SeamCarving实验报告

笔记本:	SeamCarving		
创建时间:	2018/6/16 17:28	**更新时间:**	2018/6/30 1:12
标签:	图像大作业		

SeamCarving实验报告

计63 李志鹏 2016011283

实验目标

复现论文**Seam Carving for Content-Aware Image Resizing, SIGGRAPH 2007**，实现内容敏感的图片缩放及区域保护和删除功能。

完成情况

我实现了

- **图片双向缩小**
- **图片双向放大**
- **图像区域保护和删除**

实现细节

每一个功能以代码段的形式给出

公用部分

找最小能量线

```
std::vector<int> searchMinSeam(cv::Mat1i &mat)
{
    int m = mat.size[0];
    int n = mat.size[1];
```

```
    for (int i = 0; i < n; i++)
        dp[0][i] = mat.at<int>(0, i);
    for (int i = 1; i < m; i++)
        for (int j = 0; j < n; j++)
            dp[i][j] = searchMin(&dp[i - 1][std::max(0, j - 1)], std::min(n, j + 2) - std::max(0, j - 1)) + mat.at
<int>(i, j);

    std::vector<int> cols(m);
    cols[m - 1] = searchMinPtr(dp[m - 1], n) - dp[m - 1];

    for (int i = m - 2; i >= 0; i--)
        cols[i] = searchMinPtr(&dp[i][std::max(0, cols[i + 1] - 1)], std::min(n, cols[i + 1] + 2) - std::max(0, co
ls[i + 1] - 1)) - dp[i];
    for (int i = 0; i < m; i++)
    {
        mat.at<int>(i, cols[i]) = INF;
        mat.at<int>(i, std::max(0, cols[i] - 1)) = INF;
        mat.at<int>(i, std::min(n - 1, cols[i] + 1)) = INF;
    }

    return cols;
}
```

先做一次dp，然后迭代中当前将当前最小的能量线取出，并且将其能量值赋为正无穷。

双向缩小

```
cv::Mat shrinkCarving(const cv::Mat src, const std::vector<int> &seams)
{
    cv::Mat ans = cv::Mat(src.size[0], src.size[1] - 1, src.type());

    for (int i = 0; i < src.size[0]; i++)
        memcpy(ans.data + ans.step[0] * i,
            src.data + src.step[0] * i,
            src.step[1] * seams[i]);

    for (int i = 0; i < src.size[0]; i++)
        memcpy(ans.data + ans.step[0] * i + ans.step[1] * seams[i],
            src.data + src.step[0] * i + src.step[1] * (seams[i] + 1),
            src.step[1] * (src.size[1] - seams[i] - 1));

    return ans;
}

cv::Mat shrink(const cv::Mat &src, int x, int y, cv::Mat &mask, function f)
{
    auto image = src.clone();
    int col = src.size[1] - y > 0 ? src.size[1] - y : 0;
    int row = src.size[0] - x > 0 ? src.size[0] - x : 0;
```

```
    while (col--)
    {
        auto tmat = applyMask(f(image), mask);
        auto seams = searchMinSeam(tmat);
        mask = shrinkCarving(mask, seams);
        image = shrinkCarving(image, seams);
    }

    image = transpose(image);
    mask = transpose(mask);

    while (row--)
    {
        auto tmat = applyMask(f(image), mask);
        auto seams = searchMinSeam(tmat);
        mask = shrinkCarving(mask, seams);
        image = shrinkCarving(image, seams);
    }

    image = transpose(image);
    mask = transpose(mask);

    return image;
}
```

