UE versions; include all texture and plugin dependencies.
- **Usage conditions:** Use and distribute assets under their licenses and retain source records.

Asset marketplaces shorten production; filter by format, version, dependencies, and license.

Chapter Outline

3D Simulation Fundamentals and Tool Selection

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

5 Units in This Section

3.1.1 Background and 3D Simulation Workflow
Textbook: Secs. 3.1.1–3.1.2

3.1.2 Model Creation and Editing
Textbook: Sec. 3.1.2 · Model Creation and Editing

3.1.3 Modeling and Scene Construction Tools
Textbook: Secs. 3.1.3.1 and 3.1.3.3

3.1.4 Auxiliary Tools
Textbook: Sec. 3.1.3.3; data collection: Sec. 3.1.2

3.1.5 Platform Comparison and Selection
Textbook: Secs. 3.1.3.1–3.1.3.2

3.1.4 Auxiliary Tools: Real-Environment Capture

Real-Environment Assets: Scanning, Photogrammetry, and Geodata

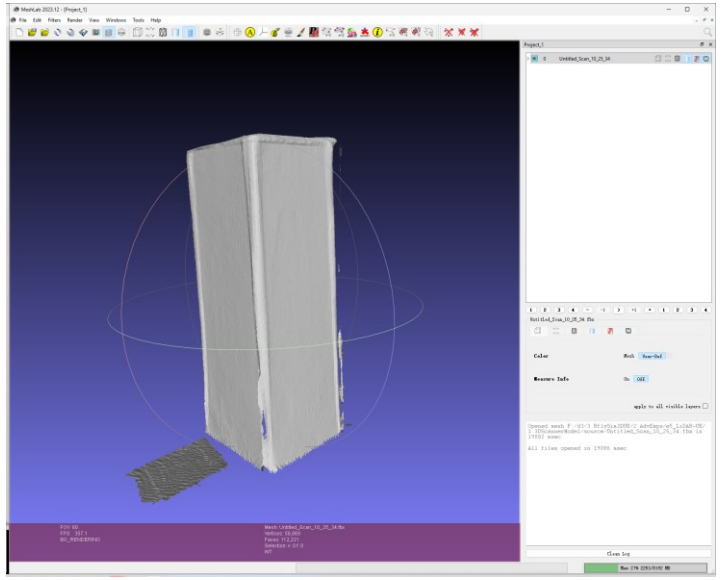


Figure: A scanned model imported into MeshLab (supplied example)

- **Photogrammetry:** Reconstruct geometry and textures from multiple views; clean meshes in MeshLab.
- **Depth / laser scanning:** Sample points or meshes; process point clouds with Open3D.
- **Geodata:** Preserve position and elevation datums. Cesium loads regional data; geographic information systems (GIS) and CityEngine support urban modeling.

Capture provides geometry and appearance; cleaning and georeferencing turn the data into usable assets.

3.1.4 Auxiliary Tools: Repairing Scanned Meshes

MeshLab: Clean and Repair Scanned Meshes

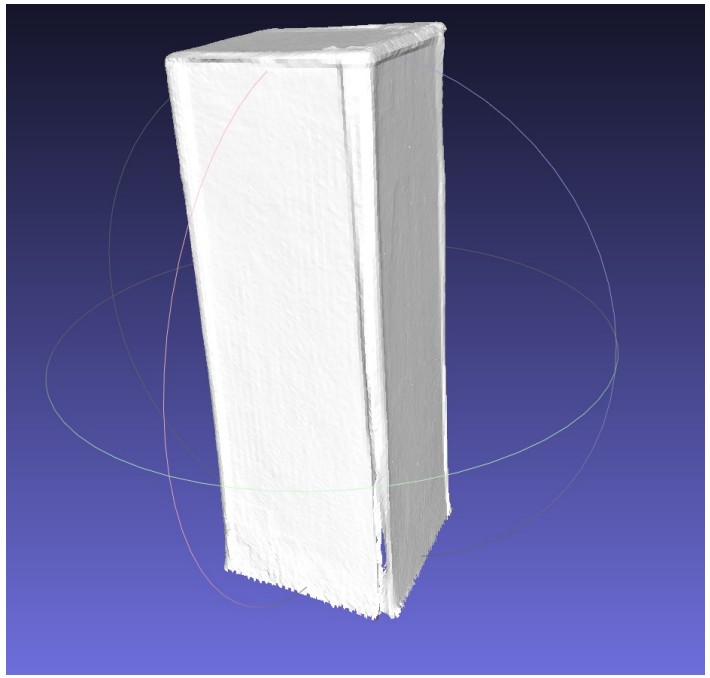


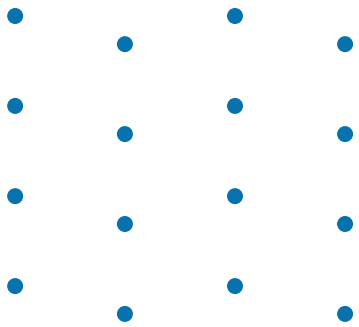
Figure: The cleaned scanned mesh in the example

