ClipForge2 - Bug Fixes & Production Readiness Checklist

Critical Bug Fixes

1. Memory Leak Prevention

Issue: Texture and Buffer Memory Leaks

Location: `/app/src/main/cpp/gpu/`

Fix in GPUEffect.cpp:

```
// Add proper cleanup in destructor
GPUEffect::~GPUEffect() {
    cleanup();
}

void GPUEffect::cleanup() {
    if (program_ != 0) {
        glDeleteProgram(program_);
        program_ = 0;
    }

    if (vao_ != 0) {
        glDeleteVertexArrays(1, &amp;vao_);
        vao_ = 0;
    }

    if (vbo_ != 0) {
        glDeleteBuffers(1, &amp;vbo_);
        vbo_ = 0;
    }

    // Clear framebuffers
    for (auto fbo : framebuffers_) {
        glDeleteFramebuffers(1, &amp;fbo);
    }
    framebuffers_.clear();
}
```

Issue: Activity Memory Leaks

Location: `/app/src/main/kotlin/ui/EditorActivity.kt`

Fix:

```
override fun onDestroy() {
    super.onDestroy()

    // Clear all observers
    viewModel.clips.removeObservers(this)
    viewModel.selectedClip.removeObservers(this)
    viewModel.isPlaying.removeObservers(this)

    // Release video engine resources
    binding.previewView.release()

    // Clear adapters
    binding.timelineRecycler.adapter = null
    binding.effectsRecycler.adapter = null

    // Clear binding reference
    _binding = null
}
```

2. Thread Safety Issues

Issue: Concurrent Access to Clip List

Location: `/app/src/main/cpp/core/Timeline.cpp`

Fix:

```
#include &lt;mutex&gt;

class Timeline {
private:
    std::vector&lt;std::shared_ptr&lt;Clip&gt;&gt; clips_;
    mutable std::mutex clipsMutex_;

public:
    void addClip(std::shared_ptr&lt;Clip&gt; clip) {
        std::lock_guard&lt;std::mutex&gt; lock(clipsMutex_);
        clips_.push_back(clip);
    }

    void removeClip(const std::string&amp;amp; clipId) {
        std::lock_guard&lt;std::mutex&gt; lock(clipsMutex_);
        clips_.erase(
            std::remove_if(clips_.begin(), clips_.end(),
                [&amp;amp;clipId](const auto&amp;amp; clip) {
                    return clip-&gt;getId() == clipId;
                }),
            clips_.end()
        );
    }

    std::vector&lt;std::shared_ptr&lt;Clip&gt;&gt; getClips() const {
        std::lock_guard&lt;std::mutex&gt; lock(clipsMutex_);
```

```
        return clips_; // Returns copy
    }
};
```

3. Null Pointer Exceptions

Issue: Unhandled Null Views

Location: Throughout Kotlin UI files

Fix - Add ViewBinding Null Safety:

```
class EditorActivity : AppCompatActivity() {
    private var _binding: ActivityEditorBinding? = null
    private val binding get() = _binding!!

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _binding = ActivityEditorBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }

    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }
}
```

4. Export Crashes

Issue: Export Fails on Large Videos

Location: `/app/src/main/cpp/encoding/VideoEncoder.cpp`

Fix - Add Memory Management:

```
bool VideoEncoder::encodeFrame(AVFrame* frame) {
    // Check memory before encoding
    size_t availableMemory = getAvailableMemory();
    size_t requiredMemory = estimateFrameMemory(frame);

    if (availableMemory &lt; requiredMemory * 2) {
        Logger::warning("Low memory, clearing cache");
        clearEncoderCache();
    }

    // Add timeout for encoding
    auto future = std::async(std::launch::async, [this, frame]() {
        return avcodec_send_frame(codecContext_, frame);
    });
```

```
    auto status = future.wait_for(std::chrono::seconds(5));
    if (status == std::future_status::timeout) {
        Logger::error("Frame encoding timeout");
        return false;
    }

    int ret = future.get();
    if (ret &lt; 0) {
        Logger::error("Error sending frame: %d", ret);
        return false;
    }

    return true;
}

size_t VideoEncoder::getAvailableMemory() {
    #ifdef __ANDROID__
    // Android-specific memory check
    struct mallinfo info = mallinfo();
    return info.fordblks;
    #else
    return 0;
    #endif
}
```

