#WWDC18

Inside SwiftShot

Creating an augmented reality game

Session 605

Alex Rosenberg, Tools Foundation

Designing Games for AR

SwiftShot Internals

World Map Sharing

Networking

Physics

Wrap-up

Designing Games for AR

Gameplay must come first

Designing Games for Augmented Reality

Designing Games for Augmented Reality

Start with the gameplay

Designing Games for Augmented Reality

Start with the gameplay

Short sessions with easy engagement

Designing Games for Augmented Reality

Start with the gameplay

Short sessions with easy engagement

Variety of content to keep it fresh

Designing Games for Augmented Reality

Start with the gameplay

Short sessions with easy engagement

Variety of content to keep it fresh

Spectating turned out to be interesting

Designing Games for Augmented Reality

Start with the gameplay

Short sessions with easy engagement

Variety of content to keep it fresh

Spectating turned out to be interesting

Social interaction and personal connection

Designing Games for Augmented Reality

Start with the gameplay

Short sessions with easy engagement

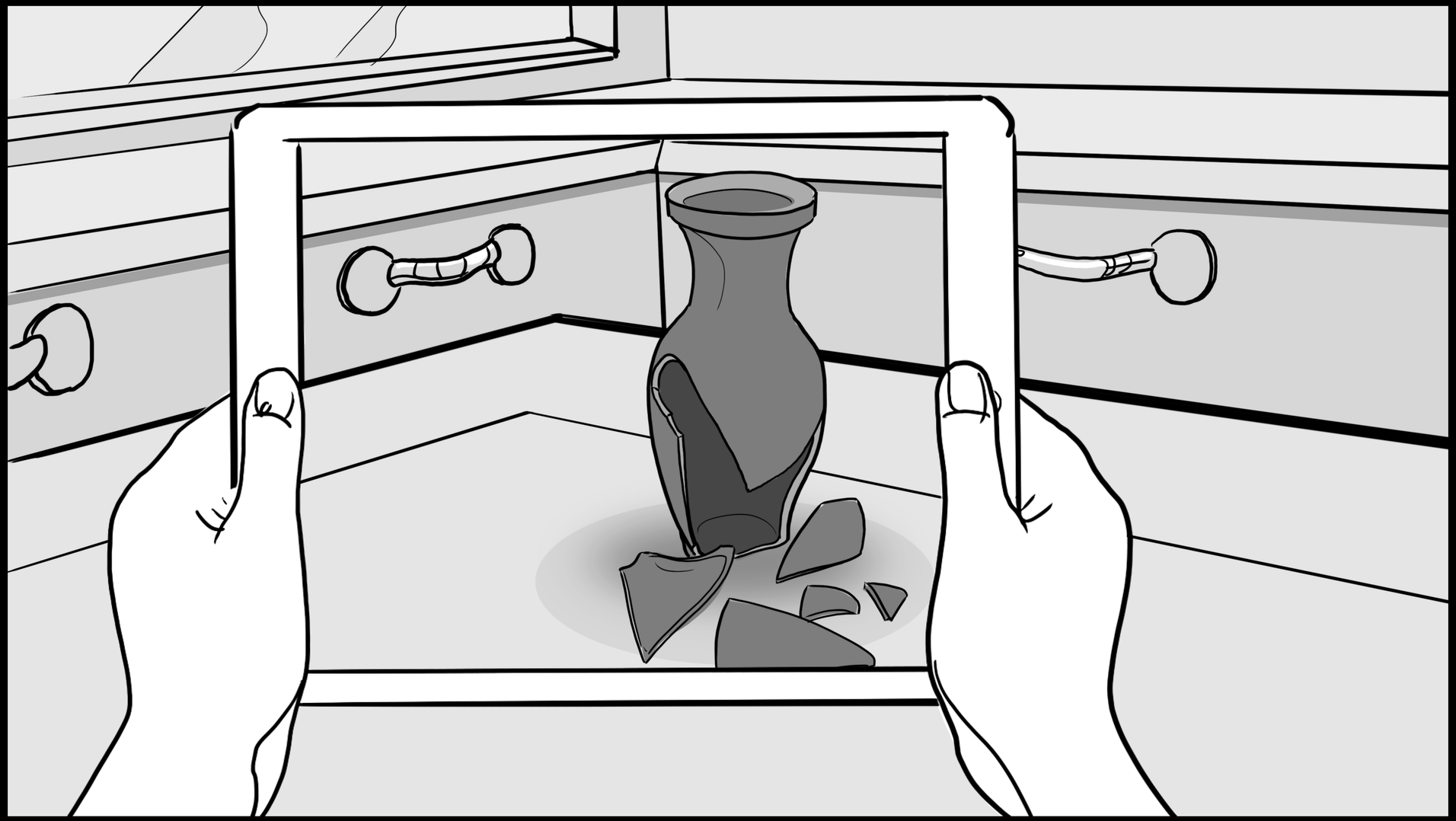
Variety of content to keep it fresh

Spectating turned out to be interesting

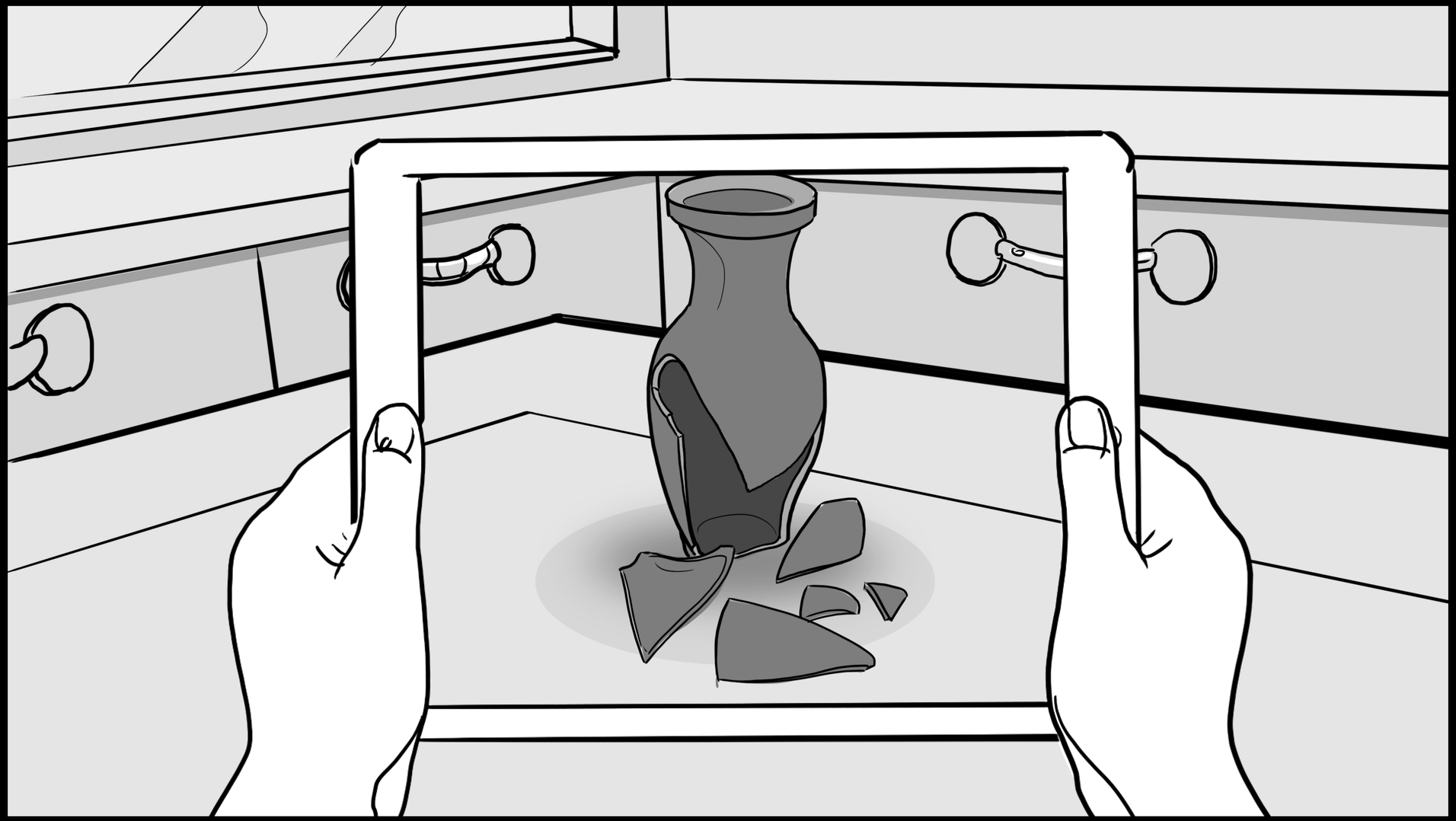
Social interaction and personal connection

