|  |  |
| --- | --- |
| **成绩** | （采用四级记分制） |



****

**本科毕业论文（设计）**

**题目：** 雾霾图像的清晰化处理

**学生姓名 李文驰**

**学 号 2015118140**

**指导教师 范巡礼**

**院 系 信息科学与技术学院**

**专 业 软件工程**

**年 级 2015级**

**教务处制**

诚信声明

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或在网上发表的论文。

特此声明。

论文作者签名：

日 期： 年 月 日

摘 要

随着当代科技的不断发展，近些年来的空气污染愈发严重，随之而来的雾霾现象以及原有的浓雾天气、水汽现象，都为人们的生活带来了种种不便。本篇论文将介绍一个以暗通道先验规律为理论基础的图像去雾应用程序。暗通道先验规律是一种简单但是十分有效的先验规律。它指的是，在户外的无雾图像中存在这样的一些像素，它们至少在一个颜色通道下（红、绿、蓝），强度十分的低。利用这个先验规律和雾图像形成模型可以直接估计雾浓度并且恢复出无雾的图像。这个应用程序使用B/S架构，采用了传统的前后端分离思想。其中，前端采用渐进式MVVM框架进行构建，它负责数据的组织与用户界面的呈现；后端负责单利用去雾算法进行有雾图像的清晰化处理；在前后端之间，对图片进行Base64编码后，采用Ajax进行数据通信。

关键词**:** 单幅图像去雾; 暗通道先验规律; Web应用设计

**ABSTRACT**

With the continuous development of technology, air pollution has become more and more serious in recent years, and the haze phenomenon and the original dense fog weather have brought inconvenience to people's lives. This paper will introduce a single image haze removal application based on the dark channel prior. The dark channel prior is a simple but effective a priori. It refers to the fact that there are some pixels in the outdoor haze-free image, which are at least one color channel and have very low intensities. Using this prior and the haze imaging model, the haze thickness can be directly estimated and a haze-free image can be recovered. This application uses the B/S architecture and adopts the traditional idea that front-end and back-end are separated. The front-end use the progressive MVVM framework, which is responsible for data organization and user interface, and the back-end is responsible for removing the haze from single images. The image is encoded by BASE64 and transported by AJAX between front-end and back-end.

**Keywords:** single haze removal; dark channel prior; web application design

目 录

1. 序言
   1. 设计背景
   2. 设计目标
2. 需求分析
   1. 功能描述
   2. 需求概述
3. 系统设计
   1. 用户界面设计
   2. 接口设计
4. 核心模块及系统实现
   1. 去雾模块
      1. 算法介绍
      2. 暗通道计算
      3. 大气光值计算
      4. 透射率计算
      5. 代码优化
   2. 文件上传下载模块
      1. 文件上传
      2. 文件下载
5. 去雾效果
6. 系统安装与调试
7. 小结

参考文献

1 序言

1.1 设计背景

随着科学技术日新月异的发展，环境污染是一个我们不得不面对的问题。近些年来，尤其是在北方地区的冬季，雾霾现象频发，已经严重影响到了我们的正常生活。雾霾现象的频繁发生对日常生活造成了极大的不便，尤其是对于交通、摄影等行业。例如，在雾霾指数严重的情况下，交通摄像头可能无法清晰地拍到违章车辆的车牌号；航班无法正常起飞、降落；摄影工作者在室外无法正常地取景、拍摄。还有很多诸如此类的情况，但其实不只是雾霾，在水雾、烟气等环境下，如何获得清晰的图像、视频便成为了一个亟待解决的重要问题。

在计算机视觉领域，对图像去雾的研究一直没有停歇。2009 年何恺明的一篇论文《Single Image Haze Removal Using Dark Channel Prior》获得了计算机视觉领域三大国际会议之一CVPR的最佳论文奖。文章中提到了暗通道先验去雾理论，并获得了国际的认可，这也为本设计选题提供了理论基础与实证准备。

1.2 设计目标

图像去雾可以提高图像的清晰度，还原图像本身的色彩，相当于一个预处理，为接下来的图像处理工作打好基础，例如图像识别、物体检测等都至少需要不过于失真的样张。本文以暗通道先验去雾理论为算法基础进行软件设计，从而给出完整的符合现代审美的Web SPA（Single Page Application），以进行单幅图像去雾，并且要达到软件设计的可用性、易用性、有效性等要求。同时，对页面进行响应式设计，对不同屏幕尺寸的终端设备进行显示适配，给用户带来更加方便、满意的用户体验。

