作业1

算法：

采用讲义上的延长-规整算法：假设0 <= k < 1，则先将端点的x值round，然后计算直线在 round(x) 处对应y值，对该y值取整得到规整后的端点。

规整函数：Line RoundEndPoints(Line l); 位于MyController.cpp

算法的合理性：

该算法将一条线段的两个端点规整后得到另一条线段。仍然假设0 <=k < 1，首先对于端点的x值round，以使新线段的端点不至于离未归整的端点过远。在这个 round(x) 值处，对应2个候选的y值，此时选择离直线较近的y值作为最终归整结果。

下面讨论归整点的一些性质：（以下都基于0 <= k < 1）

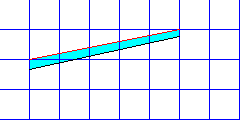
首先，应用上面的归整算法归整后的x值，有原直线的两个y值和归整后的2个y值，由此得到4个点。上述归整采用的点使得这4点所围成的梯形面积最小。

证明：设归整后起点，终点的x值分别为xs和xe，原直线在这两点的y坐标分别为y(xs)和y(xe)，规整后分别为ys和ye，则该梯形的面积为：S = (|ys-y(xs)|+|ye-y(xe)|)\*|xs-xe| / 2，xs-xe不变，而ys和ye分别是离原直线最近的整数y值，故S最小。

不做归整直接四舍五入x和y值得缺点在于：对于x采用了上述相同的策略，但对于y则不能保证四舍五入后的值是离直线最近的，事实上这取决于直线的斜率。而上述方法考虑了这一点。

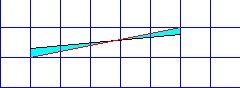
但上述得到结果仍然不是最优解。理论上最优解可以通过实验两个端点的4种情况（此处首先接受对于x的归整），然后对于每种情况计算光栅化点的总误差，取最小的得到。但这样的讨论较为麻烦。

注意到一般情况下，如果归整后的线段上的点离原线段较近，得到的线段较佳。这事实上就是比较夹于两线段之间的梯形或三角形面积。当中间所夹为梯形时，



面积为：S = (|ys-y(xs)|+|ye-y(xe)|)\*|xs-xe| / 2，

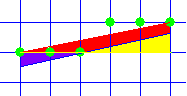
而中间所夹为两个三角形时，



面积为：S = (|ys-y(xs)|\*|xs-xi| + |ye-y(xe)|\*|xi-xe|)/2，其中xi为原直线与归整后直线的交点的x值（不必是整数）。

这样的计算，虽然比逐点计算距离高效很多，但仍然需要讨论4种情况，还需要计算交点，对于一个归整算法来说依旧太显复杂，故最终没有采用。但我觉得若对于上述面积算法作进一步的讨论，得到较为简单的算式，可能能够优于现在算法。但限于时间所限，没有深入讨论。

然而作业实现的算法已经有一些很好的性质：作业实现的算法（而非后面提到的面积判断法）的优点在于计算简单，而效果在一定范围内也可以接受，且最终直线的斜率也更接近于原直线（简要说明如下：对于本归整算法关于原线段所围的图形是梯形，而面积法围得两个三角形的情况下，显然作业实现的算法斜率更接近；而对于面积法，不可能得到另一个梯形使得面积小于作业实现的算法（前面已证）；对于作业实现的算法关于原直线已经围成两个三角形的情况，两种算法得到的结果一致）。下面是作业实现的算法和面积最小判别法产生分歧的一个例子：



采用作业实现的算法得到红色线段（红色区域上沿），而面积最小判别的得到黄色线段（水平的），因为紫色区域和黄色区域面积之和小于紫色与红色面积之和。绿色点为作业实现的算法得到的光栅化点，而面积法的光栅化点位水平，此处没有画出。但光栅化点的距离和上看，确实是面积法更小。