OpenHarmonyTestRunner.ts

import hilog from '@ohos.hilog';  
import TestRunner from '@ohos.application.testRunner';  
import AbilityDelegatorRegistry from '@ohos.app.ability.abilityDelegatorRegistry';  
var abilityDelegator = undefined  
var abilityDelegatorArguments = undefined  
async function onAbilityCreateCallback() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'onAbilityCreateCallback');  
}  
async function addAbilityMonitorCallback(err: any) {  
 hilog.info(0x0000, 'testTag', 'addAbilityMonitorCallback : %{public}s', JSON.stringify(err) ?? '');  
}  
export default class OpenHarmonyTestRunner implements TestRunner {  
 constructor() {  
 }  
 onPrepare() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'OpenHarmonyTestRunner OnPrepare ');  
 }  
 async onRun() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'OpenHarmonyTestRunner onRun run');  
 abilityDelegatorArguments = AbilityDelegatorRegistry.getArguments()  
 abilityDelegator = AbilityDelegatorRegistry.getAbilityDelegator()  
 var testAbilityName = abilityDelegatorArguments.bundleName + '.TestAbility'  
 let lMonitor = {  
 abilityName: testAbilityName,  
 onAbilityCreate: onAbilityCreateCallback,  
 };  
 abilityDelegator.addAbilityMonitor(lMonitor, addAbilityMonitorCallback)  
 var cmd = 'aa start -d 0 -a TestAbility' + ' -b ' + abilityDelegatorArguments.bundleName  
 var debug = abilityDelegatorArguments.parameters['-D']  
 if (debug == 'true')  
 {  
 cmd += ' -D'  
 }  
 hilog.info(0x0000, 'testTag', 'cmd : %{public}s', cmd);  
 abilityDelegator.executeShellCommand(cmd,  
 (err: any, d: any) => {  
 hilog.info(0x0000, 'testTag', 'executeShellCommand : err : %{public}s', JSON.stringify(err) ?? '');  
 hilog.info(0x0000, 'testTag', 'executeShellCommand : data : %{public}s', d.stdResult ?? '');  
 hilog.info(0x0000, 'testTag', 'executeShellCommand : data : %{public}s', d.exitCode ?? '');  
 })  
 hilog.info(0x0000, 'testTag', '%{public}s', 'OpenHarmonyTestRunner onRun end');  
 }  
}

List.test.ets

import abilityTest from './Ability.test'  
export default function testsuite() {  
 abilityTest()  
}

Ability.test.ets

import hilog from '@ohos.hilog';  
import { describe, beforeAll, beforeEach, afterEach, afterAll, it, expect } from '@ohos/hypium'  
export default function abilityTest() {  
 describe('ActsAbilityTest', function () {  
 beforeAll(function () {  
 })  
 beforeEach(function () {  
 })  
 afterEach(function () {  
 })  
 afterAll(function () {  
 })  
 it('assertContain',0, function () {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'it begin');  
 let a = 'abc'  
 let b = 'b'  
 expect(a).assertContain(b)  
 expect(a).assertEqual(a)  
 })  
 })  
}

TestAbility.ets

import UIAbility from '@ohos.app.ability.UIAbility';  
import AbilityDelegatorRegistry from '@ohos.app.ability.abilityDelegatorRegistry';  
import hilog from '@ohos.hilog';  
import { Hypium } from '@ohos/hypium';  
import testsuite from '../test/List.test';  
import window from '@ohos.window';  
export default class TestAbility extends UIAbility {  
 onCreate(want, launchParam) {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onCreate');  
 hilog.info(0x0000, 'testTag', '%{public}s', 'want param:' + JSON.stringify(want) ?? '');  
 hilog.info(0x0000, 'testTag', '%{public}s', 'launchParam:'+ JSON.stringify(launchParam) ?? '');  
 var abilityDelegator: any  
 abilityDelegator = AbilityDelegatorRegistry.getAbilityDelegator()  
 var abilityDelegatorArguments: any  
 abilityDelegatorArguments = AbilityDelegatorRegistry.getArguments()  
 hilog.info(0x0000, 'testTag', '%{public}s', 'start run testcase!!!');  
 Hypium.hypiumTest(abilityDelegator, abilityDelegatorArguments, testsuite)  
 }  
 onDestroy() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onDestroy');  
 }  
 onWindowStageCreate(windowStage: window.WindowStage) {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onWindowStageCreate');  
 windowStage.loadContent('testability/pages/Index', (err, data) => {  
 if (err.code) {  
 hilog.error(0x0000, 'testTag', 'Failed to load the content. Cause: %{public}s', JSON.stringify(err) ?? '');  
 return;  
 }  
 hilog.info(0x0000, 'testTag', 'Succeeded in loading the content. Data: %{public}s',  
 JSON.stringify(data) ?? '');  
 });  
 }  
 onWindowStageDestroy() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onWindowStageDestroy');  
 }  
 onForeground() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onForeground');  
 }  
 onBackground() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onBackground');  
 }  
}

Index.ets

import hilog from '@ohos.hilog';  
@Entry  
@Component  
struct Index {  
 aboutToAppear() {  
 hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility index aboutToAppear');  
 }  
 @State message: string = 'Hello World'  
 build() {  
 Row() {  
 Column() {  
 Text(this.message)  
 .fontSize(50)  
 .fontWeight(FontWeight.Bold)  
 Button() {  
 Text('next page')  
 .fontSize(20)  
 .fontWeight(FontWeight.Bold)  
 }.type(ButtonType.Capsule)  
 .margin({  
 top: 20  
 })  
 .backgroundColor('#0D9FFB')  
 .width('35%')  
 .height('5%')  
 .onClick(()=>{  
 })  
 }  
 .width('100%')  
 }  
 .height('100%')  
 }  
 }

index.d.ts

export const add: (a: number, b: number) => number;  
export const drawRectangle:()=> number;  
export const loadYuv:(file: string)=> number;  
export const drawLine:()=> number;  
export const generate\_x509\_certificate: (a: string, b: string) => number;  
export const verify\_signature: (a: Uint8Array, b: Uint8Array, c: Uint8Array) => boolean;  
export const sign\_message: (message: Uint8Array, key: Uint8Array,) => Uint8Array;  
export const decrypt: (message: Uint8Array, key: Uint8Array,) => Uint8Array;  
export const encrypt: (message: Uint8Array, key: Uint8Array,) => Uint8Array;  
export const openSlEsPlayer\_sendPcmData: (message: Uint8Array) => void;  
export declare class CurlClient {  
 close()  
 get(url: string, timeout: Number, client: string, key: string): Promise<Uint8Array>;  
}  
export declare class VideoStatus{  
 decoder: string  
 totalFps: Number  
 receivedFps: Number  
 decodedFps: Number  
 renderedFps: Number  
 networkDroppedRate: number  
 networkDroppedFrames: number  
 decodeTime: number  
 receivedTime: number  
}  
export declare class MoonBridgeNapi {  
 onClStage(key:string, callback:(stage:string)=> void)  
 onClStageFailed(key:string, callback:(stage:string, code: number)=> void)  
 onClConnection(key:string, callback:(code:number)=> void)  
 onVideoStatus(callback:(any:VideoStatus)=> void)  
 startConnection(  
 address: string, appVersion: string, gfeVersion: string,  
 rtspSessionUrl: string, serverCodecModeSupport: number,  
 width: number, height: number, fps: number,  
 bitrate: number, packetSize: number, streamingRemotely: number,  
 audioConfiguration: number, supportedVideoFormats: number,  
 clientRefreshRateX100: number,  
 encryptionFlags: number,  
 riAesKey: Uint8Array, riAesIv: Uint8Array,  
 videoCapabilities: number,  
 colorSpace: number, colorRange: number  
 ): number;  
 stopConnection(): void;  
 sendMultiControllerInput(  
 controllerNumber: number,  
 activeGamepadMask: number,  
 buttonFlags: number,  
 leftTrigger: number,  
 rightTrigger: number,  
 leftStickX: number,  
 leftStickY: number,  
 rightStickX: number,  
 rightStickY: number  
 ): void;  
 static interruptConnection(): void;  
 static sendMouseMove(deltaX: number, deltaY: number): void;  
 static sendMousePosition(x: number, y: number, referenceWidth: number, referenceHeight: number): void;  
 static sendMouseMoveAsMousePosition(deltaX: number, deltaY: number, referenceWidth: number, referenceHeight: number): void;  
 static sendMouseButton(buttonEvent: number, mouseButton: number): void;  
 static sendTouchEvent(  
 eventType: number, pointerId: number, x: number, y: number, pressure: number,  
 contactAreaMajor: number, contactAreaMinor: number, rotation: number  
 ): number;  
 static sendPenEvent(  
 eventType: number, toolType: number, penButtons: number, x: number, y: number,  
 pressure: number, contactAreaMajor: number, contactAreaMinor: number,  
 rotation: number, tilt: number  
 ): number;  
 sendControllerArrivalEvent(  
 controllerNumber: number, activeGamepadMask: number, type: number, supportedButtonFlags: number, capabilities: number  
 ): number;  
 static sendControllerTouchEvent(  
 controllerNumber: number, eventType: number, pointerId: number, x: number, y: number, pressure: number  
 ): number;  
 static sendControllerMotionEvent(controllerNumber: number, motionType: number, x: number, y: number, z: number): number;  
 static sendControllerBatteryEvent(controllerNumber: number, batteryState: number, batteryPercentage: number): number;  
 static sendKeyboardInput(keyMap: number, keyDirection: number, modifier: number, flags: number): void;  
 static sendMouseHighResScroll(scrollAmount: number): void;  
 static sendMouseHighResHScroll(scrollAmount: number): void;  
 static sendUtf8Text(text: string): void;  
 static getStageName(stage: number): string;  
 static findExternalAddressIP4(stunHostName: string, stunPort: number): string;  
 static getPendingAudioDuration(): number;  
 static getPendingVideoFrames(): number;  
 static testClientConnectivity(testServerHostName: string, referencePort: number, testFlags: number): number;  
 static getPortFlagsFromStage(stage: number): number;  
 static getPortFlagsFromTerminationErrorCode(errorCode: number): number;  
 static stringifyPortFlags(portFlags: number, separator: string): string;  
 static getEstimatedRttInfo(): bigint;  
 static guessControllerType(vendorId: number, productId: number): number;  
 static guessControllerHasPaddles(vendorId: number, productId: number): boolean;  
 static guessControllerHasShareButton(vendorId: number, productId: number): boolean;  
 static init(): void;  
}

NativeVideoDecoder.cpp

#include "NativeVideoDecoder.h"  
#include <stdarg.h>  
#include <hilog/log.h>  
#include <multimedia/player\_framework/native\_avcodec\_videodecoder.h>  
#define DECODER\_BUFFER\_SIZE 92 \* 1024 \* 2  
void decodeLog(const char \*format, ...) {  
 va\_list va;  
 va\_start(va, format);  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "NativeVideoDecoder", format, va);  
 va\_end(va);  
}  
NativeVideoDecoder::NativeVideoDecoder() {}  
NativeVideoDecoder::~NativeVideoDecoder() {}  
static void OnError(OH\_AVCodec \*codec, int32\_t errorCode, void \*userData) {  
 (void)codec;  
 (void)errorCode;  
 (void)userData;  
 decodeLog("Error received, errorCode: %{public}d", errorCode);  
}  
static void OnOutputFormatChanged(OH\_AVCodec \*codec, OH\_AVFormat \*format, void \*userData) {  
 (void)codec;  
 (void)format;  
 (void)userData;  
 decodeLog("OnOutputFormatChanged received");  
}  
static void OnInputBufferAvailable(OH\_AVCodec \*codec, uint32\_t index, OH\_AVMemory \*data, void \*userData) {  
 (void)codec;  
 VDecSignal \*signal\_ = static\_cast<VDecSignal \*>(userData);  
 std::unique\_lock<std::mutex> lock(signal\_->inMutex\_);  
 signal\_->inQueue\_.push(index);  
 signal\_->inBufferQueue\_.push(data);  
 signal\_->inCond\_.notify\_all();  
}  
static void OnOutputBufferAvailable(OH\_AVCodec \*codec, uint32\_t index, OH\_AVMemory \*data, OH\_AVCodecBufferAttr \*attr,  
 void \*userData) {  
 (void)codec;  
 VDecSignal \*signal\_ = static\_cast<VDecSignal \*>(userData);  
 if (attr) {  
 decodeLog("OnOutputBufferAvailable received, index: %{public}d, attr->size: %{public}d", index, attr->size);  
 std::unique\_lock<std::mutex> lock(signal\_->outMutex\_);  
 signal\_->outQueue\_.push(index);  
 signal\_->outBufferQueue\_.push(data);  
 signal\_->attrQueue\_.push(\*attr);  
 signal\_->outCond\_.notify\_all();  
 } else {  
 decodeLog("OnOutputBufferAvailable error, attr is nullptr!");  
 }  
}  
int NativeVideoDecoder::setup(DECODER\_PARAMETERS params) {  
 m\_stream\_fps = params.frame\_rate;  
 decodeLog(  
 "Setup with format: %{public}s, width: %{public}d, height: %{public}d, fps: %{public}d",  
 params.video\_format == VIDEO\_FORMAT\_H264 ? "H264" : "HEVC",  
 params.width, params.height,  
 params.frame\_rate);  
 switch (params.video\_format) {  
 case VIDEO\_FORMAT\_H264:  
 decodeLog(" find decoder 264");  
 m\_decoder = OH\_VideoDecoder\_CreateByMime(OH\_AVCODEC\_MIMETYPE\_VIDEO\_AVC);  
 break;  
 case VIDEO\_FORMAT\_H265:  
 decodeLog(" find decoder HEVC");  
 m\_decoder = OH\_VideoDecoder\_CreateByMime(OH\_AVCODEC\_MIMETYPE\_VIDEO\_AVC);  
 break;  
 }  
 if (m\_decoder == NULL) {  
 decodeLog(" Couldn't find decoder");  
 return -1;  
 }  
 m\_signal = new VDecSignal();  
 OH\_AVFormat \*format = OH\_AVFormat\_Create();  
 OH\_AVFormat\_SetIntValue(format, OH\_MD\_KEY\_WIDTH, params->width);  
 OH\_AVFormat\_SetIntValue(format, OH\_MD\_KEY\_HEIGHT, params->height);  
 OH\_AVFormat\_SetIntValue(format, OH\_MD\_KEY\_PIXEL\_FORMAT, AV\_PIXEL\_FORMAT\_NV21);  
 int err = OH\_VideoDecoder\_Configure(m\_decoder, format);  
 OH\_AVFormat\_Destroy(format);  
 OH\_AVCodecAsyncCallback callback = {  
 .onNeedInputData = OnInputBufferAvailable,  
 .onNeedOutputData = OnOutputBufferAvailable};  
 OH\_VideoDecoder\_SetCallback(m\_decoder, callback, m\_signal);  
 if (params->context != nullptr) {  
 OHNativeWindow \*window = static\_cast<OHNativeWindow \*>(params->context);  
 OH\_VideoDecoder\_SetSurface(m\_decoder, window);  
 } else {  
 decodeLog(" Couldn't find set surface");  
 }  
 bool isSurfaceMode = true;  
 OH\_AVCodecBufferAttr info;  
 return DR\_OK;  
}  
void NativeVideoDecoder::start() {  
 m\_is\_running.store(true);  
 m\_inputLoop = std::make\_unique<std::thread>(&NativeVideoDecoder::inputFunc, this);  
 m\_outputLoop = std::make\_unique<std::thread>(&NativeVideoDecoder::outputFunc, this);  
 OH\_VideoDecoder\_Start(m\_decoder);  
}  
void NativeVideoDecoder::stop() {  
 m\_is\_running.store(false);  
 if (m\_inputLoop != nullptr && m\_inputLoop->joinable()) {  
 std::unique\_lock<std::mutex> lock(m\_signal->inMutex\_);  
 m\_signal->inCond\_.notify\_all();  
 lock.unlock();  
 m\_inputLoop->join();  
 }  
 if (m\_outputLoop != nullptr && m\_outputLoop->joinable()) {  
 std::unique\_lock<std::mutex> lock(m\_signal->outMutex\_);  
 m\_signal->outCond\_.notify\_all();  
 lock.unlock();  
 m\_outputLoop->join();  
 }  
 decodeLog("start stop!");  
 OH\_VideoDecoder\_Stop(m\_decoder);  
}  
void flush() {  
}  
void NativeVideoDecoder::cleanup() {  
 OH\_VideoDecoder\_Destroy(m\_decoder);  
}  
VIDEO\_STATS \*NativeVideoDecoder::video\_decode\_stats() {  
 return nullptr;  
}  
int NativeVideoDecoder::ExtractPacket() {  
 m\_pkt = m\_signal->dataPacketQueue\_.front();  
 m\_signal->dataPacketQueue\_.pop();  
 return 0;  
}  
void NativeVideoDecoder::inputFunc() {  
 while (true) {  
 if (!m\_is\_running.load()) {  
 break;  
 }  
 std::unique\_lock<std::mutex> lock(m\_signal->inMutex\_);  
 m\_signal->inCond\_.wait(lock, [this]() { return (m\_signal->inQueue\_.size() > 0 || !m\_is\_running.load()); });  
 if (!m\_is\_running.load()) {  
 break;  
 }  
 uint32\_t index = m\_signal->inQueue\_.front();  
 auto buffer = m\_signal->inBufferQueue\_.front();  
 lock.unlock();  
 if ((ExtractPacket() != AV\_ERR\_OK)) {  
 continue;  
 }  
 OH\_AVCodecBufferAttr info;  
 info.size = m\_pkt->size;  
 info.offset = 0;  
 info.pts = m\_pkt->pts;  
 if (buffer == nullptr) {  
 decodeLog("Fatal: GetInputBuffer fail");  
 }  
 memcpy(OH\_AVMemory\_GetAddr(buffer), m\_pkt->data, m\_pkt->size);  
 int32\_t ret = 0;  
 if (m\_isFirst\_frame) {  
 info.flags = AVCODEC\_BUFFER\_FLAGS\_SYNC\_FRAME;  
 ret = OH\_VideoDecoder\_PushInputData(m\_decoder, index, info);  
 m\_isFirst\_frame = false;  
 } else {  
 info.flags = AVCODEC\_BUFFER\_FLAGS\_NONE;  
 ret = OH\_VideoDecoder\_PushInputData(m\_decoder, index, info);  
 }  
 if (ret != AV\_ERR\_OK) {  
 decodeLog("Fatal error, exit");  
 break;  
 }  
 lock.lock();  
 m\_signal->inQueue\_.pop();  
 m\_signal->inBufferQueue\_.pop();  
 }  
}  
void NativeVideoDecoder::outputFunc() {  
 while (true) {  
 if (!m\_is\_running.load()) {  
 decodeLog("stop, exit");  
 break;  
 }  
 std::unique\_lock<std::mutex> lock(m\_signal->outMutex\_);  
 m\_signal->outCond\_.wait(lock, [this]() { return (m\_signal->outQueue\_.size() > 0 || !m\_is\_running.load()); });  
 if (!m\_is\_running.load()) {  
 decodeLog("wait to stop, exit");  
 break;  
 }  
 uint32\_t index = m\_signal->outQueue\_.front();  
 OH\_AVCodecBufferAttr attr = m\_signal->attrQueue\_.front();  
 OH\_AVMemory \*data = m\_signal->outBufferQueue\_.front();  
 lock.unlock();  
 if (attr.flags == AVCODEC\_BUFFER\_FLAGS\_EOS) {  
 decodeLog("decode eos, write frame: ${public}d");  
 m\_is\_running.store(false);  
 }  
 if (OH\_VideoDecoder\_RenderOutputData(m\_decoder, index) != AV\_ERR\_OK) {  
 decodeLog("Fatal: RenderOutputData fail");  
 break;  
 }  
 lock.lock();  
 m\_signal->outBufferQueue\_.pop();  
 m\_signal->attrQueue\_.pop();  
 m\_signal->outQueue\_.pop();  
 }  
}  
int NativeVideoDecoder::submitDecodeUnit(PDECODE\_UNIT du) {  
 if (m\_frames\_in == 0 && du->frameType != FRAME\_TYPE\_IDR) {  
 return DR\_NEED\_IDR;  
 }  
 if (du->fullLength < DECODER\_BUFFER\_SIZE) {  
 PLENTRY entry = du->bufferList;  
 if (!m\_last\_frame) {  
 m\_video\_decode\_stats.measurementStartTimestamp = LiGetMillis();  
 m\_last\_frame = du->frameNumber;  
 } else {  
 m\_video\_decode\_stats.networkDroppedFrames +=  
 du->frameNumber - (m\_last\_frame + 1);  
 m\_video\_decode\_stats.totalFrames +=  
 du->frameNumber - (m\_last\_frame + 1);  
 m\_last\_frame = du->frameNumber;  
 }  
 m\_video\_decode\_stats.receivedFrames++;  
 m\_video\_decode\_stats.totalFrames++;  
 int length = 0;  
 while (entry != NULL) {  
 if (length > DECODER\_BUFFER\_SIZE) {  
 decodeLog("FFmpeg: Big buffer to decode... !");  
 }  
 memcpy(m\_ffmpeg\_buffer + length, entry->data, entry->length);  
 length += entry->length;  
 entry = entry->next;  
 }  
 m\_video\_decode\_stats.totalReassemblyTime +=  
 LiGetMillis() - du->receiveTimeMs;  
 m\_frames\_in++;  
 uint64\_t before\_decode = LiGetMillis();  
 if (length > DECODER\_BUFFER\_SIZE) {  
 decodeLog("FFmpeg: Big buffer to decode...");  
 }  
 DataPacket \*pkt = {};  
 pkt->data = (uint8\_t \*)m\_ffmpeg\_buffer;  
 pkt->size = length;  
 if (du->frameType == FRAME\_TYPE\_IDR) {  
 pkt->flags = AVCODEC\_BUFFER\_FLAGS\_INCOMPLETE\_FRAME;  
 } else {  
 pkt->flags = 0;  
 }  
 m\_signal->dataPacketQueue\_.push(pkt);  
 m\_frames\_out++;  
 m\_video\_decode\_stats.totalDecodeTime +=  
 LiGetMillis() - before\_decode;  
 m\_video\_decode\_stats.totalDecodeTime +=  
 (m\_frames\_in - m\_frames\_out) \* (1000 / m\_stream\_fps);  
 m\_video\_decode\_stats.decodedFrames++;  
 } else {  
 decodeLog("FFmpeg: Big buffer to decode... 2");  
 }  
 return DR\_OK;  
}