2 需求分析

2.1用户需求分析

* 用户上传待去雾图片到系统
* 用户在线预览去雾结果
* 用户下载去雾后的图片

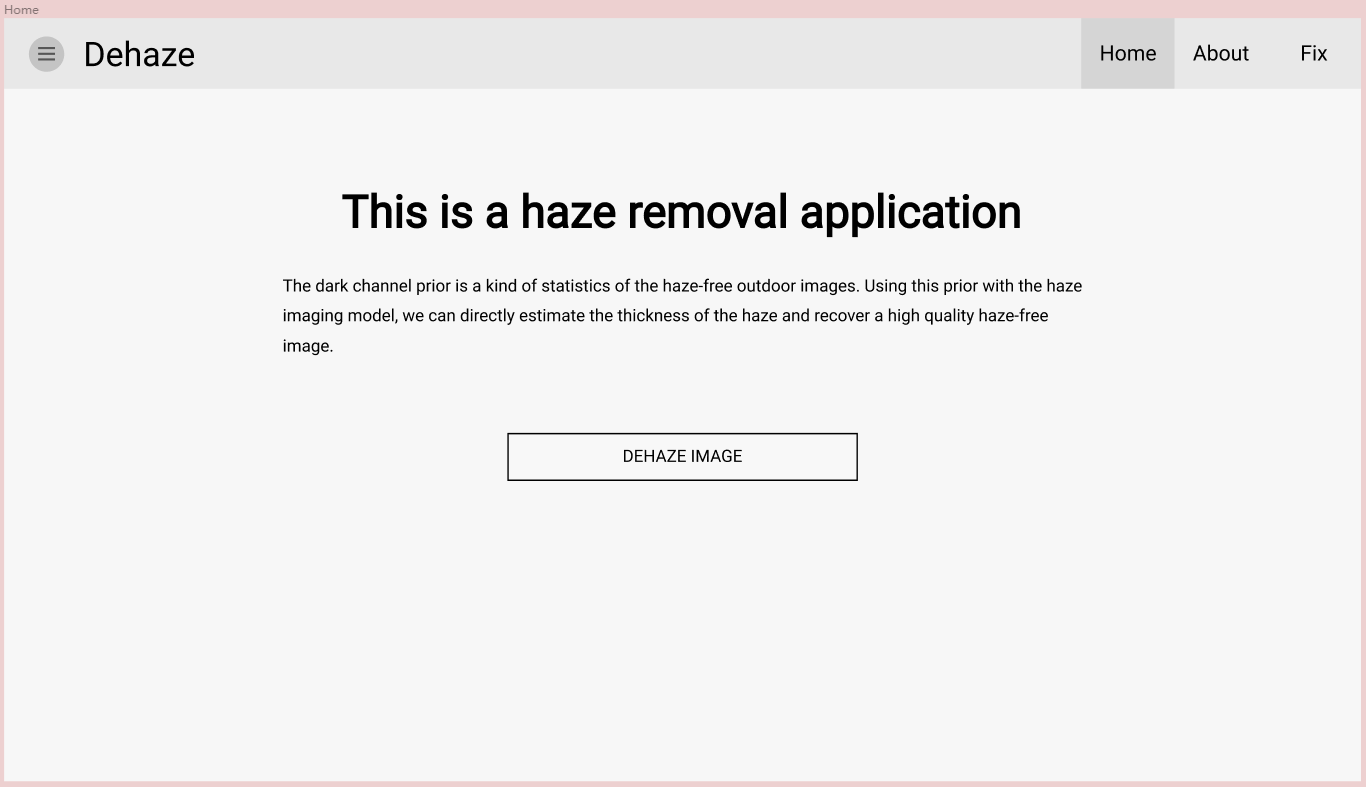
2.2功能描述

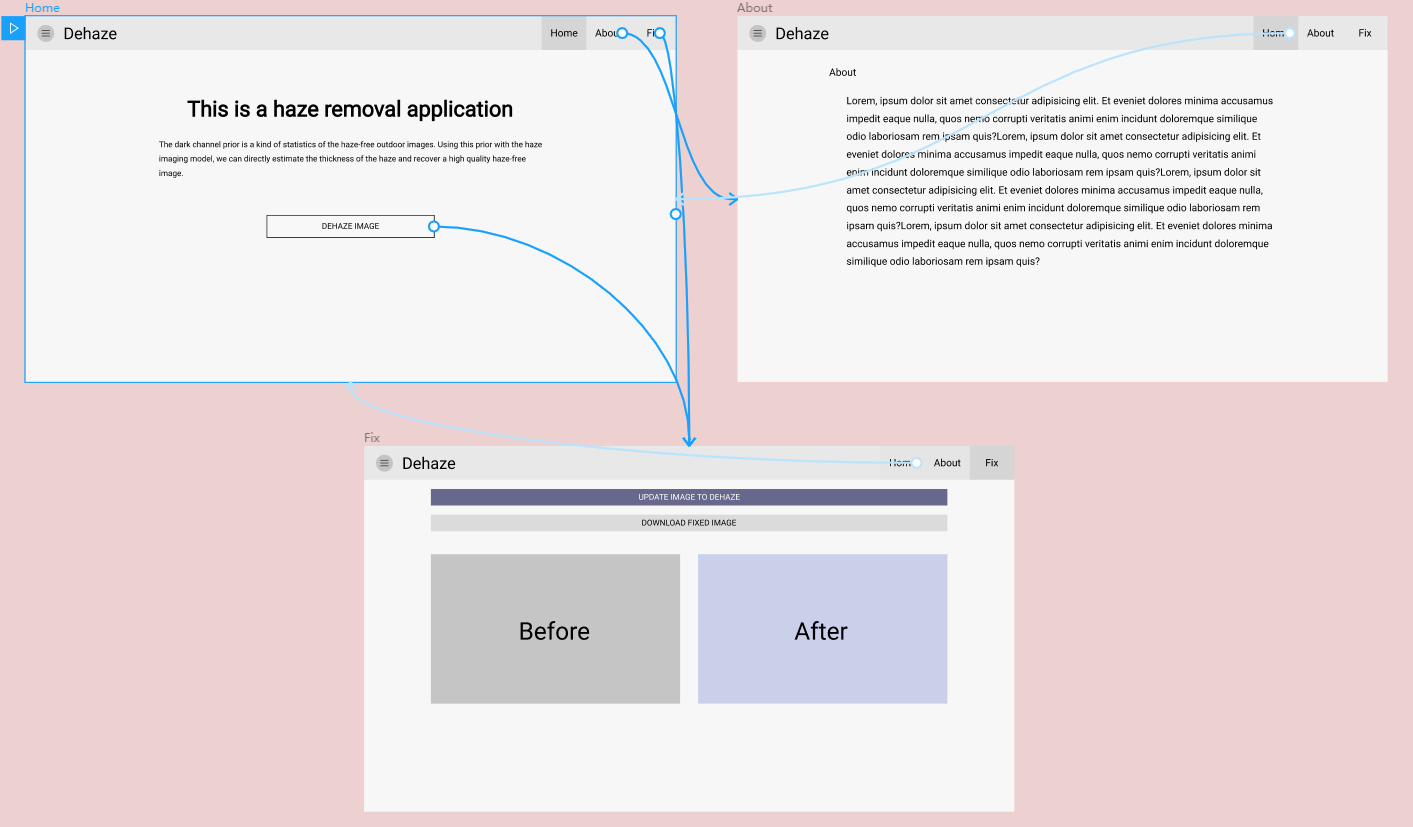
* 前端页面自动适配各种尺寸的终端设备
* 支持用户上传待去雾图片
* 保存用户上传的待去雾图片到服务器的静态文件库
* 后台利用算法对待去雾图片去雾，并保存到服务器的静态文件库
* 支持用户在线预览去雾后的图片
* 支持用户下载去雾后的图片

