Mobile Computing Group Project Final Report

Jiaxin Ding, Yuxin Ding, Ziling Dong, Pai Peng, Qinxin Ren, Junyi Wei

December 16, 2021

Abstract

For the prosperity of live streaming, more and more various display is needed. Therefore, there are diversified platforms for live streaming. However, all these platforms only provide single-view display and makes the change of camera a very fussy procedure. To give the anchors a more satisfied experience and meet their demand, this project aims at providing a multi-view display on Android devices. Multiple UVCCameras can be accessed in the live streaming room, while anchors can choose a certain camera's view as the main view. Filters, audio wave and camera manager are provided in the room at the same time to convenient the anchor. The external accessed UVCCameras also meet the need of high revolution. The platform was built as an Android application using the MVC framework and is compatible with Android-based mobile devices.

I. Introduction

i. Motivation

With the rapid development of Internet technology, more accessible information communication has made webcasting a new way of transmitting and sharing information in the new media environment. As an interactive social platform with a highly entertaining nature, more people are willing to share their lives live through the use of portable mobile devices.

However, given the limitations of the screen size of mobile devices, most mobile live streaming platforms are currently fixed to a single screen, unable to show multiple images at the same time, and can generally only use the front and rear cameras for filming. So if a host wants to show a richer picture and make the live room more informative, they need more professional equipment and technical support to ensure the live room screen switching is smooth, which also means higher live streaming costs. For example, usually for a live cooking presenter, they need to show the whole picture of chopping vegetables and the details, and even the pan being cooked at the same time. With the current mobile platform, the anchor would have to place the camera farther away to get a panoramic view of the kitchen, or move the camera around frequently to capture more of the scene. But obviously this clumsy way of broadcasting misses out on a lot of details, not only to convey the anchor's movements more accurately, but also to help viewers watch and learn from the anchor's cooking methods. This obviously runs counter to the original intention of making live streaming a more convenient way to share your life socially. Thus, the objective of this project is to create an app that would allow users to broadcast live from their mobile devices anywhere anytime, and to switch between multiple cameras conveniently.

ii. Related Work

Considering the limitations of hardware devices and the complexity of the environment in which most users use mobile devices for live streaming, many mobile-based live streaming platforms on the market are

relatively homogeneous in terms of functionality. Figures 1 and 2 show the interface of an anchor live streaming room on bilibili and weibo platforms, and it can see that both platforms only show the screen captured by the currently selected camera. If the user intends to switch the screen, they can only do so by moving the device or flipping the camera on the phone, which obviously reduces the anchor's user experience by frequently switching the screen. The platform is also unable to support the display of additional content (e.g., external cameras) if the presenter wishes to do so. In addition, there is no preview function for flipping the camera, which means that when the anchor flips the camera, it is not certain that the target image can be accurately presented to the audience at the first time, and it takes time to adjust it, which also reduces the user experience of the audience.

In addition, for the above mentioned live streaming platforms on mobile the missing feature of previewing the footage captured by multiple cameras is implemented on the PC side. Taobao Live is one such example. Hosts need to log in to Taobao Live on their computers and go to the main function screen to set up their live stream. Such a product fulfils the function of multi-camera streaming, and the anchor can add multiple cameras at the same time for display and preview. However, such a product requires a certain amount of user learning costs and is not as easy to use as the mobile live streaming platform. In addition, for professional anchors to use the features of the PC-based live streaming platform and to pursue the compactness of the live streaming process, they need the collaboration of other technical staff to complete the process, which in effect increases the cost of live streaming and raises the threshold of live streaming, and obviously contradicts the original intention of creating a more convenient way of socializing.


Figure 1: Taobao and Bilibi Page

iii. Contribution

Yuxin Ding (1822288)

- Responsible for audio broadcast part, collect, handle and package the audio input data.
- Design the front-end visualization of audio, realize an animation of voice wave.
- Work on combination of front end and back end of project, provide interim version for code combination.

Jiaxin Ding (1824389)

- Responsible for the interface and interaction design of all interfaces in the app by figma software.
- Responsible for the front-end development of the app, including: launch page, login page, registration page, home page, personal homepage and basic version of the live room.
- Development of the front-end for the horizontal and vertical versions of the live room.
- Linking all interfaces together to form a complete app.
- Writing the introduction and evaluation sections of the report.

Zilin Dong (1825191)

- Take part in the overall rough design of the whole application.
- Collect some information about SurfaceView Render, multi-camera, image switch, and live streaming.
- Finish the single original camera filter application, design the front-end interaction of filter function and try to integrate filter camera to the whole project.

- Modify the workflow chart and do some part job in pre and report

Pai Peng (1822600)

- Control the app developing flow of entire project.
- Implementation of original camera.
- Implementation of Muti-UVCCamera.
- Implementation of view switching function.

Qinxin Ren (1824098)

- Responsible for realizing the function of bullet screen display and sending.
- Responsible for realizing the display of local multiple MP4 videos at the same time.
- Responsible for the push streaming function.
- Responsible for composing the report and presentation slides.

Junyi Wei (1823146)

- Configure the overall Android development environment, including: establishing the SDK version, NDK version, Gradle version and version to match the project's core technical support: the UVCCameralibrary [1].
- Integrating the front-end of projects with core functional pages.
- Based on an understanding of the overall project structure, a flow chart of the overall project structure is drawn in the report section.

iv. Why is our work useful?

This project solves this problem by introducing multi-camera preview and switching technology on mobile devices. Whether the anchor is a professional or an amateur, the anchor can manage the live broadcast without the help of other technicians, which greatly improves the efficiency of the anchor and the user experience of the viewers.