plugin\_render.cpp

#include <stdint.h>  
#include <string>  
#include <js\_native\_api.h>  
#include <js\_native\_api\_types.h>  
#include <hilog/log.h>  
#include "moon\_bridge.h"  
#include "plugin\_render.h"  
std::unordered\_map<std::string, PluginRender \*> PluginRender::m\_instance;  
OH\_NativeXComponent\_Callback PluginRender::m\_callback;  
void OnSurfaceCreatedCB(OH\_NativeXComponent \*component, void \*window)  
{  
 MoonBridgeApi::api->nativewindow = window;  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "Callback", "OnSurfaceCreatedCB");  
 if ((nullptr == component) || (nullptr == window)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnSurfaceCreatedCB: component or window is null");  
 return;  
 }  
 char idStr[OH\_XCOMPONENT\_ID\_LEN\_MAX + 1] = { '\0' };  
 uint64\_t idSize = OH\_XCOMPONENT\_ID\_LEN\_MAX + 1;  
 if (OH\_NATIVEXCOMPONENT\_RESULT\_SUCCESS != OH\_NativeXComponent\_GetXComponentId(component, idStr, &idSize)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnSurfaceCreatedCB: Unable to get XComponent id");  
 return;  
 }  
 std::string id(idStr);  
 auto render = PluginRender::GetInstance(id);  
 uint64\_t width;  
 uint64\_t height;  
 int32\_t xSize = OH\_NativeXComponent\_GetXComponentSize(component, window, &width, &height);  
 if ((OH\_NATIVEXCOMPONENT\_RESULT\_SUCCESS == xSize) && (nullptr != render)) {  
 DECODER\_PARAMETERS params;  
 params.context = window;  
 params.width = 1280;  
 params.height = 720;  
 }  
}  
void OnSurfaceChangedCB(OH\_NativeXComponent \*component, void \*window)  
{  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "Callback", "OnSurfaceChangedCB");  
 if ((nullptr == component) || (nullptr == window)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnSurfaceChangedCB: component or window is null");  
 return;  
 }  
 char idStr[OH\_XCOMPONENT\_ID\_LEN\_MAX + 1] = { '\0' };  
 uint64\_t idSize = OH\_XCOMPONENT\_ID\_LEN\_MAX + 1;  
 if (OH\_NATIVEXCOMPONENT\_RESULT\_SUCCESS != OH\_NativeXComponent\_GetXComponentId(component, idStr, &idSize)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnSurfaceChangedCB: Unable to get XComponent id");  
 return;  
 }  
 std::string id(idStr);  
 auto render = PluginRender::GetInstance(id);  
 if (nullptr != render) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "Callback", "surface changed");  
 }  
}  
void OnSurfaceDestroyedCB(OH\_NativeXComponent \*component, void \*window)  
{  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "Callback", "OnSurfaceDestroyedCB");  
 if ((nullptr == component) || (nullptr == window)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnSurfaceDestroyedCB: component or window is null");  
 return;  
 }  
 char idStr[OH\_XCOMPONENT\_ID\_LEN\_MAX + 1] = { '\0' };  
 uint64\_t idSize = OH\_XCOMPONENT\_ID\_LEN\_MAX + 1;  
 if (OH\_NATIVEXCOMPONENT\_RESULT\_SUCCESS != OH\_NativeXComponent\_GetXComponentId(component, idStr, &idSize)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnSurfaceDestroyedCB: Unable to get XComponent id");  
 return;  
 }  
 std::string id(idStr);  
 PluginRender::Release(id);  
}  
void DispatchTouchEventCB(OH\_NativeXComponent \*component, void \*window)  
{  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "Callback", "DispatchTouchEventCB");  
 if ((nullptr == component) || (nullptr == window)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "DispatchTouchEventCB: component or window is null");  
 return;  
 }  
 uint64\_t width, height;  
 OH\_NativeXComponent\_GetXComponentSize(component, window, &width, &height);  
 OH\_NativeXComponent\_TouchEvent touchEvent;  
 OH\_NativeXComponent\_GetTouchEvent(component, window, &touchEvent);  
 MoonBridge\_sendTouchEvent(touchEvent, width, height);  
}  
void OnMouseEventCB(OH\_NativeXComponent \*component, void \*window)  
{  
 if ((nullptr == component) || (nullptr == window)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "DispatchTouchEventCB: component or window is null");  
 return;  
 }  
 uint64\_t width, height;  
 OH\_NativeXComponent\_GetXComponentSize(component, window, &width, &height);  
 OH\_NativeXComponent\_MouseEvent touchEvent;  
 OH\_NativeXComponent\_GetMouseEvent(component, window, &touchEvent);  
 MoonBridge\_sendMouseEvent(touchEvent, width, height);  
}  
void OnHoverEventCB(OH\_NativeXComponent \*component, bool isHover)  
{  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "Callback", "OnHoverEventCB");  
 if ((nullptr == component)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "Callback",  
 "OnHoverEventCB: component or window is null");  
 return;  
 }  
}  
PluginRender::PluginRender(std::string &id)  
{  
 this->m\_id = id;  
 this->m\_eglCore = new EglVideoRenderer();  
 OH\_NativeXComponent\_Callback \*renderCallback = &PluginRender::m\_callback;  
 renderCallback->OnSurfaceCreated = OnSurfaceCreatedCB;  
 renderCallback->OnSurfaceChanged = OnSurfaceChangedCB;  
 renderCallback->OnSurfaceDestroyed = OnSurfaceDestroyedCB;  
 renderCallback->DispatchTouchEvent = DispatchTouchEventCB;  
}  
PluginRender \*PluginRender::GetInstance(std::string &id)  
{  
 if (m\_instance.find(id) == m\_instance.end()) {  
 PluginRender \*instance = new PluginRender(id);  
 m\_instance[id] = instance;  
 return instance;  
 } else {  
 return m\_instance[id];  
 }  
}  
void PluginRender::Export(napi\_env env, napi\_value exports)  
{  
   
 if ((nullptr == env) || (nullptr == exports)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender", "Export: env or exports is null");  
 return;  
 }  
 napi\_property\_descriptor desc[] = {  
 { "drawRectangle", nullptr, PluginRender::NapiDrawRectangle, nullptr, nullptr, nullptr, napi\_default, nullptr }  
 };  
 if (napi\_ok != napi\_define\_properties(env, exports, sizeof(desc) / sizeof(desc[0]), desc)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender", "Export: napi\_define\_properties failed");  
 }  
}  
napi\_value PluginRender::NapiDrawRectangle(napi\_env env, napi\_callback\_info info)  
{  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "PluginRender", "NapiDrawRectangle");  
 if ((nullptr == env) || (nullptr == info)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender", "NapiDrawRectangle: env or info is null");  
 return nullptr;  
 }  
 napi\_value thisArg;  
 if (napi\_ok != napi\_get\_cb\_info(env, info, nullptr, nullptr, &thisArg, nullptr)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender", "NapiDrawRectangle: napi\_get\_cb\_info fail");  
 return nullptr;  
 }  
 napi\_value exportInstance;  
 if (napi\_ok != napi\_get\_named\_property(env, thisArg, OH\_NATIVE\_XCOMPONENT\_OBJ, &exportInstance)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender",  
 "NapiDrawRectangle: napi\_get\_named\_property fail");  
 return nullptr;  
 }  
 OH\_NativeXComponent \*nativeXComponent = nullptr;  
 if (napi\_ok != napi\_unwrap(env, exportInstance, reinterpret\_cast<void \*\*>(&nativeXComponent))) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender", "NapiDrawRectangle: napi\_unwrap fail");  
 return nullptr;  
 }  
 char idStr[OH\_XCOMPONENT\_ID\_LEN\_MAX + 1] = { '\0' };  
 uint64\_t idSize = OH\_XCOMPONENT\_ID\_LEN\_MAX + 1;  
 if (OH\_NATIVEXCOMPONENT\_RESULT\_SUCCESS != OH\_NativeXComponent\_GetXComponentId(nativeXComponent, idStr, &idSize)) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_ERROR, LOG\_DOMAIN, "PluginRender",  
 "NapiDrawRectangle: Unable to get XComponent id");  
 return nullptr;  
 }  
 std::string id(idStr);  
 PluginRender \*render = PluginRender::GetInstance(id);  
 if (render) {  
 OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "PluginRender", "render->m\_eglCore->Draw() executed");  
 }  
 return nullptr;  
}  
void PluginRender::Release(std::string &id)  
{  
 PluginRender \*render = PluginRender::GetInstance(id);  
 if (nullptr != render) {  
 render->m\_eglCore->Release();  
 delete render->m\_eglCore;  
 render->m\_eglCore = nullptr;  
 delete render;  
 render = nullptr;  
 m\_instance.erase(m\_instance.find(id));  
 }  
}