5. Audio Sync Issues

Issue: Audio/Video Out of Sync

Location: `/app/src/main/cpp/audio/AudioMixer.cpp`

Fix - Proper PTS Handling:

```
class AudioMixer {
private:
    int64_t audioPts_ = 0;
    int64_t videoPts_ = 0;
    const int64_t MAX_SYNC_DIFF = 50000; // 50ms

public:
    void syncAudioToVideo(int64_t videoPts) {
        int64_t diff = videoPts - audioPts_;

        if (std::abs(diff) &gt; MAX_SYNC_DIFF) {
            Logger::warning("Audio sync drift: %lld ms", diff / 1000);

            // Resync audio
            audioPts_ = videoPts;

            // Adjust audio buffer
            if (diff &gt; 0) {
                // Video ahead - skip audio frames
                skipAudioFrames(diff);
            } else {
```

```
                // Audio ahead - insert silence
                insertSilence(-diff);
            }
        }
    }

    void skipAudioFrames(int64_t duration) {
        int framesToSkip = (duration * sampleRate_) / 1000000;
        audioBuffer_.erase(
            audioBuffer_.begin(),
            audioBuffer_.begin() + framesToSkip
        );
    }

    void insertSilence(int64_t duration) {
        int samplesToAdd = (duration * sampleRate_) / 1000000;
        audioBuffer_.insert(
            audioBuffer_.begin(),
            samplesToAdd,
            0.0f
        );
    }
};
```

6. UI Freezing

Issue: Main Thread Blocking

Location: `/app/src/main/kotlin/ui/viewmodels/EditorViewModel.kt`

Fix - Use Coroutines Properly:

```
class EditorViewModel : ViewModel() {

    private val ioDispatcher = Dispatchers.IO
    private val mainDispatcher = Dispatchers.Main

    fun exportVideo(config: ExportConfig) {
        viewModelScope.launch(ioDispatcher) {
            try {
                _isExporting.postValue(true)

                // Heavy export work on background thread
                val result = withContext(ioDispatcher) {
                    videoEngine.export(config) { progress -&gt;
                        // Update progress on main thread
                        launch(mainDispatcher) {
                            _exportProgress.value = progress
                        }
                    }
                }

                // Update UI on main thread
                withContext(mainDispatcher) {
```

```
                    _isExporting.value = false
                    _exportResult.value = result
                }

            } catch (e: Exception) {
                withContext(mainDispatcher) {
                    _error.value = "Export failed: ${e.message}"
                    _isExporting.value = false
                }
            }
        }
    }

    fun processEffect(effect: Effect) {
        viewModelScope.launch(ioDispatcher) {
            try {
                // Process effect in background
                val processed = videoEngine.processEffect(effect)

                // Update UI on main thread
                withContext(mainDispatcher) {
                    _previewFrame.value = processed
                }
            } catch (e: Exception) {
                withContext(mainDispatcher) {
                    _error.value = "Effect processing failed"
                }
            }
        }
    }
}
```

7. File I/O Errors

Issue: Scoped Storage Handling (Android 10+)

Location: `/app/src/main/kotlin/data/repository/MediaRepository.kt`

Fix - Proper File Access:

```
class MediaRepository(private val context: Context) {

    fun saveExportedVideo(
        inputPath: String,
        fileName: String
    ): Uri? {
        return if (Build.VERSION.SDK_INT &gt;= Build.VERSION_CODES.Q) {
            saveVideoToMediaStore(inputPath, fileName)
        } else {
            saveVideoToExternalStorage(inputPath, fileName)
        }
    }

    @RequiresApi(Build.VERSION_CODES.Q)
```

```
    private fun saveVideoToMediaStore(
        inputPath: String,
        fileName: String
    ): Uri? {
        val contentValues = ContentValues().apply {
            put(MediaStore.Video.Media.DISPLAY_NAME, fileName)
            put(MediaStore.Video.Media.MIME_TYPE, "video/mp4")
            put(MediaStore.Video.Media.RELATIVE_PATH, Environment.DIRECTORY_MOVIES + "/Cl
            put(MediaStore.Video.Media.IS_PENDING, 1)
        }

        val resolver = context.contentResolver
        val uri = resolver.insert(
            MediaStore.Video.Media.EXTERNAL_CONTENT_URI,
            contentValues
        ) ?: return null

        try {
            resolver.openOutputStream(uri)?.use { output -&gt;
                File(inputPath).inputStream().use { input -&gt;
                    input.copyTo(output)
                }
            }

            // Mark as not pending
            contentValues.clear()
            contentValues.put(MediaStore.Video.Media.IS_PENDING, 0)
            resolver.update(uri, contentValues, null, null)

            return uri

        } catch (e: Exception) {
            // Delete partial file
            resolver.delete(uri, null, null)
            return null
        }
    }

    private fun saveVideoToExternalStorage(
        inputPath: String,
        fileName: String
    ): Uri? {
        val moviesDir = Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_MOVIES
        )
        val clipForgeDir = File(moviesDir, "ClipForge")

        if (!clipForgeDir.exists()) {
            clipForgeDir.mkdirs()
        }

        val outputFile = File(clipForgeDir, fileName)

        return try {
            File(inputPath).copyTo(outputFile, overwrite = true)
            Uri.fromFile(outputFile)
```

```
        } catch (e: Exception) {
            null
        }
    }
}
```

