// VoxelCraft iOS – a Minecraft‑like starter you can build & run in Xcode

// Single‑file SwiftUI + SceneKit app. Paste into a new Xcode iOS App project

// (SwiftUI lifecycle), replace ContentView.swift with this file's contents.

// Tested for iOS 16+ (SceneKit).

import SwiftUI

import SceneKit

@main

struct VoxelCraftApp: App {

var body: some Scene {

WindowGroup {

ContentView()

.ignoresSafeArea()

}

}

}

struct ContentView: View {

@StateObject private var world = VoxelWorld()

var body: some View {

ZStack(alignment: .topLeading) {

SceneViewContainer(world: world)

.ignoresSafeArea()

HUD(world: world)

.padding(12)

}

}

}

// MARK: - HUD

struct HUD: View {

@ObservedObject var world: VoxelWorld

@State private var selectedType: BlockType = .grass

var body: some View {

VStack(alignment: .leading, spacing: 8) {

Text("VoxelCraft")

.font(.title2).fontWeight(.bold)

.padding(.horizontal, 8).padding(.vertical, 4)

.background(.ultraThinMaterial)

.cornerRadius(10)

HStack(spacing: 8) {

ForEach(BlockType.allCases, id: \.self) { type in

Button(action: { selectedType = type; world.currentType = type }) {

RoundedRectangle(cornerRadius: 10)

.fill(type.color)

.frame(width: 36, height: 36)

.overlay(RoundedRectangle(cornerRadius: 10).stroke(Color.white.opacity(selectedType == type ? 1 : 0.2), lineWidth: 2))

}

}

}

.padding(6)

.background(.ultraThinMaterial)

.cornerRadius(14)

Text("Tap = remove, Long‑press = add.\nOne finger drag = orbit, Two fingers = pan, Pinch = zoom.")

.font(.footnote)

.padding(8)

.background(.ultraThinMaterial)

.cornerRadius(10)

}

}

}

// MARK: - SceneKit container

struct SceneViewContainer: UIViewRepresentable {

@ObservedObject var world: VoxelWorld

func makeUIView(context: Context) -> SCNView {

let view = SCNView(frame: .zero)

view.scene = world.scene

view.allowsCameraControl = true // orbit/pan/pinch

view.defaultCameraController.interactionMode = .orbitTurntable

view.showsStatistics = false

view.antialiasingMode = .multisampling4X

view.backgroundColor = UIColor.systemTeal

// Gestures

let tap = UITapGestureRecognizer(target: context.coordinator, action: #selector(Coordinator.handleTap(\_:)))

tap.numberOfTouchesRequired = 1

view.addGestureRecognizer(tap)

let long = UILongPressGestureRecognizer(target: context.coordinator, action: #selector(Coordinator.handleLongPress(\_:)))

long.minimumPressDuration = 0.25

view.addGestureRecognizer(long)

return view

}

func updateUIView(\_ uiView: SCNView, context: Context) {}

func makeCoordinator() -> Coordinator { Coordinator(world: world) }

final class Coordinator: NSObject {

let world: VoxelWorld

init(world: VoxelWorld) { self.world = world }

@objc func handleTap(\_ gr: UITapGestureRecognizer) {

guard let view = gr.view as? SCNView else { return }

let p = gr.location(in: view)

let results = view.hitTest(p, options: [.rootNode: world.rootNode, .firstFoundOnly: true])

if let r = results.first, let key = r.node.name, key.hasPrefix("block\_") {

world.removeBlock(named: key)

}

}

@objc func handleLongPress(\_ gr: UILongPressGestureRecognizer) {

guard gr.state == .began, let view = gr.view as? SCNView else { return }

let p = gr.location(in: view)

let results = view.hitTest(p, options: [.rootNode: world.rootNode, .firstFoundOnly: true, .boundingBoxOnly: true])

if let r = results.first {

// Place a block adjacent to the face we touched

let normal = r.localNormal

let node = r.node

// Convert hit to world grid position

let worldPos = node.convertPosition(r.localCoordinates, to: world.rootNode)

var gx = Int(round(worldPos.x))

var gy = Int(round(worldPos.y))

var gz = Int(round(worldPos.z))

// Nudge one unit along face normal

if abs(normal.x) > 0.5 { gx += Int(normal.x.sign == .plus ? 1 : -1) }

if abs(normal.y) > 0.5 { gy += Int(normal.y.sign == .plus ? 1 : -1) }

if abs(normal.z) > 0.5 { gz += Int(normal.z.sign == .plus ? 1 : -1) }

world.placeBlock(at: Int3(gx, gy, gz))

} else {

// If we didn't hit a block, try to place at ground y=0 under cursor via unproject

if let ray = view.ray(from: p) { world.placeOnGround(ray: ray) }

}

}

}

}

// MARK: - World model

final class VoxelWorld: ObservableObject {

let scene = SCNScene()

let rootNode = SCNNode()

@Published var currentType: BlockType = .grass

private var blocks: [Int3: BlockType] = [:]

private var nodes: [Int3: SCNNode] = [:]

// World bounds for starter (can expand/chunk later)

let sizeX = 24, sizeZ = 24, baseY = 0

init() {

scene.rootNode.addChildNode(rootNode)

setupCameraAndLight()

generateFlatWorld()

addGrid()

