Convex Hull

Thursday, October 5, 2017 11:11 PM

1. Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Linq;
using _1_convex_hull;

namespace _2_convex_hull
{
   class ConvexHullSolver
   {
      System.Drawing.Graphics g;
      System.Windows.Forms.PictureBox pictureBoxView;

      public ConvexHullSolver(System.Drawing.Graphics g, System.Windows.Forms.PictureBox pictureBoxView)
      {
         this.g = g;
         this.pictureBoxView = pictureBoxView;
      }

      public void Refresh()
      {
         // Use this especially for debugging and whenever you want to see what you have drawn so far
         pictureBoxView.Refresh();
      }

      public void Pause(int milliseconds)
      {
         // Use this especially for debugging and to animate your algorithm slowly
         pictureBoxView.Refresh();
         System.Threading.Thread.Sleep(milliseconds);
      }

      public void Solve(List<System.Drawing.PointF> pointList)
      {
         List<System.Drawing.PointF> sortedPoints = pointList.OrderBy(o => o.X).ToList();
         ConvexHull hull = DivideAndConquer(sortedPoints);
         Draw(hull);
      }

      private ConvexHull DivideAndConquer(List<System.Drawing.PointF> p_pointList)
      {
         if (p_pointList.Count == 2)
         {
            MakeClockwise(p_pointList);
            return new ConvexHull(p_pointList);
         }
         else if (p_pointList.Count == 3)
         {
            MakeClockwise(p_pointList);
            return new ConvexHull(p_pointList);
         }
         else
         {
            int midpoint;
            ConvexHull leftHull;
            ConvexHull rightHull;

            if (p_pointList.Count % 2 == 1)
```

```
        {
            midpoint = (p_pointList.Count / 2) + 1;
            leftHull = DivideAndConquer(p_pointList.GetRange(0, midpoint));
            rightHull = DivideAndConquer(p_pointList.GetRange(midpoint, midpoint-1));
        }
        else
        {
            midpoint = p_pointList.Count / 2;
            leftHull = DivideAndConquer(p_pointList.GetRange(0, midpoint));
            rightHull = DivideAndConquer(p_pointList.GetRange(midpoint, midpoint));
        }

        return Merge(leftHull, rightHull);
    }
}

private ConvexHull Merge(ConvexHull p_left, ConvexHull p_right)
{
    FindUpperTangent(p_left, p_right);
    FindLowerTangent(p_left, p_right);
    List<PointF> newHullPoints = new List<PointF>();

    List<PointF> leftList = p_left.Points;
    int leftStartIndex = leftList.IndexOf(p_left.UpperTangentPoint);
    int leftIndex = leftStartIndex;
    int leftEndIndex = leftList.IndexOf(p_left.LowerTangentPoint);

    if (leftList.Count < 4)
    {
        while (leftIndex != leftEndIndex)
        {
            newHullPoints.Add(leftList[leftIndex]);
            leftIndex--;
            if (leftIndex < 0)
            {
                leftIndex = leftList.Count - 1;
            }
        }
        newHullPoints.Add(leftList[leftEndIndex]);
    }
    else
    {
        for (int i = 0; i <= leftEndIndex; i++)
        {
            newHullPoints.Add(leftList[i]);
        }
        for (int i = leftStartIndex; i < leftList.Count; i++)
        {
            //newHullPoints.Add(leftList[leftIndex]);
        }
    }

    List<PointF> rightList = p_right.Points;
    int rightStartIndex = rightList.IndexOf(p_right.LowerTangentPoint);
    int rightIndex = rightStartIndex;
    int rightEndIndex = rightList.IndexOf(p_right.UpperTangentPoint);

    if (rightList.Count < 4)
    {
        while (rightIndex != rightEndIndex)
        {
            newHullPoints.Add(rightList[rightIndex]);
            rightIndex--;
            if (rightIndex < 0)
            {
```