Consider the benefits of AR

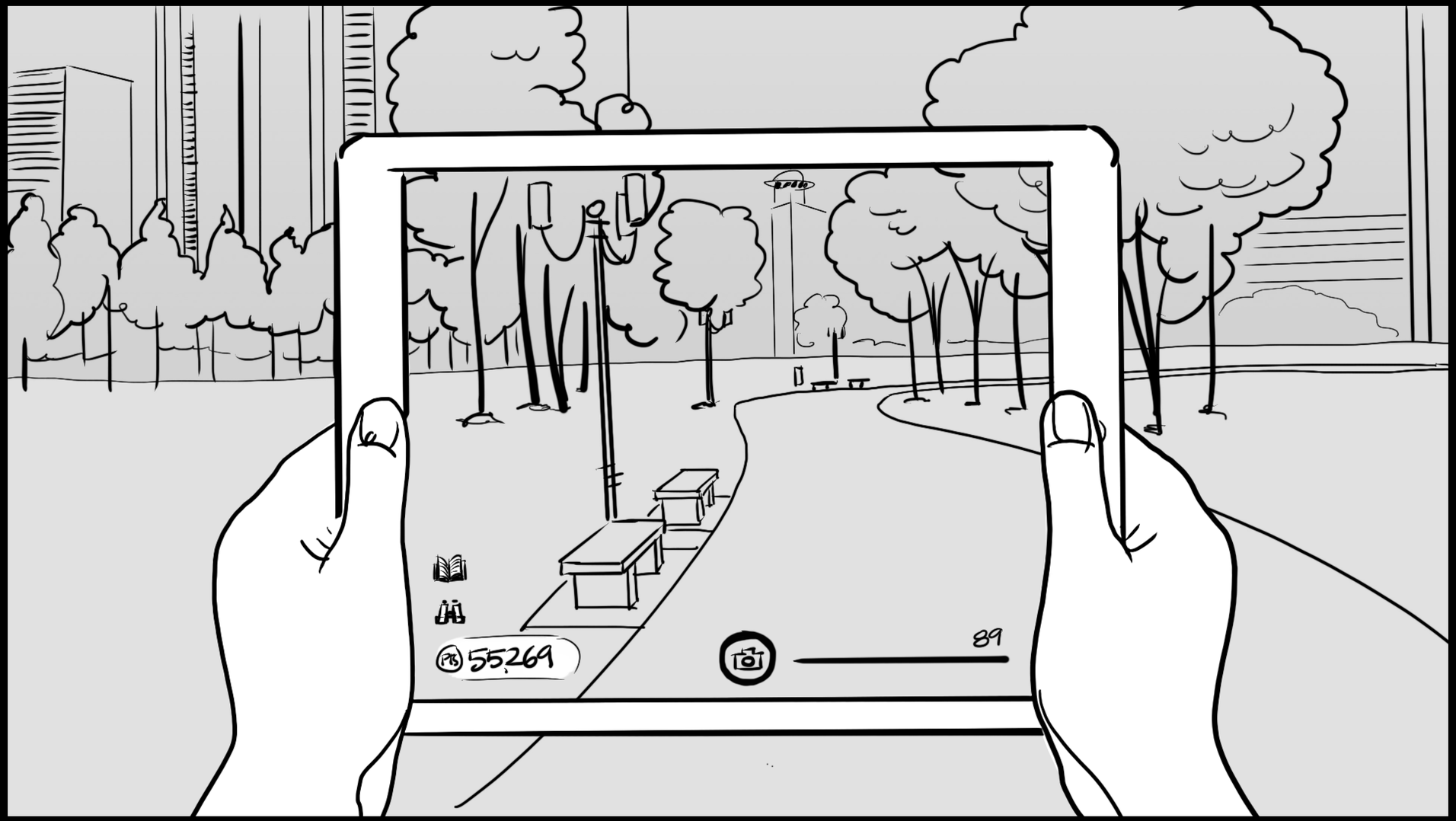
Designing Games for Augmented Reality



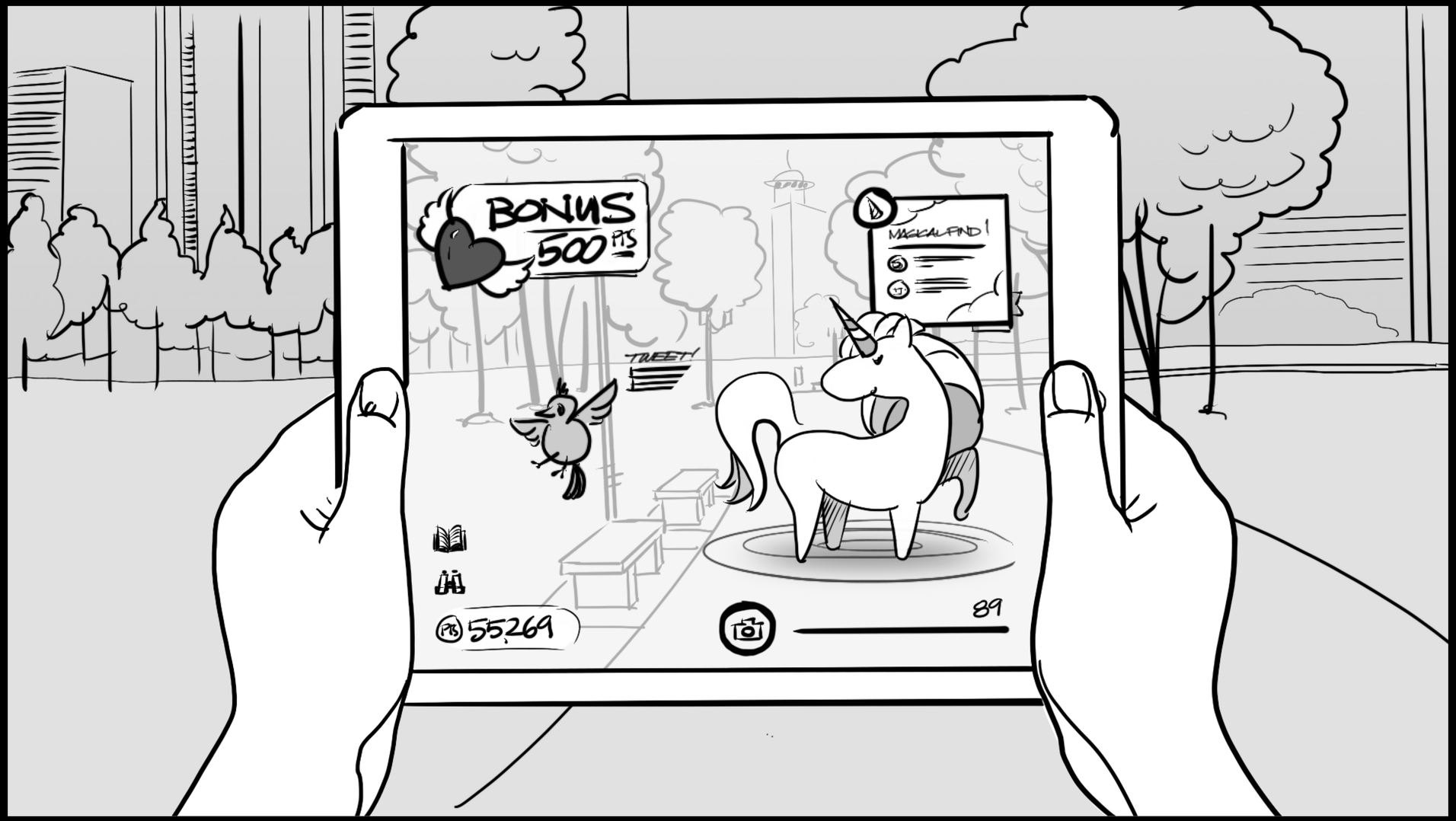
Designing Games for Augmented Reality



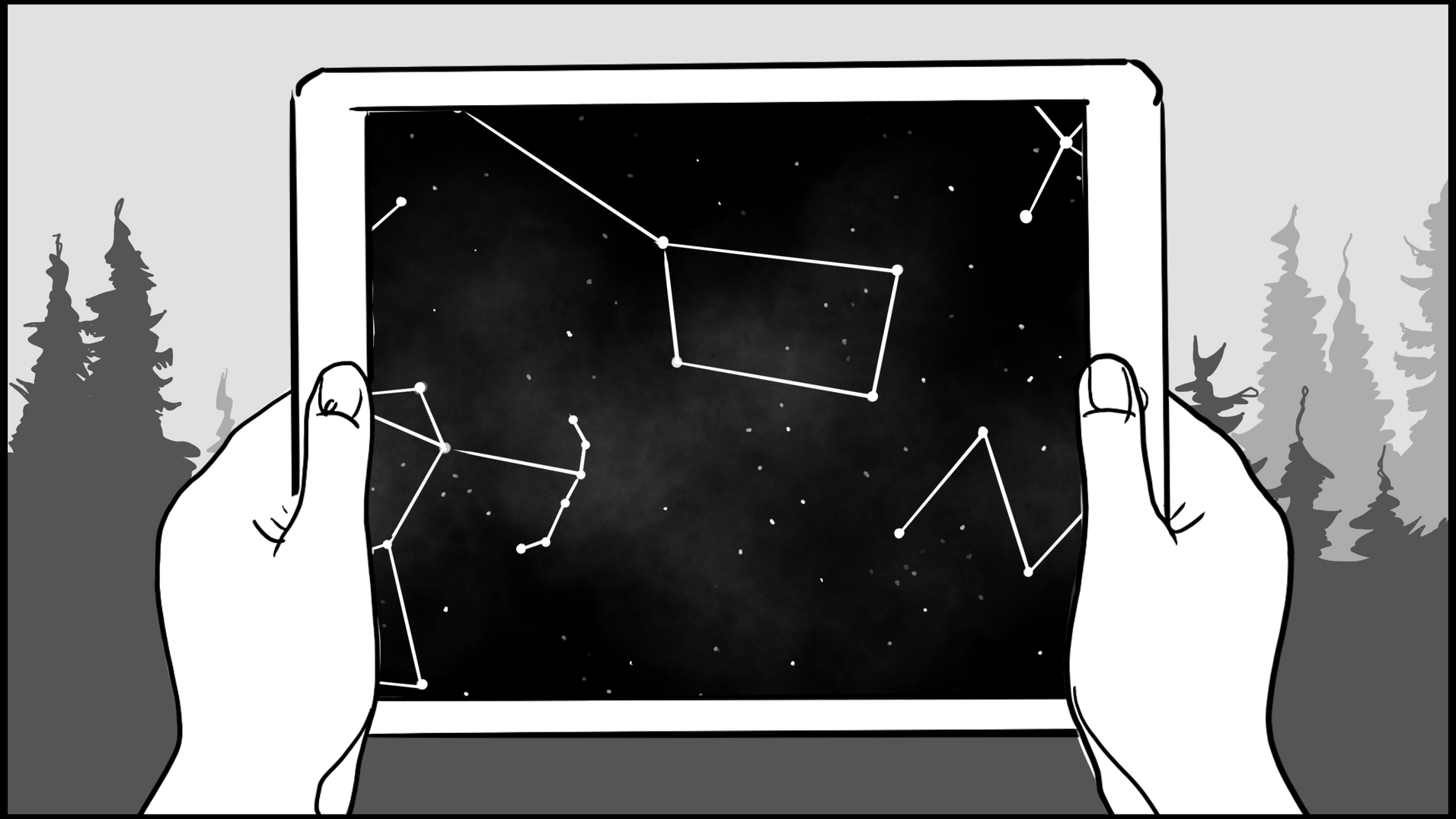
Designing Games for Augmented Reality



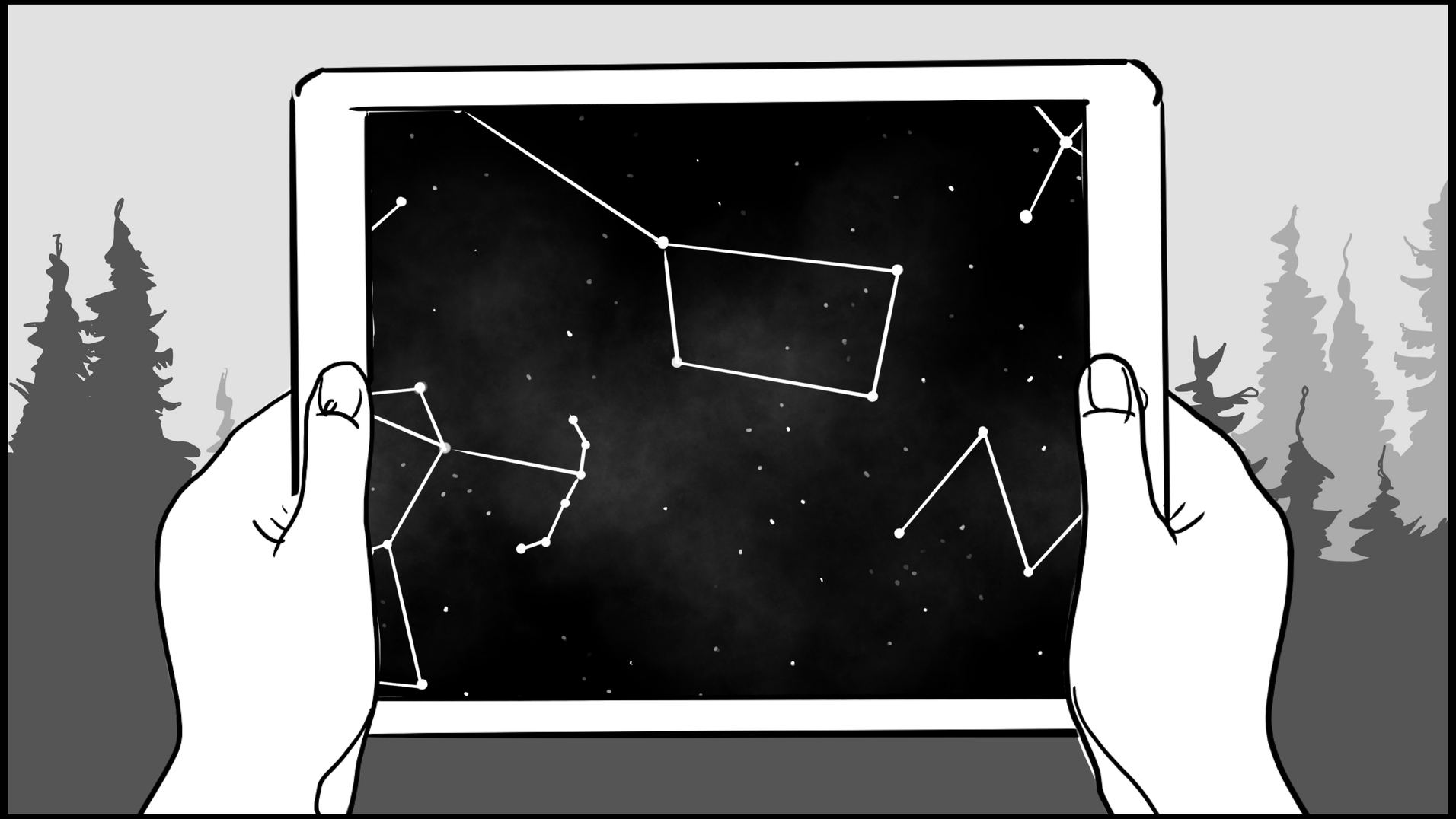
Designing Games for Augmented Reality



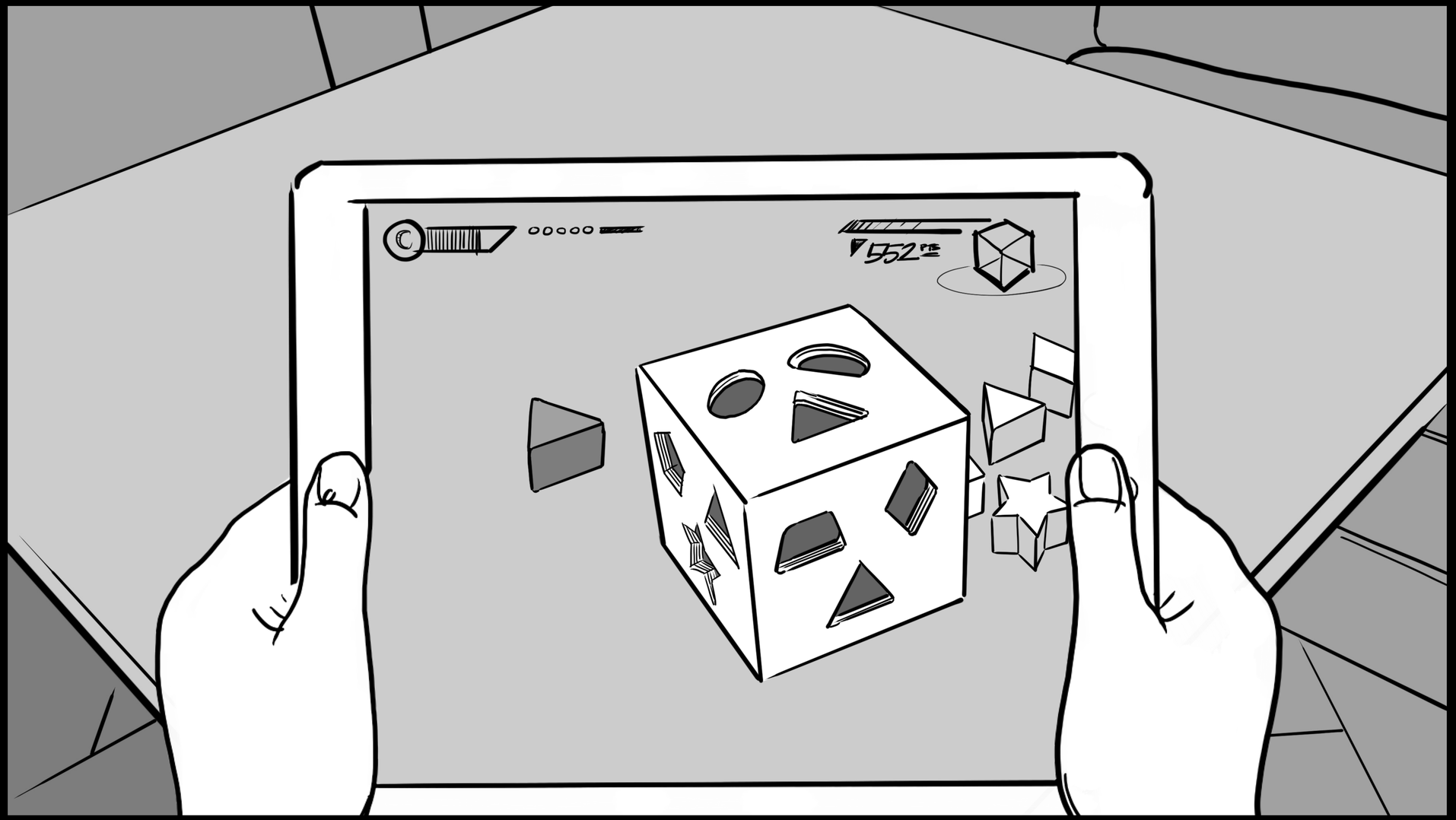
Designing Games for Augmented Reality



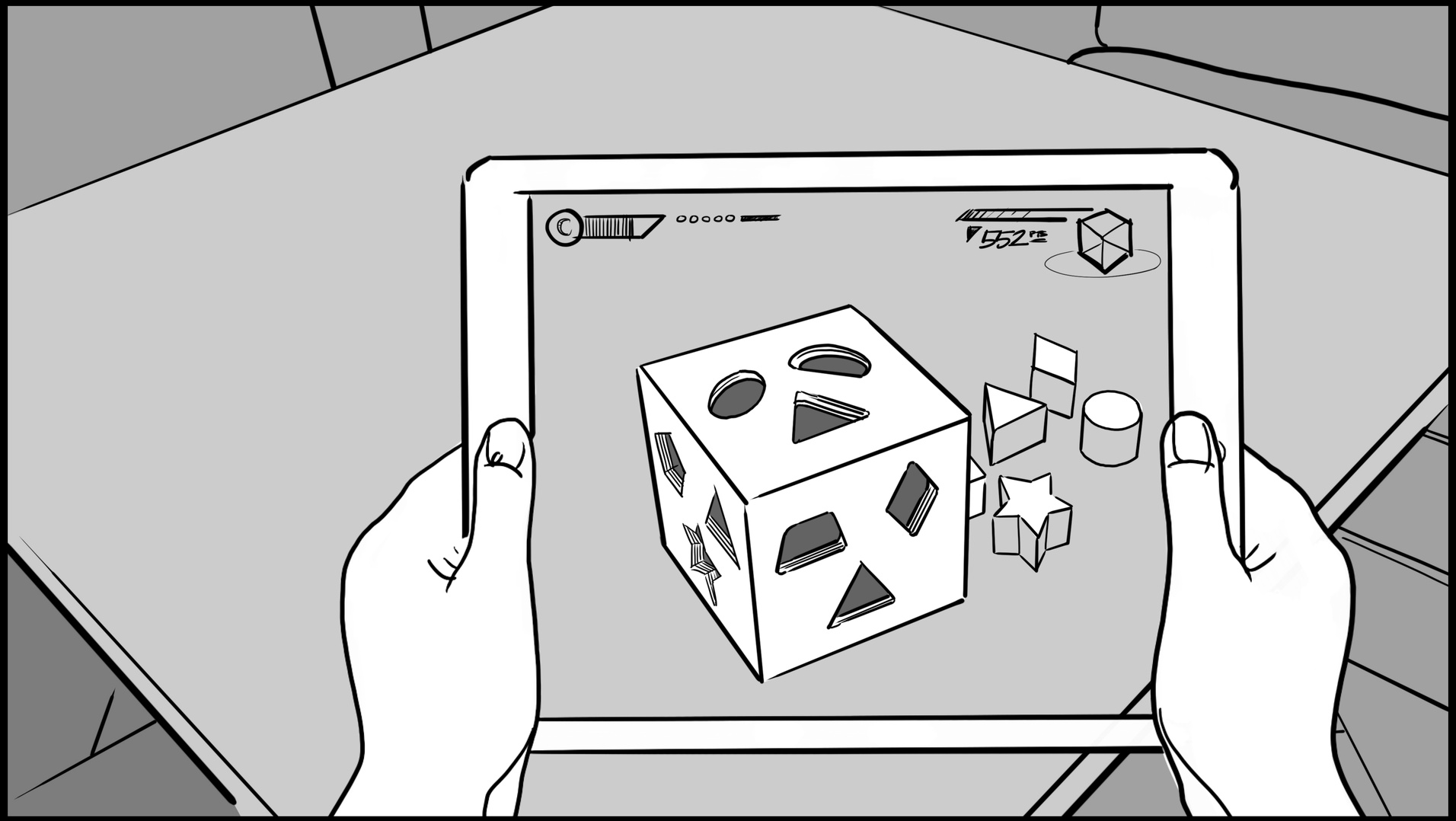
Designing Games for Augmented Reality



Designing Games for Augmented Reality



Designing Games for Augmented Reality



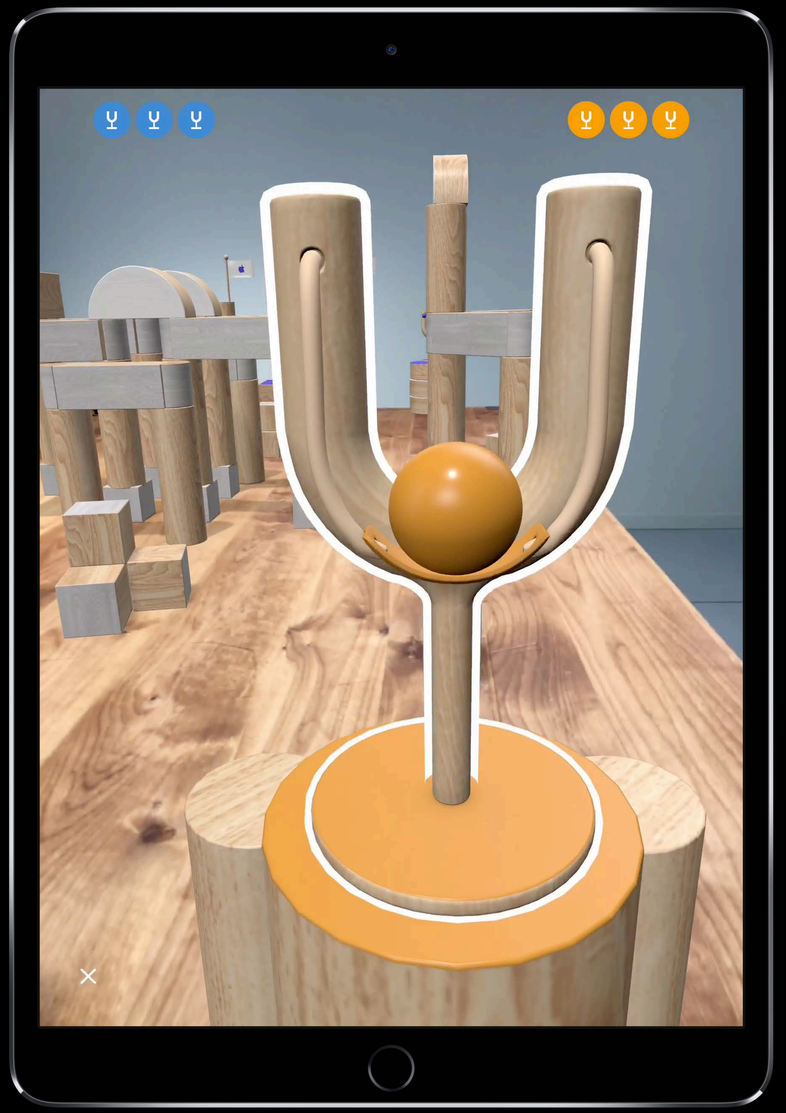
Control Mechanisms

Encourage slow movement of the device

Control Mechanisms

Encourage slow movement of the device

