ДОДАТОК А

ФРАГМЕНТИ ЗАСТОСОВАНИХ ЛІСТИНГІВ

```swift
class TerrainScene {
  let scene: SCNScene
  private let terrainGenerator: TerrainGenerator

  // Налаштування камери
  private func setupCamera() {
    let cameraNode = SCNNode()
    cameraNode.camera = SCNCamera()
    cameraNode.position = SCNVector3(x: 0, y: 10, z: 10)
    cameraNode.eulerAngles = SCNVector3(x: -Float.pi / 4, y: 0, z: 0)
  }

  // Генерація нового ландшафту
  func generateNewTerrain() {
    let terrainNode = terrainGenerator.generateTerrain()
    let waterNode = terrainGenerator.generateWater()
    scene.rootNode.addChildNode(terrainNode)
    scene.rootNode.addChildNode(waterNode)
  }
}
```

Рисунок А.1 – Фрагмент реалізації класу TerrainScene з налаштуванням камери та генерацією ландшафту

```swift
class TerrainGenerator {
  private let meshBuilder: TerrainMeshBuilder
  private let noiseGenerator: NoiseGenerator
  private let gridSize: Int
  private let waterLevel: Float

  // Генерація ландшафту
  func generateTerrain() -> SCNNode {
    let heights = generateHeightMap()
    let geometry = meshBuilder.createGeometry(from: heights)
    return createTerrainNode(with: geometry, heights: heights)
  }

  // Генерація води
  func generateWater() -> SCNNode {
    let waterGeometry = SCNPlane(width: CGFloat(gridSize), height: CGFloat(gridSize))
    return createWaterNode(with: waterGeometry)
  }
}

```

Рисунок А.2 – Фрагмент реалізації класу TerrainGenerator з методами генерації ландшафту та води

```swift
class TerrainMeshBuilder {
  private let gridSize: Int

  // Створення геометрії
  func createGeometry(from heights: [Float]) -> SCNGeometry {
    let (vertices, indices) = createVerticesAndIndices(from: heights)
    var normals = createNormals(vertices: vertices, indices: indices)

    return SCNGeometry(
      sources: [
        SCNGeometrySource(vertices: vertices),
        SCNGeometrySource(normals: normals),
      ],
      elements: [createGeometryElement(indices: indices)]
    )
  }

  // Розрахунок нормалей
  private func createNormals(vertices: [SCNVector3], indices: [UInt32]) -> [SCNVector3] {
    // Реалізація розрахунку нормалей
  }
}
```

Рисунок А.3 – Фрагмент реалізації класу TerrainMeshBuilder з побудовою геометрії на основі карти висот

```swift
class NoiseGenerator {
  private let p: [Int]

  // Генерація шуму
  func noise(x: Float, y: Float) -> Float {
    // Реалізація шуму Перліна
  }

  // Допоміжні функції
  private func fade(_ t: Float) -> Float {
    return t * t * t * (t * (t * 6 - 15) + 10)
  }

  private func lerp(_ t: Float, _ a: Float, _ b: Float) -> Float {
    return a + t * (b - a)
  }
}
```

Рисунок А.4 – Фрагмент реалізації класу NoiseGenerator для обчислення шуму Перліна

```swift
class LandscapeViewController: UIViewController {
  private let sceneView = SCNView()
  private let terrainScene = TerrainScene()

  // Налаштування UI
  private func setupUI() {
    setupSceneView()
    setupGenerateButton()
  }

  // Налаштування сцени
  private func setupScene() {
    sceneView.scene = terrainScene.scene
    sceneView.allowsCameraControl = true
    sceneView.autoenablesDefaultLighting = true
  }

  // Обробка генерації нового ландшафту
  @objc private func generateButtonTapped() {
    terrainScene.generateNewTerrain()
  }
}
```