```
                rightIndex = rightList.Count - 1;
            }
        }
        newHullPoints.Add(rightList[rightEndIndex]);
    }


    return new ConvexHull(newHullPoints);
}

private void FindUpperTangent(ConvexHull p_left, ConvexHull p_right)
{
    Boolean leftPointChange = true;
    Boolean rightPointChange = true;
    List<PointF> leftList = p_left.Points;
    List<PointF> rightList = p_right.Points;
    PointF leftPoint = p_left.Rightmost;
    PointF rightPoint = p_right.Leftmost;

    Boolean tangentChanged = true;
    while (tangentChanged) {
        float slope = CalculateSlope(leftPoint, rightPoint);
        float checkSlope = slope;
        int leftIndex = leftList.IndexOf(leftPoint) == 0 ? leftList.Count-1 : leftList.IndexOf(leftPoint) - 1;
        do {
            float currSlope = CalculateSlope(leftList[leftIndex], rightPoint);
            if (currSlope < checkSlope) {
                checkSlope = currSlope;
                leftPoint = leftList[leftIndex];
                leftPointChange = true;
                if (leftIndex == 0) {
                    leftIndex = leftList.Count;
                }
                leftIndex--;
            }
            else {
                leftPointChange = false;
            }
        } while (leftPointChange);

        int rightIndex = rightList.IndexOf(rightPoint) == rightList.Count - 1 ? 0 : rightList.IndexOf(rightPoint) + 1;
        do {
            float currSlope = CalculateSlope(leftPoint, rightList[rightIndex]);
            if (currSlope > checkSlope) {
                checkSlope = currSlope;
                rightPoint = rightList[rightIndex];
                rightPointChange = true;
                if (rightIndex == rightList.Count - 1) {
                    rightIndex = -1;
                }
                rightIndex++;
            }
            else {
                rightPointChange = false;
            }
        } while (rightPointChange);
        if (checkSlope != slope) {
            tangentChanged = true;
        }
        else {
            tangentChanged = false;
        }
    }
    p_left.UpperTangentPoint = leftPoint;
    p_right.UpperTangentPoint = rightPoint;
```

```
}

private void FindLowerTangent(ConvexHull p_left, ConvexHull p_right)
{
   Boolean leftPointChange = false;
   Boolean rightPointChange = false;
   List<PointF> leftList = p_left.Points;
   List<PointF> rightList = p_right.Points;
   PointF leftPoint = p_left.Rightmost;
   PointF rightPoint = p_right.Leftmost;

   Boolean tangentChanged = true;
   while (tangentChanged)
   {
      float slope = CalculateSlope(leftPoint, rightPoint);
      int leftIndex = leftList.IndexOf(leftPoint) == leftList.Count - 1 ? 0: leftList.IndexOf(leftPoint) + 1;
      float checkSlope = slope;
      do {
         float currSlope = CalculateSlope(leftList[leftIndex], rightPoint);
         if (currSlope > checkSlope) {
            checkSlope = currSlope;
            leftPoint = leftList[leftIndex];
            leftPointChange = true;
            if (leftIndex == leftList.Count -1) {
               leftIndex = -1;
            }
            leftIndex++;
         }
         else {
            leftPointChange = false;
         }
      } while (leftPointChange);

      int rightIndex = rightList.IndexOf(rightPoint) == 0 ? rightList.Count - 1 : rightList.IndexOf(rightPoint) - 1;
      do {
         float currSlope = CalculateSlope(leftPoint, rightList[rightIndex]);
         if (currSlope < checkSlope) {
            checkSlope = currSlope;
            rightPoint = rightList[rightIndex];
            rightPointChange = true;
            if (rightIndex == 0)
            {
               rightIndex = rightList.Count;
            }
            rightIndex--;
         }
         else
         {
            rightPointChange = false;
         }
      } while (rightPointChange);

      if (checkSlope != slope)
      {
         tangentChanged = true;
      }
      else
      {
         tangentChanged = false;
      }
   }
   p_left.LowerTangentPoint = leftPoint;
   p_right.LowerTangentPoint = rightPoint;
}
```

