# 特效能力开发API中的背景分割

快速浏览：mediapipe事例的构建，参数说明、技术介绍与接口使用等。

时间：2020年11月30日

## Mediapipe安装

#### 写在前面

所有的命令和配置都是在**Ubuntu 20.10**下进行，如果你使用的是WSL，其过程与步骤大同小异，也许查看[Troubleshooting](https://google.github.io/mediapipe/getting_started/troubleshooting.html)或者~~科学上网~~能够解决你在安装过程中的大多数问题：

#### 步骤

* 首先，从github中clone mediapipe。

|  |
| --- |
| git clone https://github.com/google/mediapipe.git |

* 配置Bazel工具。

*“什么是Bazel？Google自开源的构建工具，类似Gradle，Bazel许多特性能够使它更好的适应Google的业务，以及构建Google庞大的代码库，Gradle和Bazel之间有一些不同，详细内容可以查阅官方文档，在这里需要用Bazel构建一些东西，不过大部分内容还是可以借助Gradle（应该是我们更熟悉的形式）来代替的。”*

**选择3.4或者以上版本**，如果你使用的是基于arm架构的机器，那么需要从源代码安装Bazel。其余架构的机器方式如下：

* 添加发行源

|  |
| --- |
| sudo apt install curl gnupg  curl -fsSL https://bazel.build/bazel-release.pub.gpg | gpg --dearmor > bazel.gpg  sudo mv bazel.gpg /etc/apt/trusted.gpg.d/  echo "deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list |

* 下载指定版本的Bazel

|  |
| --- |
| sudo apt install bazel-3.4.0 |

如果报错，无法找到package，那么更新索引后再次执行上面的命令：

|  |
| --- |
| sudo apt update |

安装完毕后可用bazel --version 检查安装是否成功。

* 下载OpenCV和FFmpeg，FFmpeg工具程序将通过libopencv-video-dev来安装。

|  |
| --- |
| sudo apt-get install libopencv-core-dev libopencv-highgui-dev \  libopencv-calib3d-dev libopencv-features2d-dev \  libopencv-imgproc-dev libopencv-video-dev |

或者你也可以通过手动安装，或者运行脚本程序setup\_opencv.sh来安装（在你下载mediapipe的根目录下，下同）。

检查WORKSPACE文件，查看以下内容的path是否对应安装的目录文件（如果你手动修改过那么需要更改路径），如果你没有更改过路径，那么可以忽略过这一步。

|  |
| --- |
| new\_local\_repository(  name = "linux\_opencv",  build\_file = "@//third\_party:opencv\_linux.BUILD",  path = "/usr/local",  )  new\_local\_repository(  name = "linux\_ffmpeg",  build\_file = "@//third\_party:ffmpeg\_linux.BUILD",  path = "/usr/local",  )  cc\_library(  name = "opencv",  srcs = glob(  [  "lib/libopencv\_core.so",  "lib/libopencv\_highgui.so",  "lib/libopencv\_imgcodecs.so",  "lib/libopencv\_imgproc.so",  "lib/libopencv\_video.so",  "lib/libopencv\_videoio.so",  ],),  hdrs = glob([  # For OpenCV 3.x  "include/opencv2/\*\*/\*.h\*",  # For OpenCV 4.x  # "include/opencv4/opencv2/\*\*/\*.h\*", ]),  includes = [  # For OpenCV 3.x  "include/",  # For OpenCV 4.x  # "include/opencv4/", ],  linkstatic = 1,  visibility = ["//visibility:public"],  )  cc\_library(  name = "libffmpeg",  srcs = glob(  [  "lib/libav\*.so",  ],),  hdrs = glob(["include/libav\*/\*.h"]),  includes = ["include"],  linkopts = [ "-lavcodec", "-lavformat", "-lavutil",],  linkstatic = 1,  visibility = ["//visibility:public"],) |

* 想要在桌面程序中获得GPU加速支持，需要下载：

|  |
| --- |
| sudo apt-get install mesa-common-dev libegl1-mesa-dev libgles2-mesa-dev |

注册**系统变量**：

|  |
| --- |
| export GLOG\_logtostderr=1 |

使用GPU加速运行helloworld程序：

|  |
| --- |
| bazel run --copt -DMESA\_EGL\_NO\_X11\_HEADERS --copt -DEGL\_NO\_X11 \ mediapipe/examples/desktop/hello\_world:hello\_world |

如果你的屏幕输出多行 *Hello world！*那么恭喜运行成功～

*“遇到问题了？⏩查看*[*Troubleshooting*](https://google.github.io/mediapipe/getting_started/troubleshooting.html)*或许能有帮助”*

## 文件夹中都有什么？

如果你已经安装好了mediapipe，并且成功运行了helloworld程序，那么这个教程已经完成了一半了！接下来，我们打开mediapipe项目目录，看看它都包含哪些文件，这些文件都是做些什么的呢？（里面或许包含我的一些理解）

