**Assignment Report**

**By**

**Mr Pariwat Ramphuengnit**

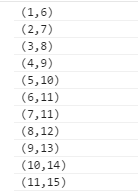
**ID: 55070701615**

**Assignment Report**

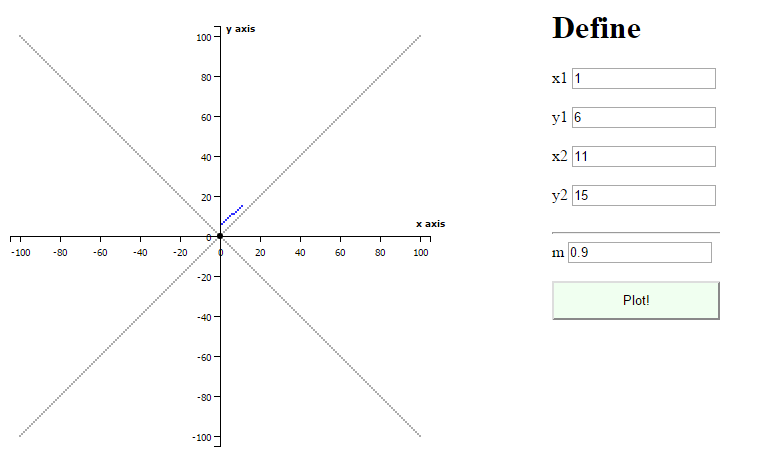
1. (10 Points) Scan Conversion (Line Segment) using original Bresenham’s line algorithm.

1.1 (2.5 Points) Scan conversion of a line segment, where two points are (1, 6) and (11, 15).

Result:



Graph:

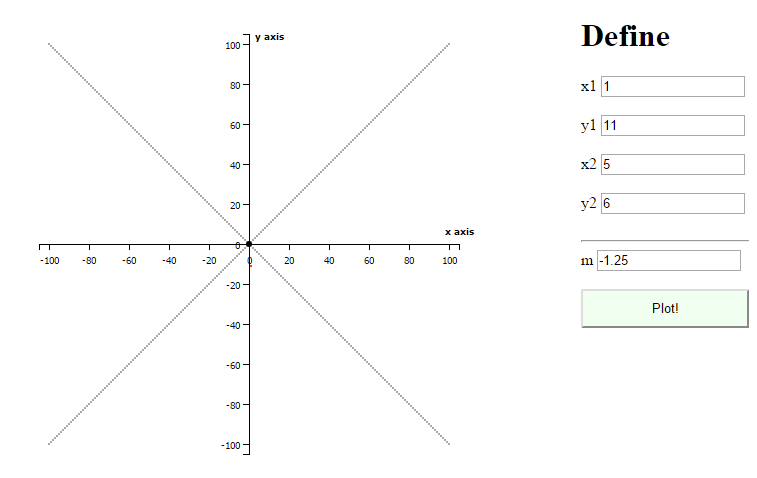


1.2 (2.5 Points) Scan conversion of a line segment, where two points are (1, 11) and (5, 6).

Result:



Graph:

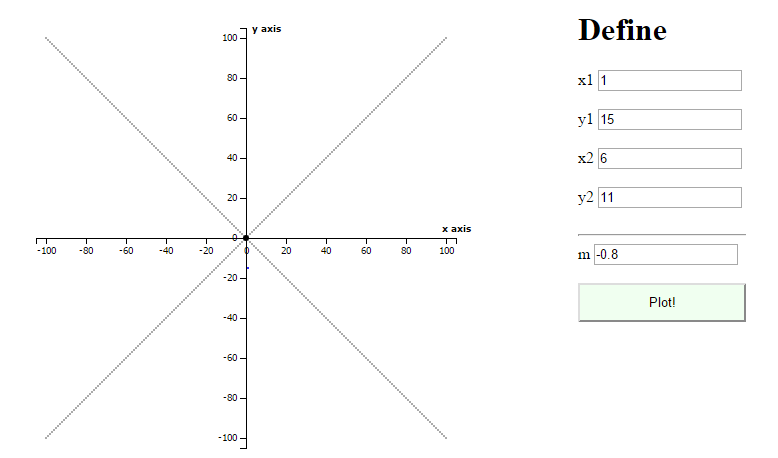


1.3 (2.5 Points) Scan conversion of a line segment, where two points are (1, 15) and (6, 11).

Result:



Graph:

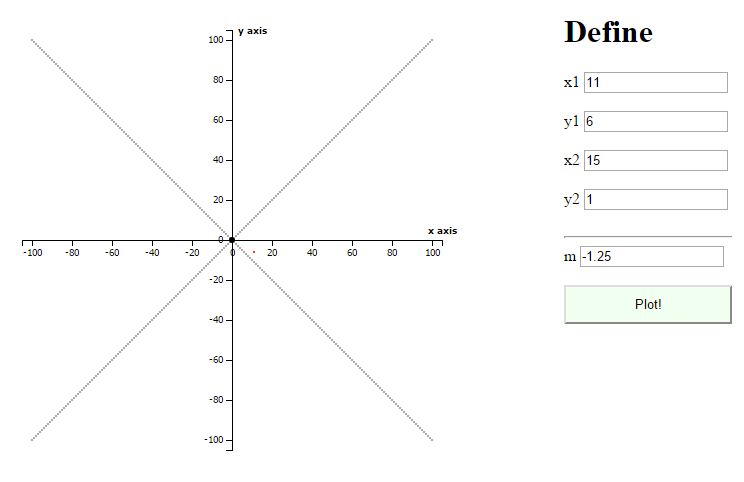


1.4 (2.5 Points) Scan conversion of a line segment, where two points are (11, 6) and (15, 1).

