

Seam-Carving算法的JAVA实现

2024春季 数据结构与算法分析(B) 课程项目报告

Seam-Carving算法的JAVA实现

- 基本信息
- 问题描述
- 算法原理与实现

3.1 Shrink

- 像素能量计算
- 动态规划寻找最优路径
- 图片裁剪

3.2 Expand

3.3 Selected Area

- 选中区域保护
- 选中区域移除

3.4 GUI Construction

3.4.1 界面功能展示

3.4.2 合理性分析

4. 运行结果分析

4.1 Shrink

4.2 Expand

4.3 Selected Area

- 选中区域保护
- 选中区域删除

4.4 运行过程分析

5. 程序运行指南

5.1 代码架构

5.2 运行注意事项

1. 基本信息

Member	Student ID	Lab Session	Contribution Rate
许耀锦	12112122	Tuesday 7-8 Yang	33.3%
高赫廷	12210816	Tuesday 7-8 Yang	33.3%
李怡萱	12212959	Tuesday 7-8 Yang	33.3%

Contribution of work

许耀锦：Shrink功能设计, Expand功能设计

高赫廷：Selected Area相关功能设计

李怡萱：GUI 建立

2. 问题描述

在本项目中，我们需要通过JAVA语言来实现Seam-Carving（切割缝隙）算法，使程序功能实现在对图片进行缩小或扩张的同时，保留图片中重要内容；另外，还需要提供能让用户手动选择相关区域保留或删除选定内容的功能；最后，我们还需要建立一个GUI用户界面，使用户可以在界面上对相应的图片进行操作。

根据题目要求，为实现图片操作的算法，我们需要通过对像素能量值的计算，通过排序算法实现对所切割（或复制）的缝隙的最终选择，来实现图片的缩小（Shrink）、放大（Expand）效果。

综合以上的思路和要求，我们设计并完成了我们的项目，实现的功能如下：

- 图片缩小功能（Shrink）：可以通过输入缩小数值或比例，来实现对导入图片的缩小，同时尽量保证图中重点区域不发生改变。
- 图片放大功能（Expand）：可以通过输入放大数值或比例，来实现对导入图片的扩张，同时尽量保证图中重点区域不发生改变。
- 选定区域功能（Selected Area）
 - 保护选定区域的缩小图片功能（Select to Keep）：可以在用户选择特定区域后，输入缩小数值或比例，来实现对导入图片的缩小，同时保证图中选择区域不发生改变。
 - 删除选定区域的缩小图片功能（Select to Remove）：可以在用户选择特定区域后，输入缩小数值或比例，来实现对导入图片的缩小，同时尽量让选定区域被移除。
- 建立一个合理的GUI用户界面：在用户界面中划分出图片操作区域和行为按钮区域，可以支持对图片的导入、放缩、选定、重做、导出操作，同时提供图片显示大小尺度调整功能，并设置图片大小显示标签和图片操作状态来增进用户的使用体验。

在算法原理和结果分析部分，我们将详细阐述各功能的具体实现算法与功能执行效果。

3. 算法原理与实现

3.1 Shrink

3.1.1 像素能量计算

图片shrink的原理，是通过计算图片的x或y方向的最小能量路径来确定图片中最不重要的部分，然后沿该最小路径删除像素，使图片x或y方向的尺寸减少且尽可能保留原图重要信息。