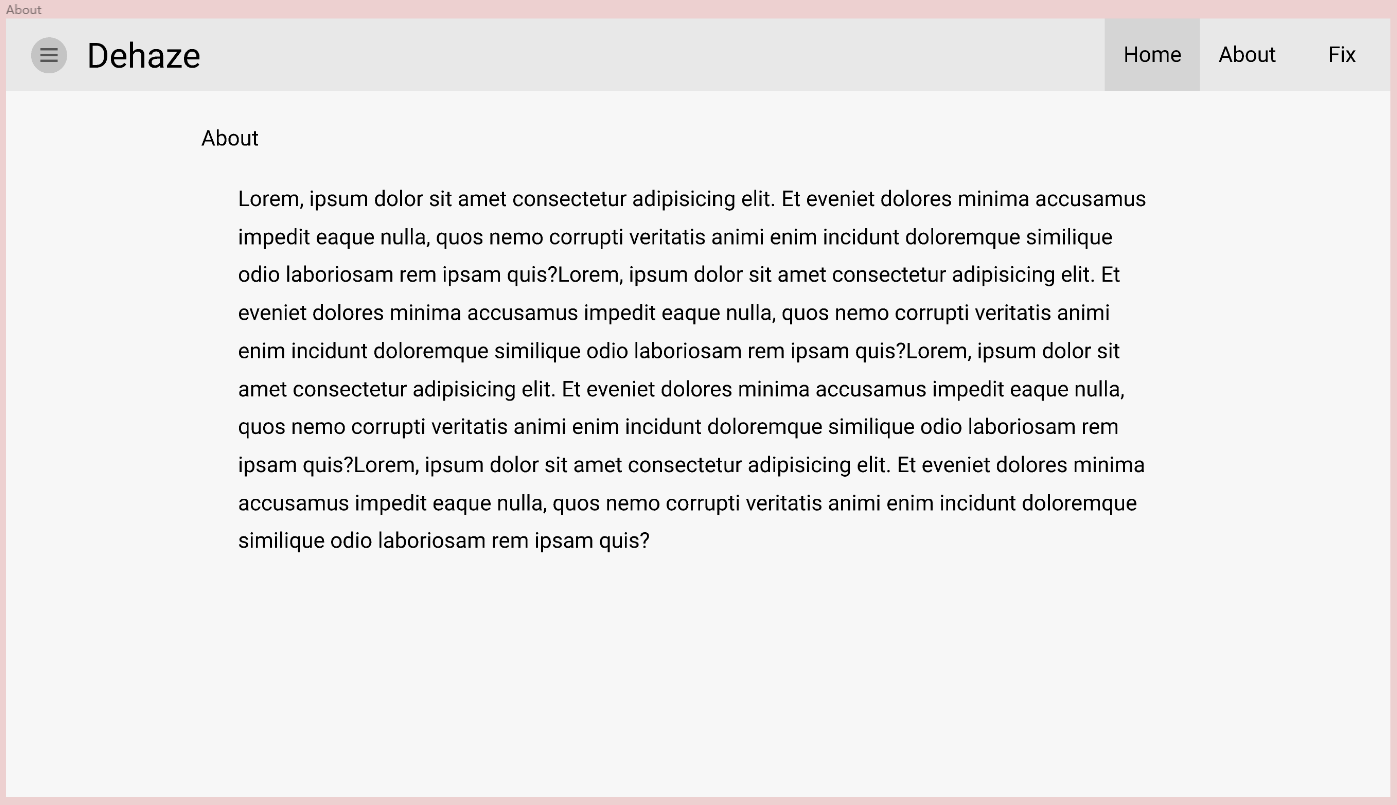
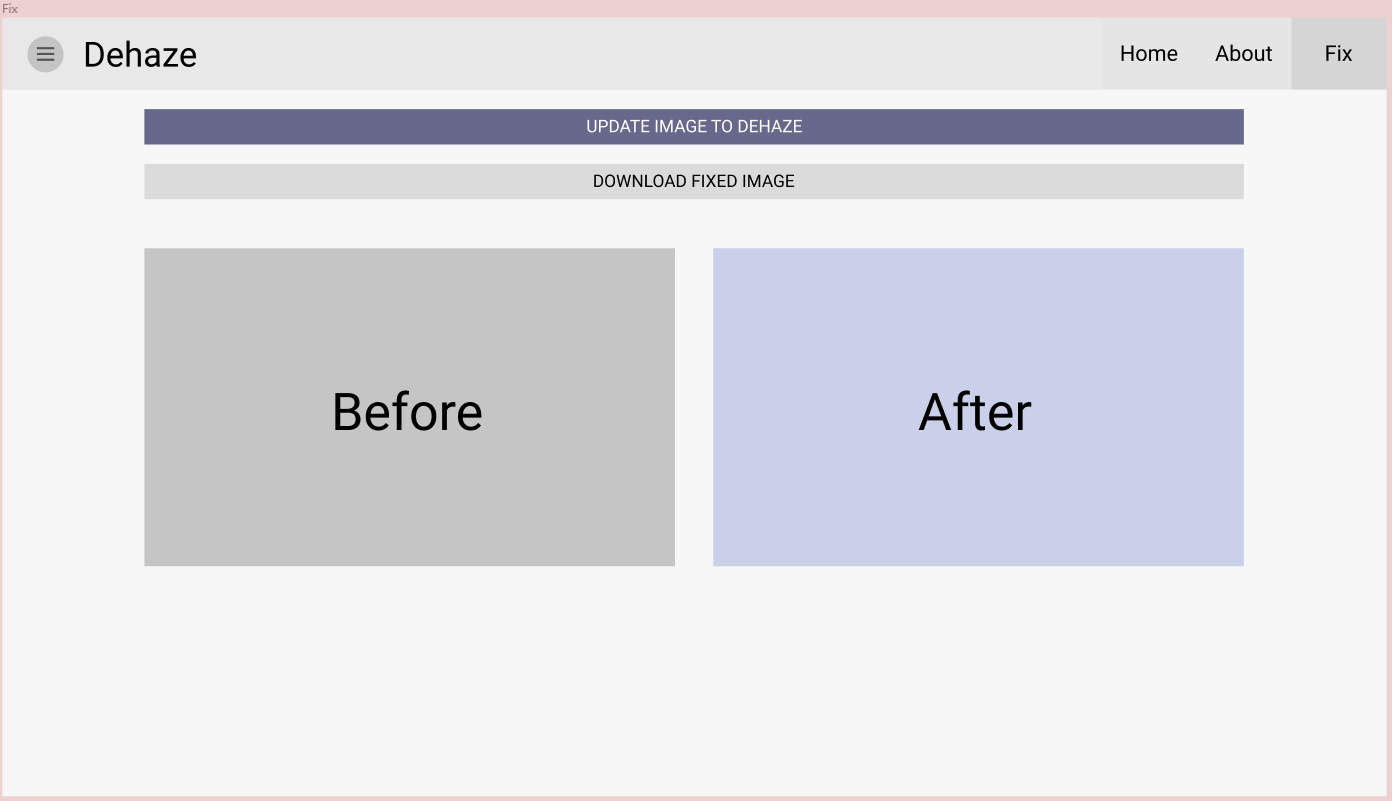
3 系统设计