Рисунок А.5 – Фрагмент класу LandscapeViewController з логікою відображення сцени та генерації ландшафту

```swift
class NoiseGenerator {
  private let p: [Int]

  init() {
    var permutation = Array(0...255)
    permutation.shuffle()
    p = permutation + permutation
  }
}
```

Рисунок А.6 – Ініціалізація масиву перестановок у класі NoiseGenerator для детермінованої генерації шуму

```
func noise(x: Float, y: Float) -> Float {
  let X = Int(floor(x)) & 255
  let Y = Int(floor(y)) & 255

  let xf = x - floor(x)
  let yf = y - floor(y)

  let u = fade(xf)
  let v = fade(yf)

  let A = p[X] + Y
  let B = p[X + 1] + Y

  return lerp(
    v,
    lerp(
      u,
      grad(p[A], xf, yf),
      grad(p[B], xf - 1, yf)),
    lerp(
      u,
      grad(p[A + 1], xf, yf - 1),
      grad(p[B + 1], xf - 1, yf - 1)))
}
```

Рисунок А.7 – Функція генерації шуму Перліна на основі інтерполяції градієнтів у двохвимірному просторі

```
private func fade(_ t: Float) -> Float {
  return t * t * t * (t * (t * 6 - 15) + 10)
}

private func lerp(_ t: Float, _ a: Float, _ b: Float) -> Float {
  return a + t * (b - a)
}

private func grad(_ hash: Int, _ x: Float, _ y: Float) -> Float {
  let h = hash & 15
  let u = h < 8 ? x : y
  let v = h < 4 ? y : (h == 12 || h == 14 ? x : 0)
  return ((h & 1) == 0 ? u : -u) + ((h & 2) == 0 ? v : -v)
}
```

Рисунок А.8 – Допоміжні функції для генерації шуму Перліна: fade, lerp та grad

```swift
struct TerrainParameters {
  let gridSize: Int
  let heightScale: Float
  let baseFrequency: Float
  let baseAmplitude: Float
  let octaves: Int
  let persistence: Float
  let lacunarity: Float
}
```

Рисунок А.9 – Структура TerrainParameters для параметризації процесу генерації багатооктавного шуму

```swift
func generateHeightMap() -> [Float] {
  var heights: [Float] = []

  for row in 0..<gridSize {
    for col in 0..<gridSize {
      let height = generateHeightAt(x: col, y: row)
      heights.append(height)
    }
  }

  return heights
}
```

Рисунок А.10 – Функція generateHeightMap для створення карти висот із використанням багатооктавного шуму

```swift
private func generateHeightAt(x: Int, y: Int) -> Float {
  var height: Float = 0
  var amplitude = baseAmplitude
  var frequency = baseFrequency

  for _ in 0..<octaves {
    let nx = Float(x) * frequency
    let ny = Float(y) * frequency
    height += noiseGenerator.noise(x: nx, y: ny) * amplitude
    amplitude *= persistence
    frequency *= lacunarity
  }

  return height * heightScale
}
```

Рисунок А.11 – Функція generateHeightAt для обчислення висоти точки за допомогою багатооктавного шуму Перліна

```swift
struct TerrainPresets {
  static let mountains = TerrainParameters(
    gridSize: 50,
    heightScale: 2.0,
    baseFrequency: 0.1,
    baseAmplitude: 1.0,
    octaves: 4,
    persistence: 0.5,
    lacunarity: 2.0
  )

  static let hills = TerrainParameters(
    gridSize: 50,
    heightScale: 1.0,
    baseFrequency: 0.05,
    baseAmplitude: 0.5,
    octaves: 3,
    persistence: 0.3,
    lacunarity: 1.5
  )
}
```

Рисунок А.12 – Структура TerrainPresets із прикладами параметрів для гірського та пагористого ландшафтів

```
class TerrainGenerator {
  private let noiseGenerator: NoiseGenerator
  private let parameters: TerrainParameters

  func generateHeightMap() -> [Float] {
    var heights: [Float] = []

    for row in 0..<parameters.gridSize {
      for col in 0..<parameters.gridSize {
        let height = generateHeightAt(x: col, y: row)
        heights.append(height)
      }
    }

    return heights
  }
}
```