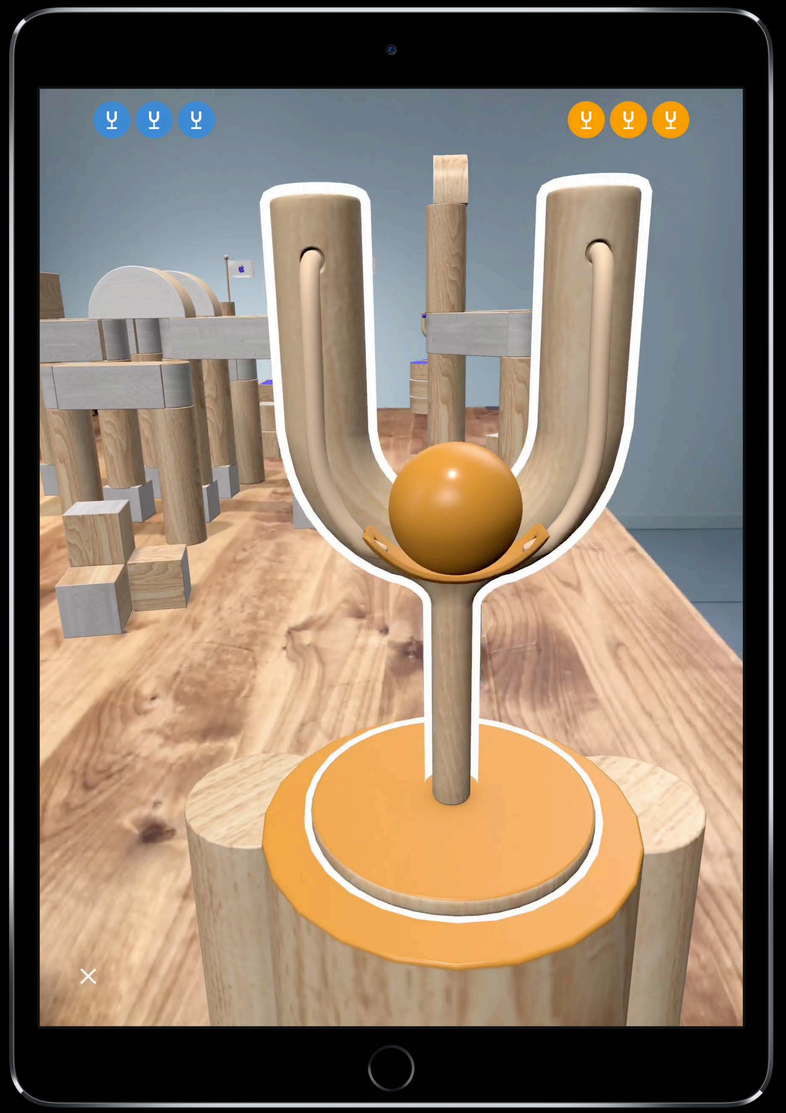
Encourage the player to move



Control Mechanisms

Encourage slow movement of the device

Encourage the player to move



Control Mechanisms

Encourage slow movement of the device

Encourage the player to move

Control feedback is important for immersion

SwiftShot Internals

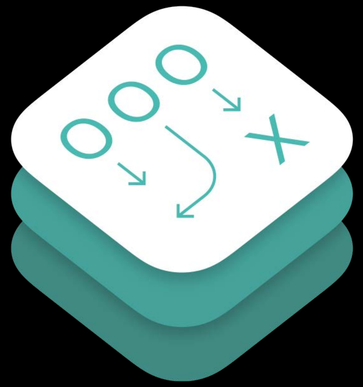
David Paschich, Tools Foundation


AR

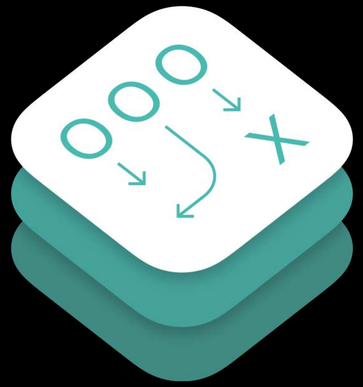

AR


AR


AR

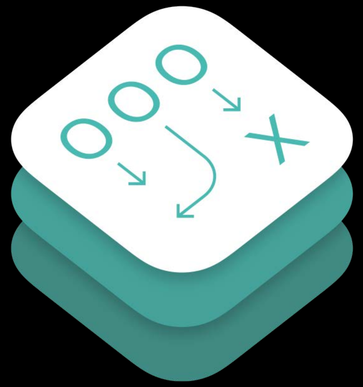



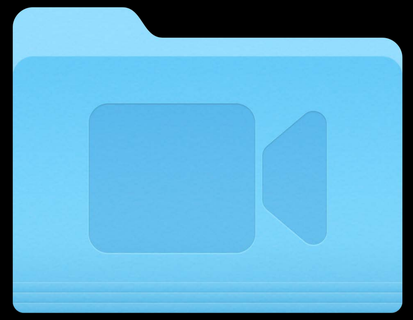

AR

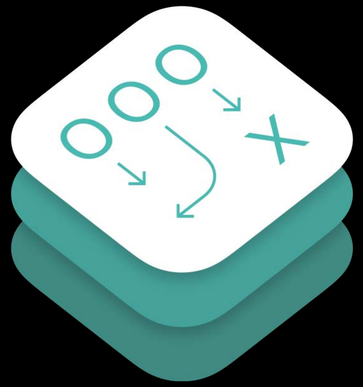


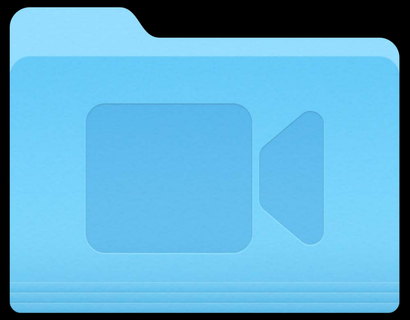
AR




AR




SwiftShot Internals

Establishing a shared coordinate space

Networking and state sharing

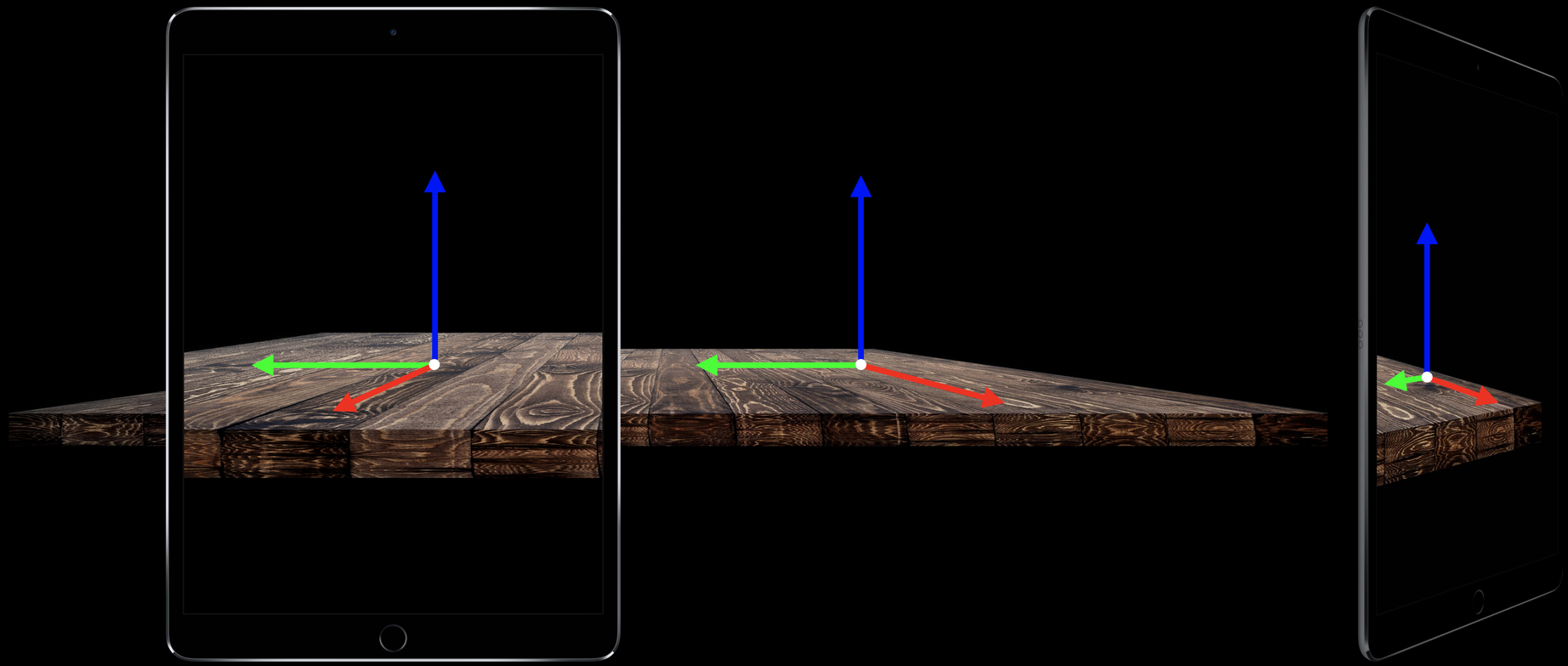
Physics

