

密级状态：绝密( ) 秘密( ) 内部( ) 公开(√)

# Rockchip API Reference SVEP SR

(图形计算平台中心)

文件状态： [ ] 正在修改 [√] 正式发布	文档标识：	RK-KF-YF-409
	当前版本：	2.2.0
	作 者：	李斌
	完成日期：	2024-08-06
	审 核：	熊伟
	完成日期：	2024-08-06

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

(版本所有,翻版必究)

Rockchip

## 免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

## 商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。  
本文档可能提及的其他所有注册商标或商标，由其各自所有者所有。

## 版权所有 © 2024 瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址：福建省福州市铜盘路软件园 A 区 18 号

网址： [www.rock-chips.com](http://www.rock-chips.com)

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： [fae@rock-chips.com](mailto:fae@rock-chips.com)

## 更新记录

版本	修改人	修改日期	修改说明	核定人
V1.9.1	李斌	2023-06-07	<b>【优化】</b> 优化 SR 系统负载 <b>【新增】</b> 完成 Linux 说明文档	熊伟
V2.1.0	李斌	2023-12-12	<b>【优化】</b> 优化用户调用接口 <b>【优化】</b> RK3588 支持 SRAM 优化 <b>【新增】</b> 支持 RK3568/RK3566 平台 <b>【修复】</b> 修复内部稳定性问题	熊伟
V2.2.0	李斌	2024-08-06	<b>【新增】</b> 支持 RK3576 平台 <b>【新增】</b> 支持 SR 后级缩放与色域转换 <b>【新增】</b> 支持 UFS 授权码 <b>【优化】</b> 优化算法处理耗时 <b>【修复】</b> 修复内部稳定性问题 <b>【修改】</b> 移除对 librknrt-svep.so 依赖	熊伟

# 目 录

<b>1 SVEP 简介 .....</b>	<b>5</b>
1.1 软件架构 .....	5
1.2 支持模式 .....	6
1.2.1 RK3588 .....	6
1.2.2 RK3568/RK3566 .....	7
1.2.3 RK3576 .....	8
<b>2 软件授权说明 .....</b>	<b>10</b>
2.1 授权码申请 .....	10
2.2 授权码存储位置 .....	10
2.2.1 简易授权码导入工具 .....	10
2.3 授权验证 .....	11
2.3.1 快速授权验证方法 .....	11
2.4 授权实现流程 .....	12
<b>3 软件流程说明 .....</b>	<b>13</b>
3.1 处理流程图说明 .....	13
3.2 用户调用流程图 .....	14
<b>4 数据结构说明 .....</b>	<b>16</b>
4.1 SrImageInfo 结构 .....	17
4.2 SrBufferInfo 结构 .....	17
4.3 SrRect 结构 .....	20
<b>5 用户接口说明 .....</b>	<b>21</b>
5.1 SvpSr .....	21
5.1.1 头文件说明 .....	21

5.1.2 接口应用举例 .....	21
5.2 Init .....	21
5.2.1 头文件说明 .....	21
5.2.2 接口应用举例 .....	22
5.2.3 常见日志说明 .....	22
5.3 SetEnhancementRate .....	24
5.3.1 头文件说明 .....	24
5.3.2 接口应用举例 .....	24
5.3.3 效果举例 .....	24
5.4 SetOsdMode .....	25
5.4.1 头文件说明 .....	25
5.4.2 接口应用举例 .....	25
5.4.3 效果展示 .....	26
5.5 SetContrastMode .....	26
5.5.1 头文件说明 .....	26
5.5.2 接口应用举例 .....	27
5.5.3 效果展示 .....	27
5.6 SetRotateMode .....	28
5.6.1 头文件说明 .....	28
5.6.2 接口应用举例 .....	28
5.7 MatchSrMode .....	29
5.7.1 头文件说明 .....	29
5.7.2 接口应用举例 .....	29
5.7.3 常见日志说明 .....	30
5.7.4 效果展示 .....	31
5.8 GetDetImageInfo .....	32

5.8.1 头文件说明 .....	32
5.8.2 接口应用举例 .....	32
5.9 Run .....	32
5.9.1 头文件说明 .....	32
5.9.2 接口应用举例 .....	33
5.9.3 内部处理流程说明 .....	34
5.9.4 常见日志说明 .....	34
5.10 RunAsync .....	35
5.10.1 头文件说明 .....	35
5.10.2 接口应用举例 .....	36
5.10.3 内部流程处理说明 .....	37
<b>6 测试用例说明 .....</b>	<b>38</b>
6.1 编译说明 .....	38
6.1.1 Android .....	38
6.1.2 Linux .....	38
6.2 使用说明 .....	39
6.2.1 参数说明 .....	39
6.2.2 执行说明 .....	41
<b>7 调试日志说明 .....</b>	<b>46</b>
7.1 命令说明 .....	46
7.1.1 Android .....	46
7.1.2 Linux .....	46
7.2 默认输出日志 .....	46
7.3 Info 日志 .....	47
<b>8 SRAM 优化 .....</b>	<b>49</b>

8.1 SRAM 优化对比 .....	49
8.2 SRAM 优化补丁 .....	49
<b>9 常见问题 .....</b>	<b>51</b>
9.1 简单介绍 SR 具体做了哪些处理? .....	51
9.2 SR 效果的评价手段和指标有哪些? .....	51
9.3 非标准分辨率输出存在绿边问题 .....	51
9.4 最新版本获取 .....	53



# 1 SVEP 简介

SVEP（Super Vision Enhancement Process，超级视觉增强处理），是一项利用深度学习实现的图像增强处理技术，目前实现算法主要有 SR 与 MEMC 两类：

- **SR（Super Resolution，超级分辨率）**，SR 算法利用深度神经网络补充图片纹理细节，将原始低分辨率输入重建为清晰高分辨输出，以提升视频清晰度获得更高主观感知。
- **MEMC（Motion Estimation and Motion Compensation，运动补偿）**，MEMC 算法利用深度神经网络从前后两帧原始帧计算出中间的预测帧，提高视频帧率以获得较原始视频更流畅的感官体验。

SVEP 算法主要面向电视盒子、教育平板等领域，提供对在线、离线图像的超分辨率（SR）和运动补偿（MEMC）图像增强功能，提高视觉效果。

## 1.1 软件架构

SR 软件架构如下图 1-1 所示：

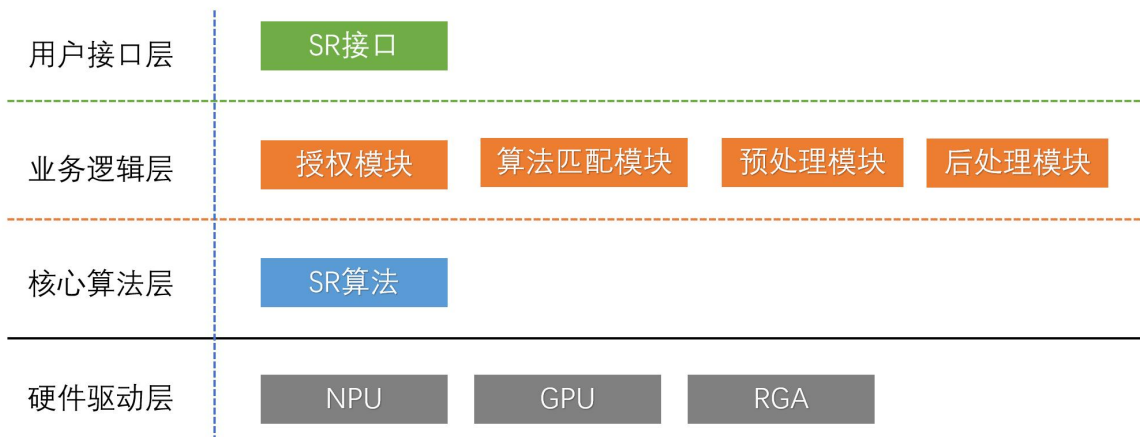


图 1-1 SR 软件架构图

- **用户接口层**：主要提供简洁的 SR 用户调用接口，也是本文档介绍的重点；
- **业务逻辑层**：主要由授权模块、算法匹配模块组成、预处理模块，后处理模块组成：
  - **授权模块**：主要用于算法软件授权，未经授权不允许调用相关核心算法；

- **算法匹配模块：**主要针对输入图像参数进行算法匹配；
- **预处理模块：**主要对输入图像进行图像预处理，使输入数据满足算法相关约束；
- **算法处理模块：**主要调用核心算法进行图像处理；
- **后处理模块：**主要对算法输出模块进行图像后处理，例如添加 OSD，实现对比模式等；
- **核心算法层：**提供 SR 算法实现；
- **硬件驱动层：**底层驱动提供 NPU/GPU/RGA 硬件支持；

## 1.2 支持模式

### 1.2.1 RK3588

#### 1.2.1.1 算法支持模式

SR 算法针对不同输入分辨率提供专门的优化模式，目前支持的 SR 模式如下表 1-1 所示：

表 1-1 RK3588 SR 算法模式支持列表

SR 模式	输入分辨率	输出分辨率	实时支持帧率	放大倍数
480p	768x480	2304x1440	30	3.0
540p	960x540	2880x1620	30	3.0
720p	1280x720	3840x2160	30	3.0
1080p	1920x1080	3840x2160	30	2.0
2160p	3840x2160	3840x2160	30	1.0
4320p	3840x2160	7680x4320	30	2.0

**注意：**若输入分辨率不满足标准 SR 模型的分辨率，系统端会适配到更大一级的 SR 模型 进行画质增强。例如：若输入 1024x600 分辨率视频，则采用 1280x720 SR 模型进行处理。利用此方法执行 SR 操作的输出 Buffer 将会出现绿边，需要用后续业务流程进行裁剪读取有效图像区域，详情见[“8.3 非标准分辨率输出存在绿边问题”](#)

### 1.2.1.2 色域空间支持

RK3588 SR 算法色域空间支持情况，如下表 1-2 所示：

表 1-2 RK3588 SR 算法色域空间支持情况

输入色域空间	RGB 格式支持情况	YUV 格式支持情况	YUV 输出色域空间	RGB 输出色域色域
BT601 Limit Range	不存在	支持	BT601 Limit Range	BT709 Limit/Full
BT601 Full Range	不存在	支持	BT601 Limit Range	BT709 Limit/Full
BT709 Limit Range	不存在	支持	BT709 Limit Range	BT709 Limit/Full
BT709 Full Range	支持	支持	BT709 Limit Range	BT709 Limit/Full
BT2020	不支持	不支持	不支持	不支持

### 1.2.2 RK3568/RK3566

RK3568/RK3566 以下统称 RK356X。

#### 1.2.2.1 算法支持模式

SR 算法针对不同输入分辨率提供专门的优化模式，目前支持的 SR 模式如下表 1-3 所示：

表 1-3 RK356X SR 算法模式支持列表

SR 模式	输入分辨率	输出分辨率	实时支持帧率	放大倍数
480p	864x480	1728x960	30	2.0
720p	1280x720	1920x1080	30	1.5
1080p	1920x1080	1920x1080	30	1.0

**注意：**若输入分辨率不满足标准 SR 模型的分辨率，系统端会适配到更大一级的 SR 模型 进行画质增强。例如：若输入 1024x600 分辨率视频，则采用 1280x720 SR 模型进行处理。利用此方法执行 SR 操作的输出 Buffer 将会出现绿边，需要用后续业务流程进行裁剪读取有效图像区域，详情见[“8.3 非标准分辨率输出存在绿边问题”](#)

### 1.2.2.2 色域空间支持

RK356X SR 算法色域空间支持情况，如下表 1-4 所示：

表 1-4 RK356X 算法色域空间支持情况

输入色域空间	RGB 格式支持情况	YUV 格式支持情况	YUV 输出色域空间	RGB 输出色域色域
BT601 Limit Range	不存在	支持	BT601 Limit Range	BT709 Limit/Full
BT601 Full Range	不存在	支持	BT601 Limit Range	BT709 Limit/Full
BT709 Limit Range	不存在	支持	BT709 Limit Range	BT709 Limit/Full
BT709 Full Range	支持	支持	BT709 Limit Range	BT709 Limit/Full
BT2020	不支持	不支持	不支持	不支持

## 1.2.3 RK3576

### 1.2.3.1 算法支持模式

SR 算法针对不同输入分辨率提供专门的优化模式，目前支持的 SR 模式如下表 1-5 所示：

表 1-5 RK3576 SR 算法模式支持列表

SR 模式	输入分辨率	输出分辨率	实时支持帧率	放大倍数
480p	768x480	2304x1440	30	3.0
540p	960x540	2880x1620	30	3.0
720p	1280x720	3840x2160	30	3.0
1080p	1920x1080	3840x2160	30	2.0

**注意：**若输入分辨率不满足标准 SR 模型的分辨率，系统端会适配到更大一级的 SR 模型 进行画质增强。例如：若输入 1024x600 分辨率视频，则采用 1280x720 SR 模型进行处理。利用此方法执行 SR 操作的输出 Buffer 将会出现绿边，需要用后续业务流程进行裁剪读取有效图像区域，详情见[“8.3 非标准分辨率输出存在绿边问题”](#)