```
    private float CalculateSlope(PointF leftPoint, PointF rightPoint)
    {
      return ((leftPoint.Y - rightPoint.Y)/(rightPoint.X - leftPoint.X));
    }

    private void MakeClockwise(List<PointF> p_list)
    {
      if (p_list.Count == 2)
      {
        float slope = CalculateSlope(p_list[0], p_list[1]);
        if (slope > 0)
        {
          PointF temp = p_list[0];
          //p_list[0] = p_list[1];
          //p_list[1] = temp;
        }
      }
      else
      {
        float slopeOne = CalculateSlope(p_list[0], p_list[1]);
        float slopeTwo = CalculateSlope(p_list[0], p_list[2]);
        if (slopeOne < slopeTwo)
        {
          PointF temp = p_list[1];
          p_list[1] = p_list[2];
          p_list[2] = temp;
        }
      }

    }

    public void Draw(ConvexHull p_hull)
    {
      List<PointF> points = p_hull.Points;
      for (int i = 0; i < points.Count; i++)
      {
        PointF currPoint = points[i];
        PointF nextPoint;
        if (i == points.Count-1)
        {
          nextPoint = points[0];
        }
        else
        {
          nextPoint = points[i + 1];
        }
        g.DrawLine(Pens.Black, currPoint, nextPoint);

      }
    }
  }


class ConvexHull
  {
    List<PointF> m_Points;
    PointF m_leftmost;
    PointF m_rightmost;
    PointF m_upperTangentPoint;
    PointF m_lowerTangentPoint;

    public ConvexHull() { }

    public ConvexHull(List<PointF> _list)
    {
```

```
    m_Points = _list;
    m_leftmost = findLeftMost();
    m_rightmost = findRightMost();
    if (_list.Count == 3)
    {
        MakeClockwise();
    }
}

private void MakeClockwise()
{
    if (m_Points.Count == 2)
    {
        float slope = CalculateSlope(m_Points[0], m_Points[1]);
        if (slope > 0)
        {
            PointF temp = m_Points[0];
            //m_Points[0] = m_Points[1];
            //m_Points[1] = temp;
        }
    }
    else
    {
        float slopeOne = CalculateSlope(m_Points[0], m_Points[1]);
        float slopeTwo = CalculateSlope(m_Points[0], m_Points[2]);
        if (slopeOne < slopeTwo)
        {
            PointF temp = m_Points[1];
            m_Points[1] = m_Points[2];
            m_Points[2] = temp;
        }
    }
}

public List<PointF> Points { get => m_Points; set => m_Points = value; }
public PointF Leftmost { get => m_leftmost; set => m_leftmost = value; }
public PointF Rightmost { get => m_rightmost; set => m_rightmost = value; }
public PointF UpperTangentPoint { get => m_upperTangentPoint; set => m_upperTangentPoint = value; }
public PointF LowerTangentPoint { get => m_lowerTangentPoint; set => m_lowerTangentPoint = value; }

private PointF findLeftMost()
{
    PointF leftMost = m_Points[0];
    for (int i = 0; i < m_Points.Count; i++)
    {
        if (m_Points[i].X < leftMost.X)
        {
            leftMost = m_Points[i];
        }
    }
    return leftMost;
}

private PointF findRightMost()
{
    PointF rightMost = m_Points[m_Points.Count-1];
    for (int i = 0; i < m_Points.Count; i++)
    {
        if (m_Points[i].X > rightMost.X)
        {
            rightMost = m_Points[i];
        }
    }
    return rightMost;
}
```

	private float CalculateSlope(PointF leftPoint, PointF rightPoint)
	{
	return ((leftPoint.Y - rightPoint.Y) / (rightPoint.X - leftPoint.X));
	}
	}
	}

2. Time and Space Complexity	DivideAndConquer: Space was O(log n). Time was O(nlog n) Merge: , FindUpperTangent: Worst case O(n) if each side moved on each iteration of the loop. But very unlikely. Probably closer to O(log n) with large input size FindLowerTangent: Worst case O(n), for same reasoning as finding the upper tangent. Average case O(log n) for large input size. Draw:
3. Raw and mean experimental outcomes, plot, and discussion	
4. Observations	
5. Screenshot	