eglRender.cpp

#include "video/render/eglRender.h"  
#include "hilog/log.h"  
#include "video/common/common.h"  
#include "Shader.h"  
#include <multimedia/player\_framework/native\_avcodec\_videodecoder.h>  
#define eglLog(level, ...) OH\_LOG\_Print(LOG\_APP, level, LOG\_DOMAIN, "EglCore", \_\_VA\_ARGS\_\_)  
static const char \*fragYUV420P =  
 "#version 300 es\n"  
 "precision mediump float;\n"  
 "//纹理坐标\n"  
 "in vec2 vTextCoord;\n"  
 "//输入的yuv三个纹理\n"  
 "uniform sampler2D yTexture;//采样器\n"  
 "uniform sampler2D uTexture;//采样器\n"  
 "uniform sampler2D vTexture;//采样器\n"  
 "out vec4 FragColor;\n"  
 "void main() {\n"  
 "//采样到的yuv向量数据\n"  
 " vec3 yuv;\n"  
 "//yuv转化得到的rgb向量数据\n"  
 " vec3 rgb;\n"  
 " //分别取yuv各个分量的采样纹理\n"  
 " yuv.x = texture(yTexture, vTextCoord).r;\n"  
 " yuv.y = texture(uTexture, vTextCoord).g - 0.5;\n"  
 " yuv.z = texture(vTexture, vTextCoord).b - 0.5;\n"  
 " rgb = mat3(\n"  
 " 1.0, 1.0, 1.0,\n"  
 " 0.0, -0.183, 1.816,\n"  
 " 1.540, -0.459, 0.0\n"  
 " ) \* yuv;\n"  
 " //gl\_FragColor是OpenGL内置的\n"  
 " FragColor = vec4(rgb, 1.0);\n"  
 " }";  
static const char \*vertexShaderWithMatrix =  
 " #version 300 es\n"  
 " layout (location = 0) \n"  
 " in vec4 aPosition;//输入的顶点坐标，会在程序指定将数据输入到该字段\n"//如果传入的向量是不够4维的，自动将前三个分量设置为0.0，最后一个分量设置为1.0  
 " layout (location = 1) \n"  
 " in vec2 aTextCoord;//输入的纹理坐标，会在程序指定将数据输入到该字段\n"  
 "\n"  
 " out\n"  
 " vec2 vTextCoord;//输出的纹理坐标;\n"  
 " uniform mat4 uMatrix;"//变换矩阵  
 "\n"  
 " void main() {\n"  
 " //这里其实是将上下翻转过来（因为安卓图片会自动上下翻转，所以转回来）\n"  
 " vTextCoord = vec2(aTextCoord.x, 1.0 - aTextCoord.y);\n"  
 " //直接把传入的坐标值作为传入渲染管线。gl\_Position是OpenGL内置的\n"  
 " gl\_Position = aPosition;\n"  
 " }";  
static const char \*texture\_mappings[] = {"ymap", "umap", "vmap"};  
static const float vertices[] = {-1.0f, -1.0f, 1.0f, -1.0f,  
 -1.0f, 1.0f, 1.0f, 1.0f};  
static const float \*gl\_color\_offset(bool color\_full) {  
 static const float limitedOffsets[] = {16.0f / 255.0f, 128.0f / 255.0f,  
 128.0f / 255.0f};  
 static const float fullOffsets[] = {0.0f, 128.0f / 255.0f, 128.0f / 255.0f};  
 return color\_full ? fullOffsets : limitedOffsets;  
}  
static const float \*gl\_color\_matrix(enum AVColorSpace color\_space,  
 bool color\_full) {  
 static const float bt601Lim[] = {1.1644f, 1.1644f, 1.1644f, 0.0f, -0.3917f,  
 2.0172f, 1.5960f, -0.8129f, 0.0f};  
 static const float bt601Full[] = {  
 1.0f, 1.0f, 1.0f, 0.0f, -0.3441f, 1.7720f, 1.4020f, -0.7141f, 0.0f};  
 static const float bt709Lim[] = {1.1644f, 1.1644f, 1.1644f, 0.0f, -0.2132f,  
 2.1124f, 1.7927f, -0.5329f, 0.0f};  
 static const float bt709Full[] = {  
 1.0f, 1.0f, 1.0f, 0.0f, -0.1873f, 1.8556f, 1.5748f, -0.4681f, 0.0f};  
 static const float bt2020Lim[] = {1.1644f, 1.1644f, 1.1644f,  
 0.0f, -0.1874f, 2.1418f,  
 1.6781f, -0.6505f, 0.0f};  
 static const float bt2020Full[] = {  
 1.0f, 1.0f, 1.0f, 0.0f, -0.1646f, 1.8814f, 1.4746f, -0.5714f, 0.0f};  
 switch (color\_space) {  
 case AVCOL\_SPC\_SMPTE170M:  
 case AVCOL\_SPC\_BT470BG:  
 return color\_full ? bt601Full : bt601Lim;  
 case AVCOL\_SPC\_BT709:  
 return color\_full ? bt709Full : bt709Lim;  
 case AVCOL\_SPC\_BT2020\_NCL:  
 case AVCOL\_SPC\_BT2020\_CL:  
 return color\_full ? bt2020Full : bt2020Lim;  
 default:  
 return bt601Lim;  
 }  
}  
EglVideoRenderer::~EglVideoRenderer() {  
}  
bool EglVideoRenderer::initialize(DECODER\_PARAMETERS \*params) {  
 m\_width = params->width;  
 m\_height = params->height;  
 if (0 < m\_width) {  
 m\_widthPercent = FIFTY\_PERCENT \* m\_height / m\_width;  
 }  
 if (params->context == nullptr) {  
 eglLog(LOG\_INFO, "EglContextInit execute");  
 return false;  
 }  
 OHNativeWindow \*window = static\_cast<OHNativeWindow \*>(params->context);  
 if ((nullptr == window) || (0 >= params->width) || (0 >= params->height)) {  
 eglLog(LOG\_ERROR, "EglContextInit: param error");  
 return false;  
 }  
 m\_eglWindow = static\_cast<EGLNativeWindowType>(window);  
 if (nullptr == m\_eglWindow) {  
 eglLog(LOG\_ERROR, "m\_eglWindow is null");  
 return false;  
 }  
 m\_eglDisplay = eglGetDisplay(EGL\_DEFAULT\_DISPLAY);  
 if (EGL\_TRUE != eglInitialize(m\_eglDisplay, 0, 0)) {  
 eglLog(LOG\_ERROR, "eglInitialize failed");  
 return false;  
 }  
 EGLConfig eglConfig;  
 EGLint configNum;  
 EGLint configSpec[] = {  
 EGL\_RED\_SIZE, 8,  
 EGL\_GREEN\_SIZE, 8,  
 EGL\_BLUE\_SIZE, 8,  
 EGL\_SURFACE\_TYPE, EGL\_WINDOW\_BIT,  
 EGL\_NONE};  
 if (EGL\_TRUE != eglChooseConfig(m\_eglDisplay, configSpec, &eglConfig, 1, &configNum)) {  
 eglLog(LOG\_ERROR, "eglChooseConfig failed");  
 return false;  
 }  
 m\_eglSurface = eglCreateWindowSurface(m\_eglDisplay, eglConfig, m\_eglWindow, nullptr);  
 if (m\_eglSurface == EGL\_NO\_SURFACE) {  
 eglLog(LOG\_ERROR, "eglCreateWindowSurface failed");  
 return false;  
 }  
 const EGLint ctxAttr[] = {  
 EGL\_CONTEXT\_CLIENT\_VERSION, 2, EGL\_NONE};  
 m\_eglContext = eglCreateContext(m\_eglDisplay, eglConfig, EGL\_NO\_CONTEXT, ctxAttr);  
 if (m\_eglContext == EGL\_NO\_CONTEXT) {  
 eglLog(LOG\_ERROR, "eglCreateContext failed");  
 return false;  
 }  
 if (EGL\_TRUE != eglMakeCurrent(m\_eglDisplay, m\_eglSurface, m\_eglSurface, m\_eglContext)) {  
 eglLog(LOG\_ERROR, "eglMakeCurrent failed");  
 return false;  
 }  
 Shader shader(vertexShaderWithMatrix, fragYUV420P);  
 m\_program = shader.use();  
 if (PROGRAM\_ERROR == m\_program) {  
 eglLog(LOG\_ERROR, "CreateProgram: unable to create program");  
 return false;  
 }  
 static float ver[] = {  
 1.0f, -1.0f, 0.0f,  
 -1.0f, -1.0f, 0.0f,  
 1.0f, 1.0f, 0.0f,  
 -1.0f, 1.0f, 0.0f};  
 GLuint apos = static\_cast<GLuint>(glGetAttribLocation(m\_program, "aPosition"));  
 glEnableVertexAttribArray(apos);  
 glVertexAttribPointer(apos, 3, GL\_FLOAT, GL\_FALSE, 0, ver);  
 static float fragment[] = {  
 1.0f, 0.0f,  
 0.0f, 0.0f,  
 1.0f, 1.0f,  
 0.0f, 1.0f};  
 GLuint aTex = static\_cast<GLuint>(glGetAttribLocation(m\_program, "aTextCoord"));  
 glEnableVertexAttribArray(aTex);  
 glVertexAttribPointer(aTex, 2, GL\_FLOAT, GL\_FALSE, 0, fragment);  
 int width = this->m\_width;  
 int height = this->m\_height;  
 glUniform1i(glGetUniformLocation(m\_program, "yTexture"), 0);  
 glUniform1i(glGetUniformLocation(m\_program, "uTexture"), 1);  
 glUniform1i(glGetUniformLocation(m\_program, "vTexture"), 2);  
 m\_texture\_id[3] = {0};  
 glGenTextures(3, m\_texture\_id);  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[0]);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
 glTexImage2D(GL\_TEXTURE\_2D,  
 0, // 细节基本 默认0  
 GL\_LUMINANCE, // gpu内部格式 亮度，灰度图（这里就是只取一个亮度的颜色通道的意思）  
 width, // 加载的纹理宽度。最好为2的次幂(这里对y分量数据当做指定尺寸算，但显示尺寸会拉伸到全屏？)  
 height, // 加载的纹理高度。最好为2的次幂  
 0, // 纹理边框  
 GL\_LUMINANCE, // 数据的像素格式 亮度，灰度图  
 GL\_UNSIGNED\_BYTE, // 像素点存储的数据类型  
 NULL // 纹理的数据（先不传）  
 );  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[1]);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
 glTexImage2D(GL\_TEXTURE\_2D,  
 0, // 细节基本 默认0  
 GL\_LUMINANCE, // gpu内部格式 亮度，灰度图（这里就是只取一个颜色通道的意思）  
 width / 2, // u数据数量为屏幕的4分之1  
 height / 2,  
 0, // 边框  
 GL\_LUMINANCE, // 数据的像素格式 亮度，灰度图  
 GL\_UNSIGNED\_BYTE, // 像素点存储的数据类型  
 NULL // 纹理的数据（先不传）  
 );  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[2]);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
 glTexImage2D(GL\_TEXTURE\_2D,  
 0, // 细节基本 默认0  
 GL\_LUMINANCE, // gpu内部格式 亮度，灰度图（这里就是只取一个颜色通道的意思）  
 width / 2,  
 height / 2, // v数据数量为屏幕的4分之1  
 0, // 边框  
 GL\_LUMINANCE, // 数据的像素格式 亮度，灰度图  
 GL\_UNSIGNED\_BYTE, // 像素点存储的数据类型  
 NULL // 纹理的数据（先不传）  
 );  
 return true;  
}  
void EglVideoRenderer::renderFrame(AVFrame \*frame) {  
 int width = m\_width;  
 int height = m\_height;  
 glActiveTexture(GL\_TEXTURE0);  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[0]);  
 glTexSubImage2D(GL\_TEXTURE\_2D, 0,  
 0, 0, // 相对原来的纹理的offset  
 width, height, // 加载的纹理宽度、高度。最好为2的次幂  
 GL\_LUMINANCE, GL\_UNSIGNED\_BYTE,  
 frame->data[0]);  
 glActiveTexture(GL\_TEXTURE1);  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[1]);  
 glTexSubImage2D(GL\_TEXTURE\_2D, 0, 0, 0, width / 2, height / 2, GL\_LUMINANCE,  
 GL\_UNSIGNED\_BYTE,  
 frame->data[1]);  
 glActiveTexture(GL\_TEXTURE2);  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[2]);  
 glTexSubImage2D(GL\_TEXTURE\_2D, 0, 0, 0, width / 2, height / 2, GL\_LUMINANCE,  
 GL\_UNSIGNED\_BYTE,  
 frame->data[2]);  
 glDrawArrays(GL\_TRIANGLE\_STRIP, 0, 4);  
 eglSwapBuffers(m\_eglDisplay, m\_eglSurface);  
}  
void EglVideoRenderer::bindTexture(int id) {  
 float borderColorInternal[] = {borderColor[id], 0.0f, 0.0f, 1.0f};  
 glBindTexture(GL\_TEXTURE\_2D, m\_texture\_id[id]);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);  
 textureWidth[id] = id > 0 ? m\_frame\_width / 2 : m\_frame\_width;  
 textureHeight[id] = id > 0 ? m\_frame\_height / 2 : m\_frame\_height;  
 glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RED, textureWidth[id], textureHeight[id],  
 0, GL\_RED, GL\_UNSIGNED\_BYTE, nullptr);  
 glUniform1i(m\_texture\_uniform[id], id);  
}  
void EglVideoRenderer::checkAndUpdateScale(AVFrame \*frame) {  
 if ((m\_frame\_width != frame->width) || (m\_frame\_height != frame->height)) {  
 m\_frame\_width = frame->width;  
 m\_frame\_height = frame->height;  
 glBindBuffer(GL\_ARRAY\_BUFFER, m\_vbo);  
 glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices,  
 GL\_STATIC\_DRAW);  
 int positionLocation =  
 glGetAttribLocation(m\_program, "a\_position");  
 glEnableVertexAttribArray(positionLocation);  
 glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 0, nullptr);  
 for (int i = 0; i < 3; i++) {  
 if (m\_texture\_id[i]) {  
 glDeleteTextures(1, &m\_texture\_id[i]);  
 }  
 }  
 glGenTextures(3, m\_texture\_id);  
 for (int i = 0; i < 3; i++) {  
 bindTexture(i);  
 }  
 bool colorFull = frame->color\_range == AVCOL\_RANGE\_JPEG;  
 glUniform3fv(m\_offset\_location, 1, gl\_color\_offset(colorFull));  
 glUniformMatrix3fv(m\_yuvmat\_location, 1, GL\_FALSE,  
 gl\_color\_matrix(frame->colorspace, colorFull));  
 float frameAspect = ((float)m\_frame\_height / (float)m\_frame\_width);  
 float screenAspect = ((float)m\_height / (float)m\_width);  
 if (frameAspect > screenAspect) {  
 float multiplier = frameAspect / screenAspect;  
 glUniform4f(m\_uv\_data\_location, 0.5f - 0.5f \* (1.0f / multiplier),  
 0.0f, multiplier, 1.0f);  
 } else {  
 float multiplier = screenAspect / frameAspect;  
 glUniform4f(m\_uv\_data\_location, 0.0f,  
 0.5f - 0.5f \* (1.0f / multiplier), 1.0f, multiplier);  
 }  
 }  
}

AVFrameHolder.cpp

#include "video/AVFrameHolder.h"  
AVFrameHolder::AVFrameHolder() {  
   
}  
AVFrameHolder\* AVFrameHolder::m\_holder = nullptr;  
AVFrameHolder \*AVFrameHolder::GetInstance() {  
 if (m\_holder == nullptr) {  
 m\_holder = new AVFrameHolder();  
 return m\_holder;  
 } else {  
 return m\_holder;  
 }  
}

