基于 JavaScript 的铅笔笔触模拟应用

2012.3.25

招蕴豪，陈跃群，吴垚，刘文辉，郭雨

摘要

本开题报告主要介绍此铅笔笔触模拟应用的各个组成部分。其中包括基本算法，基本笔触模拟实现以及最后将图片转化为扫描成品所涉及的算法以及实现。在结合前人所建立模型的基础上，我们对算法作出了适当的优化并且给出其 JavaScript 的实现，并结合实现给出了细致的讨论。

1. 动机

由于本身对艺术有所追求，对美有一种莫名的相吸之势，于是我们选了这个题目。希望通过数字化方法使一些没有经过真正艺术训练的人能够自己创造出属于自己的艺术作品。我们所做工作的最终目的是为了达到对照片进行素描艺术风格化的效果。对照片进行素描艺术风格化是指通过图像处理方法，将照片处理成具有素描画的效果。

在传统的图像处理方法中，已经有多种风格化的处理算法，而其中以直接做滤波的居多。考虑到直接使用滤波器进行处理逼真度不够高，并且滤波器的选择就非常困难，所以我们决定另辟蹊径，采取一种对真实素描创作过程的模拟方法来解决问题（后面详述）。

在语言方面，我们选取了与 web 融合度最高的 JavaScript。这样的选择有几方面的原因：首先用 web 来做界面的话可以方便演示，以及方便中间过程的测试；再者，使用一种比较高级的脚本语言能够使得实现更为灵活，不用拘泥于某些细微的问题，如内存分配问题，这样能够是我们能够真正专注于整个系统的设计；最后，用 web 来实现可以不用考虑平台的问题，只需要一个浏览器就可以使用我们的成品，并且移植也不用花太多的力气。

2. 原理概述

在进一步叙述之前有必要对基本的原理进行阐述。上面说道希望使用一种对真实素描创作过程的模拟方法来实现素描艺术风格化的效果，这实际上说的就是*我们希望能通过分析指定的照片，通过对其分析，能够使用笔触效果在一张白纸上面对这张照片的内容进行重现，这样就等同于看着实物来进行素描画的创作，逼真度足够高*。而要达到这个目的，首先得实现的基本功能正是对铅笔笔触的模拟。

在对铅笔笔触模拟这个课题上，前人已经做了非常多的工作，我们也是通过结合前人的工作来提供 JavaScript 的实现。在整个笔触模拟的系统中，有三个重要的组成部分：*铅笔*，*纸张*以及*铅笔与纸张的交互*。

铅笔模型实际上就是笔尖的模型，当笔尖划过纸张的时候，会在纸张上面留下石墨，而留下的石墨量的不同导致了铅笔笔触在纸上留下了灰暗交错的纹理。

纸张模型可以看成是一个沟壑纵横的地形，这种地形的基本单位是一些不规则形状的凸起，当笔划过这些凸起的时候，会在凸起上留下石墨，而凸起的高低不同会导致留下的石墨量的不同，从而影响到其灰度的不同。

在铅笔与纸交互的过程中还有很多细致的问题，会在下面详细讨论。

3. 主要原理及算法

下面将详细叙述整个应用所涉及到的算法以及原理。整个叙述过程分为两个部分，首先是对铅笔笔触模拟的各个部分的细致阐述，然后是对素描风格化算法的阐述。

3.1. 铅笔笔触模拟

如上所述，铅笔笔触模拟中有三个主要的模型，分别是*铅笔*，*纸张*以及*铅笔与纸张的交互*。整个模拟过程遵循下面的主要步骤：

1. 在铅笔停留的地方，通过笔尖所定义的多边形来计算出受影响的纸张凸起集合。
2. 初始化纸张凸起的相关参数。
3. 通过线性插值来得到笔尖各个点的压力系数，然后对于每个受影响的纸张凸起：
   - 计算纸张凸起的各个高度所能承受的最大的笔屑数量。
   - 计算纸张凸起的各个高度所接收到的笔屑数量。
   - 计算留在纸张凸起的笔屑数量。
   - 计算由于铅笔与纸交互而产生的损害。
4. 根据纸张凸起的笔屑数量来为之附上相应的灰度值。