### 1.2.3.2 色域空间支持

RK3576 SR 算法色域空间支持情况，如下表 1-6 所示：

表 1-6 RK3576 算法色域空间支持情况

输入色域空间	RGB 格式支持情况	YUV 格式支持情况	YUV 输出色域空间	RGB 输出色域色域
BT601 Limit Range	不存在	支持	BT601 Limit Range	BT709 Limit/Full
BT601 Full Range	不存在	支持	BT601 Limit Range	BT709 Limit/Full
BT709 Limit Range	不存在	支持	BT709 Limit Range	BT709 Limit/Full
BT709 Full Range	支持	支持	BT709 Limit Range	BT709 Limit/Full
BT2020	不支持	不支持	不支持	不支持

## 2 软件授权说明

SR 算法须要软件授权才能使用，目前是采用激活码授权方式进行终端网络授权。

### 2.1 授权码申请

SR 授权码可联系 RK 业务申请。

### 2.2 授权码存储位置

在获得 SR 算法授权码后，需要将授权码导入设备指定的储存位置，SR 算法接口库初始化时才可进行算法鉴权。目前授权码设计存储在芯片 VendorStorage 分区，分区 ID=60，关于 VendorStorage 分区可参考以下路径文档，进行阅读了解：

[libsvepsr/docs/Rockchip\\_Application\\_Notes\\_Storage\\_CN.pdf](#)

生产的授权码写码工具请参考 [Rockchip\\_Application\\_Notes\\_Storage\\_CN.pdf](#) 文档，用户自行开发。

#### 2.2.1 简易授权码导入工具

本章节提供本地测试的简易写授权码工具 `vendor-storage-test`。

工具名称：`vendor-storage-test`，源码地址位于：

[libsvepsr/tools/vendor-storage-test](#)

工具帮助信息：直接执行 `vendor-storage-test` 即可输出以下日志

```
$ adb shell vendor-storage-test
Usage : vendor_storage id len [code]
Config Code : vendor_storage 17 50 71581714-9736-4EEE-C029-B...
Read Code   : vendor_storage 17 50
```

授权码导入说明：超分授权码存放 ID 为 60，长度为 50 char，命令执行如下：

```
$ adb shell vendor-storage-test 60 50 \
71581714-9736-4EEE-C029-B65E54280A91
```

导入日志：导入后，输出日志如下：

```
$ vendor-storage-test 60 50 71581714-9736-4EEE-C029-B65E54280A91
Write Success: id=60 len=50 code \
71581714-9736-4EEE-C029-B65E54280A91 (EMMC)
Read Success: id=60 len=50 code = \
```

71581714-9736-4EEE-C029-B65E54280A91 (EMMC)

## 2.3 授权验证

确保设备端满足以下条件后，即可进行授权验证：

1. 设备端 VendorStorage ID=60 存在可靠授权码；
2. 设备已连接互联网；

### 2.3.1 快速授权验证方法

本地快速搭建授权验证环境可利用 libsvpsr 工具提供的 sr\_demo 工具，源码位于以下路径：

```
libsvpsr/examples/sr-demo
```

利用以下命令进行授权验证：

```
$ sr_demo -i 1280x720+0+0:1280x720@NV12
```

授权情况日志可通过以下命令打印：

```
$ logcat -s SvpAuth
```

首次授权需要联网，授权成功输出日志如下图 2-1 所示：

```
rk3588_box:/ # logcat -s SvpAuth
----- beginning of main
12-01 07:45:12.714 15299 I SvpAuth: ReadAuthCode,line=475, CheckAuth: AuthCode: tag=0x56524551 id=60 len=50 Code: 3FE6CC5D-7BA6-54F8-99DC-1EA798EED465
12-01 07:45:12.852 15299 I SvpAuth: DoAuth,line=380, CheckAuth: rkauth_activate2 auth code success, write license to /data/system/svp_key.lic
```

图 2-1 初次授权成功日志

授权成功后，SR 算法库会将离线的授权文件保存在以下路径：

```
/data/system/svp_key.lic
```

再次授权会直接校验离线的授权文件，授权成功后日志如下图 2-2 所示：

```
rk3588_box:/ # logcat -s SvpAuth
----- beginning of main
12-01 07:48:56.950 15376 I SvpAuth: DoAuth,line=359, CheckAuth: rkauth_verify_license /data/system/svp_key.lic Success. ret=RKAUTH_OK
```

图 2-2 再次授权成功日志

**注意：**由于离线授权文件需要存在在 /data/system/ 目录，若设备端没有这个目录建议提前创建，否则离线授权文件将无法输出；

**注意：**授权码授权成功后，将与 CPU efuse 绑定，已授权的激活码无法再次授权第二颗 CPU，请妥善使用授权码资源。

## 2.4 授权实现流程

SR 算法授权内部实现流程如下图 2-3 所示：

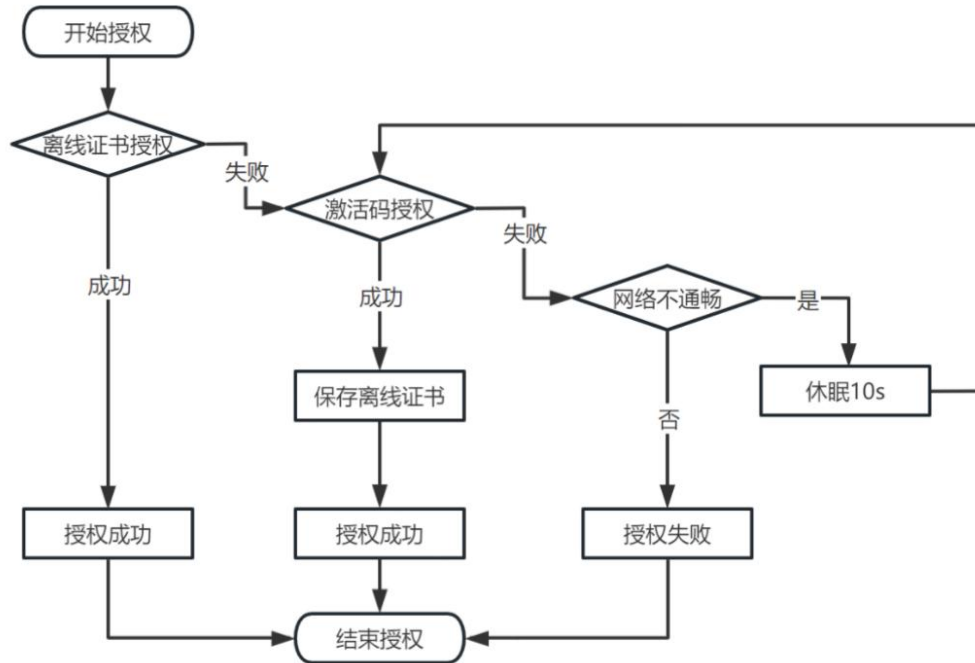


图 2-3 授权实现流程图

流程说明如下：

1. **离线证书授权：**初次授权由于离线证书不存在，则离线证书授权失败，进入激活码授权流程；
2. **激活码授权：**获取 VendorStorage ID=60 激活码，进行网络授权：
  - (1) 若由于网络原因导致授权失败，则休眠 10s 再尝试激活码授权；
  - (2) 若激活码无效则直接判断授权失败；
3. **保存离线证书：**初次授权成功后，系统会将离线授权证书保存至 /data/system/svep\_key.lic 位置，用于下次离线授权；
4. **授权成功：**完成授权。



## 3 软件流程说明

### 3.1 处理流程图说明

算法处理流程图如下图 3-1 所示：

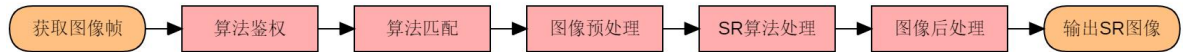


图 3-1 算法处理流程图

流程图说明如下：

1. **获取图像帧：**用户设置需要 SR 处理的图像数据
2. **算法鉴权：**算法鉴权通过才允许调用 SR 相关算法；
3. **算法匹配：**根据输入图像参数进行算法模型匹配；
4. **图像预处理：**执行输入图像进行图像预处理，使输入数据满足算法相关约束；
5. **SR 算法处理：**调用核心 SR 算法进行图像处理；
6. **图像后处理：**执行 SR 图像后处理，例如添加 OSD，实现对比模式等；
7. **输出 SR 图像。**

## 3.2 用户调用流程图

用户调用 SR 接口流程图如下图 3-2 所示：

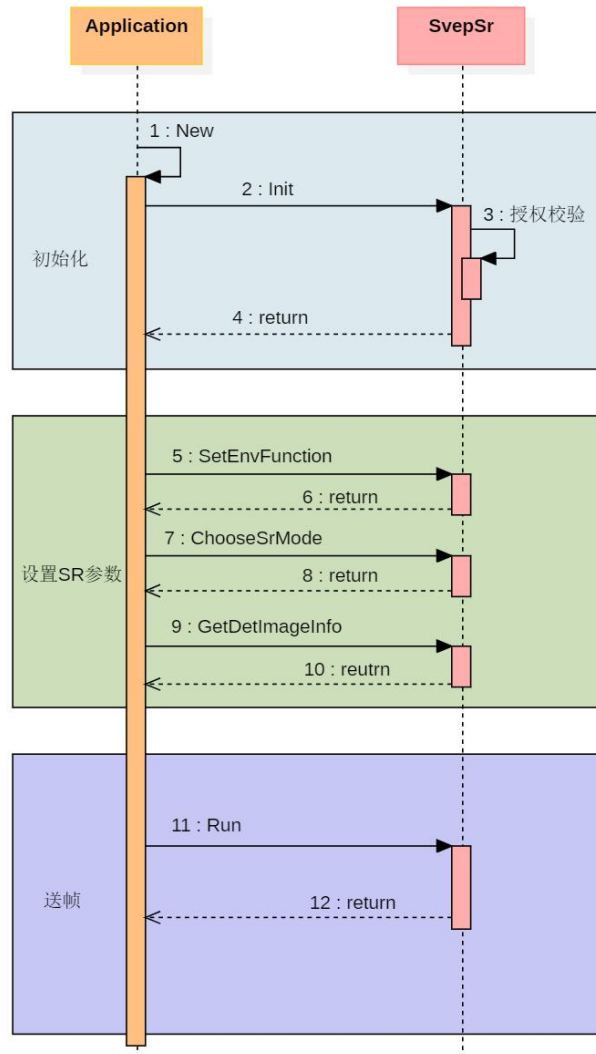


图 3-2 用户调用流程图

用户调用步骤说明：

1. 实例化 SvpSr 接口调用类；
2. 调用 Init 方法，尝试通过授权校验，授权通过校验后初始化成功；
3. 设置 SR 环境变量函数（SetEnv 函数）；
4. 根据输入图像参数匹配 SR 分辨率模式；
5. 获取目标图像参数，申请目标图像缓冲区；
6. 请求 SvpSr 执行 SR 请求；

相关函数接口如下表 3-1 所示：

表 3-1 函数接口表

函数名	详细说明
SvepSr()	Sr 构造函数
Init(const char *version_str, ..)	Sr 初始化函数
SetEnhancementRate(int rate);	SetEnv 函数：设置 SR 增强强度
SetOsdMode(SrOsdMode mode, ..)	SetEnv 函数：设置 OSD 字幕模式
SetContrastMode(bool enable, ..)	SetEnv 函数：设置对比模式
SetRotateMode(SrRotateMode rotate);	SetEnv 函数：设置旋转模式
MatchSrMode(const SrImageInfo *int_src, ..);	SetEnv 函数：匹配 SR 处理模型
GetDstImageInfo(SrImageInfo *out_dst)	SetEnv 函数：获取目标图像参数
Run(const SrImageInfo *int_src, ..)	同步处理模式，SR 执行完成后返回
RunAsync(const SrImageInfo *int_src, ..)	异步处理模式，SR 图像预处理后立即返回

## 4 数据结构说明

了解图像数据结构之前，先了解有关图像内存描述信息：

- Width / Height / Stride / HeightStride 信息；
- Crop 包含的 Left / Top / Right / Bottom 信息；

