浙江大学

图形学研究进展课程项目报告

项目名称：GrabCut 实现

姓　　名：张程易 张瑜安 杨晗

学　　号：3150104031 3150102227 3150104251

2018 年　6 月 25 日

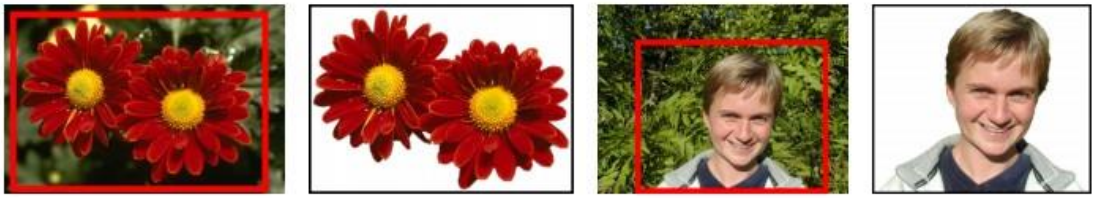
GrabCut 实现

一、 项目介绍

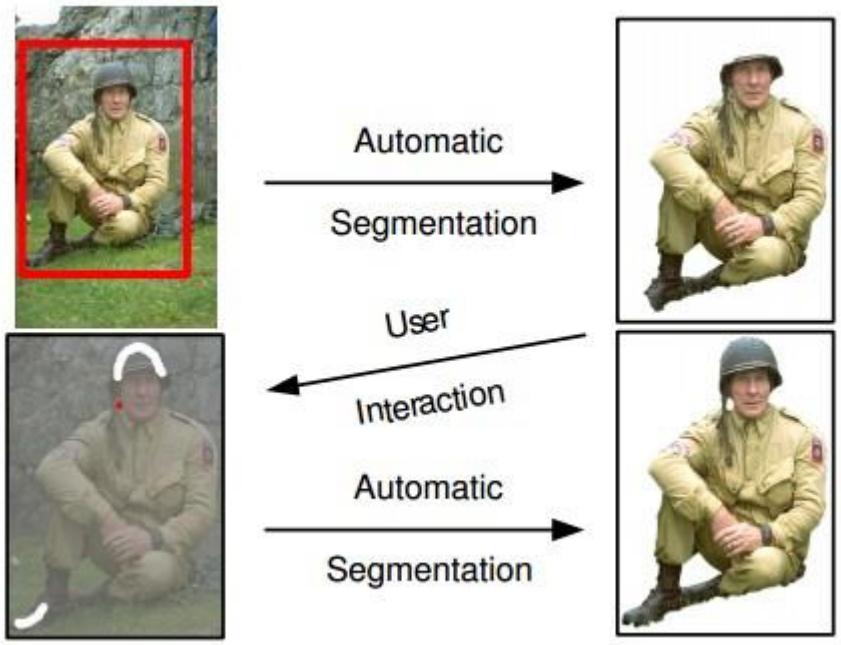
本项目源于微软研究院的 Grab Cut。在这次课程设计中项目实现了基于论文《“GrabCut” — Interactive Foreground Extraction using Iterated Graph Cuts》的用于图像分割的工程。该工程分为两部分，首先是基于 GMM 的图像硬分割的实现，然后是优化分割结果的 Border Matting 的实现。

“GrabCut”，利用迭代的 graph cuts 进行交互式前景提取。对静态图像高效的、交互的前景/背景分割，对图像编辑具有重大的现实意义。

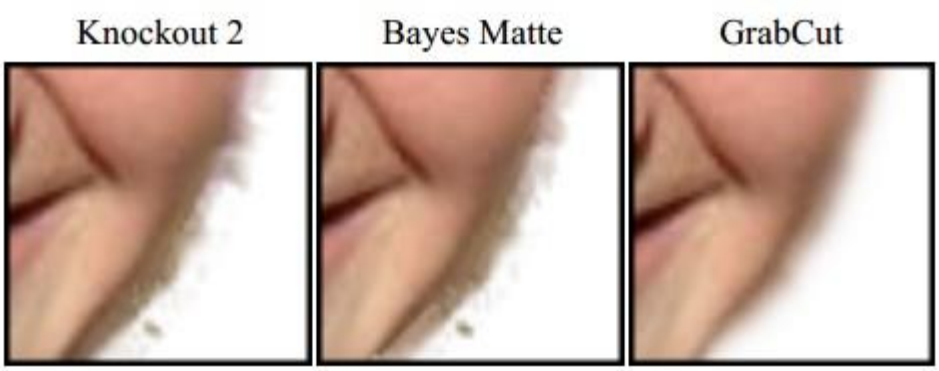
（1）GrabCut 实现的抠图效果。用户只需要在物体外围拉一个包住目标物体的矩形框，就可以实现较好的分割：