先进行横向的缩小，然后将图片旋转，再利用同样的函数进行纵向的缩小。

双向放大

```
void quicksort(int a[], std::vector<std::vector<int>> seams, int l, int r)
{
    if (l < r)
    {
        int i = l, j = r;
        std::vector<int> temp = seams[l];
        int tempv = seams[l][0];
        int tempi = a[l];
        while (i < j)
        {
            while (i < j && seams[j][0] >= tempv)
                j--;
            if (i < j)
            {
                a[i] = a[j];
                seams[i] = seams[j];
                i++;
            }
            while (i < j && seams[i][0] < tempv)
                i++;
```

```
            if (i < j)
            {
                a[j] = a[i];
                seams[j] = seams[i];
                j--;
            }
        }
        a[i] = tempi;
        seams[i] = temp;
        quicksort(a, seams, l, i - 1);
        quicksort(a, seams, i + 1, r);
    }
}

cv::Mat enlargeCarving(const cv::Mat src, const std::vector<std::vector<int>> &seams)
{
    cv::Mat ans = cv::Mat(src.size[0], src.size[1] + seams.size(), src.type());

    for (int i = 0; i < seams.size(); i++)
        id[i] = i;
    quicksort(id, seams, 0, seams.size() - 1);

    std::vector<int> last = std::vector<int>(src.size[0], 0);
    std::vector<int> current = seams[id[0]];
    for (int i = 0; i < seams.size(); i++)
    {
        current = seams[id[i]];
        for (int j = 0; j < src.size[0]; j++)
        {
            memcpy(ans.data + ans.step[0] * j + ans.step[1] * (last[j] + i),
                src.data + src.step[0] * j + src.step[1] * last[j],
                src.step[1] * (current[j] - last[j]));
            if (last[j] != 0)
                ans.at<cv::Vec3b>(j, last[j] + i - 1) = ((src.at<cv::Vec3b>(j, last[j]) + src.at<cv::Vec3b>(j, las
t[j] - 1)) / 2);
        }
        last = current;
    }

    for (int j = 0; j < src.size[0]; j++)
    {
        memcpy(ans.data + ans.step[0] * j + ans.step[1] * (last[j] + seams.size()),
            src.data + src.step[0] * j + src.step[1] * last[j],
            src.step[1] * (src.size[1] - last[j]));
        if (last[j] != 0)
            ans.at<cv::Vec3b>(j, last[j] + seams.size() - 1) = ((src.at<cv::Vec3b>(j, last[j]) + src.at<cv::Vec3b>
(j, last[j] - 1)) / 2);
    }

    return ans;
}
```

```
cv::Mat enlarge(const cv::Mat &src, int x, int y, function f)
{
    auto image = src.clone();


    auto mat = f(image);
    auto seams = searchMinSeam(mat, (y - src.size[1]));
    image = enlargeCarving(image, seams);

    image = transpose(image);

    auto mat = f(image);
    auto seams = searchMinSeam(mat, (x - src.size[1]));
    image = enlargeCarving(image, seams);

    image = transpose(image);

    return image;
}
```

双向放大和双向缩小思路类似，不过要避免之前插入的能量线对之后的计算造成干扰，所以一次找出所有需要插入的能量线的位置。

区域保护和删除

```
void ObjectRemove(const cv::Mat image, function f)
{
    cv::Mat showImg = image.clone();
    cv::Mat remove = showImg.clone();
    for(int i = 0; i < image.rows; i++)
        for (int j = 0; j < image.cols; j++)
            remove.at<cv::Vec3b>(i, j) = cv::Vec3b(0, 0, 0);

    cv::Mat removeMask(image.rows, image.cols, CV_8U, cv::Scalar(0, 0, 0));
    cv::Mat remainMask(image.rows, image.cols, CV_8U, cv::Scalar(0, 0, 0));

    cv::Vec3b red(0, 0, 255);
    cv::Vec3b green(0, 255, 0);

    MouseArgs *args0 = new MouseArgs(showImg, removeMask, remove, green);
    MouseArgs *args1 = new MouseArgs(showImg, removeMask, remove, red);
    MouseArgs *args2 = new MouseArgs(showImg, remainMask, remove, green);

    cv::setMouseCallback("Remove & Remain", OnMouse, (void*)args0);
    while (1)
    {
        cv::imshow("Remove & Remain", args0->img);
        if (cv::waitKey(100) == 27)
```