每个非边缘像素的能量由其相邻四个像素的RGB值决定，能量的计算公式如下：

$$E(x, y) = \sqrt{\Delta_x^2(x, y) + \Delta_y^2(x, y)} \quad (2)$$

$$\Delta_x^2(x, y) = R_x(x, y)^2 + G_x(x, y)^2 + B_x(x, y)^2 \quad (3)$$

$$R_x(x, y) = R(x+1, y) - R(x-1, y) \quad (4)$$

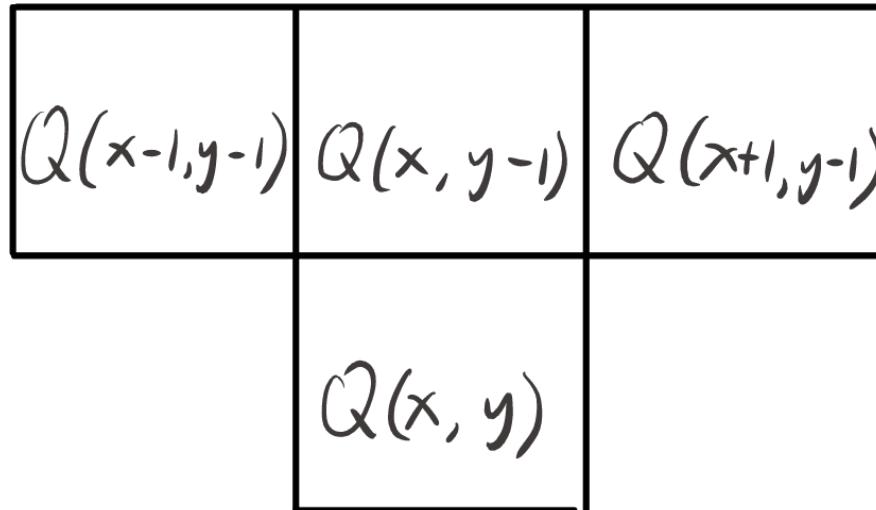
其中， $R(x, y)$ 表示像素 (x, y) 的R通道数值，同理可计算 $\Delta_y^2(x, y)$ 。

对于边缘像素的能量，我们统一赋值常数1000。

3.1.2 动态规划寻找最优路径

从起点计算每个像素点的能量，计算出所有可能路径，最后将所有路径进行排序以选出能量最小的路径时间和空间成本都十分昂贵，因此我们应该寻求更优的路径计算方式。

寻找最小能量路径是一个优化问题，我们可以将每个像素点的能量重新定义为“前面已选择能量最小路径的能量之和加上当前像素的能量”，于是在计算每行像素的能量时，每个像素点的能量只与其前一行最近的三个像素点的能量有关，关系如下图：



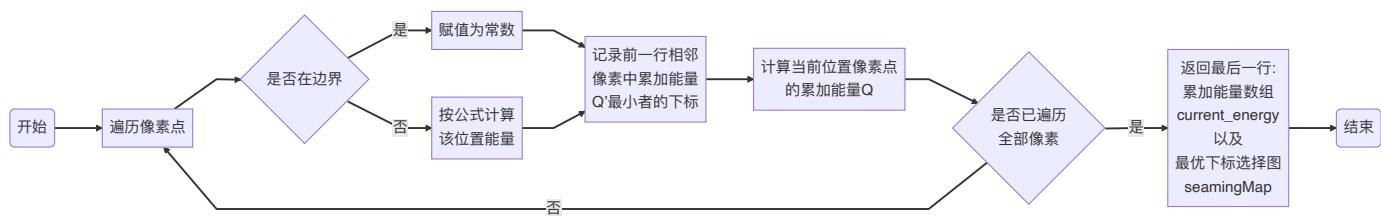
其中当前像素点 $Q(x, y)$ 的计算方式：

$$Q(x, y) = \min\{Q(x - 1, y - 1), Q(x, y - 1), Q(x + 1, y - 1)\} + E(x, y) \quad (5)$$

于是在计算行方向的像素能量时，只需要维护两个double类型的数组，在代码中分别为 `pre_energy`（代表前一行的能量Q的集合）和 `current_energy`（当前行的能量Q的集合）。每个像素点相关的变量除了对应的路径能量总和Q，还有前一行最近的三个像素中Q最小的像素所对应的下标（储存在二维数组 `seamingMap` 中）。

需要寻找最小能量路径时，只需要在最后一行的能量计算完成后，对 `current_energy` 数组进行排序，获取最小的Q值对应的下标，并按照 `seamingMap` 中存储的下标进行自下而上的搜索即可。

以下是对每行像素能量计算方法 `calculateRowEnergy()` 的流程图，每列像素能量的计算方法 `calculateColumnEnergy()` 同理。



上述方法返回的两个参数 `current_energy` 和 `seamingMap` 储存在一个自定义类的实例 `result` 中。

在获得 `result` 之后，使用Insertion Sort对最后一行的能量总和Q进行由小到大排序，排序完成后按照第一位像素对应记录的下标在 `seamingMap` 中进行自下而上的读取，即可获得能量最小路径对应的每行像素的坐标。