- **Clean:** Remove background fragments and isolated geometry;
check surface orientation (normals).
- **Repair:** Fill gaps in the capture while preserving
real openings needed by the task.
- **Handoff:** Export assets such as OBJ while retaining
texture and material dependencies.

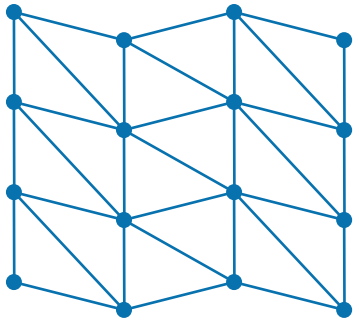
MeshLab mesh processing and scanning tutorials: https://www.meshlab.net/

3.1.4 Auxiliary Tools: Point Clouds and Meshes

Point Clouds and Meshes Store Different Information



Sampled points



Connected mesh

Schematic: The same positions, organized differently

- **Point cloud:** Stores point positions and may include color,
normals, and other attributes.
- **Mesh:** Connects vertices with faces to represent and render a continuous surface.
- **Heightmap:** Stores one height per planar position; it cannot represent both a bridge deck and the ground below.

Point clouds store samples; meshes add surface connectivity for different processing tasks.

3.1.4 Auxiliary Tools: Data Processing with Open3D

Process 3D Data in Python

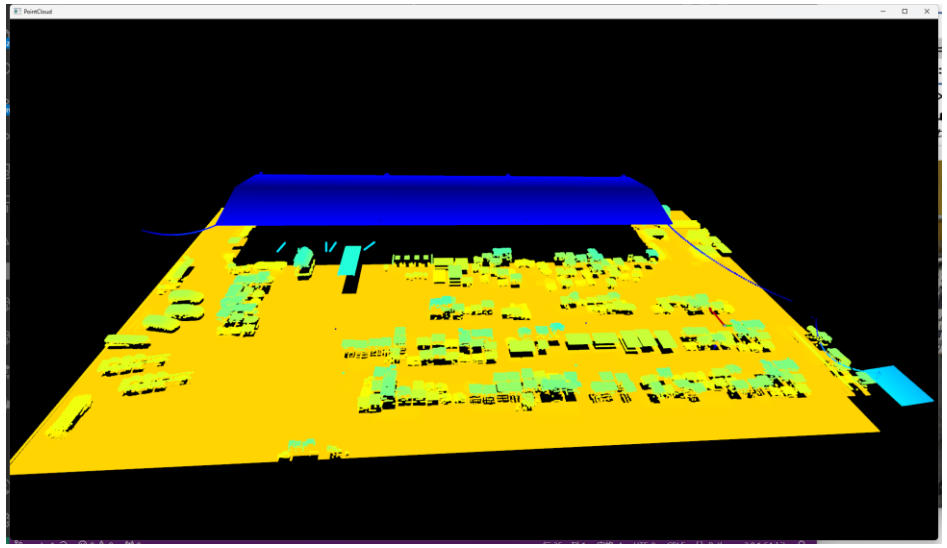


Figure: Terrain point cloud in Open3D

- **Interfaces:** Use Python or C++ to read, write, and process point clouds and meshes.
- **Processing:** Downsample dense points; register overlapping clouds to estimate their relative pose.
- **Example:** VisTerrainPcd.py loads terrain coordinates into a point-cloud object and displays it.

Open3D point-cloud tutorial: https://www.open3d.org/docs/release/tutorial/geometry/pointcloud.html

Open3D organizes geometric samples as data objects for input/output, computation, and visualization.