II. APP Design

i. Application Design

a Development flow

The overall work flow can be divided into these following seven stages.

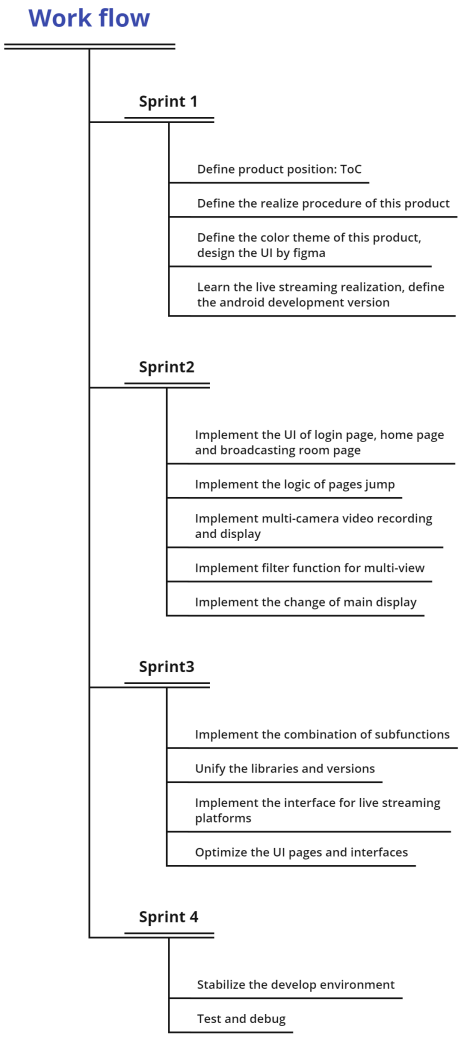


Figure 2: Development workflow

In stage 1, the initial thing is to identify the product position. As a helpful tool for anchors to manage the live coverage, it is defined as a ToC product. Knowledge for the display of multi-camera and the realization procedure of live streaming should also be known at first. The initial UI design of this application is made by figma. Since the android development version varies a lot, version 8.1 is chosen.
In stage 2, the UI of login page, main page and broadcasting room page is implemented. Besides, the logic of pages jump like the entrance of broadcasting room is carefully designed. sub-functions are implemented at the same time: multi-camera video display, filter function for multi-view, and display are also realized respectively. Each of these functions are tested separately.
In stage 3, the combination of sub-functions is realized. Since the dependent libraries are different and work in different development versions, works need to be done to unify the versions and combine these functions together. The interface for live streaming platforms is realized. UI pages and interfaces are optimized again.
In stage 4, the development environment is stabilized and the whole application is tested and debugged.

b User Story

This project has three key design objectives, shown as the user stories below. These target users are: normal user, user with high resolution needs, and user with multi-camera needs. For the normal user, this application provides basic functions. The user can use the native camera to start live streaming, browse the homepage, use the filter the manage the display and visualize the audio speech.
For the user with high resolution needs, this application can provide: basic functions of the broadcasting room, management of filters, visualization of audio speech, access of high resolution cameras and push streams management.
The third kind of user is someone with the need of multi-camera. The application will provide these functions: basic operation functions of broadcasting room, management of filters, visualization of audio speech, access of multi-cameras, preview of multi-views, switch of multi views and the management of push streams.
Moreover, the access towards the broadcasting room is simplified to convenient the user. The operation procedure is: 1. log in the account; 2. register an account; 3. start the live streaming. In the broadcasting room, there are functions for the anchor to choose: 1. access different UVCCameras through Matedock; 2. manage the display of views by changing the main view; 3. choose to switch on/off the filter; 4. visualization of audio speech. The users can choose all these functions in terms of their requirements.

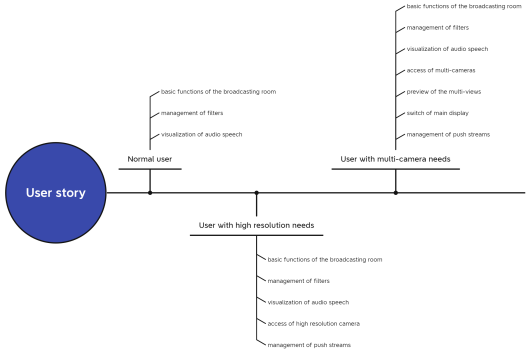


Figure 3: User stories

ii. System Architecture

The framework flowchart is shown as below:

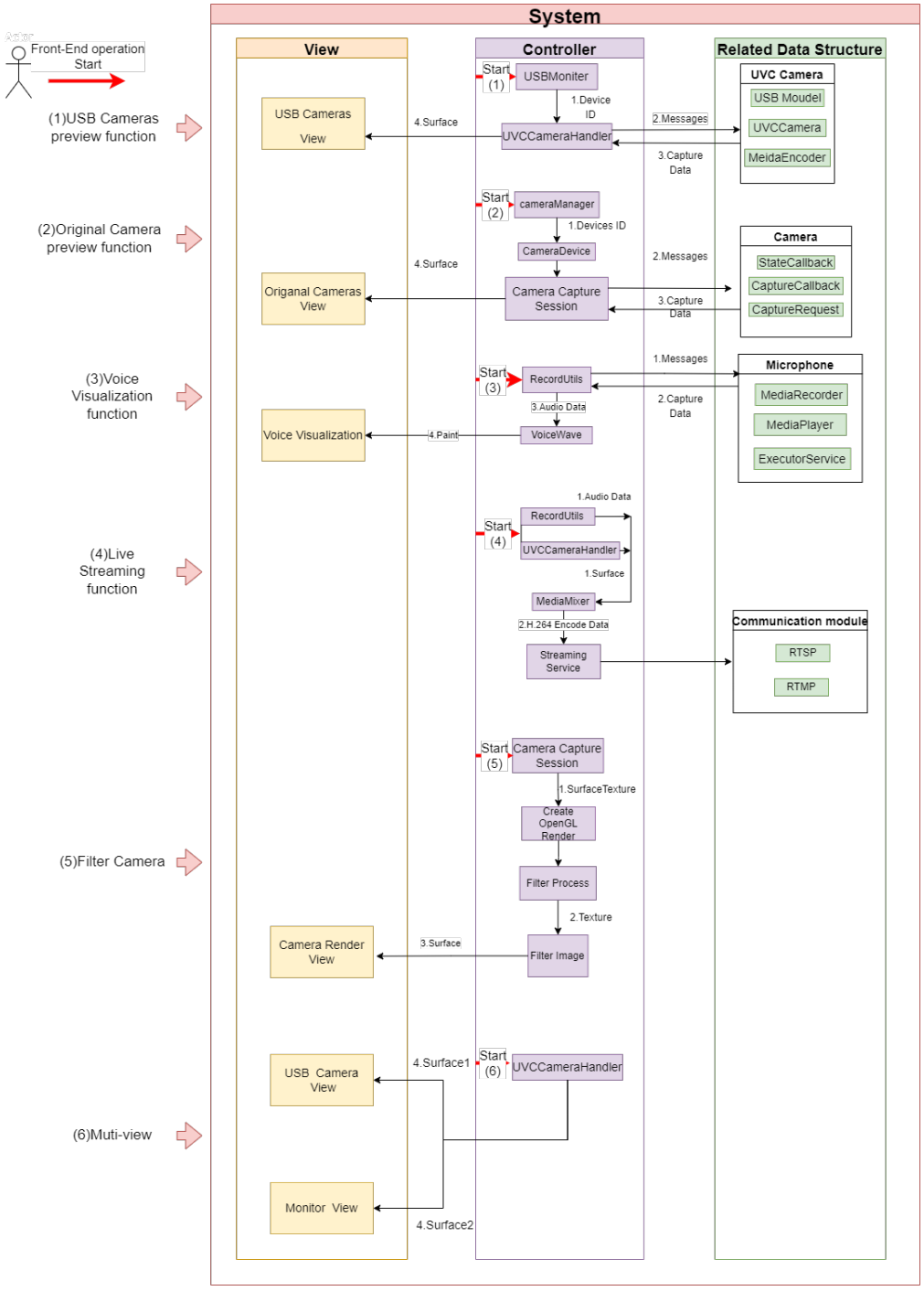


Figure 4: Framework Flowchart

The producer-consumer pattern is prevalent among Android developers and is defined by the fact that the producer's job is to generate data, put it into a buffer, and start over, at the same time, consumers consume data one piece at once. A similar system architecture is used in this project, where the main process is to perform matching between producers and consumers. The figure shows the detailed work process of the Controller to match the producer and consumer.

- Video and Image Processing : In the USB multi-camera preview function, the main participants of the matching work are USBMoniter.java and UVCCameraHandler.java. CameraCapture.java plays an important role in the original camera preview function.
- Audio Processing : Audio visualization uses a similar principle, i.e. RecordUtils.java does the matching and VoiceWave.java draws the visual waveform graph.
- Streaming Service : The principle of live streaming function is slightly different. StreamingService.java matches the production that not belongs to the internal system, but the IP address of the corresponding server.

This application aims at solve the existing technical difficulties. The basic components can be categorized as front end and back end. For front end, Surface and Surface/TextureView component is used. For back end, there are: utilization of original camera, utilization of UVC camera through OTG connection, and utilization of microphone. With these components, these functions can be implemented: vocal information visualization, access and management to multi-camera, multi-view switching and management, management of filter, live streaming control.

a Front End Component

Surface/TextureView The View class is the basic building block of the user interface. View represents a rectangular area of the screen that is responsible for drawing and event handling, where we can draw the graphics we want the user to see. Although both View and Surface/TextureView can be used as video and screen displays, Surface/TextureView has its own drawing Surface, meaning it doesn't share the same drawing Surface with its host window. With a separate drawing Surface, the Surface/TextureView UI can draw rows in a separate thread, and the SurfaceView can achieve a complex and efficient UI by not using mainline resources. Therefore, for the simultaneous display of multiple screens, we choose Surface/TextureView to manage the screen.
In this project, we override these classes based on Surface/TextureView:

- Surface/TextureView method, where GLSurfaceView is used for filter management.
- AspectRatioTextureView, which is used to adjust the resolution and size of images.
- UVCCameraInterface, which is used to upload camera images.