FFmpegVideoDecoder.cpp

#include <Limelight.h>  
#include "FFmpegVideoDecoder.h"  
#include <hilog/log.h>  
#include <stdlib.h>  
#include "video/AVFrameHolder.h"  
#define ffDecodeLog(...) OH\_LOG\_Print(LOG\_APP, LOG\_INFO, LOG\_DOMAIN, "testTag", \_\_VA\_ARGS\_\_);  
#define DECODER\_BUFFER\_SIZE 92 \* 1024 \* 2  
FFmpegVideoDecoder::FFmpegVideoDecoder() {}  
FFmpegVideoDecoder::~FFmpegVideoDecoder() {}  
DECODER\_PARAMETERS\* FFmpegVideoDecoder::getParams(){  
 return &m\_params;  
}  
int FFmpegVideoDecoder::setup(DECODER\_PARAMETERS dparams) {  
 m\_params = dparams;  
 DECODER\_PARAMETERS\* params = &dparams;  
 m\_stream\_fps = params->frame\_rate;  
 ffDecodeLog(  
 "FFmpeg: Setup with format: %{public}s, width: %{public}d, height: %{public}d, fps: %{public}d",  
 params->video\_format == VIDEO\_FORMAT\_H264 ? "H264" : "HEVC",  
 params->width, params->height,  
 params->frame\_rate);  
 av\_log\_set\_level(AV\_LOG\_DEBUG);  
#if LIBAVCODEC\_VERSION\_INT < AV\_VERSION\_INT(58, 10, 100)  
 avcodec\_register\_all();  
#endif  
 m\_packet = av\_packet\_alloc();  
 int perf\_lvl = LOW\_LATENCY\_DECODE;  
 switch (params->video\_format) {  
 case VIDEO\_FORMAT\_H264:  
 m\_decoder = avcodec\_find\_decoder\_by\_name("h264");  
 break;  
 case VIDEO\_FORMAT\_H265:  
 m\_decoder = avcodec\_find\_decoder\_by\_name("hevc");  
 break;  
 }  
 if (m\_decoder == NULL) {  
 ffDecodeLog("FFmpeg: Couldn't find decoder");  
 return -1;  
 }  
 m\_decoder\_context = avcodec\_alloc\_context3(m\_decoder);  
 if (m\_decoder\_context == NULL) {  
 ffDecodeLog("FFmpeg: Couldn't allocate context");  
 return -1;  
 }  
 m\_decoder\_context->width = params->width;  
 m\_decoder\_context->height = params->height;  
 m\_decoder\_context->pix\_fmt = AV\_PIX\_FMT\_VIDEOTOOLBOX;  
 int err = avcodec\_open2(m\_decoder\_context, m\_decoder, NULL);  
 if (err < 0) {  
 ffDecodeLog("FFmpeg: Couldn't open codec");  
 return err;  
 }  
 m\_frames\_count = 2;  
 m\_frames = (AVFrame \*\*)malloc(m\_frames\_count \* sizeof(AVFrame \*));  
 if (m\_frames == NULL) {  
 ffDecodeLog("FFmpeg: Couldn't allocate frames");  
 return -1;  
 }  
 tmp\_frame = av\_frame\_alloc();  
 for (int i = 0; i < m\_frames\_count; i++) {  
 m\_frames[i] = av\_frame\_alloc();  
 if (m\_frames[i] == NULL) {  
 ffDecodeLog("FFmpeg: Couldn't allocate frame");  
 return -1;  
 }  
 m\_frames[i]->format = AV\_PIX\_FMT\_YUV420P;  
 m\_frames[i]->width = params->width;  
 m\_frames[i]->height = params->height;  
 int err = av\_frame\_get\_buffer(m\_frames[i], 256);  
 if (err < 0) {  
 ffDecodeLog("FFmpeg: Couldn't allocate frame buffer:");  
 return -1;  
 }  
 }  
 if (perf\_lvl & DISABLE\_LOOP\_FILTER)  
 m\_decoder\_context->skip\_loop\_filter = AVDISCARD\_ALL;  
 if (perf\_lvl & LOW\_LATENCY\_DECODE)  
 m\_decoder\_context->flags |= AV\_CODEC\_FLAG\_LOW\_DELAY;  
 m\_ffmpeg\_buffer =  
 (char \*)malloc(DECODER\_BUFFER\_SIZE + AV\_INPUT\_BUFFER\_PADDING\_SIZE);  
 if (m\_ffmpeg\_buffer == NULL) {  
 ffDecodeLog("FFmpeg: Not enough memory");  
 cleanup();  
 return -1;  
 }  
 ffDecodeLog("FFmpeg: Setup done!");  
 return DR\_OK;  
}  
void FFmpegVideoDecoder::cleanup() {  
 ffDecodeLog("FFmpeg: Cleanup...");  
 av\_packet\_free(&m\_packet);  
 if (hw\_device\_ctx) {  
 av\_buffer\_unref(&hw\_device\_ctx);  
 }  
 if (m\_decoder\_context) {  
 avcodec\_close(m\_decoder\_context);  
 av\_free(m\_decoder\_context);  
 m\_decoder\_context = NULL;  
 }  
 if (m\_frames) {  
 for (int i = 0; i < m\_frames\_count; i++) {  
 if (m\_frames[i])  
 av\_frame\_free(&m\_frames[i]);  
 }  
 free(m\_frames);  
 m\_frames = nullptr;  
 }  
 if (tmp\_frame) {  
 av\_frame\_free(&tmp\_frame);  
 }  
 if (m\_ffmpeg\_buffer) {  
 free(m\_ffmpeg\_buffer);  
 m\_ffmpeg\_buffer = nullptr;  
 }  
 ffDecodeLog("FFmpeg: Cleanup done!");  
}  
int FFmpegVideoDecoder::submitDecodeUnit(PDECODE\_UNIT du) {  
 if (m\_frames\_in == 0 && du->frameType != FRAME\_TYPE\_IDR) {  
 return DR\_NEED\_IDR;  
 }  
 if (du->fullLength < DECODER\_BUFFER\_SIZE) {  
 PLENTRY entry = du->bufferList;  
 if (!m\_last\_frame) {  
 m\_video\_decode\_stats.measurementStartTimestamp = LiGetMillis();  
 m\_last\_frame = du->frameNumber;  
 } else {  
 m\_video\_decode\_stats.networkDroppedFrames +=  
 du->frameNumber - (m\_last\_frame + 1);  
 m\_video\_decode\_stats.totalFrames +=  
 du->frameNumber - (m\_last\_frame + 1);  
 m\_last\_frame = du->frameNumber;  
 }  
 m\_video\_decode\_stats.receivedFrames++;  
 m\_video\_decode\_stats.totalFrames++;  
 int length = 0;  
 while (entry != NULL) {  
 if (length > DECODER\_BUFFER\_SIZE) {  
 }  
 memcpy(m\_ffmpeg\_buffer + length, entry->data, entry->length);  
 length += entry->length;  
 entry = entry->next;  
 }  
 m\_video\_decode\_stats.totalReassemblyTime +=  
 LiGetMillis() - du->receiveTimeMs;  
 m\_frames\_in++;  
 uint64\_t before\_decode = LiGetMillis();  
 if (length > DECODER\_BUFFER\_SIZE) {  
 ffDecodeLog("FFmpeg: Big buffer to decode...");  
 }  
 if (du->frameType == FRAME\_TYPE\_IDR) {  
 m\_packet->flags = AV\_PKT\_FLAG\_KEY;  
 } else {  
 m\_packet->flags = 0;  
 }  
 if (decode(m\_ffmpeg\_buffer, length) == 0) {  
 m\_frames\_out++;  
 m\_video\_decode\_stats.totalDecodeTime +=  
 LiGetMillis() - before\_decode;  
 m\_video\_decode\_stats.totalDecodeTime +=  
 (m\_frames\_in - m\_frames\_out) \* (1000 / m\_stream\_fps);  
 m\_video\_decode\_stats.decodedFrames++;  
 m\_frame = get\_frame(true);  
 ffDecodeLog("frame size %{public}d X %{public}d", m\_frame->width, m\_frame->height);  
 AVFrameHolder::GetInstance()->push(m\_frame);  
 }  
 } else {  
 ffDecodeLog("FFmpeg: Big buffer to decode... 2");  
 }  
 return DR\_OK;  
}  
int FFmpegVideoDecoder::decode(char \*indata, int inlen) {  
 m\_packet->data = (uint8\_t \*)indata;  
 m\_packet->size = inlen;  
 int err = avcodec\_send\_packet(m\_decoder\_context, m\_packet);  
 if (err != 0) {  
 char error[512];  
 av\_strerror(err, error, sizeof(error));  
 char \*message = error;  
 ffDecodeLog("FFmpeg: Decode failed - %{public}s", message);  
 }  
 return err != DR\_OK ? err : DR\_OK;  
}  
AVFrame \*FFmpegVideoDecoder::get\_frame(bool native\_frame) {  
 int err = avcodec\_receive\_frame(m\_decoder\_context, tmp\_frame);  
 if (hw\_device\_ctx) {  
 if ((err = av\_hwframe\_transfer\_data(m\_frames[m\_next\_frame], tmp\_frame, 0)) < 0) {  
 ffDecodeLog("FFmpeg: Error transferring the data to system memory with error {}", err);  
 return NULL;  
 }  
 av\_frame\_copy\_props(m\_frames[m\_next\_frame], tmp\_frame);  
 } else {  
 m\_frames[m\_next\_frame] = tmp\_frame;  
 }  
 if (err == 0) {  
 m\_current\_frame = m\_next\_frame;  
 m\_next\_frame = (m\_current\_frame + 1) % m\_frames\_count;  
 if (/\*ffmpeg\_decoder == SOFTWARE ||\*/ native\_frame)  
 return m\_frames[m\_current\_frame];  
 } else if (err != AVERROR(EAGAIN)) {  
 char error[512];  
 av\_strerror(err, error, sizeof(error));  
 ffDecodeLog("FFmpeg: Receive failed - %d/%s", err, error);  
 }  
 return NULL;  
}  
VIDEO\_STATS \*FFmpegVideoDecoder::video\_decode\_stats() {  
 uint64\_t now = LiGetMillis();  
 m\_video\_decode\_stats.totalFps =  
 (float)m\_video\_decode\_stats.totalFrames /  
 ((float)(now - m\_video\_decode\_stats.measurementStartTimestamp) /  
 1000);  
 m\_video\_decode\_stats.receivedFps =  
 (float)m\_video\_decode\_stats.receivedFrames /  
 ((float)(now - m\_video\_decode\_stats.measurementStartTimestamp) /  
 1000);  
 m\_video\_decode\_stats.decodedFps =  
 (float)m\_video\_decode\_stats.decodedFrames /  
 ((float)(now - m\_video\_decode\_stats.measurementStartTimestamp) /  
 1000);  
 return (VIDEO\_STATS \*)&m\_video\_decode\_stats;  
}

x509Utils.cpp

#include "x509Utils.h"  
#include <openssl/bio.h>  
#include <openssl/err.h>  
#include <openssl/pem.h>  
#include <openssl/x509.h>  
#include <openssl/x509v3.h>  
#include <moon\_bridge.h>  
void THROW\_BAD\_ALLOC\_IF\_NULL(void \*target) {  
 if (target == nullptr) {  
 ERR\_print\_errors\_fp(stderr);  
 abort();  
 }  
}  
long getFileSize(FILE \*file) {  
 long size;  
 long currentPosition = ftell(file);  
 fseek(file, 0, SEEK\_END);  
 size = ftell(file);  
 fseek(file, currentPosition, SEEK\_SET);  
 return size;  
}  
EVP\_PKEY \*generateKey() {  
 EVP\_PKEY\_CTX \*ctx = EVP\_PKEY\_CTX\_new\_id(EVP\_PKEY\_RSA, NULL);  
 THROW\_BAD\_ALLOC\_IF\_NULL(ctx);  
 EVP\_PKEY\_keygen\_init(ctx);  
 EVP\_PKEY\_CTX\_set\_rsa\_keygen\_bits(ctx, 2048);  
 EVP\_PKEY \*pk = NULL;  
 EVP\_PKEY\_keygen(ctx, &pk);  
 EVP\_PKEY\_CTX\_free(ctx);  
 THROW\_BAD\_ALLOC\_IF\_NULL(pk);  
 return pk;  
}  
int generate\_x509\_certificate(char \*cert\_path, char \*key\_path) {  
 EVP\_PKEY \*pk = nullptr;  
 X509 \*cert = nullptr;  
 FILE \*cert\_file = nullptr;  
 FILE \*key\_file = nullptr;  
 OpenSSL\_add\_all\_algorithms();  
 cert = X509\_new();  
 pk = generateKey();  
 X509\_set\_version(cert, 2);  
 ASN1\_INTEGER\_set(X509\_get\_serialNumber(cert), 0);  
#if OPENSSL\_VERSION\_NUMBER < 0x10100000L  
 X509\_gmtime\_adj(X509\_get\_notBefore(cert), 0);  
 X509\_gmtime\_adj(X509\_get\_notAfter(cert), 60 \* 60 \* 24 \* 365 \* 20); // 20 yrs  
#else  
 ASN1\_TIME \*before = ASN1\_STRING\_dup(X509\_get0\_notBefore(cert));  
 THROW\_BAD\_ALLOC\_IF\_NULL(before);  
 ASN1\_TIME \*after = ASN1\_STRING\_dup(X509\_get0\_notAfter(cert));  
 THROW\_BAD\_ALLOC\_IF\_NULL(after);  
 X509\_gmtime\_adj(before, 0);  
 X509\_gmtime\_adj(after, 60 \* 60 \* 24 \* 365 \* 20); // 20 yrs  
 X509\_set1\_notBefore(cert, before);  
 X509\_set1\_notAfter(cert, after);  
 ASN1\_STRING\_free(before);  
 ASN1\_STRING\_free(after);  
#endif  
 X509\_set\_pubkey(cert, pk);  
 X509\_NAME \*name = X509\_get\_subject\_name(cert);  
 X509\_NAME\_add\_entry\_by\_txt(name, "CN", MBSTRING\_ASC,  
 reinterpret\_cast<unsigned char \*>(const\_cast<char \*>("NVIDIA GameStream Client")),  
 -1, -1, 0);  
 X509\_set\_issuer\_name(cert, name);  
 X509\_sign(cert, pk, EVP\_sha256());  
 cert\_file = fopen(cert\_path, "w");  
 int ret = PEM\_write\_X509(cert\_file, cert);  
 fclose(cert\_file);  
 key\_file = fopen(key\_path, "w");  
 ret = PEM\_write\_PrivateKey(key\_file, pk, nullptr, nullptr, 0, nullptr, nullptr);  
 fclose(key\_file);  
 FILE \*key\_cer\_file = nullptr;  
 key\_cer\_file = fopen(strcat(key\_path, ".cer"), "w");  
 ret = i2d\_PrivateKey\_fp(key\_cer\_file, pk);  
 fclose(key\_cer\_file);  
 EVP\_PKEY\_free(pk);  
 X509\_free(cert);  
 EVP\_cleanup();  
 return 0;  
}  
napi\_value generate\_certificate(napi\_env env, napi\_callback\_info info) {  
 size\_t argc = 2;  
 napi\_value args[2] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 char \*certPath = get\_value\_string(env, args[0]);  
 char \*keyPath = get\_value\_string(env, args[1]);  
 generate\_x509\_certificate(certPath, keyPath);  
 return 0;  
}  
napi\_value verifySignature(napi\_env env, napi\_callback\_info info) {  
 size\_t argc = 3;  
 napi\_value args[3] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 void \*data;  
 void \*signature;  
 void \*serverCertificate;  
 size\_t dataLength;  
 size\_t signatureLength;  
 size\_t serverCertificateLength;  
 napi\_get\_typedarray\_info(  
 env,  
 args[0],  
 nullptr,  
 &dataLength,  
 &data,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 napi\_get\_typedarray\_info(  
 env,  
 args[1],  
 nullptr,  
 &signatureLength,  
 &signature,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 napi\_get\_typedarray\_info(  
 env,  
 args[2],  
 nullptr,  
 &serverCertificateLength,  
 &serverCertificate,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 BIO \*bio = BIO\_new\_mem\_buf(serverCertificate, serverCertificateLength);  
 THROW\_BAD\_ALLOC\_IF\_NULL(bio);  
 X509 \*cert = PEM\_read\_bio\_X509(bio, nullptr, nullptr, nullptr);  
 BIO\_free\_all(bio);  
 EVP\_PKEY \*pubKey = X509\_get\_pubkey(cert);  
 THROW\_BAD\_ALLOC\_IF\_NULL(pubKey);  
 EVP\_MD\_CTX \*mdctx = EVP\_MD\_CTX\_create();  
 THROW\_BAD\_ALLOC\_IF\_NULL(mdctx);  
 EVP\_DigestVerifyInit(mdctx, nullptr, EVP\_sha256(), nullptr, pubKey);  
 EVP\_DigestVerifyUpdate(mdctx, data, dataLength);  
 int result = EVP\_DigestVerifyFinal(mdctx, reinterpret\_cast<unsigned char \*>(signature), signatureLength);  
 EVP\_PKEY\_free(pubKey);  
 EVP\_MD\_CTX\_destroy(mdctx);  
 X509\_free(cert);  
 napi\_value ret;  
 napi\_get\_boolean(env, result > 0, &ret);  
 return ret;  
}  
napi\_value createTypedArray(napi\_env env, size\_t length, napi\_typedarray\_type type, void \*data) {  
 napi\_value arrayBuffer;  
 void \*arrayBufferPtr;  
 napi\_value uint8Array;  
 napi\_status status = napi\_create\_arraybuffer(env, length, &arrayBufferPtr, &arrayBuffer);  
 if (status != napi\_ok) {  
 return NULL;  
 }  
 memcpy(arrayBufferPtr, data, length);  
 status = napi\_create\_typedarray(env, type, length, arrayBuffer, 0, &uint8Array);  
 if (status != napi\_ok) {  
 return NULL;  
 }  
 return uint8Array;  
}  
napi\_value signMessage(napi\_env env, napi\_callback\_info info) {  
 size\_t argc = 2;  
 napi\_value args[2] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 void \*message;  
 size\_t messageLength;  
 void \*privateKey;  
 size\_t privateKeyLength;  
 napi\_get\_typedarray\_info(  
 env,  
 args[0],  
 nullptr,  
 &messageLength,  
 &message,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 napi\_get\_typedarray\_info(  
 env,  
 args[1],  
 nullptr,  
 &privateKeyLength,  
 &privateKey,  
 nullptr,  
 nullptr);  
 EVP\_MD\_CTX \*ctx = EVP\_MD\_CTX\_create();  
 THROW\_BAD\_ALLOC\_IF\_NULL(ctx);  
 BIO \*bio = BIO\_new\_mem\_buf(privateKey, privateKeyLength);  
 THROW\_BAD\_ALLOC\_IF\_NULL(bio);  
 EVP\_PKEY \*m\_PrivateKey = PEM\_read\_bio\_PrivateKey(bio, nullptr, nullptr, nullptr);  
 BIO\_free\_all(bio);  
 EVP\_DigestSignInit(ctx, NULL, EVP\_sha256(), NULL, m\_PrivateKey);  
 EVP\_DigestSignUpdate(ctx, reinterpret\_cast<unsigned char \*>(message), messageLength);  
 size\_t signatureLength = 0;  
 EVP\_DigestSignFinal(ctx, NULL, &signatureLength);  
 unsigned char \*signature = (unsigned char \*)malloc(signatureLength);  
 EVP\_DigestSignFinal(ctx, signature, &signatureLength);  
 napi\_value uint8Array =  
 createTypedArray(env, signatureLength, napi\_uint8\_array, signature);  
 EVP\_MD\_CTX\_destroy(ctx);  
 return uint8Array;  
}  
napi\_value getSignatureFromPemCert(napi\_env env, napi\_callback\_info info) {  
 size\_t argc = 1;  
 napi\_value args[1] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 void \*serverCertificate;  
 size\_t serverCertificateLength;  
 napi\_get\_typedarray\_info(  
 env,  
 args[0],  
 nullptr,  
 &serverCertificateLength,  
 &serverCertificate,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 BIO \*bio = BIO\_new\_mem\_buf(serverCertificate, serverCertificateLength);  
 THROW\_BAD\_ALLOC\_IF\_NULL(bio);  
 X509 \*cert = PEM\_read\_bio\_X509(bio, nullptr, nullptr, nullptr);  
 BIO\_free\_all(bio);  
#if (OPENSSL\_VERSION\_NUMBER < 0x10002000L)  
 ASN1\_BIT\_STRING \*asnSignature = cert->signature;  
#elif (OPENSSL\_VERSION\_NUMBER < 0x10100000L)  
 ASN1\_BIT\_STRING \*asnSignature;  
 X509\_get0\_signature(&asnSignature, NULL, cert);  
#else  
 const ASN1\_BIT\_STRING \*asnSignature;  
 X509\_get0\_signature(&asnSignature, NULL, cert);  
#endif  
 X509\_free(cert);  
 return createTypedArray(env, asnSignature->length, napi\_uint8\_array, asnSignature->data);  
}  
napi\_value encrypt(napi\_env env, napi\_callback\_info info) {  
 size\_t argc = 2;  
 napi\_value args[2] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 void \*message;  
 size\_t messageLength;  
 void \*privateKey;  
 size\_t privateKeyLength;  
 napi\_get\_typedarray\_info(  
 env,  
 args[0],  
 nullptr,  
 &messageLength,  
 &message,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 napi\_get\_typedarray\_info(  
 env,  
 args[1],  
 nullptr,  
 &privateKeyLength,  
 &privateKey,  
 nullptr,  
 nullptr);  
 void \*ciphertext = malloc(messageLength);  
 EVP\_CIPHER\_CTX \*cipher;  
 int ciphertextLen;  
 cipher = EVP\_CIPHER\_CTX\_new();  
 THROW\_BAD\_ALLOC\_IF\_NULL(cipher);  
 EVP\_EncryptInit(cipher, EVP\_aes\_128\_ecb(), reinterpret\_cast<const unsigned char \*>(privateKey), NULL);  
 EVP\_CIPHER\_CTX\_set\_padding(cipher, 0);  
 EVP\_EncryptUpdate(cipher,  
 reinterpret\_cast<unsigned char \*>(ciphertext),  
 &ciphertextLen,  
 reinterpret\_cast<const unsigned char \*>(message),  
 messageLength);  
 EVP\_CIPHER\_CTX\_free(cipher);  
 return createTypedArray(env, messageLength, napi\_uint8\_array, ciphertext);  
}  
napi\_value decrypt(napi\_env env, napi\_callback\_info info) {  
 size\_t argc = 2;  
 napi\_value args[2] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 void \*message;  
 size\_t messageLength;  
 void \*privateKey;  
 size\_t privateKeyLength;  
 napi\_get\_typedarray\_info(  
 env,  
 args[0],  
 nullptr,  
 &messageLength,  
 &message,  
 nullptr, // 可选的 ArrayBuffer  
 nullptr // 可选的偏移  
 );  
 napi\_get\_typedarray\_info(  
 env,  
 args[1],  
 nullptr,  
 &privateKeyLength,  
 &privateKey,  
 nullptr,  
 nullptr);  
 void \*plaintext = malloc(messageLength);  
 EVP\_CIPHER\_CTX \*cipher;  
 int plaintextLen;  
 cipher = EVP\_CIPHER\_CTX\_new();  
 THROW\_BAD\_ALLOC\_IF\_NULL(cipher);  
 EVP\_DecryptInit(cipher, EVP\_aes\_128\_ecb(), reinterpret\_cast<const unsigned char \*>(privateKey), NULL);  
 EVP\_CIPHER\_CTX\_set\_padding(cipher, 0);  
 EVP\_DecryptUpdate(cipher,  
 reinterpret\_cast<unsigned char \*>(plaintext),  
 &plaintextLen,  
 reinterpret\_cast<const unsigned char \*>(message),  
 messageLength);  
 EVP\_CIPHER\_CTX\_free(cipher);  
 return createTypedArray(env, messageLength, napi\_uint8\_array, plaintext);  
}