3.1.1 笔尖多边形

笔尖多边形的形成方法可归纳为如下几个步骤。

1. 选择多边形结构体的属性。在初始化之前，我们首先可以让用户选取铅笔笔尖的形状。根据其选取形状的相应的特征，例如有：Typical，Broad，以及Chisel。我们就可以知道笔尖的初始形状是六边形，五边形，长方形还是正方形。接下来就可以选择数组传进多边形的结构体。由于铅笔笔尖的实时变化性，光有数组存放顶点还不够，我们还要有一个缩放系数a。因为随着铅笔笔屑的掉落，铅笔的笔尖也越来越钝，所以这个a随着时间的推移而慢慢变大，在之后的计算中用a乘以顶点坐标就可以使形状变大。
2. 计算笔尖多边形的准确形状。由于铅笔与纸之间并不是90度，所以多边形的形状会跟据笔尖与纸的角度变化。在我们的实现中，假定这个笔尖与纸的夹角一经设

定并不再变化。现在，由于铅笔触纸的角度没有变化，这样，我们就可以通过在初始化的时候通过角度计算出多边形的形状，而且，在其后面的计算一直保持形状不变。

3. 计算多边形边上所有点的坐标。经过之前步骤的设定，我们随时都可以很容易地得出当前多边形的每个顶点的坐标，由于两点确定一条直线，可以根据这些顶点得出多边形每一条边的方程。同时，由于数学意义上的线段由无数顶点构成，而在我们实现中，最小的单位是1像素，这样通过每条边上两个顶点的x坐标的限制，对x从小到大依次递增1，把x坐标代入多边形对应边的方程，求出y的坐标，这样就能很容易地把所有落在边上的点的坐标求出来了。
4. 求出落在多边形内部的所有点的坐标。因为我们在计算笔屑时只关心落在多边形内部的点。所以就有必要知道哪些点落在多边形的内部。我们在笔触的类中一直存放着当前笔触中心点的坐标，并且随着鼠标的移动而不断地更新坐标。此外，我们又有多边形边上所有点的坐标，那么就可以通过广度搜索，列举出所有落在多边形内部的点的坐标。（广度搜索从中心点开始，慢慢向周围扩散，而结束条件就是当搜索碰到了多边形边上的点。）

3.1.2. 纸张凸起

由于把纸张放大之后，它的凸起就有点像地形，同样是凹凸不平。这样的设想使我们有了一点灵感，既然像地形，那么我们就可以利用这点特性，把纸张想象成地形来生成。由于生成地形在以前已经有人研究过，经过我们查阅资料，发现其实现已经有了比较成熟的算法----分形算法。我们可以借鉴前人的经验，利用分形算法来实现初始化纸张凸起。

分形算法的主要思想是：

1. 先初始化纸张大小，将纸张的四个顶点赋高度为0；
2. 菱形阶段：利用构成正方形的四个点的高度，在这个正方形的中点，也就是两条对角线交汇的地方，生成一个随机值。中点的值就等于，四个边角点值求平均后再加上这个随机值。这样，当在网格中有多个正方形时，这样就会为你产生菱形；
3. 正方形阶段：利用构成菱形的四个点，在这个菱形的中点生成一个和上一步相同取值范围区间的随机值。同样这个中点值等于，四个边角点值求平均后再加上这个随机值。这样就又会给你产生正方形；
4. 减小随机数的幅度；
5. 不断递归第2、3、4步，直到相邻像素的高度都求出来为止。

当然，由于这个算法是针对生成地形的，毕竟还是跟纸张有区别。我们要对它进行实现，就要对其中进行一些系数的调整，改进它高度变化的幅度，只保留算法的中心思想。同时，还要设置成可以通过调节随机数范围来调节纸张的粗糙与细致。

3.2. 素描风格化模拟

有了铅笔笔触模拟的基础以后，就可以更为便捷地进行素描风格化的模拟了。之前提到，我们的素描风格化的实现是基于现实中素描过程的一个模拟。也就是说整个模拟过程大致分为四个步骤，分别是：

1. **轮廓勾勒。**在这一步里面，对照片的大体轮廓进行大致的识别，然后用比较深色的铅笔笔触对这些识别出来的轮廓进行勾勒。
2. **阴影分块。**这里的阴影并不是平常所说的由于遮挡光而形成的阴影，而是指大面积颜色相近的色块。通过将这些阴影块识别出来，然后赋予比较浅的铅笔笔触涂抹。
3. **阴影加强。**同理，这里的阴影所指跟步骤一是一样的。在这一个步骤里面，通过通道阈值的调整，进一步对色块进行细分，对细分出来的各个色块进行更为深色的铅笔笔触涂抹，使得整张素描更具有立体感。
4. **细节突出。**最后一步通过找到照片中的特征点，对这些特征点进行细致的，突出的铅笔笔触进行勾画。从而使得整张构图丰富，深浅有致，又不乏特点。