（2）当前景和背景颜色比较相似的时候，相似的部分可能难以分割出来。这时候可以利用前/背景画刷进行交互，达到更理想的抠图效果：



（3）除此之外，GrabCut 的 Border Matting 功能可使分割边缘自然而平滑，下图是和其他两种 Matting 算法效果的对比：



这次实验的目标就是完成 GrabCut 的算法，包括 Border Matting 部分，实现交互式抠图。

二、 基本原理

流程

1. 用户输入一个矩形。矩形外的所有区域肯定都是背景（我们在前面已经提到，所有的对象都要包含在矩形框内）。矩形框内的东西是未知的。同样用户确定前景和背景的任何操作都不会被程序改变。
2. 计算机会对我们的输入图像做一个初始化标记。它会标记前景和背景像素。
3. 使用一个高斯混合模型（GMM）对前景和背景建模。
4. 根据我们的输入， GMM 会学习并创建新的像素分布。对那些分类未知的像素（可能是前景也可能是背景），可以根据它们与已知分类（如背景）的像素的关系来进行分类（就像是在做聚类操作）。
5. 这样就会根据像素的分布创建一副图。图中的节点就是像素点。除了像素点做节点之外还有两个节点： Source_node 和 Sink_node。所有的前景像素都和 Source_node 相连。所有的背景像素都和 Sink_node 相连。
6. 将像素连接到 Source_node/end_node 的（边）的权重由它们属于同一类（同是前景或同是背景）的概率来决定。两个像素之间的权重由边的信息或者两个像素的相似性来决定。如果两个像素的颜色有很大的不同，那么它们之间的边的权重就会很小。
7. 使用 MaxFLow 算法对上面得到的图进行分割。它会根据最低成本方程将图分为 Source_node 和 Sink_node。成本方程就是被剪掉的所有边的权重之和。在裁剪之后，所有连接到 Source_node 的像素被认为是前景，所有连接到 Sink_node 的像素被认为是背景。
8. 继续这个过程直到分类收敛。

K-Means

用于初始化 GMM 模型。
根据论文中设置聚类数为 5，即有 5 个高斯分量。然后初始化各个分量的参数 mean 和 var。 图像有三个通道，需要分别初始化每一个通道在每一个分量中的

参数。

GMM

GrabCut 采用 GMM 方法来聚类前景点和背景点。并根据聚类训练出的 GMM 参数来判定一个点属于前景或背景的概率，从而得到数据项。

EM

GMM 通过 EM 算法进行训练。
估计数据由每个 高斯分量生成的概率。对于每个数据 x_i，它由第 k 个高斯分量生成的概率为:

$$\gamma(i,k) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

通过极大似然法估计每个参数，就上上式中的均值 μ 和方差 Σ.

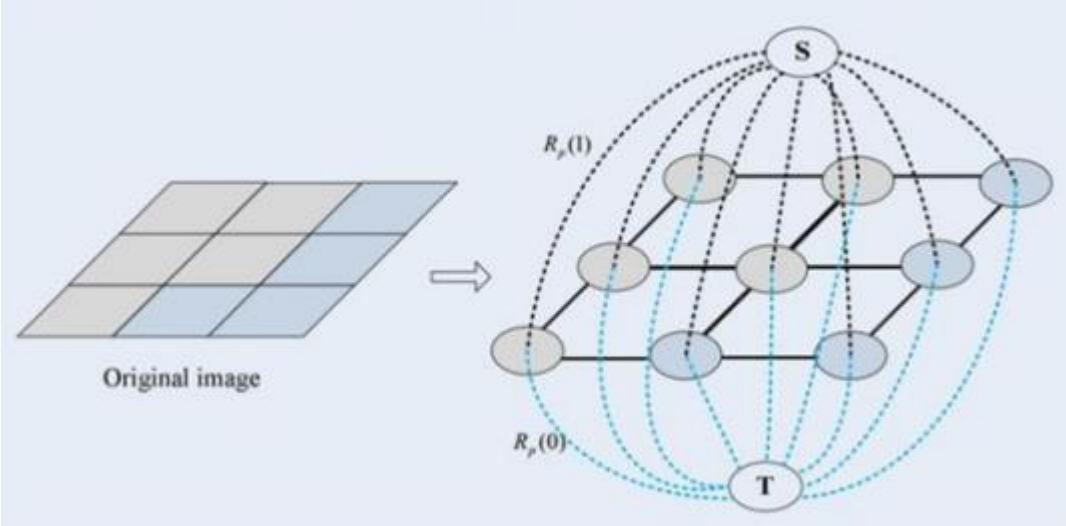
$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(i,k) x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(i,k)(x_i - \mu_k)(x_i - \mu_k)^T$$

重复迭代前两步直到方差和均值收敛即变化小于阈值或达到最大迭代次数不收敛。

MaxFlow

对于每一个像素与它邻接的 8×8 像素之间构造边， 并构造出两个虚拟的节点来表示源节点 S 与汇聚点 T，以合适的权值对这些边赋值，然后应用网络流算法中的 MaxFlow，就能判定出节点是属于源点还是汇聚点



GrabCut 能量函数

用于给最大流的图的边的权重赋值。
能量函数定义如下：

$$\mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) + V(\underline{\alpha}, \mathbf{z}),$$

U 的定义如下：

$$D(\alpha_n, k_n, \theta, z_n) = -log[p(z_n|\alpha_n, k_n, \theta) \cdot \pi(\alpha_n, k_n)]$$

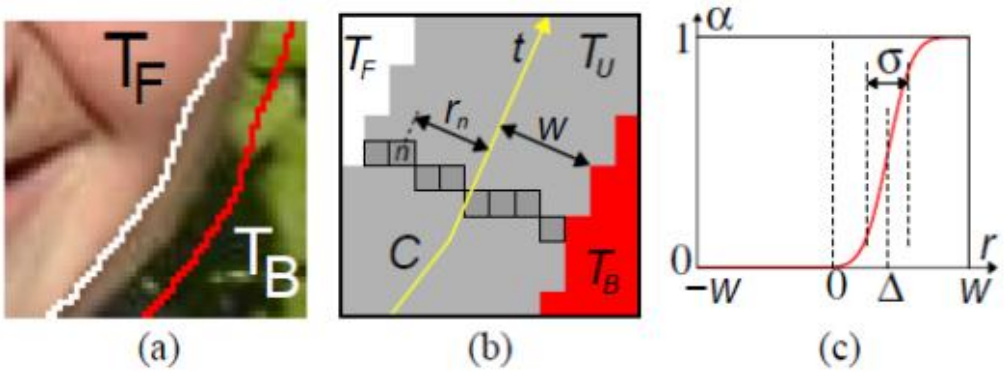
U 是在描述该像素点在 k_n 这个高斯分量中的概率，在乘上该高斯分量在 GMM 模型中的系数，然后再取负对数，这样就得到了数据项的能量，概率越小则能量越大。
V 是平滑项，定义如下：

$$V(\underline{\alpha}, \mathbf{z}) = \gamma \sum_{(m,n)\in \mathbf{C}} [\alpha_n \neq \alpha_m] \exp -\beta \|z_m - z_n\|^2 .$$