http\_curl.cpp

#include "http\_curl.h"  
#include "moonlight-core/moon\_bridge.h"  
#include <curl/easy.h>  
#include <stdlib.h>  
#include "string.h"  
#include "napi/native\_api.h"  
#include "x509Utils.h"  
struct AsyncCallbackInfo {  
 napi\_env env;  
 napi\_async\_work asyncWork;  
 napi\_deferred deferred;  
 const char \*url;  
 const int timeout;  
 const char \*clientPath;  
 const char \*keyPath;  
 void \*result;  
 size\_t size;  
 const char \*error;  
};  
struct HTTP\_DATA {  
 char \*memory;  
 size\_t size;  
};  
size\_t write\_callback(void \*contents, size\_t size, size\_t nmemb, void \*userp) {  
 size\_t realsize = size \* nmemb;  
 HTTP\_DATA \*mem = (HTTP\_DATA \*)userp;  
 mem->memory = (char \*)realloc(mem->memory, mem->size + realsize + 1);  
 if (mem->memory == NULL)  
 return 0;  
 memcpy(&(mem->memory[mem->size]), contents, realsize);  
 mem->size += realsize;  
 mem->memory[mem->size] = 0;  
 return realsize;  
}  
static CURL \*curl;  
int http\_init( AsyncCallbackInfo \*cb) {  
 if (!curl) {  
 curl\_global\_init(CURL\_GLOBAL\_ALL);  
 } else {  
 return 0;  
 }  
 curl = curl\_easy\_init();  
 if (!curl)  
 return 1;  
 curl\_easy\_setopt(curl, CURLOPT\_SSL\_VERIFYHOST, 0L);  
 curl\_easy\_setopt(curl, CURLOPT\_SSLENGINE\_DEFAULT, 1L);  
 curl\_easy\_setopt(curl, CURLOPT\_SSL\_VERIFYPEER, 0L);  
 curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, write\_callback);  
 curl\_easy\_setopt(curl, CURLOPT\_FAILONERROR, 1L);  
 curl\_easy\_setopt(curl, CURLOPT\_SSL\_SESSIONID\_CACHE, 0L);  
 return 0;  
}  
void http\_request(AsyncCallbackInfo \*cb) {  
 HTTP\_DATA \*http\_data = (HTTP\_DATA \*)malloc(sizeof(HTTP\_DATA));  
 http\_data->memory = (char \*)malloc(1);  
 http\_data->size = 0;  
 curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, http\_data);  
 curl\_easy\_setopt(curl, CURLOPT\_URL, cb->url);  
 curl\_easy\_setopt(curl, CURLOPT\_TIMEOUT, cb->timeout);  
 if (cb->clientPath != nullptr) {  
 curl\_easy\_setopt(curl, CURLOPT\_SSLCERTTYPE, "PEM");  
 curl\_easy\_setopt(curl, CURLOPT\_SSLCERT, cb->clientPath);  
 }  
 if (cb->keyPath != nullptr) {  
 curl\_easy\_setopt(curl, CURLOPT\_SSLKEYTYPE, "PEM");  
 curl\_easy\_setopt(curl, CURLOPT\_SSLKEY, cb->keyPath);  
 }  
 CURLcode res = curl\_easy\_perform(curl);  
 if (res != CURLE\_OK) {  
 cb->error = curl\_easy\_strerror(res);  
 } else if (http\_data->memory == NULL) {  
 cb->error = "Curl: memory = NULL";  
 }  
 cb->result = http\_data->memory;  
 cb->size = http\_data->size;  
 free(http\_data->memory);  
 free(http\_data);  
}  
void getCurl(napi\_env env, AsyncCallbackInfo \*cb) {  
 CURL \*curl;  
 CURLcode res;  
 curl = curl\_easy\_init();  
 if (curl) {  
 curl\_easy\_setopt(curl, CURLOPT\_URL, cb->url);  
 curl\_easy\_setopt(curl, CURLOPT\_TIMEOUT, cb->timeout);  
 curl\_easy\_setopt(curl, CURLOPT\_VERBOSE, 1L);  
 curl\_easy\_setopt(curl, CURLOPT\_SSL\_VERIFYPEER, 0L);  
 curl\_easy\_setopt(curl, CURLOPT\_SSL\_VERIFYHOST, 0L);  
 curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, write\_callback);  
 HTTP\_DATA \*http\_data = (HTTP\_DATA \*)malloc(sizeof(HTTP\_DATA));  
 http\_data->memory = (char \*)malloc(1);  
 http\_data->size = 0;  
 curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, http\_data);  
 res = curl\_easy\_perform(curl);  
 if (res != CURLE\_OK) {  
 cb->error = curl\_easy\_strerror(res);  
 } else if (http\_data->memory == NULL) {  
 cb->error = "Curl: memory = NULL";  
 }  
 cb->result = http\_data->memory;  
 cb->size = http\_data->size;  
 free(http\_data->memory);  
 free(http\_data);  
 }  
}  
napi\_value GetRequest(napi\_env env, napi\_callback\_info info) {  
 napi\_deferred deferred;  
 napi\_value promise;  
 napi\_create\_promise(env, &deferred, &promise);  
 size\_t argc = 4;  
 napi\_value args[4] = {nullptr};  
 napi\_get\_cb\_info(env, info, &argc, args, nullptr, nullptr);  
 char \*url = get\_value\_string(env, args[0]);  
 int timeout;  
 napi\_get\_value\_int32(env, args[1], &timeout);  
 char \*clientPath = get\_value\_string(env, args[2]);  
 char \*keyPath = get\_value\_string(env, args[3]);  
 AsyncCallbackInfo \*asyncCallbackInfo = new AsyncCallbackInfo{  
 .env = env,  
 .asyncWork = nullptr,  
 .deferred = deferred,  
 .url = url,  
 .timeout = timeout,  
 .clientPath = clientPath,  
 .keyPath = keyPath,  
 .result = nullptr};  
 napi\_value resourceName;  
 napi\_create\_string\_latin1(env, url, NAPI\_AUTO\_LENGTH, &resourceName);  
 napi\_create\_async\_work(  
 env, nullptr, resourceName,  
 [](napi\_env env, void \*data) {  
 http\_request((AsyncCallbackInfo \*)data);  
 },  
 [](napi\_env env, napi\_status status, void \*data) {  
 AsyncCallbackInfo \*asyncCallbackInfo = (AsyncCallbackInfo \*)data;  
 if (asyncCallbackInfo->error == nullptr) {  
 napi\_value result = createTypedArray(env, asyncCallbackInfo->size, napi\_uint8\_array, asyncCallbackInfo->result);  
 napi\_resolve\_deferred(asyncCallbackInfo->env, asyncCallbackInfo->deferred, result);  
 } else {  
 napi\_value result;  
 napi\_create\_string\_utf8(env, asyncCallbackInfo->error, NAPI\_AUTO\_LENGTH, &result);  
 napi\_reject\_deferred(env, asyncCallbackInfo->deferred, result);  
 }  
 napi\_delete\_async\_work(env, asyncCallbackInfo->asyncWork);  
 delete asyncCallbackInfo;  
 },  
 (void \*)asyncCallbackInfo, &asyncCallbackInfo->asyncWork);  
 napi\_queue\_async\_work(env, asyncCallbackInfo->asyncWork);  
 return promise;  
}  
static napi\_value CurlClientClassConstructor(napi\_env env, napi\_callback\_info info) {  
 http\_init(nullptr);  
 napi\_value thisArg = nullptr;  
 void \*data = nullptr;  
 napi\_get\_cb\_info(env, info, nullptr, nullptr, &thisArg, &data);  
 napi\_value global = nullptr;  
 napi\_get\_global(env, &global);  
 return thisArg;  
}  
static napi\_value Close(napi\_env env, napi\_callback\_info info) {  
   
 return 0;  
}  
void HttpCurlInit(napi\_env env, napi\_value exports) {  
 napi\_property\_descriptor descriptors[] = {  
 {"get", nullptr, GetRequest, nullptr, nullptr, nullptr, napi\_default, nullptr},  
 {"close", nullptr, Close, nullptr, nullptr, nullptr, napi\_default, nullptr}};  
 napi\_value result = nullptr;  
 napi\_define\_class(env, "CurlClient", NAPI\_AUTO\_LENGTH, CurlClientClassConstructor, nullptr,  
 sizeof(descriptors) / sizeof(\*descriptors), descriptors, &result);  
 napi\_set\_named\_property(env, exports, "CurlClient", result);  
}

GamePage.ets

import gameViewModel from 'ets/entryability/GameViewModel'  
import { VideoStatus } from 'libentry.so';  
import router from '@ohos.router';  
import { Spinner } from './compoments/Loading'  
import { StreamSettings } from '../uitls/StreamSetttings';

import { loadSettings} from '../uitls/StreamSetttings';  
import { VirtualController, } from '../virtual\_controller/VirtualController';  
import { VirtualControllerConfigurationLoader } from '../virtual\_controller/VirtualControllerConfigurationLoader';  
import { VirtualControllerButton } from '../virtual\_controller/common';  
import { Icon } from './compoments/Title';  
@Entry  
@Component  
struct GamePage {  
 @State videoStatus: VideoStatus = null  
 xComponentContext: any | undefined = undefined;  
 @State virtualController: VirtualController = null  
 @State settings: StreamSettings = null  
 dialogController: CustomDialogController = new CustomDialogController({  
 builder: Spinner({ title: $r('app.string.conn\_establishing\_title'), text: "" }),  
 autoCancel: false,  
 alignment: DialogAlignment.Center,  
 customStyle: true  
 })  
 aboutToDisappear() {  
 this.dialogController.close()  
 gameViewModel.stop()  
 delete this.dialogController, // 删除dialogController  
 this.dialogController = undefined // 将dialogController置空  
 }  
 async onInit(){  
 this.dialogController.open()  
 const settings = await loadSettings(getContext(this))  
 this.settings = settings  
 await gameViewModel.init(router.getParams()["computer"], router.getParams()["app"], settings)  
 await gameViewModel.start(this.dialogController, getContext(this))  
 if (settings.touchscreen\_trackpad){  
 const loader = new VirtualControllerConfigurationLoader()  
 loader.createDefaultLayout(gameViewModel.virtualController, settings)  
 VirtualControllerConfigurationLoader.loadFromPreferences(gameViewModel.virtualController, getContext(this))  
 this.virtualController = gameViewModel.virtualController  
 }  
 gameViewModel.conn.onVideoStatus((s)=>{  
 this.videoStatus = s;  
 })  
 }  
 aboutToAppear(){  
 this.onInit()  
 }  
 build() {  
 Stack({ alignContent: Alignment.TopStart }){  
 XComponent({ id: 'xcomponentId1', type: 'surface', libraryname: 'entry' })  
 .onLoad((context: any) => {  
 this.xComponentContext = context  
 })  
 .onDestroy(() => {  
 console.log("onDestroy");  
 }).borderWidth(0).height("100%").width('100%')  
 if(this.videoStatus){  
 Column(){  
 Text(`视频流: ${this.settings?.resolution\_list} ${this.videoStatus?.totalFps?.toFixed(2) || 0 } FPS`).fontColor(Color.White)  
 Text("解码器: ").fontColor(Color.White)  
 Text(`网络接收帧数: ${this.videoStatus?.receivedFps?.toFixed(2) || 0 } FPS`) .fontColor(Color.White)  
 Text(`渲染帧数: ${this.videoStatus?.renderedFps?.toFixed(2) || 0 } FPS`).fontColor(Color.White)  
 Text(`网络丢失帧: ${this.videoStatus?.networkDroppedRate?.toFixed(2) || 0 } %`).fontColor(Color.White)  
 Text(`平均网络延迟: ${this.videoStatus?.receivedTime?.toFixed(2) || 0 } `).fontColor(Color.White)  
 Text(`平均解码时间: ${this.videoStatus?.decodeTime?.toFixed(2) || 0 } ms`).fontColor(Color.White)  
 }.backgroundColor(Color.Black).opacity(0.5)  
 }  
 if(this.virtualController){  
 ForEach(this.virtualController.elements.convertToArray(), (d) => {  
 VirtualControllerButton({ element: d, layout: d.layout })  
 }, (d) =>d.elementId.toString())  
 Button(){  
 Icon({icon:$r("app.media.settings")})  
 }.offset({y: 20}).onClick(()=>{  
 this.virtualController.onSettingsClick(getContext(this))  
 })  
 }  
 }.backgroundColor(Color.Black)  
 .width('100%')  
 .height('100%')  
 }  
}

AppPage.ets

import router from '@ohos.router'  
import viewModel from '../entryability/ComputerManagerViewModel'  
import { ComputerDetails } from '../entryability/computers/ComputerDetails'  
import { NvHttp } from '../entryability/http/NvHttp'  
import { Icon} from './compoments/Title'

import { NavTitle } from './compoments/Title'  
import { NvApp } from '../entryability/http/NvApp'  
import image from '@ohos.multimedia.image'  
@Component  
struct AppView {  
 computer: ComputerDetails  
 app: NvApp  
 @State image: PixelMap = undefined;  
 @Builder  
 pcMenu() {  
 Menu() {  
 MenuItem({ content: $r("app.string.applist\_menu\_resume") })  
 MenuItem({ content: $r("app.string.applist\_menu\_quit") })  
 MenuItem({ content: $r("app.string.applist\_menu\_quit\_and\_start") })  
 }  
 }  
 aboutToAppear(){  
 viewModel.readImageByDisk(this.app).then((res)=>{  
 let options = {  
 alphaType: 0, // 透明度  
 editable: false, // 是否可编辑  
 pixelFormat: 3, // 像素格式  
 scaleMode: 1, // 缩略值  
 size: { height: 100, width: 100}  
 }  
 let imageSource = image.createImageSource(res.buffer);  
 if(imageSource)  
 imageSource.createPixelMap(options).then((pixelMap) => {  
 this.image = pixelMap  
 })  
 })  
 }  
 build(){  
 Column(){  
 Stack(){  
 if (this.image){  
 Image(this.image).height(150).width(100)  
 } else {  
 Text(this.app.appName).fontColor(Color.White)  
 }  
 if(this.computer.runningGameId == this.app.appId){  
 Column(){  
 Icon({icon: $r("app.media.play\_arrow\_FILL1\_wght700\_GRAD200\_opsz48"), iconSize:48})  
 Icon({icon: $r("app.media.stop\_FILL1\_wght700\_GRAD200\_opsz48"), iconSize: 48})  
 }  
 }  
 }  
 }.onClick(()=>{  
 router.pushUrl({ url:"pages/GamePage", params: { app: this.app, computer: this.computer } })  
 })  
 }  
}  
@Entry  
@Component  
struct AppPage {  
 computer: ComputerDetails  
 @State appList: NvApp[] = []  
 aboutToAppear(){  
 const params = router.getParams();  
 viewModel.getComputerByUUid(params["uuid"]).then((d)=>{  
 this.computer = d  
 const appList = d.appList  
 this.appList = appList.filter((d) => d.appId != null);  
 })  
 }  
 build(){  
 Column(){  
 NavTitle({ title: router.getParams()["computerName"] }).width("100%")  
 Grid() {  
 ForEach(this.appList, (d:NvApp)=>{  
 GridItem(){  
 AppView({app: d, computer: this.computer})  
 }  
 },(item) => JSON.stringify(item))  
 }  
 .rowsTemplate('1fr 1fr 1fr')  
 .columnsTemplate('1fr 1fr 1fr')  
 }.padding(10).height("100%").width("100%").backgroundColor($r("app.color.page\_background"))  
 }  
}