`findMinPath`方法根据最后一行能量总和的数组，寻找最小路径对应的每行像素下标，并储存在 `seamMap` 中返回：

```

1 private int[] findMinPath(double[] energy, int[][] map, int length) {
2     int[] seamMap = new int[length];
3     int minIndex = (int) selectPath(energy, map[0])[1];
4     for (int i = length - 1; i >= 0; i--) {
5         seamMap[i] = minIndex;
6         minIndex = map[i][minIndex];
7     }
8     return seamMap;
9 }
```

其中 `minIndex` 用于记录 `seamMap` 中储存的下标，并根据此下标进行逐行向上的搜索，将最终得到的路径储存 在 `seamMap` 中返回。

3.1.3 图片裁剪

`seamMap` 中储存了最小能量的路径信息，按沿该路径裁剪后将得到的新图片的尺寸创建空白图片，并将原图片每个像素的RGB信息复制到空白图片，并跳过 `seamMap` 中所记录的像素，即可得到裁剪一次之后的图片。用新图片重复执行上述能量计算与路径寻优步骤并裁剪，迭代直到图片尺寸达到目标尺寸。

3.2 Expand

对图片进行expand操作的原理和shrink大致相同，首先仍需要进行能量计算，然后根据得到的最小能量路径进行像素的复制扩张。

在shrink的部分我们每次计算能量后只对能量最小的接缝进行裁切，然而在expand操作中，如果采用和shrink相同的策略每次扩张一条接缝后重新计算能量，由于上一条扩张的接缝的RGB值是从相邻像素中复制获取的，新的能量最小接缝又会落到刚刚扩张的接缝上，以此扩张的图片会出现像素重叠痕迹明显的问题。

为了避免该问题的出现，在expand方法中我们采取计算一次能量后选取最小的n条路径进行扩张的策略。操作仍是在计算完能量和储存好 `seamingMap` 变量后，对最后一层的像素使用Insertion Sort排序，根据裁剪尺寸的需求选取前n个像素进行向上搜索，即可得到n条能量最小的路径。

在扩张方法中，根据 `seamMaps` 记录的像素下标，将最小能量接缝的像素RGB值向右复制一份，不同接缝中有相同的像素下标则根据重复的次数复制多份，以此完成原图片的扩张。

3.3 Selected Area

3.3.1 选中区域保护

对于保护选中的区域使得该区域不被算法移除，我们采用了增大该区域像素能量的办法。由于对于不同的图片，能量分布情况以及能量大小可能会有显著不同，直接对区域能量进行数值改变的操作稳定性较差。因此，我们采用在计算能量的过程中进行干预以达到较好的效果。

在Shrink中我们提及：“每个像素点的能量只与其前一行最近的三个像素点的能量有关”，“每个像素点的能量重新定义为‘前面已选择能量最小路径的能量之和加上当前像素的能量’”。于是，路径进入框选区域要计算对应像素时，我们将其前一行最近的三个像素点的能量进行数乘并且加上一个极大的数，如下：

$$pre_energy[x] = 10 * pre_energy[x] + 100000000; \quad (6)$$

$$pre_energy[x - 1] = 10 * pre_energy[x - 1] + 100000000; \quad (7)$$

$$pre_energy[x + 1] = 10 * pre_energy[x + 1] + 100000000; \quad (8)$$

随后，路径的每一次像素延伸都需要进行该操作，于是，路径的能量经过了很多次加上极大的数同时数乘，在减少原能量影响的情况下，极大的增加了所选区域的能量，避免选择穿过框选区域的路径。

此外，我们在每次SeamCarving后重新修正框选区域，防止被已去除的像素干扰行列的判断。

3.3.2 选中区域移除

对于使选中的区域更容易被算法移除，我们采用了特殊的算法：我们正常计算照片的能量，随后我们将选中的区域的像素能量从小到大依次推入栈，之后再从大到小弹回对应位置。

代码如下：

```
1 int[] sortedArray = stack.stream().mapToInt(i -> i).toArray();
2     Arrays.sort(sortedArray);
3     stack.clear();
4     for (int num : sortedArray) {
5         stack.push(num);
6     }
7     for (int x = topLeft.x; x <= bottomRight.x; x++) {
8         for (int y = topLeft.y; y <= bottomRight.y; y++) {
9             if (topLeft.x <= y && bottomRight.x >= y && topLeft.y <= x &&
bottomRight.y >= x) {
10                 newSeamingMap[x][y] = stack.pop();
11             }
12         }
13     }
```