Result:



Graph:



Source Code for Line Segment

Step Condition and Pass value to “Calculate Point Function”

|  |
| --- |
| m = (\_\_y2 - \_\_y1) / (\_\_x2 - \_\_x1) ;  // Q1  if(m >=0 && m<=1 && \_\_x1 < \_\_x2 )  {  var rets = bresenham(\_\_x1,\_\_x2,\_\_y1,\_\_y2);  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x))  .attr("cy", pos\_y(ret.y))  .attr("fill", "blue")  ;  }  }  else if(m > 1 && \_\_y1 < \_\_y2 )  {  var rets = bresenham(\_\_y1,\_\_y2,\_\_x1,\_\_x2);  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y))  .attr("cy", pos\_y(ret.x))  .attr("fill", "red")  ;  }  }  // Q2  else if( m>= -1 && m < 0 && \_\_x1 < \_\_x2 )  {  // alert('');  var rets = bresenham(Math.abs(\_\_y1),Math.abs(\_\_y2),Math.abs(\_\_x1),Math.abs(\_\_x2));  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y))  .attr("cy", pos\_y(ret.x \* -1))  .attr("fill", "blue")  ;  }  }  else if( m < -1 && \_\_y1 > \_\_y2 )  {  // alert('');  var rets = bresenham(Math.abs(\_\_y1),Math.abs(\_\_y2),Math.abs(\_\_x1),Math.abs(\_\_x2));  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y))  .attr("cy", pos\_y(ret.x \* -1))  .attr("fill", "red")  ;  }  }  // Q3  else if(m >=0 && m<=1 && \_\_x1 > \_\_x2 )  {  var rets = bresenham(Math.abs(\_\_x1),Math.abs(\_\_x2),Math.abs(\_\_y1),Math.abs(\_\_y2));  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x\* -1))  .attr("cy", pos\_y(ret.y\* -1))  .attr("fill", "red")  ;  }  }  else if(m > 1 && \_\_y1 > \_\_y2 )  {  var rets = bresenham(Math.abs(\_\_y1),Math.abs(\_\_y2),Math.abs(\_\_x1),Math.abs(\_\_x2));  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y\* -1))  .attr("cy", pos\_y(ret.x\* -1))  .attr("fill", "blue")  ;  }  }  // Q4  else if( m>= -1 && m < 0 && \_\_x1 > \_\_x2 )  {  // alert('');  var rets = bresenham(Math.abs(\_\_x1),Math.abs(\_\_x2),Math.abs(\_\_y1),Math.abs(\_\_y2));  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x \* -1))  .attr("cy", pos\_y(ret.y))  .attr("fill", "brown")  ;  }  }  else if( m < -1 && \_\_y1 < \_\_y2 )  {  // alert('');  var rets = bresenham(Math.abs(\_\_y1),Math.abs(\_\_y2),Math.abs(\_\_x1),Math.abs(\_\_x2));  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y \* -1))  .attr("cy", pos\_y(ret.x))  .attr("fill", "green")  ;  }  } |

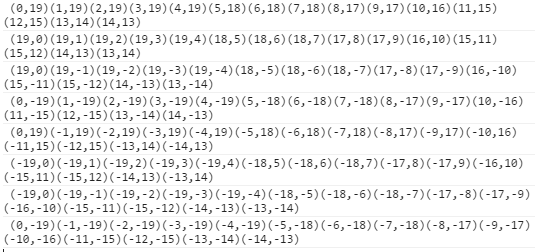
Calculate Point Function

|  |
| --- |
| function bresenham (\_x1,\_x2,\_y1,\_y2) {    var x1 = parseInt(\_x1);  var y1 = parseInt(\_y1);  var x2 = parseInt(\_x2);  var y2 = parseInt(\_y2);  var dy = y2-y1;  var dx = x2-x1;  var d = 2\*dy -dx;  var dD = 2 \* dy;  var dU = 2 \* (dy - dx);  var x = x1;  var y = y1;    var result = [];  result.push({"x": x,"y": y});  console.log("("+ x +"," + y +")") ;  while (x < x2)  {  if (d < 0)  {  d = d + dD;  x = x + 1;  }  else  {  d = d + dU;  x = x + 1;  y = y + 1;  }  result.push({"x": x,"y": y});  console.log("("+ x +"," + y +")") ;  }  return result;  } |