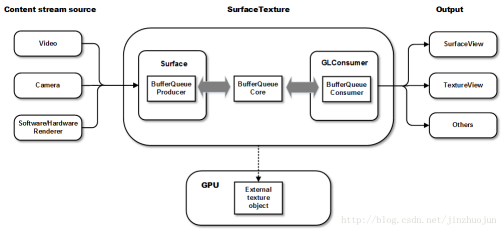


Figure 5: Surface/Texture View hierarchy

Surface In the project, the surface class refers to the most significant interface to support the preview function of both the

original camera and UVC camera. It defines the form of image or video data transferring form from hardware layer output to visual layer (UI) input. For instance, in the preview session for the UVC camera, a capture session from the camera thread output its capture frame to a specific surface. In the meantime, the surface was bonded to a specific TextureView, enabling users to see the real-time video preview in a particular area. The surface class build up a connection between the video stream and the front view, which set the base for further stream management camera preview and live streaming.

b Back End Components

Utilization of original camera This session would introduce the utilization and control of original camera from back-end based on camera2 API. CameraManager is used to manage the system's native cameras. It controls the camera image parameter adjustment and Capture process in the Control Module. The CameraDevice is directly connected to the system camera, which is quite an abstract camera. CaptureRequest-Session is used for submitting a created CaptureRequest. CaptureRequest is used to set and output a single image taken from a camera. The workflow is shown as below:

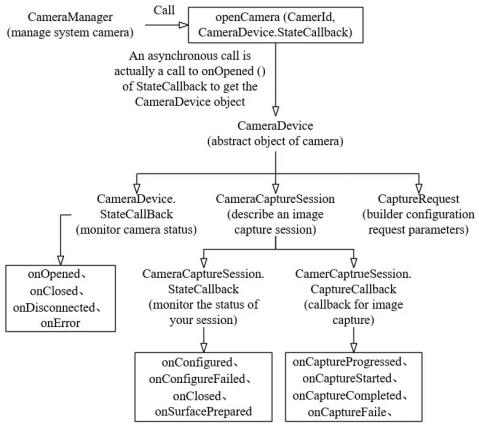


Figure 6: Native Camera Management

Utilization of UVC camera through OTG connection In this project, UVC (USB Video Class) is used to manage multiple cameras because it involves the access of multiple USB cameras to expand the picture display. UVC is a protocol standard specifically defined for USB video capture devices. In order to make the interface of USB device freely connected, OTG protocol is also introduced in this project. In this way, external devices, such as external cameras, can be connected through USB ports.
This session would introduce the utilization and control of UVC camera from back-end based on UVCCamera API. For different Camera access, Camera encapsulation can be roughly divided into data acquisition and rendering. You can use UVCCameraHelper to manage multiple cameras in a unified manner. In this class, you can connect the USB device, turn on the camera to collect data and render the data to the Surface.

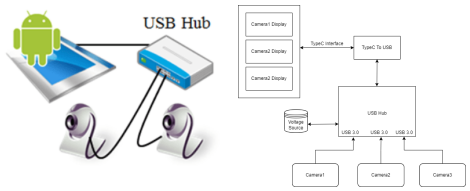


Figure 7: USB devices

Utilization of microphone This session aims to combine the audio collection step with visualization effects. During the live broadcast, the audio is firstly collected from the microphone, then the sampled PCM audio is encoded into AAC (Advanced Audio Coding) format, then the encoded AAC is packaged into RTMP package, and push it into the server. In order to better visualize this process, a sound wave effect is added, which will present animation of real time change according to the decibels collected. In the audio processing process, we take samples in PCM format by using Au-

dioRecord, then get and set suitable parameters for the encoder. After encoding the data into AAC format, the AAC data is packaged into RTMP packet and push the packets into our server.
After realizing these components, functions can be achieved based on the components.

c Function Implementation

Vocal information visualization First is to combine the audio collection step with visualization effects. A little project of sound recording is tested. In this small project, both the audio input collection, storage and broadcast, combined with an aesthetic animation of voice wave is realized. Then, after finishing the layout design and framework deployment, the audio treatment process into the frame is implemented. Some adjustment work and usability test to improve the performance are also implemented. Finally is to further encode and package the audio data, then push the audio packages together with the media packages into the server.
To visualize the voice wave, firstly set the datalist based on different decibels, also initialize the stroke width, height rate and some other related parameters to decide the size and range. At each coordinate the line lengths are divided into long and short to simulate the fluctuation effects. Then the painting functions to keep drawing lines are used all the time.

Figure 8: Vocal Information Visulization

Access and management to multi-camera To connect multiple cameras, perform the following steps:

- Connect a USB camera using an expansion dock. After obtaining camera permissions, call processConnect method to establish USB connection.
- Call the listening connector of USBMonitor. MOnDeviceConnectListener to get a successful usb connection callback to start the camera with the open method in UVCCameraHandler.
- After the camera is started, it continues back to the handleOpen method of CameraThread, where callOnOpen is called to inform the external camera that it is started.

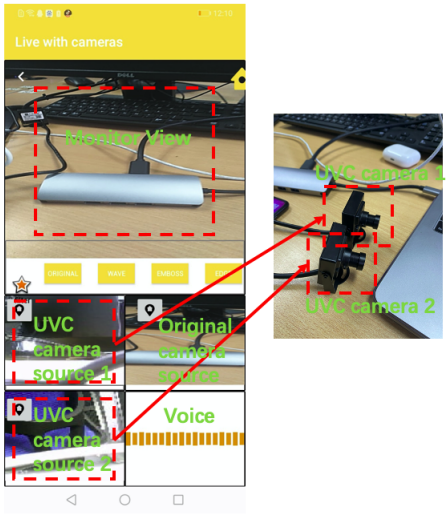


Figure 9: UVCCamera Access

A surface of type Object is passed in startPreview, where each frame of data captured by the camera can be drawn on this texture. The Surface will be drawn to the screen via The SurfaceView or TextureView encapsulated by Android to display multi-camera images. The workflow is shown as below:

Figure 10: UVCCamera Management

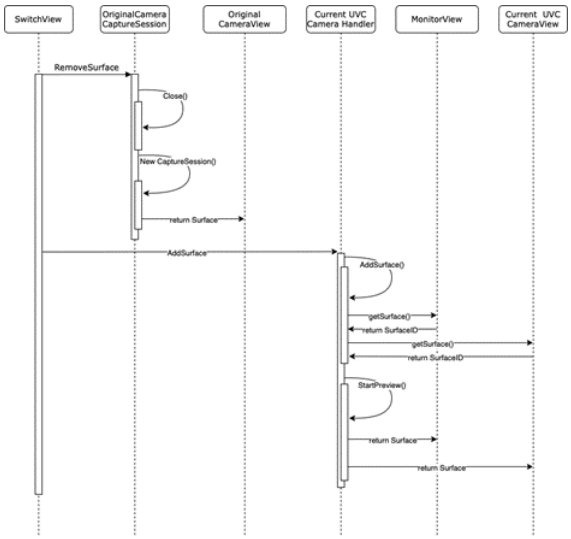


Figure 11: Switch structure

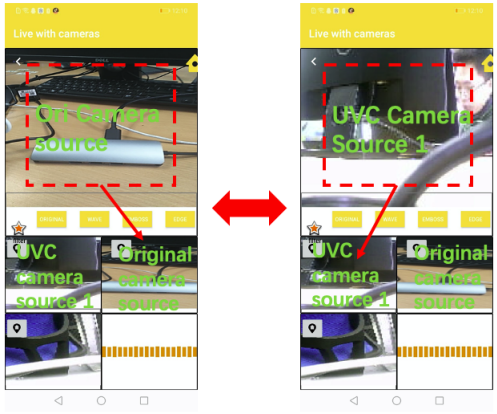


Figure 12: Switch the View

Multi-view switching and management The view switch function rearranges the bonding relationships between MonitorView, CameraView, and Surface sent by different controllers. Due to the differences in control processes between the original camera and UVC camera, the integration of the two controllers became a major challenge for this feature. The figure shows the view switching process from the original camera to the UVC camera. As shown in the figure, switching between different views generally involves two phases: the removal of the old surface and the addition of the new surface. The first stage is to remove the current surface bonding relationship with the monitor view. Because the CameraCaptureSession class is a camera thread and render handler, the removal process should be synchronous given the potential for bugs in multithreaded systems. Therefore, because of the complexity of the Camera2 API, we used a re-creation of the new controller rather than surface sharing. On the other hand, additional processes can be viewed as implementations in a single-provider, multi-consumer situation. UVCCameraHandler with multi-surface is designed to solve the shunting problem. It successfully distributes the stream from UVCCamera to UVCCameraView and MonitorView. Through this process, we realized multi-view management and switching functions.

Management of filter After the user makes the request for real-time preview of filters, the request is transmitted from the Java layer to the Filter rendering control layer of Native layer through the JNI layer. The pre-rendering work includes initialization, and the successful initialization state is returned after member variables are passed in. Then, the application will obtain the original image data in the interface callback provided by the Camera on Android system at the

Java level. The data conversion operation will be run on GPU via the previous analysis and filter processing operation, and the high efficiency of the execution will be maintained. As a result, the data is passed to the native filter rendering system through the JNI layer. The rendering system transmits the data and the processing script written by GLSL to the GPU via OpenGL for the two-stage data format conversion and data filter processing operation. After processing, OpenGL draws the picture on the screen, and the drawing operation is also within the native layer. Finally, for the saving of the filter image, the Java layer notifies the Native layer to perform the saving operation through the interface provided by JNI. The result will be called back on the Java layer if the saving is successful. At this point, the whole filter sequence has ended. The real-time performance of the application is guaranteed by OpenGL controlling GPU.
The development hierarchy of filter implementation is shown in the figure.

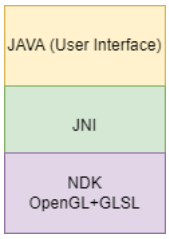


Figure 13: Filter structure

Live streaming control The streaming service mainly contain 2 stage – the encapsulation of video and the forwarding to sever. The aims of encapsulation is to remove the redundant information within the video in order to encoder into video stream with low bandwidth need. The common type for image encoding is H.264 and AAC for sound encoding. The transmission protocol used in this framework is RTSP (Real Time Stream Protocol). It is a TCP-based network transport protocol that lies between the application layer and the transport layer and is responsible for encapsulating streaming data and enabling real-time transmission of media streams. In our application, we use SWVideoEncoder.class and AACEncoder.class to consume the capture data from microphone and camera and transfer them into RTSP packages.The forwarding process to server is achieved by RtspSessionManager.class. Meanwhile, we use VLC player to examine the streaming service from the computer ends.

