**Technical Presentation: Advanced C++ Concepts in AOSP**

**(( .cpp file ))**

**#Overview of SurfaceFlinger**

**SurfaceFlinger** is Android’s display compositor, responsible for:

- Composing layers (windows, UI elements, etc.) into a single display image.

- Managing interactions with the Hardware Composer (HWC).

- Handling virtual and physical displays.

- Optimizing rendering via frame scheduling, resource management, and synchronization.

**#Core C++ Syntax and Constructs in SurfaceFlinger.cpp**

1. **Variable Declarations**

( .cpp file )

namespace hal = android::hardware::graphics::composer::hal;

static constexpr int FOUR\_K\_WIDTH = 3840;

static constexpr int FOUR\_K\_HEIGHT = 2160;

constexpr float FALLBACK\_DENSITY = ACONFIGURATION\_DENSITY\_TV;

const float density = property\_get(property, value, nullptr) > 0 ? std::atof(value) : 0.f;

**Explanation:**

- Namespace aliasing: `namespace hal` provides a shorthand for accessing nested namespaces.

- Static constexpr: Compile-time constants for screen resolution.

- Runtime constant: `const float density` uses a system property to calculate the screen density.

**2. Control Statements**

**If Statement Example:**

( .cpp file )

if (!density && required) {

ALOGE("%s must be defined as a build property", property);

return FALLBACK\_DENSITY;

}

- Explanation: Checks if `density` is invalid and logs an error if required.

**For Loop Example:**

( .cpp file )

for (ui::Hdr type : displayHdrTypes) {

if (type != ui::Hdr::DOLBY\_VISION\_4K30) {

outMode.supportedHdrTypes.push\_back(type);

}

}

- Explanation: Filters out unsupported HDR types and appends valid types to `outMode`.

**3. Functions**

**Function Example:**

( .cpp file )

float getDensityFromProperty(const char property, bool required) {

char value[PROPERTY\_VALUE\_MAX];

const float density = property\_get(property, value, nullptr) > 0 ? std::atof(value) : 0.f;

if (!density && required) {

ALOGE("%s must be defined as a build property", property);

return FALLBACK\_DENSITY;

}

return density;

}

- Purpose: Fetches density from system properties with error handling for missing properties.

**4. Lambda Expressions**

( .cpp file )

static\_cast<void>(mScheduler->schedule([=, this]() FTL\_FAKE\_GUARD(kMainThreadContext) {

if (input == nullptr) {

ALOGE("Failed to link to input service");

} else {

mInputFlinger = interface\_cast<os::IInputFlinger>(input);

}

}));

- Explanation:

- Lambda passed as a callback.

- Captures `this` to log errors and initialize `mInputFlinger`.

**Object-Oriented Programming (OOP)**

**1. Classes and Objects**

**SurfaceFlinger Constructor:**

( .cpp file )

SurfaceFlinger::SurfaceFlinger(Factory& factory, SkipInitializationTag)

: mFactory(factory),

mPid(getpid()),

mTimeStats(std::make\_shared<impl::TimeStats>()),

mCompositionEngine(mFactory.createCompositionEngine()) {

ALOGI("Using HWComposer service: %s", mHwcServiceName.c\_str());

}

- Explanation:

- Initializer lists efficiently initialize member variables.

- `mFactory` is passed by reference, avoiding unnecessary copies.

**Destructor Example:**

( .cpp file )

virtual ~DisplayToken() {

Mutex::Autolock \_l(flinger->mStateLock);

flinger->mCurrentState.displays.removeItem(wp<IBinder>::fromExisting(this));

flinger->setTransactionFlags(eDisplayTransactionNeeded);

}

- Explanation: Safely cleans up resources by acquiring a lock and updating shared state.

**2. Inheritance and Polymorphism**

**Virtual Function Overriding:**

( .cpp file )

virtual void onTransact(uint32\_t code, const Parcel& data, Parcel reply, uint32\_t flags) override;

- Purpose: Allows derived classes to customize IPC request handling.