该操作的思路是选中的区域如果大致图形无明显变化，能量值呈现均匀态势，即使像素能量进行了更换，也不会出现路径集中向某区域聚集或避开的现象，不会对消除进行干扰；在选中的区域存在物体或者复杂图形即能量值有明显变大时，更换像素能量能够倾向于先消除对应的物体以及复杂图形，符合使用者的意愿。

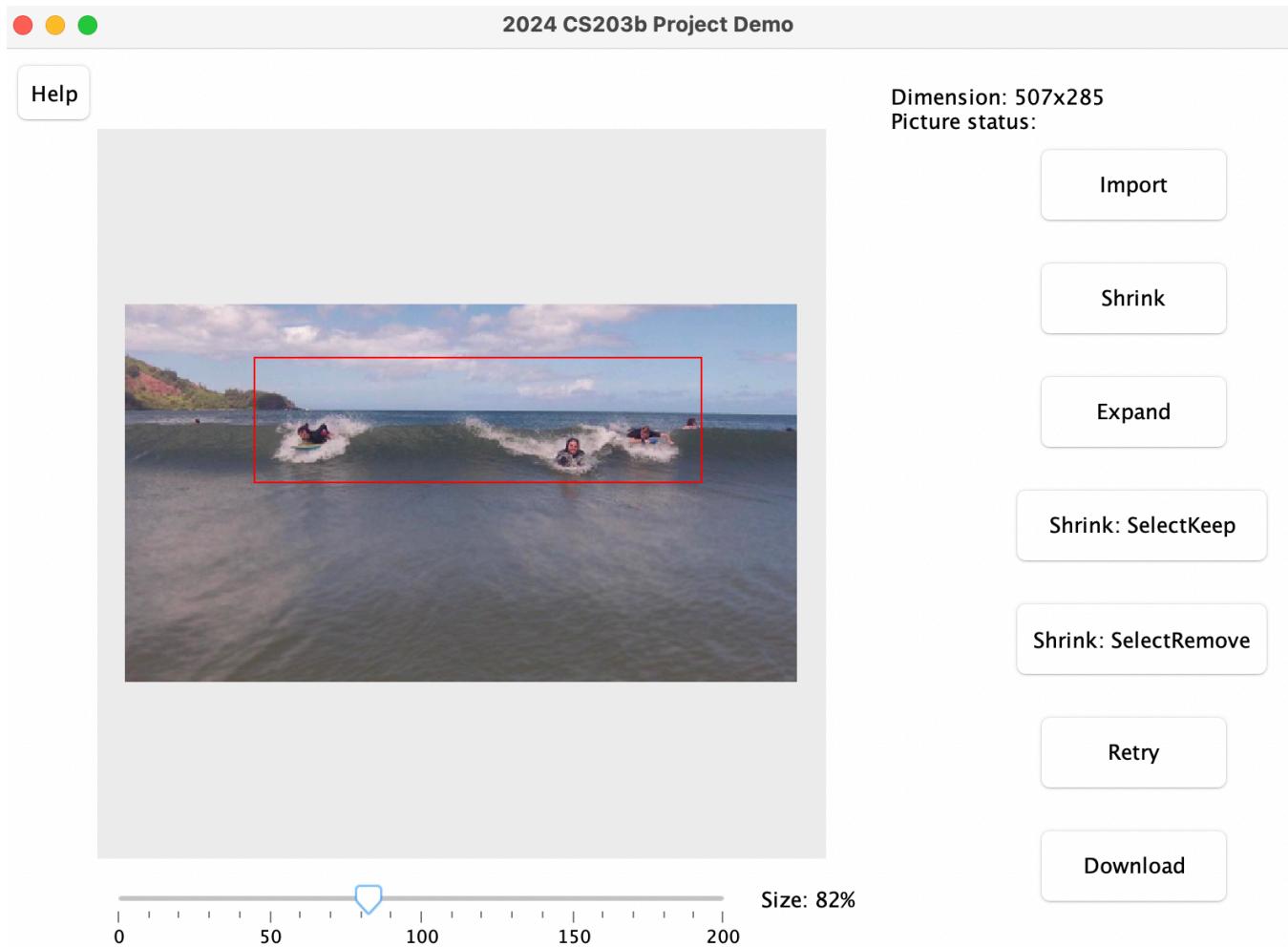
除此之外，我们以从边缘到中心指数衰减的形式调整能量，以便路径能够更容易通过这片区域且通过的相对平滑，不至于出现大块断层。代码如下：

```
1 double distance = Math.sqrt(Math.pow(x - centerX, 2) + Math.pow(y - centerY, 2));
2 double decayFactor = Math.exp(-(distance / maxDistance));
3 pre_energy[x] = pre_energy[x] * decayFactor;
```