Asset import and management

Flag simulation

Dynamic audio

ARKit—Establishing a Shared Coordinate Space



Establishing a Shared Coordinate Space

Establishing a Shared Coordinate Space

Image detection

Establishing a Shared Coordinate Space

Image detection

Object detection

Establishing a Shared Coordinate Space

Image detection

Object detection

World Map sharing

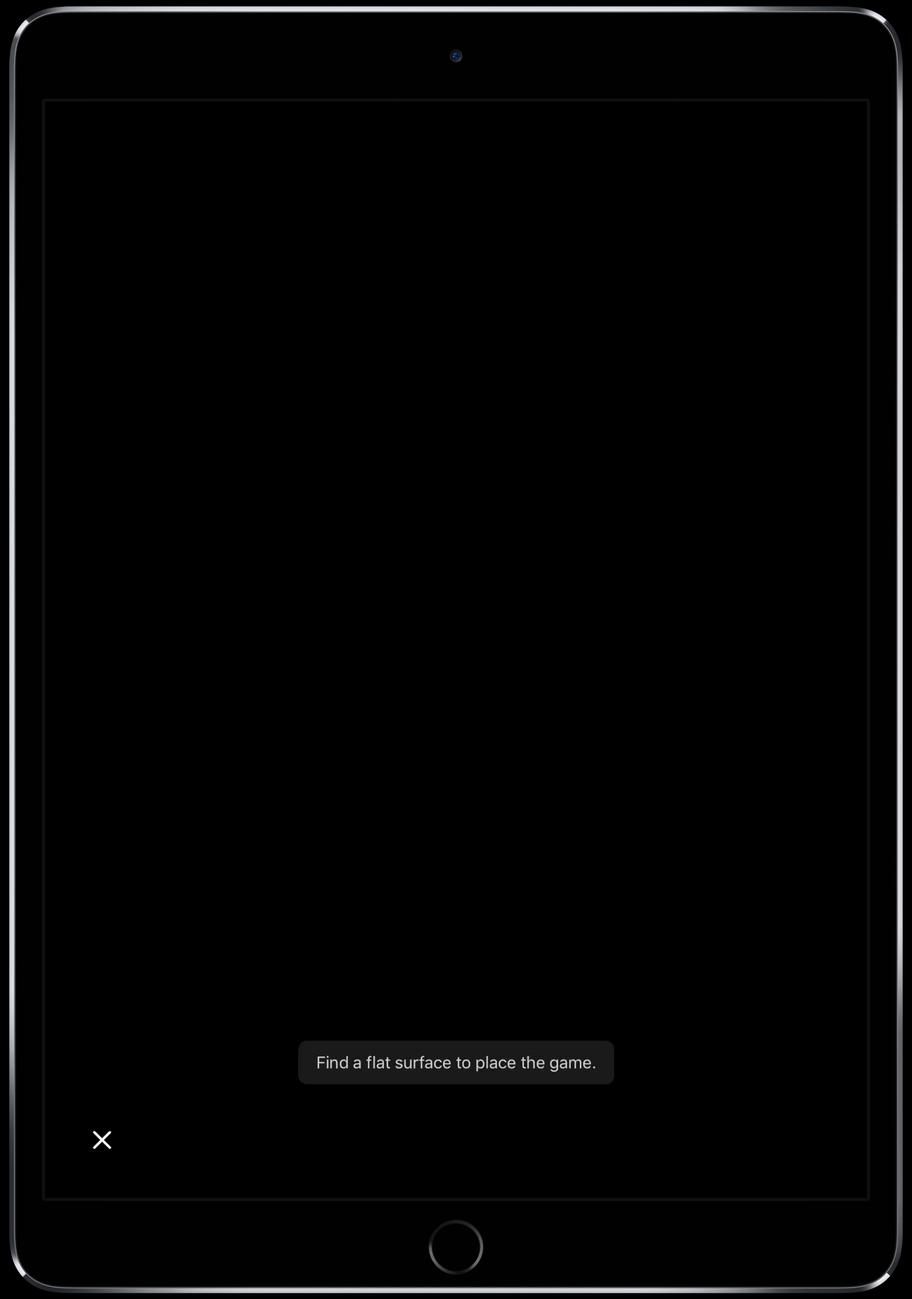
Establishing a Shared Coordinate Space

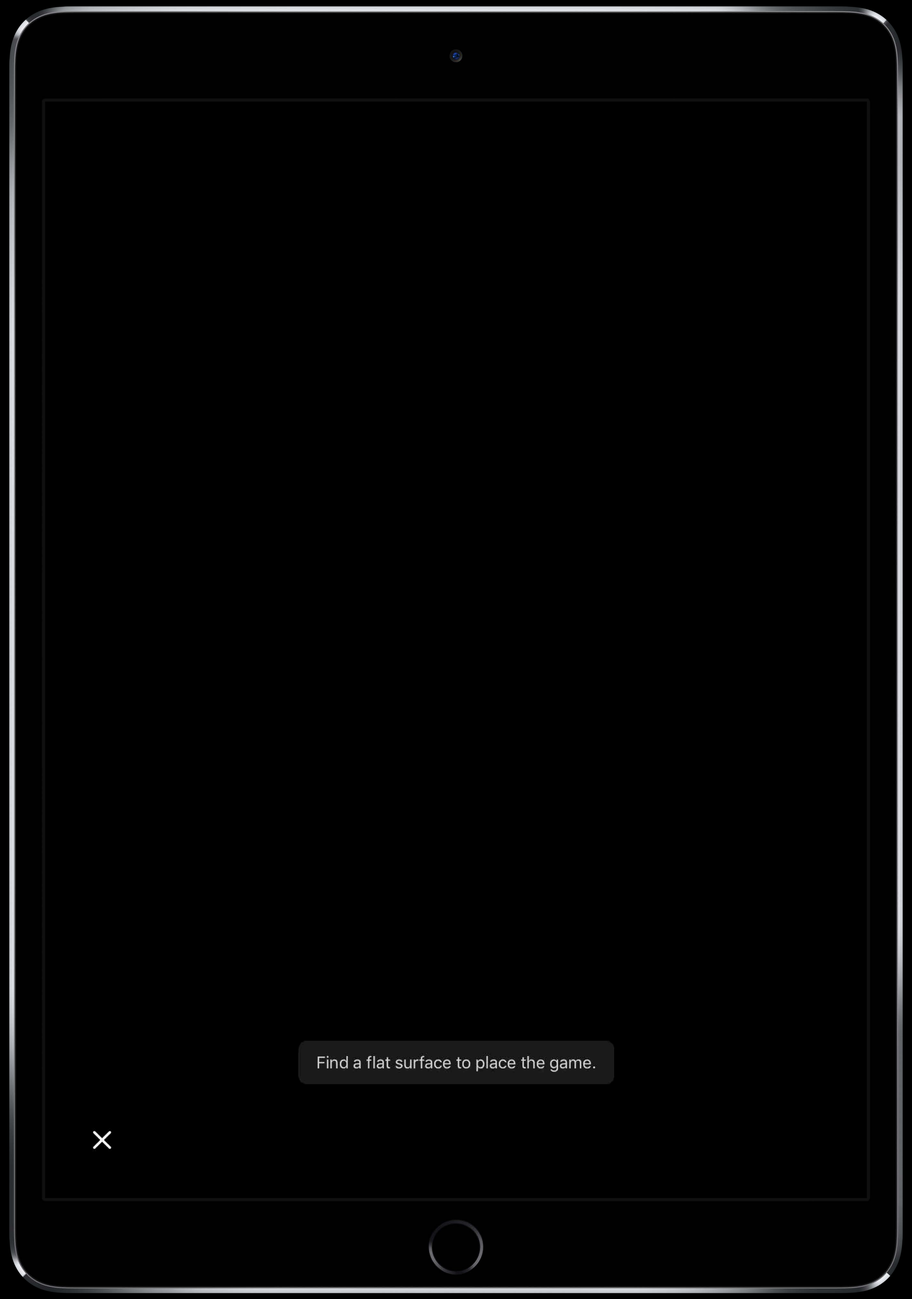
Image detection

Object detection

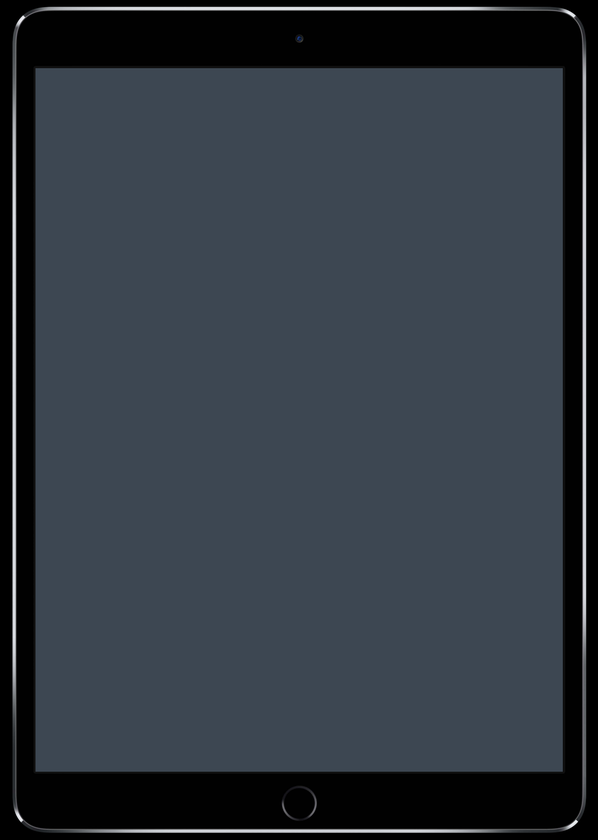
World Map sharing

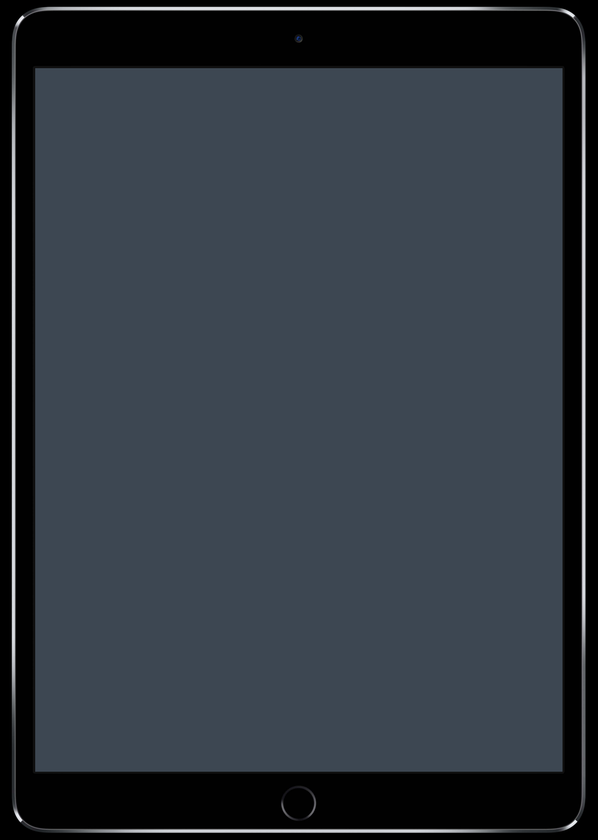
iBeacons for fixed installations


Find a flat surface to place the game.

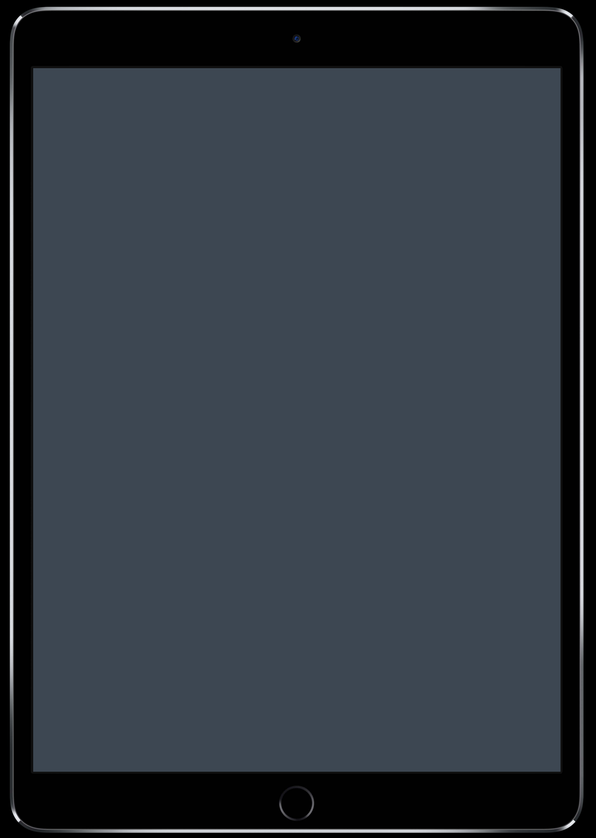

Find a flat surface to place the game.