构造出这样的图后应用 MaxFlow 的算法就可以得到更新后的 α 。

Border Matting

Border Matting 的核心想法在于如何把 Grab Cut 中迭代得到的硬分割边缘变成软分割，即 α 不再简单取 0,1 而是可以取[0-1]，这样就能得到比较光顺的边缘效果。



(a) (b) (c)

如图中所示概念，首先构造出图像边缘及其邻居的集合 Tu，采用平滑函数 g 来计算 α 的取值，核心想法是如何计算出中心点及其附近邻居的 α 使得结果最优，那么调整 g 的参数可以做到这一点，论文中提出的方法也是采用能量最小化的思想，这里的能量函数如下定义：

$$E = \sum_{n \in T_U} \tilde{D}_n(\alpha_n) + \sum_{t=1}^{T} \tilde{V}(\Delta_t, \sigma_t, \Delta_{t+1}, \sigma_{t+1})$$

这里的数据项 D 可以直观的理解为当前像素在周围像素的融洽程度，而这里的平滑项 V 则是指相邻的中心点前后之间的变化幅度，因此这个函数最小化的目的就是使得轮廓上的像素尽可能的有着比较好的 α 使得它在周围像素看起来不那么突兀比较融洽，同时轮廓的相邻点之间应该尽可能的平滑，变化幅度不应太大以至于产生撕裂状的锯齿。 所以 D 定义如下：

$$\tilde{D}_n(\alpha_n) = -\log \mathbf{N}\left(z_n; \mu_{t(n)}(\alpha_n), \Sigma_{t(n)}(\alpha_n)\right)$$

即融洽程度被刻画为当前 α 在周围像素近似以高斯分布的情况下的概率， 这个高斯分布的参数以如下的方式由前景和背景参数刻画：

$$\begin{aligned} \mu_t(\alpha) &= (1-\alpha)\mu_t(0) + \alpha\mu_t(1) \\ \Sigma_t(\alpha) &= (1-\alpha)^2\Sigma_t(0) + \alpha^2\Sigma_t(1). \end{aligned}$$

而平滑项 V 则是前后像素的拟合函数 g 不能差别太大：

$$\tilde{V}(\Delta, \sigma, \Delta', \sigma') = \lambda_1(\Delta - \Delta')^2 + \lambda_2(\sigma - \sigma')^2.$$

论文中提出使得该能量函数最小化用的是一种近似的 DP 算法，并指出如果不用 distance transform 则复杂度为 O(N^4)。如果加上 distance transform 则可以 O(N)完成。

三、 项目框架

依赖的库

OpenCV
CVUI

文件结构

main.cpp
WindowFramework.cpp
WindowFramework.h
GaussMixtureModel.cpp
GaussMixtureModel.h
GrabCut.cpp
GrabCut.h
KMeansImpl.cpp
KMeansImpl.h
BorderMatting.cpp
BorderMatting.h
Dinic.cpp
Dinic.h

各部分介绍

Main 函数里就是启动 GUI。
WindowFramework 是基于 CVUI 库实现的一个 GUI 框架。
GrabCut 中实现了 GrabCutImplementation 类，实现了 GrabCut 的主要算法流程。
KmeansImpl 中有 KMeans 算法的实现
GaussMixtureModel 中有 GMM 及其训练算法的实现
Dinic 用于 MaxFlow 算法的实现
BorderMatting 中实现了 BorderMatting 算法

四、 实现细节

Main.cpp

启动 GUI

```
1 #include "WindowFramework.h"
 2 #define CVUI_IMPLEMENTATION
```

```
#include "cvui.h"

WindowFramework windowFramework;

int main(int argc, char **argv) {
    windowFramework.initWindow();
    windowFramework.loadImage();
    windowFramework.initState();
    for (;;) {
        windowFramework.listenButton();
        windowFramework.listenMouse();
        windowFramework.display();
        if (cv::waitKey(20) == 27 || windowFramework.shouldExit()) {
            break;
        }
    }
    return 0;
}
```

GUI 的接口

```
class WindowFramework {
public:
    void initWindow();
    void loadImage();
    void initState();
    void display();
    void listenMouse();
    void doGrabCut();
    void listenButton();
    bool shouldExit();
    bool initGrabCut();
    GrabCutImplementation grabCutImplementation;
private:
    static const Scalar PEN_BGD_COLOR;
    static const Scalar PEN_FGD_COLOR;
```

```
16      static const Scalar PEN_RECT;
17
18      static const int DRAW_BGD;
19      static const int DRAW_FGD;
20      static const int DRAW_RECT;
21
22      Mat frame;
23      Mat image;
24      Mat mask;
25
26      Rect choosingBox;
27      vector<Point> foregroundPixels, backgroundPixels;
28      int drawingFlag;
29      bool exitFlag;
30      drawState drawingState;
31      bool beforeGraphCut;
32      bool hasMask;
33 };
```