3.4 GUI Construction

3.4.1 界面功能展示

我们的程序用户界面如下：



程序包含的元素及支持的功能如下：

- `JFrame` 类型的图像操作界面，包含1个 `JPanel` 类型的图片操作区域，左上和右侧8个 `JButton` 类型的按钮，1个 `JSlider` 类型的进度条，3个 `JLabel` 类型的显示标签。
- `Import` 和 `Download` 按钮可以将系统中指定路径的图片文件导入至操作区域中，并支持将图片导出到系统中指定路径的功能。
- `Shrink`、`Expand`、`Shrink: SelectKeep`、`Shrink: SelectRemove`、`Retry`、`Help` 按钮分别可以实现对操作区域中图片的缩小、放大、保护选择区域的缩小、删除选择区域的缩小、恢复初始图片、查看操作指南的功能。
- 在导入图片后，可以在左侧灰色操作区域中，通过长按拖动鼠标实现对图片某区域的框选；同时可以在下方拖动进度条调整图片在灰色操作区域的显示比例（不改变图片原始尺寸）。
- 在右侧 `Import` 按钮上方，标签 `Dimension` 可以显示当前操作图片的长宽尺寸；标签 `Picture status` 可以在用户执行图片缩放按钮操作后显示图片是否已操作完成的状态。

3.4.2 合理性分析

1. 此GUI界面中，图片操作区域和功能按钮分位于界面两侧，区域划分合理。
2. 用户可以在GUI中自由指定路径以导入或导出图片，并通过进度条缩放图片显示比例，同时框选特定区域的操作简单，符合用户的使用习惯。
3. 在进行缩放功能操作时，点击相应按钮后会有缩放模式选择、输入数据（比例）文本框及提示，操作步骤清晰；同时缩放操作支持输入数据、比例的缩放方式，为用户提供多元化的缩放选择。

4. 运行结果分析

4.1 Shrink

对一张508*285的图片进行横向裁剪，使其宽度减少150个像素单位的效果如下：



左图：缩小前



右图：缩小后

可见裁剪后的图片保存了原图的重要信息且没有明显的裁切痕迹，由此可见我们的接缝裁剪功能正常工作。

4.2 Expand

对尺寸为508*285的测试图片进行纵向扩张，使其高度增加100个像素单位的效果如下：



左图：放大前



右图：放大后

可以观察到新图片的右侧仍很大程度上保留了原图的特征，而中间和左侧出现像素重复的情况是由于100条最小能量接缝的主要区别集中在后半部分的像素能量上，前半部分的路径重合，如下图所示：



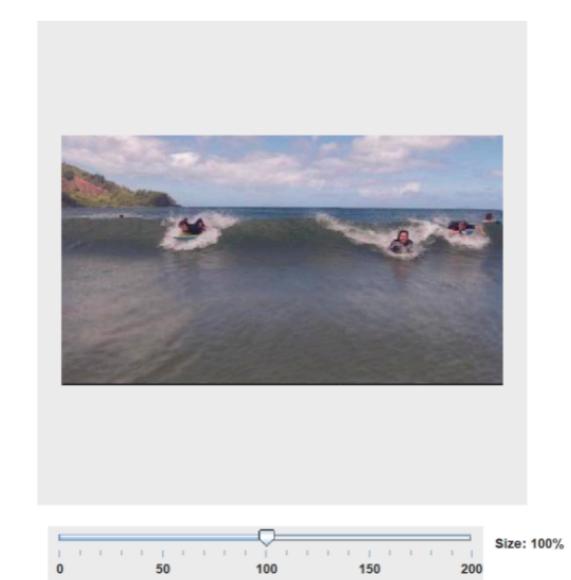
4.3 Selected Area

4.3.1 选中区域保护

我们以一张508*285像素的图片为例，我们先用鼠标选定一个区域（选择了能量值比较小的海水部分，本应容易被移除），其次再点选 Shrink: SelectKeep 按钮，使这个区域不容易被移除。接着将像素高度和宽度的缩放比例都定为缩小到原来的0.8倍，操作后得到的结果如下：