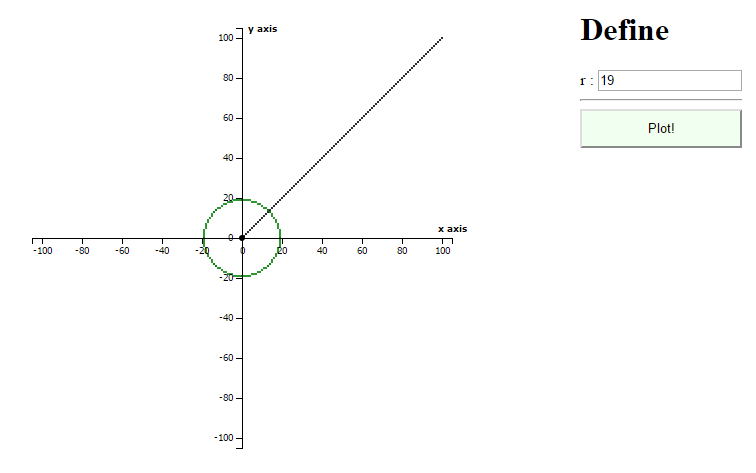
2. (10 Points) Scan Conversion (Circle)

2.1 (4 Points) Scan conversion of a circle with the radius 19 pixels using original midpoint circle algorithm. Show only the second octant of the circle.

Result:



Graph



Source code for Circle

Step Condition and Pass value to “Calculate Point Function”

|  |
| --- |
| var \_r = $("#p\_r").val();  var radi = 1;  var rets = circle\_mid(\_r);  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x))  .attr("cy", pos\_y(ret.y))  .attr("fill", "green")  ;  }  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y))  .attr("cy", pos\_y(ret.x))  .attr("fill", "green")  ;  }  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y))  .attr("cy", pos\_y(ret.x \* -1))  .attr("fill", "green")  ;  }  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x))  .attr("cy", pos\_y(ret.y \* -1))  .attr("fill", "green")  ;  }  // test  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x\* -1))  .attr("cy", pos\_y(ret.y))  .attr("fill", "green")  ;  }  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y\* -1))  .attr("cy", pos\_y(ret.x))  .attr("fill", "green")  ;  }  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.y\* -1))  .attr("cy", pos\_y(ret.x \* -1))  .attr("fill", "green")  ;  }  for(ret of rets)  {  svg.append("circle")  .attr("r", radi)  .attr("cx", pos\_x(ret.x\* -1))  .attr("cy", pos\_y(ret.y \* -1))  .attr("fill", "green")  ;  } |

Calculate Point Function

|  |
| --- |
| function circle\_mid (r) {  // body...    var h = 1-r; /\* initialization \*/  var x = 0;  var y = r;  var result = [];  // Plot(x,y);  console.log(x,y);  result.push({"x": x,"y": y});  // While y > x  while (y > x)  {  if (h < 0 ) /\* Select U \*/  {  dU = 2\*x + 3;  h = h + dU;  x = x + 1;  }  else /\* Select D \*/  {  dD = 2\*(x-y) + 5;  h = h + dD;  x = x + 1;  y = y - 1;  }  // Plot(x,y);  console.log(x,y);  result.push({"x": x,"y": y});  }  return result;    } |

5. (10 Points) Line Clipping.

Let the window coordinates be (100, 100) and (150, 160) for the lower-left and upper-right corners, respectively. Fill in the blank for the following line clipping processes:

1. Compute 4-Bit codes for each point represented line segments.

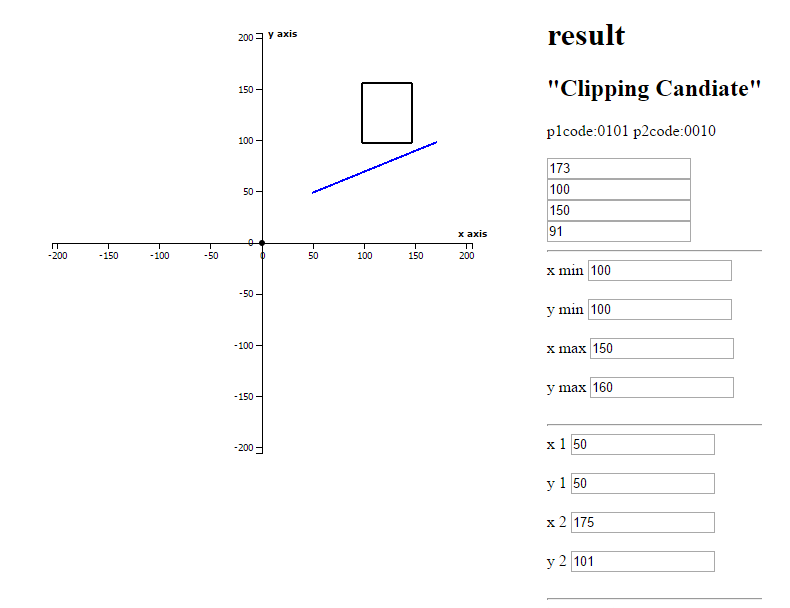
2. Use Cohen-Sutherland algorithm for the classification of these line segments into: Visible, Invisible or Clipping candidate.