下面将对着四个步骤所涉及到的方法进行细致地阐述。

3.2.1. 轮廓勾勒

轮廓勾勒是整个素描风格化的第一步，所以轮廓勾勒的效果会大致地影响整个页面的布局。其实对于所有的输入图片，可以大致分为**主角明确**以及**主角不明确**两类。对于主角明确的图片来说，如人物图，动物图，静物图等，在进行边缘提取以后，会有一个比较大并且清晰的区域被勾勒出来，即主角；但是对于一些主角不明确的图片来说，如风景图，由于颜色一般比较杂乱，所以难以提取出一个主要的主角部分轮廓出来，所以对于下面的讨论也是需要对图片进行分类讨论。

这样的分类其实可以通过简单的直方图来判断。对于主角明确的图片，其直方图起落比较有至，而对于主角不明确的图片，由于颜色比较复杂，所以可以从其直方图中看出波动比较厉害。可以用一种极值统计法来确定图片属于那一种图。将图片的直方图进行归一化，然后对灰度值进行适当的平滑，对得到的函数进行极值进行统计，并设定一个阈值 **n**，如果极值数比 **n** 要大，那就将其划分为主角不明确的图，反之，就将其归为主角明确的图。

对于主角不明确的图来说，可能并没有边缘提取的需要，但现在还没有想到太好的办法，所以先对主角明确的图进行讨论。边缘提取的滤波器可以使用 **laplace** 或者 **sobel**，得到的结果可能还会有一些异常点，可以用腐蚀的方法将边缘处理干净。另外，由于滤波之后，边缘可能会有多种灰度值的像素值，所以可以用一个阈值处理，最终用 **1** 代表边缘像素，用 **0** 代表非边缘像素。

对于这些边缘像素，我们并不能立刻利用，因为我们需要将笔触运用在这些像素上面，既然如此，那么我们有必要将这些边缘分成一笔一笔，而不应该是杂乱无章的一堆点（由于提取边缘之后，我们只是知道边缘图像中，标记 **1** 的那些像素点是边缘像素，但我们并没有关于这

些点的一个确切的描述，那么我们又怎么能够将笔触应用与这个边缘呢？），所以我们必须对这些点进行分析，将其分为一些边的集合，即将这些边缘像素划分成一条条边的集合。

现在假设图像中只有一条边，并且这一条边上面的所有像素都是相邻接的，如图 3.2.1-1（为了统一术语，我们将这种提取边缘之后得到的图称为**边缘图**）：

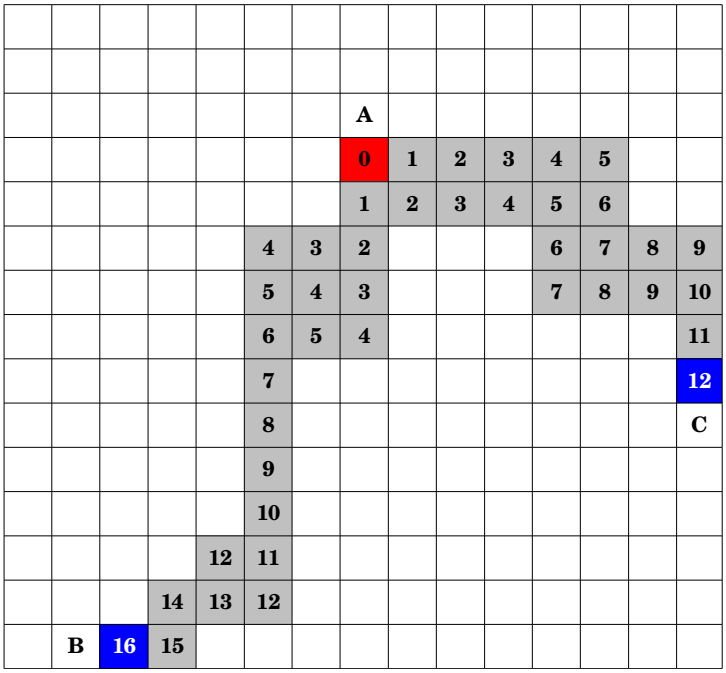

图 3.2.1-1

如果提取出来的边缘是像上面这样的，那么将会非常容易处理。我们可以对边缘图进行扫描，当遇到一个边缘像素，我们便停止扫描，将这个边缘像素设定为起始像素，并对着个边缘像素进行广度搜索，并在广度搜索的过程中记录下边缘点于起始点的距离，如上图标注的数字所示。根据得到的距离，我们可以通过对距离进行排序而得到一条有序的边，对于上图的情况，我们可以认为 AB 是一条边，AC 是一条边，也可以通过对距离值进行适当的处理，得到 BAC 一条边，两种均可。