Production Readiness Checklist

✅ Performance Optimization

- [ ] **ProGuard/R8 Configuration**

```
# File: /app/proguard-rules.pro

-keepattributes SourceFile,LineNumberTable
-renamesourcefileattribute SourceFile

# Keep native methods
-keepclasseswithmembernames class * {
    native &lt;methods&gt;;
}

# Keep ViewModels
-keep class * extends androidx.lifecycle.ViewModel {
    &lt;init&gt;();
}

# Keep Parcelize
-keep interface android.os.Parcelable
-keep class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator *;
}

# FFmpeg
-keep class com.arthenica.** { *; }

# OpenGL
-keep class javax.microedition.khronos.** { *; }
```

- [ ] **Enable Code Shrinking in build.gradle**

```
android {
    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'progu
        }
    }
}
```

✅ Crash Reporting

- [ ] **Integrate Firebase Crashlytics**

```
// File: /app/src/main/kotlin/ClipForgeApplication.kt

class ClipForgeApplication : Application() {
    override fun onCreate() {
        super.onCreate()

        // Initialize Crashlytics
        FirebaseCrashlytics.getInstance().apply {
            setCrashlyticsCollectionEnabled(!BuildConfig.DEBUG)
        }

        // Set custom exception handler
        Thread.setDefaultUncaughtExceptionHandler { thread, throwable -&gt;
            FirebaseCrashlytics.getInstance().recordException(throwable)
            // Original handler
        }
    }
}
```

✅ Analytics

- [ ] **Track Key User Actions**

```
class AnalyticsManager(private val context: Context) {
    private val firebaseAnalytics = FirebaseAnalytics.getInstance(context)

    fun logProjectCreated(template: String) {
        firebaseAnalytics.logEvent("project_created") {
            param("template", template)
        }
    }

    fun logVideoExported(
        duration: Long,
        quality: String,
        codec: String
    ) {
        firebaseAnalytics.logEvent("video_exported") {
            param("duration_seconds", duration / 1000)
            param("quality", quality)
            param("codec", codec)
        }
    }

    fun logEffectApplied(effectName: String) {
        firebaseAnalytics.logEvent("effect_applied") {
            param("effect_name", effectName)
        }
    }
}
```

✅ App Size Optimization

- [ ] **Enable App Bundle**

```
// build.gradle (app)
android {
    bundle {
        language {
            enableSplit = true
        }
        density {
            enableSplit = true
        }
        abi {
            enableSplit = true
        }
    }
}
```

- [ ] **Use WebP for Images**
- [ ] **Enable Vector Drawables**

```
android {
    defaultConfig {
        vectorDrawables.useSupportLibrary = true
    }
}
```

✅ Testing

- [ ] **Unit Tests for ViewModels**

```
class EditorViewModelTest {

    @get:Rule
    val instantTaskExecutorRule = InstantTaskExecutorRule()

    private lateinit var viewModel: EditorViewModel

    @Before
    fun setup() {
        viewModel = EditorViewModel()
    }

    @Test
    fun `test add clip updates clips list`() {
        // Given
        val testClip = createTestClip()

        // When
        viewModel.addClip(testClip.filePath)

        // Then
```

```
        val clips = viewModel.clips.getOrAwaitValue()
        assertEquals(1, clips.size)
    }

    @Test
    fun `test toggle playback changes state`() {
        // When
        viewModel.togglePlayback()

        // Then
        assertTrue(viewModel.isPlaying.getOrAwaitValue())

        // When
        viewModel.togglePlayback()

        // Then
        assertFalse(viewModel.isPlaying.getOrAwaitValue())
    }
}
```

- [ ] **Instrumented Tests for UI**

```
@RunWith(AndroidJUnit4::class)
class EditorActivityTest {

    @get:Rule
    val activityRule = ActivityScenarioRule(EditorActivity::class.java)

    @Test
    fun testPlayButtonTogglesPlayback() {
        onView(withId(R.id.btn_play)).perform(click())
        onView(withId(R.id.btn_play))
            .check(matches(withContentDescription("Pause")))
    }

    @Test
    fun testImportMediaOpensMediaPicker() {
        onView(withId(R.id.btn_import)).perform(click())
        // Verify media picker opened
        intended(hasAction(Intent.ACTION_PICK))
    }
}
```