}

private func setupCameraAndLight() {

let cameraNode = SCNNode()

cameraNode.camera = SCNCamera()

cameraNode.position = SCNVector3(0, 20, 36)

cameraNode.camera?.zFar = 500

scene.rootNode.addChildNode(cameraNode)

let light = SCNNode()

light.light = SCNLight()

light.light?.type = .directional

light.eulerAngles = SCNVector3(-.pi/3, .pi/4, 0)

scene.rootNode.addChildNode(light)

let amb = SCNNode()

amb.light = SCNLight()

amb.light?.type = .ambient

amb.light?.intensity = 200

scene.rootNode.addChildNode(amb)

}

private func generateFlatWorld() {

for x in -sizeX/2..<(sizeX/2) {

for z in -sizeZ/2..<(sizeZ/2) {

let pos = Int3(x, baseY, z)

addBlock(type: .grass, at: pos)

}

}

}

private func addGrid() {

let grid = SCNFloor()

grid.reflectivity = 0

grid.firstMaterial?.diffuse.contents = UIColor.clear

let node = SCNNode(geometry: grid)

node.position = SCNVector3(0, Float(baseY) - 0.51, 0)

node.geometry?.firstMaterial?.isDoubleSided = true

scene.rootNode.addChildNode(node)

}

func placeBlock(at gpos: Int3) {

guard blocks[gpos] == nil else { return }

addBlock(type: currentType, at: gpos)

}

func removeBlock(named name: String) {

guard let g = Int3(name.replacingOccurrences(of: "block\_", with: "")) else { return }

removeBlock(at: g)

}

func removeBlock(at gpos: Int3) {

blocks[gpos] = nil

if let node = nodes[gpos] { node.removeFromParentNode() }

nodes[gpos] = nil

}

func placeOnGround(ray: Ray) {

// Intersect ray with y = baseY plane: P = O + tD, solve for y

let denom = ray.dir.y

guard abs(denom) > 1e-5 else { return }

let t = (Float(baseY) - ray.origin.y) / denom

guard t > 0 else { return }

let p = ray.origin + ray.dir \* t

let g = Int3(Int(round(p.x)), baseY + 1, Int(round(p.z)))

placeBlock(at: g)

}

private func addBlock(type: BlockType, at gpos: Int3) {

blocks[gpos] = type

let node = makeBlockNode(type: type)

node.position = SCNVector3(Float(gpos.x), Float(gpos.y), Float(gpos.z))

node.name = "block\_\(gpos.stringKey)"

rootNode.addChildNode(node)

nodes[gpos] = node

}

private func makeBlockNode(type: BlockType) -> SCNNode {

let box = SCNBox(width: 1, height: 1, length: 1, chamferRadius: 0)

let mat = SCNMaterial()

mat.diffuse.contents = UIColor(type.color)

mat.locksAmbientWithDiffuse = true

box.materials = [mat]

return SCNNode(geometry: box)

}

}

// MARK: - Types & helpers

enum BlockType: CaseIterable { case grass, dirt, stone, wood, leaves }

extension BlockType {

var color: Color {

switch self {

case .grass: return Color(red: 0.42, green: 0.74, blue: 0.36)

case .dirt: return Color(red: 0.47, green: 0.31, blue: 0.20)

case .stone: return Color(red: 0.55, green: 0.56, blue: 0.58)

case .wood: return Color(red: 0.62, green: 0.45, blue: 0.28)

case .leaves: return Color(red: 0.33, green: 0.66, blue: 0.36)

}

}

}

// Integer 3D vector keyed by grid coordinates

struct Int3: Hashable { var x: Int; var y: Int; var z: Int

init(\_ x: Int, \_ y: Int, \_ z: Int) { self.x = x; self.y = y; self.z = z }

init?(\_ key: String) {

let parts = key.split(separator: ",").compactMap { Int($0) }

guard parts.count == 3 else { return nil }

self.x = parts[0]; self.y = parts[1]; self.z = parts[2]

}

var stringKey: String { "\(x),\(y),\(z)" }

}

// Simple ray type for ground placement

struct Ray { var origin: SCNVector3; var dir: SCNVector3 }

extension SCNView {

func ray(from screenPoint: CGPoint) -> Ray? {

guard let pNear = unprojectPoint(SCNVector3(Float(screenPoint.x), Float(screenPoint.y), 0)),

let pFar = unprojectPoint(SCNVector3(Float(screenPoint.x), Float(screenPoint.y), 1)) else { return nil }

let dir = (pFar - pNear).normalized

return Ray(origin: pNear, dir: dir)

}

func unprojectPoint(\_ point: SCNVector3) -> SCNVector3? {

// SceneKit has unprojectPoint(\_:), but SCNView's version is not optional in Swift.

// We wrap to keep code symmetrical.

return unprojectPoint(point)

}

}

// MARK: - SCNVector3 helpers

func +(lhs: SCNVector3, rhs: SCNVector3) -> SCNVector3 { SCNVector3(lhs.x+rhs.x, lhs.y+rhs.y, lhs.z+rhs.z) }

func -(lhs: SCNVector3, rhs: SCNVector3) -> SCNVector3 { SCNVector3(lhs.x-rhs.x, lhs.y-rhs.y, lhs.z-rhs.z) }

func \*(lhs: SCNVector3, rhs: Float) -> SCNVector3 { SCNVector3(lhs.x\*rhs, lhs.y\*rhs, lhs.z\*rhs) }

extension SCNVector3 {

var length: Float { sqrt(x\*x + y\*y + z\*z) }

var normalized: SCNVector3 {

let l = max(1e-6, length)

return SCNVector3(x/l, y/l, z/l)

}

}

extension Float { var sign: FloatingPointSign { self >= 0 ? .plus : .minus } }