下面考虑一种稍微复杂的情况，如果图上的边并不值一条，如图 3.2.1-2，图中有三条边，但每一条边上的边缘像素都是邻接的，那么我们可以采取同样的方法，但是稍微有一点不同的是，被划分为某一条边的像素必须做标记。算法开始的时候依然是对图上的像素进行逐行扫描，当遇到边缘像素的时候，将其设置为起始点，开始广度搜索，每逢访问到一个边缘点，就将其标记为以访问的，这样直至扫描完一条边为止。扫描完一条边以后，从这一条边的起始点重新开始逐行扫描，寻找另一个没有被标记访问的边缘像素。比如图 3.2.1-2 中，当对边 AB 完成扫描以后，从 A 点开始逐行的像素扫描，扫描到 C'的时候，由于 C'属于边 AB，并且在扫描 AB 边的时候已经被标记为访问了，所以并不会将其设置为起始点进行另一条边的扫描，而是到 C 点的时候才会将其设为起始点，由此开始了对边 DCE 的扫描。剩下的边也如此类推。直至扫描到图片的最后一个像素为止。这种方法就解决了一个边缘图中有多条边的情况了。

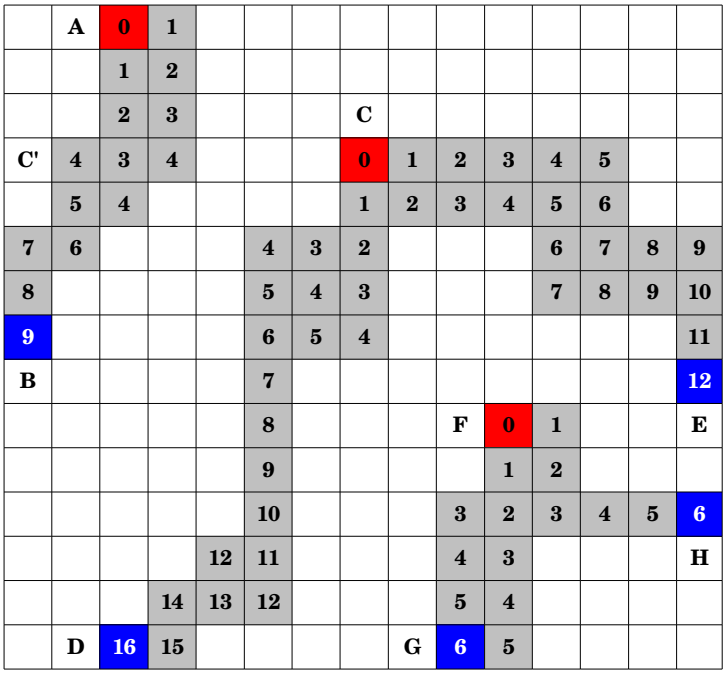

图 3.2.1-2

最后的情况是最为复杂的。由于实际边缘提取中边缘并不可能是所有边缘像素都是邻接的，所以实际中并不能使用上面的算法。实际中的边缘图有可能是图 3.2.1-3 那样的：

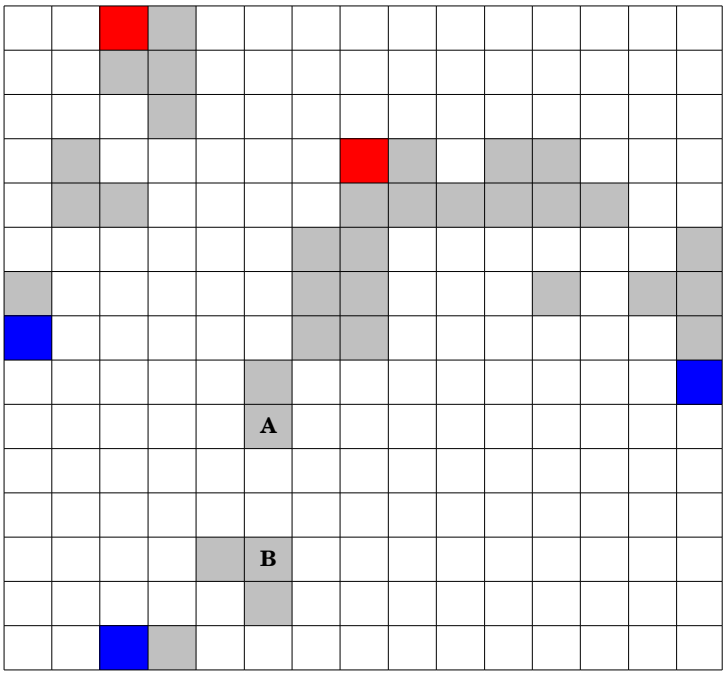

图 3.2.1-3

这样的边缘图才是实际应用中会出现的，图 3.2.1-3 从肉眼可以看出是有两条主要的边缘，但是计算机怎么能够识别出来呢？最关键的步骤在于广度搜索，我们可以通过放宽广度搜索的