3.1 用户界面设计

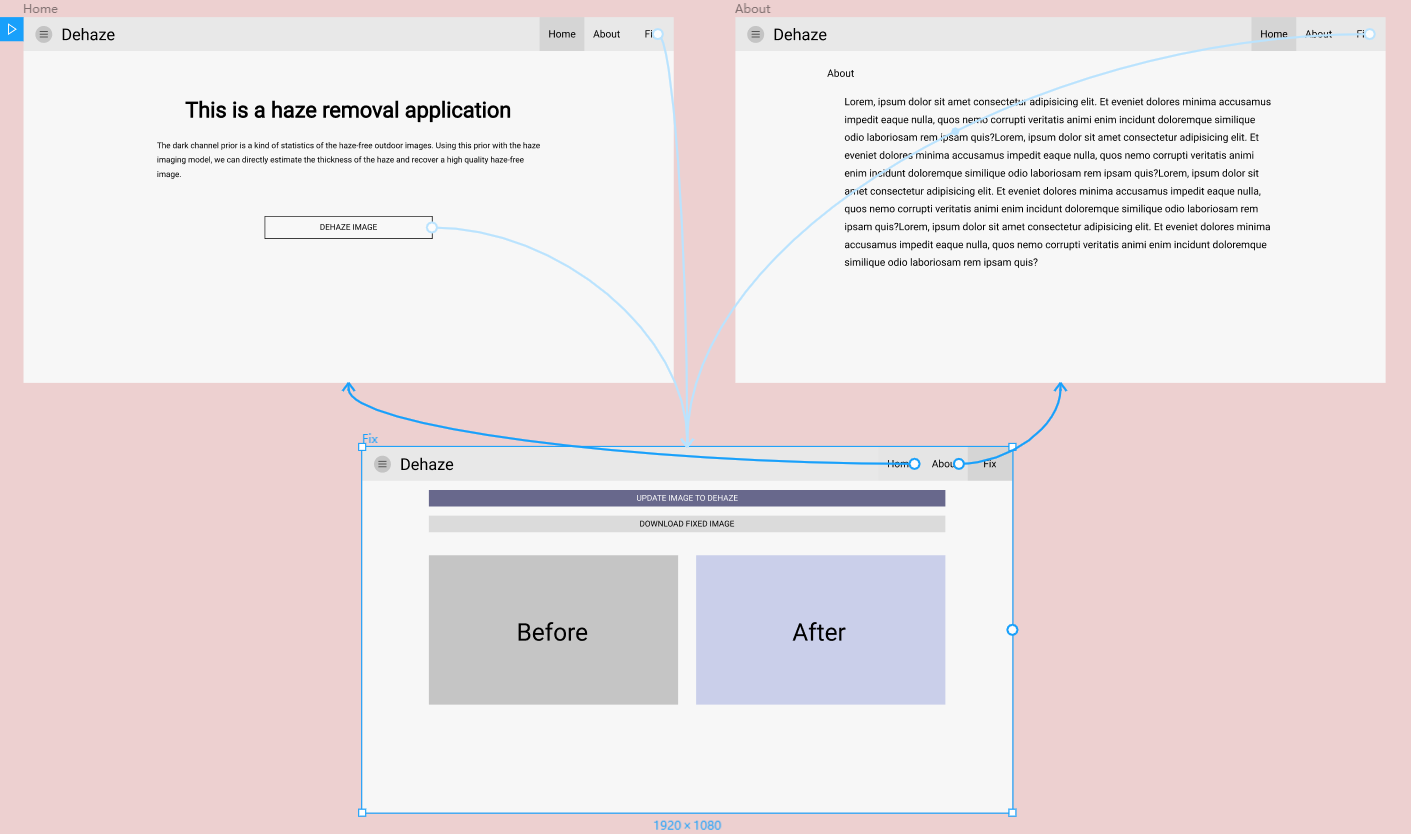
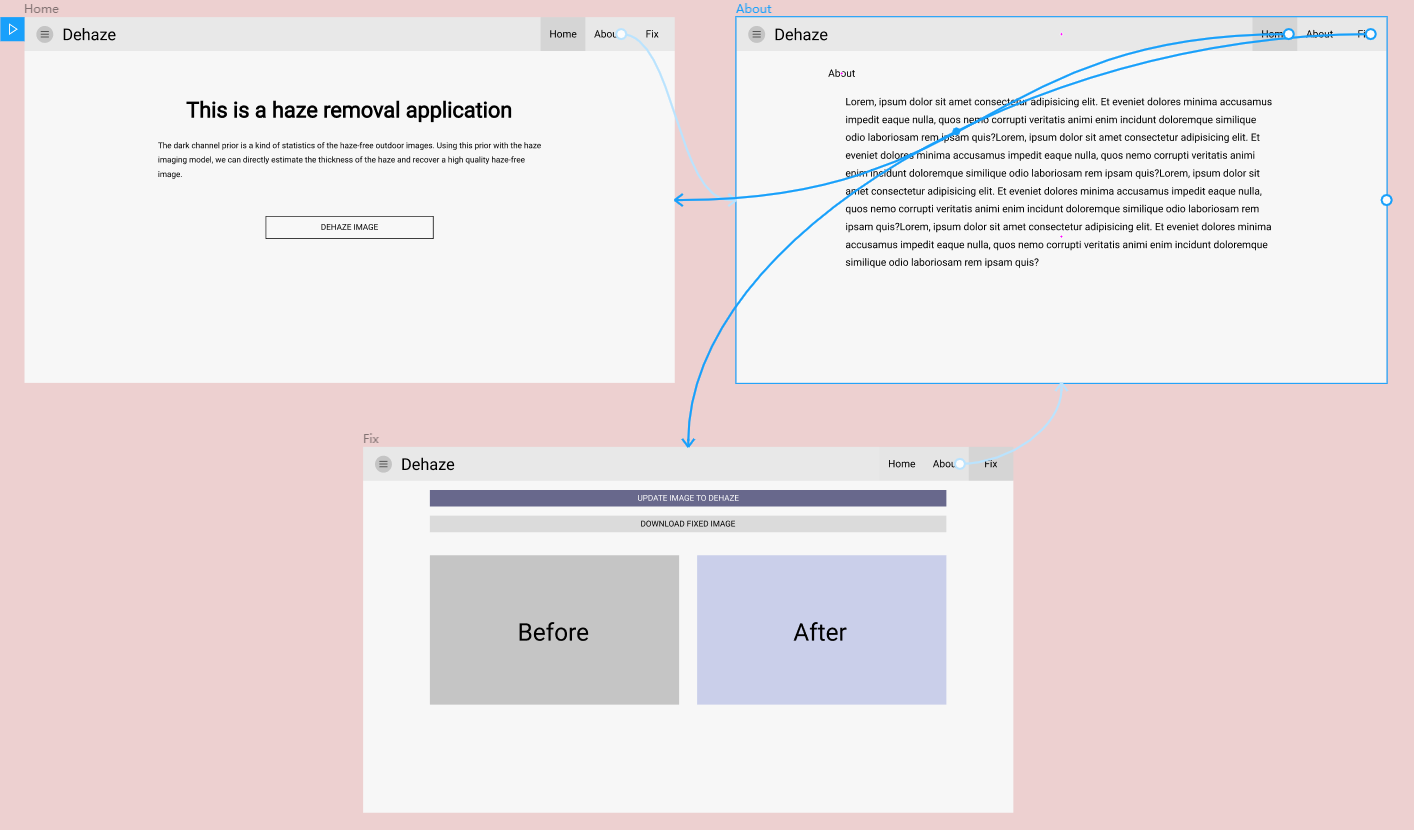
该SPA主要由四个页面构成，Home页面、About页面、Fix页面、Page not found页面。图片3.1(a) 是Home页面的草图，图片3.1(b)是Home页面到其他页面的跳转逻辑图。图片3.2(a)是About页面的草图，图片3.2(b)是About页面到其他页面的跳转逻辑图。图片3.3(a) 是Fix页面的草图，图片3.3(b)是Fix页面到其他页面的跳转逻辑图。其中Fix页面也是这个软件核心逻辑所在的页面，从上至下、从左至右分别是上传图像button、下载图像button、待去雾的原图和去雾后的图像。Home页面右上角的About标签和Fix标签可以分别跳转到About页面和Fix页面，中间的Button也可以跳转到Fix页面。其余跳转逻辑如图3.2(b)、3.3(b)所示。