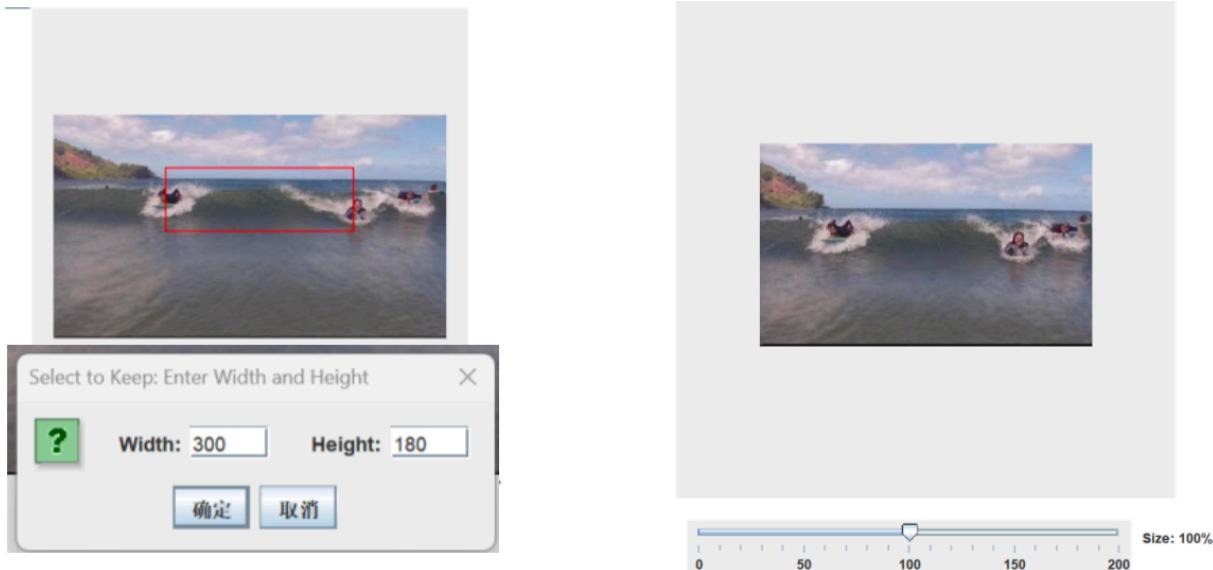
左图：缩小前



右图：缩小后

由图可见，选定的海水区域宽度几乎保持不变，选定区域成功被保留，没有被删除像素。

为了使结果更有说服力，我们在此基础上继续操作。这时将图片缩小至300*180像素，结果如图所示：



左图：缩小前

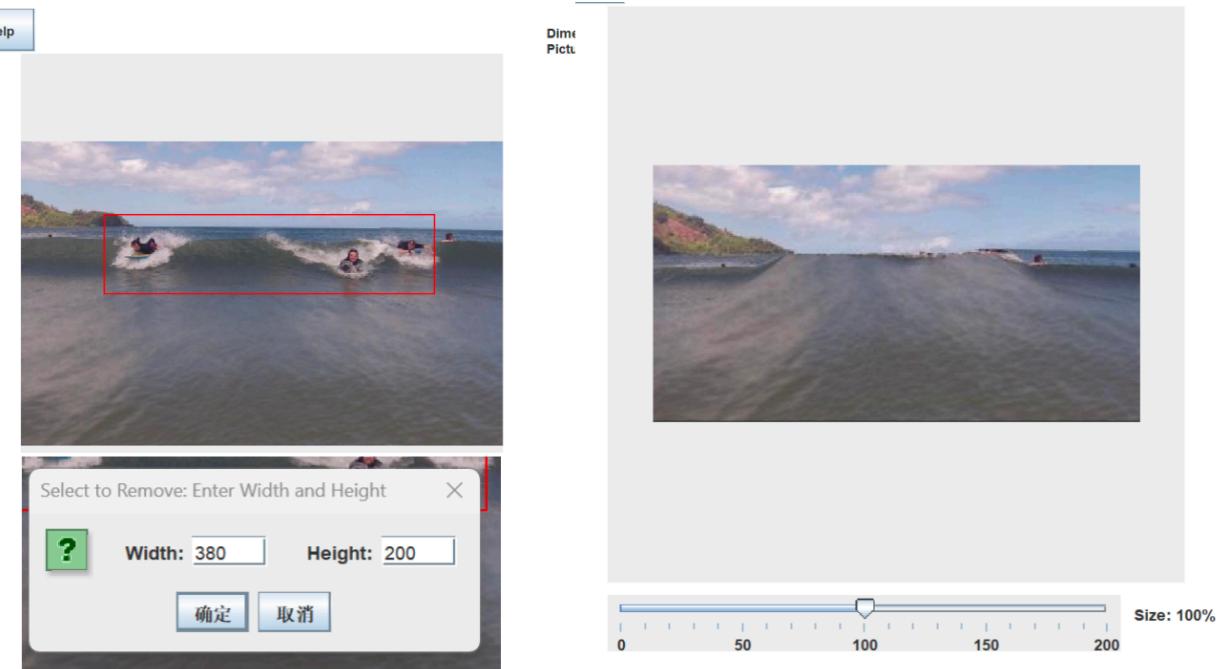
右图：缩小后

由图可见，选定的海水区域也没有被删除像素，与自动删除的功能形成对比。同时选定区域之外呈现很好的 SeamCarving 效果，故证明该功能成功实现。

4.3.2 选中区域删除

我们同样以两组 508*285 像素的图片为例进行演示。

我们先用鼠标选定图像中的某区域（选择了能量值比较大的人部分，本应容易被保留）；接着再点选 `shrink: SelectRemove` 按钮，使这个区域更容易被移除；接着输入像素高度和宽度将缩放到的数值（380*200 像素），操作后得到的结果如下图组 1 所示：



左图 1：缩小前

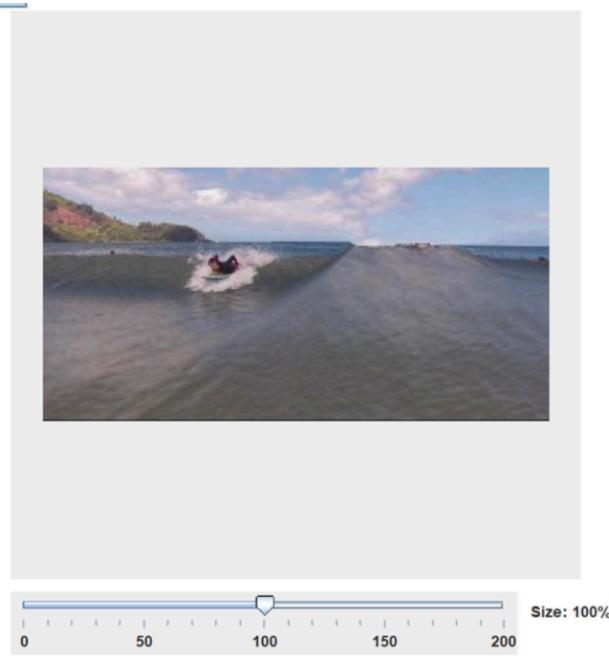
右图 1：缩小后

由图可见，原本能量值较高的人所在的区域，成功根据人为的选择被移除。

接着以原先的图片为例，以同样的方式选定右半部分的人的区域，再点选 shrink: SelectRemove 按钮，输入数值将图片缩小至400*200像素，操作结果如图组2所示：



左图 2: 缩小前



右图 2: 缩小后

由结果可见，右侧的人成功被移除，同时左侧的人得以保留，与先前图组1形成对比。可由此得出结论：这两组图片都呈现较好的优先删除效果，该功能成功实现。

4.4 运行过程分析

1. 操作时间分析

- 在对图片进行各操作的运行时间方面，shrink功能所需时间随图片尺寸的增大和需要裁剪的像素数量的增多而显著增长。在原图尺寸一致，修改比例一致的情况下，expand操作所消耗的时间远远少于shrink操作。
- 造成时间复杂度差距悬殊的原因是两种操作的执行逻辑不同，shrink操作每次裁剪1像素宽度的接缝，每裁剪一次就要重新计算图片的能量并寻找下一条接缝，完成单次能量计算的时间复杂度为 $O(n^2)$ ；而expand操作只会对图片进行一次能量计算，因此在时间花费上expand会比shrink短很多。而选中区域保护及删除是在shrink的基础上运行的，通常选中区域在整张图片的占比较小，因此与shrink的时间花费不会有较大差距。

