Scene Builder Architecture Document

Overview

The Scene Builder is a WebGL-based 3D scene composition tool that follows the **Model-View-Controller (MVC)** architecture pattern. It allows users to import GLB/GLTF models, position them in 3D space, configure lighting and wind effects, and save/load scenes.

Architecture Diagram

```
graph TB
    subgraph App["Scene Builder Application"]
        subgraph Controller["CONTROLLER (index.js)"]
            C1["Orchestrates Model and View"]
            C2["Handles user input"]
            C3["Manages lifecycle"]
            C4["Scene file save/load"]
        end

        subgraph ModelLayer["MODEL LAYER"]
            Scene["Scene (scene.js)
• Objects array
• Selection state
• Groups
• Transform ops"]
            SceneGraph["SceneGraph (core)
• Spatial hierarchy
• Raycasting
• Bounds computation"]
            LightingManager["LightingManager
• SunLight
• HDRLight"]
            WindManager["WindManager
• Global wind params
• Physics simulation"]
        end

        subgraph ViewLayer["VIEW LAYER"]
            Viewport["Viewport (viewport.js)
• WebGL context
• Render loop
• Camera controller
• Transform gizmo
• Renderers"]
            Panels["Component Panels
• ObjectsPanel
• ObjectPanel
• EnvironmentPanel"]
        end
```

```
        Controller --> ModelLayer
        Controller --> ViewLayer
        ModelLayer <-.->|"Callbacks"| ViewLayer
    end
```

Core Components

1. Controller (`index.js`)

The central orchestrator that initializes all components and manages communication between Model and View.

Responsibilities:

- Create and wire up Model (Scene) and View (Viewport)
- Handle user input (keyboard shortcuts, menu bar)
- Manage scene file save/load operations
- Coordinate state sync between Model → View
- Instantiate UI panels with shared context

Key Functions:

```
// Model → View sync
updateGizmoTarget()      // Sync selection to gizmo position
updateRenderData()       // Push objects/settings to viewport
updateLightingState()    // Push lighting params to viewport

// Event handlers (wired to Viewport callbacks)
handleGizmoTransform(type, value)   // Apply transform from gizmo
handleObjectClicked(objectId)        // Select object on click
handleUniformScaleChange(newScale)   // Scale from mouse drag
```

---

2. Model Layer

2.1 Scene (`scene.js`)

The primary data model managing 3D objects, selection state, and groups.

State:

```
{
  objects: [],          // Array of scene objects
  selectedIds: Set(),   // Currently selected object IDs
  groups: Map(),        // Group ID → { name, childIds, collapsed }
}
```

Scene Object Structure:

```
{
  id: 'object-1',
  name: 'Tree',
  modelPath: '/models/tree.glb',
  model: GLBModel,                    // Parsed GLB data
  renderer: ObjectRenderer,            // GPU resources
  position: [0, 0, 0],
  rotation: [0, 45, 0],               // Degrees
  scale: [1, 1, 1],
  groupId: 'group-1' | null,
}
```

Event Callbacks:

```
scene.onSelectionChanged = () => { ... }
scene.onObjectAdded = () => { ... }
scene.onObjectRemoved = () => { ... }
scene.onGroupChanged = () => { ... }
```

2.2 SceneGraph (`core/sceneGraph.js`)

Spatial data structure for efficient raycasting and bounds queries.

API:

```
sceneGraph.addNode(id, { position, rotation, scale, localBounds })
sceneGraph.updateNode(id, { position, rotation, scale })
sceneGraph.removeNode(id)
sceneGraph.castRay(origin, direction) → { node, distance }
```

2.3 LightingManager (`lights.js`)

OOP lighting system with multiple light types.

Class Hierarchy:

```
classDiagram
    class Light {
        +type
        +enabled
        +intensity
        +color
        +castsShadow
```

```
        +getDirection()
        +getAmbient()
        +getLightParams()
        +serialize()
        +deserialize()
    }

    class SunLight {
        +azimuth
        +elevation
        +shadowResolution
        +getDirection()
        +getAmbient()
        +getSunColor()
    }

    class HDRLight {
        +texture
        +exposure
        +filename
        +setTexture()
    }

    class PointLight {
        +position
        +radius
        +falloff
        +getDirectionFrom()
        +getAttenuation()
    }

    Light <|-- SunLight : extends
    Light <|-- HDRLight : extends
    Light <|-- PointLight : extends
```

Key Methods:

```
lightingManager.setMode('sun' | 'hdr')
lightingManager.getLightParams()   // Returns shader uniforms
lightingManager.sunLight.getSunColor()    // Dynamic sunset colors
lightingManager.sunLight.getAmbient()     // Dynamic ambient
```

2.4 WindManager (`wind.js`)

Physics-based wind simulation with spring dynamics.