Sharing a World Map

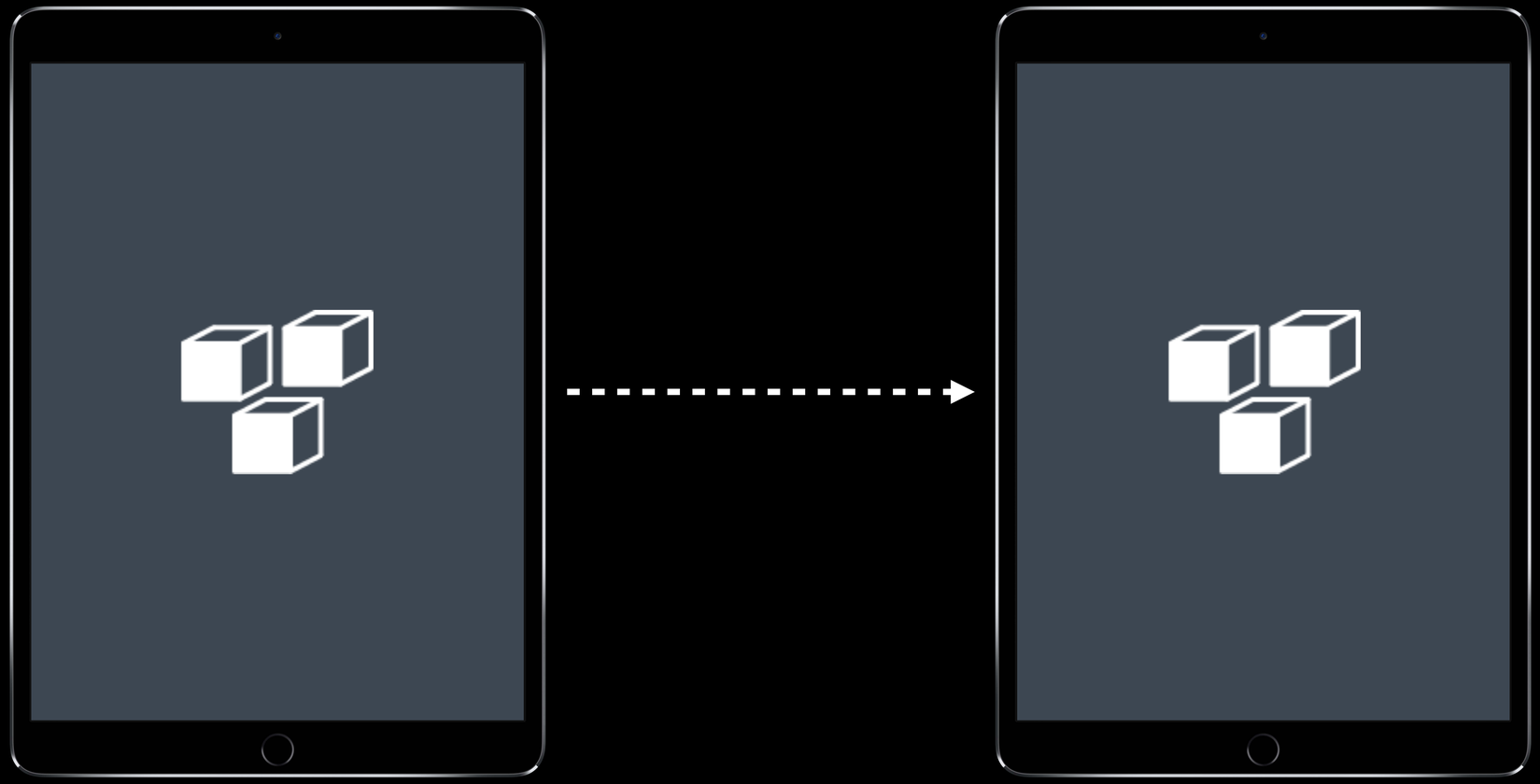




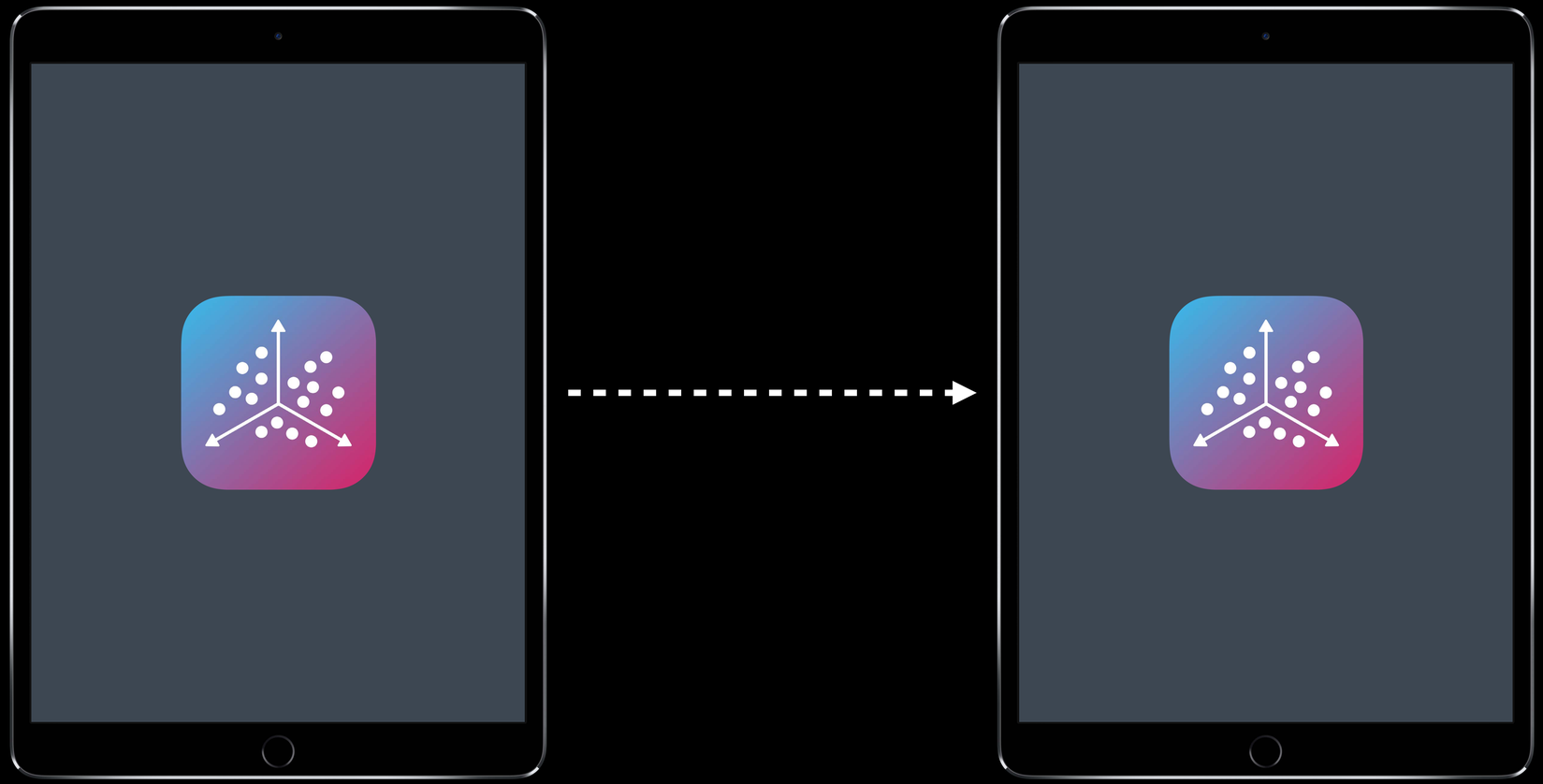
Sharing a World Map



Sharing a World Map



Sharing a World Map



Sharing a World Map

Saving

NEW

First device scans area, captures features

Asks ARSession for world map

Serializes to disk

```
sceneView.session.getCurrentWorldMap { map, error in
    if let error = error { print(error); return }
    guard let map = map, let data = try? NSKeyedArchiver.archivedData (
            withRootObject: map, requiringSecureCoding: true)
        else { return }
        // save or send over network
    }
}
```

Sharing a World Map

Saving

NEW

First device scans area, captures features

Asks ARSession for world map

Serializes to disk

```
sceneView.session.getCurrentWorldMap { map, error in
    if let error = error { print(error); return }
    guard let map = map, let data = try? NSKeyedArchiver.archivedData (
            withRootObject: map, requiringSecureCoding: true)
        else { return }
        // save or send over network
    }
}
```

Sharing a World Map

Saving

NEW

First device scans area, captures features

Asks ARSession for world map

Serializes to disk

```
sceneView.session.getCurrentWorldMap { map, error in
    if let error = error { print(error); return }
    guard let map = map, let data = try? NSKeyedArchiver.archivedData (
            withRootObject: map, requiringSecureCoding: true)
        else { return }
        // save or send over network
    }
}
```

Sharing a World Map

Saving

NEW

First device scans area, captures features

Asks ARSession for world map

Serializes to disk

```
sceneView.session.getCurrentWorldMap { map, error in
    if let error = error { print(error); return }
    guard let map = map, let data = try? NSKeyedArchiver.archivedData (
            withRootObject: map, requiringSecureCoding: true)
        else { return }
        // save or send over network
    }
}
```

Sharing a World Map

Ad-hoc gaming

Share over network

- Peer-to-peer network connection
- Encrypt data in flight
- UI guidance to aid relocalization

Sharing a World Map

Fixed installations

Prerecord world maps for specific installation areas

Distribute to managed devices

Use iBeacons to automatically select correct world map for each location

Sharing a World Map

Loading

NEW

```
// Unarchive data to ARWorldMap
let worldMap = try NSKeyedUnarchiver.unarchivedObject(ofClass: ARWorldMap.self, from: data)

// Create tracking configuration
let configuration = ARWorldTrackingConfiguration()
configuration.initialWorldMap = worldMap

// Run session
sceneView.session.run(configuration, options: [.resetTracking, .removeExistingAnchors])
```

Sharing a World Map

Loading

NEW

```
// Unarchive data to ARWorldMap
let worldMap = try NSKeyedUnarchiver.unarchivedObject(ofClass: ARWorldMap.self, from: data)

// Create tracking configuration
let configuration = ARWorldTrackingConfiguration()
configuration.initialWorldMap = worldMap

// Run session
sceneView.session.run(configuration, options: [.resetTracking, .removeExistingAnchors])
```

Sharing a World Map

Loading

NEW

```
// Unarchive data to ARWorldMap
let worldMap = try NSKeyedUnarchiver.unarchivedObject(ofClass: ARWorldMap.self, from: data)

// Create tracking configuration
let configuration = ARWorldTrackingConfiguration()
configuration.initialWorldMap = worldMap

// Run session
sceneView.session.run(configuration, options: [.resetTracking, .removeExistingAnchors])
```

Sharing a World Map

Loading

NEW

```
// Unarchive data to ARWorldMap
let worldMap = try NSKeyedUnarchiver.unarchivedObject(ofClass: ARWorldMap.self, from: data)

// Create tracking configuration
let configuration = ARWorldTrackingConfiguration()
configuration.initialWorldMap = worldMap

// Run session
sceneView.session.run(configuration, options: [.resetTracking, .removeExistingAnchors])
```

Sharing a World Map

Privacy

ARWorldMap uses features of the world around you

- No latitude/longitude information
- May include personally identifiable information

Sharing a World Map

Privacy

ARWorldMap uses features of the world around you

- No latitude/longitude information
- May include personally identifiable information

Treat serialized ARWorldMap as user-private data

- Encrypt at rest and in motion
- Obtain user consent for extended usage

Sharing a World Map

ARAnchor

ARAnchor represents a location and orientation in the physical world

- ARKit adds anchors for planes, objects, and images
- Also created and added via API

```
let anchor = ARAnchor(name: “Touched”, transform: transform)
session.add(anchor: anchor)
```

```swift
// Custom subclass of ARAnchor to annotate game location
class BoardAnchor: ARAnchor {
    private(set) var size: CGSize

    init(name: String, transform: float4x4, size: CGSize) {
        self.size = size
        super.init(name: name, transform: transform)
    }

    required init?(coder aDecoder: NSCoder) {
        self.size = aDecoder.decodeCGSize(forKey: "size")
        super.init(coder: aDecoder)
    }

    override func encode(with aCoder: NSCoder) {
        super.encode(with: aCoder)
        aCoder.encode(size, forKey: "size")
    }
```

```
// Custom subclass of ARAnchor to annotate game location
class BoardAnchor: ARAnchor {
    private(set) var size: CGSize

    init(name: String, transform: float4x4, size: CGSize) {
        self.size = size
        super.init(name: name, transform: transform)
    }

    required init?(coder aDecoder: NSCoder) {
        self.size = aDecoder.decodeCGSize(forKey: "size")
        super.init(coder: aDecoder)
    }

    override func encode(with aCoder: NSCoder) {
        super.encode(with: aCoder)
        aCoder.encode(size, forKey: "size")
    }
```

```
// Custom subclass of ARAnchor to annotate game location
class BoardAnchor: ARAnchor {
    private(set) var size: CGSize

    init(name: String, transform: float4x4, size: CGSize) {
        self.size = size
        super.init(name: name, transform: transform)
    }

    required init?(coder aDecoder: NSCoder) {
        self.size = aDecoder.decodeCGSize(forKey: "size")
        super.init(coder: aDecoder)
    }

    override func encode(with aCoder: NSCoder) {
        super.encode(with: aCoder)
        aCoder.encode(size, forKey: "size")
    }
```

Networking with Multipeer Connectivity

Peer-to-peer connectivity, no central server

Encryption and authentication are built in

Advertisement and discovery

Networking with Multipeer Connectivity

Networking with Multipeer Connectivity

Device starting the game creates a session, starts advertising

Networking with Multipeer Connectivity

Device starting the game creates a session, starts advertising

Other devices see session listed in menu

Networking with Multipeer Connectivity

Device starting the game creates a session, starts advertising

Other devices see session listed in menu

User selects game to join

- Device sends request
- Advertising device accepts or denies

Networking with Multipeer Connectivity

Device starting the game creates a session, starts advertising

Other devices see session listed in menu

User selects game to join

- Device sends request
- Advertising device accepts or denies

Once session set up, devices are peers in the network

Networking with Multipeer Connectivity

Networking with Multipeer Connectivity

API to send

- Data packets
- Resources as URLs
- Streams

Networking with Multipeer Connectivity

API to send

- Data packets
- Resources as URLs
- Streams

UDP for transport

```
enum Action {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)
}

struct HitCatapult: Codable {
    var catapultID: Int
    var hitPosition: float3
    var hitVel: float3
    var vortex: Bool
}

extension Action: Codable {
    init(from decoder: Decoder) throws { /* */ }
    func encode(to encoder: Encoder) throws { /* */ }
}
```

```
enum Action {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)
}

struct HitCatapult: Codable {
    var catapultID: Int
    var hitPosition: float3
    var hitVel: float3
    var vortex: Bool
}

extension Action: Codable {
    init(from decoder: Decoder) throws { /* */ }
    func encode(to encoder: Encoder) throws { /* */ }
}
```

```
enum Action {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)
}

struct HitCatapult: Codable {
    var catapultID: Int
    var hitPosition: float3
    var hitVel: float3
    var vortex: Bool
}

extension Action: Codable {
    init(from decoder: Decoder) throws { /* */ }
    func encode(to encoder: Encoder) throws { /* */ }
}
```

```
enum Action {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)
}

struct HitCatapult: Codable {
    var catapultID: Int
    var hitPosition: float3
    var hitVel: float3
    var vortex: Bool
}

extension Action: Codable {
    init(from decoder: Decoder) throws { /* */ }
    func encode(to encoder: Encoder) throws { /* */ }
}
```

```
// Sending physics data
func send(_ syncData: PhysicsSyncData) throws {
    let action = Action.physics(syncData)
    let encoder = PropertyListEncoder()
    encoder.outputFormat = .binary
    let data = try encoder.encode(action)

    try session.send(data, toPeers: peers, with: .unreliable)
}
```

```
// Sending physics data
func send(_ syncData: PhysicsSyncData) throws {
    let action = Action.physics(syncData)
    let encoder = PropertyListEncoder()
    encoder.outputFormat = .binary
    let data = try encoder.encode(action)

    try session.send(data, toPeers: peers, with: .unreliable)
}
```

```swift
// Sending physics data
func send(_ syncData: PhysicsSyncData) throws {
    let action = Action.physics(syncData)
    let encoder = PropertyListEncoder()
    encoder.outputFormat = .binary
    let data = try encoder.encode(action)

    try session.send(data, toPeers: peers, with: .unreliable)
}
```