3.1(a)Home页面

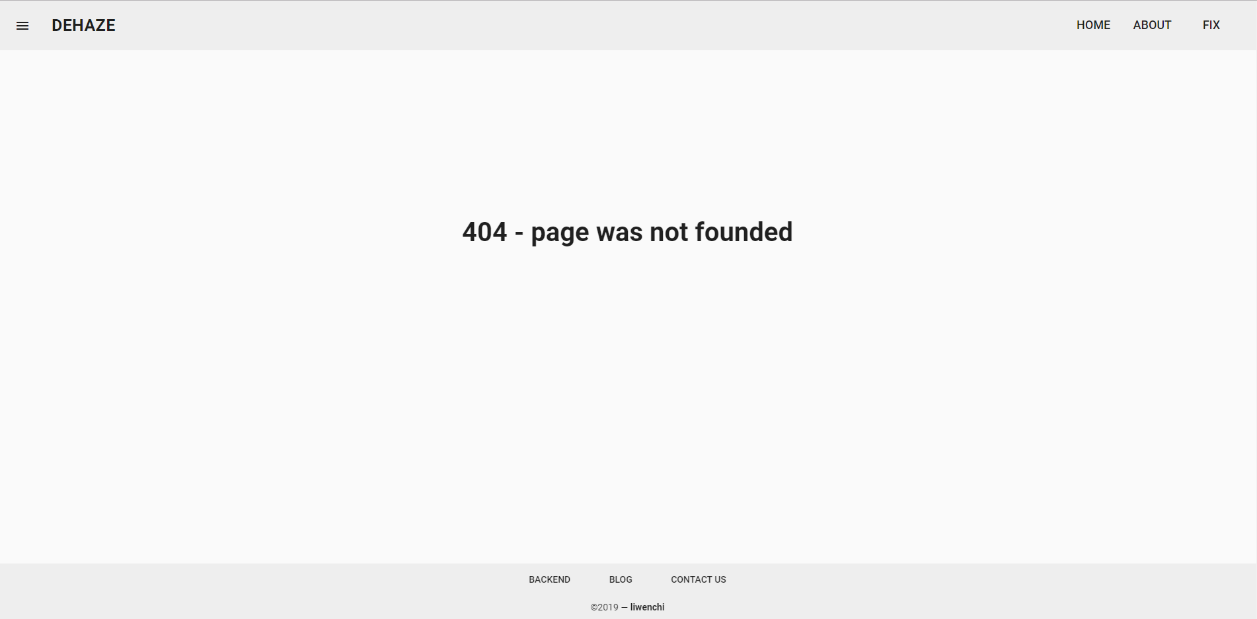
3.1(b)Home页面的跳转逻辑

3.2(a)About页面

3.2(b)Home页面的跳转逻辑

3.3(a)Fix页面

3.3(b)Fix页面的跳转逻辑

3.4notfound404页面

3.2 接口设计

前端页面：

http请求方式：GET

urls：localhost:8080//[/.\*/]

返回值：SPA的html页面

注意，由于设计了404页面，不论端口号后面跟什么地址，都可以找到对应的SPA。

当相对url路径为’/’时返回Home页面，如图3.1(1)。

当相对url路径为’/about’时返回About页面，如图3.2(1)。

当相对url路径为’/fix’时返回Fix页面，如图3.3(1)。

当相对url路径为任意其他时时返回404 Page not found页面，如图3.4所示。

后端接口：

http请求方式：POST

url：localhost:8081//dehaze/

参数：[file]:file的Base64编码

返回值：图片的base64编码

4 核心模块及系统实现

4.1 去雾模块

4.1.1 算法介绍

暗通道先验规律是一种从大量的户外无雾图片中统计得出的统计学规律，它的具体内容是指：在无雾图像中，处于大多数的非天空局部区域里，几乎总是有一些像素在至少一个颜色通道中有着非常低的强度，这些像素被称为“暗像素”。在有雾图像中，这些颜色通道里的暗像素的强度主要由大气光提供。因此，这些暗像素可以用来精确估算雾的透射率和计算大气光。这样一来，就可以借助雾图像模型来恢复出高质量的无雾图像。

4.1.2 计算暗通道

算法的第一步是计算图像的暗通道。暗通道先验基于如下的规律：在大多数的非天空像素块中，至少有一些像素在某个颜色通道中具有非常低的强度，换句话说，在这样的像素中，具有最低强度的那个通道的强度将非常的低。因此，对于一个图片J，它的暗通道这样定义

公式中的 为J的一个颜色通道，c的取值是R、G、B中的一个值（红、绿、蓝）。公式中代表以像素x为中心的一个像素块，这个像素块被称为窗口，窗口一般为正方形，实际上可取任何形状，在这篇论文中统一取为15\*15的正方形。根据公式的描述，应该是一个二维数组，且数组的取值范围应该在0到255之间。进一步来说，如果J是一幅户外的无雾图像，根据之前提到的暗通道先验规律，的强度应该非常的低，并且这个二维数组里面的绝大部分的数的值应该非常地接近0。因为在无雾图像中，总有一些区域存在阴影或暗色的物体，当第一次对某个像素周围的窗口取最小值时，无论当前像素的强度有多高，只要周围有暗像素，那么这个像素的各个通道的强度都将被降低，这其实就是最小值滤波的原理。第二次取最小值就是对一个三通道的像素点，从它的三个通道中取最小值并作为单一替代值。例如一个R、G、B通道分别为(15,155,176)的像素点，当进行第二次取最小值之后，将变为15。也就是说，这个图像从一个彩色图像经过两次取最小值操作之后变为了一个强度极低的灰度图像。