打开mediapipe目录：

* calculator目录：这一部分包含程序构建所需的计算器，也是使得mediapipe能够实时运行的一部分原因，因为所有的计算器都是c++书写的。
* docs目录：教程文档。
* examples：mediapipe提供的实例程序。
* framework：包括框架处理的基本代码，例如输入输出流控制等。
* graphs：mediapipe比较重要的一部分（我认为是），流程图为pbtxt格式，可以通过谷歌的可视化网页展现程序的流程图，也可以自行设计与更改，它决定程序的输入输出流顺序，参数调整等。
* gpu: gpu加速的程序设计（C++）。

## 在Andtroid Stduio（AS）中开发mediapipe程序

官网提供了一份命令行方式搭建程序的步骤，但我们更希望能够在好用的IDE上开发我们的程序，例如Android Studio（下称AS），在这里采用aar引入我们需要的模块功能，以handtracking为例子：

* 首先我们新建目录（这个可以自己随便定义一个，但要记住你的这个路径）：

mediapipe/examples/android/src/java/com/google/mediapipe/apps/aar\_example/

新建BUILD文件，内容如下：

|  |
| --- |
| load("//mediapipe/java/com/google/mediapipe:mediapipe\_aar.bzl","mediapipe\_aar")mediapipe\_aar(name = "mp\_face\_detection\_aar", calculators = ["//mediapipe/graphs/hand\_tracking:mobile\_calculators"],) |

* 生成aar（这里需要填自己的路径，例如我刚才设置的路径为：

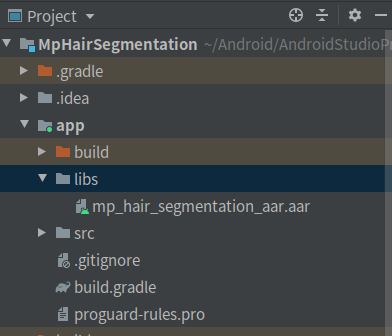
mediapipe/examples/android/src/java/com/google/mediapipe/apps/aar\_example/

那么构建的命令为：（双斜杠开头代表用绝对路径解析）

|  |
| --- |
| bazel build –c opt --host\_crosstool\_top=@bazel\_tools//tools/cpp:toolchain --fat\_apk\_cpu=arm64-v8a,armeabi-v7a \ //mediapipe/examples/android/src/java/com/google/mediapipe/apps/aar\_example:mp\_hand\_tracking\_aar |

这会在你的bazel-bin/mediapipe/examples/android/src/java/com/google/medaipie/apps/arr\_examples下生成相应的aar文件。

* 新建一个AS的项目，在将aar文件拷贝到app的libs目录下



* 构建Midiapipe所需的二进制图文件、模型文件等：

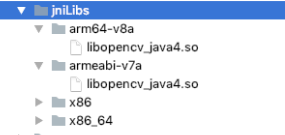
|  |
| --- |
| bazel build -c opt mediapipe/mediapipe/graphs/hand\_tracking:mobile\_gpu\_binary\_graph |

然后将文件拷贝进入项目的app/src/main/assets目录下，handtracking例子所需的文件包括hand\_landmark.tflite、 handedness.txt、mobile\_gpu.binarypb.

前两个分别位于mediapipe/models/目录下，后者位于graphs/hand\_tracking目录下。

*“如何知道我的项目需要哪些文件呢？这取决于你的graph图中的节点，如果你接受一个新的项目却不知道需要哪些依赖文件，可以查阅graph图中的每个节点中的信息（node中）”*

* 拷贝opencv的JNI库到app/src/main/jniLibs（没有的话新建一个），在这里[下载](https://github.com/opencv/opencv/releases/download/3.4.3/opencv-3.4.3-android-sdk.zip)对应库。



* 拷贝对应的依赖到app的gradle当中：

|  |
| --- |
| dependencies {  implementation fileTree(dir: 'libs', include: ['\*.jar', '\*.aar'])  implementation 'androidx.appcompat:appcompat:1.0.2'  implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  testImplementation 'junit:junit:4.12'  androidTestImplementation '  androidx.test.ext:junit:1.1.0'  androidTestImplementation '  androidx.test.espresso:espresso-core:3.1.1'  // MediaPipe deps  implementation 'com.google.flogger:flogger:0.3.1'  implementation 'com.google.flogger:flogger-system-backend:0.3.1'  implementation 'com.google.code.findbugs:jsr305:3.0.2'  implementation 'com.google.guava:guava:27.0.1-android'  implementation 'com.google.guava:guava:27.0.1-android'  implementation 'com.google.protobuf:protobuf-java:3.11.4'  // CameraX core library  def camerax\_version = "1.0.0-beta10"  implementation "androidx.camera:camera-core:$camerax\_version"  implementation "androidx.camera:camera-camera2:$camerax\_version"  implementation "androidx.camera:camera-lifecycle:$camerax\_version"} |