Рисунок А.13 – Клас TerrainGenerator з методом generateHeightMap для створення карти висот на основі заданих параметрів

```
class HeightMapProcessor {
  // Нормалізація висот до діапазону [0, 1]
  func normalizeHeights(_ heights: [Float]) -> [Float] {
    guard let minHeight = heights.min(),
      let maxHeight = heights.max()
    else {
      return heights
    }

    let range = maxHeight - minHeight
    return heights.map { height in
      (height - minHeight) / range
    }
  }

  // Масштабування висот до заданого діапазону
  func scaleHeights(
    _ heights: [Float],
    minHeight: Float,
    maxHeight: Float
  ) -> [Float] {
    return heights.map { height in
      minHeight + (height * (maxHeight - minHeight))
    }
  }
}
```

Рисунок А.14 – Клас HeightMapProcessor з методами нормалізації та масштабування висот

```swift
class WaterLevelProcessor {
  // Визначення оптимального рівня води
  func calculateWaterLevel(
    _ heights: [Float],
    waterPercentage: Float
  ) -> Float {
    let sortedHeights = heights.sorted()
    let waterIndex = Int(Float(sortedHeights.count) * waterPercentage)
    return sortedHeights[waterIndex]
  }

  // Розділення ландшафту на сушу та воду
  func separateLandAndWater(
    _ heights: [Float],
    waterLevel: Float
  ) -> (land: [Float], water: [Float]) {
    var land: [Float] = []
    var water: [Float] = []

    for height in heights {
      if height > waterLevel {
        land.append(height)
      } else {
        water.append(waterLevel)
      }
    }

    return (land, water)
  }
}
```

Рисунок А.15 – Клас WaterLevelProcessor з методами обчислення рівня води та розділення ландшафту

```swift
class TerrainGenerator {
  private let heightProcessor = HeightMapProcessor()
  private let waterProcessor = WaterLevelProcessor()

  func generateProcessedHeightMap() -> ProcessedHeightMap {
    let rawHeights = generateHeightMap()
    let normalizedHeights = heightProcessor.normalizeHeights(rawHeights)
    let scaledHeights = heightProcessor.scaleHeights(
      normalizedHeights,
      minHeight: parameters.minHeight,
      maxHeight: parameters.maxHeight)
    let waterLevel = waterProcessor.calculateWaterLevel(
      scaledHeights,
      waterPercentage: parameters.waterPercentage)
    let (land, water) = waterProcessor.separateLandAndWater(
      scaledHeights,
      waterLevel: waterLevel)

    return ProcessedHeightMap(
      heights: scaledHeights,
      waterLevel: waterLevel,
      landHeights: land,
      waterHeights: water)
  }
}
```

Рисунок А.16 – Комплексна генерація та обробка карти висот у методі generateProcessedHeightMap класу TerrainGenerator

```swift
struct ProcessedHeightMap {
  let heights: [Float]
  let waterLevel: Float
  let landHeights: [Float]
  let waterHeights: [Float]

  // Додаткові характеристики
  var averageHeight: Float {
    heights.reduce(0, +) / Float(heights.count)
  }

  var maxHeight: Float {
    heights.max() ?? 0
  }

  var minHeight: Float {
    heights.min() ?? 0
  }
}
```

Рисунок А.17 – Структура ProcessedHeightMap для збереження результатів генерації карти висот та супутньої інформації

```swift
struct HeightMapParameters {
  let minHeight: Float
  let maxHeight: Float
  let waterPercentage: Float
  let smoothness: Float

  static let defaultParameters = HeightMapParameters(
    minHeight: -1.0,
    maxHeight: 2.0,
    waterPercentage: 0.3,
    smoothness: 0.5
  )
}
```