GrabCut 的接口

```
1 class GrabCutImplementation {
 2 public:
 3      double calculateBeta(const Mat &image);
 4
 5      void calMaskByBox(Mat payloadMask, Rect box);
 6
 7      void doBorderMatting();
 8
 9      void grabCut(cv::InputArray imageInputArray,
cv::InputOutputArray maskIOArray, int iterCount);
10
11 private:
12      Mat payloadImage;
13      Mat payloadMask;
```

```
14
15      void calculateNWeights(const Mat image, double beta, Mat left, Mat
upLeft, Mat up, Mat upRight);
16
17      void constructNetwork(const Mat image, Mat mask, Mat left, Mat
upLeft, Mat up, Mat upRight,
18                            GaussMixtureModel &backgroundGmm,
19                            GaussMixtureModel &foregroundGMM, Dinic
*network);
20 };
```

GMM 初始化

GMM 通过 KMeans 的方式进行初始化。
初始化时，会把矩形内的点作为前景，矩形外的点作为背景，运行 Kmeans 算法，用于初始化 GMM 模型。

```
1 void GaussMixtureModel::init(double *data, int N) {
 2     const double MIN_VAR = 1E-10;
 3     int *Label = new int[N];
 4     KMeansImpl kmeans = KMeansImpl(dimensionNumber, mixtureNumber);
 5     kmeans.cluster(data, N, Label);
 6
 7     int *counts = new int[mixtureNumber];
 8     double *overMeans = new double[dimensionNumber];
 9     for (int i = 0; i < mixtureNumber; i++) {
10         counts[i] = 0;
11         priors[i] = 0;
12         memcpy(means[i], kmeans.means[i], sizeof(double) *
dimensionNumber);
13         memset(vars[i], 0, sizeof(double) * dimensionNumber);
14     }
15     memset(overMeans, 0, sizeof(double) * dimensionNumber);
16     memset(minVars, 0, sizeof(double) * dimensionNumber);
17
18     int size = 0;
19     size = N;
```

```
20
21      double *x = new double[dimensionNumber];
22      int label = -1;
23
24      for (int i = 0; i < size; i++) {
25          for (int j = 0; j < dimensionNumber; j++)
26              x[j] = data[i * dimensionNumber + j];
27          label = Label[i];
28
29          counts[label]++;
30          double *m = kmeans.means[label];
31          for (int d = 0; d < dimensionNumber; d++) {
32              vars[label][d] += (x[d] - m[d]) * (x[d] - m[d]);
33          }
34
35          for (int d = 0; d < dimensionNumber; d++) {
36              overMeans[d] += x[d];
37              minVars[d] += x[d] * x[d];
38          }
39      }
40
41      for (int d = 0; d < dimensionNumber; d++) {
42          overMeans[d] /= size;
43          minVars[d] = max(MIN_VAR, 0.01 * (minVars[d] / size -
overMeans[d] * overMeans[d]));
44      }
45
46      for (int i = 0; i < mixtureNumber; i++) {
47          priors[i] = 1.0 * counts[i] / size;
48
49          if (priors[i] > 0) {
50              for (int d = 0; d < dimensionNumber; d++) {
51                  vars[i][d] = vars[i][d] / counts[i];
52
53                  if (vars[i][d] < minVars[d]) {
```

```
54                      vars[i][d] = minVars[d];
55                  }
56              }
57          } else {
58              memcpy(vars[i], minVars, sizeof(double) *
dimensionNumber);
59          }
60      }
61      delete[] x;
62      delete[] counts;
63      delete[] overMeans;
64      delete[] Label;
65
66 }
```

GMM 训练

GMM 通过 EM 算法进行训练，具体原理和步骤在前面部分已经讲过，通过迭代来更新 GMM 模型的参数。
进行训练：

```
1          auto *data = new double[foregroundPixelArray.size()];
2          for (int i = 0; i < foregroundPixelArray.size(); i++)
3              data[i] = foregroundPixelArray[i];
4          foregroundGmm.train(data, foregroundPixelArray.size() / 3);
5          delete[]data;
6          data = new double[backgroundPixelArray.size()];
7          for (int i = 0; i < backgroundPixelArray.size(); i++)
8              data[i] = backgroundPixelArray[i];
9          backgroundGmm.train(data, backgroundPixelArray.size() / 3);
```

训练的具体代码：

```
1 void GaussMixtureModel::train(double *data, int N) {
 2      init(data, N);
 3
 4      int size = N;
 5
```

```
 6      double iterNum = 0;
 7      double lastL = 0;
 8      double currL = 0;
 9      int unchanged = 0;
10      double *x = new double[dimensionNumber];
11      double *nextPriors = new double[mixtureNumber];
12      double **nextVars = new double *[mixtureNumber];
13      double **nextMeans = new double *[mixtureNumber];
14
15      for (int i = 0; i < mixtureNumber; i++) {
16          nextMeans[i] = new double[dimensionNumber];
17          nextVars[i] = new double[dimensionNumber];
18      }
19
20      for(;;) {
21          memset(nextPriors, 0, sizeof(double) * mixtureNumber);
22          for (int i = 0; i < mixtureNumber; i++) {
23              memset(nextVars[i], 0, sizeof(double) *
dimensionNumber);
24              memset(nextMeans[i], 0, sizeof(double) *
dimensionNumber);
25          }
26
27          lastL = currL;
28          currL = 0;
29
30          for (int k = 0; k < size; k++) {
31              for (int j = 0; j < dimensionNumber; j++)
32                  x[j] = data[k * dimensionNumber + j];
33              double p = getProbability(x);
34
35              for (int j = 0; j < mixtureNumber; j++) {
36                  double pj = getProbability(x, j) * priors[j] / p;
37
38                  nextPriors[j] += pj;
```

```
39
40                  for (int d = 0; d < dimensionNumber; d++) {
41                      nextMeans[j][d] += pj * x[d];
42                      nextVars[j][d] += pj * x[d] * x[d];
43                  }
44              }
45
46              currL += (p > 1E-20) ? log10(p) : -20;
47          }
48          currL /= size;
49
50          for (int j = 0; j < mixtureNumber; j++) {
51              priors[j] = nextPriors[j] / size;
52
53              if (priors[j] > 0) {
54                  for (int d = 0; d < dimensionNumber; d++) {
55                      means[j][d] = nextMeans[j][d] / nextPriors[j];
56                      vars[j][d] = nextVars[j][d] / nextPriors[j] -
means[j][d] * means[j][d];
57                      if (vars[j][d] < minVars[d]) {
58                          vars[j][d] = minVars[d];
59                      }
60                  }
61              }
62          }
63
64          iterNum++;
65          if (fabs(currL - lastL) < endError * fabs(lastL)) {
66              unchanged++;
67          }
68          if (iterNum >= maxIterationNumber || unchanged >= 3) {
69              break;
70          }
71      }
72      delete[] nextPriors;
```

```
73      for (int i = 0; i < mixtureNumber; i++) {
74          delete[] nextMeans[i];
75          delete[] nextVars[i];
76      }
77      delete[] nextMeans;
78      delete[] nextVars;
79      delete[] x;
80  }
```