扩展条件来解决这个问题。比如可以设定一个步长 **n**，在扩展的时候并不需要只扩展接邻的点，而是可以扩展邻近距离为步长以内的点。假设步长为 3，那么在上图中的点 **A** 也是可以扩散到点 **B** 的，而只要到达了点 **B**，就可以一直扩散下去直到边缘的尽头。

由此便解决了边缘提取后对边缘的分组操作。这里仍需要注意一点，由于不排除有一些异常值，在对边缘分组以后，可以对这些边缘组进行统计，并设定一个阈值 **n**，将边缘组里面边缘像素数目少于 **n** 的那些分组去掉，这样就能够保证我们得到的确实是一些大轮廓的边缘，而不是一些细微的异常值或者突出的颜色小块。

有了边缘以后，我们便可以将笔触应用到这些边缘了。由于考虑到素描画的边缘并不是一条细长光滑的线，而是一些粗糙的，有微小角度差别的线，所以我们打算用一下的方法来解决。看下面的图 3.2.1-4：

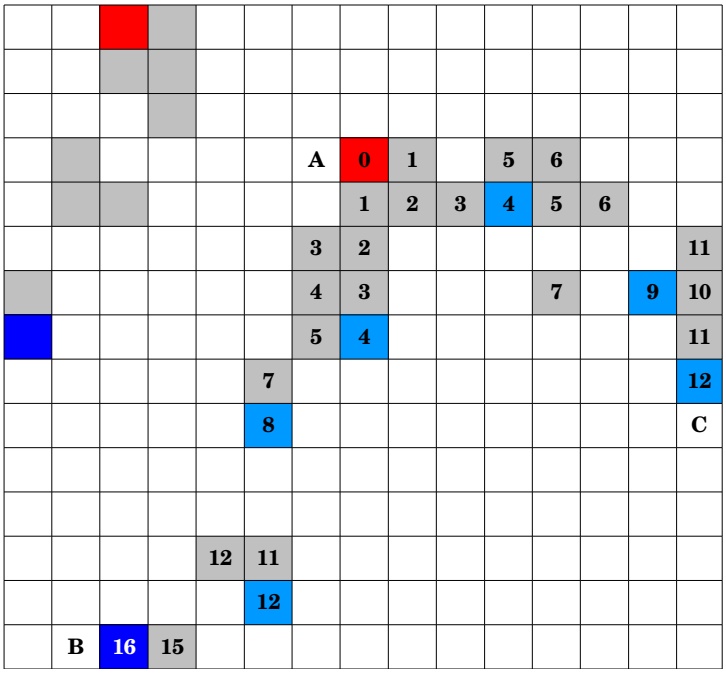

图 3.2.1-4

对于一条边来说，我们并不会让铅笔笔触顺着这条边画下去，因为这样得到的结果并不会像素描画，我们将使用下面的算法来解决。用上面的例子来说，我们将用笔触绘制边缘 **BAC**。为了达到上面说的素描效果，我们用一些线段来模拟这一条曲线，使得最后得到的效果是不太平滑的并且粗糙的轮廓线。先设定一个步长，比如 **4**，那么从起点 **B** 开始，标记下一个距离当前边缘像素 **4** 个单位的像素，如果满足要求的像素有多个，那么将随机选取一个。由此我们可以得到图 **3.2.1-4** 所显示的浅蓝色的点，姑且现将其称为**跳点**（由于它们不是连续的，跳过了步长），然后我们再使用铅笔笔触将这些相邻的跳点连起来，得到轮廓线。对于相邻的两个跳点，我们并不一定只用笔触描画一遍，我们可以用一些有细微角度差别的线将其连起来，这样最后得到的就像是素描开始起稿的效果了。

3.2.2. 阴影分块

在图像处理中有几个概念比较重要，这里先稍微提及一下，分别是：*邻接*，*通道*以及*区域*。

邻接是相邻两个像素之间的关系。对于图中的每个像素，其临近有八个像素(忽略边界)。四邻接定义其上下左右的像素与其相邻接，八邻接定义其邻近的八个像素都是与其邻接的，而混合邻接则是为了取消因为邻接而产生的自环而定义，在四邻接合法的情况下，就不考虑八邻接的情况了。

通道是图中两个任意像素之间的关系。如果这两个像素之间存在一条通路，而这条通路上每个相邻的像素都满足上面定义的邻接关系中的一种，那么我们就称这两个像素之间存在着满足某种邻接的通路。