* 同步Sync之后，此时aar已经装载到了你的项目中了。

#### 处理你的视频帧

我使用了cameraX的部分功能来调用我们的相机，然后进一步进行处理，来书写我们的MainActivity主代码：

* 在manifest中请求调用相机：

|  |
| --- |
| <!-- For using the camera -->  <uses-permission android:name="android.permission.CAMERA" />  <uses-feature android:name="android.hardware.camera" /> |

* 修改SDK的最低版本：

|  |
| --- |
| <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="27" /> |

* 在你的MainActivity.java中的OnCreate方法中添加代码发起开启相机的一个请求，用来询问用户是否开启相机权限。

|  |
| --- |
| PermissionHelper.checkAndRequestCameraPermissions(this); |

* 当请求完成后，我们尝试处理这个请求， onRequestPermissionsResult方法会在请求之后自动调用，处理相应的请求，onResume在onPause等方法后调用，回复之前的状态，在这里我们恢复状态之前，首先检查摄像头的权限，如果权限开启，打开摄像头（startCamera函数还没有实现，我们可以先空下，后文补充）。

|  |
| --- |
| @Overridepublic void onRequestPermissionsResult(  int requestCode, String[] permissions, int[] grantResults) {  super.onRequestPermissionsResult(requestCode,permissions,grantResults); PermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);}  @Overrideprotected void onResume() {  super.onResume();  if (PermissionHelper.cameraPermissionsGranted(this)) {  startCamera(); }  }  public void startCamera() {} |

* 构建UI界面，在xml界面中添加布局（FrameLayout标签），用来承载我们的相机界面。

|  |
| --- |
| <FrameLayout  android:id="@+id/preview\_display\_layout"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:layout\_weight="1">  <TextView android:id="@+id/no\_camera\_access\_view"  android:layout\_height="fill\_parent"  android:layout\_width="fill\_parent"  android:gravity="center"  android:text="@string/no\_camera\_access" />  </FrameLayout> |

同时在res/values/string.xml中注册no\_camera\_access，值为 *请授予相机权限：*

* 接下来我们定义SurfaceTexture和SurfaceView两个变量，SurfaceTexture以OpenGL ES方式存储其纹理，SurfaceView可以在层级结构上绘制图样显示（安卓采用所谓的ViewGroup布局方式，详细内容参见官方文档），他们分别为perviewFrameTextrue和previewDisplayView声明，并且作为类成员变量在MainActivity.java中：

|  |
| --- |
| private SurfaceTexture previewFrameTexture;  private SurfaceView previewDisplayView; |

定义setupPreviewDisplayView方法，这个方法将我们之前定义的preiewDisplayView添加到布局当中：

|  |
| --- |
| private void setupPreviewDisplayView() {  previewDisplayView.setVisibility(View.GONE);  ViewGroup viewGroup = findViewById(R.id.preview\_display\_layout); viewGroup.addView(previewDisplayView);} |

* 接下来声明CameraXPreviewHelper方法，他会监听通过SurfaceTexture启动的相机并且在这之后执行：

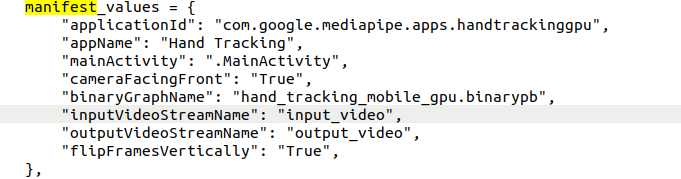
|  |
| --- |
| private CameraXPreviewHelper cameraHelper; |

然后我们完善相应的startCamera代码：

|  |
| --- |
| public void startCamera() {  cameraHelper = new CameraXPreviewHelper();  cameraHelper.setOnCameraStartedListener(  surfaceTexture -> {  previewFrameTexture = surfaceTexture; previewDisplayView.setVisibility(View.VISIBLE); });} |

我们在这里添加了一个匿名的监听器，当相机开启，并且surfaceTexture格式的帧传入的时候，我们就把他存储起来（previewFrameTexture中）。

* 注册BUILD中的变量，如果你注意到mediapipe中的example中的源代码，你会发现在MainActivity中并没有声明对应的输入输出流变量，这是由于其全都注册在BUILD文件下，我们打开BUILD文件，会发现：