MaxFlow

边 Edge 的结构设计如下：

```
1      struct edge {
 2          int x, y;//两个顶点
 3          double c;//容量
 4          double f;//当前流量
 5          edge *next, *back;//下一条边，反向边
 6          edge(int x, int y, int c, edge *next) : x(x), y(y), c(c), f(.0),
next(next), back(0) {}
 7
 8          void *operator new(size_t, void *p) { return p; }
 9      }
```

最大流算法

```
1 double Dinic::flow() {
 2      double res = 0;//结果，即总流量
 3      int path_n;//path 的大小
 4      for (;;) {
 5          DFS();
 6          if (D[T] == -1)break;
 7          memcpy(cur, E, sizeof(E));
 8          path_n = 0;
 9          int i = S;
10          for (;;) {
11              if (i == T) {//已找到一条增广路，增广之
12                  int mink = 0;
13                  double delta = INT_MAX;
```

```
                for (int k = 0; k < path_n; ++k) {
                    if (path[k]->c < delta) {
                        delta = path[k]->c;
                        mink = k;
                    }
                }
                for (int k = 0; k < path_n; ++k) {
                    path[k]->c -= delta;
                    path[k]->back->c += delta;
                }
                path_n = mink;//回退
                i = path[path_n]->x;
                res += delta;
            }
            edge *e;
            for (e = cur[i]; e; e = e->next) {
                if (isZero(e->c))continue;
                int j = e->y;
                if (D[i] + 1 == D[j])break;//找到一条弧，加到路径里
            }
            cur[i] = e;//当前弧结构，访问过的不能增广的弧不会再访问
            if (e) {
                path[path_n++] = e;
                i = e->y;
            } else {//该节点已没有任何可增广的弧，从图中删去，回退一步
                D[i] = -1;
                if (path_n == 0)break;
                path_n--;
                i = path[path_n]->x;
            }
        }
    }
```

最大流算法之后的结果用来更新 mask：

```
calculateNWeights(payloadImage, beta, left, downLeft, down, downRight);
        constructNetwork(payloadImage, payloadMask, left, downLeft, down, downRight, backgroundGmm, foregroundGmm,
                         network);
        network->flow();
        for (int row = 0; row < payloadMask.rows; row++)
            for (int col = 0; col < payloadMask.cols; col++)
                if (payloadMask.at<uchar>(row, col) != GC_FGD)
                    if (network->sourceSet(row * payloadMask.cols + col))
                        payloadMask.at<uchar>(row, col) = GC_PR_FGD;
                    else
                        payloadMask.at<uchar>(row, col) = GC_BGD;
```

BorderMatting

Border Matting 包括是图像轮廓点的采集构建并有序化和能量函数的最小化。
关于轮廓点的采集，先把 Mask 叠在原图上，然后使用 Canny 函数提取轮廓。
先叠加 Mask：

```
    Mat result = Mat(payloadImage.size(), payloadImage.type());
    payloadImage.copyTo(result);
    for (int row = 0; row < result.rows; row++)
        for (int col = 0; col < result.cols; col++) {
            if (payloadMask.at<uchar>(row, col) == 0)
                result.at<Vec3b>(row, col) = Vec3b(255, 255, 255);
        }
```

之后进行运算：

```
    borderMatting.init(payloadImage, payloadMask);
    borderMatting.run();
```

更新 alpha：

```
    for (int i = 1; i < pointArray.size(); i++) {
        Values para;
        getL(pointArray[i].p, para);
```

```
4           pointArray[i].p.para = para;
5           for (int j = 0; j < pointArray[i].neighbor.size(); j++) {
6               borderPoint &p = pointArray[i].neighbor[j];
7               getL(p, para);
8               p.para = para;
9           }
10          double min = INT_MAX;
11          for (int n = 0; n < 30; n++)
12              for (int m = 0; m < 10; m++) {
13                  double t = dataTermPoint(pointArray[i].p,
14
toGray(payloadImage.at<Vec3b>(pointArray[i].p.p.y,
pointArray[i].p.p.x)), n,
15                                                  m, pointArray[i].p.para);
16                  for (int j = 0; j < pointArray[i].neighbor.size(); j++)
{
17                      borderPoint &p = pointArray[i].neighbor[j];
18                      t += dataTermPoint(p,
toGray(payloadImage.at<Vec3b>(p.p.y, p.p.x)), n, m, p.para);
19                  }
20                  double V = 2 * (n - delta) * (n - delta) + 360 * (sigma
- m) * (sigma - m);
21                  if (t + V < min) {
22                      min = t + V;
23                      pointArray[i].p.delta = n;
24                      pointArray[i].p.sigma = m;
25                  }
26              }
27          sigma = pointArray[i].p.sigma;
28          delta = pointArray[i].p.delta;
29          pointArray[i].p.alpha = g[0][delta][sigma];
30          for (int j = 0; j < pointArray[i].neighbor.size(); j++) {
31              borderPoint &p = pointArray[i].neighbor[j];
32              p.alpha = g[p.dis][delta][sigma];
33          }
```

```
34      }
```