3. Find the visible parts of the clipped line segments. In cases of clipping candidate, the student has to calculate and answer the intersection points.

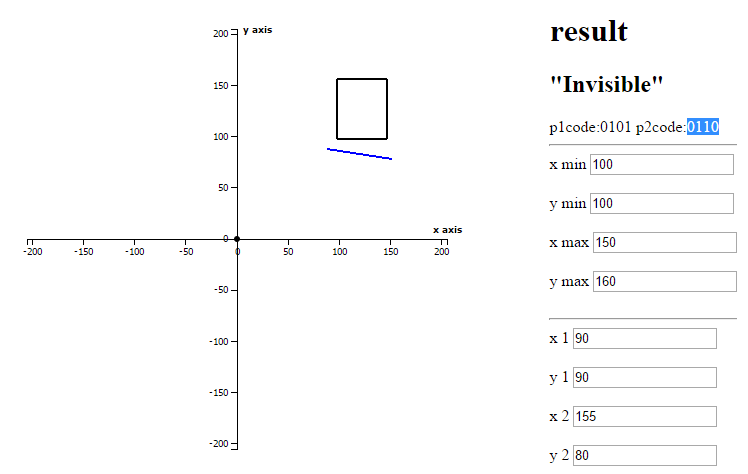
Summary Result

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Line No | Point 1 | Point 2 | 4-Bit Code point 1 | 4-Bit Code point 2 | Categories | Visible Point 1 | Visible Point 2 |
| Line1 | (50, 50) | (175, 101) | 0101 | 0010 | Clipping Candidate | (173,100) | (150,91) |
| Line2 | (90, 90) | (155, 80) | 0101 | 0110 | Invisible | - | - |
| Line3 | (95, 110) | (165, 155) | 0001 | 0010 | Clipping Candidate | (100, 113) | (150, 145) |
| Line4 | (95, 100) | (165, 135) | 0001 | 0010 | Clipping Candidate | (100, 103) | (150, 128) |
| Line5 | (110, 50) | (140, 153) | 0100 | 0000 | Clipping Candidate | (125, 100) | (140, 153) |

