# 媒体计算课程大作业报告

刘 晨

计 15

2011010539

## 0.选题

### 0.1 基于像素的图像补全算法

基于像素进行块之间的匹配关系进行像素补全，利用kd-tree进行加速。

### 0.2 基于图分割的显著区域检测算法

基于图论中的分割、最小生成树进行图像的显著性区域检测，并将这种算法的效果和基于颜色直方图（包括将256色进行压缩离散化处理）的算法进行比较。

## 1.实现

### 1.1 环境

操作系统 Ubuntu 14.10

开发语言 C++ , openGL， g++版本 4.9.1

### 1.2 通用模块

**base.h/base.cpp**

定义了一些基本用途的数据类型，几乎所有的后续文件都会引用该文件。定义了bmp文件的文件头和图片信息前面的信息部分，还定义两个基本类RGB、PAIR，前者用于表示颜色，后者用于表示二位图上的像素点。在基于图分割的显著区域检测算法中，我们还定义了LINK用于表示图论中的边（点直接用PAIR表示）。

**libbmp.h/libbmp.cpp**

bmp图片基本类（所有可执行程序的输入输出文件都是BMP文件，linux可以使用convert命令进行图图片格式转换），为每个project中处理图像的类的基类。该类包括的基本成员和函数如下所示。

```
class libbmp{
public:
    libbmp(char* filename);                                  //指定位图文件名
    libbmp(GLubyte* pixel,GLint width,GLint height);         //指定长宽和像素
    void pos2pixel();                                        //将像素信息由二维转化为一维
    void pixel2pos();                                        //将像素信息由一维转化为二维
    bool load();                                             //加载图片上的信息
    bool display();                                          //显示图片
    bool output(char* filename);                             //将图片信息保存
    GLint width;                                             //宽度
    GLint height;                                            //高度
    GLint pixellength;                                       //像素点个数
    GLubyte *pixel;                                          //一维像素点
```

```
    RGB **pos;                                          //二维像素点
protected:
    char* filename;                                     //文件名
};
```

### 1.3图像补全算法

raw文件夹下为最基本的没有任何加速方法的图像补全算法，kdtree文件夹下的算法使用了kd-tree加速。两者的使用方式都是`./demo <inputfile> <outputfile> <search_width>`，其中`search_width`的值为匹配patch的'半径大小'，例如当`search_width=3`的时候，patch的尺寸为7×7，`search_width=5`时，patch的尺寸为11×11。

在选填充点顺序方面，我优先选择周围已知点多的点。在使用kd-tree的时候，我们按照可能存在的最大维度，也就是patch的大小建立kd-tree，但是在实际查询的时候，由于我们不完全知道带填充点附近所有点的情况（有的附近多的点也是未知的），所以我们需要对kd-tree的空间进行投影。实际操作的时候，我们先找到所有在已知维度上范围包括查询点的叶子节点，然后逐个对它们进行回溯，寻找在已知维度上与查询点距离最小的点。代码层面上我们使用mask向量对维度的有效性进行标注（类似TCP/IP协议中的子网掩码）。

设图像的长宽为n，待填充点的个数为m，无加速的方法复杂度为`O(mn^2)`，使用kd-tree进行加速的方法复杂度为`O(mnlog(n))`。

该算法的核心实现类picture2类实现如下

```
class picture2: public libbmp{
public:
    picture2(char* filename,int search_width); //构造函数，指定patch大小，下同
    picture2(GLubyte* pixel, GLint width, GLint height,int search_width);
    GLubyte** mask;                                 //二维向量，该点是否需要填充
    void init_mask();                               //初始化mask，认为黑色点待填充
    void init_data();                               //建立kd-tree
    bool paint(PAIR position);                      //给对应的点染色
    PAIR choose_pixel(PAIR position);               //选取下一个需要染色的点
private:
    int find_dependents(PAIR position,int r);   //给定一个点和半径，确定附近已知点数目
    bool haspoint(PAIR point);                      //该点是否存在
    int search_width;                               //patch半径（边长的一半）
    int **data;                                     //kd-tree中节点共享的数据
    int all;                                        //需要染色点的数目
    int completed;                                  //已经完成的染色点数目
    kdtree *tree;                                   //搜索过程中维护的kd-tree
};
```

### 1.4 显著性区域检测算法

raw文件夹下为最基本的算法。使用方式为`./demo <inputfile> <outputfile> <rate>`，其中rate为256颜色空间的压缩比率，例如`rate=4`表示将相邻的4个颜色值划为一个离散值，整个颜色空间大小为`64×64×64`。我们首先求出这个颜色离散化的图的颜色分布直方图，然后以与其他所有点的距离为量度为刻画标准刻画像素显著性，显著性越大，在最后输出的图中亮度越大。

map文件夹下是使用是基于块的显著性检测方法。具体方法初始情况下将每个点视为一个区域，

是把每个像素与周围的八个像素相连，以色差为边的权值并按照非降序进行排序。按照顺序遍历每一条边，如果该边连接两个不同的区域并且这两个区域的Mint值（综合考虑最小生成树中的最大边权值与区域大小的值）大于当前边权值，则将该边连接的两个区域合并。遍历完所有边之后，整个图变成了若干区域，每个区域中的像素点的颜色比较类似，我们采样一部分点，用它们的平均数作为该区域的颜色，然后再以区域为单位运用类似raw中的方法（考虑每个区域的大小和区域之间的距离），检测每个区域的显著性。