利用 alpha 计算出结果：

```
1      Mat _alphaMask = Mat(payloadMask.size(), CV_32FC1, Scalar(0));
 2      for (int i = 0; i < payloadMask.rows; i++)
 3          for (int j = 0; j < payloadMask.cols; j++)
 4              _alphaMask.at<float>(i, j) = payloadMask.at<uchar>(i,
j);
 5      for (int i = 0; i < pointArray.size(); i++) {
 6          _alphaMask.at<float>(pointArray[i].p.p.y,
pointArray[i].p.p.x) = pointArray[i].p.alpha;
 7          for (int j = 0; j < pointArray[i].neighbor.size(); j++) {
 8              borderPoint &p = pointArray[i].neighbor[j];
 9              _alphaMask.at<float>(p.p.y, p.p.x) = p.alpha;
10          }
11      }
12
13      Mat result = Mat(payloadImage.size(), CV_8UC4);
14      for (int i = 0; i < result.rows; i++)
15          for (int j = 0; j < result.cols; j++) {
16              result.at<Vec4b>(i, j) = Vec4b(payloadImage.at<Vec3b>(i,
j)[0], payloadImage.at<Vec3b>(i, j)[1],
17
payloadImage.at<Vec3b>(i, j)[2],
18                                                  _alphaMask.at<float>(i,
j) * 255);
19          }
```

最后利用一下 PNG 的存储：

```
1      vector<int> compressionParams;
2      compressionParams.push_back(CV_IMWRITE_PNG_COMPRESSION);
3      compressionParams.push_back(9);
4      imwrite("BorderMatting.png", result, compressionParams);
5      result.copyTo(borderMattingResult);
6      printf("Boder Matting: done.");
```

五、 实验结果与分析（记录实验结果，并进行比较和分析）

实验环境

操作系统：macOS 10.13.5
CPU：i5-6267U 2.9Ghz
内存：8GB

实验结果

下面的实验对比结果已经尽量将其他环境变量控制一致，时间对比为多次测试取平均值的结果。参数中，K-means 聚类取 5 类，gamma=50,lambda=9*gamma

实验一

原图

分辨率 200*133

对比结果

第一张图像分辨率较低，图像中的目标与背景的区分度较大，两种实现都能较快且较准确地将目标与背景分割开来，我们的实现方式会比 OpenCV 的方法快一些，但是 OpenCV 的方法能够在 3 次迭代实现我们方法的 5 次迭代效果，在这个问题上，我们分析原因可能是我们的方法在构造 GMM 模型时，GMM 模型的准确度不如 OpenCV 的实现，因此导致需要迭代的次数更多才能得到更好的 GMM 模型。

实验二

原图

分辨率 255*200

对比结果

这张图像的分辨率相比第一张大，所以处理时间要比上一张长，但是我们的方法仍然比 OpenCV 的方法要快一些，且最终迭代三次的效果二者不相上下。但是两种方法在分割时都将人的肩膀部分去掉，下面采用标记前景点的方法将肩膀标记成前景。