详细说明请参考下图 4-1 所示：

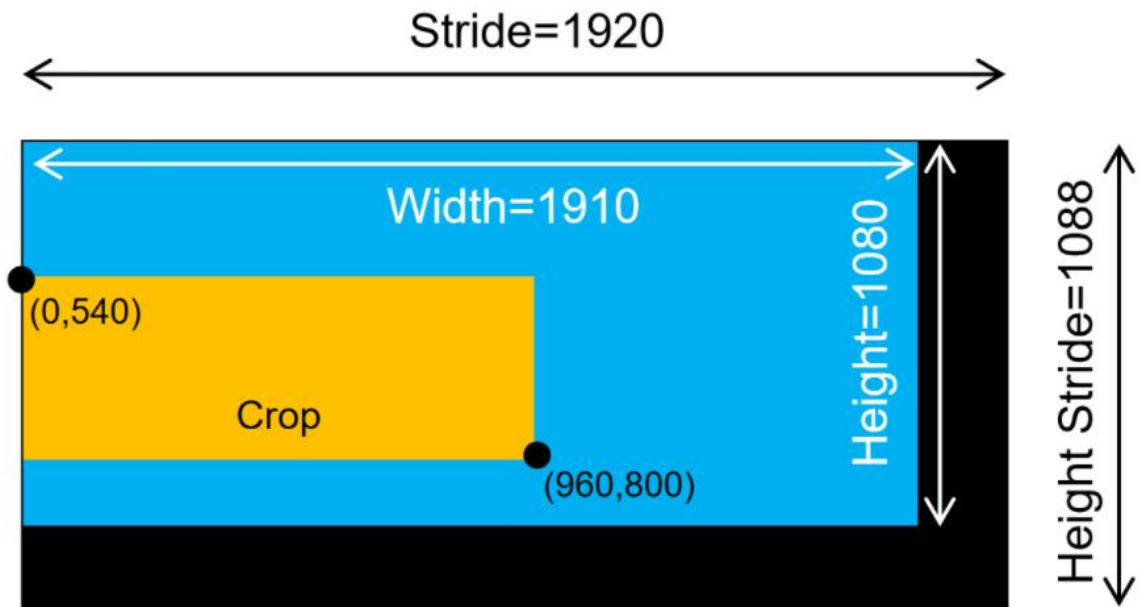


图 4-1 图像内存描述信息图

- **蓝色区域：**有效图像的矩形区域，Width=1910，Height=1080；
- **黑色区域：**图像存放在内存中示意图，硬件读写内存通常会要字节对齐，假设硬件读写内存要求宽高 16 对齐，则图像就需要在宽度和高度上填充若干“黑边”以满足硬件的对齐要求，
  - 宽度上即 Width=1910 对齐到 Stride=1920；
  - 高度上 Height=1080 对齐到 HeightStride=1088；
- **黄色区域：**为实际业务逻辑上需要读取图像的矩形区域，称为 Crop 裁剪区域，业务上用两个点（Left,Top）（Right,Bottom）围成的矩形区域表示；

## 4.1 SrImageInfo 结构

SrImageInfo 类主要描述图像的必要信息，包括图像内存存储信息，图像有效区域信息等，结构头文件定义如下：。

```
class SrImageInfo
{
public:
    /* 描述图像信息结构 */
    SrBufferInfo mBufferInfo_;
    /* 描述图像 crop 信息结构 */
    SrRect mCrop_;
    /* AcquireFence, 标志源图像已完成, 可进行读写操作 */
    SrUniqueFd mAcquireFence_;
    /* 图像是否有效 */
    bool mValid;
    /* 构造函数 */
    SrImageInfo() : mAcquireFence_(-1), mValid(0){};
    SrImageInfo(const SrImageInfo& rhs);

    SrImageInfo& operator=(const SrImageInfo& rhs);
};
```

- SrBufferInfo: 描述图像信息结构，“[4.2 SrBufferInfo 结构](#)”会详细介绍；
- SrRect: 描述图像 crop 信息结构，“[4.3 SrRect 结构](#)”会详细介绍；
- SrUniqueFd: 内部实现的唯一文件描述符结构，主要用来存放 AcquireFence 与 ReleaseFence，关于 Fence 的介绍可以参考：[同步框架 | Android 开源项目](#)文档；
- mValid: 标志图像是否有效，例如 SrBufferInfo 检查 iWidth\_ 大于 iStride 明显的错误情况；

## 4.2 SrBufferInfo 结构

SrBufferInfo 主要描述图像的缓冲相关信息，结构头文件定义如下：

```
#define SR_BUFFER_INFO_RESERVED_MAX 16
class SrBufferInfo
{
public:
    int iFd_; /* 图像内容 fd 文件描述符, 通常为 dma-buffer fd */
    int iWidth_; /* 描述图像宽度, 单位为 pixel */
    int iHeight_; /* 描述图像高度, 单位为 pixel */
    int iStride_; /* 描述图像宽 stride, 单位为 pixel */
    int iHeightStride_; /* 描述图像高 stride, 单位为 pixel */
    int iByteStride_; /* 描述图像行长度 stride, 单位为 byte */
```

```
int iFormat_; /* 描述图像格式，定义为 drm_fourcc */
int iSize_; /* 描述图像完整尺寸，单位为 byte */
uint64_t uBufferId_; /* 描述图像唯一 ID，通常由分配器唯一分配 */
SrDatapace uColorSpace_; /* 描述图像色域信息 */
SrBufferMask uMask_; /* 描述图像特殊的标志，例如 AFBC 标志 */
int iReserved_[SR_BUFFER_INFO_RESERVED_MAX]; /* 预留结构 */

SrBufferInfo()
: iFd_(-1),
  iWidth_(0),
  iHeight_(0),
  iStride_(0),
  iHeightStride_(0),
  iByteStride_(0),
  iFormat_(0),
  iSize_(0),
  uBufferId_(0),
  uColorSpace_(SR_DATASPACE_UNKNOWN),
  uMask_(SR_BUFFER_NONE)
{
    memset(iReserved_, 0x0, sizeof(iReserved_));
};
SrBufferInfo(const SrBufferInfo& rhs);
SrBufferInfo& operator=(const SrBufferInfo& rhs);
bool isValid() const;
};
```

下面对部分结构进行补充说明：

- iByteStride\_ : ByteStride = iStride\_ \* bpp / 8; (bpp 解释为 bit per pixel)
- iFormat\_ : 格式采用 drm\_fourcc 描述，具体格式描述可参考：[drm\\_fourcc.h - include/uapi/drm/drm\\_fourcc.h - Linux source code \(v6.6.3\) - Bootlin](#)
- iSize\_ : iSize = ByteStride \* HeightStride;
- uBufferId\_ : 用来唯一描述每一块图形缓冲区，通常由内存分配器分配，如果内存分配器没有提供，请用户自行实现，只要能标识用户唯一缓冲区标识即可；
- uColorSpace\_ : 描述图像的色域信息，可参考 SrDatapace 结构描述；
- uMask\_ : 描述特殊图像的标志，例如 AFBC (ARM Frame Buffer Compression) 压缩格式，具体可参考：[Arm Framebuffer Compression \(AFBC\) — The Linux Kernel documentation](#)

**SrDatapac** 结构定义如下:

```
// 定义位于 libsvepsr/include/SrType.h
enum SrDatapace
{
    SR_DATASPACE_UNKNOWN          = 0,
    SR_DATASPACE_SHIFT            = 16,
    SR_DATASPACE_UNSPECIFIED      = 0 << SR_DATASPACE_SHIFT,
    SR_DATASPACE_BT601            = 1 << SR_DATASPACE_SHIFT,
    SR_DATASPACE_BT709            = 2 << SR_DATASPACE_SHIFT,
    SR_DATASPACE_BT2020           = 3 << SR_DATASPACE_SHIFT,
    SR_DATASPACE_RANGE_SHIFT      = 27,
    SR_DATASPACE_RANGE_UNSPECIFIED = \
        0 << SR_DATASPACE_RANGE_SHIFT,
    SR_DATASPACE_LIMITED          = \
        1 << SR_DATASPACE_RANGE_SHIFT,
    SR_DATASPACE_FULL             = \
        2 << SR_DATASPACE_RANGE_SHIFT,
    SR_DATAPACE_BT601_LIMIT_RANGE = \
        (SR_DATASPACE_BT601 | SR_DATASPACE_LIMITED),
    SR_DATAPACE_BT601_FULL_RANGE = \
        (SR_DATASPACE_BT601 | SR_DATASPACE_FULL),
    SR_DATAPACE_BT709_LIMIT_RANGE = \
        (SR_DATASPACE_BT709 | SR_DATASPACE_LIMITED),
    SR_DATAPACE_BT709_FULL_RANGE = \
        (SR_DATASPACE_BT709 | SR_DATASPACE_FULL)
};
```

**SrBufferMask** 结构定义如下:

```
// 定义位于 libsvepsr/include/SrType.h
enum SrBufferMask
{
    SR_BUFFER_NONE = 0,    /* 无特殊标志 */
    SR_AFBC_FMTATE = 1 << 1 /* AFBC 压缩格式 */
};
```

## 4.3 SrRect 结构

SrRect 主要描述一块矩形区域坐标，通常应用于标识真实图像区域。

```
class SrRect
{
public:
    int iLeft_;      /* 矩形区域 left 点坐标 */
    int iTop_;       /* 矩形区域 top 点坐标 */
    int iRight_;     /* 矩形区域 right 点坐标 */
    int iBottom_;    /* 矩形区域 bottom 点坐标 */
    int iReserved_[5]; /* 预留结构 */

    int Width() const;
    int Height() const;

    SrRect() : iLeft_(0), iTop_(0), iRight_(0), iBottom_(0)
    {
        memset(iReserved_, 0x0, sizeof(iReserved_));
    }
    SrRect(const SrRect& rhs);
    SrRect& operator=(const SrRect& rhs);
    bool operator!=(const SrRect& rhs);
    bool isValid() const;
};
```

实际业务逻辑上需要读取图像的矩形区域，称为 Crop 裁剪区域，业务上用两个点（Left,Top）（Right,Bottom）围成的矩形区域表示；



## 5 用户接口说明

### 5.1 SvpSr

#### 5.1.1 头文件说明

```
/**
 * @brief 获取 SvpSr 上下文
 */
SvpSr();
```

#### 5.1.2 接口应用举例

用例源码位于 `libsvepsr/examples/sr-demo/src/main.cpp`:

```
// 1. 获得 SR 实例
std::shared_ptr<SvpSr> sr_module = \
    std::shared_ptr<SvpSr>(new SvpSr());
if (sr_module == NULL)
{
    fprintf(stderr, "Sr init fail check\n");
    return -1;
}
```

### 5.2 Init

#### 5.2.1 头文件说明

```
/**
 * @brief 初始化 SvpSr
 *
 * @param version_str [IN] 外部传入的版本信息，用于版本适配，通常为宏定义
 *                          SR_VERSION，定义位于 SrType.h 头文件；
 * @param async       [IN] 异步初始化标志，设置为 true 表示使能异步初始
 *                          化，主要目的为了避免阻塞调用线程；
 * @return SrError::
 *         - None, success
 *         - Ohter, fail
 */
SrError Init(const char *version_str, bool async_init);
```

### 5.2.2 接口应用举例

■ 同步方式初始化:

```
// 2. 同步初始化
error = sr_module->Init(SR_VERSION, false);
if (error != SrError::None)
{
    fprintf(stderr, "Sr Init fail.\n");
    return -1;
}
```

■ 异步方式初始化:

```
// 2. 异步初始化
error = sr_module->Init(SR_VERSION, true);
if (error != SrError::None)
{
    fprintf(stderr, "Sr InitAsync fail.\n");
    return -1;
}
```

### 5.2.3 常见日志说明

日志可通过以下命令获取:

```
$ adb shell logcat | grep Svp
```

● 初始化成功:

```
$ adb shell logcat | grep Svp
# 版本校验成功
I SvpSr : CheckVersion,line=211, CheckVersion Success!
  LibVesion=SR-2.1.0, CallerVersion=SR-2.1.0
# 授权码校验成功, 离线授权文件已正常输出到 /data/system/svp_key.lic 文件
I SvpAuth: ReadAuthCode,line=475, CheckAuth: AuthCode: \
          tag=0x56524551 id=60 len=50 Code:          \
          71581714-9736-4EEE-C029-B65E54280A91
I SvpAuth: DoAuth,line=380, CheckAuth: rkauth_activate2 \
          auth code success, Write license to /data/system/svp_key.lic
# SR 各模型初始化成功
I SvpSrTn: InitResource,line=129, Sr-360p Init Success!
I SvpSrTn: InitResource,line=129, Sr-480p Init Success!
I SvpSrTn: InitResource,line=129, Sr-540p Init Success!
I SvpSrTn: InitResource,line=129, Sr-720p Init Success!
I SvpSrTn: InitResource,line=129, Sr-1080p Init Success!
I SvpSrTn: InitResource,line=147, Sr-2160p Init Success!
I SvpSrTn: InitResource,line=147, Sr-4320p Init Success!
I SvpSrTn: InitResource,line=147, Sr-4320p-v2 Init Success!
# 初始化成功
```

I SvpSr : Init,line=154, SvpSr init suceess, libsvepsr version: 2.1.0  
(e7ea135@2023-12-19T17:24:55)

- 初始化失败日志:

- 头文件版本错误:

- ◆ 版本号未传递:

E SvpSr : CheckVersion,line=165, Version\_str is NULL, CreatSrCtx fail!

- ◆ 版本号不匹配:

E SvpSr : CheckVersion,line=181, CheckVersion:  
LibVesion=SR-2.1.0 , CallerVersion=SR-1.9.1 unable to match version,  
SrInit fail!

- ◆ 版本号过低:

E SvpSr : CheckVersion,line=181, CheckVersion:  
LibVesion=SR-2.1.0 , CallerVersion=SR-1.9.1 caller minor must > 1, SrInit  
fail. Please updated version into SR-2.1.0

- 算法鉴权失败:

- ◆ VendorStorage 分区不存在:

E SvpAuth: ReadAuthCode,line=412, CheckAuth: /dev/vendor\_storage  
open fail

- ◆ 错误激活码:

E SvpAuth: CheckAuthCodeValid,line=495 CheckAuth:  
AuthCode=71581714-9736-4EEE-C029-xxxxxxxxxxxxxxxx is invalid, please  
check sr-auth code.