至于区域，指的是在这个区域里面的任意两个像素之间都存在着通路。

通过这样简单的讨论，我们就可以使用这几个简单的概念进行更为复杂的描述了。首先阴影分块作为整个风格化过程的第二步，可见其重要性。阴影分块的效果直接决定了最终结果的好坏。整个连通性的界定是通过设定连通集来实现的。为了说明这个概念我们先举个简单的例子。比如要分出一块红色的区域，众所周知，红色的RGB值为(255, 0, 0)，为了定义一个容差率比较高的区域，我们暂且设定这个连通集为 $\{(r,g,b)|200\leqslant r\leqslant 255,0\leqslant g\leqslant 20,0\leqslant g\leqslant 20,\}$ ，其中红色具有最大的容差，即50，而其它两个分量则具有较小的容差，这样既能保证色块区域有一定的大小，又能保证色块的颜色是偏向于红色的。此即为连通集的概念。事实上，设定连通集并不是那么容易的，如果容差设置小了，那么连通区域将不会很大，如果容差设置大了，那么得出来的连通区域又会是五颜六色。而第一步的作用正是将整个照片的构图分成一个一个的色块区域，所以**每个色块区域都必须选取好相应的连通集，从而使得最后这些色块划分能够最大限度地覆盖整个画面，并且在每个色块内部，像素之间的颜色差别也是尽可能的小的。**

在选取色块以后，我们就能自然地将图片所有的像素进行了分组。对于每一个组里面的像素，我们都必须为之附上一层浅浅的铅笔笔触涂抹效果。这个附上铅笔涂抹效果的过程并不是那么的容易。下面将花一些篇幅进行讨论。

必须明确的是，我们在这一步中要做到下面的效果：对于一个色块，要为其附上一些有规则角度的并排铅笔笔触线，如下图3.2.2-1：

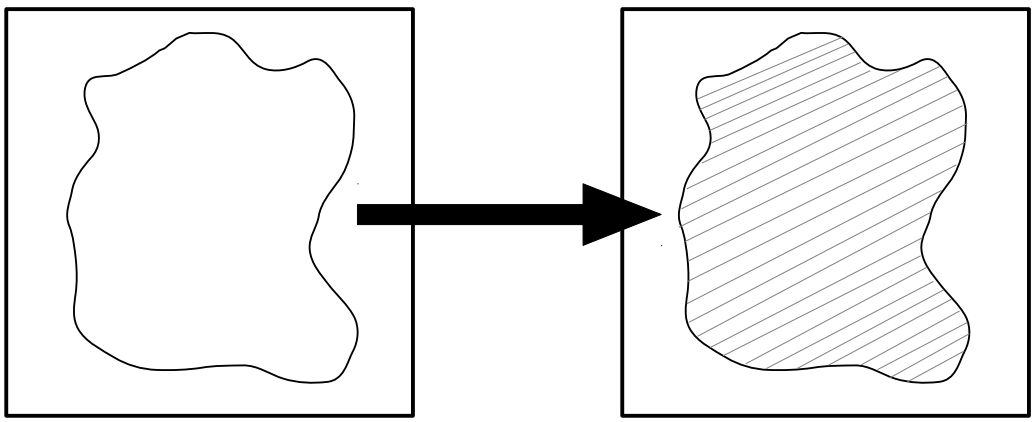

图3.2.2-1

其中图 3.2.2-1 的左边是我们得到的色块，右边是附上具有一定角度的铅笔笔触线的效果。考虑到处理不规则形状并不是那么方便，所以可以用一个非常简单的方法来解决这个问题。我们可以事先建立一个样本库，这个样本库并不需要太大，里面存放着一些具有不用角度的铅笔笔触并排线的模板，如下图 3.2.2-2：