1. **Encapsulation**

**Access Specifiers:**

( .cpp file )

private:

Factory& mFactory;

pid\_t mPid;

sp<TimeStats> mTimeStats;

- Explanation: Private members like `mFactory` ensure controlled access.

**Memory Management**

**1. Smart Pointers**

**Shared Pointer Example:**

( .cpp file )

mTimeStats = std::make\_shared<impl::TimeStats>();

- Purpose: Ensures shared ownership of `TimeStats`.

**Unique Pointer Example:**

( .cpp file )

mPowerAdvisor = std::make\_unique<Hwc2::impl::PowerAdvisor>( this);

- Purpose: Guarantees exclusive ownership and automatic cleanup.

**2. RAII**

**Destructor Example:**

( .cpp file )

class DisplayToken : public BBinder {

virtual ~DisplayToken() {

Mutex::Autolock \_l(flinger->mStateLock);

flinger->mCurrentState.displays.removeItem(wp<IBinder>::fromExisting(this));

}

};

- Purpose: Locks state during cleanup, ensuring thread-safe operations.

**Template Programming**

**1. STL Containers**

**Vector Example:**

( .cpp file )

std::vector<PhysicalDisplayId> SurfaceFlinger::getPhysicalDisplayIdsLocked() const {

std::vector<PhysicalDisplayId> displayIds;

displayIds.reserve(mPhysicalDisplays.size());

const auto defaultDisplayId = getDefaultDisplayDeviceLocked()->getPhysicalId();

displayIds.push\_back(defaultDisplayId);

for (const auto& [id, display] : mPhysicalDisplays) {

if (id != defaultDisplayId) {

displayIds.push\_back(id);

}

}

return displayIds;

}

- Purpose: Dynamically stores and processes display IDs.

**2. Android-Specific Utilities**

ftl::find\\_if Example:

( .cpp file )

return ftl::find\_if(mPhysicalDisplays, PhysicalDisplay::hasToken(displayToken))

.transform(&ftl::to\_key<PhysicalDisplays>);

- Purpose: Efficiently searches and transforms data within a collection.

**Error Handling**

**1. Error Codes**

( .cpp file )

status\_t SurfaceFlinger::destroyVirtualDisplay(const sp<IBinder>& displayToken) {

const ssize\_t index = mCurrentState.displays.indexOfKey(displayToken);

if (index < 0) {

return NAME\_NOT\_FOUND;

}

return NO\_ERROR;

}

- Explanation:

- `NAME\_NOT\_FOUND` and `NO\_ERROR` indicate operation success or failure.

**Advanced C++ Patterns**

**Singleton Example**

( .cpp file )

class FlagManager {

public:

static FlagManager& getInstance() {

static FlagManager instance;

return instance;

}

private:

FlagManager() = default;

};

- Purpose: Ensures a single instance of `FlagManager` is created globally.

**Preprocessor Directives**

**Macros**

( .cpp file )

define ATRACE\_TAG ATRACE\_TAG\_GRAPHICS

- Purpose: Defines constants for tracing tags.

Conditional Compilation

( .cpp file )

ifdef DEBUG

define LOG\_LEVEL\_VERBOSE

else

define LOG\_LEVEL\_ERROR

endif

- Purpose : Enables debug-specific logging based on compile-time flags.

**Visual Aids**

- Class Diagram: Relationships between `SurfaceFlinger`, `DisplayDevice`, `HardwareComposer`.

A diagram of a diagram

Description automatically generated

- Flowchart: Data flow from application layers to physical displays.

A diagram of a computer

Description automatically generated

**Summary**

- SurfaceFlinger showcases advanced C++ concepts like OOP, smart pointers, and templates.

- It integrates these principles into Android’s rendering pipeline for efficient resource management and display composition.