上面是两种方法分别标记前景点的结果，右侧的图像是迭代三次的结果。由于标记点不同，所以最终的结果有些许差异，但可以就看出二者都能够将肩膀处的衣服恢复出来。

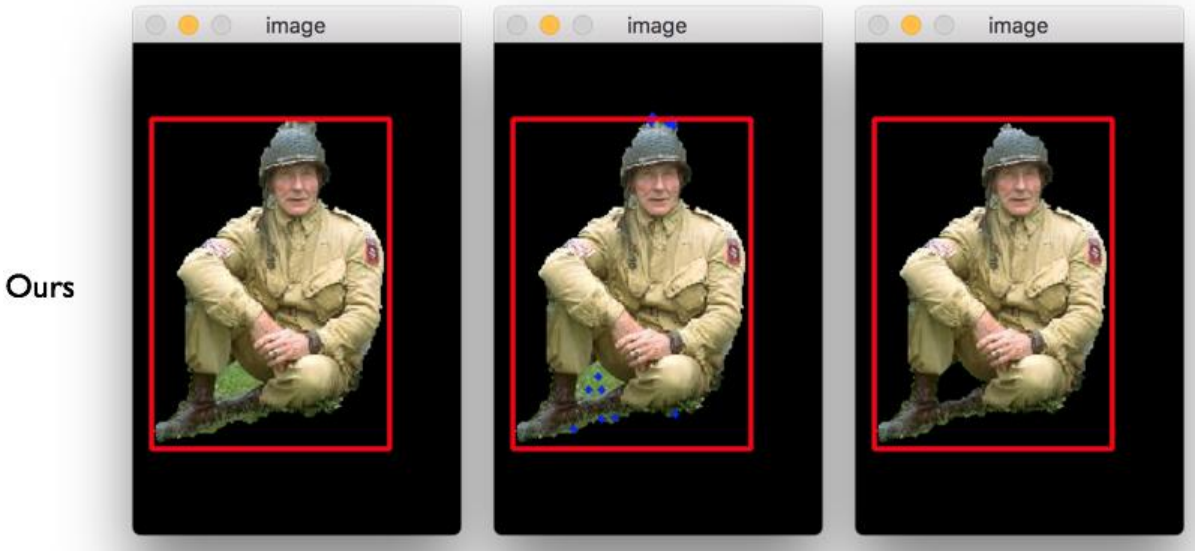
实验三

原图

分辨率 200*300

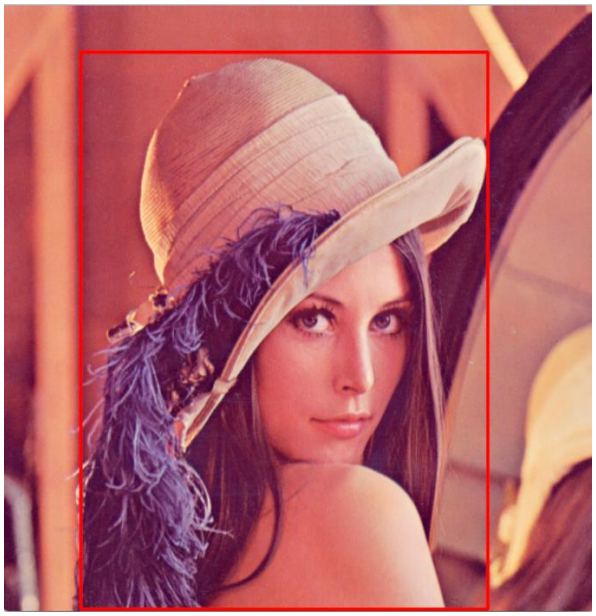
对比结果

下面左侧是 OpenCV 迭代两次的时间，右侧是我们的方法迭代两次的时间，我们的方法在时间上仍然较 OpenCV 快。这张图像背景和目标图像的颜色比较相近，因此两种方法都表现不是很好，将头盔和靴子当做了背景，然后我们可以将前景进行标记再进行迭代，后两张图像是标记与标记迭代一次的结果。我们的方法生成的结果又将草坪当成了前景，因此需要再将这一部分标记为背景，迭代后如下图所示。



