LowPoly 图像效果

—Python 大作业报告

吴克文 倪星宇 张云帆

2017 年 5 月 17 日

目录

1 准备工作

安装 Python3，并安装库 Scipy，Numpy，Pillow，Matplotlib。

2 运行

```
$ python3 lowpoly.py path/to/input.png path/of/output.png
↪ num_of_scattered_points
```

注:

input.png: 输入图片（请注意图片大小，过大请 resize 以保证速度）

output.png: 输出图片（与输入图片大小一致）

num: 图片撒点数，默认值为 2000

3 核心算法

本作业参考了 [1][2] 两篇论文，综合效果和实现难度进行了调整，根据两位一作在知乎的回答[1]，删去了繁琐的细节调整和性能优化部分。

整体算法流程为：先对图片进行边缘检测，在边缘处取点，并在全图随机撒点，之后对点进行三角剖分，最后对每个三角形进行颜色计算。

以下为算法的详细介绍。

3.1 边缘检测算法

此部分参考[2]。采用 *Sobel* 边缘检测算法。

定义 3.1 *Sobel* 算子：离散性差分算子，用来运算图像亮度函数的灰度近似值。在图像的任何一点使用此算子，将会产生对应的灰度矢量或是其法矢量。

该算子包含两组 3×3 的矩阵，分别为横向及纵向，将其与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。

如果以 A 代表原始图像，G_x 及 G_y 分别代表经横向及纵向边缘检测的图像灰度值，其公式如下：

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \times A \tag{1}$$

$$G_y = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \times A \tag{2}$$

$$\tag{3}$$

梯度 G 的计算方法为：

$$G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

通常为了提高效率，使用不开方的近似值：

$$|G| = |G_x| + |G_y| \tag{5}$$

如果梯度 G 大于某一阀值则认为该点 (x,y) 为边缘点。

[1] https://www.zhihu.com/question/29856775

[2] http://www.cnblogs.com/lancidie/archive/2011/07/17/2108885.html

然后可用以下公式计算梯度方向：

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{6}$$

Algorithm 1: *Sobel* Edge Feature Detection

Input: Source pic, Threshold

Output: List of vertices on edge

```
Initialize the vertex list;
foreach point in the src pic do
    Calculate its gradient with Sobel operator;
    if its gradient greater than threshold then
        Add it to the vertex list;
    end
end
Return the vertex list;
```

3.2 三角剖分

此部分参考[3][4]。

定义 3.2 三角剖分：假设 $\mathcal{V}$ 是二维实数域上的有限点集，边 e 是由点集中的点作为端点构成的封闭线段，$\mathcal{E}$ 为 e 的集合。那么该点集 $\mathcal{V}$ 的一个三角剖分 $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ 是一个平面图 $\mathcal{G}$，该平面图满足条件：

1. 除了端点，平面图中的边不包含点集中的任何点。

2. 没有相交边。

3. 平面图中所有的面都是三角面，且所有三角面的合集是散点集 $\mathcal{V}$ 的凸包。

在实际中运用的最多的三角剖分是 *Delaunay* 三角剖分，它是一种特殊的三角剖分。

定义 3.3 *Delaunay* 边：假设 $\mathcal{E}$ 中的一条边 $e = (a, b)$，e 若满足下列条件，则称之为 *Delaunay* 边：存在一个圆经过 a,b 两点，圆内不含点集 $\mathcal{V}$ 中任何其他的点。这一特性又称空圆特性。

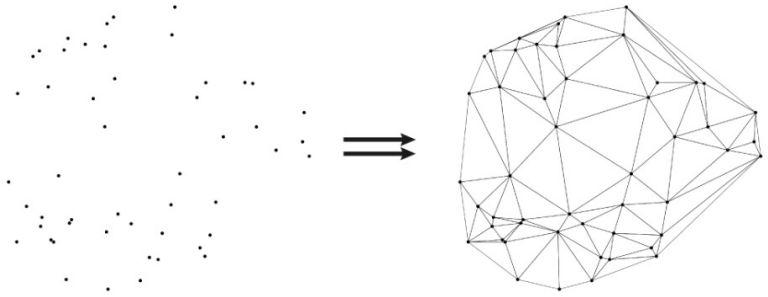
定义 3.4 *Delaunay* 三角剖分：如果点集 $\mathcal{V}$ 的一个三角剖分 $\mathcal{T}$ 只包含 *Delaunay* 边，那么该三角剖分称为 *Delaunay* 三角剖分。

[3] http://paulbourke.net/papers/triangulate/

[4] http://www.cnblogs.com/zhiyishou/p/4430017.html

定义 3.5 假设 $\mathcal{T}$ 为 $\mathcal{V}$ 的任一三角剖分，则 $\mathcal{T}$ 是 $\mathcal{V}$ 的一个 *Delaunay* 三角剖分，当前仅当 $\mathcal{T}$ 中的每个三角形的外接圆的内部不包含 $\mathcal{V}$ 中任何的点。