4.1.3 计算大气透射率

首先我们引入有雾图像形成模型，这是一个在计算机视觉和计算机图形中广泛运用的模型：

其中，是待去雾图像，是无雾图像，是大气透射率，A是大气光值。我们先假设大气光A已知，的值为，对等式两边同时取最小值：

注意到取最小值操作对于三个颜色通道都是独立的，因此，我们可以在等式两边同时分别对三个通道除以，所以，这个公式又等价于：

然后，再在R，G，B三个颜色通道中取强度的最小值：

又根据之前得出的结论：

因此，可以得出：

但是，即使是非常晴朗的天气，空气也不是百分之百纯净的，在这个雾形成模型中，可以理解为空气中仍然有一层薄雾，因此我们给等式右边的减数一个参数来限制。

的取值在0到1之间，越接近1，去雾效果就越“好”，图片就越鲜艳。但是，若想还原真实的场景，图片不一定越鲜艳越好，因此本文中图片去雾的设置为0.95。

4.1.4 估算大气光

大多数先前的方法估算大气光是通过那些雾浓度最高的像素。但是在实际的图像中，那些最亮的像素可能出现在白色的汽车以及它的反光上，或者是一些纯白的物体。在这里，又体现出了暗通道的作用，因为暗通道可以很好地展现雾的浓度。

为了避免受到极端值的影响，我们首先取暗通道中前0.1%亮的像素，这些像素的位置对应着原图中雾浓度最高的像素的位置。这些像素的强度即可代表输入图像的的全局大气光。我们在原图中遍历这些像素，对像素的强度取平均值，即可求出全局大气光A。然后再把A带回上小节末尾的公式，即可求出透射率。

4.1.5 代码优化

任何图像说到底其实是一个width\*height\*3的数组，分别是图片的宽、高、三个颜色通道。因此，图像的像素操作的速度是一个我们必须考虑的因素，即使是常见的1920\*1080的低分辨率图像，也有2,000,000个像素点要处理，如果是Large级别的图像，需要操作的像素点甚至会多达17,900,000个，三个通道分别构成的矩阵之间若要再进行矩阵基本运算，其复杂程度以及耗时令人感到担忧。

因此，对于传统嵌套循环的运算速度过于缓慢的情况，我们引入了Numpy这个计算工具库。Numpy作为一个成熟的数值计算工具，尤其是对于多维大型矩阵的处理方面十分高效。

1. 关于暗通道计算的优化

对于两次取最小值，相当于一次最小值滤波和一个在第三维上的最小值压缩。若用传统的三重循环，对于低分辨率图像，只是一个求暗通道的操作，都要至少计算千万次，这种长耗时是我们无法接受的。我们可以直接使用numpy.min(arr, axis=3)。经过实际测试，代码在性能上有着极大的提升。

对于最小值滤波，opencv也提供了腐蚀操作erode，它在效果上和最小值滤波一模一样。注意，在进行腐蚀操作的时候需要两个参数，和之前提到的窗口有关，第一个是窗口的形状，第二个是窗口的尺寸，在这里统一取为15\*15的正方形。

1. 关于大气光计算的优化

计算大气光要求我们计算出暗通道图像中前0.1%亮度的像素的位置，并在中找到对应的像素点。我们首先想到利用小根堆求前k大值。对这个方法进行一下时间复杂度分析：假设图片的像素数目是n，初建堆的时间复杂度为，接下来把其他的数遍历并和堆顶元素比较，若当前元素比堆顶元素大，则进行替换并重建堆。总时间复杂度大约为:，其中。但是仍然无法避免大量的数组操作，然而大量的列表操作是非常缓慢的，根据实际测试，仅仅是1080p的图片，完成一次大气光的计算就需要13秒左右，这根本无法让人接受。

因此，我们再次利用numpy的直方图函数，求出不同颜色通道每个区间像素强度的数量，然后求出前缀和，并除以像素总数来算出小于99.9%的临界点以及对应的平均大气光。这一操作，使得整个算法跑进了1s之内。

1. 关于透射率计算的优化

上述提到的透射率计算方法，是一种粗略的计算，如果直接打印透射率图会发现，在物体的边缘存在较为明显的白边，因此可以利用soft matting算法来进行透射率图的精细化。但是，这一算法的速度十分缓慢，后来，何又提出了导向滤波来进行精细化的加速。

4.2 文件上传下载模块

4.2.1 文件上传模块

这一部分是用原生JavaScript实现的，但是在样式方面，vuetify中并没有现成的upload组件，因此，我们不得不先自己包装一个upload组件。

先利用一个原有的带CSS样式的button作为模板，并为其添加EventListener，使其监听click事件。然后，创建一个input标签，设置其type[‘file’]，并给一个id，最关键的一步就是设置input标签的display属性为none。这样一来，当用户点击button时，click事件被事件监听器捕获，并触发已经被隐藏了的input标签的click事件，从而唤醒浏览器的文件选择器，完成一次完整的文件选择操作。

到了这一步其实只是将文件传入了input标签，我们还需要利用ajax向后台提交数据。利用JavaScript创建Image对象、FileReader对象和XMLHttpRequest对象。这里补充一个前置知识：一个img标签想要显示图片，不但可以将src的内容填充为图片的路径信息，还可以将src的内容填充为图片的Base64编码。因此，我们要利用JavaScript来完成图片到Base64编码的转换。具体做法是：先创建三个对象，分别是Image对象、FileReader对象、XMLHttpRequest对象。利用FileReader.readAsDataURL来读取file，其中触发的匿名函数的形式参数e.target.result就是图片对应的Base64编码，接下来将e.target.result赋值给Image对象的src属性，即可在页面上显示用户刚刚上传的图片。最后一步，利用XMLHttpRequest对象将图像文件传给后台。注意，由于图片的Base64编码过长，这里使用POST方式来传递参数，因此我们在利用ajax前还需要使用Formdata对象，在把图片装入Formdata后即可正常发送。