Global Parameters:

```
{
  enabled: boolean,
  strength: 0-2,
  direction: degrees (0-360),
  turbulence: 0-1,
  gustStrength: 0-1,
  gustFrequency: number,
}
```

Per-Object Wind Settings:

```
{
  enabled: boolean,
  leafMaterialIndices: Set<number>,    // Materials affected as leaves
  branchMaterialIndices: Set<number>,  // Materials affected as branches
  influence: 0-2,
  stiffness: 0-1,
  anchorHeight: number,
  // Runtime physics state
  displacement: [x, z],
  velocity: [x, z],
}
```

---

3. View Layer

3.1 Viewport (`viewport.js`)

The main rendering view encapsulating all WebGL operations.

State Received from Controller:

```
// setRenderData()
{
  objects: [],
  objectWindSettings: Map,
  objectTerrainBlendSettings: Map,
  selectedIds: Set,
  getModelMatrix: (obj) => mat4,
}

// setLightingState()
{
  mode: 'sun' | 'hdr',
  sunAzimuth, sunElevation,
  shadowEnabled, shadowResolution,
  hdrTexture, hdrExposure,
  lightColor: [r, g, b],    // Dynamic sun color
```

```
  ambient: number,           // Dynamic ambient
}

// setWindParams()
{
  enabled, strength, direction: [x, z],
  turbulence, gustStrength, time, debug,
}
```

Callbacks to Controller:

```
onFps(fps)                          // FPS counter
onUpdate(deltaTime)                 // Per-frame update (wind physics)
onGizmoTransform(type, value)       // Transform value changed
onGizmoDragEnd()                    // Transform drag completed
onUniformScaleChange(newScale)      // Mouse-based uniform scale
onObjectClicked(objectId, shiftKey) // Object ray-hit
onBackgroundClicked(shiftKey)       // Background clicked
```

3.2 Renderers

Renderer	Purpose
`objectRenderer.js`	Render GLB models with PBR-style shading, wind, terrain blend
`shadowRenderer.js`	Shadow map generation with depth encoding
`skyRenderer.js`	Procedural sun sky gradient + HDR environment
`gridRenderer.js`	Reference grid plane
`originMarkerRenderer.js`	Camera origin indicator
`depthPrePassRenderer.js`	Depth texture for terrain blend effect
`transformGizmo.js`	3D translate/rotate gizmo with screen-space overlays

3.3 Camera Controller (`cameraController.js`)

Orbit camera with pan/zoom support.

State:

```
{
  angleX, angleY,     // Orbit angles
  distance,           // Zoom distance
  originX/Y/Z,        // Look-at point
  offsetX/Y/Z,        // Pan offset
}
```

4. UI Layer - Component Panels

4.1 Panel Architecture

Each panel receives a shared `PanelContext` object for accessing scene state.

Panel Interface:

```
createXxxPanel(containerElement, panelContext) → {
  update(),      // Refresh UI from current state
  destroy(),     // Cleanup event listeners
}
```

4.2 PanelContext (`panelContext.js`)

Dependency injection container providing panels with:

- Scene reference
- GL context
- Wind/Lighting managers
- Per-object settings accessors
- Callbacks to Controller

```
panelContext = {
  // Core references
  scene, gl, windManager, lightingManager,

  // Per-object settings
  getObjectWindSettings(id) → settings,
  setObjectWindSettings(id, settings),
  getObjectTerrainBlend(id) → settings,
  setObjectTerrainBlend(id, settings),

  // Callbacks
  onGizmoModeChange(mode),
  onTransformUpdate(),
  onSelectionChanged(),
  setShadowResolution(res),
  setLightMode(mode),
  setHDRTexture(texture),
  onWindChanged(),
  onLightingChanged(),
}
```

4.3 Panel Components

Panel	Purpose
`objectsPanel.js`	Scene hierarchy list, import controls, group management
`objectPanel.js`	Selected object transform, wind modifiers, terrain blend
`environmentPanel.js`	Sun/HDR lighting controls, shadow settings, global wind

Data Flow

1. Object Import Flow

```
sequenceDiagram
    participant User
    participant ObjectsPanel
    participant Scene
    participant Loaders
    participant Controller

    User->>ObjectsPanel: drops .glb file
    ObjectsPanel->>Scene: addObject(modelPath)
    Scene->>Loaders: loadGLB(url)
    Loaders-->>Scene: GLB model data
    Scene->>Scene: createObjectRenderer()
    Scene->>Scene: addToSceneGraph()
    Scene->>Controller: onObjectAdded callback
    Controller->>ObjectsPanel: update()
    Controller->>Controller: updateRenderData()
```