但这并不是我们想要的格式，实际上我们想通过gradle构建，而不是bazel，那么我们就需要手动在原来的MainActivity.java当中声明对应的变量（类变量）：

|  |
| --- |
| private static final String BINARY\_GRAPH\_NAME = "mobile\_gpu.binarypb"; private static final String INPUT\_VIDEO\_STREAM\_NAME = "input\_video"; private static final String OUTPUT\_VIDEO\_STREAM\_NAME = "output\_video"; private static final CameraHelper.CameraFacing *CAMERA\_FACING* = CameraHelper.CameraFacing.*FRONT*;  private static final boolean FLIP\_FRAMES\_VERTICALLY = true; |

需要添加的内容按照你的需求定义，当然就是你的图（Graph），这里是handtracking需要添加的内容。

在startcamera方法后面添加语句，开启前置摄像头：

|  |
| --- |
| cameraHelper.startCamera(this, *CAMERA\_FACING*, /\*surfaceTexture=\*/ null); |

* 在Mediapipe只能处理普通的Open GL纹理对象，但是我们得到的是es版本，所以需要转

换，这里Mediapipe提供了一个类来帮助我们实现这个功能： ExternalTextureConverter。同时为了实现这个转换，我们还需要一个管理器EglManager。

|  |
| --- |
| private EglManager eglManager;  private ExternalTextureConverter converter; |

在OnCreate方法中，我们在请求权限之前将eglManager初始化：

|  |
| --- |
| eglManager = new EglManager(null); |

还记得我们之前定义的onResume方法吗？就像我刚才说的Mediapipe只能处理普通的OpenGL格式，所以我们在调用之前将converter初始化，添加以下语句在check之前：

|  |
| --- |
| converter = new ExternalTextureConverter(eglManager.getContext()); |

我们还需要一个暂停的功能，如果我们的应用切入后台了呢，这时候我们的converter应该结束了！

|  |
| --- |
| @Override  protected void onPause() {  super.onPause();  converter.close();  } |

* 下来开始我们的转换，就是把previewFrameTexture对象变成converter，阅读以下代码并

且添加到setupPreviewDisplay当中：

|  |
| --- |
| previewDisplayView .getHolder() .addCallback(  new SurfaceHolder.Callback() {  @Override  public void surfaceCreated(SurfaceHolder holder) {}  @Override  public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  // (Re-)Compute the ideal size of the camera-preview display (the area that the // camera-preview frames get rendered onto, potentially with scaling and rotation) // based on the size of the SurfaceView that contains the display.  Size viewSize = new Size(width, height);  Size displaySize = cameraHelper.computeDisplaySizeFromViewSize(viewSize); // Connect the converter to the camera-preview frames as its input (via // previewFrameTexture), and configure the output width and height as the computed // display size.  converter.setSurfaceTextureAndAttachToGLContext(  previewFrameTexture, displaySize.getWidth(), displaySize.getHeight()); } @Override  public void surfaceDestroyed(SurfaceHolder holder) {} }); |

当surface可见的时候那么必须要实现create和destroy方法。设置change方法用来自适应我们的摄像窗口。

* 在MainActivuity中使用你所需要的图graph，就是之前生成的二进制文件。在这里需要用

一个资产管理器，你可以理解为管理之前的assets目录下文件的程序。它同样需要初始化(他需要在eglManager初始化之前，因为有了资产管理才能处理图像)：

|  |
| --- |
| AndroidAssetsUtil.initializeNativeAssetManager(this); |

* 现在你需要设置一个帧处理器来处理相机帧，这些帧现在都是普通的opengl格式，我们在处理之后更新相应的previewDisplay即可：

|  |
| --- |
| private FrameProcessor processor; |

在eglManager之后我们开始初始化processor：

|  |
| --- |
| processor = new FrameProcessor(  this,  eglManager.getNativeContext(),  BINARY\_GRAPH\_NAME,  INPUT\_VIDEO\_STREAM\_NAME，  OUTPUT\_VIDEO\_STREAM\_NAME); |

在播放帧的时候，我们需要使用这个processor才能达到我们想要的效果，所以我们需要在初始化converter之后（onResume函数）设置使用这个processor参数，这个方法很形象，叫做setConsumer()：

|  |
| --- |
| converter.setConsumer(processor); |

好了现在马上就要结束了，还记得我们下一步的操作吗，就是更新我们的previewDisplayView界面，调用下列函数在创建和销毁时设置更新，特别的是，销毁的时候只需要将参数设置为null就可以了：

|  |
| --- |
| @Override  public void surfaceCreated(SurfaceHolder holder) { processor.getVideoSurfaceOutput().setSurface(holder.getSurface());}  @Override  public void surfaceDestroyed(SurfaceHolder holder) { processor.getVideoSurfaceOutput().setSurface(null);} |

结束，如果你做了任何gradle的更改，别忘了更新！然后再运行，因为mediapipe自身原因，貌似对虚拟安卓的支持不太好，我建议直接usb连接真机测试。