Рисунок А.18 – Структура HeightMapParameters з параметрами генерації карти висот

```swift
func createVerticesAndIndices(from heights: [Float]) -> (vertices: [SCNVector3], indices: [UInt32])
{
  var vertices: [SCNVector3] = []
  var indices: [UInt32] = []

  // Створення вершин та індексів
  for row in 0..<gridSize {
    for col in 0..<gridSize {
      let x = Float(col) - Float(gridSize) / 2
      let z = Float(row) - Float(gridSize) / 2
      let y = heights[row * gridSize + col]

      vertices.append(SCNVector3(x, y, z))
      // Додавання індексів для трикутників
    }
  }

  return (vertices, indices)
}
```

Рисунок А.19 – Функція createVerticesAndIndices для генерації вершин і індексів сітки

```swift
func createVerticesAndIndices(from heights: [Float]) -> (vertices: [SCNVector3], indices: [UInt32])
{
  var vertices: [SCNVector3] = []
  var indices: [UInt32] = []

  // Створення вершин та індексів
  for row in 0..<gridSize {
    for col in 0..<gridSize {
      let x = Float(col) - Float(gridSize) / 2
      let z = Float(row) - Float(gridSize) / 2
      let y = heights[row * gridSize + col]

      vertices.append(SCNVector3(x, y, z))
      // Додавання індексів для трикутників
    }
  }

  return (vertices, indices)
}
```

Рисунок А.20 – Функція calculateNormals для розрахунку нормалей до поверхні

```swift
class GeometryOptimizer {
  func optimizeGeometry(_ geometry: SCNGeometry) -> SCNGeometry {
    // Застосування різних методів оптимізації
    let smoothedGeometry = smoothGeometry(geometry)
    let reducedGeometry = reducePolygons(smoothedGeometry)
    let optimizedGeometry = optimizeVertexCache(reducedGeometry)

    return optimizedGeometry
  }
}
```

Рисунок А.21 – Клас GeometryOptimizer для оптимізації тривимірної геометрії

```swift
class TerrainScene: SCNScene {
  private let cameraNode: SCNNode
  private let lightNode: SCNNode
  private let terrainNode: SCNNode

  override init() {
    cameraNode = SCNNode()
    lightNode = SCNNode()
    terrainNode = SCNNode()

    super.init()

    rootNode.addChildNode(cameraNode)
    rootNode.addChildNode(lightNode)
    rootNode.addChildNode(terrainNode)

    setupInitialState()
  }
}
```

Рисунок А.22 – Клас TerrainScene, що формує базову структуру тривимірної сцени

```swift
private func setupInitialState() {
  background.contents = UIColor.systemBlue
  fogStartDistance = 100
  fogEndDistance = 500
  fogColor = UIColor.systemBlue
  physicsWorld.gravity = SCNVector3(0, -9.8, 0)
}
```

Рисунок А.23 – Метод setupInitialState для конфігурації початкових параметрів тривимірної сцени

```swift
private func setupCamera() {
  let camera = SCNCamera()
  cameraNode.camera = camera

  camera.zFar = 1000
  camera.zNear = 0.1
  camera.fieldOfView = 60

  cameraNode.position = SCNVector3(0, 100, 200)
  cameraNode.eulerAngles = SCNVector3(-Float.pi / 6, 0, 0)
}
```

Рисунок А.24 – Метод setupCamera для налаштування камери тривимірної сцени

```swift
private func setupLighting() {
  let sunLight = SCNLight()
  sunLight.type = .directional
  sunLight.intensity = 1000
  sunLight.castsShadow = true

  lightNode.light = sunLight
  lightNode.position = SCNVector3(100, 100, 100)
  lightNode.eulerAngles = SCNVector3(-Float.pi / 4, Float.pi / 4, 0)
}
```