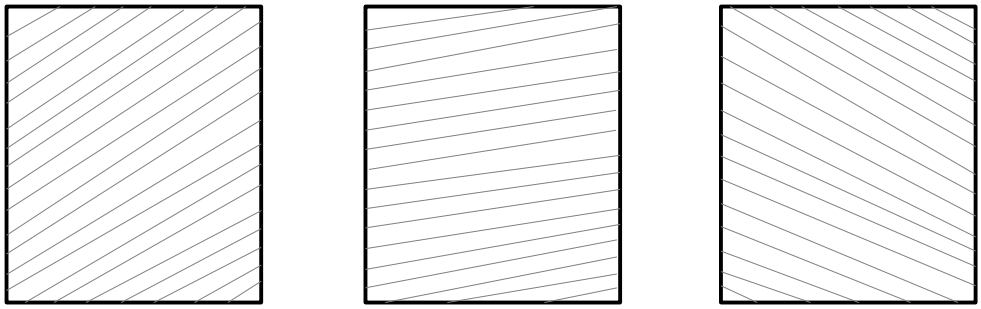

图 3.2.2-2

就这样建立一个简单的笔触排线模板库。我们可以用以下方式对这些模板进行应用。对于一个不规则形状的色块，如下图 3.2.2-3：

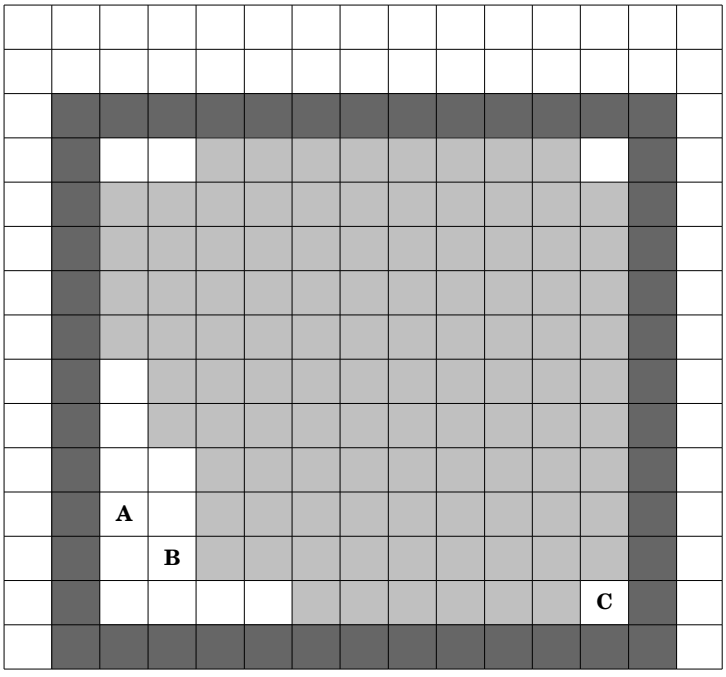

图 3.2.2-3

我们首先找出这个色块的凸矩形包，这个矩形包非常容易找，可以对色块内部的像素位置进行一个排序，分别从 **x** 方向和 **y** 方向找到最大值，用这些最值做水平线以及垂直线即可得到矩形凸包。得到矩形凸包以后我们即可进行纹理映射了。我们从笔触排线样本库里面随机抽取出一个模板，用同样的矩形凸包划出模板的一片区域，然后将模板的该区域像素值映射到色块内部的像素中去。如果映射到色块区域矩形凸包中的空白地区，如图 3.2.2-3 中的 **A**，**B**，**C** 点，那么该点就忽略。由此便可以简单地为色块附上笔触排线纹理。

3.2.3. 阴影增强

阴影增强实际上是第二个步骤的细化操作，那为什么不一开始就使用更为细致的阴影分块算法阈值呢？原因有几个，下面将会结合第二步来阐述这些问题以及给出基于直方图的阈值选择方法。

原因之一在于，这是一个基于现实素描步骤的一个模拟过程，而在一般的素描技法中，都是先对素描对象进行造型，即轮廓勾勒，然后再通过大片的色块进行一个简单的阴影渲染，之后再局部细化。这样画作看起来才会有真实感，因为这跟工笔国画有点类似，是通过层层渲染达到最终效果，而不是每个局部都一次达到最终效果，这样看起来虽然会比较省事，但是实际上却使得整个画作看起来并不是那么的自然，并且会有分崩离析的感觉，因为一块一块之间联系并没有整体打阴影之后再继续细化那么强。

再有，上述说到真正的素描是层层渲染的，而这个层层渲染体现在层层角度不同的排线相互交叠，并且在灰度的层面上，素描是通过排线交叠的层数来决定灰度值的大小，即排线层数越多，那么灰度值就越大。如果只渲染一次，并且直接根据排线的灰度值而不是层数来体现出画中的灰度值的话，那么根本就不会有交叠的排线，而整个画作就只有一层排线。