实验四

原图



分辨率 512*512

对比结果

这张图像分辨率较大，同时图像的目标与背景颜色相近，对分割算法构成了较大挑战。我们的实现在时间上呈现了大小不定的结果，整体效果要比 OpenCV 差一些。

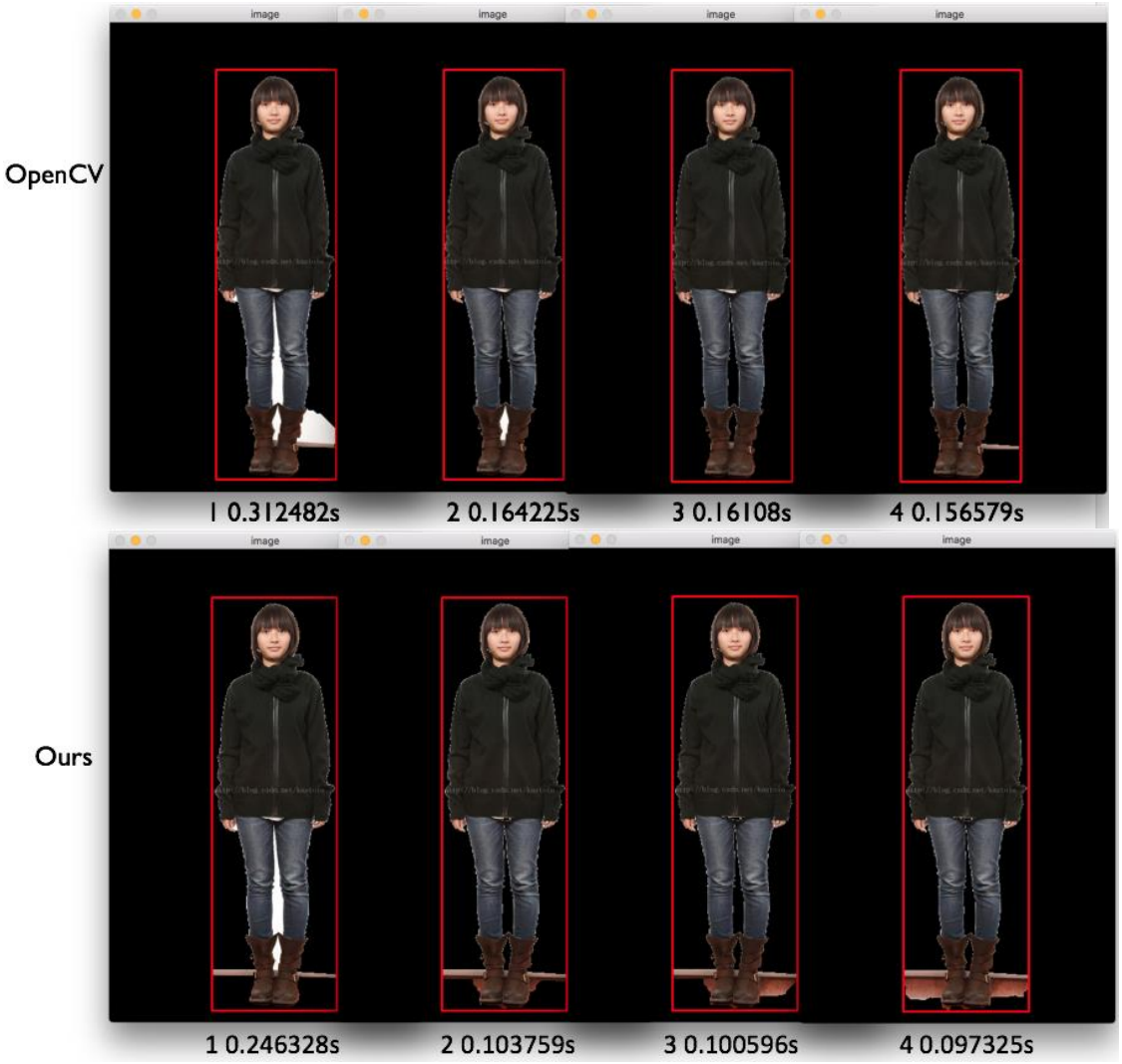
在结果图上我们标记一些前景点可以得到更好的分割结果，如下图所示。

实验五

原图

分辨率 400*600

对比结果



这张图像上半部分目标与背景图区别较明显，但最下面鞋与地板的颜色较为相近，因此也会给算法带来挑战，实验结果表明无论是 OpenCV 的实现还是我们的实现，对于鞋的部分分割结果并不能让人满意，我们的分割结果更差一些，原因依然是 GMM 模型不够精确。

Border Matting

下面是对分割图像进行 Border Matting 的结果。




Border Matting 的效果很明显，算法将硬分割的结果边缘进行了修饰，使得边缘更加平滑。

存在的问题

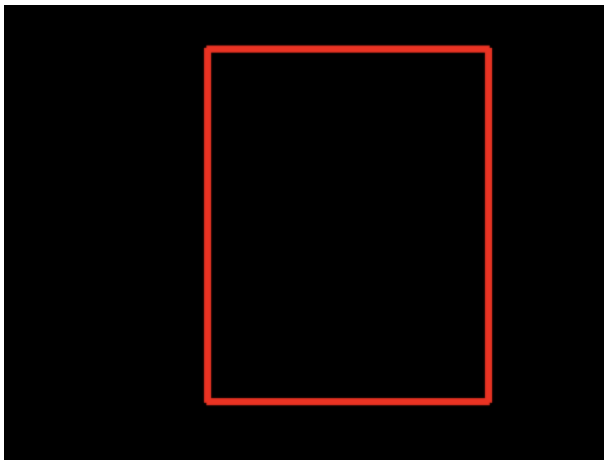
我们实现的 Grab Cut 在对某些图像进行分割后，得到的结果图与理想的图像偏差很大，例如下面两个实例。

实例一

原图

期望结果

实际结果

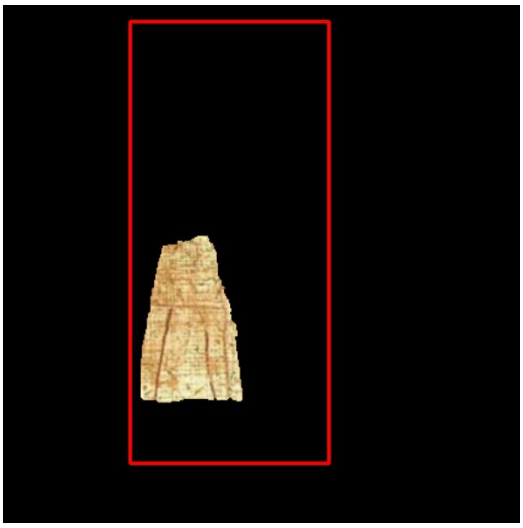


实际在第一次迭代时效果看起来不错，但在迭代的第二次所有像素点都消失了，即所有点都被认为可能是背景点。这个问题在 OpenCV 的实现中并没有出现。其中可能的原因是 GMM 模型在这张图上出现了数值异常，但因时间原因，我们没有将 GMM 模型可视化出来，因此更具体的原因我们会进一步研究。

实例二

原图

实际结果



在实际的分割过程中，无论是 OpenCV 的实现还是我们的实现都无法在直接迭代过程中得到理想的分割结果。右图是手动标记一些前景点后再迭代两次的结果，标记时已经尽可能地将脚部、身体、头和手臂都标记了，但结果仍然无法令人满意。我们还尝试了调整算法中 K-means 聚类时的聚类数量以尽可能让 GMM 模型建立起更复杂的颜色模型，但效果并不明显。

这个结果表明了 Grab Cut 算法在分割类似的图像时无能为力，对于前景和背景差异很小的图像，我们只能寻求手动标记或其他分割算法的帮助。

六、 结论与体会

这次实验实现了《“GrabCut” — Interactive Foreground Extraction using Iterated Graph Cuts》。完成了 GrabCut 的整个流程，理解并实现了算法。在项目中提高了工程能力和阅读文献、论文的能力。在项目中，对 KMeans，GMM，MaxFlow 等算法有了更深的认识。我们参考了网络上的一些实现，把所有代码都自己手写实现一遍。我们的实现相比于 OpenCV 的实现多了 BorderMatting，对于毛发边缘等地方的效果要好一些。但是仍有一些地方有效果不够好的情况，我们也进行了分析，尽可能地做出了我们的尝试。

七、 分工

张瑜安：KMeans， GMM， MaxFlow 主要步骤实现
张程易：Border Matting 实现， 项目展示
杨晗： UI 设计实现，实验结果并对比 OpenCV 等其他版本，文档书写

参考文献

1. ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. Grabcut- interactive foreground extraction using iterated graph cut. In Proc. of ACM SIGGRAPH, 2004,309–314.

2. BOYKOV, Y., AND JOLLY, M.-P. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In ICCV, 2001, 105–112.

3. http://en.wikipedia.org/wiki/Mixture_model#Gaussian_mixture_model

4. http://vision.csd.uwo.ca/code/maxflow

5. http://www.juew.org/publication/mattingSurvey.pdf

6. https://docs.opencv.org/3.2.0/d8/d83/tutorial_py_grabcut.html