USDZ Thumbnailer Script Documentation

Overview

This Swift script automatically generates PNG thumbnail images from USDZ (Universal Scene Description) 3D model files. It's designed for batch processing with intelligent caching, error logging, and automatic deployment integration.

Features

- **Batch Processing**: Processes all USDZ files in a directory
- **Smart Caching**: Only regenerates thumbnails when USDZ files are newer than existing PNGs
- **Error Logging**: Redirects verbose USD errors to a log file
- **Deployment Integration**: Automatically runs index generation and deployment scripts
- **Clean Output**: Minimal console output with clear status indicators

Usage

```bash
swift usdz_thumbnailer.swift [directory]
```

- If no directory is specified, uses current directory (`.`)
- Creates PNG thumbnails alongside USDZ files
- Runs post-processing scripts if they exist

Script Breakdown

1. Imports and Setup

```swift
#!/usr/bin/env swift
import Foundation
import SceneKit
import SceneKit.ModelIO
import AppKit
```

- **Shebang**: Allows script to run directly without `swift` command
- **Foundation**: Core system functionality
- **SceneKit/ModelIO**: 3D rendering and model loading
- **AppKit**: macOS image handling (NSImage, NSBitmapImageRep)

2. ARQLThumbnailGenerator Class

This class handles the core thumbnail generation using Apple's SceneKit framework.

```swift
class ARQLThumbnailGenerator {
    private let device = MTLCreateSystemDefaultDevice()!
```

- Creates a Metal device for hardware-accelerated rendering
- Metal provides GPU-based rendering for better performance

thumbnail() Method

```swift
func thumbnail(for url: URL, size: CGSize, time: TimeInterval = 0) -> NSImage?
```

Parameters:

- `url`: File path to the USDZ model
- `size`: Desired thumbnail dimensions (we use 512x512)
- `time`: Animation time point (default 0 for static)

Process:

1. **Renderer Setup**: Creates SCNRenderer with automatic lighting
2. **Model Loading**: Uses ModelIO to load USDZ file
3. **Texture Loading**: Calls `loadTextures()` to ensure materials display correctly
4. **Scene Creation**: Converts ModelIO asset to SceneKit scene
5. **Validation**: Checks if scene has geometry (prevents empty thumbnails)
6. **Rendering**: Takes high-quality snapshot with 4x antialiasing

3. Command Line Argument Handling

```swift
let args = CommandLine.arguments
let directory = args.count >= 2 ? args[1] : "."
```

- Defaults to current directory if no path provided
- Maintains backward compatibility while improving usability

4. File Discovery and Logging Setup

```swift
let files = try FileManager.default.contentsOfDirectory(atPath: directory).filter { $0.hasSuffix(".usdz") }
let logPath = directory + "/usdz_thumbnailer.log"
let logFile = open(logPath, O_WRONLY | O_CREAT | O_TRUNC, 0o644)
```

- **File Discovery**: Finds all `.usdz` files in the target directory
- **Log File**: Creates/overwrites log file for error capture
- **Permissions**: Sets log file to readable/writable by owner, readable by others

5. Main Processing Loop

The core logic that processes each USDZ file with intelligent caching.

Timestamp Comparison

```swift
if FileManager.default.fileExists(atPath: outputPath) {
    // Compare modification dates
    if usdzDate > pngDate {
        shouldGenerate = usdzDate > pngDate
    }
}
```

Smart Caching Logic:

- Checks if PNG thumbnail already exists
- Compares modification dates of USDZ and PNG files
- Only regenerates if USDZ is newer than existing PNG
- Skips unchanged files with "- filename (up to date)" message

Error Suppression

```swift
let originalStderr = dup(STDERR_FILENO)
dup2(logFile, STDERR_FILENO)
// ... thumbnail generation ...
dup2(originalStderr, STDERR_FILENO)
```

Why This Matters:

- USD library outputs verbose error messages to stderr
- These errors are often non-fatal but cluttered the console
- Redirects errors to log file during processing
- Restores normal error output afterward

Image Processing and Saving

```swift
let data = thumbnail.tiffRepresentation!
let bitmap = NSBitmapImageRep(data: data)!
let pngData = bitmap.representation(using: .png, properties: [:])!
try pngData.write(to: outputURL)
```

Image Conversion Process:

1. SceneKit returns NSImage (native macOS format)
2. Convert to TIFF representation (intermediate format)
3. Create bitmap representation for format control
4. Convert to PNG with no compression properties
5. Write directly to filesystem

6. Status Reporting

```swift
print("✓ \(file)")   // Success
print("✗ \(file)")   // Failure
print("- \(file) (up to date)")  // Skipped
```

Output Indicators:

- ✓: Successfully generated new thumbnail
- ✗: Failed to generate (usually corrupted USDZ)
- -: Skipped because existing thumbnail is current

7. Post-Processing Scripts

Index Generation

```swift
let generateIndexScript = directory + "/generate_index_with_USDZ.sh"
if FileManager.default.fileExists(atPath: generateIndexScript) {
    // Run script
}
```

- Looks for `generate_index_with_USDZ.sh` in the same directory
- Typically generates HTML index pages or catalogs
- Runs before deployment to ensure fresh content

Deployment

```swift
let deployScript = directory + "/deploy.sh"
if FileManager.default.fileExists(atPath: deployScript) {
    // Run script with proper working directory
}
```

- Executes `deploy.sh` for automated deployment
- Sets working directory correctly for Git operations
- Reports exit codes for debugging

Error Handling

Common Issues and Solutions

"Runtime Error: Failed to open layer"

- Indicates corrupted or invalid USDZ file
- Script continues processing other files
- Error details saved to log file

"Empty scene" warnings

- USDZ loaded but contains no visible geometry
- Often happens with corrupted files
- Results in failed thumbnail generation

Git deployment failures

- Usually requires `git pull` before push
- Script reports exit codes for troubleshooting
- Check authentication and repository access

Performance Considerations

Caching Benefits:

- First run: Processes all files (slower)
- Subsequent runs: Only processes changed files (much faster)
- Typical speedup: 10-100x for unchanged files

Hardware Requirements:

- Requires Metal-compatible GPU
- Benefits from more GPU memory for complex models
- CPU usage minimal due to GPU acceleration

File Size Impact:

- Larger USDZ files take longer to process
- Complex materials/textures increase render time
- 512x512 PNG output is good balance of quality/size

Integration Examples

Basic Usage

```bash
# Process current directory
./usdz_thumbnailer.swift

# Process specific directory
./usdz_thumbnailer.swift /path/to/models
```

With Deployment Pipeline

```bash
# Your workflow might be:
1. Edit USDZ files
2. Run thumbnailer (generates PNGs + runs scripts)
3. generate_index_with_USDZ.sh creates HTML catalog
4. deploy.sh pushes to GitHub Pages
```

Troubleshooting

```bash
# Check log file for detailed errors
cat usdz_thumbnailer.log

# Force regeneration (delete existing PNGs)
rm *.png && ./usdz_thumbnailer.swift

# Test single file
swift -c "print(ARQLThumbnailGenerator().thumbnail(for: URL(fileURLWithPath: \"test.usdz\"), size: CGSize(width
```

Output Files

Generated Files:

- `filename.png`: Thumbnail for `filename.usdz`
- `usdz_thumbnailer.log`: Error log (only if errors occur)

File Locations:

- All files created in same directory as USDZ files
- No subdirectories created
- Maintains flat file structure for web deployment