4.2.2 文件下载模块

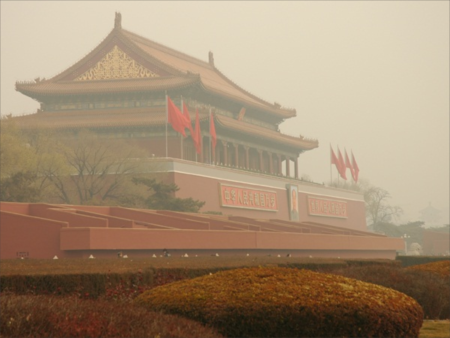
当系统对图像完成去雾以后，会将去雾后的图片编码为Base64编码并回传给前端。这时，根据上文提到的显示图片的第二种方法，将此Base64编码赋值给一个新的Image对象的src属性。

创建一个Button来实现下载功能。为此Button添加click事件监听，当按钮被点击时，利用JavaScript创建一个虚拟的a标签，并将去雾后的图片的Base64编码赋值给这个a标签的src属性，最后，触发a标签的click方法将会自动调用浏览器的下载模块，对图片进行下载。

5 去雾效果

下面的几张图片将展示该系统的去雾效果：





6 系统安装与调试

6.1 系统安装

整个系统分为前端部分和后端部分，两个模块分开开发，因此，在安装的时候也要分开安装。

安装前端部分：

1. 安装Node环境。在安装好Node环境以后，接下来前端部分的安装都可以使用Node自带的包管理器npm进行安装。
2. 安装express^4.16.4。这是一个非常灵活，并且十分轻量级的Web开发框架
3. 安装nodemon^1.18.10。这是一个可选项，Nodemon可以帮助开发人员更好地开发Node应用程序，它其实是通过监测目录中的文件变化来进行热更新，是一个十分优秀、好用的Node包装器。
4. 编写服务启动文件server.js，并指定到8080端口。
5. 启动服务。nodemon server.js。

安装后端部分：

1. 安装python^3.7。在安装好Python运行环境以后，接下来后端部分的安装都可以使用python自带的包管理器pip进行安装。
2. 安装Django^2.1.7。这是一个高集成度的Python网络框架，能极大的提升开发人员的工作速度和工作效率。
3. 安装django-cors-headers。这是一个为了完成跨域请求的库。
4. 安装opencv-python。这是一个Python版本的opencv库，用来进行图像处理。
5. 启动服务到8081端口。python manage.py runserver 8081。

6.1 系统调试

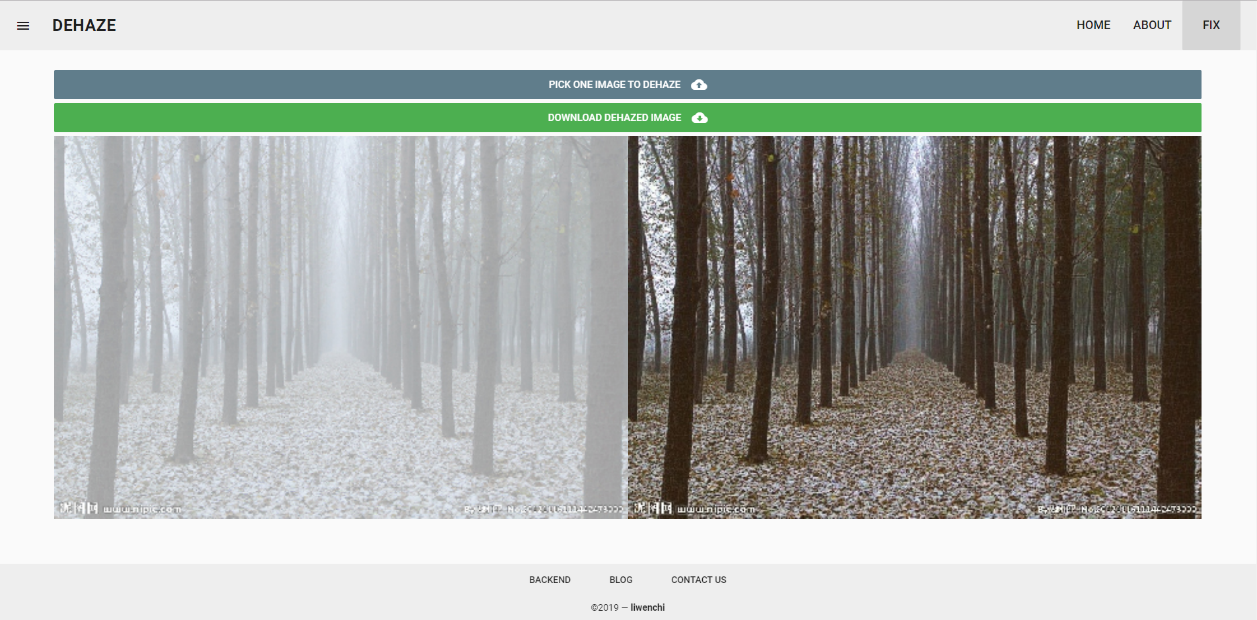
前后端都安装完成后，进行本地测试。先测试前端部分，将前端应用启动在8080端口。打开浏览器并输入：localhost:8080，应该出现如图3.1(a)的页面。若显示正常，则测试成功。接下来测试后端部分，先将后端应用启动在8081端口。然后在网页中点击页面内中间的button，进入如图3.3(a)的界面。点击上传图片，等待若干秒后，网页应显示两张图片，分别是去雾前的图片和去雾后的图片，若和下图相似，并且点击下载后，图片正常地保存在了计算机上，则后端测试成功，系统测试完毕。

图6.2(a)一次去雾完成后的界面

7 总结