2. 功能效果分析

- 在缩放效果方面，shrink操作的效果比expand操作的效果更好。
 - 由于shrink每次只裁剪一条能量最低的接缝，随后重新计算能量以寻找新的接缝，使得每次裁剪的接缝不会有重叠的部分，裁剪的效果较好。
 - 而expand操作是一次性扩展n条能量最低接缝，由于这些接缝有部分重叠，扩展的时候会将重叠的部分重复扩展，导致生成的图像失真，效果较差。
- 在选定区域进行特殊操作方面，选中区域保护的效果比选中区域删除的效果更好。

- 对于选中区域保护，由于选中区域的能量经过操作后与整张图片剩余区域的能量相差巨大，接缝触碰该区域便会使能量急剧增长，因此很难选择穿过该片区域，保证了保护的效果。
- 而对于选中区域删除，即使我们采用从边缘到中心使计算的能量值逐渐下降的原则来优先删除该区域，也难免有时遇到“能量断层”现象，即在人为控制能量的情况下多条接缝集中在某区域出现形成图像的断层，导致生成的图像失真，效果较差。同时，如果选中区域横纵平行方向有想要保留的物体，由于接缝向这里聚拢，因此有时会误删除横纵平行处的较为重要的像素。

3. 未来优化方向

对于程序的进一步优化，我们提出以下方向：

- 优化shrink操作算法的时间复杂度，尝试每次重新计算能量时只重新计算受接缝影响的像素的能量，去除对能量不发生改变的像素点能量的冗余计算。
- 优化expand操作算法的扩展效果，尝试对寻找的接缝进行筛选，找出重叠度高度接缝，计算新的重叠度接缝代替之。
- 优化选定区域删除的算法的灵活性，尝试用更精确普适的方法对框选中的能量高的区域赋予更高的权重进行删除同时避免出现“能量断层”现象，尝试用更合理的办法对接缝向选中区域“吸引”。

5. 程序运行指南

为保证用户可以正常运行我们的代码并实现相关功能，下面附上程序运行指南。为最大程度展示我们设计的程序功能效果，请在运行前阅读此说明，或在运行程序时点击GUI界面中的“Help”按钮进行使用说明查询，并在操作图像时尽可能按照界面中的提示来运行程序。

5.1 代码架构

我们的项目代码架构如下：

```

1 .Code
2   └── bin
3   └── img
4   └── lib
5   └── out
6   └── src
7     └── App
8     └── GUI
9     └── ImageArea
10    └── myTool
11    └── SeamCarving

```

在运行时，点击 `src` 文件夹中的 `App.class` 文件中的运行按钮即可。

5.2 运行注意事项

1. 在Shrink、Expand、SelectKeep、SelectRemove时，点击后若出现Mode Selection文本框，请选择依据比例（ratio）或依据数值（value）进行图片的修改操作（留意提示弹窗）：
 - 如果选择比例，选择缩小模式，您输入的ratio必须位于(0,1)间；选择放大模式，您输入的ratio必须大于1。
 - 如果选择数值，您输入的value必须合乎逻辑：缩小模式下，输入的长宽value要小于原数值；放大模式下，输入的长宽value要大于原数值。
2. 您可以在导入图片后长按图片中某一点拖出一个红色矩形框，这个框即为选择区域（Selected Area）。如果要使用SelectKeep或SelectRemove模式，请确保您在点击SelectKeep、SelectRemove按钮前已经选定相应区域（请留意提示弹窗），再进行后续操作。
3. 为保证缩放效果，请尽可能利用进度条将图片显示尺寸调整到全部位于操作区域以内，并在进行区域选择时将区域选取在图片范围内，再进行选定区域的缩放操作。
4. 缩小功能可能相对耗时较长，请随时留意 Picture status: 标签的提示；如果出现"Updated"则可见图像操作完成，同时在操作区域也会同步更新操作后的图像。
5. 如果遇到无法操作的报错，请尝试点击 Retry 按钮或者 Import 按钮重新操作图片。