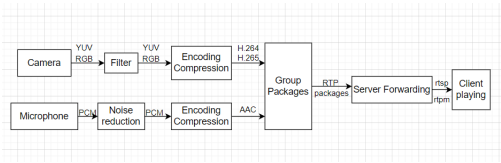
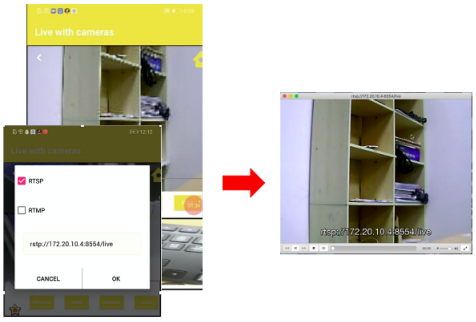


Figure 14: Push Stream



(1) Streaming Service **(2) Receiver on VLC player**

Figure 15: Push the Stream

iii. UI Design

The application is technical and designed to be simple for user to use. We used numerous design concepts when creating the system UI in order to implement a series of user-friendly UI [1]. The homepage, buttons on the broadcasting room were all designed with the goal of achieving a user-

friendly principle and improving the user experience. In terms of adaptability, The font, color theme and typography are uniform and clean, making it easier for users to navigate the application's interfaces and discover the information and functions they are looking for.
The interfaces are also compatible with the screens of a variety of Android smartphones. Simultaneously, the software offers users with feedback after they have taken action, allowing for more effective involvement. By using this application, Android smartphones can be turned into a portable production studio. Simple and easy UI design lower the shelter of live streaming. In this application, UI design can be categorized by pages: login page, register page, home page and broadcasting page.

Login page there are two buttons "Login" and "Create New Account". User can choose to log in with their accounts via "Login" button, or register a new account via "Create an Account" button.

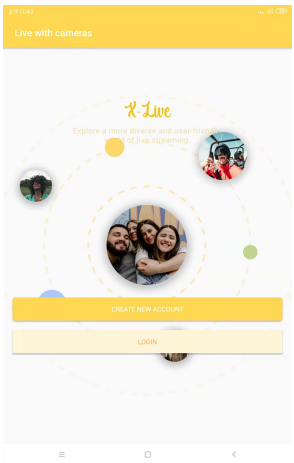


Figure 16: Login Page

Register page the user can fill their information to create an account. These information are : full name, email, password and certification of password. After registering, the user can log in via the account.

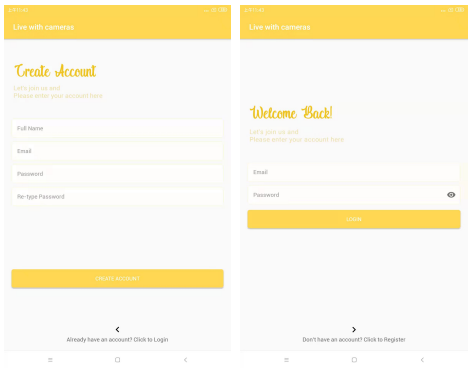


Figure 17: Register Page

Home page user can browse others' live streaming videos in the home page. In the follow page, the user can watch the videos of his/her followed anchor. Moreover, in this application, every user has a personal home page. In the personal home page, every user is able to upload his/her own videos.

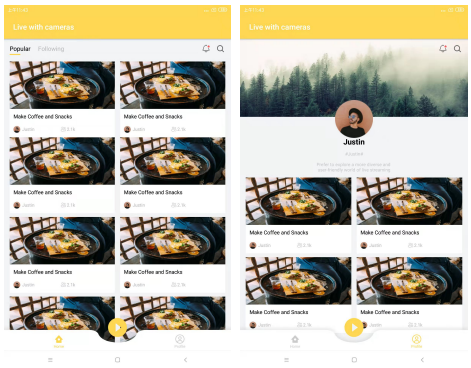


Figure 18: Home Page

Live streaming page user can have multiple operations in this live streaming room page, which is also the key part of this project.
In this page, the user can access multiple external USB cameras, switch the main view, choose the filter. The audio of the user is also

recorded and shown in the waveform. Moreover, if the user want to push the stream to another platform, the application also provides this function.
Here is the procedure of accessing external USB camera. The user can access the external USB camera through the matedock and then plug the matedock to the Android device. Then the user can choose to access the camera to the display. This application allows 3 cameras to access. The procedure is shown below:

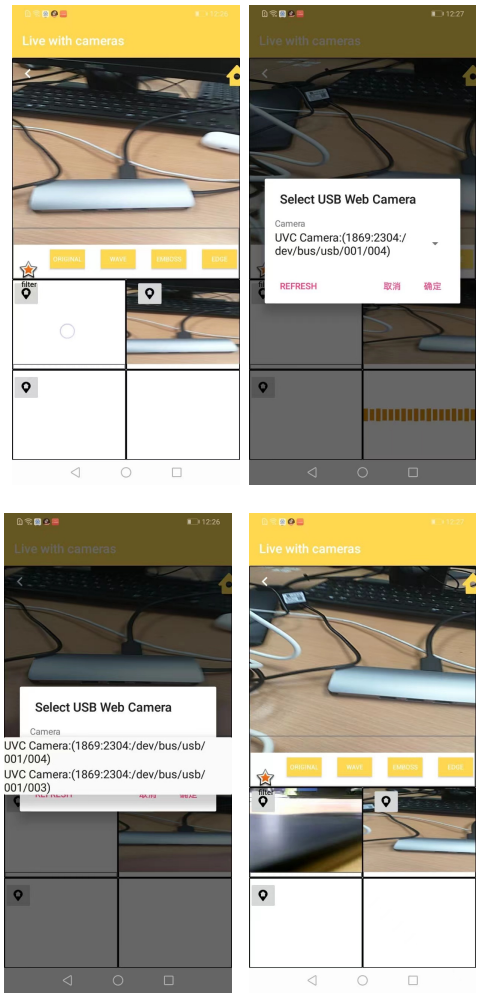


Figure 19: Procedure of Accessing Cameras

The user can also click the switch button to switch the main display according to the demand. The main display is originated from each camera.
Besides, this application also provides multiple choices of filters. The user can choose the filters according to their specific requirement. The procedure is shown as below:

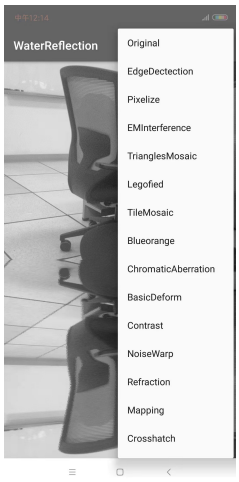


Figure 20: Login Page

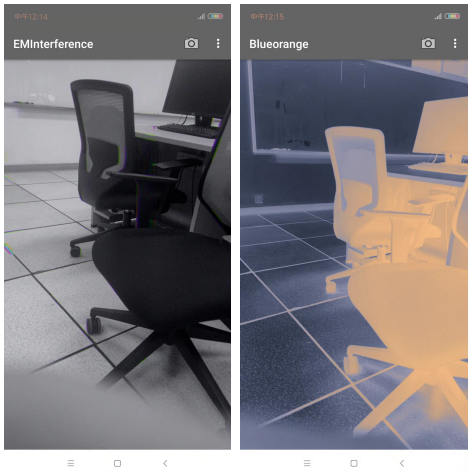


Figure 21: Filter Effects

The user can also click the push stream button, choose the specific protocol and input the address. After these operations, the video can be received on another device as shown.

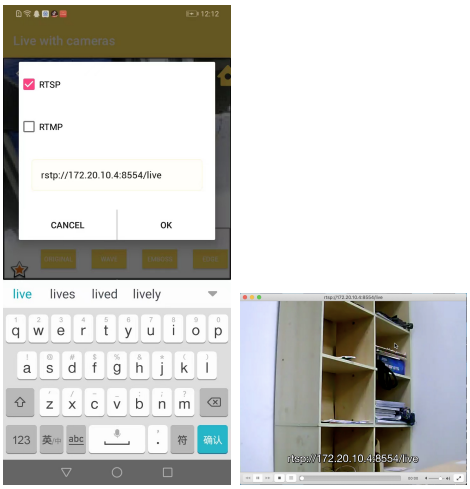


Figure 22: Push Stream Effect

iv. User Friendly

Since it is a technical application focused on hardware, the user friendly can be divided into hardware friendly and target group friendly.

a Hardware Friendly

The application is considered as hardware friendly for these reasons.
First of all, the ecosystem which combines both cell phone and multi-camera makes the display of views much more convenient. The previous display of camera view only contains the native camera. It is quite inconvenient and confined to the local positions.
Secondly, this application expand the Android functions. The previous applications are mainly focus on the software layer. Therefore, the usage is limited. This application combines hardware and software together to give the new meaning of Android applications. Multi-view display of the live streaming will bu push to the platform simultaneously via Wifi or cellular network.
Thirdly, this application has good compatibility. If a user want to use a smartphone as a video source, all kinds of Android smartphones with system version between 2.2 and 8.1 are compatible. For compatible cameras, this application has been tested and approved for all UVCCameras. Therefore, the user can choose high-resolution camera to access if they have the demand for revolution, he can also choose multi-cameras to access if he wants to have the display of multi-views. The accessed UVCCameras are not limited, and the display is depending on the number of cameras.

b Target User Friendly

As mentioned before, this application satisfy the demand of multi-view display. If someone needs to capture the display from different locations, it will helps a lot. Previous applications like bilibili and huya only provide a single view display. While using this application, different cameras can be placed in different positions and form the multi-view display. This application also makes live streaming devices more portable. All the needed devices are smartphone/pad, and UVCCameras, getting rid of the limit of computer to manage the display. Moreover, quick-start with this application for android devices to stream the multi-view production with live switching to platforms like bilibili, huya and other RTMP destinations.

III. Evaluation

a Problems and Solutions

Issue 1: Remote Debug issue Since our application involves the implementation of USB device , the port of USB connection was usually occupied by UVC camera during the testing of Muti-camera function. As we could not get the exact error happens in our application running , we could not locate where the program go wrong.

Solution 1: ADB WIFI plugin Since the USB port is occupied , the idea of using WIFI or Bluetooth connection to transfer the error message pop out. ADB (Android Debug Bridge) is a powerful tool in android application development. Thankfully , there are some wifi Debug plugins for Android Studio. After failing to activate WIFI ADB plugin by Denzi [1] due to version difference, we choose to adopt ADB WIFI plugin by Yury [2] to debug program when usb port is used.

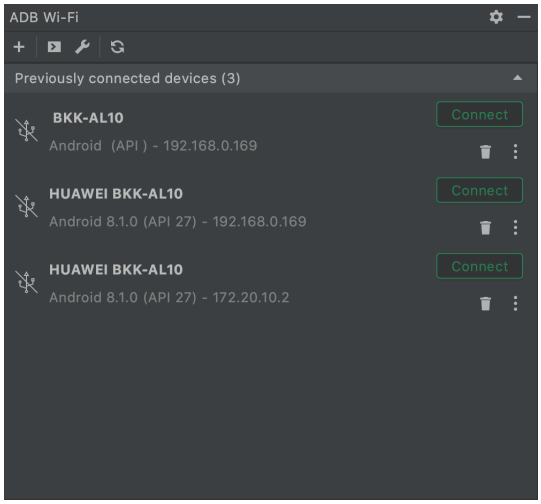


Figure 23: Solution1

Issue 2: Bandwidth Issue for using multiple UVC camera The issue is discovered during the testing of multiple UVC camera connection. Since the XIAOMI tablet and common android device with android 8.1 use USB2.0 and OTG as transport protocol , the upper limit of bandwidth is 480 Mbps , which could hardly achieve multiple UVC camera device with 480p resolution with 5 FPS.