Рисунок А.25 – Метод setupLighting для конфігурації освітлення тривимірної сцени

```swift
private func setupMaterials() {
  let groundMaterial = SCNMaterial()
  groundMaterial.diffuse.contents = UIColor.brown
  groundMaterial.roughness.contents = 0.8
  groundMaterial.metalness.contents = 0.2

  setupNormalMapping(for: groundMaterial)
}
```

Рисунок А.26 – Метод setupMaterials для конфігурації матеріалів тривимірної сцени

```swift
private func setupNormalMapping(for material: SCNMaterial) {
  if let normalMap = UIImage(named: "terrain_normal") {
    material.normal.contents = normalMap
    material.normal.intensity = 1.0
  }

  material.diffuse.wrapS = .repeat
  material.diffuse.wrapT = .repeat
}
```

Рисунок А.27 – Метод setupNormalMapping для налаштування нормал-мапінгу матеріалу

```swift
private func setupMaterialEffects() {
  let material = SCNMaterial()
  material.lightingModel = .physicallyBased
  material.isDoubleSided = true
  material.transparency = 1.0
  material.reflective.contents = UIColor.white
  material.reflective.intensity = 0.5
}
```

Рисунок А.28 – Метод setupMaterialEffects для конфігурації ефектів матеріалу тривимірної сцени

```swift
class TerrainGeometryBuilder {
  func createGeometry(from heightMap: [[Float]], size: CGSize) -> SCNGeometry {
    // Створення вершин на основі карти висот
    let vertices = createVertices(from: heightMap, size: size)

    // Створення нормалей для правильного освітлення
    let normals = calculateNormals(vertices: vertices)

    // Створення текстурних координат
    let texCoords = generateTextureCoordinates(vertices: vertices)

    // Створення індексів для трикутників
    let indices = createIndices(vertices: vertices)

    // Формування геометрії
    return buildGeometry(
      vertices: vertices,
      normals: normals,
      texCoords: texCoords,
      indices: indices)
  }
}
```

Рисунок А.29 – Клас TerrainGeometryBuilder для формування геометрії тривимірного ландшафту

```swift
class TerrainScene {
  func addTerrainGeometry(_ geometry: SCNGeometry) {
    // Створення вузла для геометрії
    let terrainNode = SCNNode(geometry: geometry)

    // Налаштування трансформацій
    terrainNode.position = SCNVector3(0, 0, 0)
    terrainNode.scale = SCNVector3(1, 1, 1)

    // Додавання до сцени
    rootNode.addChildNode(terrainNode)

    // Налаштування фізики
    setupPhysics(for: terrainNode)
  }
}
```

Рисунок А.30 – Метод addTerrainGeometry для інтеграції тривимірної геометрії ландшафту в сцену

```swift
class TerrainMaterialManager {
  func setupMaterials(for geometry: SCNGeometry) {
    // Створення основного матеріалу
    let material = SCNMaterial()

    // Налаштування базових параметрів
    material.diffuse.contents = UIColor.brown
    material.roughness.contents = 0.8
    material.metalness.contents = 0.2

    // Застосування текстур
    applyTextures(to: material)

    // Налаштування додаткових ефектів
    setupMaterialEffects(material)

    // Призначення матеріалу геометрії
    geometry.materials = [material]
  }
}
```

Рисунок А.31 – Клас TerrainMaterialManager для конфігурації матеріалів тривимірної геометрії

```swift
class LandscapeViewController: UIViewController {
  // Основні компоненти UI
  private let sceneView: SCNView
  private let controlPanel: UIView
  private let generationControls: GenerationControlsView
  private let terrainSettings: TerrainSettingsView

  // Менеджер сцени
  private let sceneManager: TerrainSceneManager

  override func viewDidLoad() {
    super.viewDidLoad()
    setupUI()
    configureSceneView()
    setupGestureRecognizers()
  }
}
```