AddPage.ets

import { NavTitle } from './compoments/Title'  
import viewModel from '../entryability/ComputerManagerViewModel'  
import router from '@ohos.router'  
import { Alert } from './compoments/Loading'

import { Loading} from './compoments/Loading'  
import promptAction from '@ohos.promptAction'  
@Entry  
@Component  
struct AddPage {  
 viewModel = viewModel  
 @State ip: string = ''  
 loadingDialog: CustomDialogController = new CustomDialogController({  
 builder: Loading({  
 title: $r("app.string.title\_add\_pc"),  
 text: $r("app.string.msg\_add\_pc"),  
 }),  
 autoCancel: false,  
 alignment: DialogAlignment.Center,  
 customStyle: true  
 })  
 build(){  
 Column(){  
 NavTitle({ title: $r("app.string.title\_add\_pc")})  
 Row(){  
 TextInput({placeholder:'串流电脑的ip地址', text: "[2409:8a00:79a1:5ae1:27d5:8223:fcba:4b17]"}).fontColor(Color.White).onChange((value: string) => {  
 this.ip = value;  
 }).layoutWeight(1).type(InputType.Normal)  
 Button("确定").width(150).margin({left:5}).onClick(()=>{  
 this.loadingDialog.open()  
 this.viewModel.addPc(this.ip).then((result)=>{  
 this.loadingDialog.close()  
 if(result.success){  
 router.back()  
 }else{  
 promptAction.showDialog({ title: $r('app.string.conn\_error\_title'), message: result.message})  
 }  
 }).catch(()=>{  
 promptAction.showDialog({ title: $r('app.string.conn\_error\_title'), message: "未知错误"})  
 this.loadingDialog.close()  
 })  
 })  
 }.width("100%")  
 }.padding(20).height("100%").width("100%").backgroundColor($r("app.color.page\_background"))  
 }  
}

SettingsPage.ets

import { NavTitle } from './compoments/Title';  
import dataPreferences from '@ohos.data.preferences';  
import { getResString } from '../uitls/ResString';  
import { loadSettings } from '../uitls/StreamSetttings';

import { StreamSettings} from '../uitls/StreamSetttings';  
import router from '@ohos.router';  
@Entry  
@Component  
struct SettingsPage {  
 scroller: Scroller = new Scroller();  
 preferences: dataPreferences.Preferences = null  
 @Provide settings: StreamSettings = null  
 onUpdateValue = (key: string, value: any)=>{  
 this.preferences.put(key, value)  
 this.preferences.flush()  
 }  
 aboutToAppear(){  
 this.loadPreferences(getContext(this))  
 }  
 async loadPreferences(context: Context){  
 this.preferences = await dataPreferences.getPreferences(context, "StreamSettings")  
 this.settings = await loadSettings(context)  
 }  
 build() {  
 Column() {  
 NavTitle({ title: "设置" })  
 Scroll(this.scroller) {  
 Column({ space: 10 }) {  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_basic\_settings"))  
 ListPreference({  
 pKey: "resolution\_list",  
 title: $r("app.string.title\_resolution\_list"),  
 summary: $r("app.string.summary\_resolution\_list"),  
 names:[$r("app.string.resolution\_360p"), $r("app.string.resolution\_720p"), $r("app.string.resolution\_1080p"), $r("app.string.resolution\_1440p"), $r("app.string.resolution\_4k")],  
 entries: ["640x360", "1280x720", "1920x1080", "2560x1440", "3840x2160"],  
 value: "1280x720",  
 onChange: this.onUpdateValue  
 })  
 ListPreference({  
 pKey: "fps\_list",  
 title: $r("app.string.title\_fps\_list"),  
 summary: $r("app.string.summary\_fps\_list"),  
 value: "60",  
 names:[$r("app.string.fps\_30"), $r("app.string.fps\_60"), $r("app.string.fps\_90"), $r("app.string.fps\_120")],  
 entries: ["30", "60", "90", "120"],  
 onChange: this.onUpdateValue  
 })  
 SeekBarPreference({  
 pKey: "seekbar\_bitrate",  
 title: $r("app.string.title\_seekbar\_bitrate"),  
 summary: $r("app.string.summary\_seekbar\_bitrate"),  
 value: "",  
 onChange: this.onUpdateValue  
 })  
 ListPreference({  
 pKey: "frame\_pacing",  
 title: $r("app.string.title\_frame\_pacing"),  
 summary: $r("app.string.summary\_frame\_pacing"),  
 names:[$r("app.string.pacing\_latency"), $r("app.string.pacing\_balanced"), $r("app.string.pacing\_balanced\_alt"), $r("app.string.pacing\_smoothness")],  
 entries: ["latency", "balanced", "cap-fps", "smoothness"],  
 value: "",  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "stretch\_video",  
 title: $r("app.string.title\_checkbox\_stretch\_video"),  
 summary: null,  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_audio\_settings"))  
 ListPreference({  
 pKey: "audio\_config\_list",  
 title: $r("app.string.title\_audio\_config\_list"),  
 summary: $r("app.string.summary\_audio\_config\_list"),  
 names:[$r("app.string.audioconf\_stereo"), $r("app.string.audioconf\_51surround"), $r("app.string.audioconf\_71surround")],  
 entries: ["2", "51", "71"],  
 value: "1280x720",  
 onChange: this.onUpdateValue  
 })  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_gamepad\_settings"))  
 SeekBarPreference({  
 pKey: "seekbar\_deadzone",  
 title: $r("app.string.title\_seekbar\_deadzone"),  
 summary: $r("app.string.summary\_seekbar\_deadzone"),  
 value: "",  
 onChange: this.onUpdateValue  
 })  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_input\_settings"))  
 CheckBoxPreference({  
 pKey: "touchscreen\_trackpad",  
 title: $r("app.string.title\_checkbox\_touchscreen\_trackpad"),  
 summary: $r("app.string.summary\_checkbox\_touchscreen\_trackpad"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_on\_screen\_controls\_settings"))  
 CheckBoxPreference({  
 pKey: "show\_onscreen\_controls",  
 title: $r("app.string.title\_checkbox\_show\_onscreen\_controls"),  
 summary: $r("app.string.summary\_checkbox\_show\_onscreen\_controls"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "vibrate\_osc",  
 title: $r("app.string.title\_checkbox\_vibrate\_osc"),  
 summary: $r("app.string.summary\_checkbox\_vibrate\_osc"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "only\_l3r3",  
 title: $r("app.string.title\_only\_l3r3"),  
 summary: $r("app.string.summary\_only\_l3r3"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "osc\_opacity",  
 title: $r("app.string.dialog\_title\_osc\_opacity"),  
 summary: $r("app.string.summary\_osc\_opacity"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "reset\_osc",  
 title: $r("app.string.title\_reset\_osc"),  
 summary: $r("app.string.summary\_reset\_osc"),  
 value: false,  
 onChange: ()=>{  
 router.pushUrl({url:"pages/VirtualControllerSettings"})  
 }})  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_host\_settings"))  
 CheckBoxPreference({  
 pKey: "enable\_sops",  
 title: $r("app.string.title\_checkbox\_enable\_sops"),  
 summary: $r("app.string.summary\_checkbox\_enable\_sops"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "host\_audio",  
 title: $r("app.string.title\_checkbox\_host\_audio"),  
 summary: $r("app.string.summary\_checkbox\_host\_audio"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_ui\_settings"))  
 }.alignItems(HorizontalAlign.Start)  
 Column({ space: 10 }) {  
 Text($r("app.string.category\_advanced\_settings"))  
 CheckBoxPreference({  
 pKey: "unlock\_fps",  
 title: $r("app.string.title\_unlock\_fps"),  
 summary: $r("app.string.summary\_unlock\_fps"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "refresh\_rate",  
 title: $r("app.string.title\_checkbox\_reduce\_refresh\_rate"),  
 summary: $r("app.string.summary\_checkbox\_reduce\_refresh\_rate"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "disable\_warnings",  
 title: $r("app.string.title\_checkbox\_disable\_warnings"),  
 summary: $r("app.string.summary\_checkbox\_disable\_warnings"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 ListPreference({  
 pKey: "video\_format",  
 title: $r("app.string.title\_video\_format"),  
 summary: $r("app.string.summary\_video\_format"),  
 names:[$r("app.string.videoformat\_auto"), $r("app.string.videoformat\_hevcalways"), $r("app.string.videoformat\_h264always")],  
 entries: ["auto", "h265", "h264"],//$r("app.string.videoformat\_av1always"), "av1",  
 value: "h264",  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "enable\_hdr",  
 title: $r("app.string.title\_enable\_hdr"),  
 summary: $r("app.string.summary\_enable\_hdr"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "full\_range",  
 title: $r("app.string.title\_full\_range"),  
 summary: $r("app.string.summary\_full\_range"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "enable\_perf\_overlay",  
 title: $r("app.string.title\_enable\_perf\_overlay"),  
 summary: $r("app.string.summary\_enable\_perf\_overlay"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 CheckBoxPreference({  
 pKey: "enable\_post\_stream\_toast",  
 title: $r("app.string.title\_enable\_post\_stream\_toast"),  
 summary: $r("app.string.summary\_enable\_post\_stream\_toast"),  
 value: false,  
 onChange: this.onUpdateValue  
 })  
 }.alignItems(HorizontalAlign.Start)  
 Blank().height(30)  
 }.padding(10)  
 }.scrollable(ScrollDirection.Vertical)  
 }.height('100%').backgroundColor($r("app.color.page\_background"))  
 }  
}  
@Builder  
function BaseReference(hasCheck: Boolean = false, onClick: () => void) {  
 Row() {  
 Column({ space: 2 }) {  
 Text(this.title).fontSize(20).fontColor(0xFFFFFF)  
 Text(this.summary).fontSize(16).fontColor(0xC3C3C3)  
 }.layoutWeight(1).alignItems(HorizontalAlign.Start)  
 if (hasCheck) {  
 Toggle({ type: ToggleType.Checkbox, isOn: this.value == true })  
 }  
 }.padding({ right: 20}).width('100%').onClick(onClick)  
}  
@CustomDialog  
struct SeekBarDialog{  
 @Link inputValue : number  
 title: string| Resource = ""  
 summary: Resource  
 dialogController: CustomDialogController  
 onConfirm: () => void  
 build(){  
 Column() {  
 Text(this.title)  
 Text(this.summary)  
 Blank().height(50)  
 Slider({  
 value: this.inputValue,  
 min: 0,  
 max: 100,  
 style: SliderStyle.InSet  
 })  
 .blockColor('#191970')  
 .trackColor('#ffe0eaec')  
 .selectedColor('#c3c3c3')  
 .showTips(true)  
 .onChange((value: number, mode: SliderChangeMode) => {  
 this.inputValue = value  
 console.info('value:' + value + 'mode:' + mode.toString())  
 })  
 Blank().height(30)  
 Button('确定').onClick(()=>{  
 this.onConfirm()  
 this.dialogController.close()  
 })  
 }.padding(20)  
 }  
}  
@Component  
struct SeekBarPreference {  
 @Consume @Watch('onSettingsUpdated')  
 settings: StreamSettings  
 pKey: string  
 title: Resource = null  
 summary: Resource = null  
 @State value: string = ""  
 onChange: (key, value) => void  
 aboutToAppear() {  
 }  
 onSettingsUpdated(){  
 if(this.settings[this.pKey] != null){  
 this.value = this.settings[this.pKey]  
 }  
 }  
 dialogController: CustomDialogController = new CustomDialogController({  
 builder: SeekBarDialog({  
 inputValue: $value,  
 title: this.title,  
 summary: this.summary,  
 onConfirm: ()=>{  
 this.onChange(this.pKey, this.value)  
 }  
 }),  
 autoCancel: true,  
 alignment: DialogAlignment.Bottom,  
 customStyle: false  
 })  
 build() {  
 BaseReference(false, () => {  
 this.dialogController.open()  
 this.onChange(this.pKey, this.value)  
 })  
 }  
}  
@Component  
struct ListPreference {  
 @Consume @Watch('onSettingsUpdated')  
 settings: StreamSettings  
 pKey: string  
 title: Resource  
 summary: Resource  
 @State value: string = ""  
 onChange: (key, value) => void  
 private select: number = 0  
 entries: string[] = []  
 names: Resource[]  
 private labels: string[] = []  
 aboutToAppear() {  
 this.getLabels()  
 }  
 onSettingsUpdated(){  
 if(!this.settings || this.settings[this.pKey] == null){  
 this.select == this.entries.indexOf(this.value)  
 } else {  
 this.value = this.settings[this.pKey]  
 this.select = this.entries.indexOf(this.value)  
 }  
 }  
 async getLabels() {  
 this.onSettingsUpdated()  
 for (let r of this.names) {  
 this.labels.push(await getResString(this, r))  
 }  
 }  
 build() {  
 BaseReference(false, () => {  
 TextPickerDialog.show({  
 range: this.labels,  
 selected: this.select,  
 onAccept: (value: TextPickerResult) => {  
 this.select = value.index  
 this.value = this.entries[value.index]  
 this.onChange(this.pKey, this.value)  
 },  
 })  
 })  
 }  
}  
@Component  
struct CheckBoxPreference {  
 @Consume @Watch('onSettingsUpdated')  
 settings: StreamSettings  
 pKey: string  
 title: Resource  
 summary: Resource  
 @State value: boolean = false  
 onChange: (key, value) => void  
 onSettingsUpdated(){  
 if(this.settings[this.pKey] != null){  
 this.value = this.settings[this.pKey]  
 }  
 }  
 build() {  
 BaseReference(true, () => {  
 this.value = !this.value  
 this.onChange(this.pKey, this.value)  
 })  
 }  
}

Title.ets

import router from '@ohos.router'  
@Component  
export struct NavTitle {  
 title: string | Resource  
 build() {  
 Stack({alignContent:Alignment.Start}){  
 Row(){  
 Text(this.title).fontColor(Color.White)  
 .fontSize(23).textAlign(TextAlign.Center)  
 }.  
 width("100%")  
 .justifyContent(FlexAlign.Center)  
 .height(50)  
 Icon({icon:$r('app.media.arrow\_left')}).onClick(()=>{  
 router.back()  
 })  
 }.width("100%").padding(10)  
 }  
}  
@Component  
export struct Icon {  
 icon: Resource  
 iconSize: number = 48  
 build(){  
 Image(this.icon)  
 .width(this.iconSize).height(this.iconSize)  
 }  
}  
@Component  
export struct MainTitle {  
 title: string  
 build() {  
 Row(){  
 Icon({icon:$r("app.media.settings")}).onClick(()=>{  
 router.pushUrl({ url:"pages/SettingsPage" })  
 })  
 Icon({icon: $r('app.media.ic\_add\_to\_queue\_white\_48px')}).onClick(()=>{  
 router.pushUrl({ url:"pages/AddPage" })  
 })  
 }.justifyContent(FlexAlign.SpaceBetween)  
 .width("100%").padding(10)  
 .height(50)  
 }  
}

DialogUtils.ets

Loading.ets

import emitter from '@ohos.events.emitter'  
import { SpinnerEventId } from '../../uitls/CommonEvents'  
@CustomDialog  
export struct Spinner {  
 title: string | Resource  
 @State text: string = ""  
 controller: CustomDialogController  
 aboutToAppear(){  
 let innerEvent = {  
 eventId: SpinnerEventId,  
 };  
 emitter.on(innerEvent, (eventData)=>{  
 this.text = eventData.data["message"]  
 })  
 }  
 build() {  
 Column(){  
 Text(this.title).fontSize(18)  
 Row(){  
 LoadingProgress().height(100).width(80)  
 if (this.text == ""){  
 Text($r("app.string.conn\_establishing\_msg")).maxLines(2).height(100).textOverflow({overflow: TextOverflow.Ellipsis}).layoutWeight(1)  
 } else {  
 Text(this.text).maxLines(2).height(100).textOverflow({overflow: TextOverflow.Ellipsis}).layoutWeight(1)  
 }  
 }  
 }.padding(20)  
 .alignItems(HorizontalAlign.Start).backgroundColor(Color.White).borderRadius(10)  
 .width("40%")  
 .height(140)  
 }  
}  
@CustomDialog  
export struct Loading {  
 title: string | Resource  
 text: string | Resource = ""  
 controller: CustomDialogController  
 build() {  
 Column(){  
 Text(this.title).fontSize(18)  
 Row(){  
 LoadingProgress().height(100).width(80)  
 Text(this.text).maxLines(2).height(100).textOverflow({overflow: TextOverflow.Ellipsis}).layoutWeight(1)  
 }  
 }.padding(20)  
 .alignItems(HorizontalAlign.Start).backgroundColor(Color.White).borderRadius(10)  
 .width("40%")  
 .height(140)  
 }  
}  
@CustomDialog  
export struct Alert {  
 title: string | Resource = "建立连接中"  
 message: string | Resource  
 controller: CustomDialogController  
 build() {  
 Column(){  
 Text(this.title).fontSize(18)  
 Blank().height(20)  
 Text(this.message)  
 }.padding(20)  
 .alignItems(HorizontalAlign.Start).backgroundColor(Color.White).borderRadius(10)  
 .width("40%")  
 }  
}