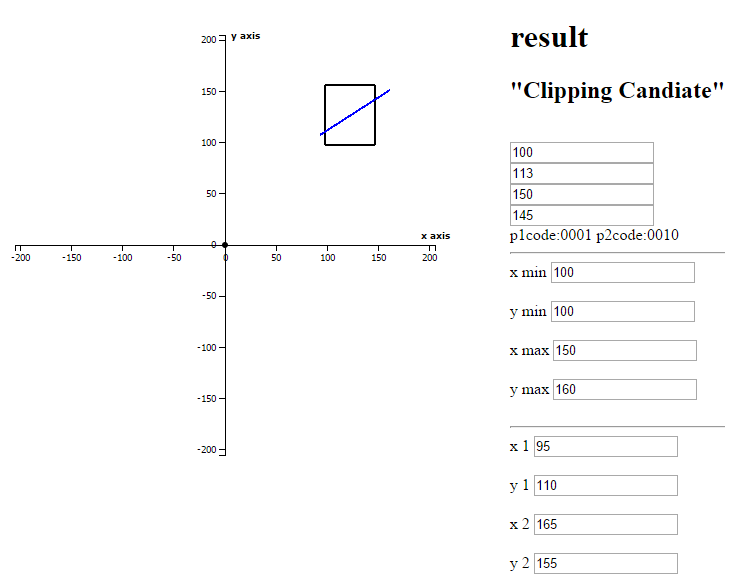
* Line 1



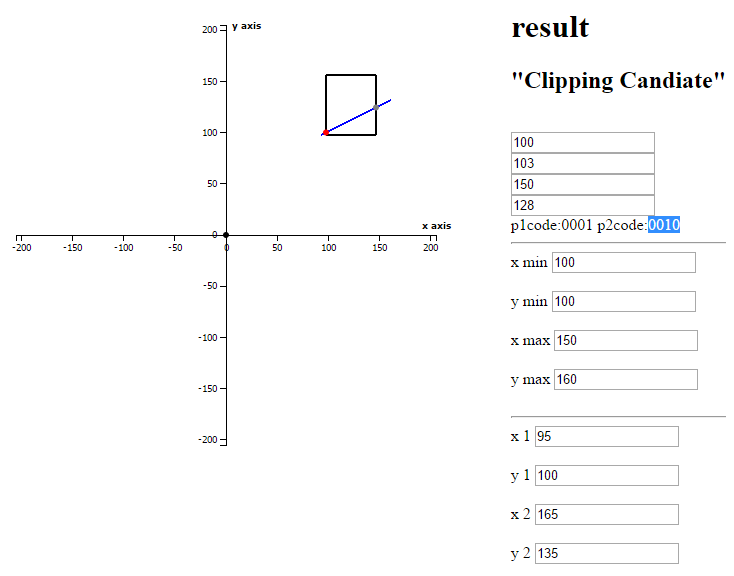
* Line 2



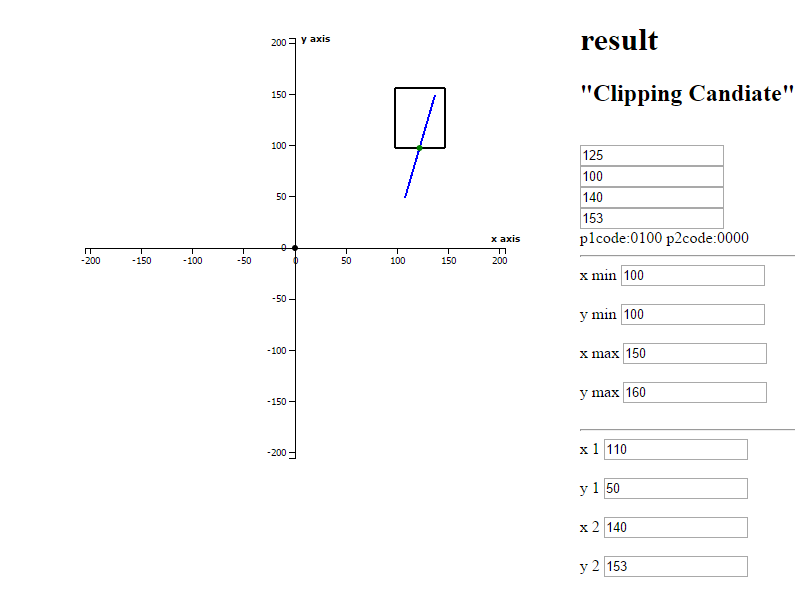
* Line 3



* Line 4



* Line 5



Source code for Line Clipping:

* Algorithm 1

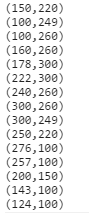
|  |
| --- |
| function conhen(xmin,xmax,ymin,ymax,x1,y1,x2,y2){  // bit represent by  // TOP BOTTOM RIGHT LEFT  var INSIDE = 0; // 0000  var LEFT = 1; // 0001  var RIGHT = 2; // 0010  var BOTTOM = 4; // 0100  var TOP = 8; // 1000  var code\_p1 = 0; // 0000  var code\_p2 = 0; // 0000  if(x1 < xmin)  {  code\_p1 = code\_p1 | LEFT;  }  else if (x1 > xmax)  {  code\_p1 = code\_p1 | RIGHT;  }  if(y1 < ymin)  {  code\_p1 = code\_p1 | BOTTOM;  }  else if (y1 > ymax)  {  code\_p1 = code\_p1 | TOP;  }  if(x2 < xmin)  {  code\_p2 = code\_p2 | LEFT;  }  else if (x2 > xmax)  {  code\_p2 = code\_p2 | RIGHT;  }  if(y2 < ymin)  {  code\_p2 = code\_p2 | BOTTOM;  }  else if (y2 > ymax)  {  code\_p2 = code\_p2 | TOP;  }  //debugger;  // Visible  // if(code\_p1 == INSIDE && code\_p2 == INSIDE){  if((code\_p1 | code\_p2) == INSIDE){  $("#lb\_visible").show();  }  // Non-Visible  // else if(code\_p1 != INSIDE && code\_p2 != INSIDE){  else if((code\_p1 & code\_p2) != INSIDE){  $("#lb\_invisible").show();  }  else  {  // y = mx + p  // m = (delta y / delta x)  var m = (y2 - y1) / (x2 -x1);  // p= y-mx;  p1 = y1 - (m \* x1);  p2 = y2 - (m \* x2);  if(p1 == p2)  {  // y = mx + p  // x = (1/m)(y-p)  var show\_alert = false;  y1cut = (m \* xmin) + p1;  if((code\_p1 | INSIDE) != 0 && y1cut>= ymin && y1cut <= ymax)  {  console.log(xmin+","+y1cut);  if(show\_alert) alert(xmin+","+y1cut);  var circle = svg.append("circle")  .attr("r", 3)  .attr("cx", pos\_x(xmin))  .attr("cy", pos\_y(y1cut))  .attr("fill", "red")  ;  }  // }  y2cut = (m \* xmax) + p2;  if((code\_p2 | INSIDE) != 0 && y2cut>= ymin && y2cut <= ymax)  {  console.log(xmax+","+y2cut);  if(show\_alert)alert(xmax+","+y2cut);  var circle = svg.append("circle")  .attr("r", 3)  .attr("cx", pos\_x(xmax))  .attr("cy", pos\_y(y2cut))  .attr("fill", "gray")  ;  }  // -------------------------  x1cut = (1/m) \* (ymin-p1);  if((code\_p1 | INSIDE) != 0 && x1cut>= xmin && x1cut <= xmax)  {  var circle = svg.append("circle")  .attr("r", 3)  .attr("cx", pos\_x(x1cut))  .attr("cy", pos\_y(ymin))  .attr("fill", "green")  ;  $("#x1\_clip\_point").val(Math.round( x1cut ));  $("#y1\_clip\_point").val(Math.round( ymin ));  $("#lb\_clipping").show();  }  x2cut = (1/m) \* (ymax-p2);  if((code\_p2 | INSIDE) != 0 && x2cut>= xmin && x2cut <= xmax)  {  var circle = svg.append("circle")  .attr("r", 3)  .attr("cx", pos\_x(x2cut))  .attr("cy", pos\_y(ymax))  .attr("fill", "blue")  ;  $("#x1\_clip\_point").val(Math.round( x2cut ));  $("#y1\_clip\_point").val(Math.round( ymax ));  $("#lb\_clipping").show();  }  if((code\_p1 | INSIDE) == 0)  {  $("#x1\_clip\_point").val(Math.round( x1 ));  $("#y1\_clip\_point").val(Math.round( y1 ));  }  if((code\_p2 | INSIDE) == 0)  {  $("#x2\_clip\_point").val(Math.round( x2 ));  $("#y2\_clip\_point").val(Math.round( y2 ));  }  }  }    $("#pcode").text("p1code:" + padDigits(code\_p1, 4) + " p2code:"+ padDigits(code\_p2, 4));  function padDigits(number, digits) {  return Array(Math.max(digits - String(number.toString(2)).length + 1, 0)).join(0) + number.toString(2);  }  } |