在第二步划分色块的时候，其实可以采取 3.2.1 提到的极值统计法来处理，假设直方图可以以最小极值划分为如下区间 $[x_1,x_2],[x_2,x_3]...[x_{n-1},x_n]$ ，而最大极值 $\{x_{1,2},x_{2,3},...,x_{n-1,n}\}$ 分别位于最小极值划分出的区间中，即有直方图各“峰”如下：

$$[x_1,x_{1,2},x_2],[x_2,x_{2,3},x_3]...[x_{n-1},x_{n-1,n},x_n]$$

这样，我们可以对各个峰所跨越的灰度值数目进行统计，即有：

$$k_1=x_2-x_1,k_2=x_3-x_2,...,k_{n-1}=x_n-x_{n-1}$$

此时，可以设定一个阈值，确定我们所需要的色块，最终得到的 $\mathbf{k}$ 值如下：

$$\{k_i,k_{i+1},...,k_j|i,j\in[1,n]\}$$

确定了这些值以后，就可以开始划分色块了，方法比较简单，跟划分边界的方法是类似的。可以对图像进行逐行扫描，当遇到一个灰度值位于上述选取的色块的灰度值范围以内的像素时，即把此像素设置为起始像素，然后对其进行广度搜索，将满足该色块灰度值的邻近点都划分到一个集合里面去。当一次广度搜索结束以后，我们便可以从上一次的起始点重新开始，继续扫描下去，如此便可以将色块都划分出来了。

在后续的细化过程中，所遵循的依然是同一原则。因为在第二步中已经将图片的色块大致划分出来了，所以在细化的时候，可以对这些已经划分出来的色块进行同样的操作，将阈值降低，将大色块分为小色块，然后为这些小色块附上排线模板库中的纹理。

3.2.4. 细节突出

最后一个步骤是细节突出，在素描整体轮廓定形，阴影分布大致做好以后，就只剩下最后的细节处理。细节指的是一些会用比较尖细笔头以及用力比较重刻画的地方，对于人物画来说，这些地方可以是眼角，耳垂，衣领拐角处等等；而对于风景画来说，可以是树叶交错之处，树干下面的杂草等等。这些地方其实就是图中的特征，图中的拐点，寻找这些细节的地方有很多算法，这里就不一一详述了。

问题是找到这些特征点以后，我们得到的依然是一些无序的，杂乱的点，我们需要对这些点进行处理，并附上相应的铅笔笔触。从滤波后的图像上看，这些地方具有“小聚局”的特点，即局部密集，但却没有方向性。这里的没方向性是指，我们并不能根据一小团杂乱的点来为其附上铅笔笔触。因为路径是铅笔笔触最为重要的输入，如果只有一堆的点，我们是不能应用铅笔笔触的。我们必须将这些一小团一小团的点转化为路径，这样才能方便铅笔笔触的勾画。

4. 交互界面

由于整个系统使用 JavaScript 来实现，在界面设计方面具有天然的优势，不过可能需要 firefox 或者是 chromium 才能够访问，因为至今，IE 内核的各浏览器并不支持 canvas 标签。

整个界面包括两个主要部分，首先是一个铅笔笔触的演示部分，另外一个是素描画风格化的演示部分。

4.1. 铅笔笔触演示部分

在铅笔笔触的演示部分中，主要使用的是 canvas 标签，通过这个标签可以实时读取用户鼠标所划过的轨迹，并将此轨迹传给铅笔笔触，让铅笔笔触顺着这样的轨迹进行描画，从而达到用鼠标描画就像是用铅笔在屏幕上描画的效果。

从上面的讨论可知，在笔触实现的过程中涉及到了诸多的参数，将这众多的参数开放给用户尚且不可能，因为面对这么多的参数，用户也不知道如何才能调整出比较满意的，比较真实的效果，所以我们会适当开放出一些有决定作用，并且用户能随意调整的参数，比如铅笔的分级，铅笔的粗细，铅笔的触纸角度，铅笔的颜色等等。有一个比较重要的参数是铅笔的压力，由于我们并不是基于触控版来实现，所以根本就不会有这样的功能，鼠标划过屏幕并不涉及到什么压力的读取，而仅仅是轨迹所起的作用，所以这一点并不能让用户调整。

4.2. 素描风格化演示部分

在素描风格化的演示部分，允许用户上传一张照片，通过使用我们的算法对这张照片进行处理，为用户呈现出处理过程中各个步骤的中间结果。

在上面的讨论中，描述了在进行风格化的过程中，有诸多不缺定的参数，比如各处提到的阈值，这种阈值的选定其实很难根据计算机智能选定。况且，不同的图像具有很大的差异性，并不能使用一个统一的算法来处理不同的图片，更不能使用同一个标准来评判不同图片最终的处理效果，所以在中途加入用户的主观因素，效果可能会更好。因为用户可以在一边调整的过程中看到调整的效果，这样我们的阈值才能取到一个合理的值，而通过使用这样比较合理的阈值来计算，我们得到的结果将会更好。

在这一个演示部分中，分步演示指的是分别将上述提到的四个步骤的结果显示出来。即在用户选定最初阈值，步长等参数以后，我们便将勾画出来的轮廓显示出来，如果用户觉得不满意，可以继续进行调整。下面的步骤也是一样。