基于GraphCut的纹理合成

2017013632 石大川

环境配置

```
conda install --yes --file requirements.txt
```

使用方法

```
chmod +x task.sh
./task.sh
```

实验内容

实现了所有**基本功能**

- 不考虑old cut的块切割算法
- 三种块偏移生成算法
    - Random placement
    - Entire patch matching
    - Sub-patch matching

实验细节

- 块切割算法
    1. 以重叠区域的像素为节点生成图并计算节点之间的权重：

    $$M(s,t,A,B) = ||A(s) - B(s)|| + ||A(t) - B(t)||$$

    其中A, B为两个patch，s, t为相邻的两个像素，A(x)表示A在像素x处的值。
    2. 添加两个terminal节点，对于逻辑上一定属于某terminal节点的非terminal节点，设置其到terminal节点的权重为INF
    3. 使用maxflow求解最大流/最小割
    4. 根据切割结果拷贝像素
- 偏移算法1 Random placement
    1. 随机生成一个偏移量，在具体实现时我保证了偏移后的结果和上一步的拷贝后的结果交集不为空
    2. 输入偏移量到块切割算法中

    测试：

    ```
    python TexturesSynthesis.py -i ./data/green.gif -m random
    python TexturesSynthesis.py -i ./data/strawberries2.gif -m random
    python TexturesSynthesis.py -i ./data/akeyboard_small.gif -m random
    ```
- 偏移算法2 Entire patch matching
    1. 按照概率函数选择偏移量：

    $$P(t) \propto e^{-C(t)/k\sigma^2}$$

其中σ为输入纹理的标准差，k为控制匹配准确度的参数，我设置为了0.001。

代价函数:

$$C(t) = \frac{1}{A_t}\sum_{p\in A_t}|I(p) - O(p+t)|^2$$

其中t为偏移量，I为输入纹理，O为已有画面，A_t表示偏移为t时，新块与已有画面的重叠区域，距离我使用的是L2距离。

2. 输入偏移量到块切割算法中

测试:

```
python TexturesSynthesis.py -i ./data/green.gif -m entire
python TexturesSynthesis.py -i ./data/strawberries2.gif -m entire
python TexturesSynthesis.py -i ./data/akeyboard_small.gif -m entire
```

- 偏移算法3 Sub-patch matching

1. 在已有画面中选择一个子块，对于测试样例green.gif和akeyboard_small.gif我选择长和宽为输入纹理$\frac{2}{3}$大小的窗口，对于测试样例strawberries2我选择长和宽为输入纹理$\frac{3}{4}$大小的窗口。
2. 按照概率函数选择输入纹理中的最佳匹配:

$$P(t) \propto e^{-C(t)/k\sigma^2}$$

其中σ为输入纹理的标准差，k为控制匹配准确度的参数，我设置为了1。

代价函数:

$$C(t) = \sum_{p\in S_o}|I(p-t) - O(p)|^2$$

其中S_o为子块，t为偏移量，距离我使用的是L2距离。

3. 输入偏移量和sub-patch到块切割算法中

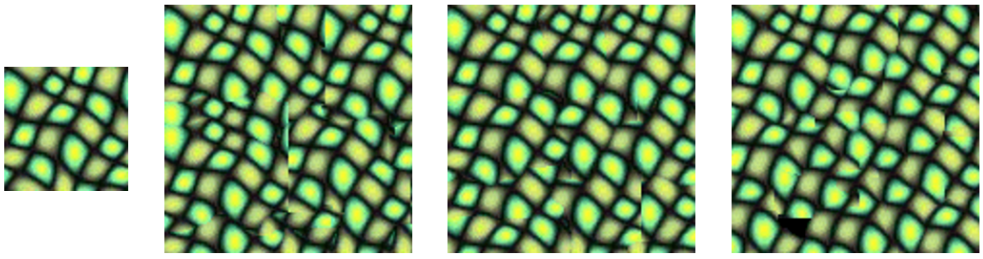
测试:

```
python TexturesSynthesis.py -i ./data/green.gif -m sub
python TexturesSynthesis.py -i ./data/strawberries2.gif -m sub
python TexturesSynthesis.py -i ./data/akeyboard_small.gif -m sub
```

- 实验效果

所有测试样例的长和宽均被扩充为原图大小的2倍

- green.gif



从左至右依次为原图，偏移算法1，偏移算法2和偏移算法3的实验结果，下同。

- strawberries2.gif

- akeyboard_small.gif

因为时间关系测例3没完全跑完，因此使用中间合成结果作为替代。

实验总结

- 从实验效果一节中可见，Entire patch matching和Sub-patch matching的合成效果总体上更加真实，而Random placement合成的纹理则artifacts更加明显。
- 在合成时间上，Random placement << Entire patch matching < Sub-patch matching，虽然后两种算法合成效果更好，但是时间成本显著增加。
- 综合考虑时间成本和合成效果，Entire patch matching是相对更好的合成方法。