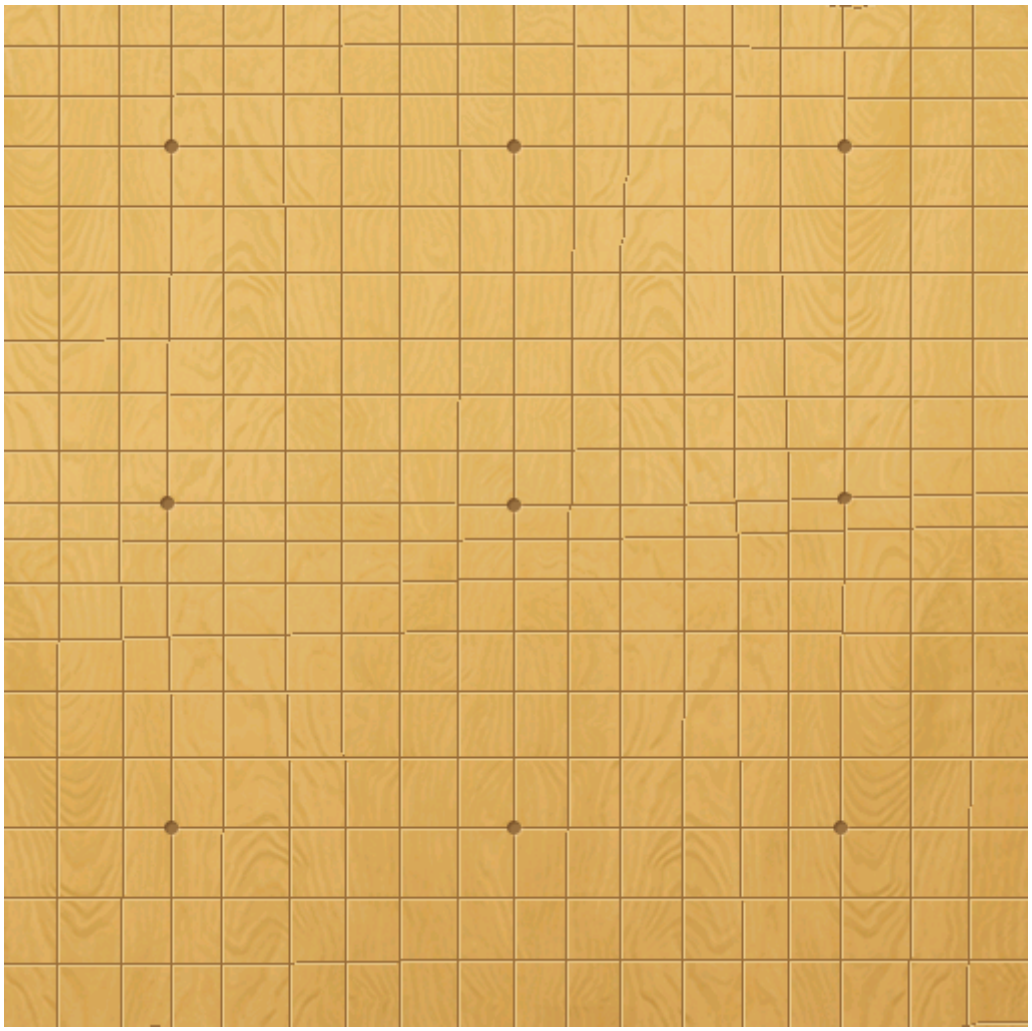
```
            break;
        }

        cv::setMouseCallback("Remove & Remain", OnMouse, (void*)args1);
        while (1)
        {
            cv::imshow("Remove & Remain", args1->img);
            if (cv::waitKey(100) == 27)
                break;
        }

        cv::setMouseCallback("Remove & Remain", OnMouse, (void*)args2);
        while (1)
        {
            cv::imshow("Remove & Remain", args2->img);
            if (cv::waitKey(100) == 27)
                break;
        }

        cv::setMouseCallback("Remove & Remain", NULL, NULL);
        cv::imwrite("remove.png", remove);
        cv::imshow("remove", remove);
        cv::waitKey(1);
        delete args1; args1 = NULL;
        delete args2; args2 = NULL;
    }
```

区域保护和删除就是将保护的部分能量值设为正无穷，删除的部分能量值设为负无穷。

实验结果

前六张图片均为缩小原图的25%左右

laplace算子



scharr算子

自己设计的算子

sobel算子

laplace算子

sobel算子

区域保护和删除样例

原图

sobel算子

seam图

双向放大样例

sobel算子

scharr算子