2. Selection Flow

```
sequenceDiagram
    participant User
    participant Viewport
    participant SceneGraph
    participant Controller
    participant Scene
    participant Panels

    User->>Viewport: clicks canvas
    Viewport->>SceneGraph: castRay(origin, direction)
    SceneGraph-->>Viewport: hit result
    Viewport->>Controller: onObjectClicked(id, shiftKey)
    Controller->>Scene: select(id, { additive })
    Scene->>Controller: onSelectionChanged callback
    Controller->>Panels: objectsPanel.update()
    Controller->>Panels: objectPanel.update()
    Controller->>Controller: updateGizmoTarget()
    Controller->>Controller: updateRenderData()
```

3. Transform Flow

```
sequenceDiagram
    participant User
    participant TransformGizmo
    participant Controller
    participant Scene
    participant ObjectPanel

    User->>TransformGizmo: drags gizmo
    TransformGizmo->>TransformGizmo: compute new transform value
    TransformGizmo->>Controller: onGizmoTransform('position', [x,y,z])
    Controller->>Scene: applyTransform('position', [x,y,z])
    Scene->>Scene: update object.position
    Scene->>Scene: update sceneGraph node
    Controller->>ObjectPanel: update()
```

4. Render Loop Flow

```
flowchart TD
    A[animationLoop.start] --> B[render deltaTime]

    B --> C[onUpdate dt]
    C --> C1[windManager.update dt]
    C1 --> C2[updateObjectPhysics]
    C2 --> C3[viewport.setWindParams]

    B --> D{Sun mode + shadows?}
    D -->|Yes| D1[Shadow Pass]
    D1 --> D2[beginShadowPass]
    D2 --> D3[for each object: shadowRenderer.render]
    D3 --> D4[endShadowPass]

    B --> E{Terrain blend enabled?}
    E -->|Yes| E1[Depth Pre-Pass]
    E1 --> E2[beginPass]
    E2 --> E3[for each object: depthPrePass.render]
    E3 --> E4[endPass]

    B --> F[Sky Render]
    F --> F1{HDR mode?}
    F1 -->|Yes| F2[renderHDRSky]
    F1 -->|No| F3[renderSunSky]

    B --> G[Grid + Origin Marker]

    B --> H[Object Render]
    H --> H1[for each object]
    H1 --> H2[objectRenderer.render
```

```
vpMatrix, modelMatrix
lightParams, windParams
terrainBlendParams]

    B --> I[Gizmo Render]
    I --> I1[transformGizmo.render vpMatrix]

    B --> J[Debug Overlay]
    J --> J1[shadow thumbnail]
```

File Structure

```
graph TD
    subgraph sceneBuilder["src/demos/sceneBuilder/"]
        index["index.js
Controller - main entry"]
        viewport["viewport.js
View - WebGL rendering"]
        scene["scene.js
Model - object management"]
        styles["styles.js
HTML template + CSS"]

        subgraph panels["componentPanels/"]
            panelIndex["index.js"]
            panelContext["panelContext.js"]
            objectsPanel["objectsPanel.js"]
            objectPanel["objectPanel.js"]
            envPanel["environmentPanel.js"]
        end

        subgraph renderers["Renderers"]
            objectRenderer["objectRenderer.js"]
            shadowRenderer["shadowRenderer.js"]
            skyRenderer["skyRenderer.js"]
            gridRenderer["gridRenderer.js"]
            originMarker["originMarkerRenderer.js"]
            depthPrePass["depthPrePassRenderer.js"]
        end

        subgraph systems["Systems"]
            lights["lights.js"]
            wind["wind.js"]
            camera["cameraController.js"]
            gizmo["transformGizmo.js"]
        end

        subgraph utils["Utilities"]
            raycast["raycastUtils.js"]
            serializer["sceneSerializer.js"]
```

```
            hdr["hdrLoader.js"]
            chunks["shaderChunks.js"]
            shaderMgr["shaderManager.js"]
            shaderDebug["shaderDebugPanel.js"]
        end
    end

    subgraph core["Core Dependencies"]
        sceneGraph["core/sceneGraph.js"]
        animLoop["core/animationLoop.js"]
        loaders["loaders.js"]
    end

    index --> viewport
    index --> scene
    index --> panels
    viewport --> renderers
    viewport --> systems
    scene --> sceneGraph
```