Рисунок А.32 – Клас LandscapeViewController для організації UI та керування тривимірною сценою

```swift
extension LandscapeViewController {
  private func configureSceneView() {
    // Налаштування SCNView
    sceneView.delegate = self
    sceneView.autoenablesDefaultLighting = true
    sceneView.allowsCameraControl = true

    // Налаштування рендерингу
    sceneView.antialiasingMode = .multisampling4X
    sceneView.preferredFramesPerSecond = 60

    // Налаштування відображення
    sceneView.backgroundColor = .systemBackground
    sceneView.overlaySKScene = createOverlayScene()
  }

  private func createOverlayScene() -> SKScene {
    let overlayScene = SKScene(size: view.bounds.size)
    overlayScene.scaleMode = .resizeFill
    return overlayScene
  }
}
```

Рисунок А.33 – Методи configureSceneView та createOverlayScene для налаштування візуалізації тривимірної сцени

```swift
class GenerationControlsView: UIView {
  // Елементи керування генерацією
  private let heightSlider: UISlider
  private let roughnessSlider: UISlider
  private let generateButton: UIButton
  private let resetButton: UIButton

  // Налаштування UI
  private func setupControls() {
    // Налаштування слайдерів
    heightSlider.minimumValue = 0
    heightSlider.maximumValue = 100
    heightSlider.value = 50

    roughnessSlider.minimumValue = 0
    roughnessSlider.maximumValue = 1
    roughnessSlider.value = 0.5

    // Налаштування кнопок
    generateButton.setTitle("Generate", for: .normal)
    resetButton.setTitle("Reset", for: .normal)

    // Додавання обробників подій
    generateButton.addTarget(self, action: #selector(generateTapped), for: .touchUpInside)
    resetButton.addTarget(self, action: #selector(resetTapped), for: .touchUpInside)
  }
}
```

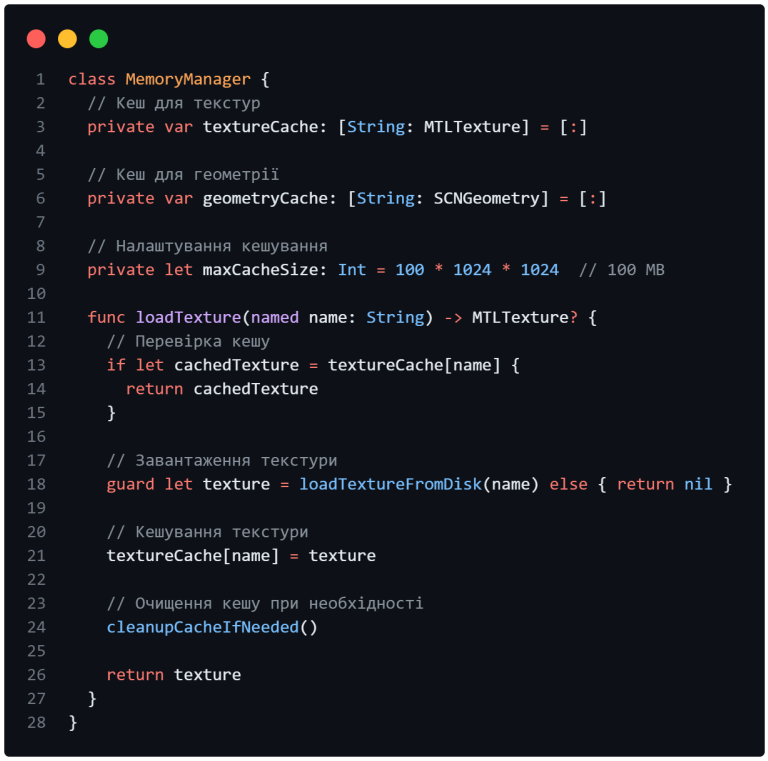
Рисунок А.34 – Клас GenerationControlsView для інтерфейсу керування генерацією тривимірного ландшафту

```swift
extension LandscapeViewController {
  private func setupGestureRecognizers() {
    // Налаштування жестів
    let panGesture = UIPanGestureRecognizer(target: self, action: #selector(handlePan(_:)))
    let pinchGesture = UIPinchGestureRecognizer(target: self, action: #selector(handlePinch(_:)))
    let rotationGesture = UIRotationGestureRecognizer(
      target: self, action: #selector(handleRotation(_:)))

    sceneView.addGestureRecognizer(panGesture)
    sceneView.addGestureRecognizer(pinchGesture)
    sceneView.addGestureRecognizer(rotationGesture)
  }
}
```