- ◆ 网络错误:

E SvpAuth: DoAuth,line=386: CheckAuth: rkauth\_activate auth code  
fail because networks err=RKAUTH\_NETWORK\_ERROR, try again.

- ◆ 无效离线授权问题件:

E SvpAuth: DoAuth,line=364: CheckAuth: rkauth\_verify\_license  
/data/system/svep\_key.lic fail, ret = RKAUTH\_LICENSE\_INVALID, to use  
AuthCode.

- 初始化失败:

SrBackend: Failed to initialize Sr bInitState\_=3

## 5.3 SetEnhancementRate

### 5.3.1 头文件说明

```
/**
 * @brief 设置 SR 增强强度，RK3588 支持，RK356x 不支持，
 *
 * @param rate [IN] 强度值，0-10，
 *                0：最低强度处理，性能最优，建议设置为 0
 *                10：最高强度处理，负载较高，效果最明显
 * @return SrError::
 *        - None, success
 *        - Other, fail
 */
SrError SetEnhancementRate(int rate);
```

### 5.3.2 接口应用举例

```
error = sr_module->SetEnhancementRate(0);
if (error != SrError::None)
{
    fprintf(stderr, "Sr SetEnhancementRate fail.\n");
    return -1;
}
```

### 5.3.3 效果举例

强度设置效果如下图 5-1 所示：



图 5-1 强度设置对比效果图

**注意：**从 V2.1.0 版本开始强度建议设置为 0，主要是基于性能考虑，强度设置大于 0，会额外占用 GPU 负载，请结合项目 GPU 负载余量进行配置。从 V2.1.0 版本开始算法模型进行了优化，强度 0 效果可等同于低于 V2.1.0 版本强度 5。

## 5.4 SetOsdMode

### 5.4.1 头文件说明

```
/**
 * @brief 设置 OSD 字幕模式
 *
 * @param mode [IN] 模式类型
 * @param osdStr [IN] OSD 显示字符串设置，可通过传递字符串的形式更改
 *                    OSD 内容
 * @return SrError::
 *         - None, success
 *         - Other, fail
 */
SrError SetOsdMode(SrOsdMode mode, const wchar_t *osdStr);
```

### 5.4.2 接口应用举例

- **SR\_OSD\_DISABLE**: 关闭 OSD 模式

```
error = sr_module->SetOsdMode(SR_OSD_DISABLE, NULL);
if (error != SrError::None)
{
    fprintf(stderr, "Sr SetOsdMode fail.\n");
    return -1;
}
```

- **SR\_OSD\_ENABLE\_VIDEO**: 使能 OSD 三行 OSD 模式:

```
#define SR_OSD_VIDEO_STR L"RKNPU-SVEP-SR"
error = sr_module->SetOsdMode(SR_OSD_ENABLE_VIDEO, \
                             SR_OSD_VIDEO_STR);

if (error != SrError::None)
{
    fprintf(stderr, "Sr SetOsdMode fail.\n");
    return -1;
}
```

- **SR\_OSD\_ENABLE\_VIDEO\_ONELINE**: 使能 OSD 单行模式:

```
#define SR_OSD_VIDEO_STR L"RKNPU-SVEP-SR"
error = sr_module->SetOsdMode(SR_OSD_ENABLE_VIDEO_ONELINE, \
                             SR_OSD_VIDEO_STR);

if (error != SrError::None)
{
    fprintf(stderr, "Sr SetOsdMode fail.\n");
    return -1;
}
```

### 5.4.3 效果展示

- 关闭 OSD 模式：
- 三行 OSD 模式，如下图 5-2：



图 5-2 三行 OSD 效果图

- 单行 OSD 模式，去除 Input/Output OSD，如下图 5-3：



图 5-3 单行 OSD 模式效果图

## 5.5 SetContrastMode

### 5.5.1 头文件说明

```
/**
 * @brief 设置对比模式，提供 SR 增强与源数据的对比模式展示
 *
 * @param enable [IN] 模式使能开关，未设置则关闭左右对比模式
 * @param offsetPercent [IN] 分割线左右占比，offsetPercent=(SR 图像/完整图像)，
 * 可实现扫描线效果，设置范围[0]与[10,90]，设置为 0，
 * 即为动态扫描模式，内部会自动偏移扫描线
 * @return SrError::
 *         - None, success
 *         - Other, fail
 */
SrError SetContrastMode(bool enable, int offsetPercent);
```

### 5.5.2 接口应用举例

- 固定对比模式:

```
error = sr_module->SetContrastMode(true, 50);  
if (error != SrError::None)  
{  
    fprintf(stderr, "Sr SetContrastMode fail.\n");  
    return -1;  
}
```

- 扫描对比模式:

```
error = sr_module->SetContrastMode(true, 50);  
if (error != SrError::None)  
{  
    fprintf(stderr, "Sr SetContrastMode fail.\n");  
    return -1;  
}
```

- 关闭对比模式:

```
error = sr_module->SetContrastMode(true, 50);  
if (error != SrError::None)  
{  
    fprintf(stderr, "Sr SetContrastMode fail.\n");  
    return -1;  
}
```

### 5.5.3 效果展示

- 固定对比模式与扫描对比模式，如下图 5-4：区别仅在于扫描线是否左右扫描：

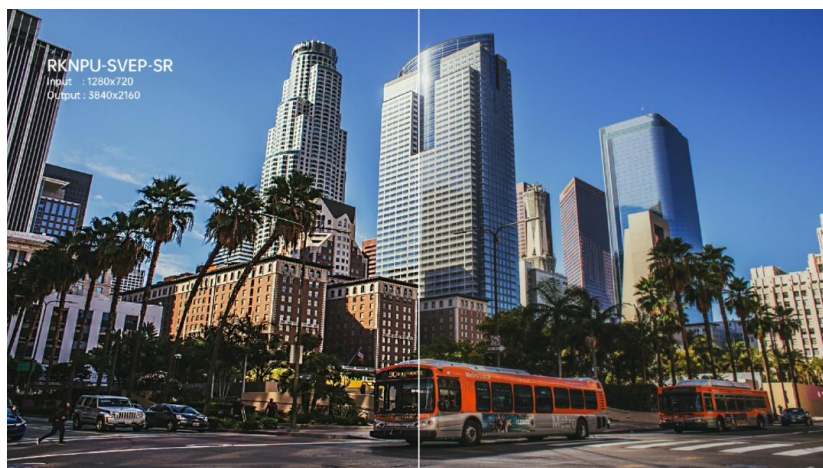


图 5-4 对比模式效果图



- 关闭对比模式，如下图 5-5:



图 5-5 关闭对比模式效果图

## 5.6 SetRotateMode

### 5.6.1 头文件说明

```
/**
 * @brief 设置旋转模式
 *
 * @param rotate [IN] 设置旋转方向
 * @return SrError::
 *         - None, success
 *         - Other, fail
 */
SrError SetRotateMode(SrRotateMode rotate);
```

### 5.6.2 接口应用举例

```
error = sr_module->SetRotateMode(SR_ROTATE_90);
if (error != SrError::None)
{
    fprintf(stderr, "Sr SetOsdMode fail.\n");
    return -1;
}
```



## 5.7 MatchSrMode

### 5.7.1 头文件说明

```
/**
 * @brief 匹配 SR 算法模型，SR 根据不同的输入分辨率，独立实现了不同的算法模
型
 *
 * @param int_src [IN] 输入图像信息
 * @param usage [IN] 输出 SR 处理模式特殊标志，目前支持两种模式配置：
 *                     SR_OUTPUT_8K_MODE : 4K 输入 8K 输出模式
 *                     SR_FACE_SR_MODE : 480p 输入优先配置人脸超分模式
 * @param out_mode [OUT] 输出 SR 匹配模式
 * @return SrError::
 *         - None, success
 *         - Other, fail
 */
SrError MatchSrMode(const SrImageInfo *int_src, SrModeUsage usage,
                    SrMode *out_mode);
```

### 5.7.2 接口应用举例

```
// 7. 申请 src 内存
Buffer *src_buffer = new Buffer(
    input_image.stride_w, input_image.stride_h, input_image.stride_w,
    input_image.fourcc_format, "SrSrc");
if (src_buffer->Init())
{
    fprintf(stderr, "Alloc src buffer fail, check error : %s\n",
            strerror(errno));
    return -1;
}
// 9. 设置源图像参数
SrImageInfo src;
src.mBufferInfo_.iFd_ = src_buffer->GetFd();
src.mBufferInfo_.iWidth_ = src_buffer->GetWidth();
src.mBufferInfo_.iHeight_ = src_buffer->GetHeight();
src.mBufferInfo_.iFormat_ = src_buffer->GetFourccFormat();
src.mBufferInfo_.iStride_ = src_buffer->GetStride();
src.mBufferInfo_.iHeightStride_ = src_buffer->GetHeightStride();
src.mBufferInfo_.uBufferId_ = src_buffer->GetBufferId();
src.mBufferInfo_.iSize_ = src_buffer->GetSize();
src.mCrop_.iLeft_ = input_image.x;
src.mCrop_.iTop_ = input_image.y;
src.mCrop_.iRight_ = input_image.x + input_image.crop_w;
src.mCrop_.iBottom_ = input_image.y + input_image.crop_h;
if (input_image.afbc)
```

```
{
    src.mBufferInfo_.uMask_ = SR_AFBC_FORMATE;
}
// 10. 获取 SR 处理模式与目标图像参数信息
SrModeUsage usage = SR_MODE_NONE;
SrMode sr_mde = SrMode::UN_SUPPORT;
error          = sr_module->MatchSrMode(&src, SR_MODE_NONE,
&sr_mde);
if (error != SrError::None)
{
    fprintf(stderr, "SwitchSrModeAndGetDstInfo fail, error=%d\n", error);
    return -1;
}
```

### 5.7.3 常见日志说明

日志可通过以下命令获取：

```
Android:
$ adb shell setprop vendor.svep.log info
$ adb shell logcat | grep Svep
Linux:
$ export RKSR_LOG_LEVEL=info
$ adb shell logcat | grep Svep
```

#### 5.7.3.1 模式匹配成功

- 720p 模式匹配成功日志：

```
MatchSrMode,line=499, SrcImage 1280x720 match mode=Sr-720p success!
This mode support max input :1280x720.
```

- FaceSr-480p 模式匹配成功日志：

```
I SvepSr : MatchSrMode,line=499, SrcImage 720x480 match
mode=Sr-480p-FaceSr success! This mode support max input :720x480.
```

#### 5.7.3.2 模式匹配异常

- 源数据 Crop 参数设置异常：top=1280， bottom=720，不满足 top < bottom 要求：

```
E SvepSr : VerifyImageInfo,line=407, inValid src image->mCrop_
(l,t,r,b)=(0,1280,1280,720)
E SvepSr : MatchSrMode,line=309, SrcImage info is inValid. Match SrMode
fail.
```

- 源数据 SrcBuffer 参数设置异常：width=1280, stride=720, 不满足 width < stride 要求

```
E SvepSr : VerifyImageInfo,line=413, inValid src image->mBufferInfo,
fd=47 w=1280 h=720 stride=720 height_stride=720 size=1382400
format=Nv12
```

```
E SvpSr : MatchSrMode,line=309, SrcImage info is inValid. Match SrMode fail.
```

- SR 模式匹配异常: width=12800, 不满足任意模型的输入条件

```
E SvpSr : MatchSrMode,line=330, Can't find a suitable mode.  
SrcImage info: fd=47 mCrop_=(l,t,r,b)=(0,0,12800,720)  
mBufferInfo, w=12800 h=720 stride=12800 height_stride=720  
size=13824000 format=Nv12 usage=0xd2f000usage=0x0
```

- 源数据参数不支持处理:

```
E SvpSr : VerifyRgaTransform,line=523, im2d imcheck_t fail , Error=xxx  
E SvpSr : MatchSrMode,line=317, SvpSr can't input SrcImage info:  
fd=63 mCrop_=(l,t,r,b)=(0,0,1280,720) mBufferInfo,  
w=12800 h=720 stride=12800 height_stride=720  
size=13824000 format=Nv12 usage=0xd2f000
```

## 5.7.4 效果展示

### 5.7.4.1 人脸超分

人脸超分效果如下图 5-6 所示, 当前 V2.2.0 版本还未集成该功能, 如需请咨询相关开发人员:



图 5-6 人脸超分图像对比

(从左往右依次为原图像, 一般超分模型, 人脸超分模型)

## 5.8 GetDetImageInfo

### 5.8.1 头文件说明

```
/**
 * @brief 获取 SR 输出图像参数要求
 *
 * @param out_dst [OUT] out_dst 主要返回 SrBufferInfo 与 SrRect 两个图
像信息:
 *   SrBufferInfo: 包含 width / height / stride / height_stride / format 信息
 *   SrRect : 包含真实图像矩形区域坐标, left / top / right / bottom
 * @return SrError::
 *   - None, success
 *   - Other, fail
 */
SrError GetDetImageInfo(SrImageInfo *out_dst);
```