3.1.4 Auxiliary Tools: Geospatial Loading with Cesium

Load Large 3D Regions by Geographic Location

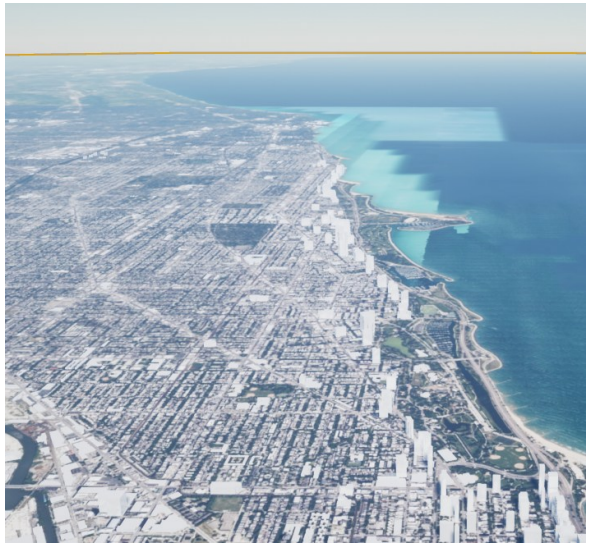


Figure: The city region loaded in the Cesium example

- **Role:** Load terrain, imagery, and 3D Tiles and other geographic assets in UE.
- **Strength:** Organize large datasets by location and viewing distance,
reducing manual modeling of individual objects.
- **Georeferencing:** The origin links geographic and local scene positions;
the elevation datum defines the height reference surface.

Cesium terrain and buildings quickstart: https://cesium.com/learn/unreal/unreal-quickstart/

Georeferencing and tiled loading bring large environments into local simulation experiments.

3.1.4 Auxiliary Tools: Online and Offline Assets

Online and Offline Cesium Assets

- **Online loading:** Fetch data from services as needed; network access and permissions affect completeness.
- **Offline loading:** Deploy licensed datasets locally and organize experiments around a defined version.
- **Data preparation:** Tools such as CesiumLab convert and publish 3D geographic data for Cesium to load.
- **Reproducibility:** Record the region, source, and version; match offline examples and licenses to the installed version.

Online services simplify access and updates; offline datasets help fix experimental conditions.

3.1.4 Auxiliary Tools: Procedural Cities

CityEngine and GIS: Build Urban Environments at Scale

GIS data
Parcels, imagery, coordinates

➡

CityEngine rules
Building generation

➡

UE / RflySim
Scenes and experiments

- **Geographic information system (GIS):** Manage spatial references, regional extents, and geospatial processing.
- **Rules:** Apply building-height and floor-count rules to parcels to generate volumes and facades.

Learn procedural modeling in CityEngine:
https://learn.arcgis.com/en/paths/learn-the-basics-of-procedural-modeling/

GIS manages spatial data; CityEngine applies rules to turn regional geometry into buildings.

Chapter Outline

3D Simulation Fundamentals and Tool Selection

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

5 Units in This Section

3.1.1 Background and 3D Simulation Workflow
Textbook: Secs. 3.1.1–3.1.2

3.1.2 Model Creation and Editing
Textbook: Sec. 3.1.2 · Model Creation and Editing

3.1.3 Modeling and Scene Construction Tools
Textbook: Secs. 3.1.3.1 and 3.1.3.3

3.1.4 Auxiliary Tools
Textbook: Sec. 3.1.3.3; data collection: Sec. 3.1.2

3.1.5 Platform Comparison and Selection
Textbook: Secs. 3.1.3.1–3.1.3.2

3.1.5 Platform Comparison and Selection: Engines and Frameworks

3D Engines, Graphics Toolkits, and Frameworks

Choice	Level and Organization	When It Fits
UE	Engine: Levels, Actors, components, Blueprints	Existing UE assets or RflySim project
Unity / Godot	Engines: Scenes, objects / nodes, scripts	Scripting experience; matching ecosystem
OpenSceneGraph (OSG) / Panda3D	OSG: Graphics toolkit Panda3D: 3D engine	Existing code or custom rendering
Delta3D	Simulation application framework	Existing framework and interface dependencies

Existing assets and team code determine migration costs; choose an engine to meet project deliverables.

3.1.5 Platform Comparison and Selection: Simulation Platforms

Choose Simulation Platforms by Models and Interfaces

Platform	Main Use	Interfaces to Study
RflySim	Unmanned-system development and verification	States, control, sensors; HITL*
Gazebo	Robot simulation and system integration	Models, sensors, communication
CarSim	Vehicle dynamics and co-simulation	Vehicle inputs/outputs; model parameters
Original AirSim project	UAV and car simulation; vision experiments	Images, states, external control

* Hardware-in-the-loop (HITL; HIL in the textbook).