Physics

SceneKit physics

All peers run physics simulation

“Server” sends updates to clients to ensure synchronization

Only “game state” relevant information is shared

Objects scaled approximately 10x for better performance

Physics

SceneKit physics

All peers run physics simulation

“Server” sends updates to clients to ensure synchronization

Only “game state” relevant information is shared

Objects scaled approximately 10x for better performance

Physics Data Optimization

Physics Data Optimization

Position	x, y, z

Physics Data Optimization

Position	x, y, z
Velocity	$\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta z}{\Delta t}$

Physics Data Optimization

Position	x, y, z
Velocity	$\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta z}{\Delta t}$
Angular velocity	x, y, z, ω

Physics Data Optimization

Position	x, y, z
Velocity	$\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta z}{\Delta t}$
Angular velocity	x, y, z, $\boldsymbol{\omega}$
Orientation	$\mathbf{i}$x, $\mathbf{i}$y, $\mathbf{i}$z, $\mathbf{r}$

Physics Data Optimization

Physics Data Optimization

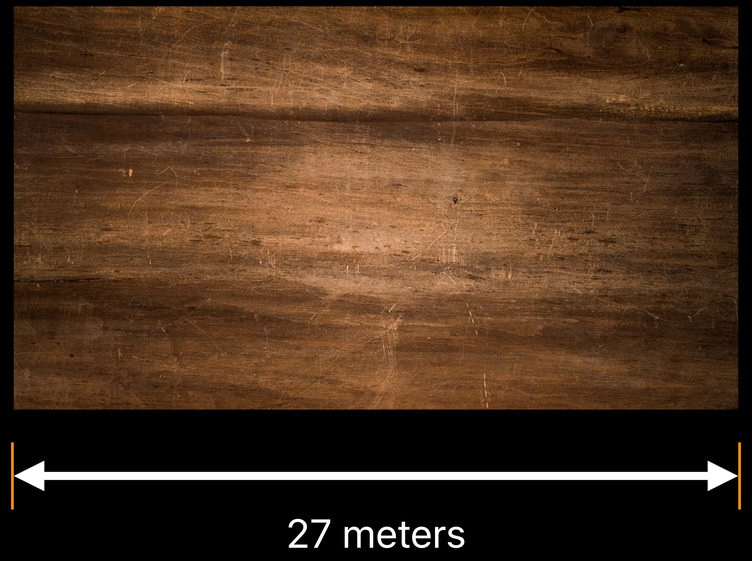
Physics Data Optimization

$\pm 10^{38}$

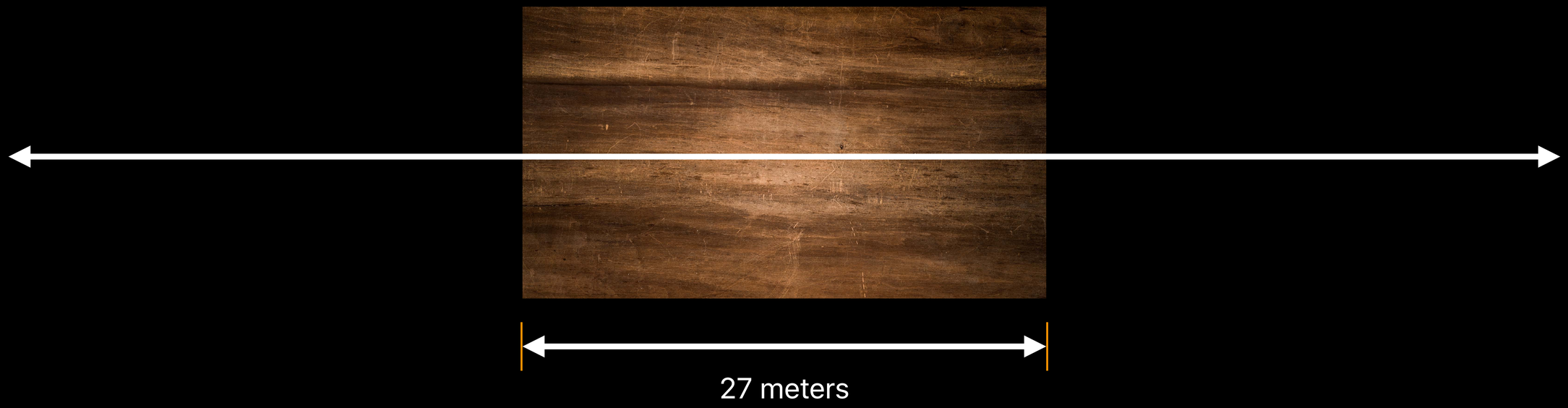
Sign Exponent Mantissa

Physics Data Optimization

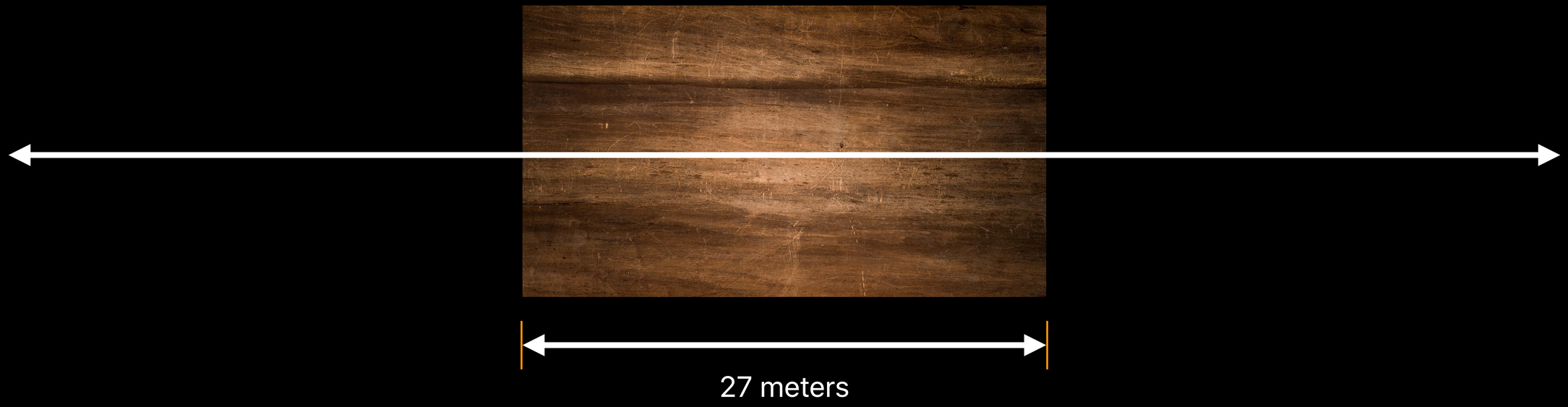
Physics Data Optimization



Physics Data Optimization



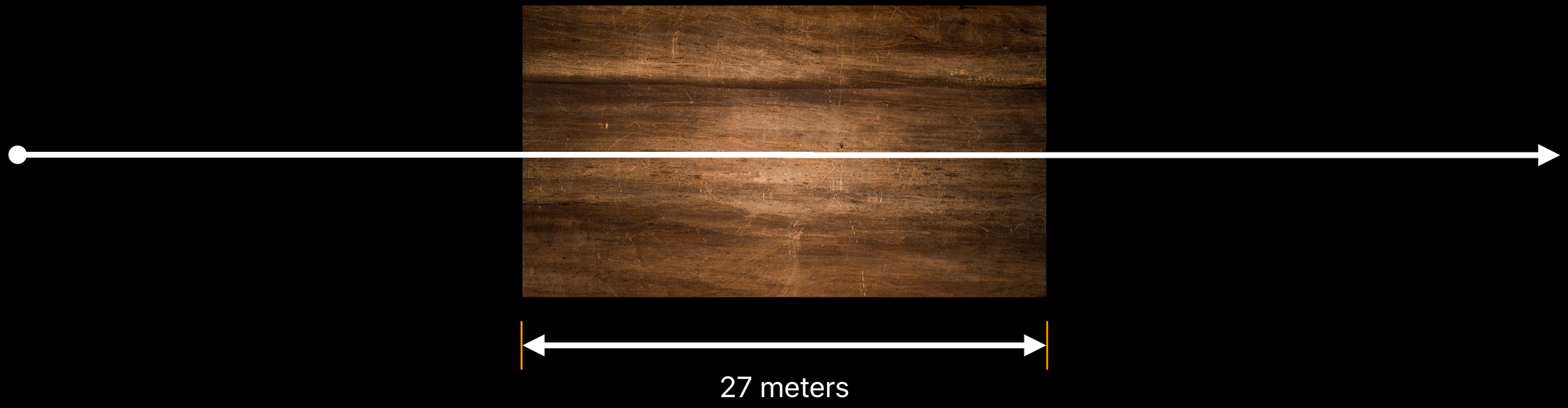
Physics Data Optimization



Physics Data Optimization

Sign Exponent Mantissa

- minValue



Physics Data Optimization

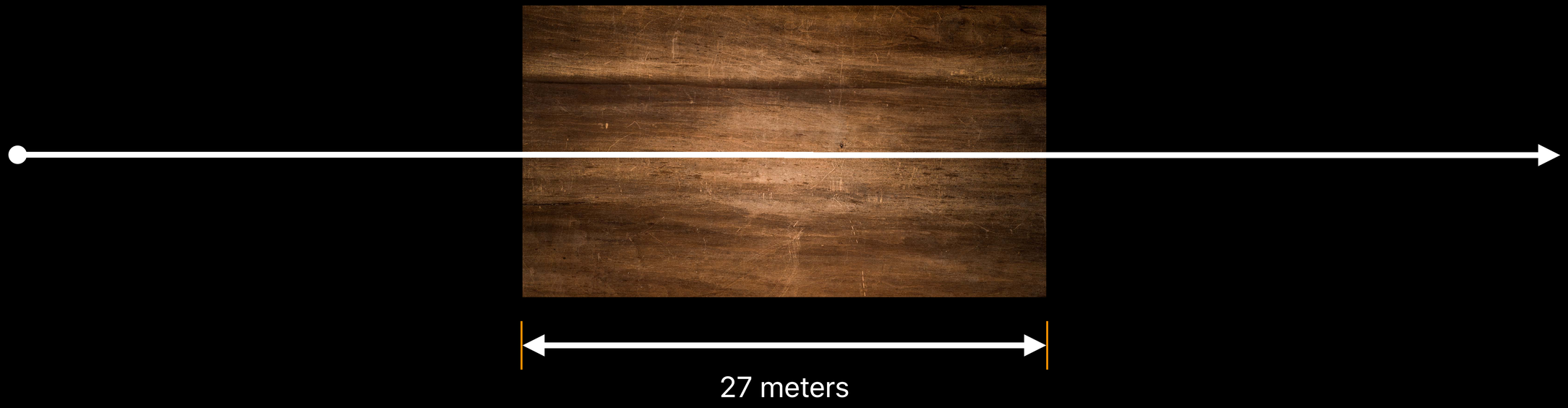
Sign

Exponent

Mantissa

- minValue

[0, 1]



Physics Data Optimization

Sign

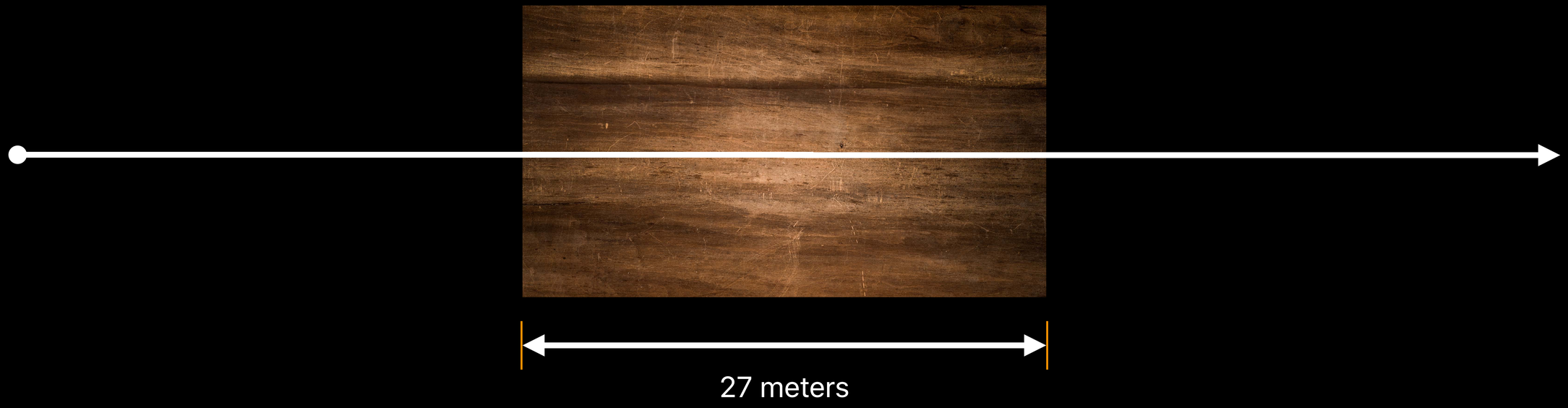
Exponent

Mantissa

- minValue

[0, 1]

* (maxBitValue)