定义 3.6 超级三角形：该三角形包含了点集中所有的点。



Algorithm 2: *Delaunay* Triangulation

Input: Vertex list

Output: Triangle list

```
Initialize the triangle list;
Determine supertriangle;
Add supertriangle vertices to the end of the vertex list;
Add supertriangle to the triangle list;
foreach sample point in vertex list do
    Initialize edge buffer;
    foreach triangle currently in the triangle list do
        calculate the triangle circumcircle center and radius;
        if the point lies in the triangle circumcircle then
            Add triangle edges to edge buffer;
            remove triangle from the triangle list;
        end
        Delete all doubly specified edges from edge buffer, this leaves the
          edges of the enclosing polygon only;
        Add to the triangle list all triangles formed between the point and
          the edges of the enclosing polygon;
    end
    Remove any triangles from triangle list that use supertriangle vertices;
    Remove the supertriangle vertices from the vertex list;
end
```

3.3 颜色计算

根据论文 [1]，将每个三角形内部的所有点颜色按 L 值排序，选取其中 $40\% \sim 60\%$ 的点计算颜色均值。

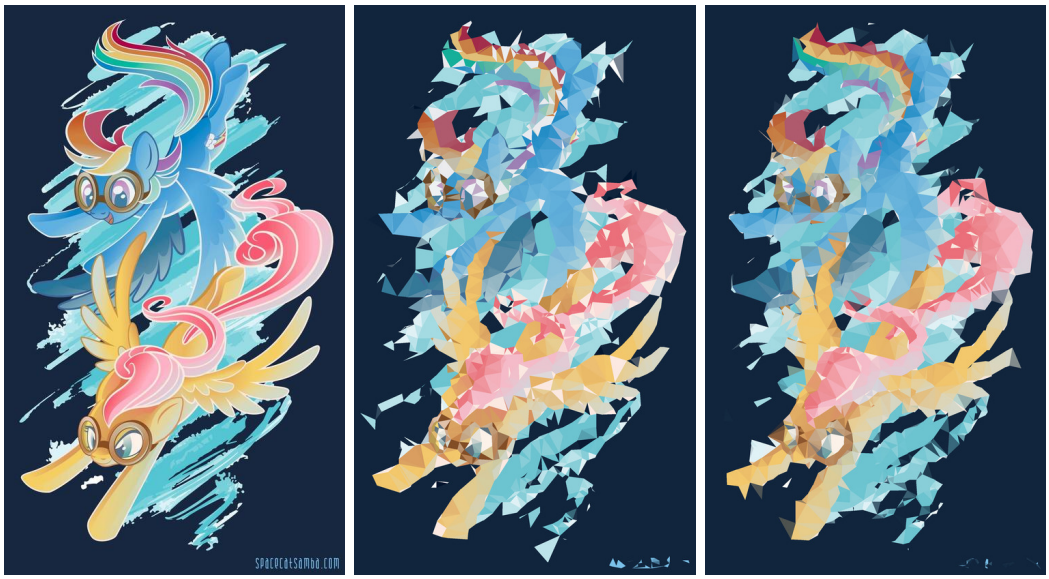
定义 3.7 Lab 模式：Lab 模式是由国际照明委员会（CIE）于 1976 年公布的一种色彩模式，是 CIE 组织确定的一个理论上包括了人眼可见的所有色彩的色彩模式。

定义 3.8 L 值：Lab 模式下的第一个通道，即，明度通道。

此外，我们还测试了 [2] 中的颜色计算方式：直接选取三角形重心处的颜色值。虽然此法对程序性能有巨大提升，但很大程度上牺牲了画面质量，故注释去。具体实现细节见代码。

4 效果展示

颜色计算的方法对比：



左图：原图，中图：重心取色，右图：排序平均取色，点数均控制为 2000。可以看出，平均取色的方法减少了莫名奇妙的白三角，使得颜色变化更为平缓。

点数对比：

左图：原图，中图：点数为 2000，右图：点数为 6000。可以看出，点数增加对于细节较多的图片质量有一定的提升。但过多的点数不仅会影响程序性能，还会导致 LowPoly 风格的不明显，故在源码中开放了 num_of_scattered_points 选项，供使用者自行尝试效果。

风景图和论文 [2] 中的雨伞图：

图 3: 点数为 2000

图 4: 点数为 2000

参考文献

[1] Meng Gai and Guoping Wang. Artistic low poly rendering for images. *The visual computer*, 32(4):491–500, 2016.

[2] Wenli Zhang, Shuangjiu Xiao, and Xin Shi. Low-poly style image and video processing. In *Systems, Signals and Image Processing (IWSSIP), 2015 International Conference on*, pages 97–100. IEEE, 2015.