Index.ets

import hilog from '@ohos.hilog';  
import router from '@ohos.router';  
import LimelightCertProvider from '../entryability/crypto/LimelightCryptoProvider';  
import { AddressTuple } from '../entryability/computers/ComputerDetails';  
import { NvHttp } from '../entryability/http/NvHttp';  
@Entry  
@Component  
struct Index {  
 @State textValue: string = ''  
   
 onCancel() {  
 console.info('Callback when the first button is clicked')  
 }  
 onAccept() {  
 console.info('Callback when the second button is clicked')  
 }  
 build() {  
 Stack({ alignContent: Alignment.Top }){  
 Row() {  
 Button("start")  
 .fontSize(50)  
 .fontWeight(FontWeight.Bold)  
 .onClick(() => {  
 router.pushUrl({ url:"pages/GamePage" })  
 })  
 Button("pair")  
 .fontSize(50)  
 .fontWeight(FontWeight.Bold)  
 .onClick(async () => {  
 try {  
 const http = new NvHttp(new AddressTuple("192.168.3.5", 47989), 47984, null, LimelightCertProvider)  
 const server = await http.getServerInfo(true)  
 await http.pm.pair(server, "12345")  
 }catch (e){  
 console.log(e);  
 }  
 })  
 Button("pc")  
 .fontSize(50)  
 .fontWeight(FontWeight.Bold)  
 .onClick(async () => {  
 try {  
 router.pushUrl({ url:"pages/PcPage" })  
 }catch (e){  
 console.log(e);  
 }  
 })  
 }  
 }  
 .width('100%')  
 .height('100%').backgroundColor($r("app.color.page\_background"))  
 }  
}

PcPage.ets

import { Icon} from './compoments/Title'

import { MainTitle } from './compoments/Title'

import { NavTitle } from './compoments/Title'  
import viewModel from '../entryability/ComputerManagerViewModel'  
import { ComputerState } from '../entryability/computers/ComputerDetails'

import { ComputerDetails} from '../entryability/computers/ComputerDetails'  
import promptAction from '@ohos.promptAction'  
import { PairState } from '../entryability/http/PairingManager'  
import limelightCertProvider from '../entryability/crypto/LimelightCryptoProvider'  
import { NvHttp } from '../entryability/http/NvHttp'  
import router from '@ohos.router'  
import { Alert } from './compoments/Loading'  
async function getResString(com: any, r: Resource): Promise<string> {  
 return await getContext(com).resourceManager.getStringValue(r)  
}  
@Component  
struct PcView {  
 @State detail: ComputerDetails = new ComputerDetails()  
 onDelete: (ComputerDetails) => void  
 @Builder  
 pcMenu() {  
 Menu() {  
 if (this.detail.pairState == PairState.PAIRED) {  
 MenuItem({ content: "浏览游戏列表" }).onClick(()=>{  
 this.doAppList(this.detail, false, false)  
 })  
 } else {  
 MenuItem({ content: "和电脑配对" }).onClick(()=>{  
 this.doPair()  
 })  
 }  
 MenuItem({ content: "测试网络连接" })  
 MenuItem({ content: "删除电脑" }).onClick(() => {  
 this.onDelete(this.detail)  
 })  
 MenuItem({ content: "查看详情" }).onClick((e) => {  
 let alertDialog = new CustomDialogController({  
 builder: Alert({  
 title: "查看详情",  
 message: this.detail.toString(),  
 }),  
 autoCancel: true,  
 alignment: DialogAlignment.Center,  
 customStyle: true  
 })  
 alertDialog.open()  
 })  
 }  
 }  
 click() {  
 if (this.detail.state == ComputerState.ONLINE && this.detail.pairState != PairState.PAIRED) {  
 this.doPair()  
 } else if(this.detail.state == ComputerState.ONLINE) {  
 this.doAppList(this.detail, false, false)  
 } else {  
 }  
 }  
 async doPair() {  
 let message;  
 let success = false;  
 const computer = this.detail  
 const httpConn = new NvHttp(this.detail.activeAddress,  
 this.detail.httpsPort, null,  
 limelightCertProvider);  
 const state = await httpConn.fetchPairState()  
 if (state == PairState.PAIRED) {  
 message = null;  
 success = true;  
 } else {  
 const pinStr = "12345";  
 const dialogText = await getResString(this, $r('app.string.pair\_pairing\_msg')) + " " + pinStr + "\n\n" + await getResString(this, $r('app.string.pair\_pairing\_help'))  
 let alertDialog = new CustomDialogController({  
 builder: Alert({  
 title: "配对中",  
 message: dialogText,  
 }),  
 autoCancel: false,  
 alignment: DialogAlignment.Center,  
 customStyle: true  
 })  
 alertDialog.open()  
 const pm = httpConn.pm  
 const pairState = await pm.pair(await httpConn.getServerInfo(true), pinStr);  
 if (pairState == PairState.PIN\_WRONG) {  
 message = await getResString(this, $r('app.string.pair\_incorrect\_pin'))  
 }  
 else if (pairState == PairState.FAILED) {  
 if (computer.runningGameId != 0) {  
 message = await getResString(this, $r('app.string.pair\_pc\_ingame'))  
 }  
 else {  
 message = await getResString(this, $r('app.string.pair\_fail'))  
 }  
 }  
 else if (pairState == PairState.ALREADY\_IN\_PROGRESS) {  
 message = await getResString(this, $r('app.string.pair\_already\_in\_progress'))  
 }  
 else if (pairState == PairState.PAIRED) {  
 message = null;  
 success = true;  
 this.detail.serverCert = true  
 viewModel.runPoll(this.detail, false)  
 }  
 else {  
 message = null;  
 }  
 alertDialog.close()  
 alertDialog = undefined  
 if (message)  
 promptAction.showToast({ message: message })  
 if (success){  
 this.doAppList(computer, true, false);  
 }  
 }  
 }  
 doAppList(computer: ComputerDetails , newlyPaired:boolean, showHiddenGames:boolean){  
 router.pushUrl({url:"pages/AppPage", params: { uuid: this.detail.uuid, computerName: this.detail.name, rawAppList: this.detail.rawAppList}})  
 }  
 aboutToAppear() {  
 }  
 build() {  
 Column() {  
 Stack({ alignContent: Alignment.Center }) {  
 Icon({ icon: $r('app.media.desktop\_windows'), iconSize: 120 })  
 if (this.detail.isLoading) {  
 LoadingProgress().width(50).height(50).color(Color.White).offset({ y: -10 })  
 } else {  
 if (this.detail.state == ComputerState.ONLINE) {  
 if (this.detail.pairState != PairState.PAIRED) {  
 Icon({ icon: $r('app.media.baseline\_lock'), iconSize: 48 }).offset({ y: -10 })  
 }  
 } else {  
 Icon({ icon: $r('app.media.baseline\_warning'), iconSize: 48 }).offset({ y: -10 })  
 }  
 }  
 }  
 Text(this.detail.name || "-").fontColor(Color.White)  
 }.onClick(() => {  
 this.click()  
 }).bindContextMenu(this.pcMenu, ResponseType.LongPress)  
 }  
}  
import taskpool from '@ohos.taskpool';  
@Entry  
@Component  
struct PcPage {  
 scroller: Scroller = new Scroller();  
 @State pcList: ComputerDetails[] = []  
 heightValue: number  
 gridRowTemplate: string  
 aboutToAppear() {  
 viewModel.getComputerList().then((list) => {  
 for(let d of list){  
 d.isLoading = true  
 }  
 this.pcList = list  
 viewModel.batchPollComputerList(list)  
 })  
 viewModel.onDetailsUpdate((news: ComputerDetails) => {  
 var indexes = this.pcList.findIndex((d) => d.uuid == news.uuid)  
 if (indexes < 0) {  
 this.pcList.push(news)  
 } else {  
 this.pcList[indexes] = news  
 }  
 this.updateGrid()  
 })  
 this.updateGrid()  
 }  
 updateGrid() {  
 var rows = Math.max(3, Math.round(this.pcList.length / 3))  
 this.gridRowTemplate = '1fr '.repeat(rows);  
 this.heightValue = rows \* 192 - 8;  
 }  
 build() {  
 Column() {  
 MainTitle()  
 Scroll(this.scroller) {  
 Grid() {  
 ForEach(this.pcList, (d) => {  
 GridItem() {  
 PcView({ detail: d, onDelete:(d)=>{  
 }})  
 }  
 }, (item) => JSON.stringify(item))  
 }.onKeyEvent((e)=>{  
 console.log(e.keyCode+"");  
 }).onMouse((e)=>{  
 console.log(e.button+"");  
 })  
 .rowsTemplate(this.gridRowTemplate)  
 .columnsTemplate('1fr 1fr 1fr')  
 .height(this.heightValue)  
 }.layoutWeight(1).scrollable(ScrollDirection.Vertical)  
 }.padding(20).height("100%").width("100%").backgroundColor($r("app.color.page\_background"))  
 }  
}

VirtualControllerSettings.ets

import { VirtualController, VirtualControllerBox } from '../virtual\_controller/VirtualController'  
@Entry  
@Component  
struct VirtualControllerSettings {  
 virtualController = new VirtualController()  
 aboutToAppear(){  
 }  
 build(){  
 Stack({alignContent:Alignment.TopStart}){  
 VirtualControllerBox({virtualController: this.virtualController, inputContext:this.virtualController.inputContext })  
 }  
 }  
}

ControllerHandle.ts

import { NvConnection } from '../entryability/nvstream/NvConnection';  
import { StreamSettings } from '../uitls/StreamSetttings';  
import { GenericControllerContext } from './context/GenericControllerContext';

import { InputDeviceContext } from './context/GenericControllerContext';  
export class ControllerHandle {  
 conn: NvConnection  
 prefConfig: StreamSettings  
 currentControllers: number  
 initialControllers: number  
 defaultContext: InputDeviceContext = new InputDeviceContext()  
 stickDeadzone: number  
 inputDeviceContexts: Map<number, InputDeviceContext> = new Map();  
 constructor(conn: NvConnection, prefConfig: StreamSettings) {  
 this.conn = conn  
 this.prefConfig = prefConfig  
 let deadzonePercentage = parseInt(prefConfig.seekbar\_deadzone);  
 if (isNaN(deadzonePercentage) || deadzonePercentage <= 0) {  
 deadzonePercentage = 1;  
 }  
 this.stickDeadzone = deadzonePercentage / 100.0;  
 this.defaultContext.leftStickXAxis = 0; //MotionEvent.AXIS\_X;  
 this.defaultContext.leftStickYAxis = 1; // MotionEvent.AXIS\_Y;  
 this.defaultContext.leftStickDeadzoneRadius = this.stickDeadzone;  
 this.defaultContext.rightStickXAxis = 11 //MotionEvent.AXIS\_Z;  
 this.defaultContext.rightStickYAxis = 14 //MotionEvent.AXIS\_RZ;  
 this.defaultContext.rightStickDeadzoneRadius = this.stickDeadzone;  
 this.defaultContext.leftTriggerAxis = 23 // MotionEvent.AXIS\_BRAKE;  
 this.defaultContext.rightTriggerAxis = 22 // MotionEvent.AXIS\_GAS;  
 this.defaultContext.hatXAxis = 15 //MotionEvent.AXIS\_HAT\_X;  
 this.defaultContext.hatYAxis = 16 // MotionEvent.AXIS\_HAT\_Y;  
 this.defaultContext.controllerNumber = 0;  
 this.defaultContext.assignedControllerNumber = true;  
 this.defaultContext.external = false;  
 }  
 reportOscState(buttonFlags: number,  
 leftStickX: number, leftStickY: number,  
 rightStickX: number, rightStickY: number,  
 leftTrigger, rightTrigger: number) {  
 const defaultContext = this.defaultContext  
 defaultContext.leftStickX = leftStickX;  
 defaultContext.leftStickY = leftStickY;  
 defaultContext.rightStickX = rightStickX;  
 defaultContext.rightStickY = rightStickY;  
 defaultContext.leftTrigger = leftTrigger;  
 defaultContext.rightTrigger = rightTrigger;  
 defaultContext.inputMap = buttonFlags;  
 this.sendControllerInputPacket(this.defaultContext);  
 }  
 private sendControllerInputPacket(originalContext: GenericControllerContext): void {  
 const conn = this.conn  
 const controllerNumber: number = originalContext.controllerNumber;  
 let inputMap: number = 0;  
 let leftTrigger: number = 0;  
 let rightTrigger: number = 0;  
 let leftStickX: number = 0;  
 let leftStickY: number = 0;  
 let rightStickX: number = 0;  
 let rightStickY: number = 0;  
 if (this.defaultContext.controllerNumber === controllerNumber) {  
 inputMap |= this.defaultContext.inputMap;  
 leftTrigger |= this.maxByMagnitude(leftTrigger, this.defaultContext.leftTrigger);  
 rightTrigger |= this.maxByMagnitude(rightTrigger, this.defaultContext.rightTrigger);  
 leftStickX |= this.maxByMagnitude(leftStickX, this.defaultContext.leftStickX);  
 leftStickY |= this.maxByMagnitude(leftStickY, this.defaultContext.leftStickY);  
 rightStickX |= this.maxByMagnitude(rightStickX, this.defaultContext.rightStickX);  
 rightStickY |= this.maxByMagnitude(rightStickY, this.defaultContext.rightStickY);  
 }  
 if (originalContext.mouseEmulationActive) {  
 } else {  
 conn.sendControllerInput(  
 controllerNumber,  
 this.getActiveControllerMask(),  
 inputMap,  
 leftTrigger,  
 rightTrigger,  
 leftStickX,  
 leftStickY,  
 rightStickX,  
 rightStickY  
 );  
 }  
 }  
 getActiveControllerMask(): number {  
 return 1  
 }  
 maxByMagnitude(a: number, b: number): number {  
 const absA = Math.abs(a);  
 const absB = Math.abs(b);  
 if (absA > absB) {  
 return a;  
 }  
 else {  
 return b;  
 }  
 }  
}

VirtualController.ets

import { AnalogStick, } from './AnalogStick';  
import { DigitalButton } from './DigitalButton';  
import mediaquery from '@ohos.mediaquery';  
import { ElementLayoutParam, VirtualControllerButton, VirtualControllerElement } from './common';  
import List from '@ohos.util.List';  
import { DigitalPad } from './DigitalPad';  
import { createLeftStick } from './VirtualControllerConfigurationLoader';

import { VirtualControllerConfigurationLoader } from './VirtualControllerConfigurationLoader';  
import Prompt from '@system.prompt';  
import { ControllerHandle } from './ControllerHandle';  
export enum ControllerMode {  
 Active,  
 MoveButtons,  
 ResizeButtons  
}  
@Observed  
export class ControllerInputContext {  
 public inputMap: number = 0x0000;  
 public leftTrigger: number = 0x00;  
 public rightTrigger: number = 0x00;  
 public rightStickX: number = 0x0000;  
 public rightStickY: number = 0x0000;  
 public leftStickX: number = 0x0000;  
 public leftStickY: number = 0x0000;  
}  
export class VirtualController {  
 controllerHandle: ControllerHandle  
 currentMode: ControllerMode = ControllerMode.Active  
 inputContext: ControllerInputContext = new ControllerInputContext();  
 elements: List<VirtualControllerElement> = new List()  
 constructor() {  
 }  
 addElement(element: VirtualControllerElement, x: number, y: number, width: number, height: number) {  
 element.setLayout(new ElementLayoutParam(x, y, width, height))  
 this.elements.add(element)  
 }  
 onSettingsClick(context: Context){  
 let message = ""  
 if (this.currentMode == ControllerMode.Active){  
 this.currentMode = ControllerMode.MoveButtons;  
 message = "Entering configuration mode (Move buttons)";  
 } else if (this.currentMode == ControllerMode.MoveButtons) {  
 this.currentMode = ControllerMode.ResizeButtons;  
 message = "Entering configuration mode (Resize buttons)";  
 } else {  
 this.currentMode = ControllerMode.Active;  
 VirtualControllerConfigurationLoader.saveProfile(this, context);  
 message = "Exiting configuration mode";  
 }  
 Prompt.showToast({message})  
 }  
 sendControllerInputContext(){  
 const inputContext = this.inputContext  
 if(!this.controllerHandle)  
 return null;  
 this.controllerHandle.reportOscState(inputContext.inputMap,  
 inputContext.leftStickX,  
 inputContext.leftStickY,  
 inputContext.rightStickX,  
 inputContext.rightStickY,  
 inputContext.leftTrigger,  
 inputContext.rightTrigger)  
 }  
 setOpacity(opactiy: number) {  
 }  
}  
@Builder  
export function ShowVirtualController(){  
 ForEach(this.virtualController.elements.convertToArray(), (d: VirtualControllerElement) => {  
 VirtualControllerButton({ element: d, layout: d.layout })  
 }, (d:VirtualControllerElement) =>d.elementId.toString())  
 Button(){  
 Text("setting")  
 }.offset({y: 80}).onClick(()=>{  
 this.virtualController.onSettingsClick(getContext(this))  
 })  
}  
@Component  
export struct VirtualControllerBox{  
 virtualController: VirtualController  
 @ObjectLink inputContext: ControllerInputContext  
 aboutToAppear() {  
 const loader = new VirtualControllerConfigurationLoader()  
 loader.createDefaultLayout(this.virtualController)  
 VirtualControllerConfigurationLoader.loadFromPreferences(this.virtualController, getContext(this))  
 }  
 build() {  
 Stack({ alignContent: Alignment.TopStart }) {  
 ShowVirtualController()  
 Text(JSON.stringify(this.inputContext)).alignSelf(ItemAlign.Center).offset({x: 50,y:0})  
 }.height('100%')  
 .width('100%')  
 }  
}