Specialized platforms provide domain models; general platforms emphasize object extension and system integration.

3.1.5 Platform Comparison and Selection: Isaac Sim and MuJoCo

Choose Tools for Visual Data and Contact Control

- **Isaac Sim:** Uses USD to organize scenes and integrates physics, sensors, and synthetic-data tools.
- **MuJoCo:** Uses MJCF for articulated models and computes contact and control responses.
- **Grasp control:** With an existing MJCF model, iterate contact and actuator control in MuJoCo.
- **Vision tasks:** Consider the relevant Isaac Sim tools when cameras, lighting, and labeled data need to be organized.

Isaac Sim sensor configuration and data access:
https://docs.isaacsim.omniverse.nvidia.com/latest/sensors/index.html

Assets can be reused; simulators differ in scene representation, actuators, and experimental workflows.

3.1.5 Platform Comparison and Selection: Asset Tools

Choose Asset Creation Tools by Task

Problem to Solve	Tasks and Tools	Output
Buildings and vehicle components	Simple buildings: SketchUp Detailed components: 3ds Max / Blender	Geometry and separate components
Materials and environment setup	Appearance: Twinmotion Behavior: UE	Editable scene
Mesh and point-cloud processing	Interactive mesh cleaning: MeshLab Programmatic point-cloud processing: Open3D	Processed geometric data
Large geographic scenes	Geographic data loading: Cesium Procedural cities: CityEngine + GIS	Georeferenced regional assets

Identify the missing data, then learn the tool that creates or repairs it.

3.1.5 Platform Comparison and Selection: Introductory Examples

Tool Learning Entry Points in the Chapter 3 Examples

Example Folder	Introductory Topic	Checkable Output
1.UEBlueprintUsage **2.3dsMaxUsage**	Behavior and geometry editing	Box Blueprint / box.fbx
4.SketchUpUsage **5.TwinmotionUsage**	Buildings and scene decoration	house.skp / house.tm
3.CesiumForUnrealUsage	Terrain and city loading	Georeferenced scene
6.EpicFabUsage	Asset selection and acquisition	Assets meeting project requirements

Example Gallery → Chapter 3 → 0.ApiExps → e0_DevToolsUsage. Start with Readme.pdf.

Complete one small deliverable, then move to the corresponding full asset-development experiment.

3.1.5 Platform Comparison and Selection: Asset Practice

Prepare Scene Assets by Reuse, Modeling, or Scanning

Starting Point	Relative Folder Within Chapter 3	Output
Existing model	1.BasicExps/e4_CusLoadObj/ 2.loadSimpleMesh	test.glb Scene instances
Manual modeling	0.ApiExps/e0_DevToolsUsage/ 4.SketchUpUsage → 5.TwinmotionUsage	house.skp house.tm
Indoor scan	2.AdvExps/e5_LiDAR-UE/ 2.3DScannerScene	Scanned scene

Example Gallery → Chapter 3: Open the listed folder and read Readme.pdf first.

Get existing assets running first; create more when needed, then connect observations to algorithms.

3.1.5 Platform Comparison and Selection: Data Practice

Read Terrain Points, Vehicle States, and Images

Data Task	Relative Folder and Script Within Chapter 3	Data Obtained
Terrain point cloud	1.BasicExps/e3_RflySim3DTerrainPcd Script: VisTerrainPcd.py	Point-cloud object 3D display
States and images	0.ApiExps/e6_RflySim3DCtrlAPI/ 10.GetCamObjDemo Script: GetCamObjDemoWithCam.py	State metadata Image arrays

Example Gallery → Chapter 3: Terrain examples process geometry; state/image examples request observations.

Terrain points support geometry processing; state and image interfaces deliver scene data to algorithms.

3.1.5 Platform Comparison and Selection: What Comes Next

How Section 3.1 Connects to Later Sections

Later Content	Introduced in 3.1	New Knowledge
3.2 UE Scenes and Rendering	Meshes, materials, UE	Object updates, rendering, Blueprints
3.3–3.4 Data and Interfaces	Observations versus assets	Imaging, sampling, coordinates, and data formats
3.5 Scene and Vehicle Assets	Formats and tool outputs	Import, configuration, baking, and deployment
3.6–3.7 Practice and Applications	Task-based selection	Reproduce, modify, explain results

To bring assets into algorithms, understand runtime states, sampling mechanisms, and interface configuration.