设图像边长为n，拥有颜色种类为m，区域个数为k，raw方法的复杂度为`O(m*n^2)`，map方法的复杂度为`O(n^2+k^2)`

map文件夹下算法核心实现类picture类如下。

```
class picture: public libbmp{
public:
    picture(char*filename,int rate);    //构造函数，rate为k的值，下同
    picture(GLubyte* pixel, GLint width, GLint height,int rate);
    void loadlinks();                   //加载像素间边的信息并排序
    void merge();                       //合并各个区域
    void paint();                       //绘图
    void calc();                        //计算各区域的显著性
    void save();                        //将当前的像素值缓存
    void resume();                      //加载之前的像素缓存
private:
    RGB **saved;                        //像素缓存
    int rate;                           //k的值
    int num_link;                       //像素间边的个数
    int* zonemap;                       //从点到区域编号的映射
    int* feature;                       //每个区域的显著性值
    int pixel2index(PAIR p);            //像素转化为编号
    PAIR index2pixel(int index);        //编号转化为像素
    bool exist(int i,int j);            //检测某一个点是否存在
    void sortlinks(int b,int e);        //给像素间边排序
    LINK* links;                        //像素间边列表
    vector<vector<PAIR> > zones;        //区域及其含有的点列表
};
```

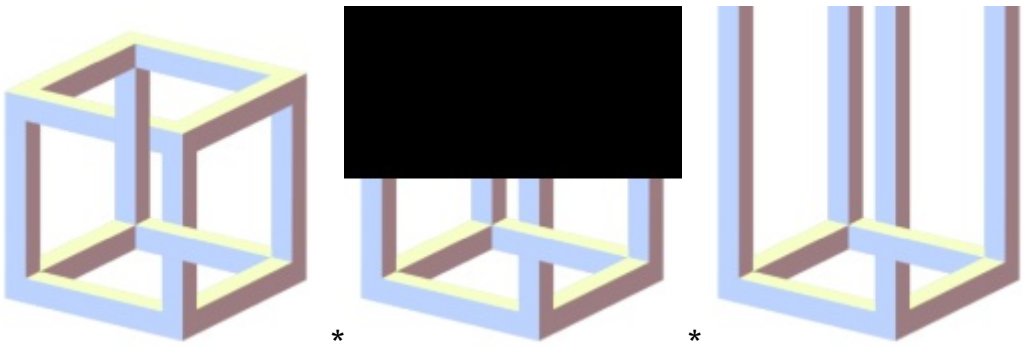
## 2.运行结果

### 2.1 图像补全

依次为原图，抠去一半的图，补全的图（由于linux上缺少像PS之类的修图软件，所以我通过程序修改BMP文件内容达到抠图效果）

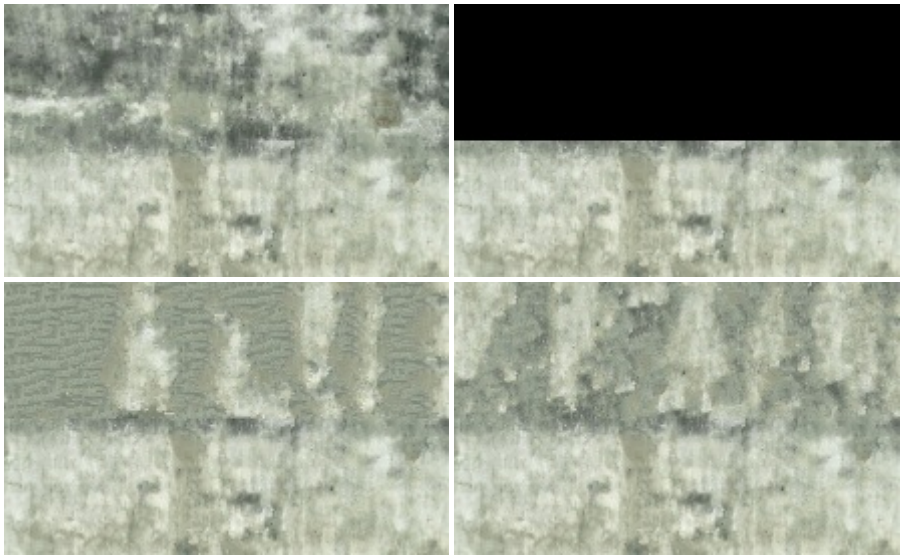
**原始方法**

几何图形



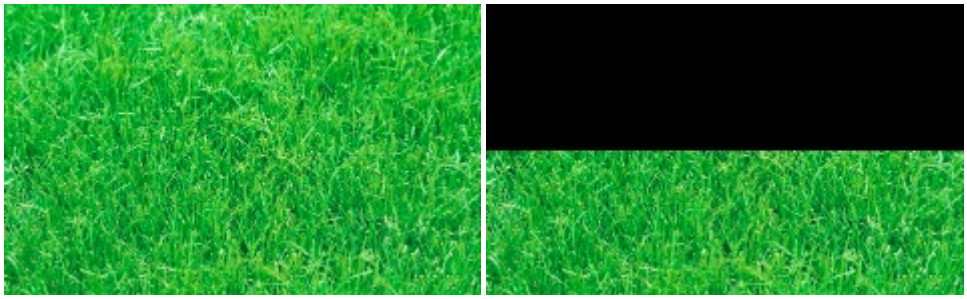
**kd-tree加速（除了特殊说明，patch=11×11）**

岩石（最后一个图为patch=5×5的结果）



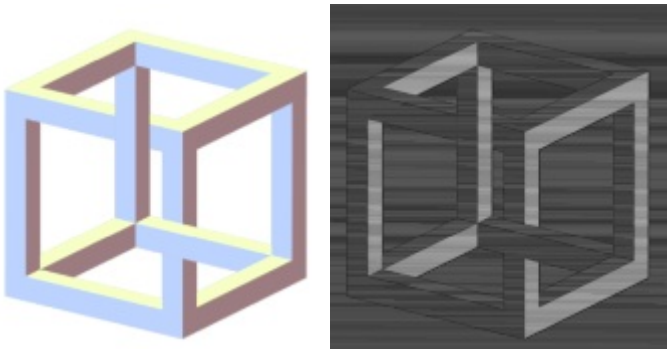
纹理

草地



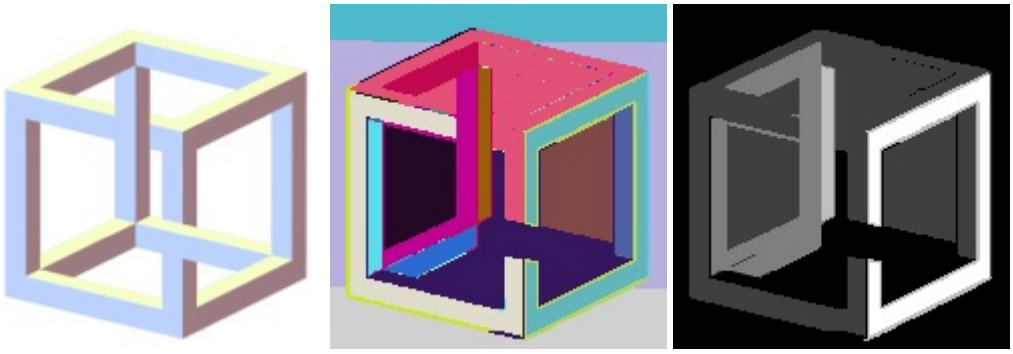
**2.2 显著性检测**

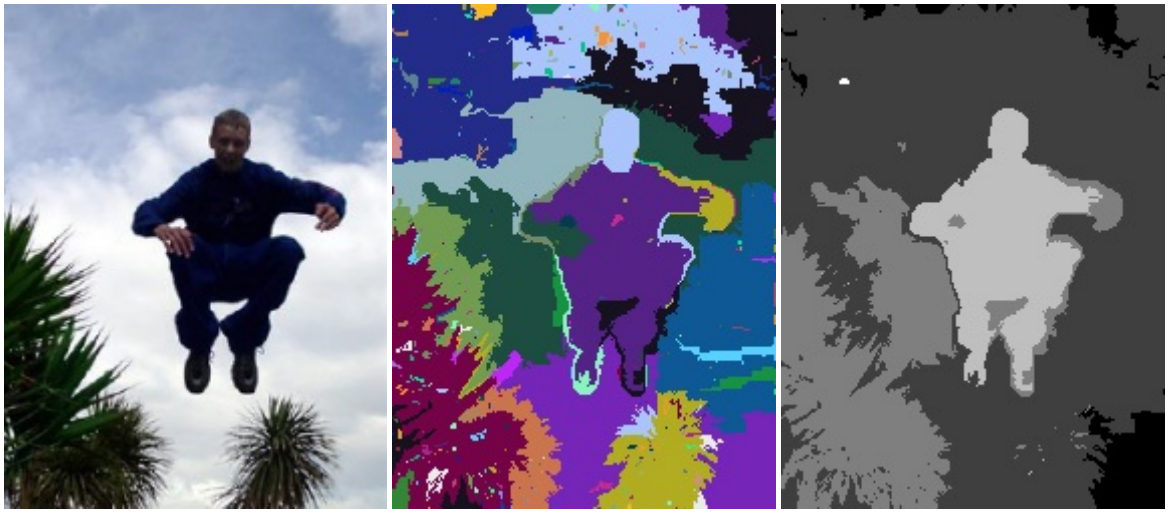
**原始方法**（原图-显著性表示，越亮越显著，256个颜色值量化为8个离散值）



**基于图分割的方法**（原图-区域分割-显著性表示，越亮越显著）

k=3000





k=2000