DigitalButton.ets

import {  
 drawLine,  
 pressedColor,  
 getPercent,  
 getHeight,  
 getWidth,  
 drawCircle,  
 getDefaultColor,  
 VirtualControllerElement  
} from './common';  
import { VirtualController } from './VirtualController';  
interface DigitalButtonListener {  
 onClick(): void;  
 onLongClick(): void;  
 onRelease(): void;  
}  
export class DigitalButton extends VirtualControllerElement {  
 private listeners: DigitalButtonListener[] = [];  
 layer: number  
 constructor(controller: VirtualController, elementId: number, layer: number) {  
 super(controller, elementId)  
 this.layer = layer  
 }  
 addDigitalButtonListener(listener: DigitalButtonListener): void {  
 this.listeners.push(listener);  
 }  
 onElementTouchEvent(event: TouchEvent) {  
 switch (event.type) {  
 case TouchType.Down: {  
 this.pressed = true  
 this.onClickCallback();  
 break;  
 }  
 case TouchType.Move: {  
 break;  
 }  
 case TouchType.Up: {  
 this.pressed = false  
 this.onReleaseCallback()  
 break;  
 }  
 }  
 }  
 onElementDraw(canvas: CanvasRenderingContext2D) {  
 canvas.clearRect(0, 0, this.context.width, this.context.height)  
 canvas.fillStyle = "#00000000"  
 const strokeWidth = 2  
 this.context.beginPath();  
 this.context.lineWidth = strokeWidth;  
 this.context.strokeStyle = this.pressed ? pressedColor : getDefaultColor(this.virtualController)  
 this.context.ellipse(getPercent(this.context.width, 50), getPercent(this.context.height, 50), getPercent(this.context.width, 50) - strokeWidth, getPercent(this.context.height, 50) - strokeWidth, 0, 0, Math.PI \* 2);  
 this.context.stroke()  
 if (this.text) {  
 this.context.textAlign = "center"  
 this.context.fillStyle = this.pressed ? pressedColor : getDefaultColor(this.virtualController)  
 const textSize = getPercent(this.context.width, 25)  
 this.context.font = vp2px(textSize) + 'px sans-serif';  
 this.context.fillText(this.text, getPercent(this.context.width, 50), getPercent(this.context.height, 65));  
 }  
 }  
 text: string = "A"  
 pressed: boolean = false  
 private onClickCallback() {  
 this.listeners.forEach(listener => listener.onClick());  
 }  
 private onLongClickCallback() {  
 this.listeners.forEach(listener => listener.onLongClick());  
 }  
 private onReleaseCallback() {  
 this.listeners.forEach(listener => listener.onRelease());  
 }  
}

DigitalPad.ets

import {  
 drawLine,  
 drawRect,  
 pressedColor,  
 getPercent,  
 getHeight,  
 getWidth,  
 getDefaultColor,  
 VirtualControllerElement  
} from './common';  
const DIGITAL\_PAD\_DIRECTION\_NO\_DIRECTION = 0;  
export const DIGITAL\_PAD\_DIRECTION\_LEFT: number = 1;  
export const DIGITAL\_PAD\_DIRECTION\_UP: number = 2;  
export const DIGITAL\_PAD\_DIRECTION\_RIGHT: number = 4;  
export const DIGITAL\_PAD\_DIRECTION\_DOWN: number = 8;  
interface DigitalPadListener {  
 onDirectionChange(direction:number);  
}  
export class DigitalPad extends VirtualControllerElement {  
 pressed: boolean = false  
 padDirection: number = DIGITAL\_PAD\_DIRECTION\_NO\_DIRECTION  
 DPAD\_MARGIN: number = 5;  
 private listeners: DigitalPadListener[] = [];  
 addDigitalPadListener(listener: DigitalPadListener): void {  
 this.listeners.push(listener);  
 }  
 private newDirectionCallback(direction: number) {  
 this.listeners.forEach(listener => listener.onDirectionChange(direction));  
 }  
 onElementTouchEvent(event: TouchEvent) {  
 switch (event.type) {  
 case TouchType.Down: {  
 break;  
 }  
 case TouchType.Move: {  
 let direction = 0;  
 let x = event.changedTouches[0].x  
 let y = event.changedTouches[0].y  
 if (x < getPercent(getWidth(this.context), 33)) {  
 direction |= DIGITAL\_PAD\_DIRECTION\_LEFT;  
 }  
 if (x > getPercent(getWidth(this.context), 66)) {  
 direction |= DIGITAL\_PAD\_DIRECTION\_RIGHT;  
 }  
 if (y > getPercent(getHeight(this.context), 66)) {  
 direction |= DIGITAL\_PAD\_DIRECTION\_DOWN;  
 }  
 if (y < getPercent(getHeight(this.context), 33)) {  
 direction |= DIGITAL\_PAD\_DIRECTION\_UP;  
 }  
 this.padDirection = direction  
 this.newDirectionCallback(direction);  
 break;  
 }  
 case TouchType.Cancel:  
 case TouchType.Up: {  
 this.padDirection = 0;  
 this.newDirectionCallback(this.padDirection);  
 break;  
 }  
 }  
 }  
 onElementDraw(canvas: CanvasRenderingContext2D) {  
 this.context.clearRect(0, 0, this.context.width, this.context.height)  
 const strokeWidth = 2  
 this.context.fillStyle = "#00000000"  
 this.context.lineWidth = 2;  
 if (this.padDirection == DIGITAL\_PAD\_DIRECTION\_NO\_DIRECTION) {  
 this.context.strokeStyle = getDefaultColor(this.virtualController)  
 drawRect(  
 this.context,  
 getPercent(getWidth(this.context), 36), getPercent(getHeight(this.context), 36),  
 getPercent(getWidth(this.context), 63), getPercent(getHeight(this.context), 63),  
 );  
 }  
 this.context.strokeStyle = (this.padDirection & DIGITAL\_PAD\_DIRECTION\_LEFT) > 0 ? pressedColor : getDefaultColor(this.virtualController)  
 drawRect(  
 this.context,  
 strokeWidth + this.DPAD\_MARGIN,  
 getPercent(getHeight(this.context), 33),  
 getPercent(getWidth(this.context), 33),  
 getPercent(getHeight(this.context), 66)  
 );  
 this.context.strokeStyle = (this.padDirection & DIGITAL\_PAD\_DIRECTION\_UP) > 0 ? pressedColor : getDefaultColor(this.virtualController)  
 drawRect(  
 this.context,  
 getPercent(getWidth(this.context), 33),

strokeWidth + this.DPAD\_MARGIN,  
 getPercent(getWidth(this.context), 66),

getPercent(getHeight(this.context), 33),  
 );  
 this.context.strokeStyle = (this.padDirection & DIGITAL\_PAD\_DIRECTION\_RIGHT) > 0 ? pressedColor : getDefaultColor(this.virtualController)  
 drawRect(  
 this.context,  
 getPercent(getWidth(this.context), 66),

getPercent(getHeight(this.context), 33),  
 getWidth(this.context) - (strokeWidth + this.DPAD\_MARGIN),  
 getPercent(getHeight(this.context), 66),  
 );  
 this.context.strokeStyle = (this.padDirection & DIGITAL\_PAD\_DIRECTION\_DOWN) > 0 ? pressedColor : getDefaultColor(this.virtualController)  
 drawRect(  
 this.context,  
 getPercent(getWidth(this.context), 33),

getPercent(getHeight(this.context), 66),  
 getPercent(getWidth(this.context), 66),

getHeight(this.context) - (strokeWidth + this.DPAD\_MARGIN),  
 );  
 this.context.strokeStyle = ((this.padDirection & DIGITAL\_PAD\_DIRECTION\_LEFT) > 0 &&  
 (this.padDirection & DIGITAL\_PAD\_DIRECTION\_UP) > 0) ? pressedColor : getDefaultColor(this.virtualController)  
 drawLine(this.context,  
 strokeWidth + this.DPAD\_MARGIN,  
 getPercent(getHeight(this.context), 33),  
 getPercent(getWidth(this.context), 33),  
 strokeWidth + this.DPAD\_MARGIN,  
 )  
 this.context.strokeStyle = ((this.padDirection & DIGITAL\_PAD\_DIRECTION\_UP) > 0 &&  
 (this.padDirection & DIGITAL\_PAD\_DIRECTION\_RIGHT) > 0) ? pressedColor : getDefaultColor(this.virtualController)  
 drawLine(this.context,  
 getPercent(getWidth(this.context), 66),  
 strokeWidth + this.DPAD\_MARGIN,  
 getWidth(this.context) - (strokeWidth + this.DPAD\_MARGIN),  
 getPercent(getHeight(this.context), 33)  
 )  
 this.context.strokeStyle = ((this.padDirection & DIGITAL\_PAD\_DIRECTION\_RIGHT) > 0 &&  
 (this.padDirection & DIGITAL\_PAD\_DIRECTION\_DOWN) > 0) ? pressedColor : getDefaultColor(this.virtualController)  
 drawLine(this.context,  
 getWidth(this.context) - (strokeWidth + this.DPAD\_MARGIN),  
 getPercent(getHeight(this.context), 66),  
 getPercent(getWidth(this.context), 66),  
 getHeight(this.context) - (strokeWidth + this.DPAD\_MARGIN),  
 )  
 this.context.strokeStyle = ((this.padDirection & DIGITAL\_PAD\_DIRECTION\_DOWN) > 0 &&  
 (this.padDirection & DIGITAL\_PAD\_DIRECTION\_LEFT) > 0) ? pressedColor : getDefaultColor(this.virtualController)  
 drawLine(this.context,  
 getPercent(getWidth(this.context), 33),  
 getHeight(this.context) - (strokeWidth + this.DPAD\_MARGIN),  
 strokeWidth + this.DPAD\_MARGIN,  
 getPercent(getHeight(this.context), 66),  
 )  
 }  
}

common.ets

import { VirtualController } from './VirtualController';

import { ControllerMode} from './VirtualController';  
export function getPercent(value: number, percent: number): number {  
 return value / 100 \* percent;  
}  
export function getCorrectWidth(context: CanvasRenderingContext2D): number {  
 return context.width > context.height ? context.height : context.width;  
}  
export function drawCircle(context: CanvasRenderingContext2D, x: number, y: number, radius: number) {  
 context.beginPath();  
 context.arc(x, y, radius, 0, Math.PI \* 2, true);  
 context.stroke()  
}  
export function drawRect(context: CanvasRenderingContext2D, left: number, top: number, right: number, bottom: number, fill: boolean = false) {  
 if (fill) {  
 context.fillRect(left, top, right - left, bottom - top)  
 } else {  
 context.strokeRect(left, top, right - left, bottom - top)  
 }  
}  
export function drawLine(context: CanvasRenderingContext2D, left: number, top: number, right: number, bottom: number) {  
 context.beginPath();  
 context.moveTo(left, top);  
 context.lineTo(right, bottom);  
 context.stroke();  
}  
export function getWidth(context): number {  
 return context.width  
}  
export function getHeight(context): number {  
 return context.height  
}  
const normalColor = "#F0888888";  
export const pressedColor = "#F00000FF";  
const configMoveColor = "#F0FF0000";  
const configResizeColor = "#F0FF00FF";  
const configSelectedColor = "#F000FF00";  
export function getDefaultColor(virtualController: VirtualController) {  
 if (virtualController.currentMode == ControllerMode.MoveButtons)  
 return configMoveColor;  
 else if (virtualController.currentMode == ControllerMode.ResizeButtons)  
 return configResizeColor;  
 else  
 return normalColor;  
}  
enum Mode {  
 Normal,  
 Resize,  
 Move  
}  
@Observed  
export class ElementLayoutParam {  
 x: number;  
 y: number;  
 width: number;  
 height: number;  
 constructor(x: number, y: number, width: number, height: number) {  
 this.x = x  
 this.y = y  
 this.width = width  
 this.height = height  
 }  
}  
export abstract class VirtualControllerElement {  
 public static EID\_DPAD = 1;  
 public static EID\_LT = 2;  
 public static EID\_RT = 3;  
 public static EID\_LB = 4;  
 public static EID\_RB = 5;  
 public static EID\_A = 6;  
 public static EID\_B = 7;  
 public static EID\_X = 8;  
 public static EID\_Y = 9;  
 public static EID\_BACK = 10;  
 public static EID\_START = 11;  
 public static EID\_LS = 12;  
 public static EID\_RS = 13;  
 public static EID\_LSB = 14;  
 public static EID\_RSB = 15;  
 elementId: number  
 virtualController: VirtualController  
 context: CanvasRenderingContext2D  
 normalColor = "#F0888888";  
 pressedColor = "#F00000FF";  
 private configMoveColor = "#0xF0FF0000";  
 private configResizeColor = "#0xF0FF00FF";  
 private configSelectedColor = "#0xF000FF00";  
 protected startSize\_x: number;  
 protected startSize\_y: number;  
 position\_pressed\_x: number = 0;  
 position\_pressed\_y: number = 0;  
 private currentMode = Mode.Normal;  
 layout: ElementLayoutParam  
 constructor(controller: VirtualController, elementId: number) {  
 this.virtualController = controller;  
 this.elementId = elementId;  
 }  
 setLayout(layout: ElementLayoutParam) {  
 this.layout = layout;  
 }  
 getWidth(): number {  
 return this.context.width  
 }  
 getHeight(): number {  
 return this.context.height  
 }  
 getDefaultStrokeWidth(): number {  
 return 2  
 }  
 resizeElement: (pressed\_x, pressed\_y, width, height) => void = (pressed\_x, pressed\_y, width, height) => {  
 const layoutParams = this.layout  
 let newHeight = height + (this.startSize\_y - pressed\_y);  
 let newWidth = width + (this.startSize\_x - pressed\_x);  
 layoutParams.height = newHeight > 20 ? newHeight : 20;  
 layoutParams.width = newWidth > 20 ? newWidth : 20;  
 }  
 moveElement: (pressed\_x, pressed\_y, x, y) => void = (pressed\_x, pressed\_y, x, y) => {  
 const layoutParams = this.layout  
 const dx = x - pressed\_x;  
 const dy = y - pressed\_y;  
 layoutParams.x += dx  
 layoutParams.y += dy  
 this.position\_pressed\_x += dx;  
 this.position\_pressed\_y += dy;  
 }  
 abstract onElementDraw(canvas: CanvasRenderingContext2D);  
 abstract onElementTouchEvent(event: TouchEvent)  
 onDraw(canvas: CanvasRenderingContext2D) {  
 this.context = canvas  
 this.onElementDraw(canvas);  
 if (this.currentMode != Mode.Normal) {  
 canvas.strokeStyle = configSelectedColor  
 const strokeWidth = this.getDefaultStrokeWidth()  
 canvas.lineWidth = strokeWidth  
 drawRect(canvas, strokeWidth, strokeWidth, getWidth(canvas) - strokeWidth, getHeight(canvas) - strokeWidth)  
 }  
 }  
 actionEnableMove() {  
 this.currentMode = Mode.Move;  
 }  
 actionEnableResize() {  
 this.currentMode = Mode.Resize;  
 }  
 actionCancel() {  
 this.currentMode = Mode.Normal;  
 }  
 getConfiguration():string {  
 return JSON.stringify(this.layout)  
 }  
 loadConfiguration(configuration: string){  
 Object.assign(this.layout, JSON.parse(configuration))  
 }  
 onSizeChanged(canvas: CanvasRenderingContext2D) {  
 }  
 touchId: number = -1  
 onTouchEvent(event: TouchEvent) {  
 if (this.virtualController.currentMode == ControllerMode.Active) {  
 return this.onElementTouchEvent(event);  
 }  
 let touch = null  
 if(this.touchId > -1){  
 touch = event.changedTouches.find((t)=>t.id == this.touchId)  
 if (touch == null)  
 return;  
 }  
 switch (event.type) {  
 case TouchType.Down: {  
 if(this.touchId == -1){  
 touch = event.changedTouches[0]  
 this.touchId = touch.id  
 }  
 this.position\_pressed\_x = touch.x;  
 this.position\_pressed\_y = touch.y;  
 this.startSize\_x = this.getWidth();  
 this.startSize\_y = this.getHeight();  
 if (this.virtualController.currentMode == ControllerMode.MoveButtons)  
 this.actionEnableMove();  
 else if (this.virtualController.currentMode == ControllerMode.ResizeButtons)  
 this.actionEnableResize();  
 return true;  
 }  
 case TouchType.Move: {  
 switch (this.currentMode) {  
 case Mode.Move: {  
 this.moveElement(  
 this.position\_pressed\_x,  
 this.position\_pressed\_y,  
 touch.x,  
 touch.y);  
 break;  
 }  
 case Mode.Resize: {  
 this.resizeElement(  
 this.position\_pressed\_x,  
 this.position\_pressed\_y,  
 touch.x,  
 touch.y);  
 break;  
 }  
 case Mode.Normal: {  
 break;  
 }  
 }  
 return true;  
 }  
 case TouchType.Cancel:{}  
 case TouchType.Up: {  
 this.touchId = -1  
 this.actionCancel();  
 return true;  
 }  
 default: {  
 }  
 }  
 return true;  
 }  
}  
@Component  
export struct VirtualControllerButton {  
 private settings: RenderingContextSettings = new RenderingContextSettings(true)  
 private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)  
 element: VirtualControllerElement  
 @ObjectLink layout: ElementLayoutParam  
 onLayout() {  
 this.element.onSizeChanged(this.context)  
 }  
 build() {  
 Canvas(this.context)  
 .height(this.layout.height)  
 .width(this.layout.width)  
 .translate({ x: this.layout.x, y: this.layout.y })  
 .opacity(1)  
 .onReady(() => {  
 this.element.onDraw(this.context)  
 })  
 .onTouch((e) => {  
 this.element.onTouchEvent(e)  
 this.element.onDraw(this.context)  
 })  
 }  
}