* Algorithm 2

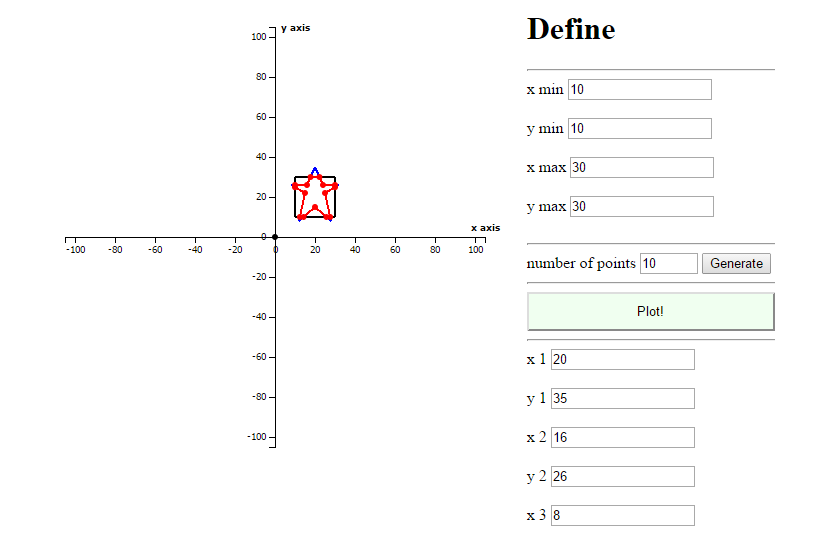
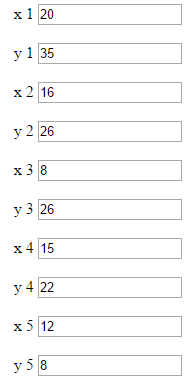
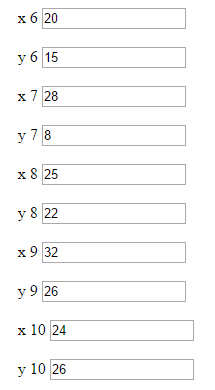
|  |
| --- |
| function conhen2(xmin,xmax,ymin,ymax,x1,y1,x2,y2){  // bit represent by  // TOP BOTTOM RIGHT LEFT  var INSIDE = 0; // 0000  var LEFT = 1; // 0001  var RIGHT = 2; // 0010  var BOTTOM = 4; // 0100  var TOP = 8; // 1000  var code\_p1 = 0; // 0000  var code\_p2 = 0; // 0000  if(x1 < xmin)  {  code\_p1 = code\_p1 | LEFT;  }  else if (x1 > xmax)  {  code\_p1 = code\_p1 | RIGHT;  }  if(y1 < ymin)  {  code\_p1 = code\_p1 | BOTTOM;  }  else if (y1 > ymax)  {  code\_p1 = code\_p1 | TOP;  }  if(x2 < xmin)  {  code\_p2 = code\_p2 | LEFT;  }  else if (x2 > xmax)  {  code\_p2 = code\_p2 | RIGHT;  }  if(y2 < ymin)  {  code\_p2 = code\_p2 | BOTTOM;  }  else if (y2 > ymax)  {  code\_p2 = code\_p2 | TOP;  }  debugger;  // Visible  // if(code\_p1 == INSIDE && code\_p2 == INSIDE){  if((code\_p1 | code\_p2) == INSIDE){  $("#lb\_visible").show();  }  // Non-Visible  // else if(code\_p1 != INSIDE && code\_p2 != INSIDE){  else if((code\_p1 & code\_p2) != INSIDE){  $("#lb\_invisible").show();  }  else  {  // find point  if ((code\_p1 & TOP) != 0)  { // point is above the clip rectangle  x1\_clip\_point = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);  y1\_clip\_point = ymax;  }  else if ((code\_p1 & BOTTOM) != 0)  { // point is below the clip rectangle  x1\_clip\_point = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);  y1\_clip\_point = ymin;  }  else if ((code\_p1 & RIGHT) != 0)  { // point is to the right of clip rectangle  y1\_clip\_point = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);  x1\_clip\_point = xmax;  }  else if ((code\_p1 & LEFT) != 0)  { // point is to the left of clip rectangle  y1\_clip\_point = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);  x1\_clip\_point = xmin;  }  else  {  if((code\_p1& INSIDE) == 0)  {  y1\_clip\_point = y1;  x1\_clip\_point = x1;  }  else  {  y1\_clip\_point = null;  x1\_clip\_point = null;  }    }  if ((code\_p2 & TOP) != 0)  { // point is above the clip rectangle  x2\_clip\_point = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);  y2\_clip\_point = ymax;  }  else if ((code\_p2 & BOTTOM) != 0)  { // point is below the clip rectangle  x2\_clip\_point = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);  y2\_clip\_point = ymin;  }  else if ((code\_p2 & RIGHT) != 0)  { // point is to the right of clip rectangle  y2\_clip\_point = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);  x2\_clip\_point = xmax;  }  else if ((code\_p2 & LEFT) != 0)  { // point is to the left of clip rectangle  y2\_clip\_point = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);  x2\_clip\_point = xmin;  }  else  {  if((code\_p2 & INSIDE) == 0)  {  y2\_clip\_point = y2;  x2\_clip\_point = x2;  }  else  {  y2\_clip\_point = null;  x2\_clip\_point = null;  }  }  $("#lb\_clipping").show();  $("#x1\_clip\_point").val(Math.round( x1\_clip\_point ) );  $("#y1\_clip\_point").val(Math.round( y1\_clip\_point ) );  $("#x2\_clip\_point").val(Math.round( x2\_clip\_point ) );  $("#y2\_clip\_point").val(Math.round( y2\_clip\_point ) );      //debugger;  }  } |