Encoding—BitStream

Bit-packed encoding of values

Minimum size, fast serialization and deserialization

Purpose-built for network communications of binary data

- Not suitable for persistence
- Not robust in face of changing data schema

```swift
// BitStreamCodable protocols

protocol BitStreamEncodable {
   func encode(to bitStream: inout WritableBitStream) throws
}

protocol BitStreamDecodable {
    init(from bitStream: inout ReadableBitStream) throws
}

typealias BitStreamCodable = BitStreamEncodable & BitStreamDecodable

extension float3: BitStreamCodable {
    /* … */
}
```

```swift
// BitStreamCodable protocols

protocol BitStreamEncodable {
   func encode(to bitStream: inout WritableBitStream) throws
}

protocol BitStreamDecodable {
    init(from bitStream: inout ReadableBitStream) throws
}

typealias BitStreamCodable = BitStreamEncodable & BitStreamDecodable

extension float3: BitStreamCodable {
    /* … */
}
```

```swift
// BitStreamCodable protocols

protocol BitStreamEncodable {
    func encode(to bitStream: inout WritableBitStream) throws
}

protocol BitStreamDecodable {
    init(from bitStream: inout ReadableBitStream) throws
}

typealias BitStreamCodable = BitStreamEncodable & BitStreamDecodable

extension float3: BitStreamCodable {
    /* … */
}
```

```swift
// BitStreamCodable protocols

protocol BitStreamEncodable {
    func encode(to bitStream: inout WritableBitStream) throws
}

protocol BitStreamDecodable {
    init(from bitStream: inout ReadableBitStream) throws
}

typealias BitStreamCodable = BitStreamEncodable & BitStreamDecodable

extension float3: BitStreamCodable {
    /* … */
}
```

```swift
// Compressing floats for encoding
struct FloatCompressor {
    var minValue: Float
    var maxValue: Float
    var bits: Int
    private var maxBitValue: Double { pow(2.0, Double(bits)) - 1 }

    func write(_ value: Float, to stream: inout WritableBitStream) {
        let ratio = Double((value - minValue) / (maxValue - minValue))
        let clampedRatio = max(0.0, min(1.0, ratio))
        let bitPattern = UInt32(clampedRatio * maxBitValue)
        stream.appendUInt32(bitPattern, numberOfBits: bits)
    }
}
```

```
// Compressing floats for encoding
struct FloatCompressor {
    var minValue: Float
    var maxValue: Float
    var bits: Int
    private var maxBitValue: Double { pow(2.0, Double(bits)) - 1 }

    func write(_ value: Float, to stream: inout WritableBitStream) {
        let ratio = Double((value - minValue) / (maxValue - minValue))
        let clampedRatio = max(0.0, min(1.0, ratio))
        let bitPattern = UInt32(clampedRatio * maxBitValue)
        stream.appendUInt32(bitPattern, numberOfBits: bits)
    }
}
```

```swift
// Compressing floats for encoding
struct FloatCompressor {
    var minValue: Float
    var maxValue: Float
    var bits: Int
    private var maxBitValue: Double { pow(2.0, Double(bits)) - 1 }

    func write(_ value: Float, to stream: inout WritableBitStream) {
        let ratio = Double((value - minValue) / (maxValue - minValue))
        let clampedRatio = max(0.0, min(1.0, ratio))
        let bitPattern = UInt32(clampedRatio * maxBitValue)
        stream.appendUInt32(bitPattern, numberOfBits: bits)
    }
}
```

```swift
// BitStream Encoding Enums
enum Action: BitStreamCodable {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)

    enum CodingKeys: UInt32 {
        case gameAction
        case boardSetup
        case physics
    }

    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let gameAction):
            bitStream.appendUInt32(CodingKeys.gameAction.rawValue, numberOfBits: 2)
            try gameAction.encode(to: &bitStream)
        // ...
    }
```

```swift
// BitStream Encoding Enums
enum Action: BitStreamCodable {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)

    enum CodingKeys: UInt32 {
        case gameAction
        case boardSetup
        case physics
    }

    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let gameAction):
            bitStream.appendUInt32(CodingKeys.gameAction.rawValue, numberOfBits: 2)
            try gameAction.encode(to: &bitStream)
        // ...
    }
```

```
// BitStream Encoding Enums
enum Action: BitStreamCodable {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)

    enum CodingKeys: UInt32 {
        case gameAction
        case boardSetup
        case physics
    }

    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let gameAction):
            bitStream.appendUInt32(CodingKeys.gameAction.rawValue, numberOfBits: 2)
            try gameAction.encode(to: &bitStream)
        // ...
    }
```

```
// BitStream Encoding Enums
enum Action: BitStreamCodable {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)

    enum CodingKeys: UInt32 {
        case gameAction
        case boardSetup
        case physics
    }

    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let gameAction):
            bitStream.appendUInt32(CodingKeys.gameAction.rawValue, numberOfBits: 2)
            try gameAction.encode(to: &bitStream)
        // ...
    }
```

```
// BitStream Encoding Enums
enum Action: BitStreamCodable {
    case gameAction(GameAction)
    case boardSetup(BoardSetupAction)
    case physics(PhysicsSyncData)

    enum CodingKeys: UInt32 {
        case gameAction
        case boardSetup
        case physics
    }

    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let gameAction):
            bitStream.appendUInt32(CodingKeys.gameAction.rawValue, numberOfBits: 2)
            try gameAction.encode(to: &bitStream)
        // ...
    }
```

```swift
// Using CaseIterable to determine bits needed to encode
extension Action.CodingKeys: CaseIterable {}

extension RawRepresentable where Self: CaseIterable, RawValue: FixedWidthInteger {
    static var bits: Int {
        let casesCount = RawValue(allCases.count) - 1
        return RawValue.bitWidth - casesCount.leadingZeroBitCount
    }
}

extension WritableBitStream {
    mutating func appendEnum<T>(_ value: T)
        where T: CaseIterable & RawRepresentable, T: FixedWidthInteger  {
            appendUInt32(UInt32(value.rawValue), numberOfBits: type(of: value).bits)
    }
}
```

```swift
// Using CaseIterable to determine bits needed to encode
extension Action.CodingKeys: CaseIterable {}

extension RawRepresentable where Self: CaseIterable, RawValue: FixedWidthInteger {
    static var bits: Int {
        let casesCount = RawValue(allCases.count) - 1
        return RawValue.bitWidth - casesCount.leadingZeroBitCount
    }
}

extension WritableBitStream {
    mutating func appendEnum<T>(_ value: T)
        where T: CaseIterable & RawRepresentable, T: FixedWidthInteger  {
            appendUInt32(UInt32(value.rawValue), numberOfBits: type(of: value).bits)
    }
}
```

```swift
// Using CaseIterable to determine bits needed to encode
extension Action.CodingKeys: CaseIterable {}

extension RawRepresentable where Self: CaseIterable, RawValue: FixedWidthInteger {
    static var bits: Int {
        let casesCount = RawValue(allCases.count) - 1
        return RawValue.bitWidth - casesCount.leadingZeroBitCount
    }
}

extension WritableBitStream {
    mutating func appendEnum<T>(_ value: T)
        where T: CaseIterable & RawRepresentable, T: FixedWidthInteger  {
            appendUInt32(UInt32(value.rawValue), numberOfBits: type(of: value).bits)
    }
}
```

```swift
// Using CaseIterable to determine bits needed to encode
extension Action.CodingKeys: CaseIterable {}

extension RawRepresentable where Self: CaseIterable, RawValue: FixedWidthInteger {
    static var bits: Int {
        let casesCount = RawValue(allCases.count) - 1
        return RawValue.bitWidth - casesCount.leadingZeroBitCount
    }
}
```

```swift
extension WritableBitStream {
    mutating func appendEnum<T>(_ value: T)
        where T: CaseIterable & RawRepresentable, T: FixedWidthInteger  {
            appendUInt32(UInt32(value.rawValue), numberOfBits: type(of: value).bits)
    }
}
```

```
extension Action: BitStreamCodable {
    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let data):
            bitStream.appendEnum(CodingKeys.gameAction)
            try data.encode(to: &bitStream)
        // …
     }
}
```

```
extension Action: BitStreamCodable {
    func encode(to bitStream: inout WritableBitStream) throws {
        switch self {
        case .gameAction(let data):
            bitStream.appendEnum(CodingKeys.gameAction)
            try data.encode(to: &bitStream)
        // …
    }
}
```

BitStreamCodable

Performance

	Size (bytes)	Encoding (µsec)	Decoding (µsec)

10.5-inch iPad Pro running iOS 12 Developer Seed 1

BitStreamCodable

Performance

	Size (bytes)	Encoding (µsec)	Decoding (µsec)
Codable with PropertyListEncoder	92	19.1	74.9

10.5-inch iPad Pro running iOS 12 Developer Seed 1

BitStreamCodable

Performance

	Size (bytes)	Encoding (µsec)	Decoding (µsec)
Codable with PropertyListEncoder	92	19.1	74.9
BitStreamCodable with custom implementation	9	10.7	5.9

10.5-inch iPad Pro running iOS 12 Developer Seed 1

Combining Encodings

Physics data encoded in BitStreams

Other commands encoded with Swift Codable, binary property lists

How to combine?

```
// Combining Codable and BitStreamCodable
extension BitStreamCodable where Self: Codable {
    func encode(to bitStream: inout WritableBitStream) throws {
        let encoder = PropertyListEncoder()
        encoder.outputFormat = .binary
        let data = try encoder.encode(self)
        bitStream.append(data)
    }

   init(from bitStream: inout ReadableBitStream) throws {
        let data = try bitStream.readData()
        let decoder = PropertyListDecoder()
        self = try decoder.decode(Self.self, from: data)
    }
}

struct StartGameMusicTime: Codable, BitStreamCodable { /* … */ }
```

```swift
// Combining Codable and BitStreamCodable
extension BitStreamCodable where Self: Codable {
    func encode(to bitStream: inout WritableBitStream) throws {
        let encoder = PropertyListEncoder()
        encoder.outputFormat = .binary
        let data = try encoder.encode(self)
        bitStream.append(data)
    }

   init(from bitStream: inout ReadableBitStream) throws {
        let data = try bitStream.readData()
        let decoder = PropertyListDecoder()
        self = try decoder.decode(Self.self, from: data)
    }
}
```

```swift
struct StartGameMusicTime: Codable, BitStreamCodable { /* … */ }
```

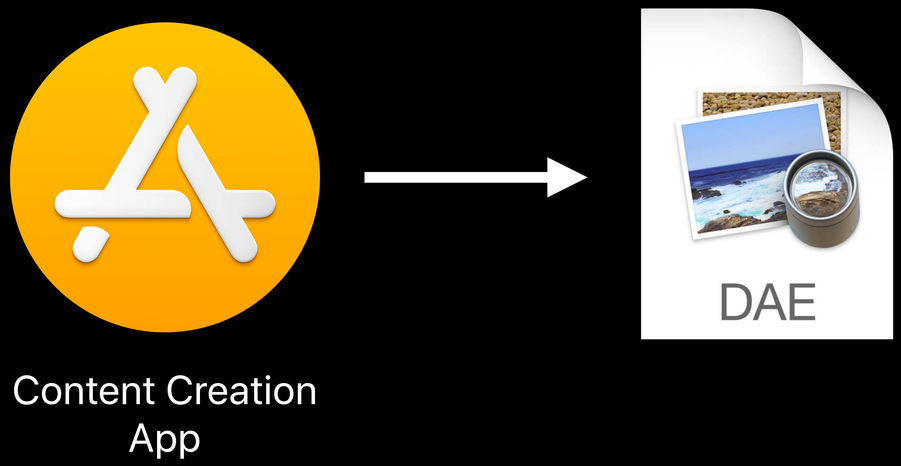
```
// Combining Codable and BitStreamCodable
extension BitStreamCodable where Self: Codable {
    func encode(to bitStream: inout WritableBitStream) throws {
        let encoder = PropertyListEncoder()
        encoder.outputFormat = .binary
        let data = try encoder.encode(self)
        bitStream.append(data)
    }

   init(from bitStream: inout ReadableBitStream) throws {
        let data = try bitStream.readData()
        let decoder = PropertyListDecoder()
        self = try decoder.decode(Self.self, from: data)
    }
}
```

```
struct StartGameMusicTime: Codable, BitStreamCodable { /* … */ }
```