### 5.8.2 接口应用举例

```
// 11. 获得目标图像参数
SrImageInfo target_image_info;
error = sr_module->GetDetImageInfo(&target_image_info);
if (error != SrError::None)
{
    fprintf(stderr, "SwitchSrModeAndGetDstInfo fail, error=%d\n", error);
    return -1;
}
// 12. 申请目标图像内存, SR 算法目前要求目标图像参数严格按照算法内部要求申请
Buffer *dst_buffer = new Buffer(
    target_image_info.mBufferInfo_.iWidth_,
    target_image_info.mBufferInfo_.iHeight_,
    target_image_info.mBufferInfo_.iStride_,
    target_image_info.mBufferInfo_.iHeightStride_,
    target_image_info.mBufferInfo_.iFormat_, 0, "SrDst");
if (dst_buffer->Init())
{
    fprintf(stderr, "Alloc dst buffer error : %s\n", strerror(errno));
    return -1;
}
```

## 5.9 Run

### 5.9.1 头文件说明

```
/**
 * @brief 同步处理模式, SR 算法完全执行完成后返回
```

```
*
* @param int_src [IN] 输入图像信息
* @param int_dst [IN] 输出图像信息, 要符合 GetDetImageInfo 接口返回要求
* @return SrError:
*         - None, success
*         - Other, fail
*/
SrError Run(const SrImageInfo *int_src, const SrImageInfo *int_dst);
```

### 5.9.2 接口应用举例

```
// 13. 更新 src 参数
SrImageInfo src;
src.mBufferInfo_.iFd_           = src_buffer->GetFd();
src.mBufferInfo_.iWidth_        = src_buffer->GetWidth();
src.mBufferInfo_.iHeight_       = src_buffer->GetHeight();
src.mBufferInfo_.iFormat_       = src_buffer->GetFourccFormat();
src.mBufferInfo_.iStride_       = src_buffer->GetStride();
src.mBufferInfo_.iHeightStride_ = src_buffer->GetHeightStride();
src.mBufferInfo_.uBufferId_     = src_buffer->GetBufferId();
src.mBufferInfo_.iSize_         = src_buffer->GetSize();
src.mCrop_.iLeft_               = input_image.x;
src.mCrop_.iTop_                = input_image.y;
src.mCrop_.iRight_              = input_image.x + input_image.crop_w;
src.mCrop_.iBottom_             = input_image.y + input_image.crop_h;
if (input_image.afbc)
{
    src.mBufferInfo_.uMask_ = SR_AFBC_FORMATE;
}
// 13. 更新 dst 参数
SrImageInfo dst;
dst.mBufferInfo_.iFd_           = dst_buffer->GetFd();
dst.mBufferInfo_.iWidth_        = dst_buffer->GetWidth();
dst.mBufferInfo_.iHeight_       = dst_buffer->GetHeight();
dst.mBufferInfo_.iFormat_       = dst_buffer->GetFourccFormat();
dst.mBufferInfo_.iStride_       = dst_buffer->GetStride();
dst.mBufferInfo_.iHeightStride_ = dst_buffer->GetHeightStride();
dst.mBufferInfo_.uBufferId_     = dst_buffer->GetBufferId();
dst.mBufferInfo_.iSize_         = dst_buffer->GetSize();
dst.mCrop_.iLeft_               = target_image_info.mCrop_.iLeft_;
dst.mCrop_.iTop_                = target_image_info.mCrop_.iTop_;
dst.mCrop_.iRight_              = target_image_info.mCrop_.iRight_;
dst.mCrop_.iBottom_             = target_image_info.mCrop_.iBottom_;
// 14. RunAsync
error = sr_module->Run(&src, &dst);
if (error != SrError::None)
{
    fprintf(stderr, "Sr RunAsync fail\n");
    return -1;
}
```

}

### 5.9.3 内部处理流程说明

- 内部处理流程图如下图 5-7:

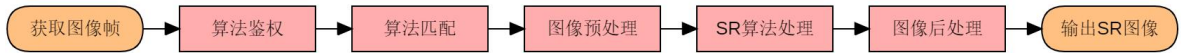


图 5-7 内部处理流程图

- 连续帧处理流程图如下图 5-8:

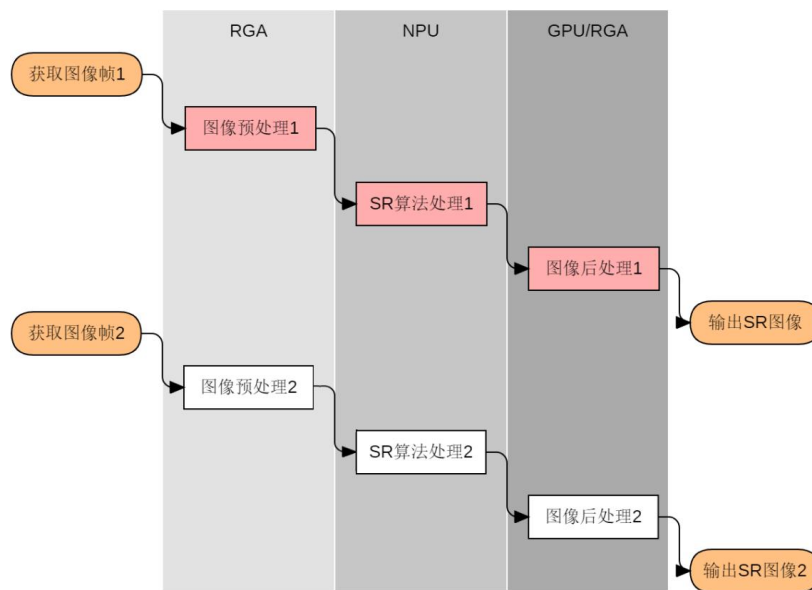


图 5-8 连续帧处理流程图

### 5.9.4 常见日志说明

打开调试日志命令如下:

- Android :

```
$ adb shell setprop vendor.svep.log info
$ adb shell logcat | grep Svep
```

- Linux:

```
$ export RKSR_LOG_LEVEL=info
$ adb shell logcat | grep Svep
```

### 5.9.4.1 执行正确日志

Run 接口执行正常日志如下图 5-9:

```
SvepSr : DumpCtx, line=632, Magic=0x3991306, version=2.0.10, mode=SR-720, EnhancementRate=0, OsMode=1, ContrastOffset=1, Rotate=0x0
1 SvepSr : DumpCtx, line=712, Src: Fd=17 Buffer=[w,h,s,hs,size,f] [1280,720,1382400,NV12] Buffer-Id=0x1cdf00000001 Crop=[l,t,r,b] [0,0,1280,720] color=0x0 mask=0x0 AcqFence=1
1 SvepSr : DumpCtx, line=73, Src: Fd=1 Buffer=[w,h,s,hs,size,f] [1280,720,1382400,NV12] Buffer-Id=0x0 Crop=[l,t,r,b] [0,0,1280,1080] color=0x0 mask=0x0 AcqFence=1
1 SvepSr : DumpCtx, line=752, Src: Fd=40 Buffer=[w,h,s,hs,size,f] [1920,1080,1920,1080,3110400,NV12] Buffer-Id=0x1cdf00000002 Crop=[l,t,r,b] [0,0,1920,1080] color=0x0 mask=0x0 AcqFence=1
0 SvepSr : Init, line=92, sr is ready.
1 SvepSr : Queue, line=194, FrameNo=1 Creat Fence Sr=1 dst-Buffer-Id=0x1cdf00000002 (FinishFence=43 size=1
1 SvepSr : Run, line=206, FrameNo=1 DoSR Success!
1 SvepSr : Queue, line=81, FrameNo=1 ToDQueue size = 1
1 SvepSr : Run, line=226, FrameNo=1 DoSRPost SRMode=4 run success, ret=0
1 SvepSr : Routine, line=106, FrameNo=1 DequeueRate SRMode=4 run success, ret=0
1 SvepSr : FinishSR, line=157, FrameNo=1 DoSR is Finish! ReleaseFence 43 Signal Success!
1 SvepSr : Routine, line=122, FrameNo=1 SrcContinue = 0x0 2113 Cx2:118 CF:113 waitFence:10 RMR:1 DR:6695 NRun:26240 waitQ:104 RQSD:6753 RQD:4 TQD:10 QRun:27264 OSD:894 CR:4419 waitDR:65 RT:98
```

图 5-9 Run 接口执行正常日志

### 5.9.4.2 执行异常日志

- Src Crop 参数设置异常: top=1280, bottom=720, 不满足 top < bottom 要求:

```
E SvepSr : PreCheckImageInfo,line=642, SrcImage: inValid
image->mCrop_ (l,t,r,b)=(0,1280,1280,720)
```

- Src Buffer 参数设置异常: width=1280, stride=720, 不满足 width < stride 要求

```
E SvepSr : PreCheckImageInfo,line=642, SrcImage: inValid
image->mBufferInfo, fd=63 w=1280 h=720 stride=720
height_stride=720 size=13824000 format=NV12
```

- Dst Crop 参数设置异常: top=3840, bottom=2160, 不满足 top < bottom 要求

```
E SvepSr : PreCheckImageInfo,line=642, DstImage: inValid
image->mCrop_ (l,t,r,b)=(0,3840,3840,2160)
```

- Dst Buffer 参数设置异常: width=3840, stride=2160, 不满足 width < stride 要求

```
E SvepSr : PreCheckImageInfo,line=642, DstImage: inValid
image->mBufferInfo, fd=64 w=3840 h=2160 stride=2160
height_stride=2160 size=12441600 format=NV12
```

## 5.10 RunAsync

### 5.10.1 头文件说明

```
/**
 * @brief 异步处理模式，利用 Pipeline 处理内部算法流程，图像连续处理可提高帧
率
 *
 * @param int_src [IN] 输入图像信息
 * @param int_dst [IN] 输出图像信息，要符合 GetDetImageInfo 接口返回要求
 * @param outFence [OUT] 是栅栏文件描述符，一种可跨进程传递的同步信号文
件描述符，Signal 信号发出后可标志 SR 任务完成
 * @return SrError:
 *         - None, success
 *         - Other, fail
```

```
*/  
SrError RunAsync(const SrImageInfo *int_src, const SrImageInfo *int_dst,  
                 int *outFence);
```

### 5.10.2 接口应用举例

```
// 13. 更新 src 参数  
SrImageInfo src;  
src.mBufferInfo_.iFd_           = src_buffer->GetFd();  
src.mBufferInfo_.iWidth_        = src_buffer->GetWidth();  
src.mBufferInfo_.iHeight_       = src_buffer->GetHeight();  
src.mBufferInfo_.iFormat_       = src_buffer->GetFourccFormat();  
src.mBufferInfo_.iStride_       = src_buffer->GetStride();  
src.mBufferInfo_.iHeightStride_ = src_buffer->GetHeightStride();  
src.mBufferInfo_.uBufferId_     = src_buffer->GetBufferId();  
src.mBufferInfo_.iSize_         = src_buffer->GetSize();  
src.mCrop_.iLeft_               = input_image.x;  
src.mCrop_.iTop_                = input_image.y;  
src.mCrop_.iRight_              = input_image.x + input_image.crop_w;  
src.mCrop_.iBottom_             = input_image.y + input_image.crop_h;  
if (input_image.afbc)  
{  
    src.mBufferInfo_.uMask_ = SR_AFBC_FORMATE;  
}  
  
// 14. 更新 dst 参数  
SrImageInfo dst;  
dst.mBufferInfo_.iFd_           = dst_buffer->GetFd();  
dst.mBufferInfo_.iWidth_        = dst_buffer->GetWidth();  
dst.mBufferInfo_.iHeight_       = dst_buffer->GetHeight();  
dst.mBufferInfo_.iFormat_       = dst_buffer->GetFourccFormat();  
dst.mBufferInfo_.iStride_       = dst_buffer->GetStride();  
dst.mBufferInfo_.iHeightStride_ = dst_buffer->GetHeightStride();  
dst.mBufferInfo_.uBufferId_     = dst_buffer->GetBufferId();  
dst.mBufferInfo_.iSize_         = dst_buffer->GetSize();  
dst.mCrop_.iLeft_               = target_image_info.mCrop_.iLeft_;  
dst.mCrop_.iTop_                = target_image_info.mCrop_.iTop_;  
dst.mCrop_.iRight_              = target_image_info.mCrop_.iRight_;  
dst.mCrop_.iBottom_             = target_image_info.mCrop_.iBottom_;  
int finish_fence = -1;  
AsyncWorker worker; // 创建线程等待 finish_fence 完成;  
// 15. RunAsync  
error = sr_module->RunAsync(&src, &dst, &finish_fence);  
if (error != SrError::None)  
{  
    fprintf(stderr, "Sr RunAsync fail\n");  
    return -1;  
}  
  
// 16. 将 finish_fence 传递给下一个线程处理, 等待 SR 模块处理完成;  
if (finish_fence > 0) worker.Queue(finish_fence);
```



### 5.10.3 内部流程处理说明

- 内部处理流程图下图 5-10:



图 5-10 内部流程处理说明

- 连续帧处理流程图如下图 5-11:

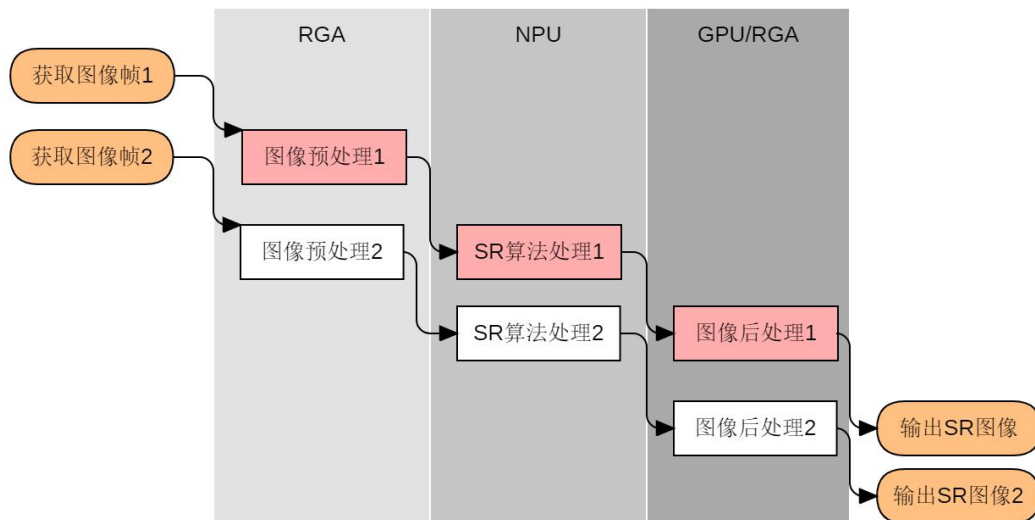


图 5-11 连续帧处理流程图

异步模式优势：通过流水线乒乓处理 SR 请求，充分利用 GPU/NPU/RGA 等硬件资源，提高帧率；

## 6 测试用例说明

SR 算法测试用例用例 `sr_demo` 工具源码位于以下路径：

```
libsvepsr/examples/sr-demo/
```

`sr_demo` 支持通过命令行构建所有用户调用接口场景，可方便快捷的构建验证目标场景。

### 6.1 编译说明

目前 `sr_demo` 支持 Android/Linux 版本编译，统一通过 `build.sh` 脚本编译，`build.sh` 路径如下：

```
libsvepsr/examples/sr-demo/build.sh
```

执行 `build.sh` 编译帮助信息如下：

```
$ ./build.sh
./build.sh -t <target> -a <arch> -s <sdk> -b <build_type>
Please select target platform: rk356x / rk3588 / rk3576

-t : target (rk356x/rk3588/3576)
-s : system (Linux/Android)
-a : arch (linux: aarch64/armhf; android: arm64-v8a/armeabi-v7a)
-n : sdk name(libsvepsr)
-b : build_type(Debug/Release/RelWithDebInfo)
-m : enable asan
```

#### 6.1.1 Android

以 RK3588 arm64-v8a 为例，尝试编译 `sr_demo` 工程。

**NDK 工具链版本：** android-ndk-r23，下载地址为：[NDK-Downloads](#)

1. **配置 NDK 工具链路径：**修改 `env_android.sh` 文件，路径为 `libsvepsr/env_android.sh`，脚本如下：

```
$ export ANDROID_NDK_PATH=path/to/android-ndk-r23
```

2. **编译命令：**

```
# RK3588 arm64-v8a
./build.sh -t rk3588 -s Android -a arm64-v8a -b Release
```

#### 6.1.2 Linux

以 RK3588 aarch64 为例，尝试编译 `sr_demo` 工程。

GCC 工具编译工具链采用 gcc-linaro-6.3.1 版本，下载地址为：[LinaroGccDownloads](#)

1. 配置编译工具链路径：修改 env\_linux.sh 文件，路径为 sr-sdk/env\_linux.sh，脚本如下：

```
export
GCC_COMPILER_PATH=path/to/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
export C_COMPILER=${GCC_COMPILER_PATH}-gcc
export CXX_COMPILER=${GCC_COMPILER_PATH}-g++
```

2. 编译命令：

```
# RK3588 aarch64
./build.sh -t rk3588 -s Linux -a aarch64 -b Release
```

## 6.2 使用说明

sr\_demo 执行即可打印帮助信息，可根据帮助信息进行快速查询命令格式：

```
$ adb shell sr_demo
```

帮助信息如下图 6-1 所示：

```
rk3588_box:/ # sr_demo
cmd_parse:
  src: crop[0,0,0,0] image[0,0,] afbc=0 path=
  dst: crop[0,0,0,0] image[0,0,] afbc=0
  mode: async=0 osd=0 spilt=0 spil-rate=0 en-rate=0 rotate=0x0
usage: sr_demo [-iomfHc]
eg: (Sr-720p) sr_demo -i 1280x720+0+0:1280x720@NV12#cs=601-1 -o 2560x1600+0+0:2560x1600@AB24#cs=709-f -m +async+osd+spilt=50+en=0+rotate=0-f /data/1280x720-nv12.yuv -c 100
eg: (FaceSr-480p) sr_demo -i 720x480+0+0:720x480@NV12#cs=601-1 -o 2880x1920+0+0:2880x1920@NV12#cs=601-1 -m +async+osd+face+spilt=50+en=0+rotate=0-f /data/720x480-nv12.yuv -c 100

Query options:
-i <crop_w>x<crop_h>+<x>+<y>:<stride_w>x<stride_h>@<format>[#afbc][#cs=601-f/601-l/709-f/709-l]
-o <crop_w>x<crop_h>+<x>+<y>:<stride_w>x<stride_h>@<format>[#afbc][#cs=601-f/601-l/709-f/709-l]
-m [+async][+osd][+spilt=50][+en=0][rotate=x] x: 0=0 1=90 2=180 3=270 4=x-filp 5=y-filp
-f <input_image_path>
-c <run_cnt> default cnt=1
-H help
```

图 6-1 sr\_demo 帮助信息

### 6.2.1 参数说明

sr\_demo 各参数说明如下：

- -i 参数：参数格式如下：

```
-i
<crop_w>x<crop_h>+<x>+<y>:<stride_w>x<stride_h>@<format>[#afbc][#cs=601-f/601-l/709-f/709-l]
```

具体参数详细信息如下表 6-1 参数支持：

表 6-1 sr\_demo input info 参数支持

参数名称	参数说明	参数是否必要	典型值
crop_w	crop width (值为 right - left )	必要	1280
crop_h	crop height (值为 bottom - top )	必要	720
x	x offset (值为 left )	必要	0
y	y offset (值为 top )	必要	0
stride_w	Stride width	必要	1280
stride_h	stride height	必要	720
format	drm_fourcc 格式	必要	NV12
#afbc	AFBC 压缩格式	可选	#afbc
#cs=601-l	设置输入色域空间信息	可选	#cs=601-l

- -o 参数: 与 -i 参数描述一致, 可直接参考 -i 参数描述
- -m 参数: 即 ENV Mode 信息, 参数格式如下:

**-m [+async][+osd][+spilt=50][+en=0][rotate=x]**

具体参数详细信息如下表 6-2:

表 6-2 sr\_demo 环境模式参数支持

参数名称	参数说明	参数是否必要	典型值
+async	开启异步处理模式, 若不添加则为同步模式	可选	+async
+osd	开启 OSD 打印, 默认值为三行 OSD 模式	可选	+osd
+face	使用人脸超分, 仅适用与 480p 输入	可选	+face
+spilt=50	开启对比模式, offset=50	可选	+spilt=80
+en=0	设置超分强度, enhancement=0	可选	+em=5
rotate=x	设置旋转模式: 0=0 度, 1=90 度, 2=180 度 3=270 度 4=x 轴镜像, 5=y 轴镜像	可选	+rotate=1

- -f 参数: 输入图像源数据路径, 参数格式如下:

**-f path/to/image.bin**

具体参数详细信息如下表 6-3：

表 6-3 sr\_demo 输入图像源数据路径

参数名称	参数说明	参数是否必要	典型值
-f	源数据输入路径，数据目前仅支持二进制图像数据	可选	/data/image.bin

- -c 参数：用例执行次数，设置为 100 则为相同参数重复执行 100 次；

-c run\_cnt

具体参数详细信息如下表 6-4：

表 6-4 sr\_demo 运行次数设置

参数名称	参数说明	参数是否必要	典型值
-f	用例运行次数	可选	1

注意：若缺省，则默认值为 1

## 6.2.2 执行说明

对执行 sr\_demo 用例作简要说明。

### 6.2.2.1 部署

将用例依赖文件部署到设备端，以使用例正常执行。

#### 6.2.2.1.1 Android

部署命令如下，以 RK3588 arm64-v8a 为例：

```
adb push build/install/Android/rk3588/arm64-v8a/lib/libsvepsr.so \  
    /vendor/lib64/      # libsvepsr.so  
adb push build/install/Android/rk3588/arm64-v8a/bin/sr_demo \  
    /vendor/bin/        # sr_demo  
adb push build/install/resource/SrOsd.ttf \  
    /vendor/etc/        # OSD 字体库文件
```

**注意：**工程中的 libOpenCL.so 文件仅用于解决编译问题，实际需要使用设备端的 libOpenCL.so，

如何判断设备端是否存在 LibOpenCL.so 可参考“[6.2.2.3.1 OpenCL 依赖问题](#)”。

### 6.2.2.1.2 Linux

部署命令如下，以 RK3588 aarch64 为例：

1. 部署 SR 资源文件：

```
adb shell mkdir -p /data/sr/
adb shell mkdir -p /vendor/etc/
adb push build/install/Linux/rk3588/aarch64/bin/sr_demo \
/data/sr/
adb push build/install/Linux/rk3588/aarch64/lib/libsvepsr.so \
/data/sr/
adb push build/install/resource/SrOsd.ttf /vendor/etc/
```

2. 创建 libOpenCL.so 软连接：

```
adb shell "ln -s /usr/lib64/libmali.so.1.9.0 /data/sr/libOpenCL.so"
```

注意：工程中的 libOpenCL.so 文件仅用于解决编译问题，实际需要使用设备端的 libOpenCL.so。

不同 Linux 平台或者 SDK 版本，libmali.so 版本可能都不一样，注意核对清楚版本，只

要支持 OpenCL 的 mali.so 版本即可，核对详情可参考“[6.2.2.3.1 OpenCL 依赖问题](#)”章节。

### 6.2.2.2 执行

● Android 命令：

```
$ adb shell 'sr_demo
-i 1280x720+0+0:1280x720@NV12#cs=601-l \
-o 2560x1600+0+0:2560x1600@AB24#cs=709-f \
-m +async+osd+spilt=50+en=0+rotate=0 \
-f /data/1280x720-nv12.yuv \
-c 20'
```

● Linux 命令：

```
$ adb shell 'export LD_LIBRARY_PATH=/data/sr ;\
./data/sr/sr_demo \
-i 1280x720+0+0:1280x720@NV12#cs=601-l \
-o 2560x1600+0+0:2560x1600@AB24#cs=709-f \
-m +async+osd+spilt=50+en=0+rotate=0 \
-f /data/1280x720-nv12.yuv \
-c 20'
```

执行命令解析如下：

- 输入图像参数：1280x720 的 NV12 图像，crop 区域为 (0,0,1280,720)，输入色域为 BT601-LimitRange
- 输出图像参数：2560x1600 的 AB24 图像，crop 区域为(0,0,2560,1600)，输出色域为

## BT709-FullRange

- 环境参数：使能 Async 模式，使能 OSD，使能对比模式，对比偏移设置为 50，SR 强度设置为 0，图像旋转关闭；
- 输入源图像地址：/data/1280x720-nv12.yuv
- 执行次数：20 次