6. (10 Points) Polygon Clipping Let a window have a lower left corner at the point X(100, 100) and a upper-right corner at the point Y(300, 300). Perform the polygon clipping of a polygon with a sequence of vertices ABCDEFGHIJ, where A(200, 350), B(160, 260), C(80, 260), D(150, 220), E(120, 80), F(200, 150), G(280, 80), H(250, 220), I(320, 260) and J(240, 260).

Result: [Normalize] => [Original] =>



Graph:



Source code for Polygon clipping

Step Calculate

|  |
| --- |
| var res = [];  var res2 = [];  var res3 = [];  var res4 = [];  for (var i = 0; i < \_\_point.length; i++)  {  var \_index1 = i;  var \_index2 = (i+1 < \_\_point.length) ? i+1 : 0;    rets = do\_down(\_\_point[\_index1],\_\_point[\_index2]);  for( ret of rets)  {  if(ret != null)  res.push( ret );  }  }  for (var i = 0; i < res.length; i++)  {  var \_index1 = i;  var \_index2 = (i+1 < res.length) ? i+1 : 0;  rets = do\_rigth(res[\_index1],res[\_index2]);  for( ret of rets)  {  if(ret != null)  res2.push( ret );  }  }  for (var i = 0; i < res2.length; i++)  {  var \_index1 = i;  var \_index2 = (i+1 < res2.length) ? i+1 : 0;  rets = do\_up(res2[\_index1],res2[\_index2]);  for( ret of rets)  {  if(ret != null)  res3.push( ret );  }  }  for (var i = 0; i < res3.length; i++)  {  var \_index1 = i;  var \_index2 = (i+1 < res3.length) ? i+1 : 0;  rets = do\_left(res3[\_index1],res3[\_index2]);  for( ret of rets)  {  if(ret != null)  res4.push( ret );  }  } |

Function Calculate Point by side

* Down

|  |
| --- |
| function do\_down (p1,p2) {  // body...  //p1  if(p1.y >= \_\_ymin && p2.y >= \_\_ymin)  {  return [p2];  }  //p2  else if(p1.y < \_\_ymin && p2.y < \_\_ymin)  {  return [];  }  //p3  else if(p1.y >= \_\_ymin && p2.y < \_\_ymin)  {  var c\_p = clipping('ymin',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p]; //clip  }  //p4  else if(p1.y < \_\_ymin && p2.y >= \_\_ymin)  {  var c\_p = clipping('ymin',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p,p2]; //clip,-2  }  } |

* Rigth

|  |
| --- |
| function do\_rigth (p1,p2) {  // body...  //p1  if(p1.x <= \_\_xmax && p2.x <= \_\_xmax)  {  return [p2];  }  //p2  else if(p1.x > \_\_xmax && p2.x > \_\_xmax)  {  return [];  }  //p3  else if(p1.x <= \_\_xmax && p2.x > \_\_xmax)  {  var c\_p = clipping('xmax',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p]; //clip  }  //p4  else if(p1.x > \_\_xmax && p2.x <= \_\_xmax)  {  var c\_p = clipping('xmax',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p,p2]; //clip,-2  }  } |