Key Design Patterns

1. Factory Functions

All components use factory functions (not classes) for creation:

```
export function createScene(gl, sceneGraph) { ... }
export function createViewport(canvas, options) { ... }
```

2. Callback-Based Communication

View communicates with Controller via callbacks, not direct references:

```
const viewport = createViewport(canvas, {
  onObjectClicked: (id) => scene.select(id),
  onGizmoTransform: (type, value) => scene.applyTransform(type, value),
});
```

3. Unidirectional Data Flow

```
flowchart LR
    UserInput[User Input] --> Controller
    Controller --> Model
    Model --> Controller
    Controller --> View
    Controller --> UIPanels[UI Panels]
```

4. Dependency Injection (Panels)

Panels receive all dependencies via `PanelContext`:

```
const panelContext = createPanelContext({ scene, gl, windManager, ... });
const objectsPanel = createObjectsPanel(container, panelContext);
```

Shader Architecture

Shared GLSL Chunks (`shaderChunks.js`)

```
// Wind displacement
${simplexNoise}
${windUniforms}
${windDisplacement}

// Terrain blend
${terrainBlendUniforms}
${terrainBlendFunctions}
```

Object Renderer Uniforms

Category	Uniforms
Transform	`uModelViewProjection`, `uModel`
Material	`uBaseColor`, `uTexture`, `uHasTexture`
Lighting	`uLightDir`, `uAmbientIntensity`, `uLightColor`, `uLightMode`
HDR	`uHdrTexture`, `uHasHdr`, `uHdrExposure`
Shadow	`uShadowMap`, `uLightSpaceMatrix`, `uShadowEnabled`, `uShadowDebug`
Wind	`uWindEnabled`, `uWindTime`, `uWindStrength`, `uWindDirection`, etc.
Terrain	`uTerrainBlendEnabled`, `uTerrainBlendDistance`, `uSceneDepthTexture`

Serialization Format

Scene files are JSON with this structure:

```
{
  "name": "My Scene",
  "objects": [
```

```
      {
        "name": "Tree",
        "modelPath": "/models/tree.glb",
        "position": [0, 0, 0],
        "rotation": [0, 45, 0],
        "scale": [1, 1, 1],
        "groupId": "group-1"
      }
    ],
    "groups": [
      {
        "id": "group-1",
        "name": "Forest",
        "childIds": ["object-1", "object-2"],
        "collapsed": true
      }
    ],
    "camera": {
      "angleX": 0.5, "angleY": 0.3,
      "distance": 5,
      "originX": 0, "originY": 0, "originZ": 0
    },
    "lighting": {
      "mode": "sun",
      "sunAzimuth": 45,
      "sunElevation": 30,
      "shadowEnabled": true,
      "shadowResolution": 2048
    },
    "wind": {
      "enabled": true,
      "strength": 0.5,
      "direction": 45,
      "turbulence": 0.5
    },
    "objectWindSettings": [
      { "enabled": true, "leafMaterialIndices": [0], "influence": 1.0, ... }
    ],
    "objectTerrainBlendSettings": [
      { "enabled": false, "blendDistance": 0.5 }
    ]
  }
```

Extension Points

Adding a New Panel

1. Create `componentPanels/myPanel.js`
2. Implement `createMyPanel(element, context)` returning `{ update(), destroy() }`
3. Add to `componentPanels/index.js` exports
4. Instantiate in Controller `init()`

Adding a New Light Type

1. Extend `Light` class in `lights.js`
2. Implement `getLightParams()` with shader uniforms
3. Handle in `LightingManager.getLightParams()`
4. Update object shader to recognize new mode

Adding a New Renderer

1. Create `myRenderer.js` following factory pattern
2. Return object with `render()` and `destroy()` methods
3. Initialize in `viewport.js` `initGL()`
4. Call in render loop at appropriate pass

Performance Considerations

1. **Shadow Map**: Only regenerated when lighting state changes
2. **Depth Pre-Pass**: Only runs when terrain blend is enabled
3. **Object Culling**: Not currently implemented (potential optimization)
4. **GPU Resource Cleanup**: `destroy()` methods clean up all WebGL resources
5. **State Diffing**: Viewport uses `Object.assign` for state updates (full replace)

Known Limitations

1. **Multi-select Scale**: Uniform scale only works for single selection
2. **Point Lights**: Defined but not rendered (future feature)
3. **Undo/Redo**: Not implemented
4. **HDR Reload**: Requires manual reload after scene load
5. **Group Nesting**: Flat groups only, no nested hierarchy