## 6.2.2.2.1 日志说明

执行日志如下图 6-2：

```
130|ctg3:/ # sr_demo -i 1280x720+0+0:1280x720@NV12#cs=601-1 -o 2560x1600+0+0:2560x1600@AB24#cs=709-f -m +async+osd+spilt=50+en=0+rotate=0 -f /data/1280x720-nv12.yuv -c 20
cmd_parse:
  src: crop[0,0,1280,720] image[1280,720,NV12] afbc=0 cs=601-1 path=/data/1280x720-nv12.yuv
  dst: crop[0,0,2560,1600] image[2560,1600,AB24] afbc=0 cs=709-f
  mode: async=1 osd=1 spilt=1 spilt-rate=50 en-rate=0 rotate=0x0
rga_api version 1.10.0 [9]
Svep Async-Thread(1) Wait success: Time = 206.82ms, FPS = 4.84, cnt=1
Svep Async-Thread(1) Wait success: Time = 96.46ms, FPS = 10.37, cnt=2
Svep Async-Thread(1) Wait success: Time = 94.75ms, FPS = 10.55, cnt=3
Svep Async-Thread(1) Wait success: Time = 86.61ms, FPS = 11.55, cnt=4
Svep Async-Thread(1) Wait success: Time = 23.07ms, FPS = 43.34, cnt=5
Svep Async-Thread(1) Wait success: Time = 24.61ms, FPS = 40.63, cnt=6
Svep Async-Thread(1) Wait success: Time = 23.78ms, FPS = 42.06, cnt=7
Svep Async-Thread(1) Wait success: Time = 24.17ms, FPS = 41.38, cnt=8
Svep Async-Thread(1) Wait success: Time = 25.17ms, FPS = 39.73, cnt=9
Svep Async-Thread(1) Wait success: Time = 19.91ms, FPS = 50.23, cnt=10
Svep Async-Thread(1) Wait success: Time = 19.67ms, FPS = 50.84, cnt=11
Svep Async-Thread(1) Wait success: Time = 19.14ms, FPS = 52.26, cnt=12
Svep Async-Thread(1) Wait success: Time = 19.18ms, FPS = 52.12, cnt=13
Svep Async-Thread(1) Wait success: Time = 20.73ms, FPS = 48.23, cnt=14
Svep Async-Thread(1) Wait success: Time = 21.05ms, FPS = 47.51, cnt=15
Svep Async-Thread(1) Wait success: Time = 20.63ms, FPS = 48.40, cnt=16
Svep Async-Thread(1) Wait success: Time = 21.24ms, FPS = 47.09, cnt=17
Svep Async-Thread(1) Wait success: Time = 13.61ms, FPS = 72.43, cnt=18
Svep Async-Thread(1) Wait success: Time = 9.94ms, FPS = 100.55, cnt=19
Svep Async-Thread(1) Wait success: Time = 11.41ms, FPS = 87.64, cnt=20
Please enter a word to dump dst data
Image output to /data/dump/I0_SrSrc_1280x720_s1280x720_NV12.bin
Image output to /data/dump/I1_SrDst_2560x1600_s2560x1600_AB24.bin
Please enter a release sr resource and exit!
```

图 6-2 sr\_demo 执行日志

日志解析如下：

- cmd\_parse: 输入参数打印，用于核对用户输入参数，用户可核对是否符合预期；
- Svep Async-Thread: 异步等待 SR 接口执行完成日志；
  - Wait Success: 异步执行成功；
  - Time: 当前帧执行耗时；
  - FPS: 单帧耗时对应的 FPS 数据；
  - Cnt: 执行次数；
- Image output: 执行输入输出数据打印到 /data/dump/ 路径，若没有该目录，建议创建，文件名格式如下：

```
I<BufferId>_<BufferName>_<Width>x<Height>_s<Stride>x<HeightStride>_<drm_fourcc>.bin
```



### 6.2.2.2.2 核对输入输出

输入图像与输出图像打印路径如下：

```
/data/dump/I1_SrSrc_1280x720_s1280x720_NV12.bin    #输入图像  
/data/dump/I1_SrDst_2560x1600_s2560x1600_AB24.bin  #输出图像
```

可利用 YUView 软件查看输出图像数据，软件下载地址为：[YUView by IENT](#)

YUView 打开输出文件如下图 6-3 YUView 预览图：

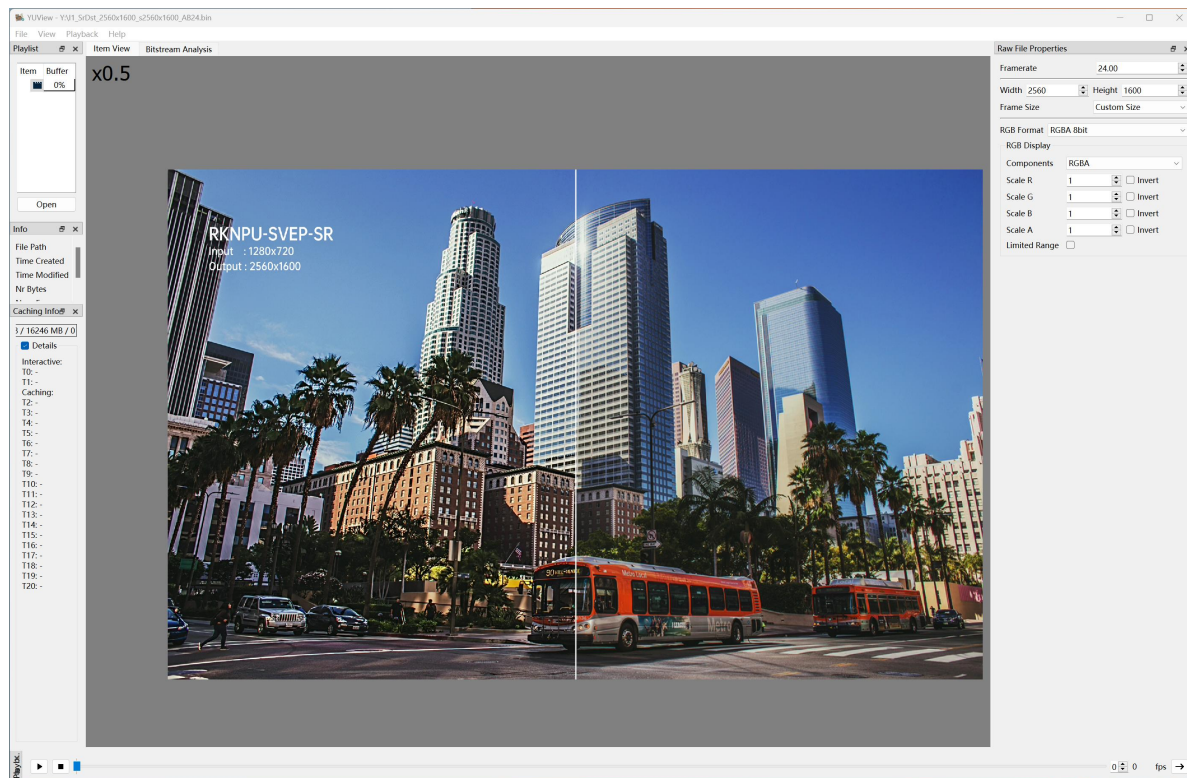


图 6-3 YUView 预览图

### 6.2.2.3 常见问题

#### 6.2.2.3.1 OpenCL 依赖问题

SR 算法依赖 OpenCL，检查 mali 库是否支持 OpenCL 命令如下：

- Android: 通常 libOpenCL.so 是 libGLES\_mali.so 的软连接，故仅需要查看 libGLES\_mali.so 是否支持即可：

```
$ adb shell busybox strings /vendor/lib64/egl/libGLES_mali.so | grep  
clGetPlatformIDs
```

- Linux: Linux 端的 libOpenCL.so 通常为 libmali.so 软链接，故仅需要查看 libmali.so 是否支持



即可：

```
$ adb shell busybox strings /usr/lib64/aarch64-linux-gnu/libmali.so.1.9.0 | grep clGetPlatformIDs
```

若支持 OpenCL，则命令会有打印 “clGetPlatformIDs”，日志如下图 6-4 OpenCL 支持日志：

```
lb@lb-pc:~/work/projects/dlss/libsvepsr$ adb shell 'busybox strings /vendor/lib64/egl/libGLES_mali.so' | grep clGetPlatformIDs
clGetPlatformIDs
```

图 6-4 OpenCL 支持日志

确认支持 OpenCL 以后，可能需要创建 libOpenCL.so 软连接，命令如下：

- Android:

```
$ adb shell ln -s /vendor/lib64/egl/libGLES_mali.so /vendor/lib64/libOpenCL.so
$ adb shell ln -s /vendor/lib/egl/libGLES_mali.so /vendor/lib/libOpenCL.so
```

- Linux:

```
$ adb shell "ln -s /usr/lib64/libmali.so.1.9.0 /data/sr/libOpenCL.so"
```

若不支持 OpenCL，请向 RK Redmine 提交工单申请支持 OpenCL 的 mali.so 文件，Redmine 地址为：<https://redmine.rockchip.com.cn/>。

### 6.2.2.3.2 Linux 初始化 Vulkan 报错

SR 强度（描述见 [5.3 SetEnhancementRate](#)）功能依赖 Vulkan 接口，由于部分 Linux 平台不支持 Vulkan，所以 SR 强度功能无法正常初始化，报错日志如下图 6-5：

```
1640 SVEP open(23): Vulkan is unavailable, install vulkan and re-start
1640 SVEP get(45): failed to open vulkan device!
1640 SVEP initVulkanApp(371): Vulkan is unavailable, SR enhancement will be disabled!
```

图 6-5 Linux SR 强度功能初始化失败日志

问题说明：SR 强度功能目前是可选功能，Vulkan 初始化只会影响 SR 强度功能，其他的 SR 功能可以正常支持，关于 SR 功能的建议如下：

- SR 强度设置为 0 以上（不包括 0），会增加 GPU 负载，建议客户将 SR 强度设置为 0。
- 如果对 SR 强度功能有强需求的，建议上 RK Redmine 平台咨询。

## 7 调试日志说明

此章节主要对 libsvpsr.so 输出的调试日志进行说明，便于用户后续调试。

使用 sr\_demo 如下命令，作为日志说明场景：

```
$ adb shell sr_demo -i 1280x720+0+0:1280x720@NV12  
-m +async+osd+spilt=50+en=0+rotate=0 -c 1
```

### 7.1 命令说明

SR 算法库调试日志通过设置 info 等级打开，命令区分 Android 与 Linux 版本。

#### 7.1.1 Android

打开调试日志命令如下：

```
$ adb shell setprop vendor.svep.log info # info 等级
```

抓打印日志建议通过以下命令：

```
$ adb shell logcat | grep Svep
```

#### 7.1.2 Linux

打开调试命令日志如下：

```
$ export RKSR_LOG_LEVEL info # info 等级
```

日志最终通过终端打印到屏幕。

## 7.2 默认输出日志

日志输出如下图 7-1 默认输出日志

```
I SvpSr : CheckVersion,line=211, CheckVersion Success! LibVesion=SR-2.1.0, CallerVersion=SR-2.1.0  
I SvpSrTn: InitResource,line=134, Sr-480p Init Success!  
I SvpSrTn: InitResource,line=134, Sr-540p Init Success!  
I SvpSrTn: InitResource,line=134, Sr-720p Init Success!  
I SvpSrTn: InitResource,line=134, Sr-1080p Init Success!  
I SvpSrTn: InitResource,line=153, Sr-2160p Init Success!  
I SvpSrTn: InitResource,line=153, Sr-4320p Init Success!  
I SvpSrTn: InitResource,line=153, Sr-4320p-v2 Init Success!  
I SvpSr : Init,line=154, SvpSr init suceess, libsvpsr version: 2.1.0 (8e1089b@2023-12-19T19:14:51)  
I SvpBuf : DumpData,line=271, output layer name=/data/dump/I1_SrSrc_1280x720_s1280x720_NV12.bin  
I SvpBuf : DumpData,line=271, output layer name=/data/dump/I2_SrDst_3840x2160_s3840x2160_NV12.bin
```

图 7-1 默认输出日志

- CheckVersion: 版本检查通过日志, 版本号打印为 SR-2.1.0
- InitResource: 模型初始化日志, 一共初始化了 480p 至 4320p-v2 7 个模型;
- Init,line=154: SvepSr 模块初始化成功日志, 打印版本号, 内部 CommitId 以及编译时间;
- DumpData: 打印输入输出图像, 写入 /data/dump/xxx 路径, 主要记得创建/data/dump 目录;

Info 日志就是日志等级设置到 info 打印输出的日志，日志如下图 7-2:

图 7-2 Info 日志

- **MatchSrMode:** 输入参数的匹配的对应 SR 算法模型打印，目前匹配到 720p 模型，并且该模型最大可支持输入的分辨率为 1280x720;
- **DumpCtx:** 用户最终调用 Run 或者 RunAsync 接口后，算法库会统一打印当前提交 SR 请求的所有参数信息;

- **Magic:**幻数，固定值用于匹配结构初始地址；
- **Version:** SR 接口内部使用的头文件版本号打印；
- **Mode:** 当前请求匹配的 SR 分辨率模型；
- **EnhancementRate:** 当前请求设置的 SR 增强强度值打印；
- **Constrast-offset:** 1-50，表示使能对比模式，offset 设置为 50；
- **Rotate:** 0 表示未开启输出旋转；
- **RgaPost:** 0 表示未启用后级缩放单元，1 表示启用后级缩放单元

- Src: 源数据设置参数信息:
  - ◆ Buffer: 源数据设置的 Width/Height/Stride/HeightStride/size/format 等信息;
  - ◆ Buffer-Id: 内存的唯一标识码, 每一块内存需要使用唯一的标识码标识, 主要用于内部的资源缓存, 减少单帧负载, 必须要设置一个唯一标识码, 否则无法正常执行;
  - ◆ Crop: 图像实际区域范围;
  - ◆ Color: 图像色域信息;
  - ◆ Mask: 内存特殊标识, 目前仅支持 AFBC 标识;
  - ◆ AcqFence: 源数据异步等待完成的 FenceFd, 用于等待源数据完成, 执行 SR 操作;
- IntSrc: 内部使用 Buffer 相关参数, SR 操作会将外部的输入图像本地缓存, 过程中会执行算法的参数对齐, 可以处理一些非标准分辨率的输入请求;
- IntDst: 内部使用 Buffer 相关参数, ;
- Dst: 输出 Buffer 相关参数, 参数参考 SrSrc;
- SvpTn/Tg/Tr: 为内部处理线程, 用户通常不需要参考这部分日志;