Assets for Game Levels

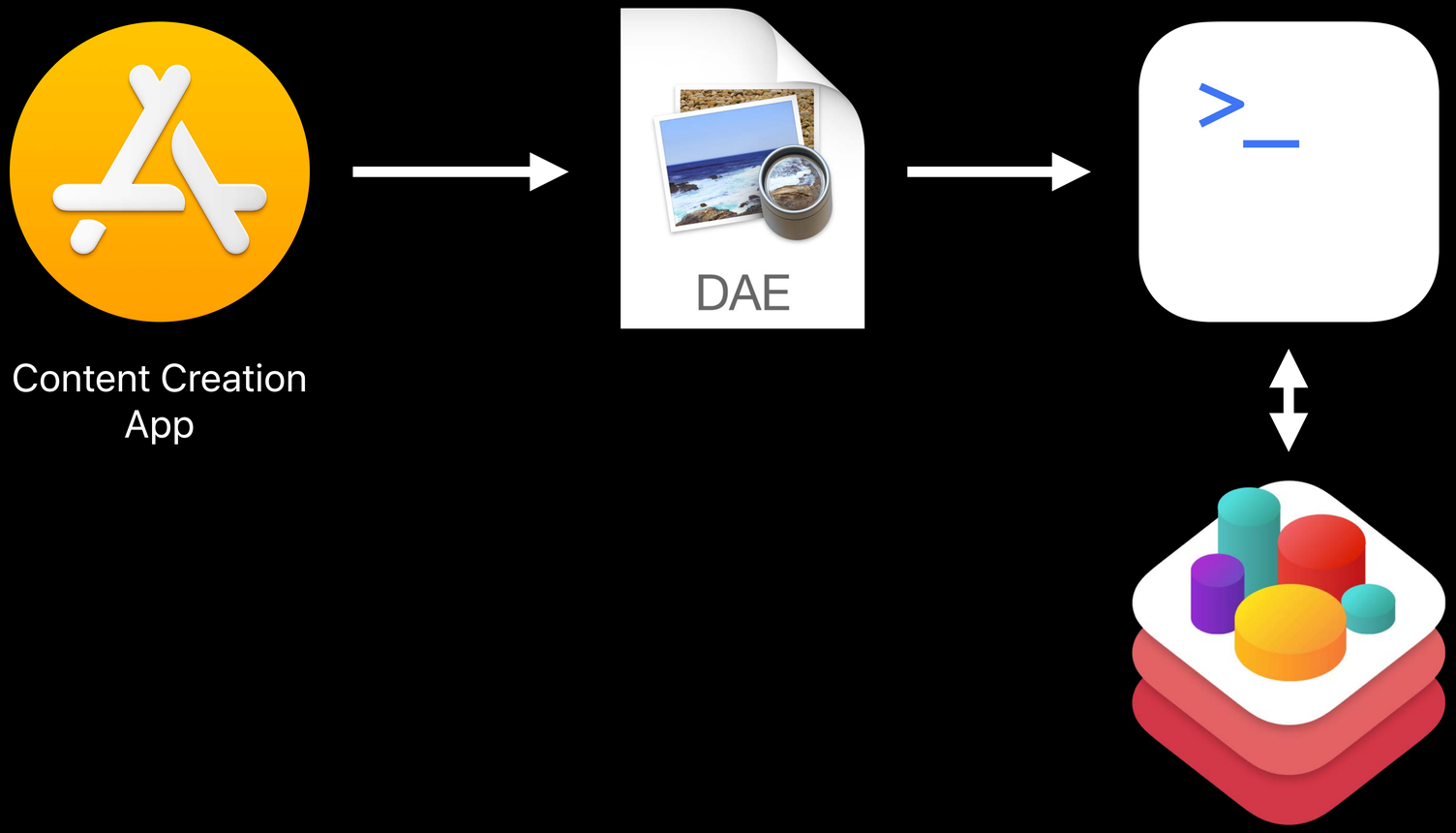
Content Creation App

Assets for Game Levels

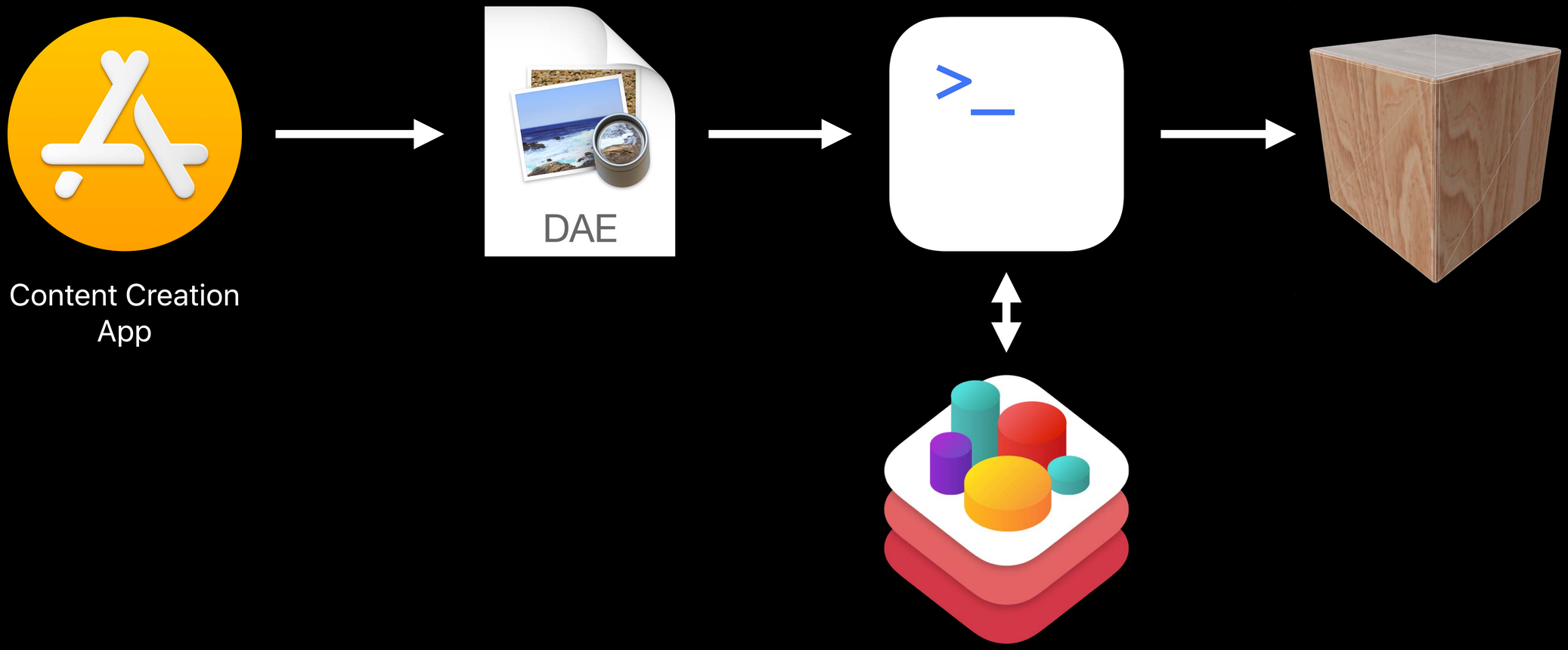


Assets for Game Levels

Assets for Game Levels



Assets for Game Levels



Assets for Game Levels

Use level of detail to optimize for different distances

- Nearby objects get chamfered edges on blocks
- Distant objects get less detailed textures, fewer polygons

Assets for Game Levels

Use level of detail to optimize for different distances

- Nearby objects get chamfered edges on blocks
- Distant objects get less detailed textures, fewer polygons

Physics bodies are the same

- Use predefined types (box, sphere, etc.) wherever possible for performance
- SceneKit will build convex hull if not otherwise specified

Assets for Game Levels

Assets for Game Levels

Assets for Game Levels

Assets for Game Levels

Flag Simulation

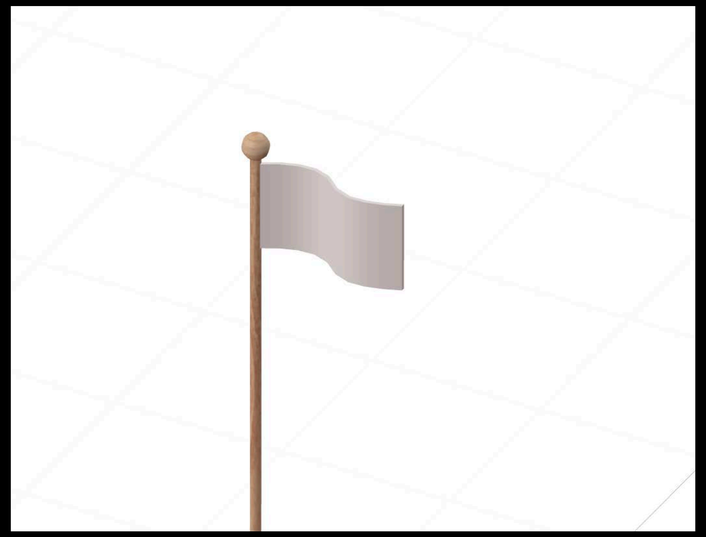
SceneKit static asset

Swift class

- Build Metal command queue
- Apply results back to SceneKit model

Metal compute shader

- Computes forces in a mesh
- Produces vertices and shadows

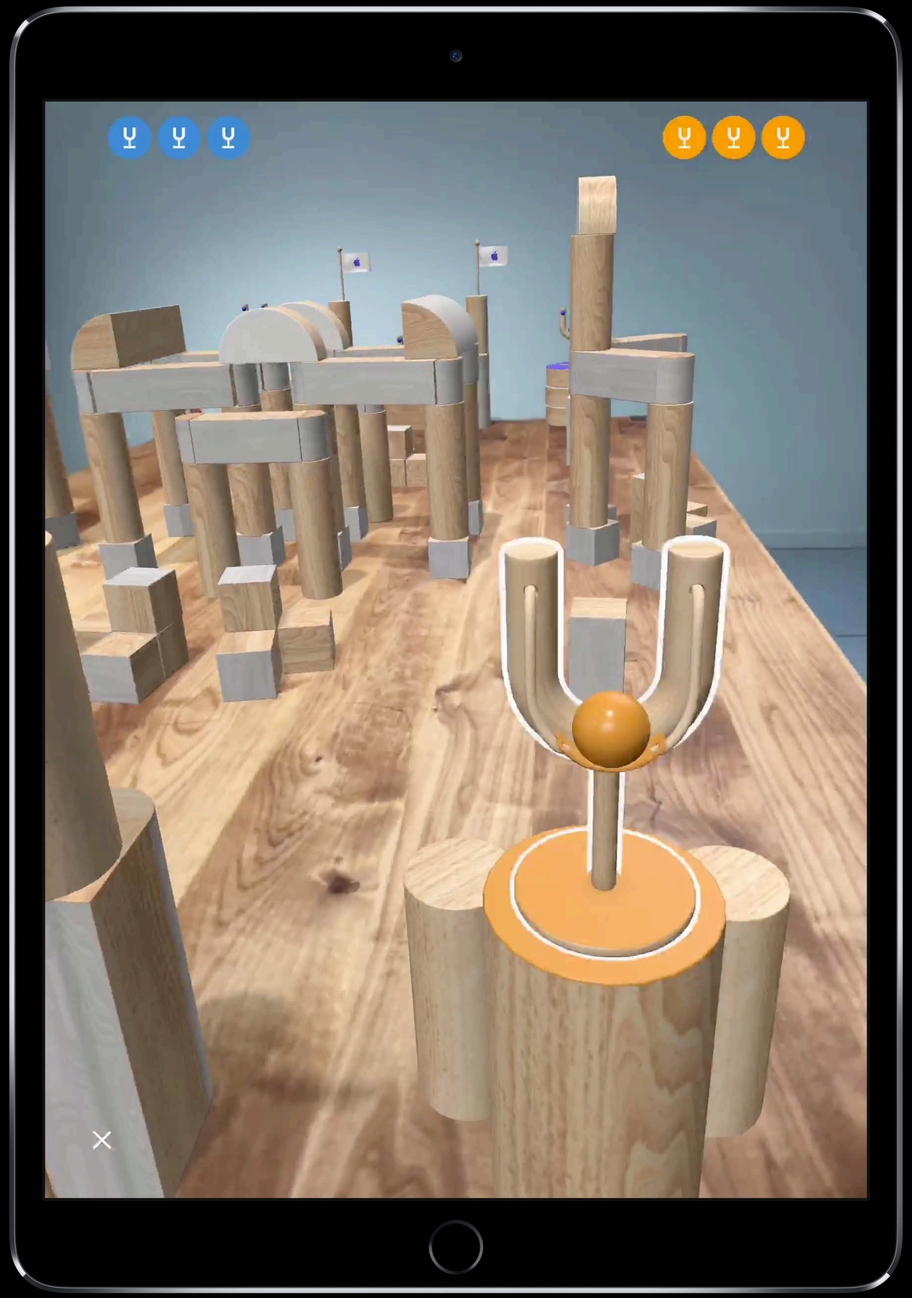


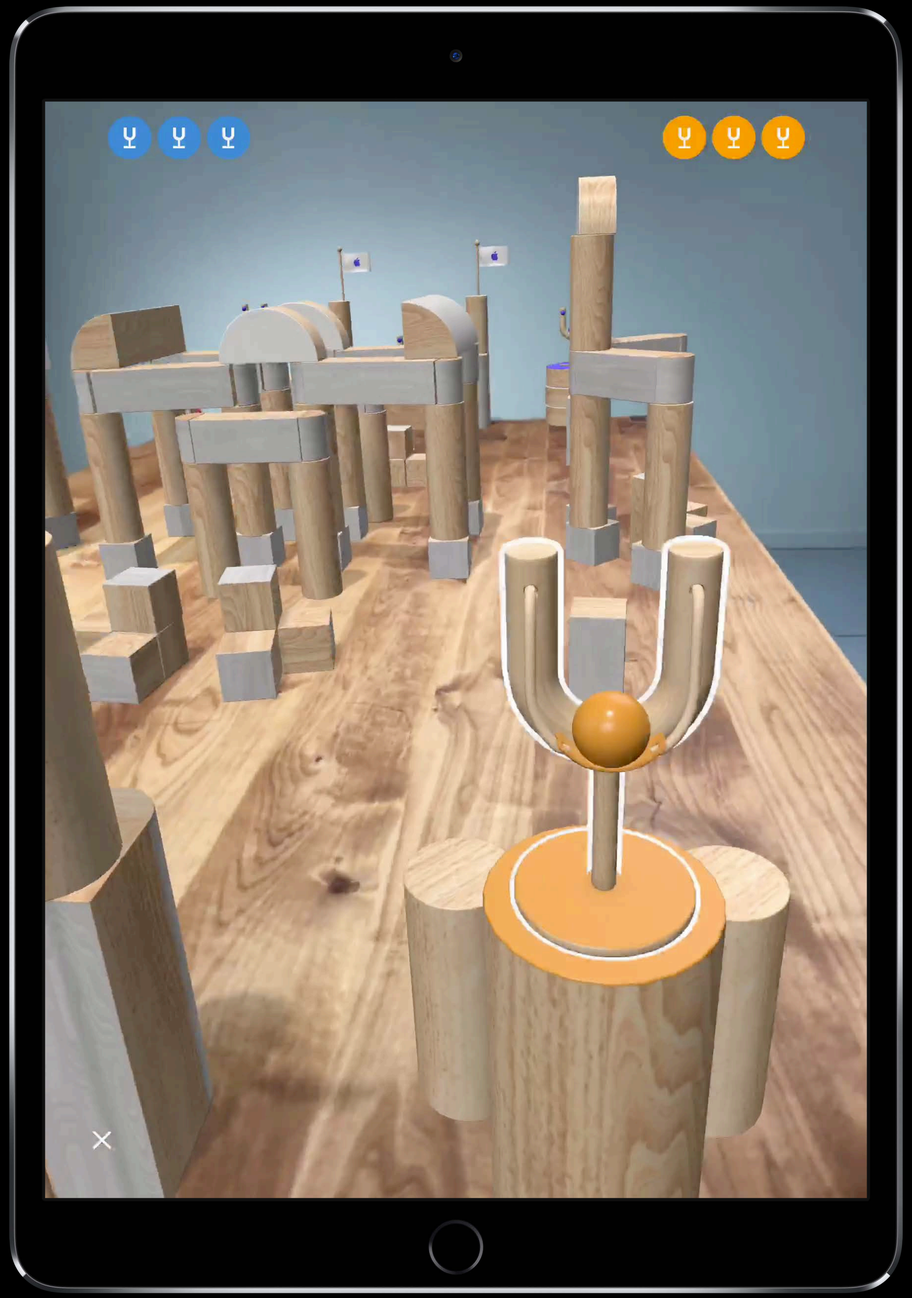
Audio in SwiftShot

Positions sounds in the world

Limit app size

Leverage MIDI instrument support from AVFoundation





```
#!/usr/bin/swift
// Audio asset processing

import Foundation

if CommandLine.argc < 3 {
    print("Usage: clean_preset <path/to/input_preset> <path/to/output_preset>")
    exit(1)
}

let inputPath = CommandLine.arguments[1]
let outputPath = CommandLine.arguments[2]

let inputURL = URL(fileURLWithPath: inputPath)
let outputURL = URL(fileURLWithPath: outputPath)

var plist: [String: Any]?
// …
```

```
#!/usr/bin/swift
// Audio asset processing

import Foundation

if CommandLine.argc < 3 {
    print("Usage: clean_preset <path/to/input_preset> <path/to/output_preset>")
    exit(1)
}

let inputPath = CommandLine.arguments[1]
let outputPath = CommandLine.arguments[2]

let inputURL = URL(fileURLWithPath: inputPath)
let outputURL = URL(fileURLWithPath: outputPath)

var plist: [String: Any]?
// …
```

Summary

Augmented Reality provides new opportunities for engaging games

Design with AR in mind from the start

Game play is the most important part of game design

Download the SwiftShot developer sample code

Play SwiftShot with us in the AR Game Room

More Information

https://developer.apple.com/wwdc18/605

ARKit Lab	Technology Lab 5	Wednesday 3:00PM
AR Get Together	Technology Lab 10	Wednesday 6:15PM


WWDC18