* Up

|  |
| --- |
| function do\_up (p1,p2) {  // body...  //p1  if(p1.y <= \_\_ymax && p2.y <= \_\_ymax)  {  return [p2];  }  //p2  else if(p1.y > \_\_ymax && p2.y > \_\_ymax)  {  return [];  }  //p3  else if(p1.y <= \_\_ymax && p2.y > \_\_ymax)  {  var c\_p = clipping('ymax',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p]; //clip  }  //p4  else if(p1.y > \_\_ymax && p2.y <= \_\_ymax)  {  var c\_p = clipping('ymax',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p,p2]; //clip,-2  }  } |

* Left

|  |
| --- |
| function do\_left (p1,p2) {  // body...  //p1  if(p1.x >= \_\_xmin && p2.x >= \_\_xmin)  {  return [p2];  }  //p2  else if(p1.x < \_\_xmin && p2.x < \_\_xmin)  {  return [];  }  //p3  else if(p1.x >= \_\_xmin && p2.x < \_\_xmin)  {  var c\_p = clipping('xmin',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p]; //clip  }  //p4  else if(p1.x < \_\_xmin && p2.x >= \_\_xmin)  {  var c\_p = clipping('xmin',\_\_xmin,\_\_xmax,\_\_ymin,\_\_ymax,p1.x,p1.y,p2.x,p2.y);  return [c\_p,p2]; //clip,-2  }  } |

Function Clipping

|  |
| --- |
| function clipping(selectLine,xmin,xmax,ymin,ymax,x1,y1,x2,y2){  // bit represent by  // TOP BOTTOM RIGHT LEFT  var INSIDE = 0; // 0000  var LEFT = 1; // 0001  var RIGHT = 2; // 0010  var BOTTOM = 4; // 0100  var TOP = 8; // 1000  var code\_p1 = 0; // 0000  var code\_p2 = 0; // 0000  if(x1 < xmin)  {  code\_p1 = code\_p1 | LEFT;  }  else if (x1 > xmax)  {  code\_p1 = code\_p1 | RIGHT;  }  if(y1 < ymin)  {  code\_p1 = code\_p1 | BOTTOM;  }  else if (y1 > ymax)  {  code\_p1 = code\_p1 | TOP;  }  if(x2 < xmin)  {  code\_p2 = code\_p2 | LEFT;  }  else if (x2 > xmax)  {  code\_p2 = code\_p2 | RIGHT;  }  if(y2 < ymin)  {  code\_p2 = code\_p2 | BOTTOM;  }  else if (y2 > ymax)  {  code\_p2 = code\_p2 | TOP;  }    // Visible  if((code\_p1 | code\_p2) == INSIDE){  return null;  }  // Non-Visible  // else if(code\_p1 != INSIDE && code\_p2 != INSIDE){  else if((code\_p1 & code\_p2) != INSIDE){  return null;  }  else  {  // find point  if ((code\_p1 & TOP) != 0)  { // point is above the clip rectangle  x1\_clip\_point = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);  y1\_clip\_point = ymax;  }  else if ((code\_p1 & BOTTOM) != 0)  { // point is below the clip rectangle  x1\_clip\_point = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);  y1\_clip\_point = ymin;  }  else if ((code\_p1 & RIGHT) != 0)  { // point is to the right of clip rectangle  y1\_clip\_point = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);  x1\_clip\_point = xmax;  }  else if ((code\_p1 & LEFT) != 0)  { // point is to the left of clip rectangle  y1\_clip\_point = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);  x1\_clip\_point = xmin;  }  else  {  y1\_clip\_point = null;  x1\_clip\_point = null;  }  if ((code\_p2 & TOP) != 0)  { // point is above the clip rectangle  x2\_clip\_point = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);  y2\_clip\_point = ymax;  }  else if ((code\_p2 & BOTTOM) != 0)  { // point is below the clip rectangle  x2\_clip\_point = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);  y2\_clip\_point = ymin;  }  else if ((code\_p2 & RIGHT) != 0)  { // point is to the right of clip rectangle  y2\_clip\_point = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);  x2\_clip\_point = xmax;  }  else if ((code\_p2 & LEFT) != 0)  { // point is to the left of clip rectangle  y2\_clip\_point = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);  x2\_clip\_point = xmin;  }  else  {  y2\_clip\_point = null;  x2\_clip\_point = null;  }      if(selectLine == 'ymin')  isOnSelectLine = (y1\_clip\_point == \_\_ymin);  if(selectLine == 'ymax')  isOnSelectLine = (y1\_clip\_point == \_\_ymax);  if(selectLine == 'xmin')  isOnSelectLine = (x1\_clip\_point == \_\_xmin);  if(selectLine == 'xmax')  isOnSelectLine = (x1\_clip\_point == \_\_xmax);    if (x1\_clip\_point != null && y1\_clip\_point != null && isOnSelectLine)  {  return { "x" : x1\_clip\_point,"y": y1\_clip\_point};  }  else  {  return { "x" : x2\_clip\_point,"y": y2\_clip\_point};  }    }//end else  }//end function clipping |

All Source code Link:

<https://www.dropbox.com/s/zuky5zjuuug0jno/55070701615-CG-Assignment.zip?dl=0>