✅ Security

- [ ] **Obfuscate Native Libraries**

```
# CMakeLists.txt
if(CMAKE_BUILD_TYPE STREQUAL "Release")
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -fvisibility=hidden")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fvisibility=hidden")
endif()
```

- [ ] **Validate File Inputs**

```
fun validateMediaFile(uri: Uri): Boolean {
    val mimeType = context.contentResolver.getType(uri)

    // Check MIME type
    if (!mimeType?.startsWith("video/") == true &amp;amp;&amp;amp;
        !mimeType?.startsWith("audio/") == true) {
        return false
    }

    // Check file size (max 2GB)
    val fileSize = getFileSize(uri)
    if (fileSize &gt; 2L * 1024 * 1024 * 1024) {
        return false
    }

    // Check file extension
    val fileName = getFileName(uri)
    val validExtensions = listOf("mp4", "mov", "avi", "mkv", "webm")
    val extension = fileName.substringAfterLast('.', "").lowercase()

    return validExtensions.contains(extension)
}
```

✅ Documentation

- [ ] **Add JavaDoc/KDoc Comments**

```
/**
 * Manages video editing operations including clip manipulation,
 * effect application, and export functionality.
 *
 * @property projectId Unique identifier for the current project
 * @property videoEngine Native video processing engine
 */
class EditorViewModel(
    private val projectId: String
) : ViewModel() {

    /**
     * Applies the specified effect to the currently selected clip.
     *
     * @param effect The effect to apply
     * @throws IllegalStateException if no clip is selected
     */
    fun applyEffect(effect: Effect) {
        // Implementation
    }
}
```

- [ ] **Create API Documentation**
- [ ] **Add README to Native Code**

✅ Build Variants

```
android {
    flavorDimensions "version"

    productFlavors {
        free {
            dimension "version"
            applicationIdSuffix ".free"
            versionNameSuffix "-free"

            buildConfigField "boolean", "PREMIUM_FEATURES", "false"
        }

        pro {
            dimension "version"
            applicationIdSuffix ".pro"
            versionNameSuffix "-pro"

            buildConfigField "boolean", "PREMIUM_FEATURES", "true"
        }
    }
}
```

✅ Error Handling

- [ ] **Global Error Handler**

```
class GlobalErrorHandler(
    private val crashlytics: FirebaseCrashlytics,
    private val analytics: AnalyticsManager
) : Thread.UncaughtExceptionHandler {

    private val defaultHandler = Thread.getDefaultUncaughtExceptionHandler()

    override fun uncaughtException(thread: Thread, throwable: Throwable) {
        try {
            // Log to Crashlytics
            crashlytics.recordException(throwable)

            // Log to Analytics
            analytics.logError(throwable.message ?: "Unknown error")

            // Save crash log locally
            saveCrashLog(throwable)

        } finally {
            // Call original handler
            defaultHandler?.uncaughtException(thread, throwable)
        }
    }

    private fun saveCrashLog(throwable: Throwable) {
        val crashFile = File(context.filesDir, "crash_${System.currentTimeMillis()}.log")
```

```
            crashFile.writeText(
                """
                Time: ${Date()}
                Thread: ${Thread.currentThread().name}
                Error: ${throwable.message}
                Stack Trace:
                ${throwable.stackTraceToString()}
                """.trimIndent()
            )
        }
    }
```

Final Production Deployment Steps

1. Version Management

```
// build.gradle (app)
android {
    defaultConfig {
        versionCode 1
        versionName "1.0.0"
    }
}
```

2. Signing Configuration

```
android {
    signingConfigs {
        release {
            storeFile file(RELEASE_STORE_FILE)
            storePassword RELEASE_STORE_PASSWORD
            keyAlias RELEASE_KEY_ALIAS
            keyPassword RELEASE_KEY_PASSWORD
        }
    }

    buildTypes {
        release {
            signingConfig signingConfigs.release
        }
    }
}
```

3. Play Store Listing

- [ ] Create app icon (512x512)
- [ ] Prepare screenshots (phone, tablet, TV)
- [ ] Write app description
- [ ] Add feature graphic (1024x500)

- [ ] Create promo video

4. Privacy Policy

- [ ] Create privacy policy document
- [ ] Host on web server
- [ ] Add link in app settings

5. Google Play Console Setup

- [ ] Complete store listing
- [ ] Set content rating
- [ ] Configure pricing & distribution
- [ ] Upload signed APK/AAB
- [ ] Submit for review

Conclusion

This document covers:

- Critical bug fixes for production stability
- Memory leak prevention
- Thread safety improvements
- Proper error handling
- Complete production readiness checklist

Follow this checklist to ensure ClipForge2 is ready for Play Store deployment!