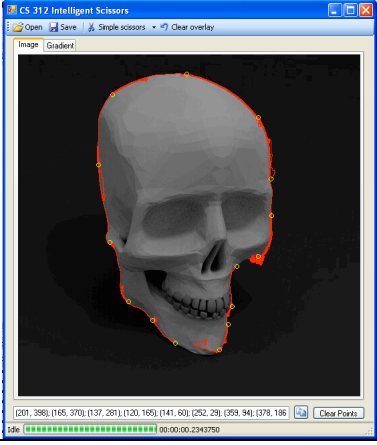
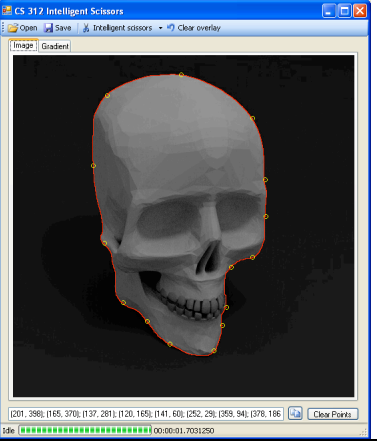
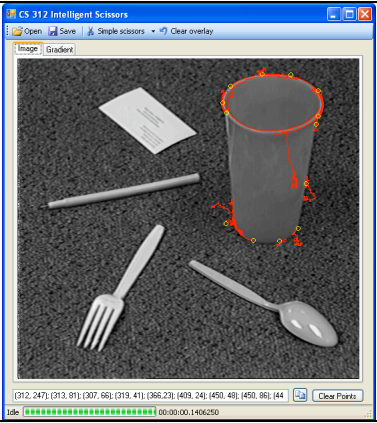
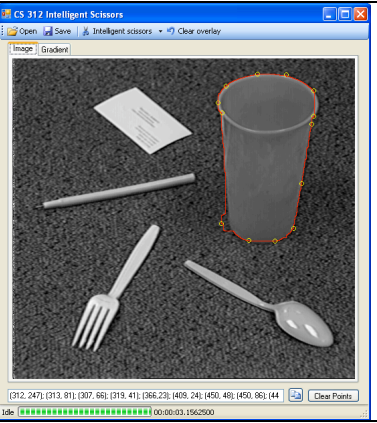




Project 4 - Intelligent Scissors

Duane Johnson

CS 312 / Section 1

1. Screenshots (zoom in to see detail)

Sample	Simple Scissors	Intelligent Scissors
skull-ascii-entire-skull		
Time:	0.234 seconds	1.703 seconds
things-ascii-cup		
Time:	0.140 seconds	3.156 seconds
xray-ascii-not-so-easy		
Time:	0.047 seconds	0.406 seconds

2. The precision of the greedy algorithm is really quite awful--in some cases, it doesn't even connect the segments in a complete loop because of the possibility of an infinite loop. On the other hand, Dijkstra's algorithm takes at least an order of magnitude more time to complete. Among the sample cases shown above, the cup example is 20 times slower. However, the result is much cleaner and probably an outcome that the user might accept.

3. One way that may prove helpful in speeding this algorithm up is to use a heuristic that tries to "guess" which direction to go, rather than esteeming all directions as equal. In such a scenario, it would be possible to urge the algorithm to head towards the next point in the path, hoping that the shortest path can be attained that way. Of course, it would then be less accurate again since we could not guarantee that the path attained is the shortest. Nevertheless, it may be sufficient for most needs.

4. Got it! Under 7 seconds in each case.

5. Code Listing:

```
public class DijkstraScissors : Scissors
{
    Pen yellowpen = new Pen(Color.Yellow);

    public DijkstraScissors() { }

    private int priority(int p)
    {
        return int.MaxValue - p;
    }

    private void consider(
        HashSet<Point> visited,
        PriorityQueue<LinkedPoint, int> pq,
        PriorityQueueItem<LinkedPoint, int> parent,
        Point p)
    {
        if (!visited.Contains(p))
        {
            visited.Add(p);
            pq.Enqueue(new LinkedPoint(p, parent.Value), priority(GetPixelWeight(p) + priority(parent.Priority)));
        }
    }

    /// <summary>
    /// this is the class you implement in CS 312.
    /// </summary>
    /// <param name="points">these are the segmentation points from the pgm file.</param>
    /// <param name="pen">this is a pen you can use to draw on the overlay</param>
    public override void FindSegmentation(ICollection<Point> points, Pen pen)
    {
        if (Image == null) throw new InvalidOperationException("Set Image property first.");
        // this is the entry point for this class when the button is clicked
        // to do the image segmentation with intelligent scissors.
        Program.MainForm.RefreshImage();
    }
}
```

```

using (Graphics g = Graphics.FromImage(Overlay))
{
    for (int i = 0; i < points.Count; i++)
    {
        Point start = points[i];
        Point end = points[(i + 1) % points.Count];

        Point here = new Point(start.X, start.Y);
        Bitmap sobel = Image.Gradient.Bitmap;

        PriorityQueue<LinkedPoint,int> pq = new PriorityQueue<LinkedPoint,int>();
        HashSet<Point> visited = new HashSet<Point>();

        // Draw a circle around the point we're exploring
        g.DrawEllipse(yellowpen, start.X - (float)3.0, start.Y - (float)3.0, 6, 6);
        g.DrawEllipse(yellowpen, end.X - (float)3.0, end.Y - (float)3.0, 6, 6);
        Program.MainForm.RefreshImage();

        // Start with the first point
        pq.Enqueue(new LinkedPoint(start), priority(0));

        int counter = 0;
        LinkedPoint lp = null;
        while (pq.Count > 0)
        {
            PriorityQueueItem<LinkedPoint,int> item = pq.Dequeue();
            lp = item.Value;

            // Are we there yet?
            if (lp.point.X == end.X && lp.point.Y == end.Y) break;
            // Explore successors of this point
            if (lp.point.Y > 1)
                consider(visited, pq, item, new Point(lp.point.X, lp.point.Y - 1));
            if (lp.point.Y < sobel.Height - 2)
                consider(visited, pq, item, new Point(lp.point.X, lp.point.Y + 1));
            if (lp.point.X > 1)
                consider(visited, pq, item, new Point(lp.point.X - 1, lp.point.Y));
            if (lp.point.X < sobel.Width - 2)
                consider(visited, pq, item, new Point(lp.point.X + 1, lp.point.Y));
        }

        while (lp != null)
        {
            Overlay.SetPixel(lp.point.X, lp.point.Y, Color.Red);
            lp = lp.prev;
        }
    }
}
}

```

Helper Classes:

```
public class LinkedPoint
{
    public Point point;
    public LinkedPoint prev;

    public LinkedPoint(Point p)
    {
        point = p;
        prev = null;
    }

    public LinkedPoint(Point p, LinkedPoint prv)
    {
        point = p;
        prev = prv;
    }
}

public struct PriorityQueueItem<TValue, TPriority>
{
    private TValue value;
    private TPriority priority;

    public PriorityQueueItem(TValue val, TPriority pri)
    {
        this.value = val;
        this.priority = pri;
    }
    // ... other priority queue methods ...
}

public class PriorityQueue<TValue, TPriority> : ICollection,
    IEnumerable<PriorityQueueItem<TValue, TPriority>>
{
    private PriorityQueueItem<TValue, TPriority>[] items;
    // ... other priority queue methods ...
}
```