Рисунок А.35 – Метод setupGestureRecognizers для обробки жестів у тривимірному середовищі

```swift
class MemoryManager {
  // Кеш для текстур
  private var textureCache: [String: MTLTexture] = [:]

  // Кеш для геометрії
  private var geometryCache: [String: SCNGeometry] = [:]

  // Налаштування кешування
  private let maxCacheSize: Int = 100 * 1024 * 1024  // 100 MB

  func loadTexture(named name: String) -> MTLTexture? {
    // Перевірка кешу
    if let cachedTexture = textureCache[name] {
      return cachedTexture
    }

    // Завантаження текстури
    guard let texture = loadTextureFromDisk(name) else { return nil }

    // Кешування текстури
    textureCache[name] = texture

    // Очищення кешу при необхідності
    cleanupCacheIfNeeded()

    return texture
  }
}
```

Рисунок А.36 – Клас MemoryManager для кешування ресурсів тривимірної сцени



```swift
class MemoryManager {
  // Кеш для текстур
  private var textureCache: [String: MTLTexture] = [:]

  // Кеш для геометрії
  private var geometryCache: [String: SCNGeometry] = [:]

  // Налаштування кешування
  private let maxCacheSize: Int = 100 * 1024 * 1024  // 100 MB

  func loadTexture(named name: String) -> MTLTexture? {
    // Перевірка кешу
    if let cachedTexture = textureCache[name] {
      return cachedTexture
    }

    // Завантаження текстури
    guard let texture = loadTextureFromDisk(name) else { return nil }

    // Кешування текстури
    textureCache[name] = texture

    // Очищення кешу при необхідності
    cleanupCacheIfNeeded()

    return texture
  }
}
```

Рисунок А.37 – Клас TerrainOptimizer для оптимізації тривимірної геометрії за допомогою рівнів деталізації

```swift
class SceneUpdateManager {
  // Черга оновлень
  private var updateQueue: DispatchQueue

  // Прапорець оновлення
  private var isUpdating: Bool = false

  func updateScene(with parameters: GenerationParameters) {
    // Додавання завдання в чергу
    updateQueue.async { [weak self] in
      self?.performUpdate(with: parameters)
    }
  }

  private func performUpdate(with parameters: GenerationParameters) {
    // Перевірка стану оновлення
    guard !isUpdating else { return }

    isUpdating = true

    // Генерація нової геометрії
    let newGeometry = generateTerrainGeometry(with: parameters)

    // Оновлення сцени
    DispatchQueue.main.async {
      self.updateSceneGeometry(newGeometry)
      self.isUpdating = false
    }
  }
}
```

Рисунок А.38 – Клас SceneUpdateManager для асинхронного оновлення тривимірної сцени

```swift
extension TerrainScene {
  func configureRendering() {
    // Налаштування антиаліасингу
    sceneView.antialiasingMode = .multisampling4X

    // Налаштування тіней
    sceneView.shadowMode = .deferred

    // Налаштування рендерингу
    sceneView.preferredFramesPerSecond = 60
    sceneView.rendersContinuously = true
  }
}
```

Рисунок А.39 – Метод configureRendering для конфігурації параметрів рендерингу тривимірної сцени