Solution 2: Change camera device and C library for libuvccamera In order to support using multiple UVC camera simultaneously , we first tried to change our UVC camera with wider resolution option to 240p. The camera capture data should required lower bandwidth. And we also go further into the C lib for libuvvcamaera and cast the data stream over the device streaming limit 512 maxim bytes per packet. The cast result in delay in our application, however it could support 2 extra camera source.

```
/* Find the endpoint with the number specified in the VS header */
for (ep_idx = 0; ep_idx < altsetting->bNumEndpoints; ep_idx++) {
    endpoint = altsetting->endpoint + ep_idx;
    if (endpoint->bEndpointAddress == format_desc->parent->bEndpointAddress) {
        endpoint_bytes_per_packet = endpoint->wMaxPacketSize;
        // wMaxPacketSize: [unused:2 (multiplier-1):3 size:11]
        // bit10..0:     maximum packet size
        // bit12..11:    the number of additional transaction opportunities per microframe for high-speed
        //               00 = None (1 transaction per microframe)
        //               01 = 1 additional (2 per microframe)
        //               10 = 2 additional (3 per microframe)
        //               11 = Reserved

        // ++++++++++++++++++++++++++add-in++++++++++++++++++++++++++++++
        if ((endpoint_bytes_per_packet & 0x07ff) > 512) {

            endpoint_bytes_per_packet = 512;}



        endpoint_bytes_per_packet
            = (endpoint_bytes_per_packet & 0x07ff)
              * (((endpoint_bytes_per_packet >> 11) & 3) + 1);
        break;
    }
```

Figure 24: Solution2

Issue3: Keep the coherence of the Android build environment

Solution3: Because of UVCcamera lib only support for Android 8.1 and below version, we must put all architecture environments under SDK 28. Sequentially, NDK version is limited at r14rb and it below version for SDK version. The Gradle and JDK versions are identical due to the influence of the Android SDK version. Another sequential environment is the Android support API version, we did front-end work in Android X, but the other part of the work is under Android. There are some differences between them, such as different programme grammar, new control in Android X. So, it costs us a lot time immigrate front end and back end together.

Issue4: Integrate all the functions into one application It is a terrible process when trying to seam all parts together. It has taken us a long time to address issues related to

liberalism, implementation and control. For example, We found controls for Camera and Filter function are different, filter function need view extended GLSurfView, but in camera part we first use is a view extended TextureView.

Solution4: We decide to overwrite the view class try to modify it by implement SurfView some function. Unfortunately, It fails. I found we must push by SurfaceView's subclass because GLshader needed initialized by surfaceview's methods. We tend to overwrite the camera part to use SurfView.

b Usability of UI

Overall, the interaction of the project is divided into three parts: the login screen that gets the user to the home page (launch page, login page and registration page), the home and personal pages that showcase recommended live streams and personal information, and the live room screen that holds the core functionality of the product.
In order to give the product an overall higher profile, we developed the brand colour of the product, the bright yellow colour that is seen everywhere in the app. This colour scheme not only deepens the user's impression of the product, but also helps them to quickly discover some of the core jump functions in the interface that are used to provide a better interactive experience.
For the three screens in the login section - the launch page, the login page and the registration page - a banner was added to the launch page to quickly inform new users of the app's positioning and increase product recognition, and a mutual page jump function was added between both the login and registration pages to improve user tolerance.
For the product home page, the "cover + title + anchor information" information flow format, which is currently used by almost all live streaming platforms, was adopted to better convey the information of each live streaming room to users. The product's home page does not show a very detailed classification of content like the popular live streaming platforms bilibili and weibo, but simply divides the recommended content into popular and following, which can block out too much information classification and help users to quickly find their favourite live streaming rooms.
Finally, in order to improve the user experience of the core live streaming room, the interface is divided into two parts: the upper part is the screen that the user sees, and the lower part is the preview of the video and audio. This design cleverly provides a more complete and convenient way for anchors using mobile devices to broadcast live, effectively reducing the cost of live streaming. When an anchor wants to stream the bottom half of the preview to the viewer, they can simply click on the corresponding toggle button, and the addition of filters adds more interest to the live stream. In addition, the audio display of the current anchor has been added to the bottom right corner of the screen, which not only satisfies the aesthetic and minimalist aspects of the design, but also provides a more visual way for the anchor to know if the microphone in the current live room is working successfully, furthering the anchor-oriented user experience.

IV. Conclusion and Future Work

In summary, this project has successfully implemented the desired function proposed by the user in the market research. This project realizes the access of multiple external UVC-Cameras on Android devices. The UI of this project is carefully desigend according to the user need. Furthermore, important lessons have been learned that may have a significant impact on both our team members' fu-

ture development and professional abilities. In this procedure, a wide range of Android application components have been learnt. Furthermore, during the development, the ability to face obstacles and solve problems was enhanced, which will be very useful in future studies and work.In the future, this application can be further enhanced by improving the function of push stream, which makes the application more adaptable for live streaming platforms. Meanwhile, due to the limited time, the interface design can still be tweaked to make it more user-friendly.

References

[1] saki, 2 Oct 2018. [Online]. Available: https://github.com/saki4510t/UVCCamera.

[2] Deng zi [Online], https://plugins.jetbrains.com/plugin/13156-android-wifiadb

[3] Yury [Online], Avaliable: https://plugins.jetbrains.com/plugin/14969-adb-wi-fi