用户主要核对 DumpCtx 日志即可确认每次请求的 SR 参数是否符合预期即可。

## 8 SRAM 优化

RK3588 片内存在一块 1024KB 大小的 Share Memory，NPU 硬件利用这块 Share Memory 存储中间计算结果，减少访问 DDR，达到减少 DDR 带宽目的。

### 8.1 SRAM 优化对比

目前仅 RK3588 支持 SRAM 优化，提供 SRAM 优化前后 NPU 端的 DDR 带宽数据，如下表 8-1：

表 8-1 SRAM 优化前后数据

算法模型	优化前 MB/s	优化后 MB/s	优化率
480p	3086.84	821.23	73%
540p	9797.67	4912.85	49.9%
720p	7947.78	2062.41	74.1%
1080p	11809.18	5433.97	54.0%

### 8.2 SRAM 优化补丁

补丁路径：

```
libsvepsr/resources/patch/rk3588/sram/62a541a.diff
```

补丁方法：请在 kernel 目录打上补丁，例如 kernel 5.10 目录。

版本要求：

- NPU driver 版本要求：版本至少高于 0.8.0，查询命令如下：

```
$ adb shell dmesg | grep "Initialized rknpu"  
[drm] Initialized rknpu 0.9.0 20230629 for fdab0000.npu on minor 1
```

- libsvepsr.so：要求版本：SR-2.1.0，查询命令如下：

```
$ strings vendor/lib64/libsvepsr.so | grep libsvepsr  
libsvepsr version: 2.1.0 (4b25f45@2023-12-19T17:01:09)
```

优化确认：

确认 NPU driver 开启 SRAM 优化可利用以下命令确认：存在 RKNPU 分配 SRAM 地址的日志则

说明 SRAM 使能成功，命令如下：

```
$ adb shell dmesg | grep sram
```

输入日志如下：

```
[ 3.030561] RKNPU fdab0000.npu: RKNPU: sram region:  
[0x00000000ff001000, 0x00000000ff0f0000), sram size: 0xef000
```

## 9 常见问题

### 9.1 简单介绍 SR 具体做了哪些处理？

利用深度学习将低分辨率的图像放大为高分辨率图像，提高在低分辨率图像在高分辨率屏幕上显示效果。

### 9.2 SR 效果的评价手段和指标有哪些？

SR 效果评测可以利用图像和视频质量的评价指标比如 PSNR, SSIM, VMAF 等可以参考以下博客：  
视频流量在整个互联网流量的占比每年都在高速增长，为降低视频存储成本和数据传输通道的负载，视频压缩标准及算法在不断积极开发和改进。视频质量的评估在其中也起着至关重要的作用，尽管已经发展出了大量视频质量评估方法，但普遍接受度最高、最知名的评价方法还是经典的 PSNR、SSIM 以及 VMAF。

原文链接：[视频质量评价 VMAF，为何让人又喜又忧](#)

在评价视频质量时，单一指标数值的高低，并不能完全反映视频的画质好坏，与人眼主观感受相比会出现偏差。

所以通常评价视频画质时，要结合多项指标和人眼的主观感受进行综合评价。

### 9.3 非标准分辨率输出存在绿边问题

如果输入分辨率并不完全等于 SR 模型输入分辨率，目前内部做法是将源图像数据点对点拷贝到模型分辨率中，再执行 SR 处理，则输出图像会存在黑边问题，下面以 sr\_demo 举例说明：

sr\_demo 输入参数如下：

```
$ adb shell 'sr_demo -i 640x360+0+0:640x360@RG24 \  
-m +async+osd+spilt=50+en=0+rotate=0 \  
-f /data/bus_640x360_s640x360_RG24.bin \  
-c 1'
```

执行日志如下图 9-1 所示：



```
rk3588_box:/ # sr_demo -i 640x360@0:640x360@RG24 -m +async+osd+spilt=50+en=0+rotate=0 -f /data/sr/bus_960x540_s960x540_RG24.bin -c 1
cmd_parse: crop[0,0,640,360] image[640,360,RG24] afbc=0 path=/data/sr/bus_960x540_s960x540_RG24.bin async=1 osd=1 spilt=1 spilt-rate=50 en-rate=0 rotate=0x0
rga_api version 1.9.3 [1]
Svep Async-Thread(1) Wait success: Time = 92.03ms, FPS = 10.87, cnt=1
Please enter a word to dump dst data

Image output to /data/dump/I1_SrSrc_640x360_s640x360_RG24.bin
Image output to /data/dump/I2_SrDst_2304x1440_s2304x1440_NV12.bin
Please enter a release sr resource and exit!
```

图 9-1 sr\_demo 非标准输入执行日志

对应 info 日志如下图 9-2 所示：

```
I SvepSr : DumpCtx, line=692, Magic=0x83991986 Version=2.0.10 Mode=SR-480p EnhancementRate=0 QsdMode=1 Contrast-Offset=1-50 Rotate=0x0
I SvepSr : DumpCtx, line=712, SrSrc: Fd=47 Buffer=[w,h,s,hs,size,f]=[640,360,640,360,691200, ,RG24] Buffer-Id=0x63ea00000001 Crop=[l,t,r,b]=[0, ,0, ,640,360] color=0x0 mask=0x0 AcqFence=-1
I SvepSr : DumpCtx, line=732, SrInt: Fd=1 Buffer=[w,h,s,hs,size,f]=[768,480,768,480, , ,NV12] Buffer-Id=0x0 Crop=[l,t,r,b]=[0, ,0, ,640,1080] color=0x0 mask=0x0 AcqFence=-1
I SvepSr : DumpCtx, line=752, SrDst: Fd=50 Buffer=[w,h,s,hs,size,f]=[2304,1440,2304,1440,4976640, ,NV12] Buffer-Id=0x63ea00000002 Crop=[l,t,r,b]=[0, ,0, ,1920,1080] color=0x0 mask=0x0 AcqFence=-1
```

图 9-2 非标准输入参数打印

关注信息如下：

- 模型匹配到 480p 模型，符合向上模型匹配规则；
- 输出 crop 信息为 (0,0,1920,1080)，符合 480p 3 倍放大情况；

YUView 打开文件如下图 9-3 所示：

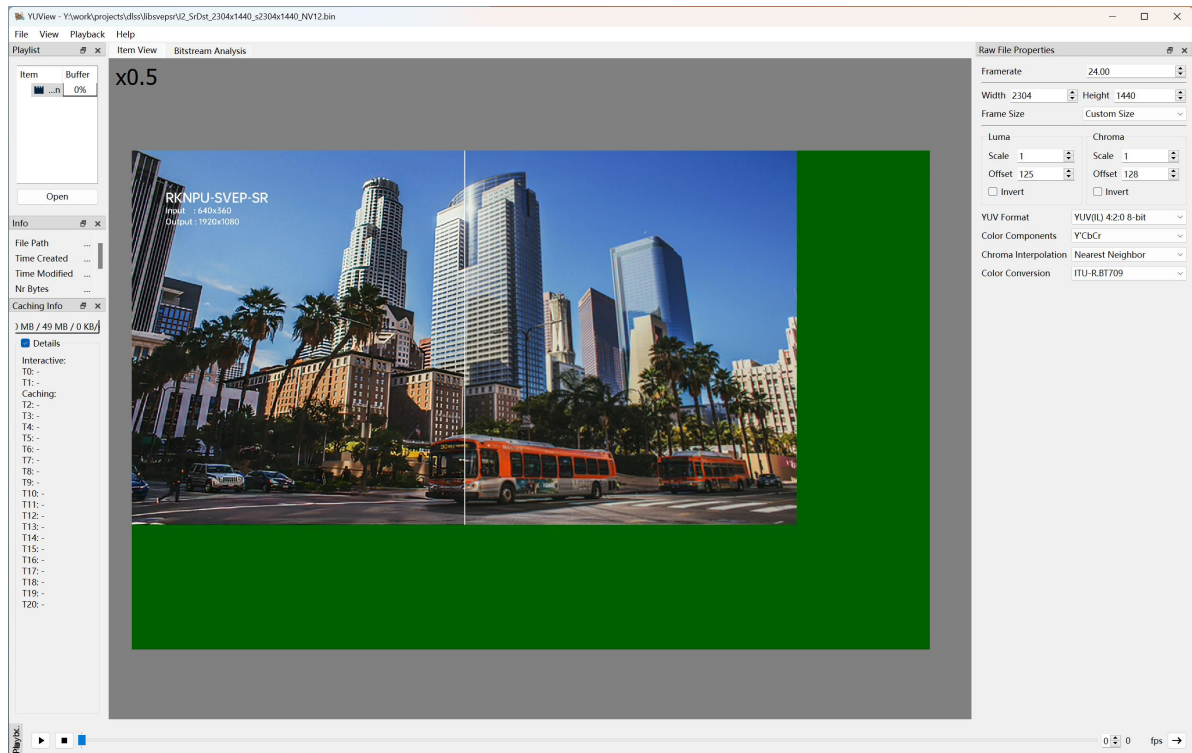


图 9-3 360p 输出图像预览图

关注点：实际图像内存大小为 2304x1440，有效图像存在于 (0,0,1920,1080) 区域内，符合预期。

外部模块要使用这份输出图像的话，需要设置 Crop 信息，否则将会存在绿边。



## 9.4 最新版本获取

SVEP 目前永久版本发布地址如下：

<https://console.box.lenovo.com/1/Q08d2o> 提取码：rksvep

目录结构如下图 9-4 所示：

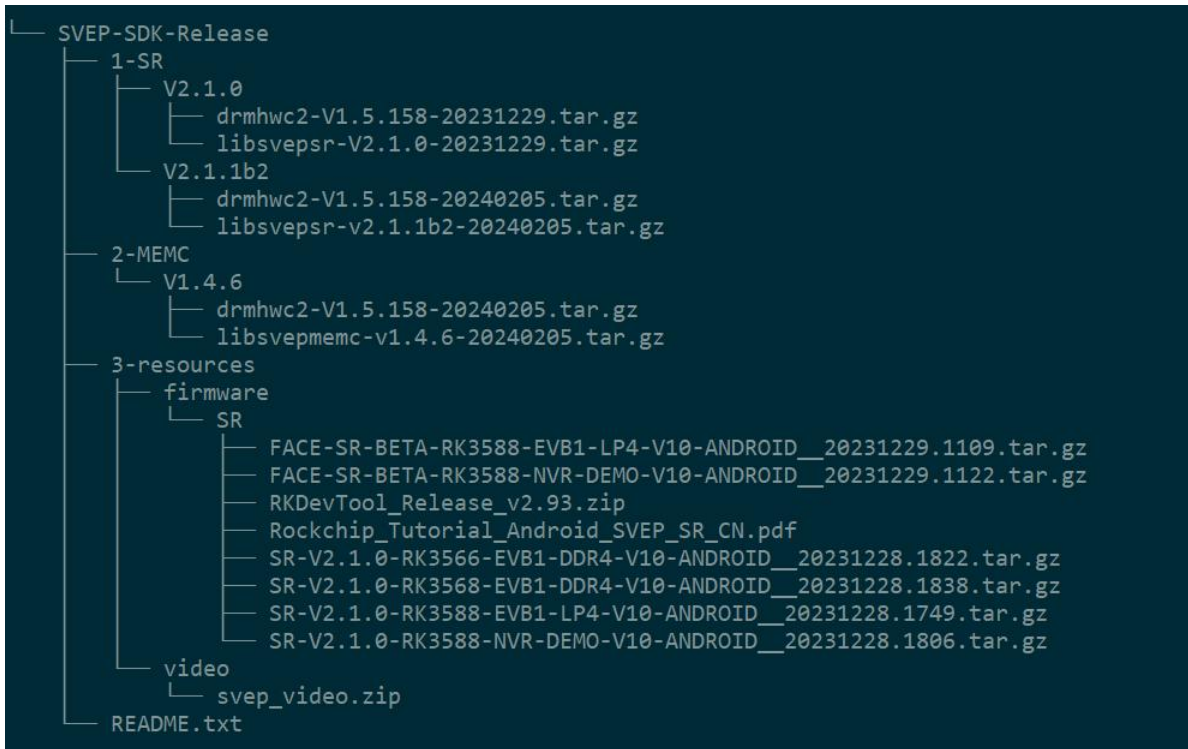


图 9-4 SVEP 发布目录结构说明

- 1-SR: SVEP-SR 版本发布目录说明
  - V2.1.0: 发布版本号
    - ◆ drmhwc2.tar.gz: 版本匹配的 HWC 源码版本, 请替换 SDK 如下目录:  
Path/to/sdk/hardware/rockchip/hwcomposer/drmhwc2
    - ◆ libsvepsr.tar.gz: 请替换 SDK 如下目录:  
Path/to/sdk/hardware/rockchip/libsvep/libsvepsr
- 2-MEMC: SVEP-MEMC 版本发布目录
  - V1.4.6: 发布版本号
- 3-Resource: 其他的资源文件, 例如视频, 演示固件等